Entwicklung einer Chatseite mit Angular und Websockets

Hochschule Karlsruhe CareerDay

03.12.2020

Luis Aceituno und Lars Debor

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Wer sind wir?

Luis Aceituno

Lars Debor

Programm

1. Einführung in Angular
2. Einführung in Websockets
3. Workshop: Chat App basteln
4. Ergebnisse, Diskussion

Programm

1. Einführung in Angular
2. Einführung in Websockets
3. Workshop: Chat App basteln
4. Ergebnisse, Diskussion

1. Einführung in Angular

Was ist Angular?

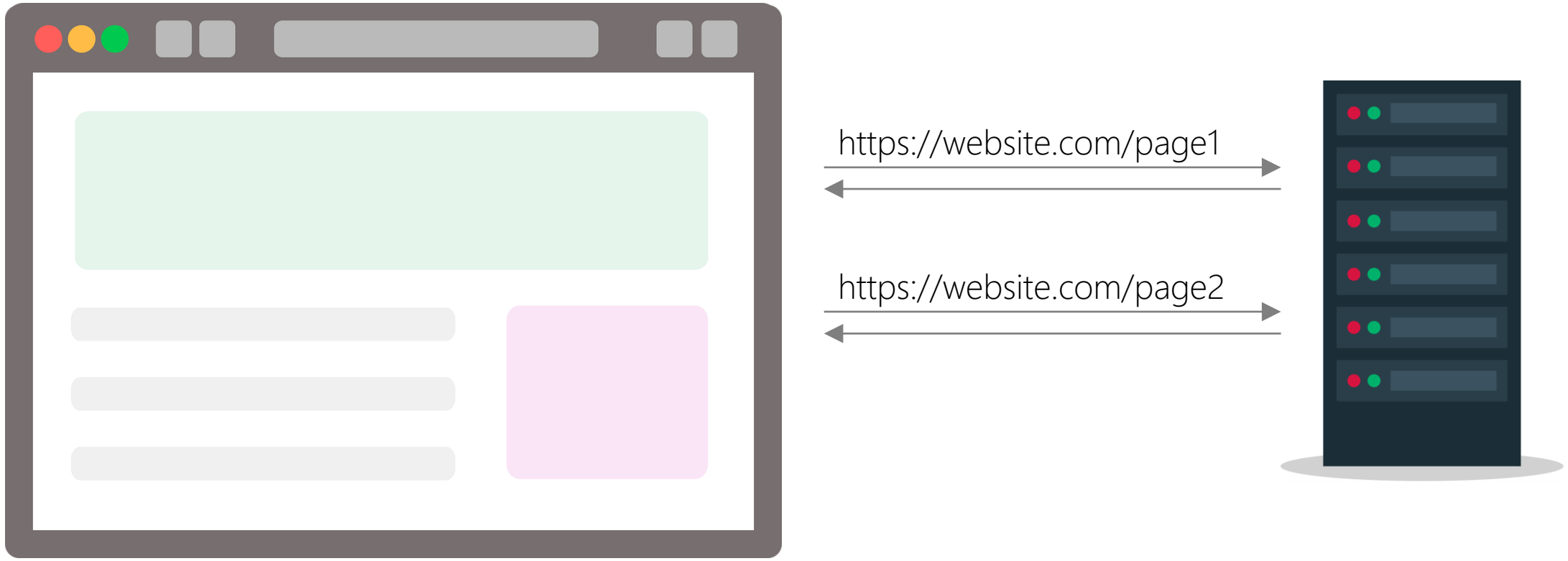
- JavaScript Framework
- Läuft im Browser
- Seit 2016 aktiv entwickelt
  - Nicht mit AngularJS verwechseln!
- Open Source (Google)

→ Single Page Application (SPA)

1. Einführung in Angular

Was ist eine Single Page Application?

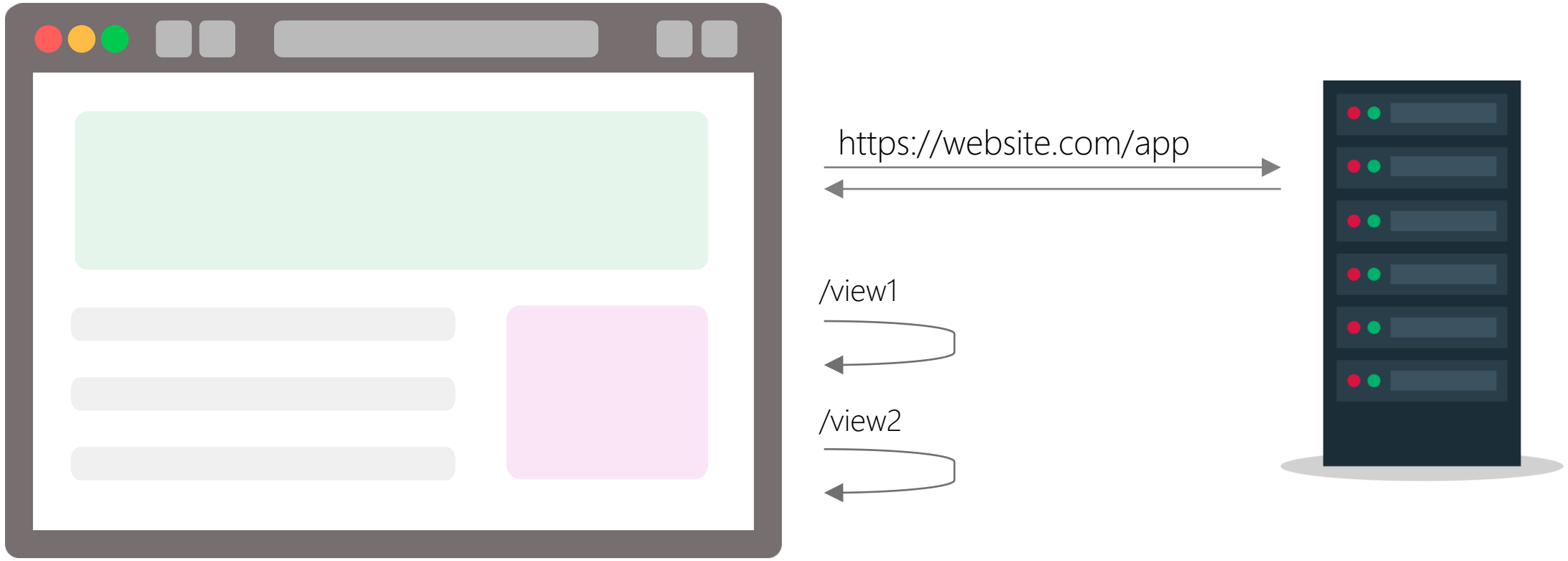
Traditionell (Multi-page)



1. Einführung in Angular

Was ist eine Single Page Application?

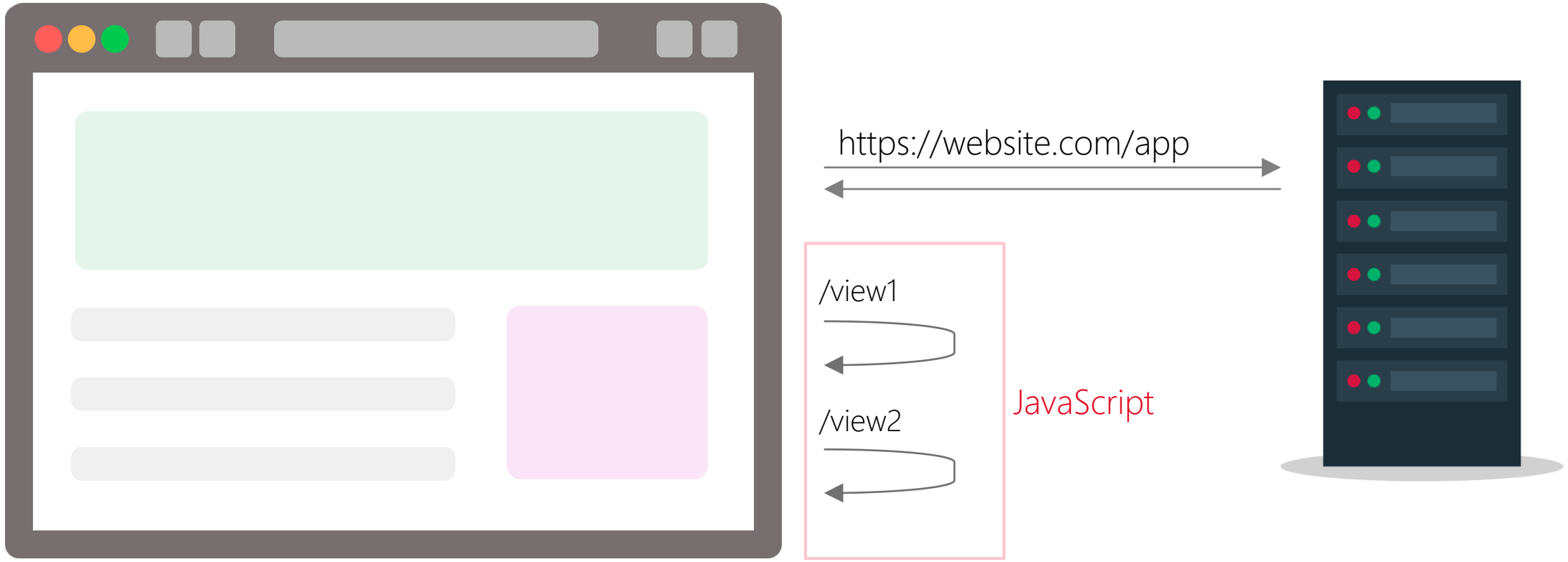
Single Page



1. Einführung in Angular

Was ist eine Single Page Application?

Single Page



1. Einführung in Angular

Wichtige Begriffe

TypeScript

RxJS

1. Einführung in Angular

Was ist RxJS?

- Bibliothek für „Reactive Programming“
  - Einfach gesagt: Observer Pattern
- Wichtigste Klasse: **Observable**

1. Einführung in Angular

Was ist RxJS?


```
const message$: Observable<Message> = this.chatService.getMessages();

message$.subscribe(
  message => this.displayMessageToUser(message)
);
```

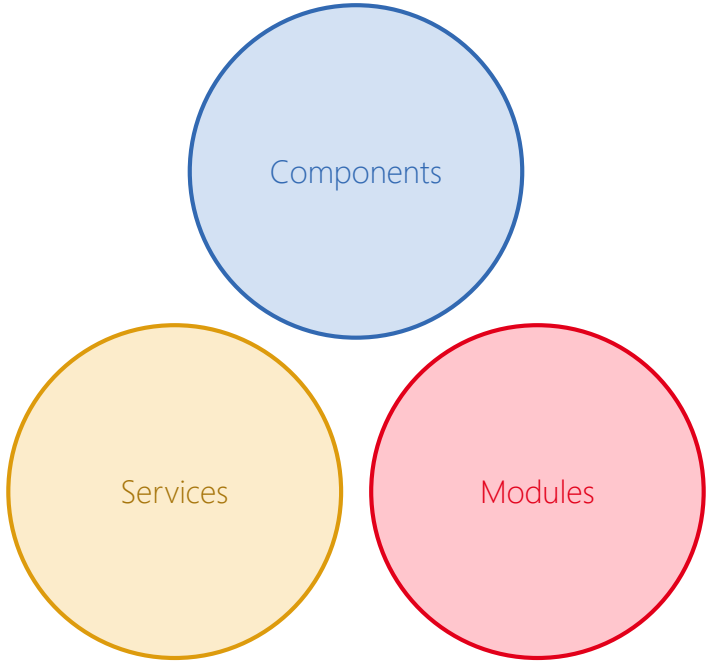
1. Einführung in Angular

Was ist RxJS?


```
message$.pipe(
    filter(message => message.length > 0),
    map(message => message.text)
).subscribe(
  text => this.displayMessageToUser(text)
);
```

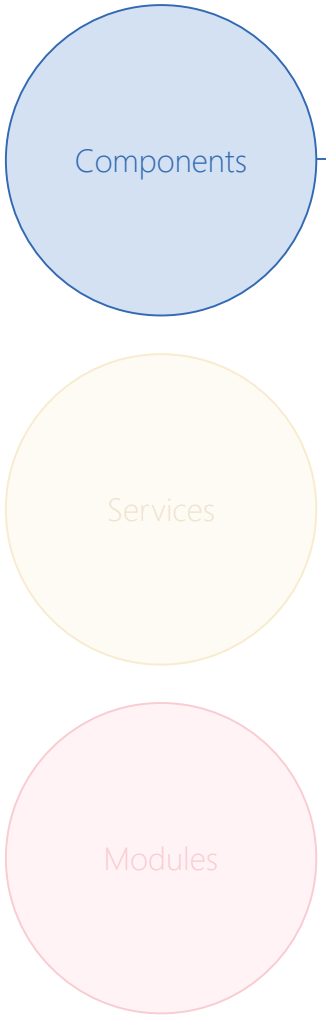
1. Einführung in Angular

Grundkonzepte

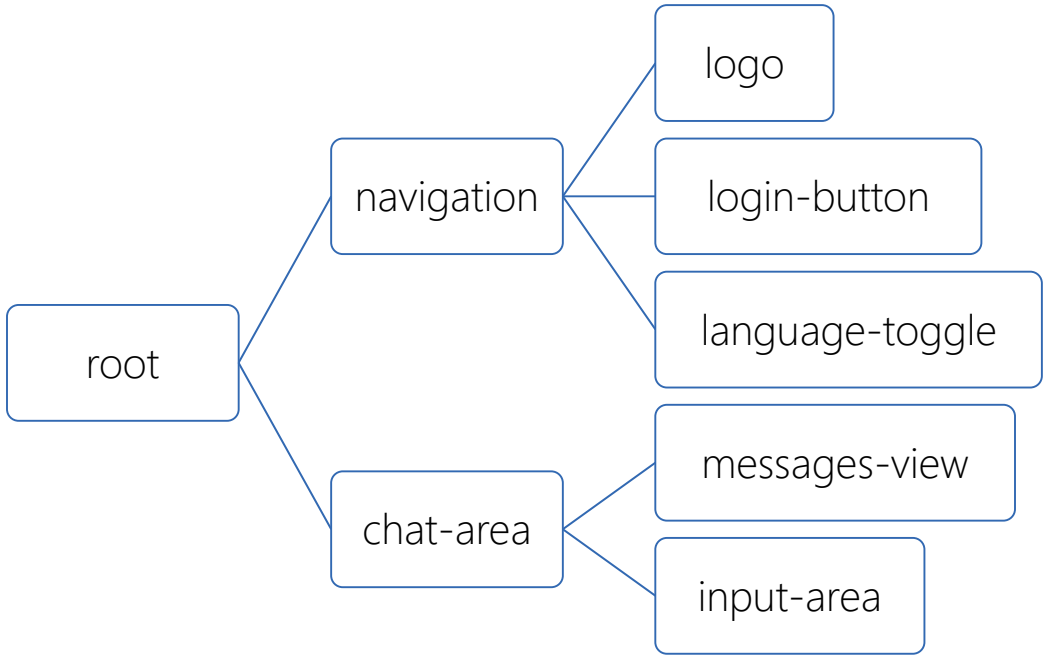


1. Einführung in Angular

Components

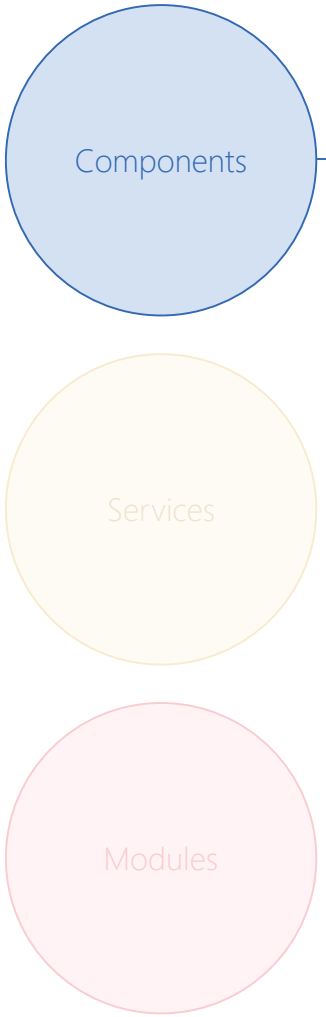


- Alles was man in einer Angular Anwendung sehen kann, ist eine Komponente
- Komponenten sind die „Bausteine“ der UI

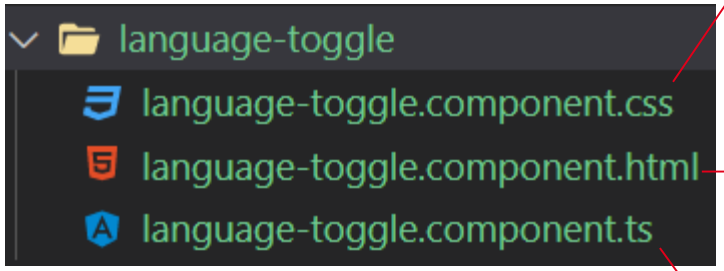


1. Einführung in Angular

Components



- Bestehen aus 3 Dateien:



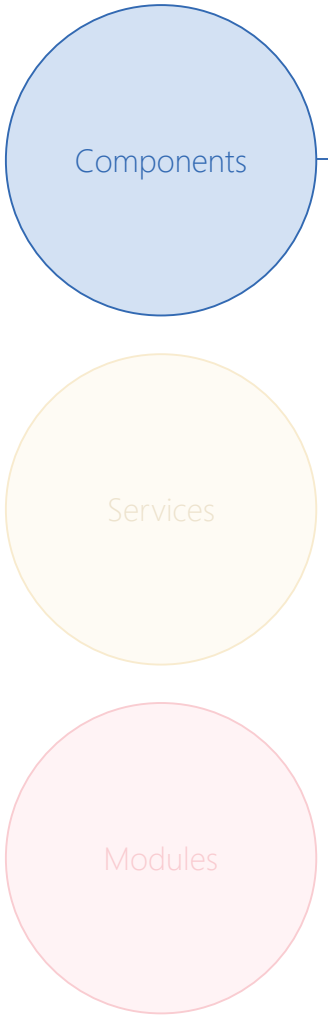
CSS Styling zum Anzeigen im Browser

HTML Template zum Anzeigen im Browser

TypeScript Klasse mit Daten (Variablen) und Logik

1. Einführung in Angular

Components



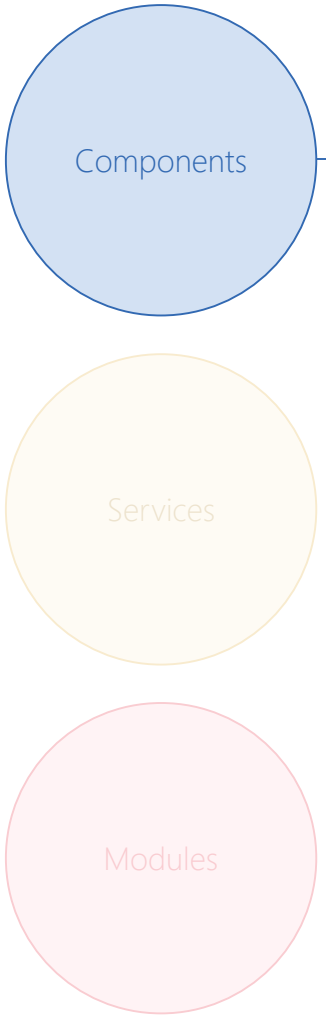
HTML Template

language-toggle.component.html

```
<div class="language-toggle-container">
  <app-language-flag [language]="language"></app-language-flag>
  <span>Language: {{language}}</span>
  <button *ngIf="canSelectGerman()" (click)="selectGerman()">Deutsch</button>
  <button *ngIf="canSelectEnglish()" (click)="selectEnglish()">English</button>
</div>
```

1. Einführung in Angular

Components



HTML Template

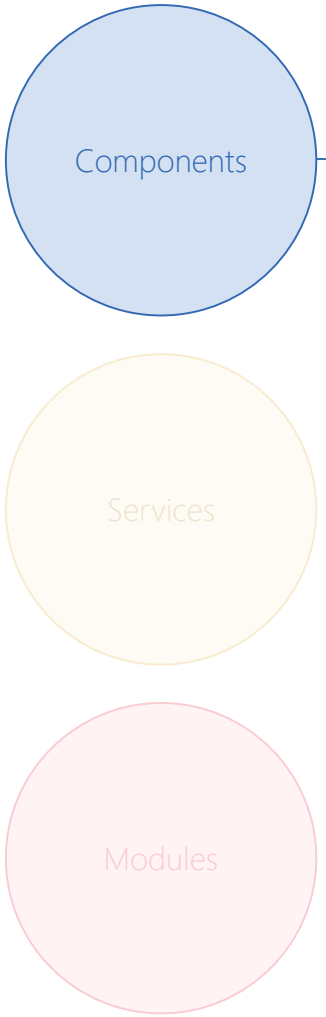
`language-toggle.component.html`

```
<div class="language-toggle-container">
  <app-language-flag [language]="language"></app-language-flag>
  <span>Language: {{language}}</span>
  <button *ngIf="canSelectGerman()" (click)="selectGerman()">Deutsch</button>
  <button *ngIf="canSelectEnglish()" (click)="selectEnglish()">English</button>
</div>
```

Unterkomponente

1. Einführung in Angular

Components



HTML Template

`language-toggle.component.html`

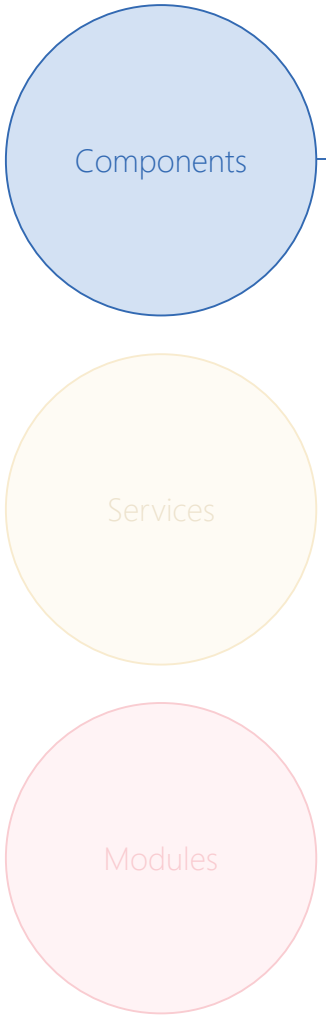
```
<div class="language-toggle-container">
  <app-language-flag [language]="language"></app-language-flag>
  <span>Language: {{language}}</span>
  <button *ngIf="canSelectGerman()" (click)="selectGerman()">Deutsch</button>
  <button *ngIf="canSelectEnglish()" (click)="selectEnglish()">English</button>
</div>
```

Übertragen von Daten zur Unterkomponente mit `[]` Syntax

- Die Unterkomponente hat eine Variable mit dem Namen „language“
- Der Wert dieser Variable wird hierüber gesetzt
- Der Wert reagiert automatisch auf Änderungen

1. Einführung in Angular

Components



HTML Template

`language-toggle.component.html`

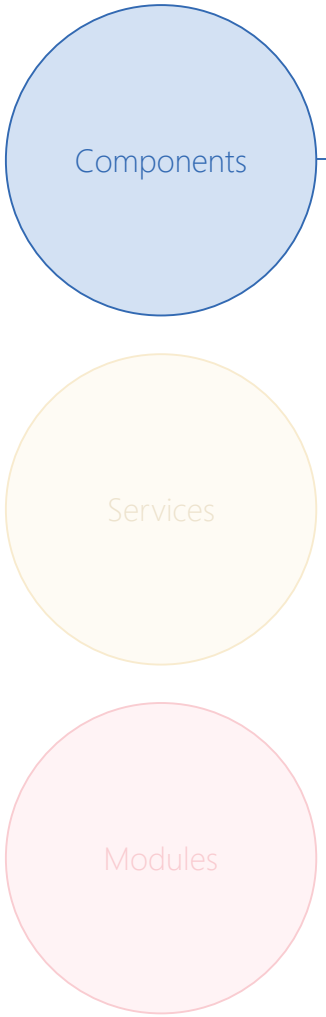
```
<div class="language-toggle-container">
  <app-language-flag [language]="language"></app-language-flag>
  <span>Language: {{language}}</span>
  <button *ngIf="canSelectGerman()" (click)="selectGerman()">Deutsch</button>
  <button *ngIf="canSelectEnglish()" (click)="selectEnglish()">English</button>
</div>
```

Den Wert einer Variable direkt im Template anzeigen mit **{{}}** Syntax

- Die Variable befindet sich in der TypeScript Klasse (sehen wir gleich)
- z.B. wenn **`language == "Deutsch"`**
  dann steht hier **`<span>Language: Deutsch</span>`**

1. Einführung in Angular

Components



HTML Template

`language-toggle.component.html`

```
<div class="language-toggle-container">
  <app-language-flag [language]="language"></app-language-flag>
  <span>Language: {{language}}</span>
  <button *ngIf="canSelectGerman()" (click)="selectGerman()">Deutsch</button>
  <button *ngIf="canSelectEnglish()" (click)="selectEnglish()">English</button>
</div>
```

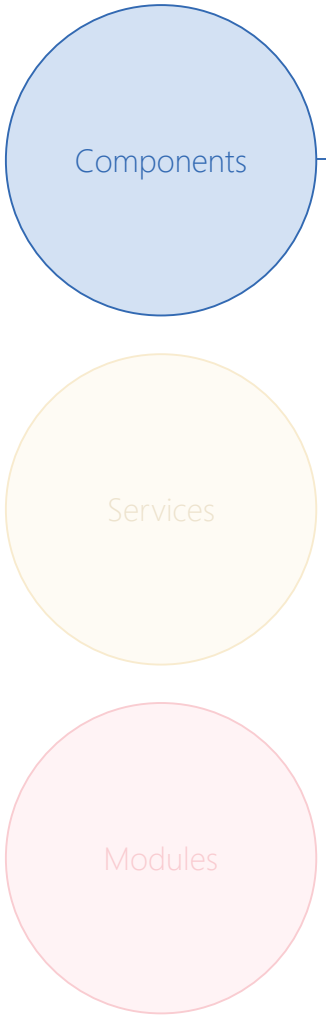
Strukturelle Direktiven verändern die Struktur des Templates
z.B. wird hier der ganze Button entfernt, falls die Funktion **`false`** zurückliefert

Eine andere sehr nützliche strukturelle Direktive ist **`*ngFor`**

```
<div *ngFor="let message of messages">{{message.text}}</div>
```

1. Einführung in Angular

Components



HTML Template

`language-toggle.component.html`

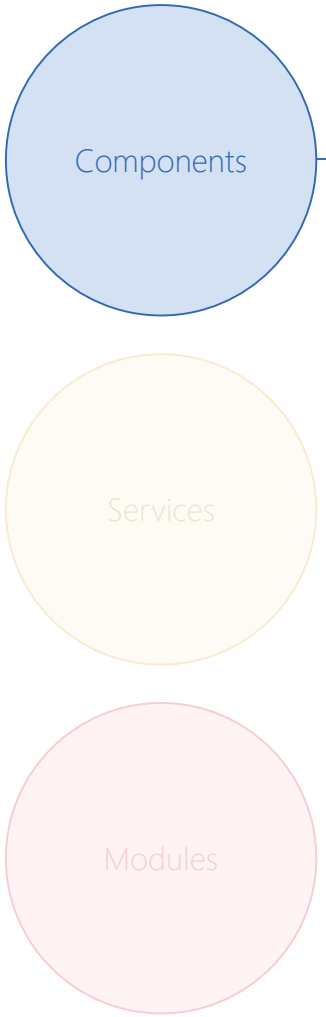
```
<div class="language-toggle-container">
  <app-language-flag [language]="language"></app-language-flag>
  <span>Language: {{language}}</span>
  <button *ngIf="canSelectGerman()" (click)="selectGerman()">Deutsch</button>
  <button *ngIf="canSelectEnglish()" (click)="selectEnglish()">English</button>
</div>
```

Auf Events von Unterkomponenten reagieren mit `()` Syntax

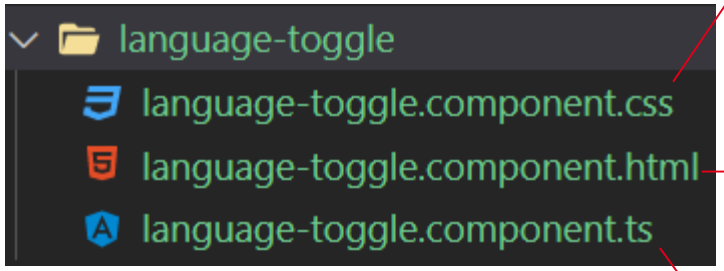
- `(click)` ist standardmäßig in Angular enthalten
- Komponenten kennen ihre übergeordneten Komponenten nicht, deshalb ist Kommunikation nach oben nur mit Events möglich

1. Einführung in Angular

Components



- Bestehen aus 3 Dateien:



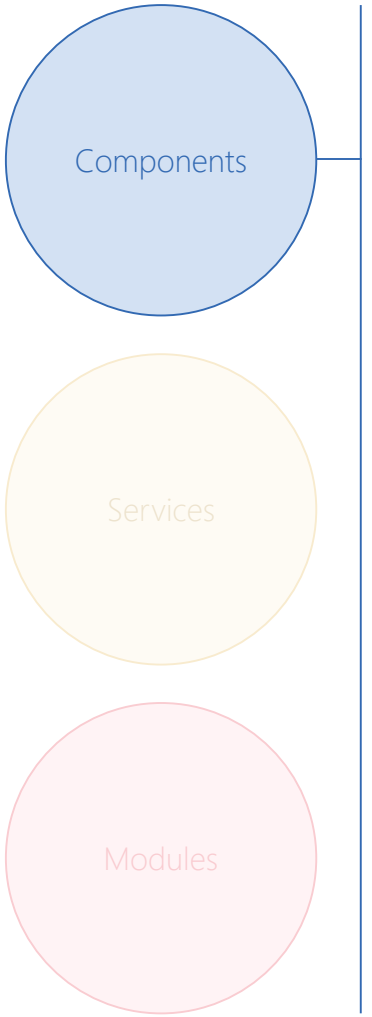
CSS Styling zum Anzeigen im Browser

HTML Template zum Anzeigen im Browser

TypeScript Klasse mit Daten (Variablen) und Logik

1. Einführung in Angular

Components



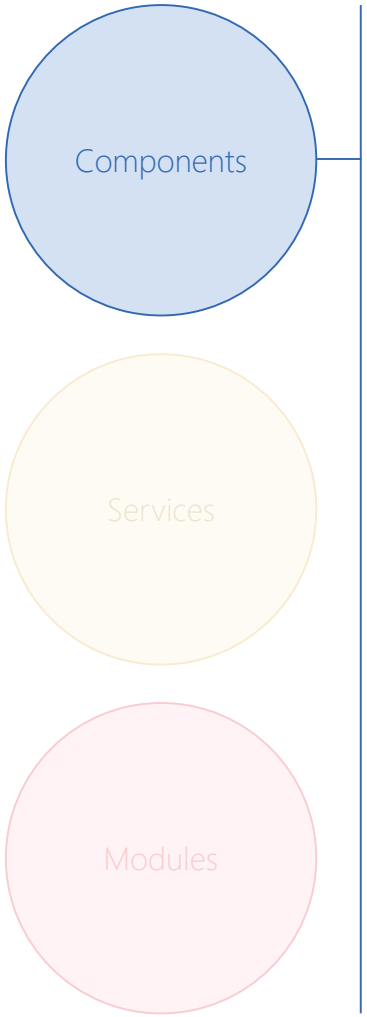
TypeScript Klasse

language-toggle.component.ts

```ts
@Component({
  selector: 'app-language-toggle',
  templateUrl: './language-toggle.component.html',
  styleUrls: ['./language-toggle.component.css']
})
export class LanguageToggleComponent {

  @Input()
  language: string;

  @Output()
  languageChange = new EventEmitter<string>();

  canSelectGerman(): boolean {
    return this.language !== 'Deutsch';
  }

  canSelectEnglish(): boolean {
    return this.language !== 'English';
  }

  selectGerman(): void {
    this.languageChange.emit('Deutsch');
  }

  selectEnglish(): void {
    this.languageChange.emit('English');
  }
}
```

1. Einführung in Angular

Components



TypeScript Klasse

Metadaten der Komponente

selector: Wie heißt der HTML Tag?

templateUrl: Wo ist das HTML Template?

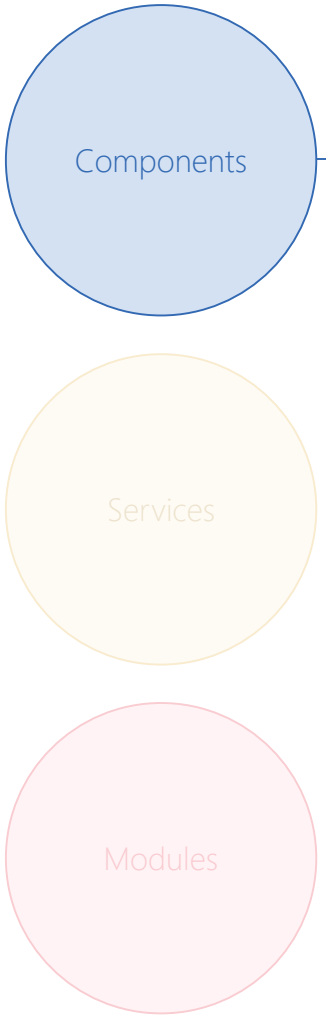
styleUrls: Wo sind die CSS Stylesheets?

language-toggle.component.ts

```
@Component({
  selector: 'app-language-toggle',
  templateUrl: './language-toggle.component.html',
  styleUrls: ['./language-toggle.component.css']
})
export class LanguageToggleComponent {

  @Input()
  language: string;

  @Output()
  languageChange = new EventEmitter<string>();

  canSelectGerman(): boolean {
    return this.language !== 'Deutsch';
  }

  canSelectEnglish(): boolean {
    return this.language !== 'English';
  }

  selectGerman(): void {
    this.languageChange.emit('Deutsch');
  }

  selectEnglish(): void {
    this.languageChange.emit('English');
  }
}
```

1. Einführung in Angular

Components



TypeScript Klasse

Eingabevariablen werden von der Überkomponente mit **[ ]** Syntax gesetzt

z.B.

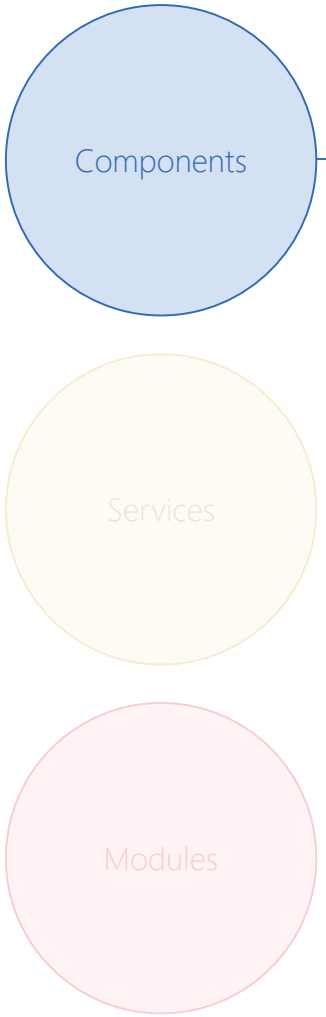
```
<app-language-toggle [language]="'Deutsch'">
</app-language-toggle>
```

language-toggle.component.ts

```
@Component({
  selector: 'app-language-toggle',
  templateUrl: './language-toggle.component.html',
  styleUrls: ['./language-toggle.component.css']
})
export class LanguageToggleComponent {

  @Input()
  language: string;

  @Output()
  languageChange = new EventEmitter<string>();

  canSelectGerman(): boolean {
    return this.language !== 'Deutsch';
  }

  canSelectEnglish(): boolean {
    return this.language !== 'English';
  }

  selectGerman(): void {
    this.languageChange.emit('Deutsch');
  }

  selectEnglish(): void {
    this.languageChange.emit('English');
  }
}
```

1. Einführung in Angular

Components



TypeScript Klasse

Mit **`@Output()`** Variablen markieren, die Überkomponenten beobachten können.

Die Klasse **`EventEmitter`** ist in Angular enthalten.

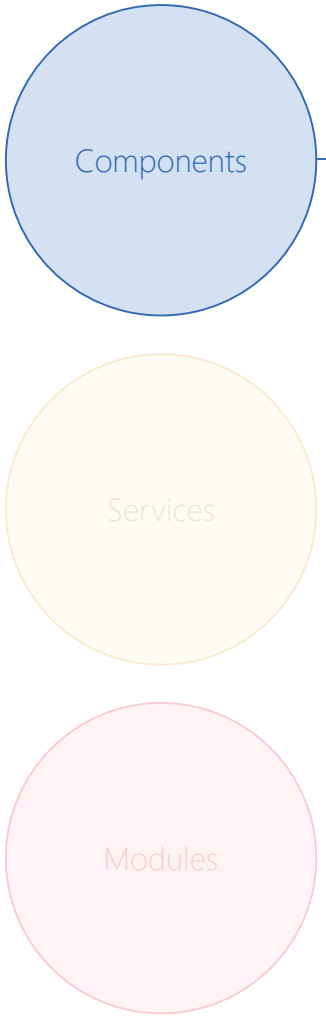
```
<app-language-toggle
  (languageChange)="languageChanged($event)"
>
</app-language-toggle>
```

language-toggle.component.ts

```
@Component({
  selector: 'app-language-toggle',
  templateUrl: './language-toggle.component.html',
  styleUrls: ['./language-toggle.component.css']
})
export class LanguageToggleComponent {

  @Input()
  language: string;

  @Output()
  languageChange = new EventEmitter<string>();

  canSelectGerman(): boolean {
    return this.language !== 'Deutsch';
  }

  canSelectEnglish(): boolean {
    return this.language !== 'English';
  }

  selectGerman(): void {
    this.languageChange.emit('Deutsch');
  }

  selectEnglish(): void {
    this.languageChange.emit('English');
  }
}
```

1. Einführung in Angular

Components

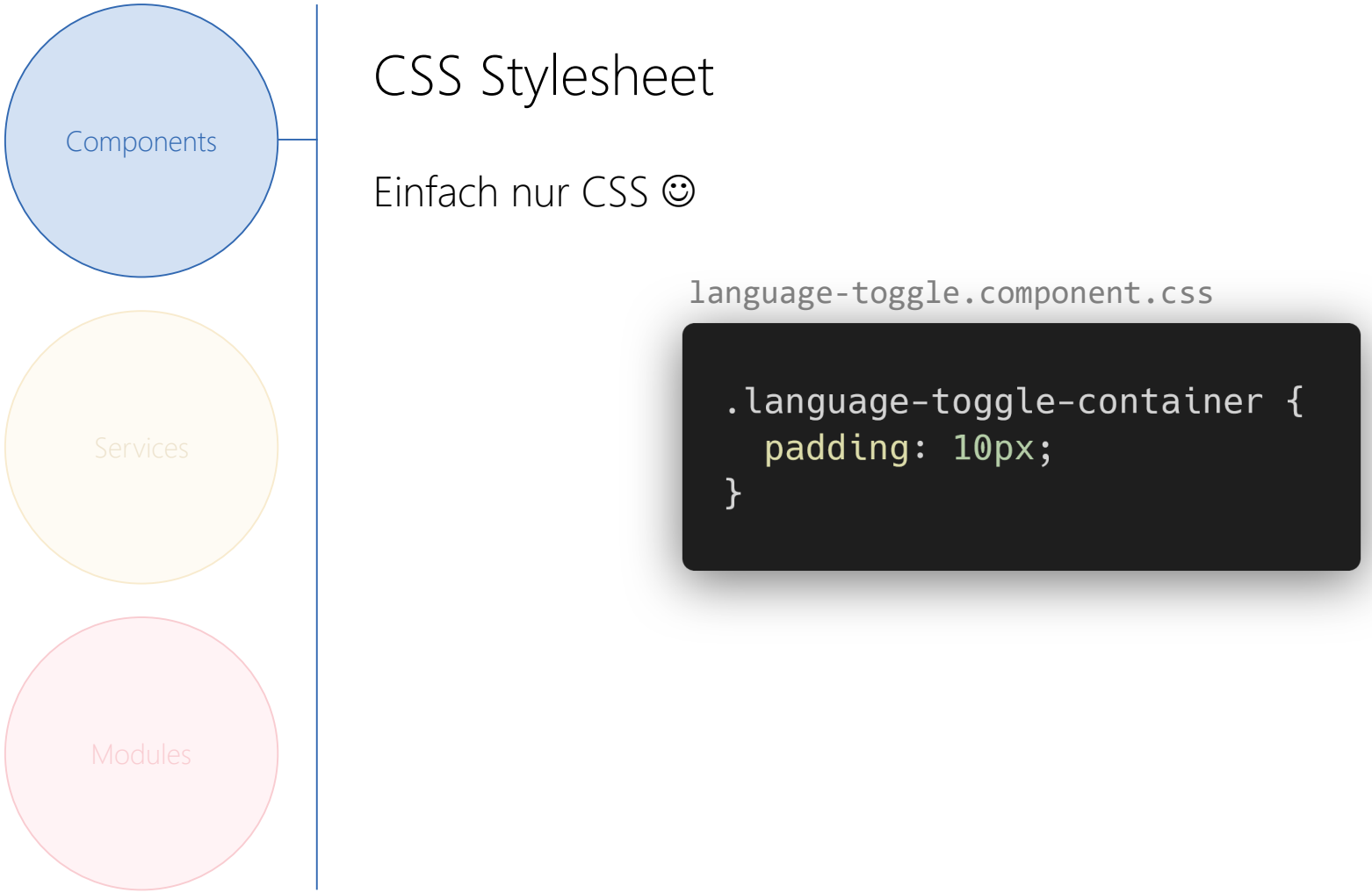



```html
<div class="language-toggle-container">
  <app-language-flag [language]="language"></app-language-flag>
  <span>Language: {{language}}</span>
  <button *ngIf="canSelectGerman()" (click)="selectGerman()">Deutsch</button>
  <button *ngIf="canSelectEnglish()" (click)="selectEnglish()">English</button>
</div>
```

```typescript
@Component({
  selector: 'app-language-toggle',
  templateUrl: './language-toggle.component.html',
  styleUrls: ['./language-toggle.component.css']
})
export class LanguageToggleComponent {

  @Input()
  language: string;

  @Output()
  languageChange = new EventEmitter<string>();

  canSelectGerman(): boolean {
    return this.language !== 'Deutsch';
  }

  canSelectEnglish(): boolean {
    return this.language !== 'English';
  }

  selectGerman(): void {
    this.languageChange.emit('Deutsch');
  }

  selectEnglish(): void {
    this.languageChange.emit('English');
  }
}
```

1. Einführung in Angular

Components



CSS Stylesheet

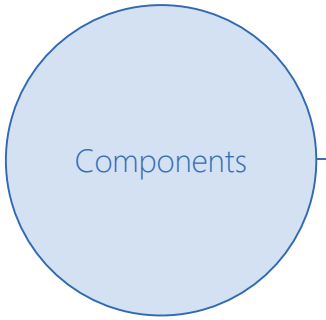
Einfach nur CSS ☺

language-toggle.component.css

```
.language-toggle-container {
  padding: 10px;
}
```

1. Einführung in Angular

Components



CSS Stylesheet

ABER:

Angular stellt sicher, dass CSS Regeln nur in der dazugehörigen Komponente gültig sind!

HTML

```
<button>
```

CSS

```
button { … }
```

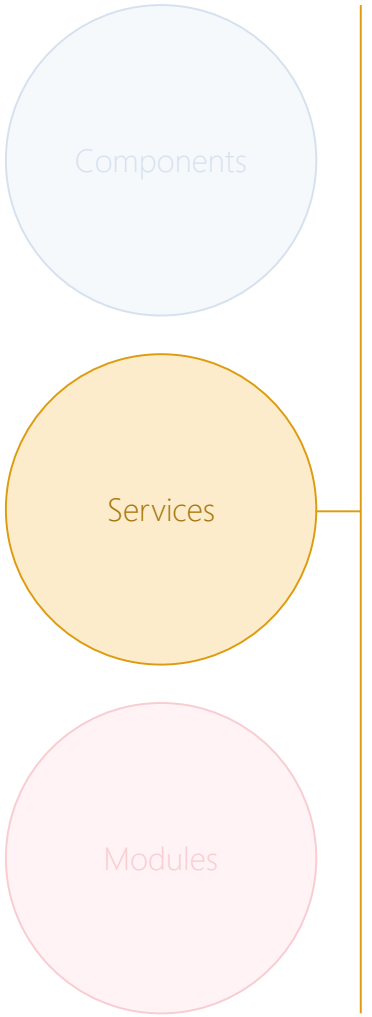
transpiliert

```
<button _ngcontent-pfn-c11>
```

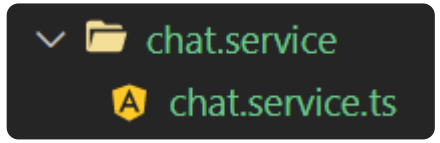
```
button [_ngcontent-pfn-c11] { … }
```

1. Einführung in Angular

Services

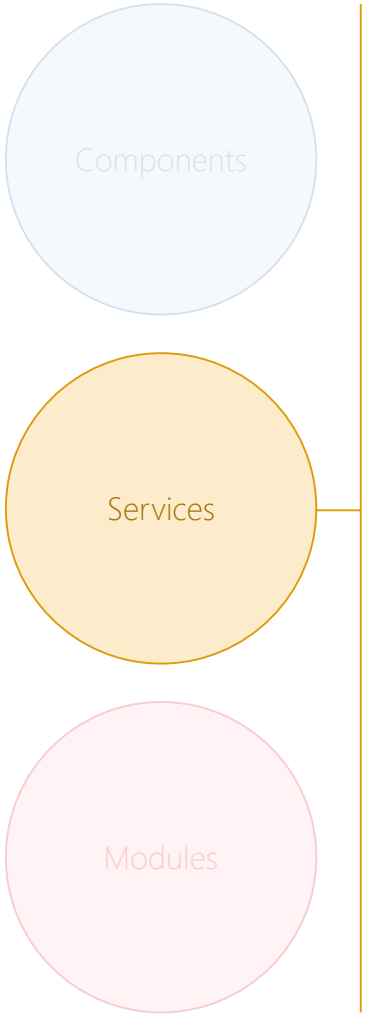


- Enthalten die sogenannte Business Logik
  - d.h. Logik, die nicht direkt mit UI zu tun hat
- Ganz normale Klassen, die
  - speziell markiert werden, damit Angular sie erkennt
  - für Dependency Injection berücksichtigt werden



1. Einführung in Angular

Services

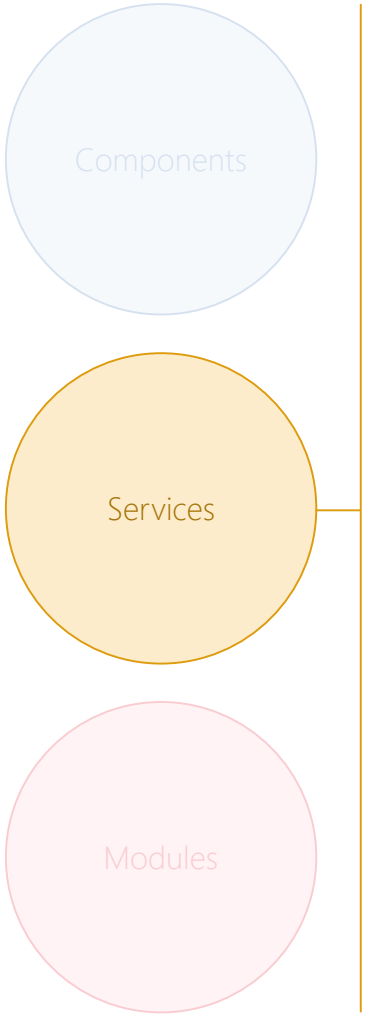


chat.service.ts

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

  constructor(private http: HttpClient) {}

  getMessages(): Observable<Message[]> {
    return this.http.get<Message[]>('https://server.com/api/messages');
  }
}
```

1. Einführung in Angular

Services

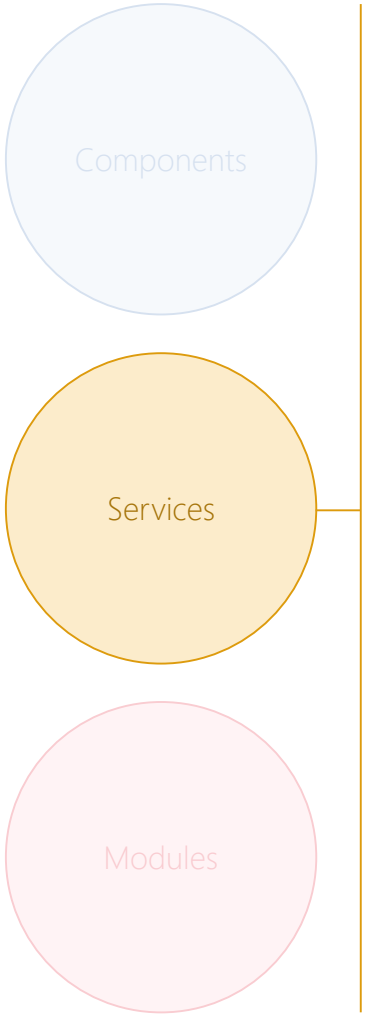


chat.service.ts

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

  constructor(private http: HttpClient) {}

  getMessages(): Observable<Message[]> {
    return this.http.get<Message[]>('https://server.com/api/messages');
  }
}
```

1. Einführung in Angular

Services

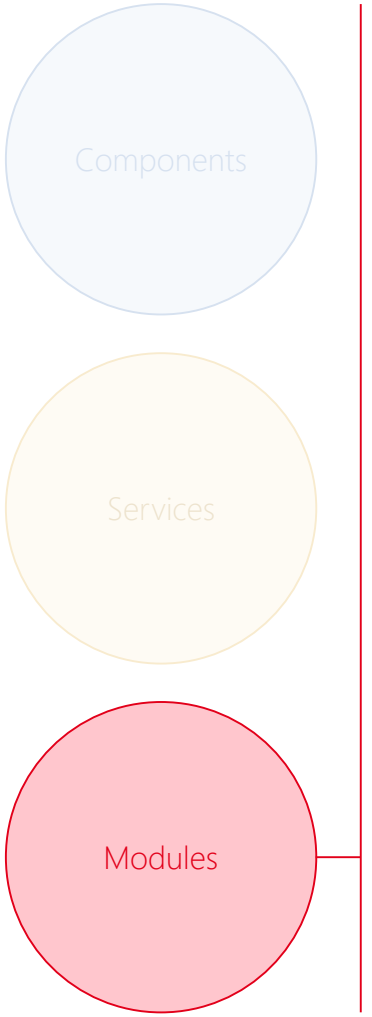


chat.service.ts

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

  constructor(private http: HttpClient) {}

  getMessages(): Observable<Message[]> {
    return this.http.get<Message[]>('https://server.com/api/messages');
  }
}
```

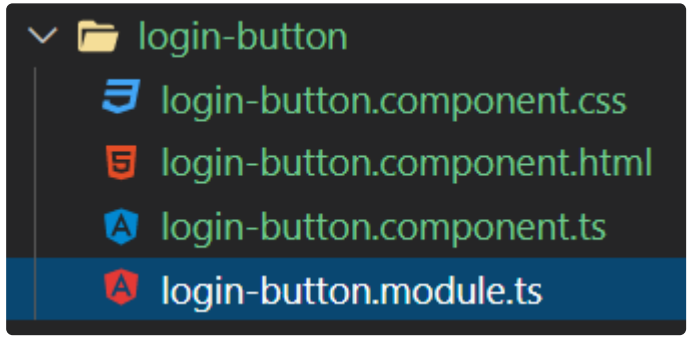
1. Einführung in Angular

Modules



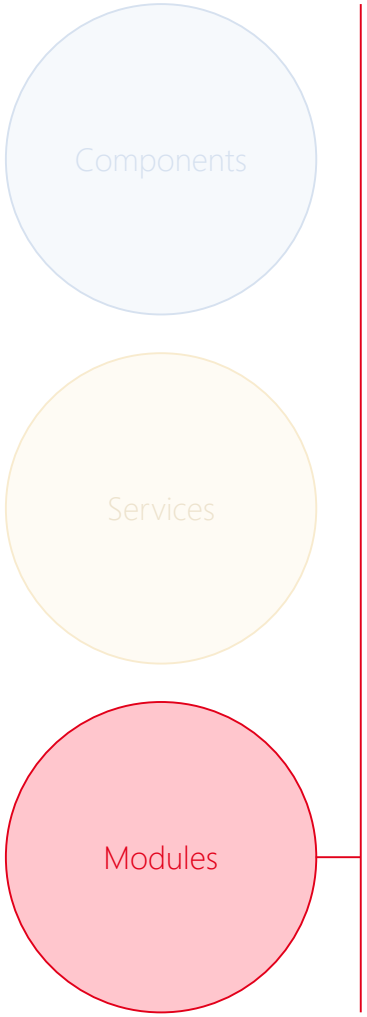
- Woher weiß Angular, wo sich Komponenten befinden? → Module
- Einfach gesagt, Gruppierung aus:
  - `0` oder mehr Komponenten

  UND
  - `0` oder mehr Module



1. Einführung in Angular

Modules



login-button.module.ts

```ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { LoginButtonComponent } from './login-button.component';

@NgModule({
  imports: [
    CommonModule
  ],
  declarations: [
    LoginButtonComponent
  ],
  exports: [
    LoginButtonComponent
  ]
})
export class LoginButtonModule { }
```

1. Einführung in Angular

Wie kann ich mit Angular starten?

1. Node.js installieren (https://nodejs.org)
   Jetzt steht der Kommandozeilenbefehl **npm** zur Verfügung

2. Angular Command Line Interface installieren
   ```
   > npm install -g @angular/cli
   ```

3. Neue Angular Anwendung generieren lassen
   ```
   > ng new chat-app
   ```

4. Anwendung starten
   ```
   > cd chat-app
   > ng serve
   ```

1. Einführung in Angular

Wie kann ich mit Angular starten?

- Beispiel: Angular Komponente generieren

```
> ng generate component <name> --skipTests=true
```

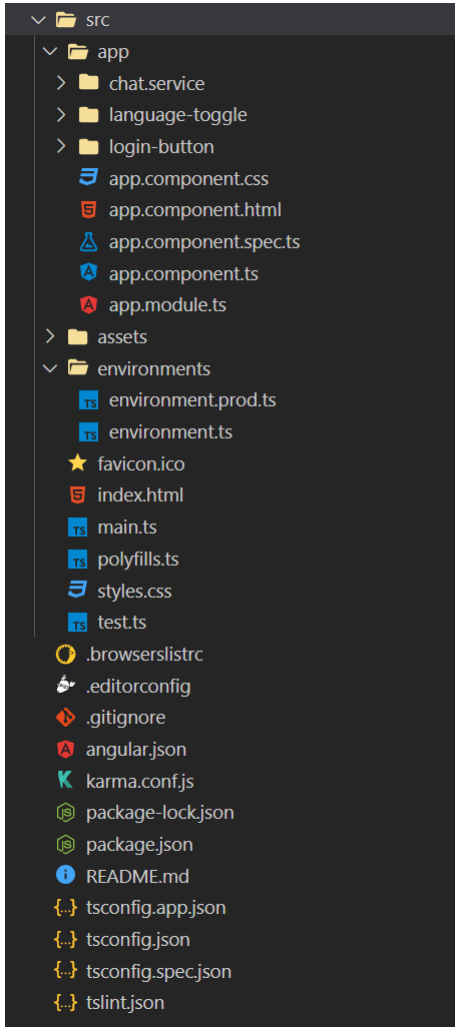
- Beispiel: Angular Service generieren

```
> ng generate service <name> --skipTests=true
```

- Die Dateien werden relativ zum aktuellen Pfad generiert

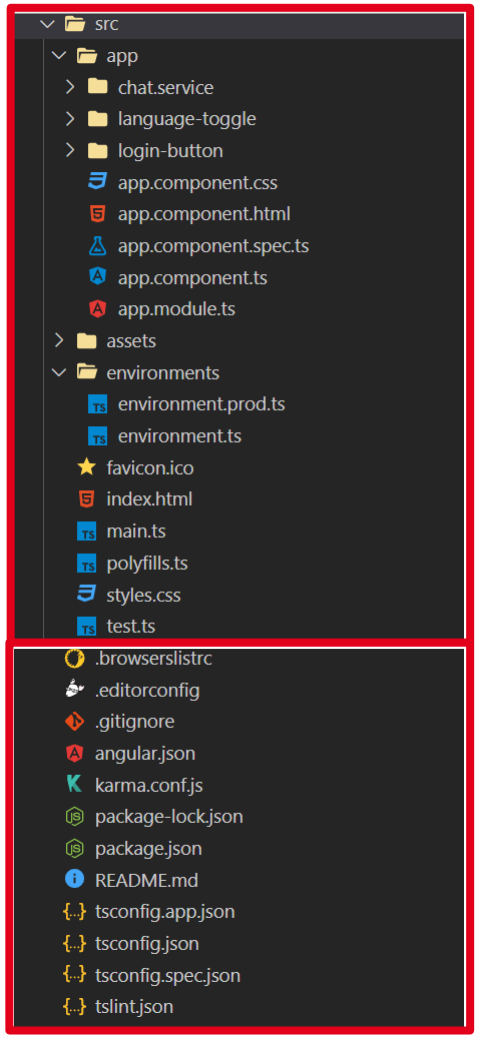
1. Einführung in Angular

Wie ist die Ordnerstruktur?



1. Einführung in Angular

Wie ist die Ordnerstruktur?

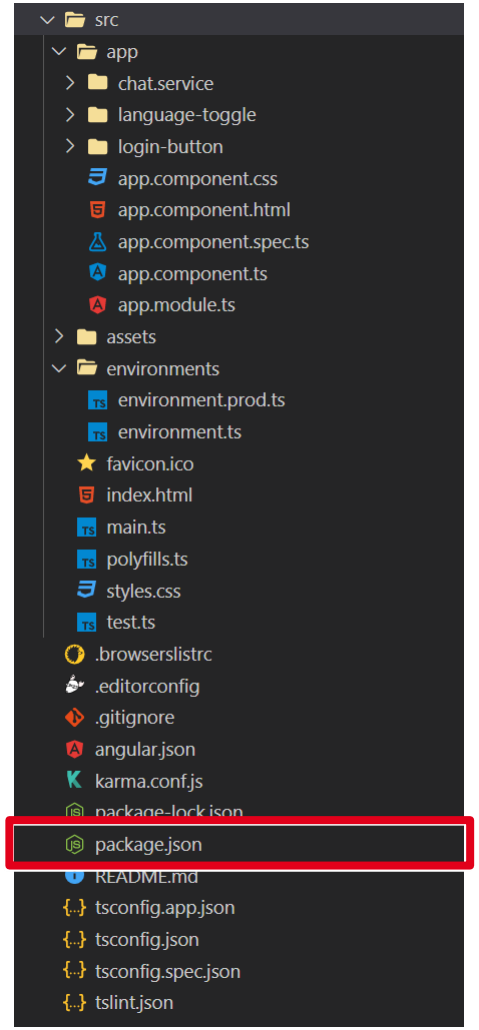


Quelldateien der Anwendung
Werden gebaut und kommen ins Endprodukt

Konfigurationsdateien
Beschreiben wie die Anwendung gebaut wird

1. Einführung in Angular

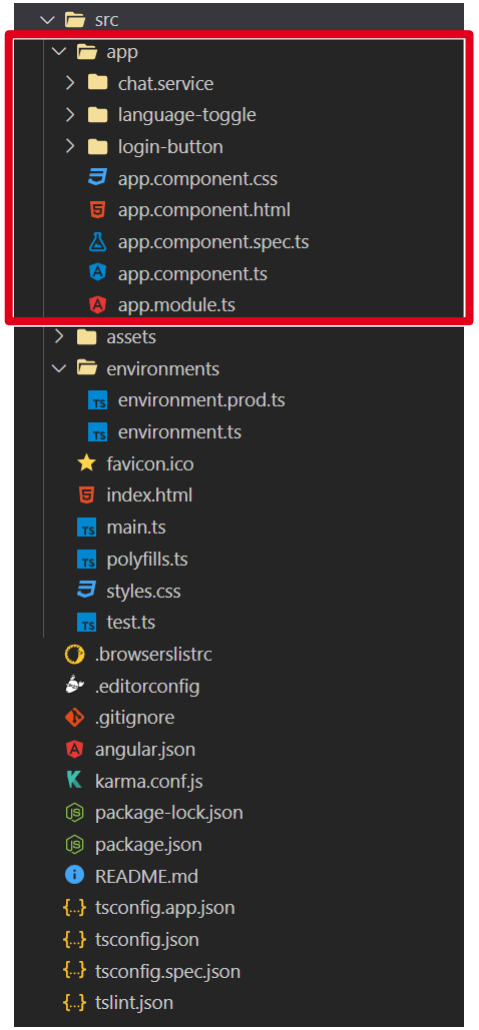
package.json




```
{
  "name": "chat-app",
  "version": "0.0.1",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    ...
  },
  "dependencies": {
    "@angular/animations": "~10.2.0",
    "@angular/common": "~10.2.0",
    "@angular/compiler": "~10.2.0",
    "@angular/core": "~10.2.0",
    "rxjs": "~6.6.0",
    ...
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "~0.1002.0",
    "@angular/cli": "~10.2.0",
    "typescript": "~4.0.2",
    ...
  }
}
```

1. Einführung in Angular

app Ordner

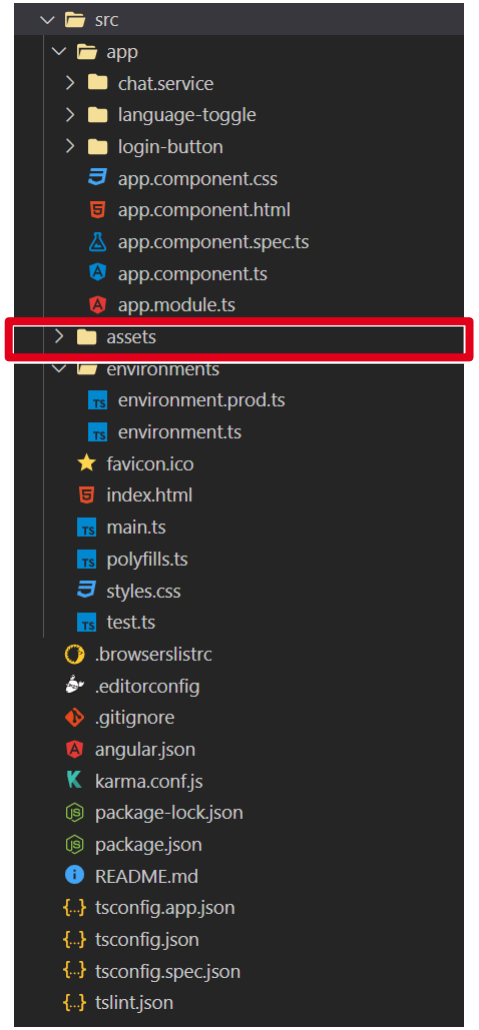


Im app Ordner liegen die Angular Modulen, Komponenten und Services.

Unterstruktur ist Geschmacksache, Hauptsache einheitlich durchgezogen.
Manche Teams gruppieren nach Themen, z.B. „Login“, „Navigation“, etc.
Anderen nach Typ, z.B. „Components“, „Services“, etc.

1. Einführung in Angular

assets Ordner



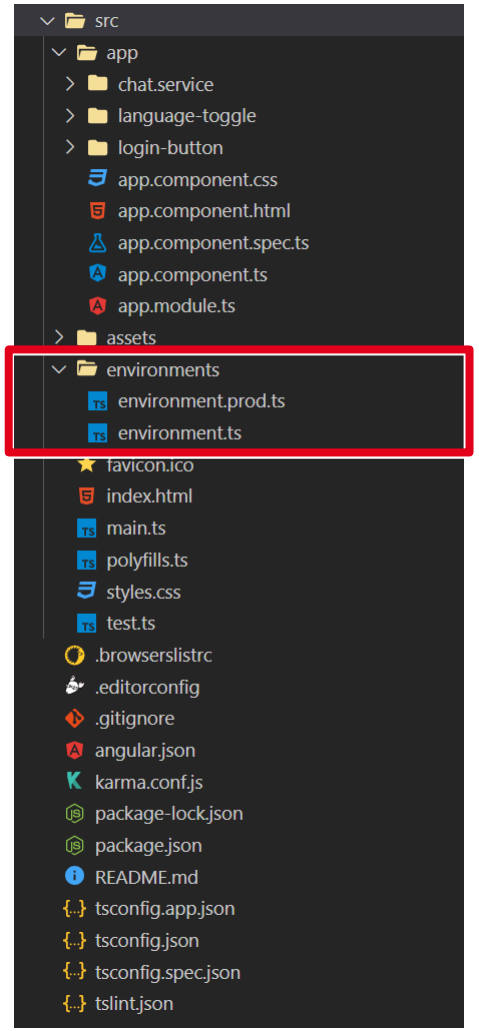
Im assets Ordner liegen statische Dateien, die der Browser braucht.

Beispiele:

- Bilder
- Schriftarten
- Dokumente

1. Einführung in Angular

environments Ordner




```
src > environments > environment.prod.ts > ...
1   export const environment = {
2     production: true
3   };
```

JSON Object mit Umgebungskonstanten

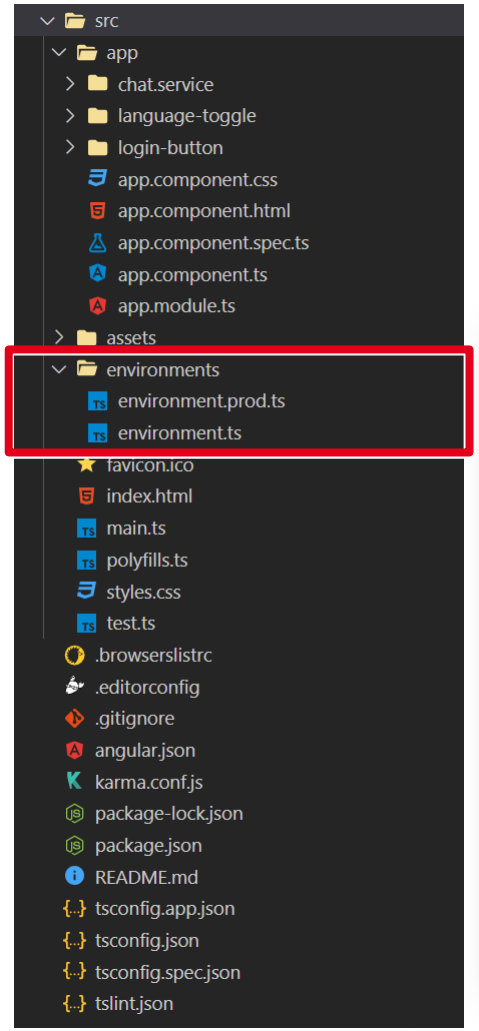
Angular nimmt beim Bauen nur eines von diesen, je nach Baumodus.

Während der Entwicklung:
`environment.ts`

Für die Produktivumgebung:
`environment.prod.ts`

1. Einführung in Angular

environments Ordner



src > environments > environment.prod.ts > ...

```ts
export const environment = {
  production: true
};
```

```ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { environment } from 'src/environments/environment';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    environment.production ? MyDevModule : [],
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

JSON Object mit Umgebungskonstanten

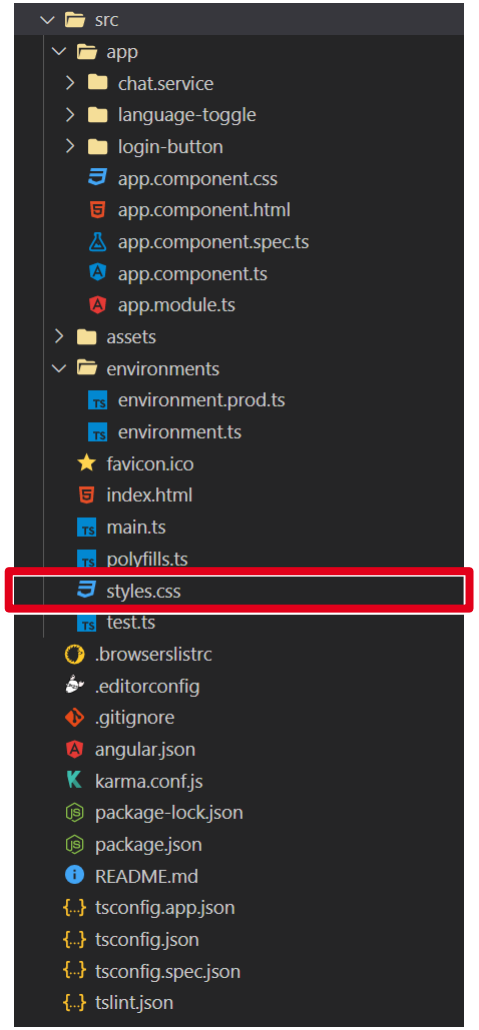
Angular nimmt beim Bauen nur eines von diesen, je nach Baumodus.

Während der Entwicklung:
`environment.ts`

Für die Produktivumgebung:
`environment.prod.ts`

1. Einführung in Angular

styles.css



Globales CSS kann in styles.css deklariert werden.

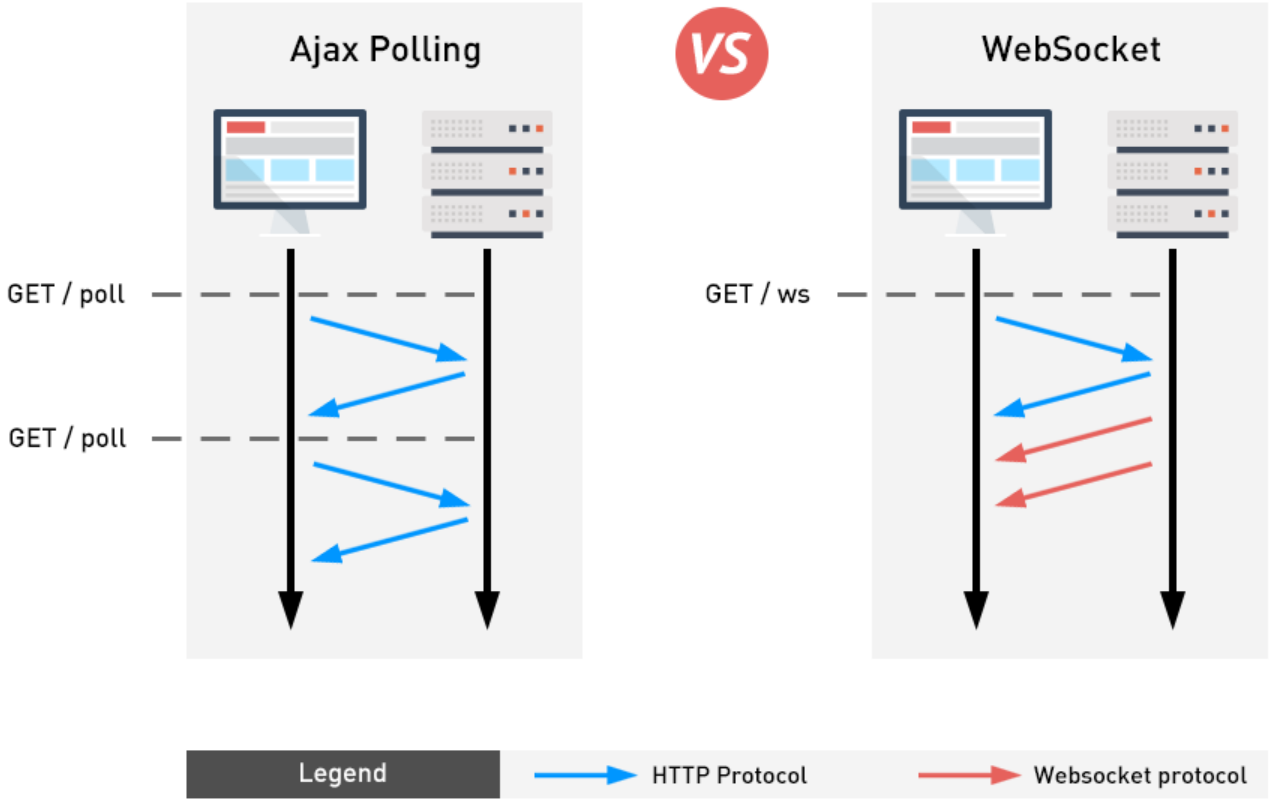
Wirkt auf alle Komponenten der Anwendung.
Nur wenn absolut nötig verwenden!

Programm

1. Einführung in Angular
2. Einführung in Websockets
3. Workshop: Chat App basteln
4. Ergebnisse, Diskussion

Einführung in Websockets

Was ist Websockets?



Quelle: https://blog.resellerspanel.com/wp-content/uploads/2016/03/websocket-scheme.png

Programm

1. Einführung in Angular
2. Einführung in Websockets
3. Workshop: Chat App basteln
4. Ergebnisse, Diskussion