ARS

Motivation

- Modulares System entwickeln
- Systemteile mit klaren Zuständigkeiten und einer einzigen Aufgabe erstellen
- Klares Konzept nutzen, um Daten auf dem Bildschirm anzuzeigen

ARS

Lernziele

- Sie beschreiben die Bedeutung und den Nutzen von Komponenten in Angular.
- Sie beschreiben den grundlegenden Aufbau einer Angular-Komponente.
- Sie erklären, wie Informationen dynamisch mittels Interpolation in HTML ausgegeben werden.
- Sie erklären den Mechanismus, über den eine Webanwendung auf Benutzerinteraktionen reagiert.
- Sie erklären den Verwendungszweck und den Nutzen von Angular-Modulen.

ARS

Komponente

- Grundlegender Baustein einer Angular Applikation
- Kapselt ein Stück View und die zugehörige Darstellungslogik
- Applikation = ein Baum von Komponenten
- Aufbau und Struktur
  - Klasse (sog. Komponentenklasse)
  - Decorator mit Metadaten
  - HTML-Template (Datei)

ARS

Komponentenklasse und Decorator (1|2)

- **Komponentenklasse**
  als Kleber zwischen HTML-Template und Services (Business-Logik)
  - Public Objektvariablen stellen Daten für das Template zur Verfügung
  - Objektmethoden entsprechen Empfänger für Events im Template (z.B. Klick), damit diese passend behandelt werden können

ARS

Komponentenklasse und Decorator (2|2)

- **@Component-Decorator** dokumentiert Metainformationen über die Komponentenklasse für das Angular-Framework
  - **`selector`**
    Entspricht dem HTML-Tag, unter dem die Komponente im HTML-Markup aufgerufen werden kann (später mehr dazu)
  - **`template`**
    Beinhaltet HTML-Elemente der Komponente.
  - **`templateUrl`**
    Name der HTML-Datei, in der sich das HTML-Template befindet
  - **`styleUrls`**
    Array von CSS-Dateien, welche das Aussehen der Komponente bestimmen

ARS

Komponentenaufbau

```
@Component({
    selector: 'create-field',
    templateUrl: 'create-field.component.html',
    styleUrls: ['create-field.component.css']
})
export class CreateFieldComponent implements OnInit {
  task: string;
  constructor(task: string) {
     this.task = task;
  }
  ngOnInit() { }
  storeTask() { ... }

}
```

} Decorator

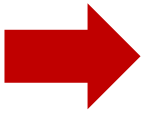
→ Klasse
→ Attribut
→ Konstruktor

→ Lifecycle-Hook
→ Methode

ARS

Komponentenaufbau

```
@Component({
   selector: 'create-field',
   templateUrl: 'create-field.component.html',
   styleUrls: ['create-field.component.css']
})
```



```
<!doctype html>
<html lang="en">
<head>
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <create-field></create-field>
</body>
</html>
```

ARS

Interpolation

- TypeScript-Code im HTML-Template einbinden
  - Beginnt mit {{ und endet mit }}
  - Zwischen den Klammern kann gewöhnlicher TypeScript-Code geschrieben werden (z.B. 1+1)
  - Meistens jedoch Zugriff auf Methoden und Objektvariablen der Komponentenklasse (Binding Template ←→ Komponentenklasse)

ARS

Interpolation – Beispiel

```
@Component({
    selector: 'hello-world',
    templateUrl: 'hello-world.component.html',
    styleUrls: ['hello-world.component.css']
})
export class HelloWorldComponent {
    public greetings: string = 'Hello Angular Component';

    public getComponentName(): string {
        return 'HelloWorldComponent';
    }
}
```

hello-world.component.html

```
<div>What is 1 + 1? {{ 1 + 1 }}</div>
<div>{{ greetings }}</div>
<div>From {{ getComponentName() }}</div>
```

ARS

Angular-Module

- **TypeScript-Modul**
  Definiert was eine Datei nach außen zur Verfügung stellt und was „private“ ist

- **Angular-Modul**
  Repräsentiert den logischen Merkmalsbereich
  - Kapselung von Features, Domänen, Workflow, Utilities, etc.
  - Gruppierung von Artefakten: Komponenten, Direktiven, Pipes, Services
  - Regelung der Sichtbarkeit beim Dependency Injection System von Angular
  - Entspricht einer Klasse mit @NgModule-Decorator

ARS

@NgModule-Decorator

```
@NgModule({
    declarations: [ContactListComponent, ContactComponent],
    exports: [ContactListComponent],
    imports: [BrowserModule],
})
export class ContactModule { }
```

ARS

@NgModule-Decorator

- **declarations**:
  Definiert View-Klassen (Komponenten, Pipes und Direktiven), die zu diesem Modul gehören, damit diese in Templates nutzbar sind
- **exports:** Gibt die Untermenge von View-Klassen (vgl. declarations) an, welche anderen Modulen zur Verfügung stehen (Sichtbarkeit)
- **imports:** Gibt Module an, von denen das aktuelle Modul abhängig ist (bspw. weil es dort exportierte View-Klassen nutzt)
- **bootstrap:**
  Nur im App-Module verfügbar;
  Definiert welche Komponente die Root-Komponente ist

ARS

Angular-Module – Zusätzliche Hinweise

- Jede Angular-App muss zumindest ein Modul definieren. nämlich das App-Modul
- Eine neue Komponente, Pipe oder Direktive erstellt?
  → Eintrag in **declarations** nicht vergessen
- Über **exports** lässt sich die Sichtbarkeit
  von Komponenten, Pipes und Direktiven steuern (private)

ARS

Zusammenfassung

Um was ging es in diesem Modul?

- Grundlegende Bausteine von Angular: Komponente und Modul
- Mechanismen der Realisierung
  - HTML-Template
  - Interpolation
- Dynamische Ausgabe von Informationen im HTML

Wozu brauche ich das? Was will ich damit machen?

- Komponenten sind der zentrale Baustein einer Angular Anwendung
- Komponenten stellen Daten zur Anzeige bereit
- Wartbarkeit sicherstellen durch Strukturierung in Komponenten und Module

ARS

Kontrollfragen

- Was sind üblicherweise die Bestandteile einer Komponente?
- Welche Möglichkeiten existieren,
  um ein Template für eine Komponente anzugeben?
- Wo muss eine Komponente bekannt gemacht werden,
  damit sie im HTML-Template anderer Komponenten nutzbar ist?
- Mittels welcher Syntax können z. B. Objektvariablen
  einer Komponentenklasse im Template
  ausgegeben werden?
- Wie kann eine Webanwendung auf
  Benutzerinteraktionen reagieren?