Hallo Angular 2!

Einstieg in die Entwicklung mit Angular 2

Gregor Biswanger | Freier Dozent, Berater, Trainer und Autor

about.me/gregor.biswanger

Über mich

Gregor Biswanger

Microsoft MVP, Intel Black Belt &
Intel Software Innovator

- Freier Dozent, Berater und Trainer
- Schwerpunkte Softwarearchitektur, XAML, Web und Cross-Plattform Entwicklung
- Technologieberater für die Intel Developer Zone
- Sprecher auf Konferenzen und User Groups
- Freier Autor für heise.de, dotnetpro, WindowsDeveloper und viele weitere Fachmagazine
- Video-Trainer bei video2brain und Microsoft

dotnet-blog.net
about.me/gregor.biswanger

Agenda

- Einleitung
  - Was ist Angular?
  - Vorteile
  - Die Neuheiten und Unterschiede
  - Die Anatomie und der Aufbau
- Sprachauswahl
  - Was ist TypeScript?
- Das Setup
  - Die AngularCLI
- Angular 2 im Detail
  - Einführung in Components
  - Templates
  - Built-In Directives
  - Components als Directive
  - Data-Binding
  - Pipes
  - Services
  - Dependency Injections
  - Routing und Navigation
  - Angular Module
- Zusammenfassung

Angular in aller Munde

Aber was ist das eigentlich?

Was ist Angular?

Ein JavaScript Framework

Zur Entwicklung von Client-Side Apps

Verwendet HTML, CSS und JavaScript

Angular ist eine Plattform...

- Unterstützt
  - HTML und XML
  - ES5, ES 2015 und TypeScript
  - DOM und native
- Wie geschaffen für das mobile Web mit
  - Material Design
  - Angular Universal
  - Web Workers
- Unterstützung nativer App-Entwicklung mit Ionic und NativeScript

Vorteile

Wieso Angular 2 sich lohnt!

Welche Vorteile bringt Angular 2 mit sich?

Schnell

Mächtig

Sauber

Simpel

Unterstützung aller gängigen Webbrowser

Internet Explorer
9, 10, 11 und Edge

Google Chrome

Mozilla Fireforx

Safari

Von AngularJS zu Angular 2.

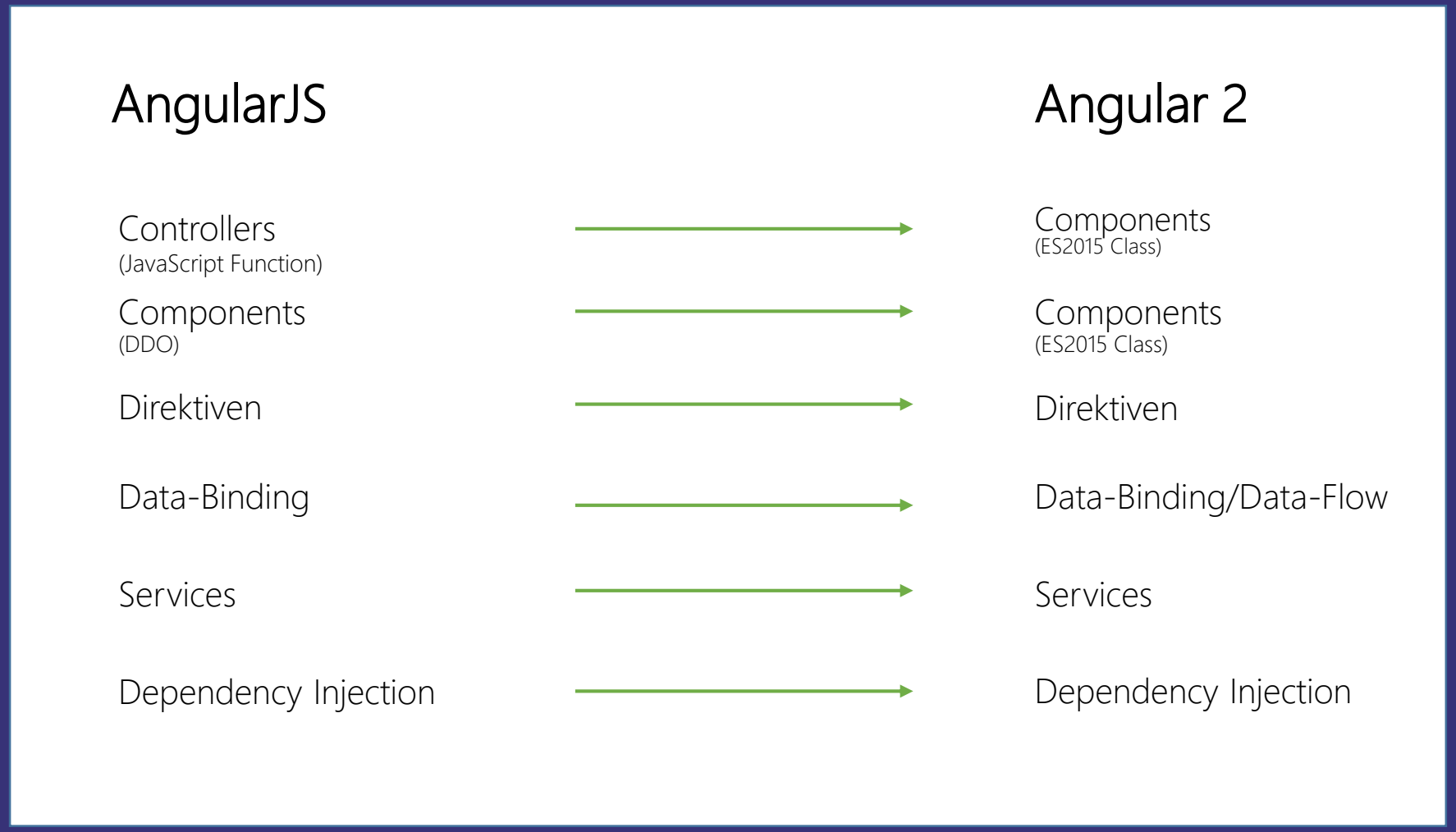
Was sind die Unterschiede?

Können wir alles aus AngularJS einfach vergessen?

Eine Neuheit mit alt bewährtem Konzept

- Angular 2 ist neue Implementierung des Konzeptes hinter AngularJS.
- Es ist kein Update zu AngularJS
- Angular 2 wurde auf die neuesten Web-Technologien aufgebaut
  - Npm
  - TypeScript
  - EcmaScript
  - HTML5
  - web workers
  - shadow dom
- Augular2 wurde für das moderne Web entwickelt
  - mobile Browser
  - Unterstützung aller gängingen Browser
  - Serverseitiges Rendern

Von AngularJS zu Angular 2



Ein kleiner Vergleich

AngularJS

```
(function () {
  angular
    .module('app', []);
})();


<html ng-app="app">
```

Dependencies

Root module

Angular 2

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

@NgModule({
  imports: [BrowserModule],
  declarations: [ ... ],
  bootstrap: [ ... ]
})
export class AppModule { }
```

Dependencies

Root module

Controller vs. Component

AngularJS

```
<body ng-controller="ProductController as vm">
    <h3>{{vm.product.name}}</h3>
</body>
(function () {
   angular
       .module("app")
       .controller("ProductController", ProductController);

   function ProductController() {
      var vm = this;
      vm.product = { name: "Surface Pro 4" };
    }
})();
```

Angular 2

```
<my-product></my-product>

import { Component } from '@angular/core'>

@Component ({
      selector: 'my-product',
      templateUrl: 'product-details.component.html'
})

export class ProductComponent {
        products: Product[] = { name: "Surface Pro 4" };
}
```

Built-In Direktiven

AngularJS

```
<ul>
    <li ng-repeat="product in vm.products">
        {{product.name}}
    </li>
</ul>
<div ng-if="vm.products.length">
    <p>Your shopping cart contains {{vm.products.length}}
products</p>
</div>
```

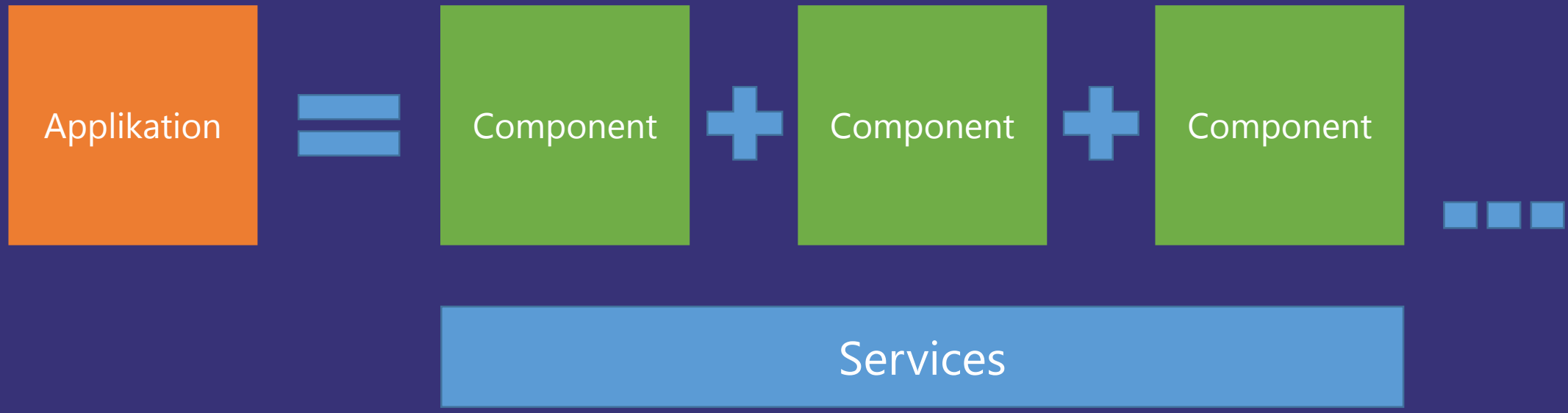
Angular 2

```
<ul>
    <li *ngFor=" let product of products">
        {{product.name}}
    </li>
</ul>
<div *ngIf ="products.length">
    <p>Your shopping cart contains {{products.length}}
products</p>
</div>
```

Die Anatomie

So ist Angular 2 aufgebaut

Die Anatomie einer Angular 2 Applikation



Component

Sprachauswahl

Welche Sprache soll´s denn sein?

Auswahl der gewünschten Sprache

ES 5	ES 2015	TypeScript	Dart
• Läuft im Browser • Kein Kompilieren notwendig	• Viele neue Features (Klassen, let, arrow-funtions, uvm.)	• Basiert auf JavaScript • Gutes IDE Tooling • Typsicher	• Kein JavaScript

Was ist TypeScript?

- TypeScript ist eine Open-Source Sprache
- Basiert auf JavaScript
- Kompiliert zu einfachem JavaScript Code
- Eine typsichere Sprache
- Implemetiert „Class-based object orientation“ aus ES 2015

Zum TypeScript Playground:
http://www.typescriptilang.org/Playground/

Das Setup

Aller Anfang ist schwer!

Die ersten Schritte

- Installation des Node Package Managers (npm)
  Download Link: https://www.npmjs.com/
- Angular 2 Applikation anlegen
  - Erstellen eines App-Verzeichnisses
  - Hinzufügen der package definition und configuration files
  - Installation der Pakete
  - Erstellen des App-Modules
  - Erstellen der main.ts Datei
  - Hinzufügen einer index.html

Entweder, oder…

- Manuelles Setup anhand der Schritt für Schritt Anleitung
  www.angular.io
- Download des fertigen Setups
  https://github.com/angular/quickstart
- AngularCLI
  https://github.com/angular/angular-cli

Die AngularCLI

Der kleine Helfer...

Was ist die AngularCLI

- Ein Command Line Interface –Tool für Angular 2
- Hilft beim Setup einer Angular 2 Anwendung
- Aktuell noch in der Beta Version

Anwendung der AngularCLI

- Installation erfolgt über npm
  npm install –g angular-cli
- Erstellen einer neuen Angluar 2 Anwendung
  ng new PROJECT_NAME
- Generieren neuer Components, Klassen, Direktiven, Services, uvm.
  z.B. ng generate component MY_COMPONENT
  oder ng g component MY_COMPONENT
- Builden
  ng build
- Unit Tests starten
  ng test

Angular 2

Ein Einblick in die Tiefe

Components

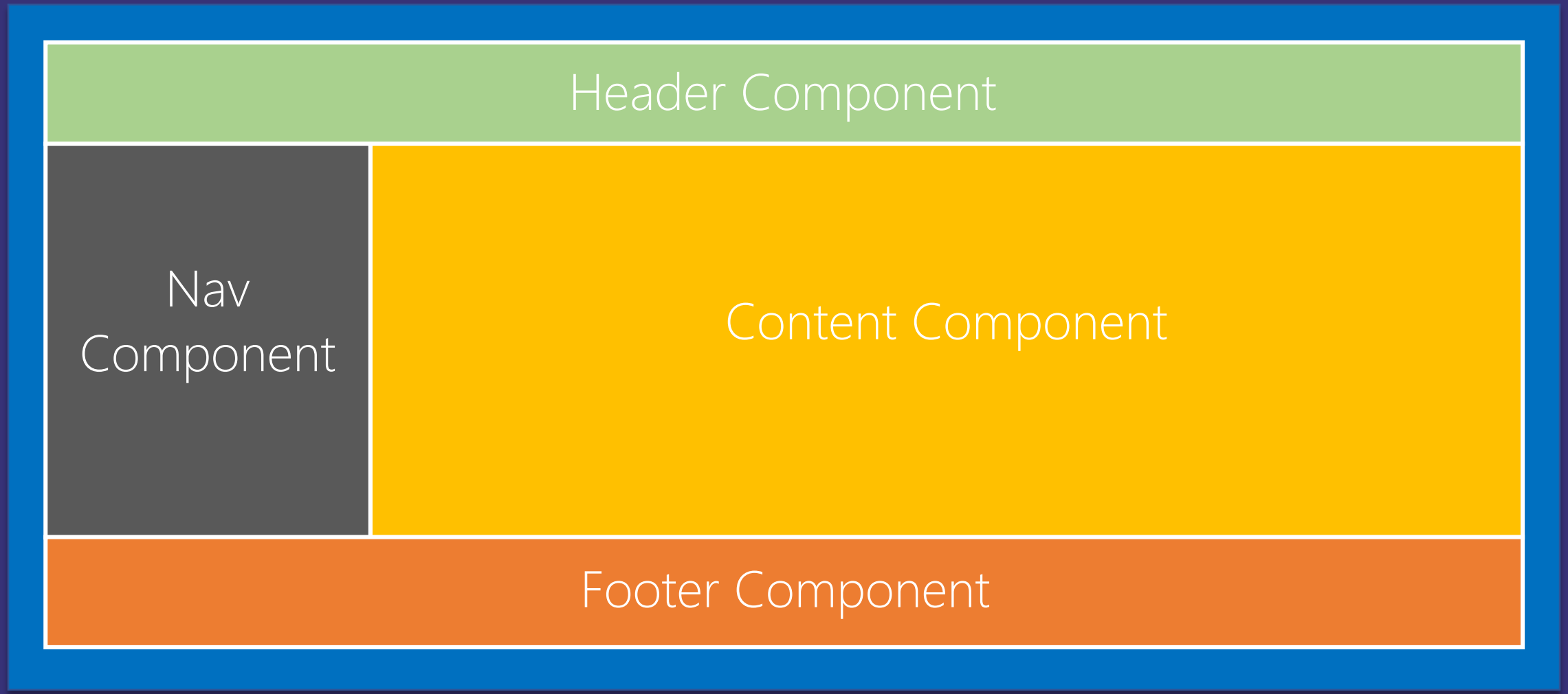
```
import { Component } from '@angular/core'>

@Component ({
        selector: 'product-details',
        templateUrl: 'product-details.component.html'
})

export class ProductDetailsComponent {
        products: Product[];
}
```

Importieren des Modules

Decorator (beschreibt die Komponente)

Class (definiert die Komponente)

Aufbau einer App anhand von Komponenten



Funktion einer Komponente

product-details.component.ts

```
@Component ({
        selector: 'product-details',
        templateUrl: 'product-details.component.html'
})
export class ProductDetailsComponent {
        productName = 'Surface Pro 4';
}
```

product-details.component.html

```
<h1>{{productName}}</h1>
```

index.html

```
<product-details>Daten werden geladen...</product-details>
```

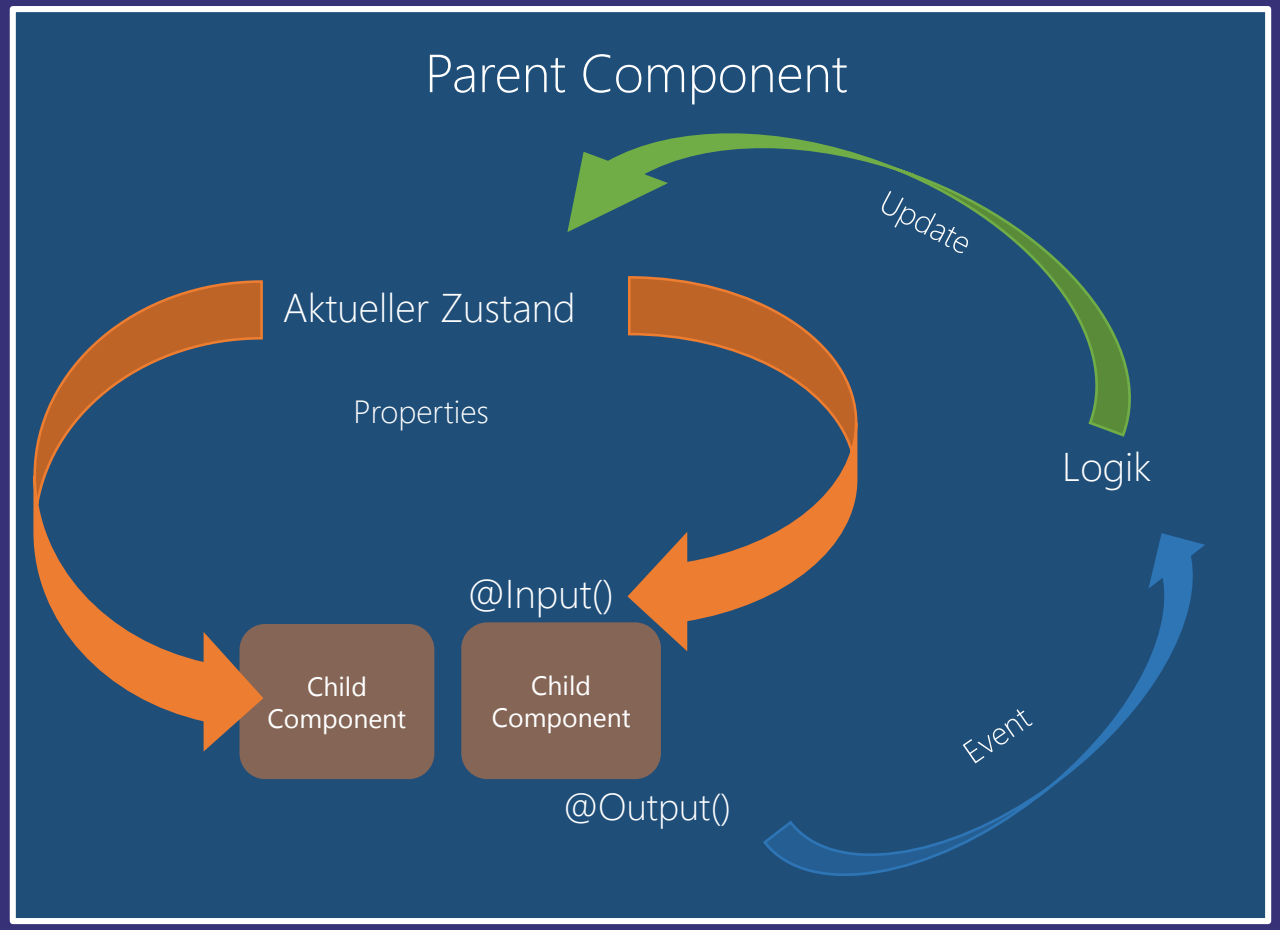
Die Logik der Komponente

wird gerendert

wo die Komponente hinterlegt ist

Nested Components

- Die Parent Component gibt aktuellen Zustand an die Child Components weiter
- Keine Veränderung des Zustands der Eltern durch die Kinder
- Child Components können auf Logik der Eltern zugreifen (Events, Properites, Actions,…)
- Angular 2 Properties *@Input()* und *@Output()* dienen dem **Uni-Directional Data-Flow**



Templates

Inline Template

```
template:
"<h1>{{product.name}}</h1>"
```

Inline Template

```
template: `
<div>
    <h1>{{product.name}}</h1>
    <div>
        {{product.description}}
    </div>
</div>
`
```

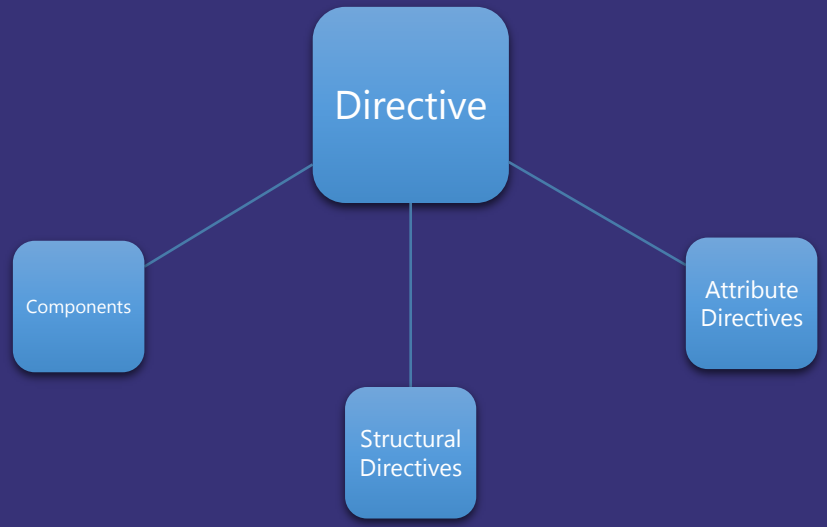
ES 2015
Back Ticks

Linked Template

```
templateUrl:
"product-details.component.html"
```

Built-in Directives

- Class und Style Directives
  - ngClass (z.B. [ngClass]= “{selected: isSelected, color: selectedColor}”)
  - ngStyle (z.B. [ngStyle]= “{font: myFont}”)
- Structural Directives
  - ngFor
  - ngIf
  - ngSwitch



Nutzen der Built-in Directives

- Importieren des BrowserModules in der App

```
import { BrowserModule } from '@angular/platform-browser';
```

- BrowserModule zur Angular Module's List hinzufügen

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

Component als Directive

app.component.ts

```
@Component ({
    selector: 'pm-app',
    template: '
        <div>
            <h1>{{product.name}}</h1>
            <div>
                <product-details></product-details>
</div>
        </div>'
})
export class AppComponent { }
```

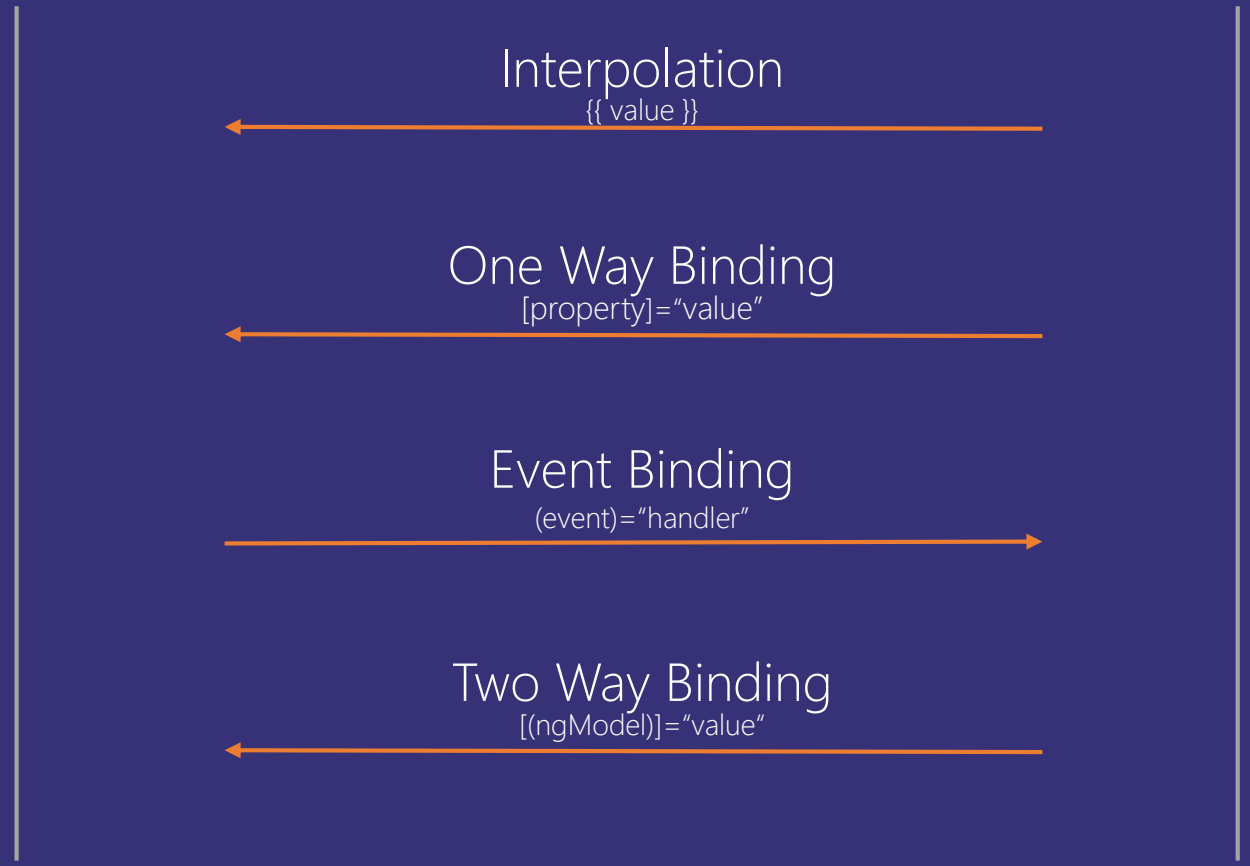
product-details.component.ts

```
@Component ({
                selector: 'product-details',
                templateUrl: 'product-details.component.html'
})
export class ProductDetailsComponent { }
```

Component als Directive

```
import { AppComponent }  from './app.component';
import { ProductListComponent }  from './products/product-list.component';
@NgModule({
  imports: [ BrowserModule ],
  declarations: [
    AppComponent,
    ProductListComponent ],
  bootstrap: [ AppComponent ]
})
```

Data-Binding



Data-Binding

Interpolation

```
<h1>{{product.name}}</h1>
```

Property Binding

```
<h1 [innerText]="product.name"></h1>
```

Binding an jedem HTML Element Property möglich.

Event Binding

```
<button (click)="SayHello()">Click me</button>
```

Two Way Binding

```
<input [(ngModel)]="product.name"/>
```

Pipes

- Built-In Pipes
  - date
  - number, decimal, percent, currency
  - json
  - uvm.
- Custom Pipes
  - @Pipe Decorator
  - *PipeTransform* Interface Implemtierung
  - Rückgabewert
  - Arguments (optional)

Beispiele:

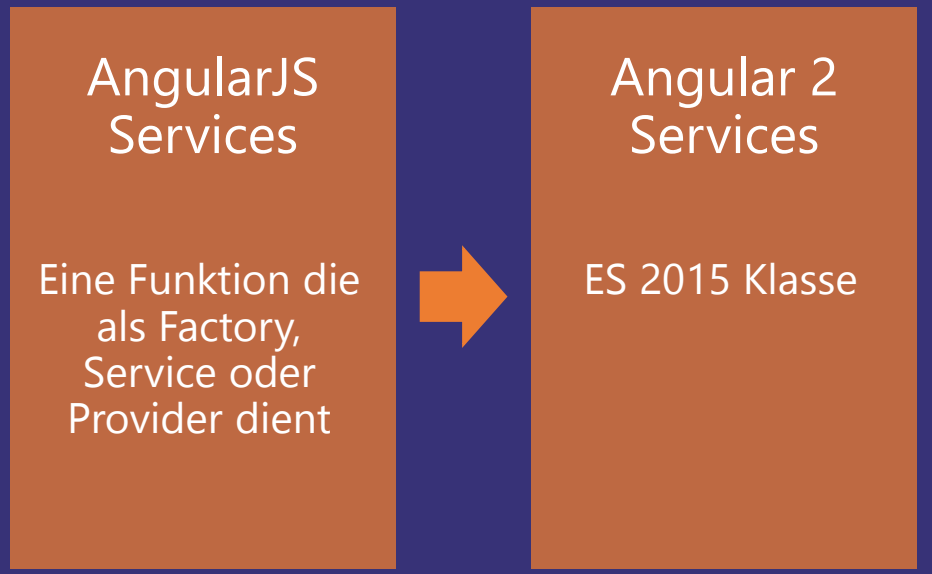
```
{{ productName | lowercase }}
{{ productPrice | currency | EUR }}
{{ expirationDate | date:format }}
```

Beispiel:

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'myPipe'})
export class MyPipe implements PipeTransform {
  transform(value: string, args: string[]): any {
        return value.toLowerCase()
        .replace(/(?:^|\s)[a-z]/g, x => x.toUpperCase());
    });
  }
}
```

Services

- Dient zur Abfrage relevanter Daten
- Beinhaltet Daten und Logik, die mit mehreren unterschiedlichen Komponenten geteilt werden muss (z.B. ein ExceptionHandler oder einem Message service)



Aufbau eines Services

Bulild

- Klasse erstellen
- Decorator definieren
- Daten importieren

Register

Einen Provider

Kann einen Service erstellen oder zurückgeben.

Meist die Service Klasse selbst

In einer Komponente

Zugriff nur von der Komponente und deren Kindern möglich

Im Angular Module

Zugriff von überall möglich

Inject

- Über den Konstruktor

Dependency Injections

Als Dependency Injection bezeichnet man den Vorgang, eine Klassen-Instanz für ein anderes Angular Feature zur Verfügung zu stellen.

In einer Komponente

```
export class ProductDetailsComponent {
    products: Product[];

    constructor(private productService: ProductService) { }
}
```

Der Service wird über den Konstruktor angelegt

In einem Service

```
@Injectable()
export class ProductService {
    constructor(private http: Http) { }

    getProducts() {
        return this.http.get(productListUrl);
    }
}
```

Das Anlegen erfolgt ebenfalls über den Konstruktor

Informiert über bestehende Injectables innerhalb des Services

Routing und Navigation

- Routing erlaubt der Applikation das Navigieren zwischen den unterschiedlichen Komponenten
- Erlaubt das Durchreichen von Parametern

Vorgehensweise beim Routing

Routing in die App einbringen

- RouterModule importieren um den Zugriff auf alle Routing Features zu erhalten
- Routes importiern, um Routes anlegen und definieren zu können
- Definieren und Importieren eines Modules mittels der Routes
- Exportieren des neu angelegten Routing Modules

```
import { Routes, RouterModules } from '@angular/router'>
```

```
const routes: Routes = [
    { path: '', pathMatch: "full", redirectTo: "products" },
    { path: 'products', component: ProductComponent },
    { path: '**', pathMatch: "full", component: PageNotFoundComponent }
];
```

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

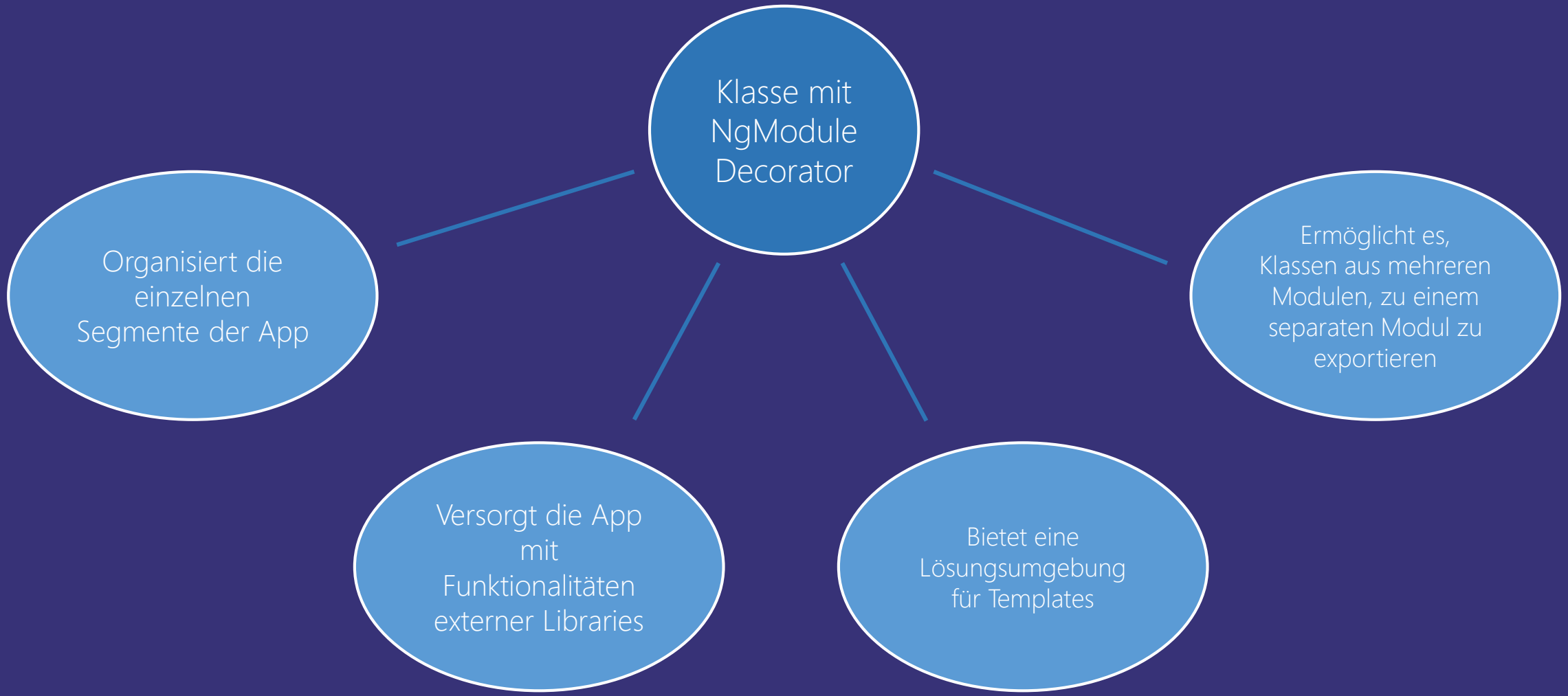
Navigation einbinden

- RouterLink-Direktive im HTML einbinden und Parameter festlegen
- RouterOutlet anlegen um zu definieren, wo der Inhalt angezeigt werden soll

```
<nav>
  <ul>
    <li><a [routerLink]="["/products"]" href="">Products</a></li>
  </ul>
</nav>
```

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

Angular Module



Angular Module

- Deklariert jede Komponente, Direktive und Pipe
- Lädt die App Component
- Exportiert andere Module (z.B. 3rd Party und @angular Module), sowie Komponenten, Direktiven und Pipes
- Importiert andere Module (RouteModule, @angular Module, ...)
- Registriert Services

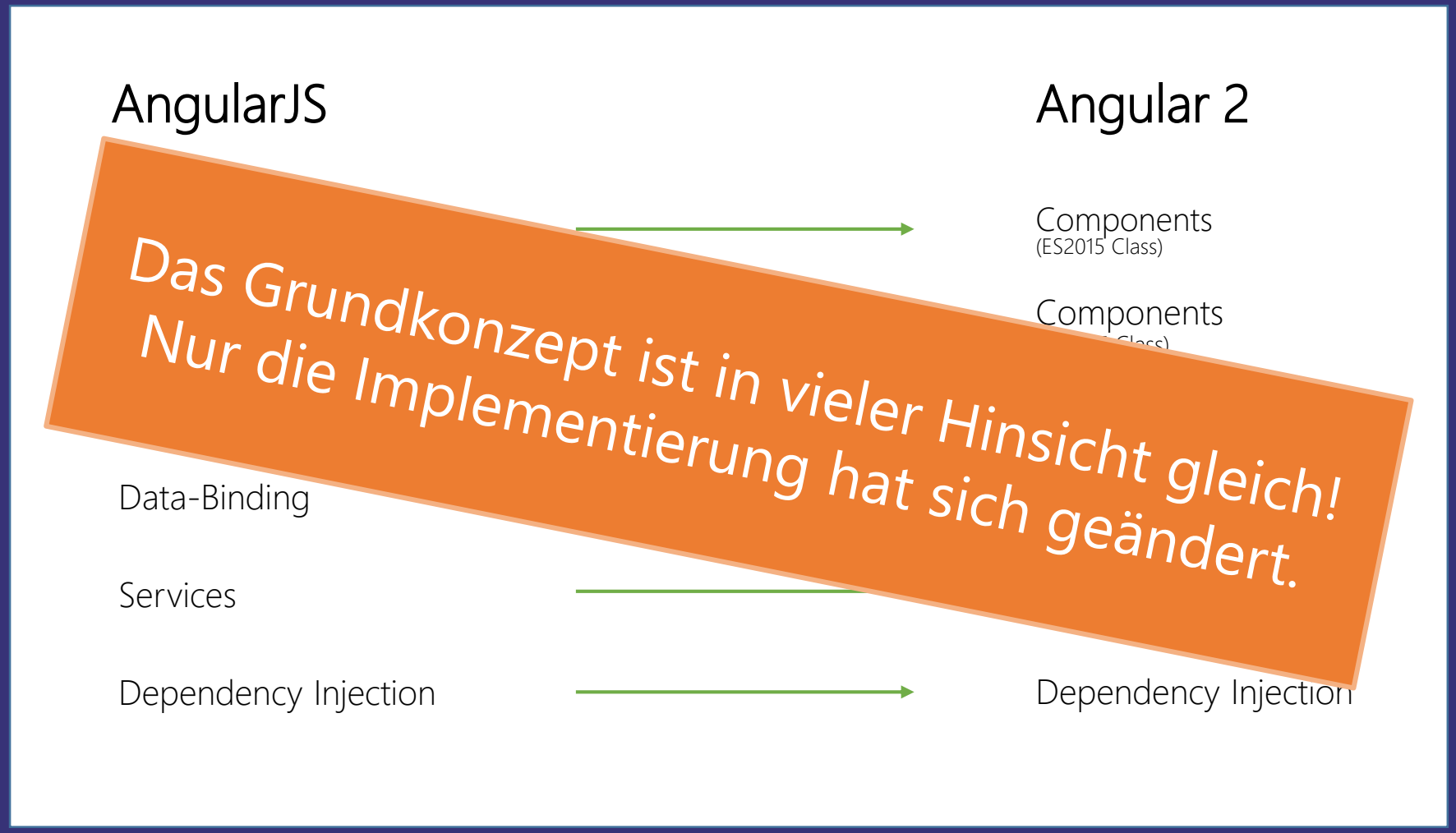
Angular Module Funktionen

Zusammenfassung

Wir haben gelernt …

- Was eine Komponente ist und aus was diese besteht (Klasse, Template und Daten)
- Wie wir unseren HTML Code in ein Template verpacken (Inline oder Linked Template)
- Welche unterschiedlichen Data-Binding Optionen es gibt und wie diese angewendet werden (Interpolation, Property-, Event- und Two-Way Binding)
- Wofür wir einen Service benötigen und wie dieser angewendet wird

Fazit


AngularJS
Angular 2
Components
(ES2015 Class)
Components
Data-Binding
Services
Dependency Injection
Dependency Injection
Das Grundkonzept ist in vieler Hinsicht gleich!
Nur die Implementierung hat sich geändert.