Angular Training

Components and Data Binding

Peter Bouda, hey@peterbouda.eu

ng-lisbon.com

What is a component?

- Basic building blocks of an Angular app
- The visible elements of a web application, like header, menu, product list
- Consist of a class (logic) and a template (view)
- We use them in a component hierarchy to construct the app

ng-lisbon.com

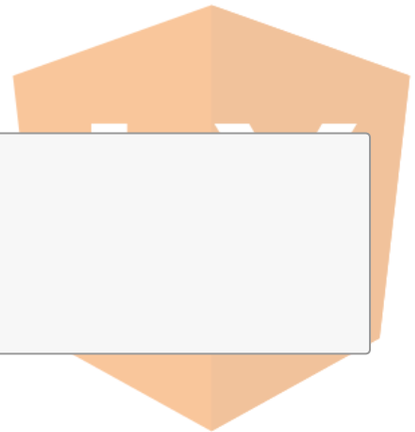
Using components

In app.component.template.html:

```
<div class="container">
        <app-header></app-header>
        <app-products></app-products>
</div>
```

In products.component.template.html:

```
<ul>
        <li><app-product-item [id]="1"></app-product-item></li>
</ul>
```



Using the CLI to create components

Create a component:

```
$ ng generate component product
```

With inline styles, without tests, no sub-directory:

```
$ ng g c header -is --spec false --flat
```

Components will be created below the app folder or in the current directory

A simple component

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

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styles: []
})
export class HeaderComponent {
  menuItem1: string = 'Home';
  menuItem2: string = 'Produkte';
  menuItem3: string = 'Über uns';

  constructor() { }

}
```

The template with string interpolation

```
<ul>
  <li>{{ menuItem1 }}</li>
  <li>{{ menuItem2 }}</li>
  <li>{{ menuItem3 }}</li>
</ul>
```

Styles

- Angular supports inline styles or external stylesheets
- Support for different pre-processors like SCSS, SASS, ...
- Styles are scoped via Shadow DOM (emulation)
- Gobal styles in `app/styles.css`
- Or add .css files in .angular-cli.json

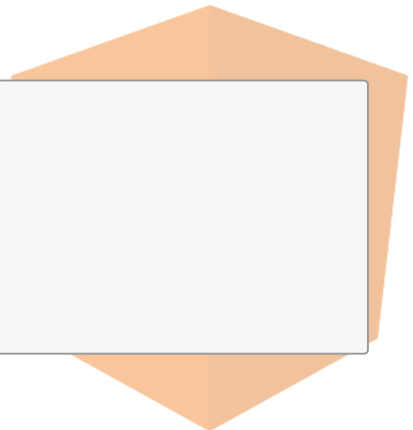
ng-content

With ng-content HTML is passed from parent to child:

```
<app-header>
  <h2>Der zweite Titel!</h2>
</app-header>
```

And then in `header.component.html`:

```
<ul>
<!-- Some code -->
</ul>
<ng-content></ng-content>
```



Input

- Components can receive data: think of arguments of a function call
- Generally: property binding via `[attributeName]="data"`
- The `@Input` decorator defines a class property as input
- Optionally you can pass a parameter to `@Input` to rename the input

ng-lisbon.com

Input

```
import { Component, Input } from '@angular/core';

…

export class HeaderComponent {
  …
  @Input('addThis') menuItem4: string;

  constructor() { }
}


<app-header [addThis]="'Only in the header'"></app-header>
```

Output

- Components can output data as events: think of (asynchronous) return values of function calls
- Generally: event binding via `(eventName)="onEvent()"`
- The `@Output` decorator defines a class property as output
- The output is an `EventEmitter` of a certain type

ng-lisbon.com

Output

```
import { Component, Input, Output, EventEmitter } from '@angular/core';

…
export class HeaderComponent {
  …
  @Output() selected = new EventEmitter();

  constructor() { }
}

<app-header [addThis]="'Only in the header'" (selected)="onSelect()"></app-heade
```

Overview of bindings and template syntax

- Property binding
  - From DOM to component
  - `[attributeName]="data"`
- Event binding
  - From component to DOM
  - `(eventName)="doSomething()"`
- Two-way binding
  - From component to DOM and back
  - `[(ngModel)]="data"`

ng-lisbon.com

Built-in property binding

```
<img [src]="srcVariable">

<button [disabled]="isValid">

<div [ngClass]="myClasses">
```

Built-in event binding

```
<button (click)="onClick()">

<input (keyup)="onKey($event)">

<div (mouseenter)="mouseenter($event)">
```

ng-lisbon.com

Two-way binding

```
<input type="number" name="number1" [(ngModel)]="number1">
<input type="number" name="number2" [(ngModel)]="number2">

{{ number1 + number2 }}

<button (click)="number1 = 10">Nummer 1 wird 10</button>
```

ng-lisbon.com

Life cycle of a component

#	Lifecycle hook	Description
1	ngOnChanges	Before `ngOnInit` and whenever there is a change of a bound property
2	ngOnInit	On component initialization and after `ngOnChanges`
3	ngDoCheck	During Angular change detection run
4	ngAfterContentInit	After content of `ng-content` was inserted
5	ngAfterContentChecked	Aftercontent of #4 was checked
6	ngAfterViewInit	After initialization of the view and child views
7	ngAfterViewChecked	After view and child views were checked
8	ngOnDestroy	Before component gets destroyed