Vježbe 7, Angular contd., 2025./2026.

Data-binding

Podatke možemo slati od komponenti koje su roditelji prema komponentama koje su djeca. To se zove property binding jer mijenjamo property-e djeteta. `AppComponent` komponenta roditelj je od `MessageComponent` komponente jer smo unutar predloška `AppComponent` stvorili instancu komponente `MessageComponent`.

Neka je unutar `AppComponent` enkapsulirano polje:

```
messages_: string[] = [
  "Embrace kindness",
  "Celebrate diversity",
  "Cherish our planet"
];
```

U `src/app/message/message.component.ts` unesimo

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

i unutar `MessageComponent` definirajmo:

```
// message: string = "Cherish our planet";
message = input<string>();
```

ili još specifičnije:

```
message: InputSignal<string> = input<string>("");
```

Primjetimo kako u predlošku te komponente trebamo postaviti

```
<p>{{ message() }}</p>
```

budući da `message` više nije string nego instanca klase `InputSignal<T>` o kojoj ćemo više reći u nadolazećim predavanjima.

Ondje gdje je `MessageComponent` instancirana, dakle ondje gdje je napisano

```
<app-message></app-message>
```

radimo data-binding koji sintaksom nalikuje uobičajenim html atributima neke html oznake. S lijeve strane (`[message]`), obgrljena uglatim zagradama dolazi vrijednost iz dječje komponente kojoj nešto pridružujemo i koja živi u scope-u dječje komponente, a s desne strane (`"messages[0]"`) vrijednost iz scope-a roditeljske komponente koju pridružujemo djetetu:

```
<app-message
  [message]="messages[0]"
>
</app-message>
```

I tako za ostale poruke. Kasnije ćemo vidjeti kako iskoristiti petlju unutar predloška kako bismo automatski prošli kroz sve elemente polja i za svaki element instancirali `MessageComponent`.

Od djeteta ka roditelju, podatke dijelimo tako da odašiljemo događaj i takvo se dijeljenje onda zove event binding. U `MessageComponent` definirajmo

```
// output event

feedback = output<object>();
```

i npr. u konstruktoru, nakon `ms_rand` milisekundi, pošaljimo ka roditelju podatak `obj_for_parent`:

```
constructor() {
  const ms_rand = Math.random() * 2000;
  setTimeout(() => {
    const obj_for_parent = { ms_rand };
    this.feedback.emit(obj_for_parent);
  }, ms_rand);
}
```

U `AppComponent` moramo definirati kako ćemo handle-ati taj event, npr. ondje definirajmo:

```
handle(event: any) {

  console.log(event);
}
```

I na kraju, ondje gdje je komponenta instancirana, navodimo da ćemo na događaj reagirati s `handle`:

```
<app-message (feedback)="handle($event)" [message]="messages[0]"></app-message>
```

Primijetite da se takav događaj ne razlikuje previše od uobičajenih događaja na koje smo navikli u JavaScriptu. Npr. isti handler možemo iskoristiti i za:

```
<h3 class="display-4" (click)="handle($event)">We learn Angular</h3>
```

Za two-way data-binding (što bi u C++-u bilo prosljeđivanje po referenci) koristimo (u komponenti koja je dijete):

```
// two-way data binding (aka pass-by-reference)
n: ModelSignal<number> = model<number>(0);
```

a ondje gdje je instancirana istovremeno koristimo i [] i ():

```
<app-message [(n)]="passedByReference" [message]="messages[2]"></app-message>
```

Petlje i grananja (control flow)

U AppComponent definirajmo

```
isAdmin: boolean = true;
```

Tada u predlošku možemo koristiti @if grananje koje će u ovisnosti o Boole-ovoj varijabli renderirati određene segmente HTML navedene u {}:

```
<!-- if -->
@if(isAdmin) {
  <h5>Admin panel</h5>
  <button>Drop database</button>
} @else {
  <h5>Unauthorized access</h5>
}
```

U AppComponent nadogradimo:

```
messages: {text: string, id: number}[] = [
  { text: "Embrace kindness", id: 0},
  { text: "Celebrate diversity", id: 1},
  { text: "Cherish our planet", id: 2}
];
```

Korištenjem petlje možemo proći kroz sve elemente polja i instancirati za svaki element instancu komponente MessageComponent:

```
@for(message of messages; track message.id; let idx = $index, e = $even) {
  <app-message [message]="message.text"></app-message>
}
```

Nužno je navesti jedinstveni identifikator svakog elementa polja po kojem iteriramo (`track`) jer time Angular interno optimizira renderiranje DOM-a.