

PROGRAMIRANJE INTERNET APLIKACIJA

Angular 2026

Elektrotehnički fakultet, Univerzitet u Beogradu
Školska 2026/2027



Sadržaj

01

Web: frontend i backend

Ko šta radi i kako se povezuju

02

Web frameworkci

Zašto postoje i koji su popularni

03

Angular – glavni koncepti

Komponente, moduli, servisi, rutiranje, signali

04

Instalacija i prvi projekat

Node.js, Angular CLI, struktura projekta

05

Komponente

Kreiranje i ugnežđavanje komponenti

06

Interpolacija i rutiranje

Prikaz podataka iz TS u HTML, navigacija

07

Signali

signal, computed, set, update, effect...

08

Servisi i dependency injection

@Service() i inject()

09

Kontrola toka

@if, @for, @switch

10

One-way binding

Property, atribut, klasa, stil, događaji

01

Frontend i backend

Dve strane svake web aplikacije i način na koji razgovaraju.

Klijent

Server

HTTP

Frontend vs. backend

Frontend

klijentska strana – izvršava se u browseru

- ▶ Sve što korisnik **vidi** i sa čim **interaguje**
- ▶ Prikaz podataka, forme, navigacija, animacije
- ▶ Validacija unosa pre slanja serveru
- ▶ Šalje zahteve backendu i prikazuje odgovore

HTML

CSS

JavaScript

TypeScript

Angular

React

Vue

VS

Backend

serverska strana – izvršava se na serveru

- ▶ **Poslovna logika** i pravila aplikacije
- ▶ Rad sa **bazom podataka** (čitanje, upis)
- ▶ Autentifikacija i autorizacija korisnika
- ▶ Izlaže **API** koji frontend poziva

Node.js

Express

Spring

.NET

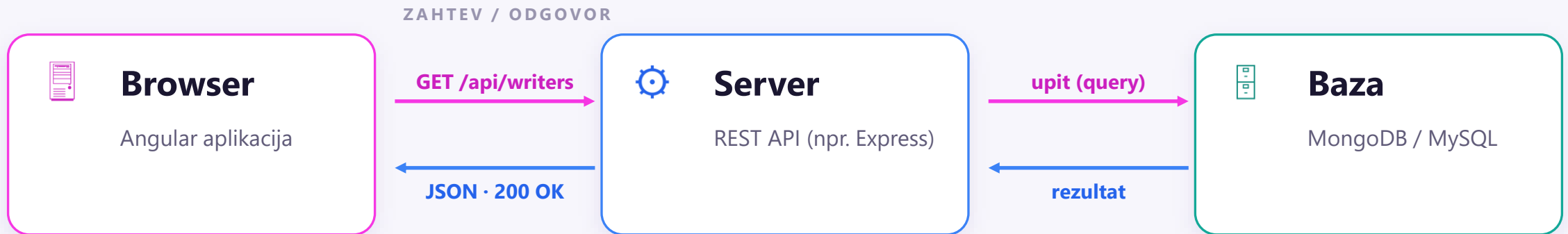
Django

MongoDB

MySQL

Kako se frontend i backend povezuju?

Komunikacija ide preko HTTP protokola – klijent šalje zahtev, server vraća odgovor (najčešće JSON).



02

Web frameworki

Zašto ne pišemo sve „ručno“ u čistom JavaScript-u?

Komponente

Angular

React

Vue

Zašto nam treba framework?

Bez frameworka sve radimo ručno: tražimo elemente, menjamo DOM i pazimo da prikaz prati podatke.

Komponente

Stranica se deli na male, ponovo upotrebljive delove sa svojim HTML-om, CSS-om i logikom.

Automatsko osvežavanje

Promenimo podatak – framework sam osveži prikaz. Nema ručnog rada sa DOM-om.

Rutiranje

Navigacija između „stranica” bez ponovnog učitavanja (SPA).

Struktura projekta

Jasna pravila gde šta stoji – lakše snalaženje u velikim projektima.

Gotovi alati

CLI generiše kod, pokreće server, pravi build za produkciju.

Rad u timu i testiranje

Svi pišu kod na isti način; testovi su deo projekta od početka.

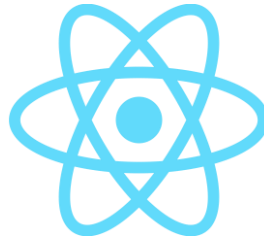
Najpopularniji frontend frameworki



Angular

Google · 2016

- ▶ Kompletan framework
- ▶ TypeScript po defaultu
- ▶ Signali, DI, router, forme, CLI



React

Meta · 2013

- ▶ Biblioteka/framework za UI
- ▶ JSX – HTML unutar JS-a
- ▶ Ogroman ekosistem



Vue

Evan You · 2014

- ▶ Progresivni framework
- ▶ Lak za učenje (HTML/CSS/JS)
- ▶ Single File Components

03

Angular

Šta je Angular i koji su njegovi glavni koncepti.

Komponente

Moduli

Servisi

DI

Rutiranje

Signali

Šta je Angular?

// **Angular je web framework koji omogućava programerima da prave brze i pouzdane aplikacije koje korisnici vole.**

— angular.dev/overview

- ▶ Održava ga **tim u Google-u**, open source na GitHub-u
- ▶ Pisan u **TypeScript-u**
- ▶ Skup alata, API-ja i biblioteka na jednom mestu
- ▶ Skalabilan
- ▶ Nova major verzija ~ na 6 meseci

Glavni koncepti na jednom mestu

</>

Komponente

Gradivni blokovi UI-a: TS klasa + HTML šablon + CSS.

{ }

Šabloni

HTML proširen sintaksom: {{ }}, [], (), [()].

@ |

Direktive i pipes

Direktive menjaju DOM, pipes menjaju prikaz: {{ datum | date }}.

()

Signali

Reaktivne vrednosti – Angular zna šta tačno da osveži.

S

Servisi

Deljena logika i podaci, odvojeni od prikaza.

DI

Dependency injection

Angular sam kreira i „ubrizgava“ zavisnosti: inject().

/

Rutiranje

Mapiranje URL-a na komponente u SPA aplikaciji.

[]

Moduli / standalone

NgModule (stariji način) ili samostalne komponente.

Moduli (NgModule) vs. standalone komponente

Stariji pristup – NgModule

app.module.ts

```
@NgModule({
  declarations: [AppComponent, AboutComponent],
  imports: [BrowserModule, FormsModule],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

- ▶ Modul grupiše komponente, direktive i servise
- ▶ Komponenta mora biti „deklarisana“ u nekom modulu
- ▶ Srećete ga u starijim projektima i bibliotekama

Danas – standalone (podrazumevano od v19)

about.ts

```
@Component({
  selector: 'app-about',
  imports: [AboutText, AboutImage],
  templateUrl: './about.html',
  styleUrls: ['./about.css'],
})
export class About {}
```

- ▶ Svaka komponenta **sama** navodi šta joj treba (**imports**)
- ▶ Nema **app.module.ts** – manje komplikovano
- ▶ Mi u nastavku koristimo **samo standalone**

04

Instalacija i prvi projekat

Node.js, Angular CLI i prolazak kroz prazan projekat.

Node.js

npm

ng new

ng serve

Struktura

Instalacija: Node.js → Angular CLI → projekat

1 Node.js + npm

Sa nodejs.org instalirati **LTS** verziju (na fakultetu **24.21.0**). npm (Node Package Manager) dolazi uz Node (na fakultetu **11.19.0**)

```
$ node -v
v24.21.0
$ npm -v
11.19.0
```

2 Angular CLI

Globalno instaliramo alat **ng** – njim pravimo projekte, komponente, servise...
(na fakultetu **22.2.0**)

```
$ npm install -g @angular/cli
$ ng version
Angular CLI: 22.2.0
```

3 Novi projekat

Pravimo projekat, ulazimo u folder i pokrećemo razvojni server.

```
$ ng new moj-projekat
$ cd moj-projekat
$ npm start
→ http://localhost:4200/
```

ng new – kreiranje projekta korak po korak

```
terminal
$ ng new moj-projekat
? Which stylesheet system would you like to use? > CSS
? Do you want to enable Server-Side Rendering (SSR)
  and Static Site Generation (SSG/Prerendering)? > No
? Which AI tools do you want to configure with
  Angular best practices? > None

CREATE moj-projekat/angular.json
CREATE moj-projekat/package.json
CREATE moj-projekat/src/main.ts
CREATE moj-projekat/src/app/app.ts ...
✓ Packages installed successfully.
  Successfully initialized git.

$ cd moj-projekat
$ npm start
→ Local: http://localhost:4200/
```

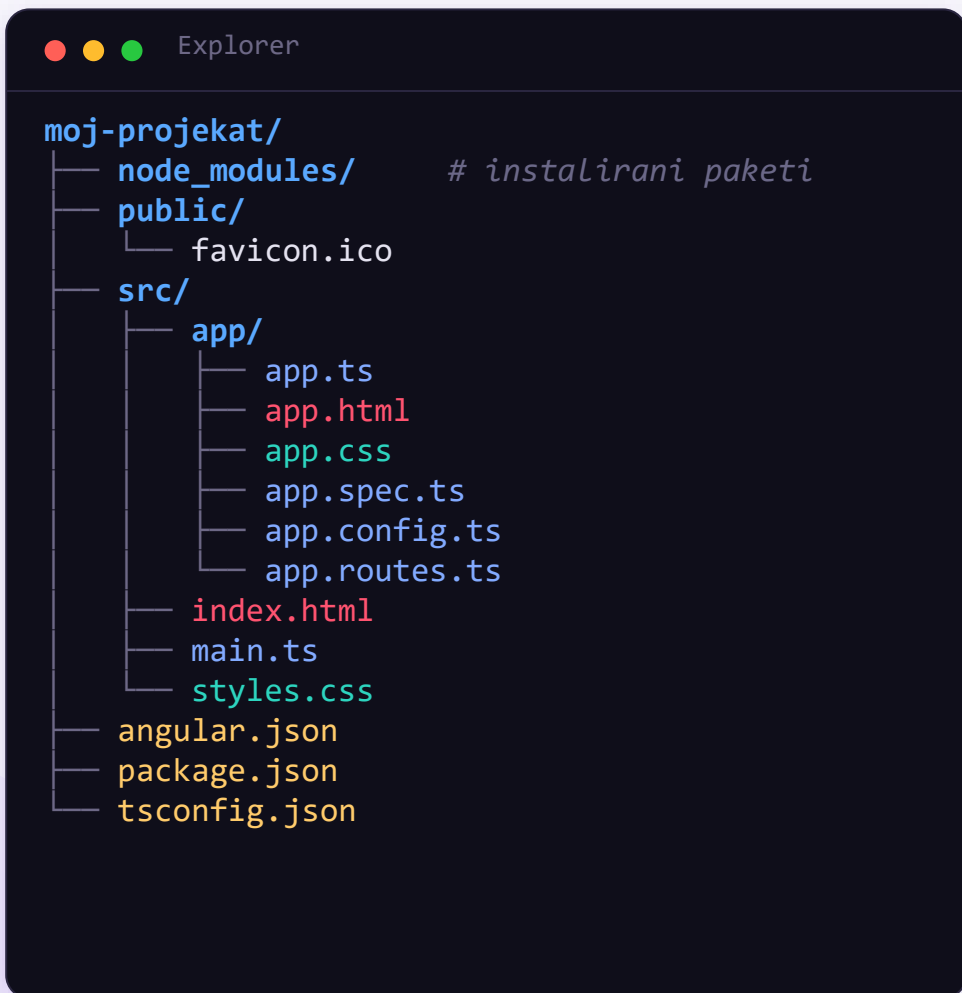
Šta odgovaramo?

- ▶ **Stil:** CSS (može kasnije i SCSS, Sass, Less, Tailwind)
- ▶ **SSR/SSG:** No – pravimo klasičnu **Single Page Application**
- ▶ **AI alati:** None

Korisne opcije

- ▶ `--style=css --ssr=false` – bez pitanja
- ▶ `--skip-tests` – bez .spec.ts fajlova
- ▶ `ng serve --open` – otvori browser
- ▶ `npm start = ng serve`
- ▶ Izmena koda → automatski reload

Struktura novog Angular projekta



Konfiguracija projekta

`package.json` – zavisnosti i skripte (npm start)

`angular.json` – podešavanja build-a i servera

`tsconfig*.json` – podešavanja TypeScript-a

Ulazna tačka

`index.html` – jedina HTML stranica, sadrži `<app-root>`

`main.ts` – pokreće (bootstrap) aplikaciju

`styles.css` – globalni stilovi

Root komponenta App

`app.ts` – logika · `app.html` – šablon · `app.css` – stil

`app.spec.ts` – testovi

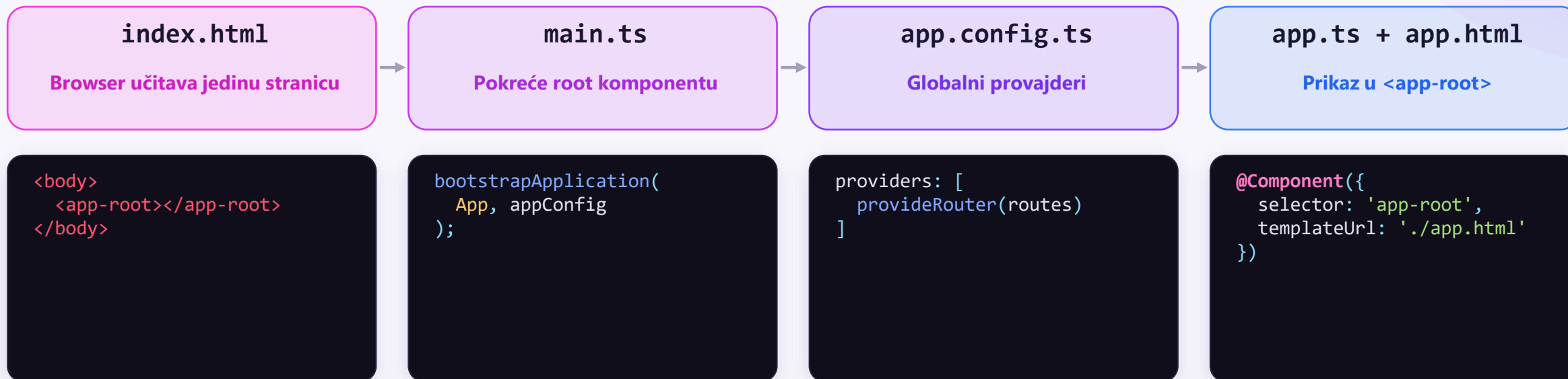
`app.config.ts` – provajderi · `app.routes.ts` – rute

Ostalo

`public/` – statički fajlovi (slike, ikonice)

`node_modules/` – ne menja se ručno, moduli koje koristimo

Kako se aplikacija pokreće?



- ▶ Angular pronađe tag `<app-root>` u `index.html` jer je to **selector** komponente `App`
- ▶ Na njegovo mesto ubaci šablon iz `app.html` – sve ostale stranice se prikazuju **unutar** njega
- ▶ Zato je ovo **SPA**: HTML stranica je jedna, menjaju se samo komponente u njoj

Anatomija komponente

```
src/app/app.ts

import { Component, signal } from '@angular/core';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.html',
  styleUrls: ['./app.css'],
})
export class App {
  protected readonly title = signal('moj-projekat');
}
```

@Component({...})

Dekorator – kaže Angular-u da je klasa komponenta

selector

Ime HTML taga: <app-root> se koristi u drugim šablonima

imports

Druge komponente / direktive koje šablon koristi

templateUrl

Putanja do HTML šablona (ili inline: template)

styleUrl

CSS koji važi samo za ovu komponentu

export class App

Podaci i logika – polja i metode klase

05

Komponente

Kako napraviti novu komponentu i ubaciti je u drugu.

`ng generate`

`selector`

`imports`

Kreiranje nove komponente

terminal

```
$ ng generate component about
# ili skraćeno:
$ ng g c about
CREATE src/app/about/about.css
CREATE src/app/about/about.html
CREATE src/app/about/about.spec.ts
CREATE src/app/about/about.ts
```

src/app/about/about.ts

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

@Component({
  imports: [],
  selector: 'app-about',
  styleUrls: ['./about.css'],
  templateUrl: './about.html',
})
export class About {}
```

Šta je napravljeno?

- ▶ Folder **about/** sa 4 fajla: **TS, HTML, CSS, test**
- ▶ Klasa **About** (bez sufiksa „Component” – novije)
- ▶ Selector **app-about** – prefiks **app** iz angular.json
- ▶ Šablon: `<p>about works!</p>`

Korisne varijante

- ▶ `ng g c pages/about` – u podfolderu
- ▶ `ng g c about --skip-tests` – bez testa
- ▶ `ng g c about -t -s` – inline šablon i stil

Ubacivanje jedne komponente u drugu

Primer: komponente AboutText i AboutImage (deca) prikazuju se unutar komponente About (roditelj).

1 Deca imaju selector

about-text.ts · about-image.ts

```
@Component({
  selector: 'app-about-text',
  templateUrl: './about-text.html',
})
export class AboutText {}

@Component({
  selector: 'app-about-image',
  templateUrl: './about-image.html',
})
export class AboutImage {}
```

2 Roditelj ih uvozi

about/about.ts

```
import { AboutText }
  from '../about-text/about-text';
import { AboutImage }
  from '../about-image/about-image';

@Component({
  imports: [AboutText, AboutImage],
  selector: 'app-about',
  templateUrl: './about.html',
})
export class About {}
```

3 Koriste se kao tagovi

about/about.html

```
<app-about-text></app-about-text>
<app-about-image></app-about-image>
```

Zaboravili ste korak 2? NG8001: 'app-about-text' is not a known element – proverite da li je komponenta navedena u `@Component.imports`.

06

Prikaz podataka i rutiranje

Od polja u TypeScript klasi do teksta na ekranu – i navigacija između stranica.

`{{ interpolacija }}`

`Routes`

`routerLink`

Interpolacija {{ }} – obična polja klase

about-text.ts

```
export class AboutText {  
  description = 'Ovo je stranica o nama';  
  year = 2026;  
  author = { firstname: 'Petar', lastname: 'Petrović' };  
  skills = ['Angular', 'TypeScript', 'Node.js'];  
  
  greet() {  
    return `Zdravo, ${this.author.firstname}!`;  
  }  
}
```

about-text.html

```
<h2>{{ description }}</h2>  
<p>Godina: {{ year }}, sledeća: {{ year + 1 }}</p>  
<p>Autor: {{ author.firstname }} {{ author.lastname }}</p>  
<p>Prva veština: {{ skills[0] }}</p>  
<p>{{ greet() }}</p>
```



localhost:4200

Ovo je stranica o nama

Godina: 2026, sledeća: 2027

Autor: Petar Petrović

Prva veština: Angular

Zdravo, Petar!

Rutiranje – definisanje ruta

U SPA aplikaciji URL određuje koja komponenta se prikazuje – bez ponovnog učitavanja stranice.

src/app/app.routes.ts

```
import { Routes } from '@angular/router';
import { About } from '../about/about/about';
import { Writers } from '../writers/writers';

export const routes: Routes = [
  { path: '', component: About },
  { path: 'writers', component: Writers },
  { path: '**', redirectTo: '' }, // default ruta
];
```

src/app/app.config.ts

```
providers: [provideBrowserGlobalErrorListeners(),
  provideRouter(routes)]
```

URL

Prikazuje

/

About

/writers

Writers

/bilo-sta

→ preusmerenje na / (About)

Default ruta – kad URL ne postoji

- ▶ ****** (wildcard) hvata **svaki** URL koji nije definisan
- ▶ Ide **na kraj** niza – rute se proveravaju redom
- ▶ Može i stranica greške: { path: '**', component: **NotFound** }

Navigacija: routerLink i <router-outlet>

src/app/app.html

```
<a routerLink="">About</a> /  
<a routerLink="writers">Writers</a>  
  
<router-outlet />
```

src/app/app.ts

```
import { Component } from '@angular/core';  
import { RouterOutlet, RouterLink } from '@angular/router';  
  
@Component({  
  imports: [RouterOutlet, RouterLink],  
  selector: 'app-root',  
  templateUrl: './app.html',  
})  
export class App {}
```

localhost:4200/writers

About / Writers

<router-outlet>
ovde se prikazuje Writers

- ▶ **routerLink** umesto **href** – **href** **ponovo učitava** celu aplikaciju!
- ▶ **<router-outlet />** – mesto gde se prikazuje komponenta rute
- ▶ **RouterLinkWithHref** koji nudi auto-import je stari naziv za **RouterLink** – isto radi

07

Signali

Reaktivne vrednosti: Angular tačno zna kada i šta treba da osveži.

`signal()`

`set()`

`update()`

`computed()`

`effect()`

Šta je signal?

Signal je **promenljiva koja javlja Angular-u kada se promeni** – i Angular tada sam osveži prikaz.

- ▶ Pravimo ga sa `signal(početnaVrednost)`
- ▶ Čitamo ga **pozivom**: `count()`
- ▶ Menjamo ga sa `set()` ili `update()`

1 `count = signal(0)`
pravimo signal

2 `count.set(5)`
menjamo vrednost

3 `{{ count() }}` → 5
prikaz se sam osveži

Writable signal – kreiranje, čitanje i izmena

```
writable.ts

import { signal } from '@angular/core';

const count = signal(0); // tip: WritableSignal<number>

console.log(count()); // 0 ← čitanje = poziv

count.set(3); // postavi novu vrednost
console.log(count()); // 3

count.update(v => v + 1); // nova vrednost iz stare
console.log(count()); // 4
```

Napomena: tip `WritableSignal<number>` TypeScript zaključuje sam iz `signal(0)` – ne mora da se piše. Za prazan niz navodimo tip: `signal<Writer[]>([])`.

`signal(početna)`

Kreira signal sa početnom vrednošću

`count()`

Čita trenutnu vrednost (i prati zavisnost)

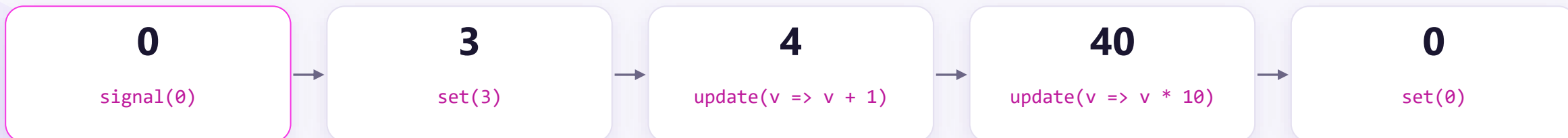
`.set(nova)`

Postavlja potpuno novu vrednost

`.update(fn)`

Računa novu vrednost na osnovu prethodne

VREMENSKA LINIJA VREDNOSTI



Prikaz signala u šablonu – interpolacijom

counter.ts

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

@Component({ selector: 'app-counter', templateUrl:
 './counter.html' })
export class Counter {
  count = signal(0);

  increment() { this.count.update(c => c + 1); }
  reset()      { this.count.set(0); }
}
```

counter.html

```
<p>Brojač: {{ count() }}</p>
<button (click)="increment()">+1</button>
<button (click)="reset()">Reset</button>
```

localhost:4200/counter

Brojač: 3

+1

Reset

← klik = automatski novi prikaz

⚠ Najčešća greška: zaboravljene zagrade

{{ count }} – prikazuje funkciju, ne vrednost

{{ count() }} – ✓ ispravno

Kompajler upozorava: „count is a function and should be invoked: count()”

Computed signal – vrednost izvedena iz drugih signala

person.ts

```
import { signal, computed } from '@angular/core';

firstName = signal('Petar');
lastName = signal('Petrović');

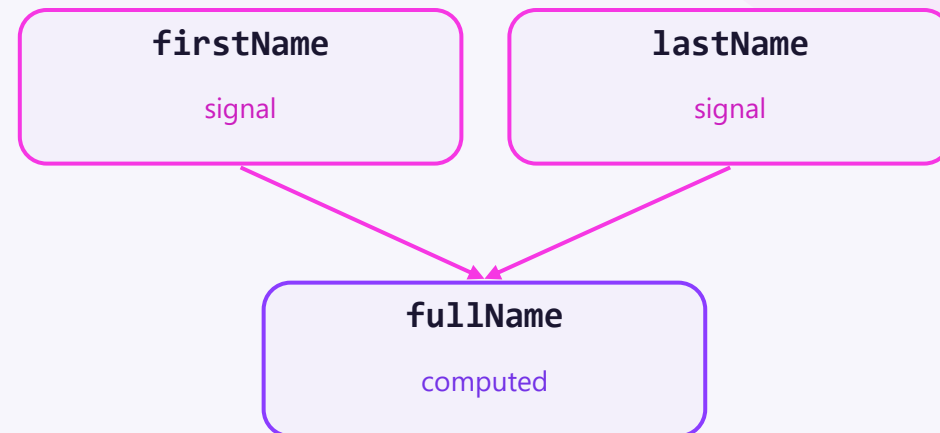
fullName = computed(() => this.firstName() + ' ' + this.lastName());
```

upotreba

```
console.log(this.fullName()); // 'Petar Petrović'

this.firstName.set('Marko');
console.log(this.fullName()); // 'Marko Petrović' ← samo se osvežio

this.fullName.set('Neko'); // X greška: computed je read-only
```

**Read-only**

nema set() ni update()

Lazy

računa se tek kad ga neko pročita

Memoizovan

rezultat se kešira dok se zavisnosti ne promene

Automatske zavisnosti

prati samo signale koje je stvarno pročitao

Primer iz projekta: AboutText komponenta

about/about-text/about-text.ts

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

@Component({
  imports: [],
  selector: 'app-about-text',
  styleUrls: ['./about-text.css'],
  templateUrl: './about-text.html',
})
export class AboutText {
  description = 'This is the page about us'; // obično polje
  authorsFirstName = signal('Petar'); // signal
  authorsLastName = signal('Petrovic');
  authorsFullName = computed(() =>
    this.authorsFirstName() + ' ' + this.authorsLastName());

  ngOnInit(): void {
    this.authorsFirstName.update(value => value + ' Drugo Ime');
  }
}
```

about/about-text/about-text.html

```
<p>{{ description }}</p>
Ovu stranicu je napravio {{ authorsFullName() }}
```



localhost:4200

This is the page about us
Ovu stranicu je napravio Petar Drugo Ime Petrovic

`ngOnInit()` je **lifecycle hook** – Angular ga poziva jednom, odmah nakon kreiranja komponente. `update()` menja ime → `computed` se sam preračuna → prikaz se osveži.

Signali sa nizovima i objektima – nemutabilne izmene

```
writers.ts

writers = signal<Writer[]>([]);

// X LOŠE – menja isti niz, signal ne zna da se nešto desilo
this.writers().push(novi);

// ✓ DOBRO – dodavanje: pravi se NOVI niz
this.writers.update(list => [...list, novi]);

// ✓ brisanje
this.writers.update(list => list.filter(w => w.id !== id));

// ✓ izmena jednog elementa
this.writers.update(list =>
  list.map(w => w.id === id ? { ...w, awards: w.awards + 1 } : w));

// ✓ objekat
user = signal({ name: 'Ana', age: 22 });
this.user.update(u => ({ ...u, age: u.age + 1 }));
```

Zašto?

- ▶ Signal poredi staru i novu vrednost sa `Object.is()`
- ▶ Isti niz (ista referenca) = „**nema promene**“
- ▶ `...` (spread) pravi kopiju

Pravilo

- ▶ Nikada ne menjaj sadržaj signala direktno
- ▶ Uvek `set()` / `update()` sa **novom** vrednošću

Primer: korpa – više computed signala zajedno

cart.ts

```
interface Item { name: string; price: number; qty: number; }

export class Cart {
  items = signal<Item[]>([
    { name: 'Knjiga', price: 1200, qty: 2 },
    { name: 'Olovka', price: 150, qty: 5 },
  ]);

  count = computed(() => this.items().reduce((s, i) => s + i.qty, 0));
  total = computed(() => this.items().reduce((s, i) => s + i.price * i.qty, 0));
  free = computed(() => this.total() >= 3000);

  add(item: Item) { this.items.update(list => [...list, item]); }
}
```

cart.html

```
<p>Stavki: {{ count() }} · Ukupno: {{ total() }} RSD</p>
<p>Besplatna dostava: {{ free() ? 'da' : 'ne' }}</p>
```

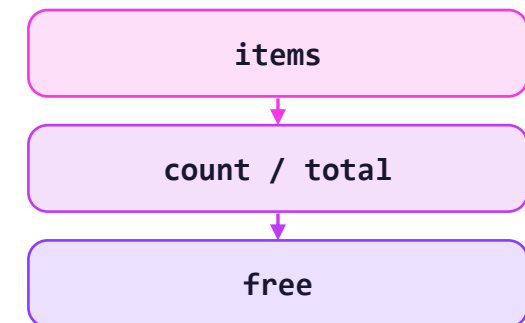
localhost:4200/cart

Stavki: 7

Ukupno: 3150 RSD

Besplatna dostava: da

LANAC ZAVISNOSTI



Sve metode i funkcije signala – podsetnik

signal(v)

Kreira writable signal

```
count = signal(0)
```

()

Čita vrednost i prati zavisnost

```
{{ count() }}
```

.set(v)

Postavlja novu vrednost

```
count.set(5)
```

.update(fn)

Nova vrednost iz prethodne

```
count.update(v => v + 1)
```

.asReadonly()

Read-only pogled na signal

```
c = _c.asReadonly()
```

computed(fn)

Izvedena read-only vrednost

```
d = computed(() => c() * 2)
```

effect(fn)

Sporedni efekat pri promeni

```
effect(() => log(c()))
```

untracked(s)

Čita bez praćenja zavisnosti

```
untracked(counter)
```

linkedSignal(fn)

Writable, resetuje se kad se izvor promeni

```
linkedSignal(() => opts()[0])
```

input()

Ulaz komponente kao signal

```
id = input<string>()
```

isSignal()

Provera da li je vrednost signal

```
isWritableSignal(c) // true
```

{ equal }

Sopstvena provera jednakosti

```
signal(x, { equal: isEqual })
```

effect() – kada treba nešto da se desi „sa strane“

```
counter.ts

import { Component, effect, signal } from '@angular/core';

export class Counter {
  count = signal(Number(localStorage.getItem('count') ?? 0));

  constructor() {
    effect(() => {
      console.log('Brojač je sada', this.count());
      localStorage.setItem('count', String(this.count()));
    });
  }
}
```

Redosled izbora: **signal** za stanje → **computed** za izvedeno → **effect** samo za sporedne efekte.

Kako radi

- ▶ Izvrši se **bar jednom**
- ▶ Ponovo se izvrši kad se promeni neki pročitani signal
- ▶ Pravi se u konstruktoru (injection context)

Kada ga koristiti

- ✓ logovanje, **localStorage**, ručni rad sa DOM-om
- ✓ sinhronizacija sa ne-reaktivnim API-jem
- ✗ **ne** za računanje novog stanja → za to je **computed**

08

Servisi i dependency injection

Gde držimo podatke i logiku koju deli više komponenti?

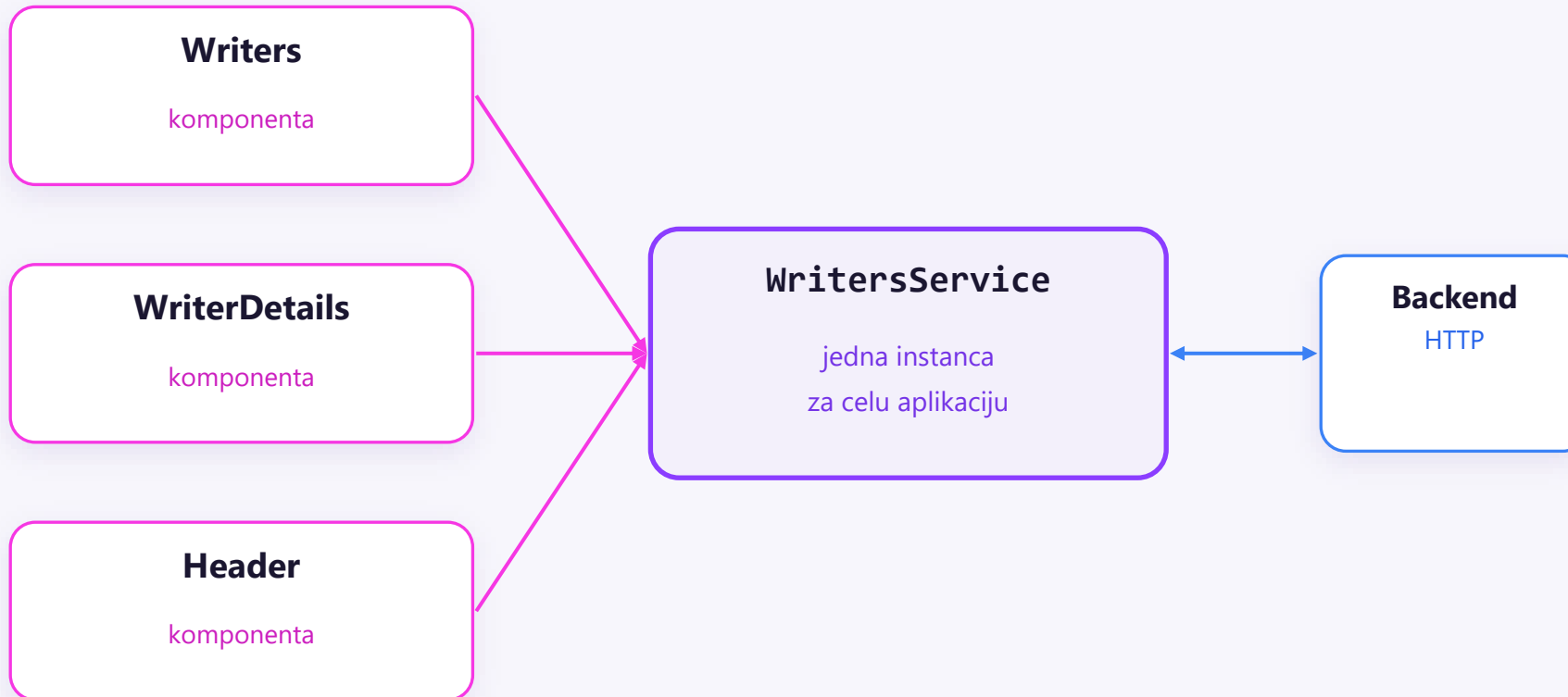
@Service()

inject()

singleton

deljeno stanje

Šta je servis i zašto nam treba?



Servis se koristi za

- ▶ Dohvatanje podataka sa backenda
- ▶ Poslovnu logiku
- ▶ Deljeno stanje između komponenti
- ▶ Logovanje, autentifikaciju...

Po defaultu (root):

- ▶ **Singleton** – jedna instanca
- ▶ Dostupan svuda

Komponenta se bavi **prikazom**, a servis **podacima i logikom**.

Kreiranje servisa

terminal

```
$ ng generate service services/writers-service
$ ng g s services/writers-service
CREATE src/app/services/writers-service.spec.ts
CREATE src/app/services/writers-service.ts
```

src/app/services/writers-service.ts

```
import { Service } from '@angular/core';
import { Writer } from '../models/writers';

@Service()
export class WritersService {
  allWriters: Writer[] = [
    { firstname: 'Pera', lastname: 'Peric', awards: 10 },
    { firstname: 'Zika', lastname: 'Zikic', awards: 2 },
    { firstname: 'Mika', lastname: 'Mikic', awards: 22 },
  ];

  getAllWriters() { return this.allWriters; }
}
```

@Service() – novo u Angular 22

Dekorator kaže Angular-u da je klasa **servis** koji može da se ubaci u komponente.



Dostupan svuda

Servis je automatski registrovan na **root** nivou – koristi se u bilo kojoj komponenti, bez dodatnog podešavanja.

1

Singleton

Angular pravi **samo jednu instancu** za celu aplikaciju – sve komponente dele iste podatke.

Dependency injection – inject()

Komponenta ne pravi servis sa new – traži ga od Angular-a, a Angular joj „ubrizga“ gotovu instancu.

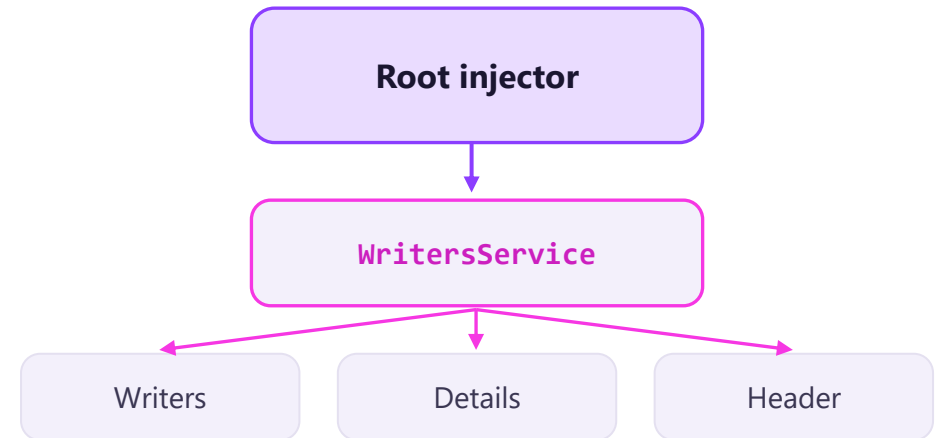
```
src/app/writers/writers.ts

import { Component, inject, OnInit, signal } from '@angular/core';
import { WritersService } from '../services/writers-service';
import { Writer } from '../models/writers';

@Component({
  imports: [],
  selector: 'app-writers',
  styleUrls: ['./writers.css'],
  templateUrl: './writers.html',
})
export class Writers implements OnInit {
  private writersService = inject(WritersService);

  allWriters = signal<Writer[]>([]);

  ngOnInit(): void {
    this.allWriters.set(this.writersService.getAllWriters());
  }
}
```



- ▶ Svi dobijaju **istu instancu** → isti podaci

09

Kontrola toka u šablonu

Uslovni prikaz i liste direktno u HTML-u.

@if

@else

@for

track

@empty

@switch

@if, @else if, @else – uslovni prikaz

osnovno

```
@if (isLoggedIn()) {
  <p>Dobrodošli nazad!</p>
} @else {
  <a routerLink="/login">Prijavi se</a>
}
```

napredno – alias i ugnežđavanje

```
@if (loading()) {
  <p>Učitavanje...</p>
} @else if (user(); as u) {
  <h3>Zdravo, {{ u.firstname }}!</h3>
  @if (u.awards > 10) { <span>🏆 Laureat</span> }
} @else {
  <p>Korisnik nije pronađen.</p>
}
```

Važno

- ▶ Element se **dodaje / uklanja iz DOM-a** (ne samo sakriva)
- ▶ Uslov je bilo koji izraz – najčešće signal **x()**
- ▶ **as u** – sačuva rezultat izraza u lokalnu promenljivu
- ▶ Ugrađeno u šablone – **nije potreban import**

Stara sintaksa

*ngIf="x"

*ngFor="let a of list"

[ngSwitch]

Nova (od v17)

@if (x) { }

@for (a of list; track a.id)

@switch (x) { @case }

Stare direktive su zastarele (deprecated) od Angular 20.

@for – prikaz liste

```
writers.html

<ul>
  @for (writer of allWriters(); track writer.lastname; let i = $index) {
    <li>{{ i + 1 }}. {{ writer.firstname }} {{ writer.lastname }}</li>
  } @empty {
    <li>Nema pisaca.</li>
  }
</ul>
```

track – obavezan

- ▶ Jedinstveni ključ svakog elementa (najčešće **id**)
- ▶ Angular po njemu zna koji red je dodat, pomeren ili obrisan → menja **samo** njega
- ▶ Za liste prostih vrednosti: **track \$index**

Implicitne promenljive

Promenljiva	Značenje
\$index	indeks trenutnog elementa (od 0)
\$count	ukupan broj elemenata
\$first	true za prvi element
\$last	true za poslednji element
\$even	true za parne indekse
\$odd	true za neparne indekse

- ▶ **@empty** – prikazuje se kada je lista prazna
- ▶ **let i = \$index, e = \$even** – više alias-a

@switch i @let

@switch

```
@switch (role()) {  
  @case ('admin') {  
    <app-admin-panel />  
  }  
  @case ('editor') {  
    <app-editor />  
  }  
  @default {  
    <p>Gost – samo čitanje.</p>  
  }  
}
```

- ▶ Poredi vrednost sa svakim **@case** (kao **===**)
- ▶ **@default** kada nijedan slučaj ne odgovara

@let – lokalna promenljiva u šablonu

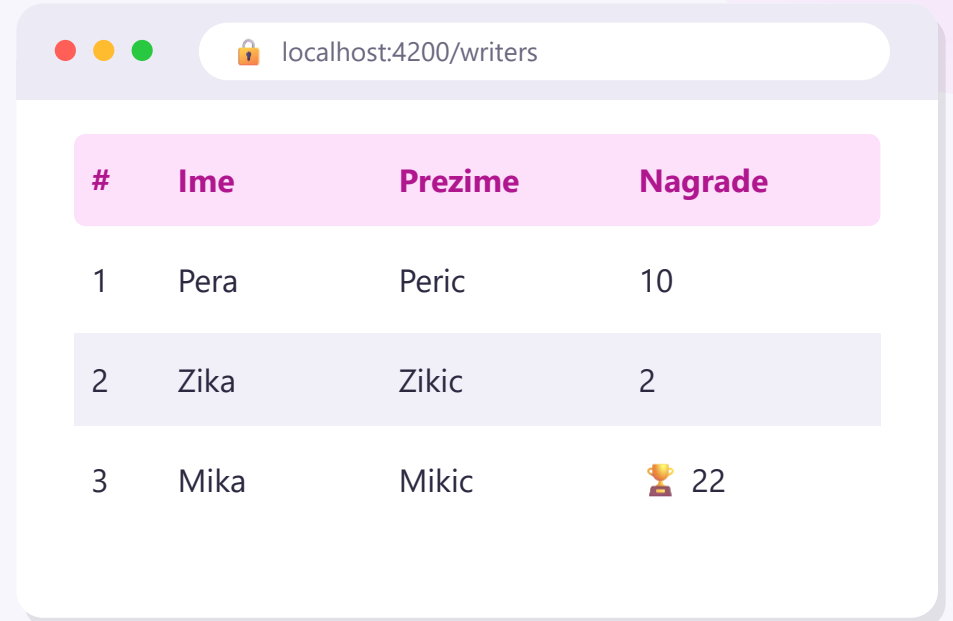
```
@let total = price() * qty();  
@let user = currentUser();  
  
<p>Ukupno: {{ total }} RSD</p>  
@if (total > 5000) {  
  <p>{{ user.firstname }}, dobijate popust!</p>  
}
```

- ▶ Izraz se izračuna jednom i koristi više puta
- ▶ Promenljiva je read-only i važi samo u šablonu
- ▶ Čini šablon čitljivijim

Sve zajedno: tabela pisaca iz servisa

```
writers.html

<table>
  <tr><th>#</th><th>Ime</th><th>Prezime</th><th>Nagrada</th></tr>
  @for (w of allWriters(); track w.lastname; let i = $index) {
    <tr [class.odd]="$odd">
      <td>{{ i + 1 }}</td>
      <td>{{ w.firstname }}</td>
      <td>{{ w.lastname }}</td>
      <td>
        @if (w.awards > 10) { 🏆 }
        {{ w.awards }}
      </td>
    </tr>
  } @empty {
    <tr><td colspan="4">Nema podataka.</td></tr>
  }
</table>
```



#	Ime	Prezime	Nagrada
1	Pera	Peric	10
2	Zika	Zikic	2
3	Mika	Mikic	🏆 22

- ▶ Podaci stižu iz servisa preko signala
- ▶ @for + @if + [class.odd]!

10

One-way binding

Povezivanje TS klase i šablona – podaci teku u jednom smeru.

[property]

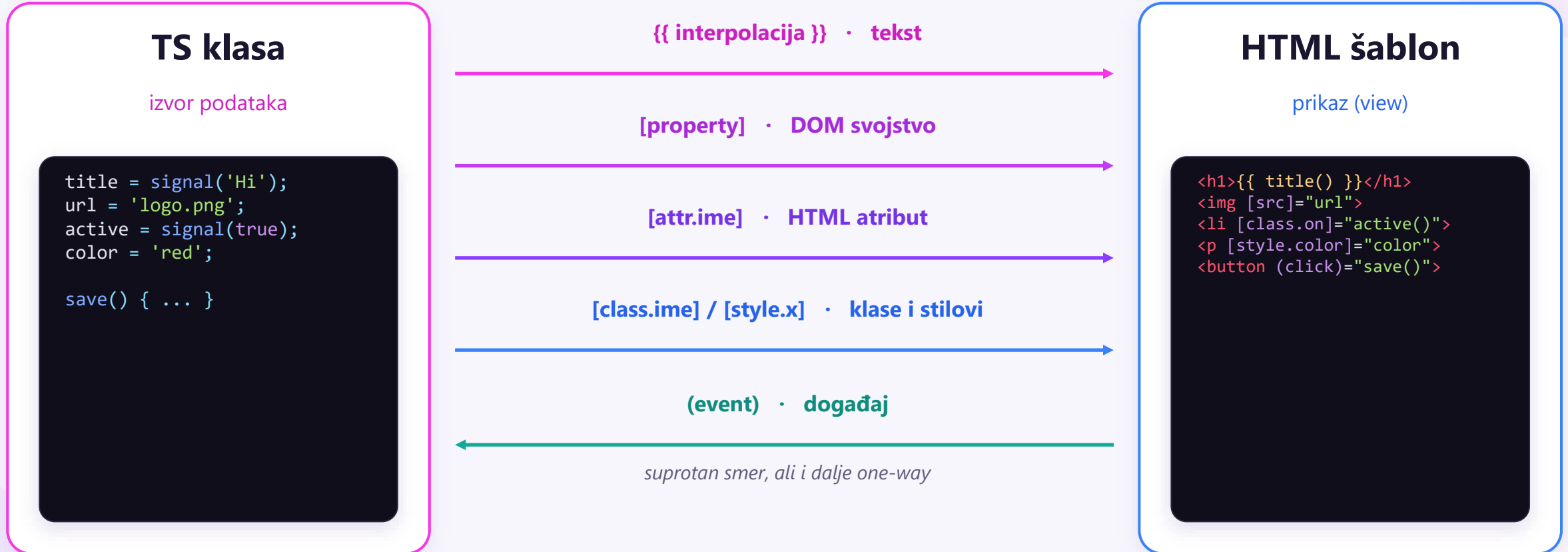
[attr.x]

[class.x]

[style.x]

(event)

Pregled: sve vrste one-way binding-a



Svi su **one-way**: podatak ide samo u jednom smeru. Two-way binding `[ ( ) ]` – na sledećem času.

Property binding – [svojstvo]="izraz"

profile.ts

```
imageUrl = signal('logo.png');  
isSaving = signal(false);  
username = 'pera';
```

profile.html

```
<img [src]="imageUrl()" [alt]="'Slika: ' + username">  
<button [disabled]="isSaving()">Sačuvaj</button>  
<input [value]="username">
```

Sa zagradama ili bez?

src="imageUrl"

→ string „imageUrl“ doslovno

[src]="imageUrl()"

→ vrednost izraza → „logo.png“

Pravila

- ▶ Postavlja **DOM svojstvo** elementa (JS objekat), ne HTML atribut
- ▶ Desna strana je **TS izraz**
- ▶ Kad se signal promeni → svojstvo se automatski osveži
- ▶ Radi i za ulaze komponenti: `<app-card [writer]="w" />`

Attribute binding – [attr.ime]="izraz"

atributi.html

```
<button [attr.aria-label]="actionName()">
  {{ actionName() }}
</button>

<td [attr.colspan]="columns()">Ukupno</td>

<div [attr.data-id]="writer.id" [attr.role]="role"></div>
```

atributi.ts

```
actionName = signal('Obriši pisca');
columns = signal(4);
role = 'listitem';
```

Atribut vs. svojstvo

- ▶ **Atribut** – piše se u HTML-u, početna vrednost
- ▶ **Svojstvo (property)** – trenutna vrednost DOM objekta u JS-u
- ▶ Neki atributi nemaju odgovarajuće svojstvo → tada **attr.**

Kada koristiti

- ▶ **aria-*** (pristupačnost), **role**
- ▶ **data-*** atributi, **colspan**, SVG atributi
- ▶ Vrednost **null** **uklanja** atribut

Class binding – [class.ime]="uslov"

nav.html

```
<!-- jedna klasa: dodaje se ako je uslov true -->
<li [class.active]="isActive()">Početna</li>

<!-- više klasa preko objekta -->
<div [class]="{ active: isActive(), disabled: isDisabled() }">

<!-- string izraz -->
<div [class]="'card ' + theme()"></div>
```

nav.css

```
.active {
  background: #f637e3;
  color: white;
}
```

nav.ts

```
isActive = signal(true);
isDisabled = signal(false);
theme = signal('dark');
```

localhost:4200

Početna

Pisci

O nama

isActive() === true → klasa „active“

- ▶ Klasa se **dodaje** kad je uslov **true**, **uklanja** kad je **false**
- ▶ Stilove pišemo u CSS-u, a binding samo bira klasu

Style binding – [style.svojstvo]="izraz"

progress.html

```
<p [style.color]="color()">Obojen tekst</p>

<!-- sa jedinicom -->
<div [style.width.px]="width()"></div>
<div class="bar" [style.width.%"="progress()"></div>

<!-- više stilova odjednom -->
<div [style]="{ 'background-color': bg(), padding: '8px' }"></div>
```

progress.ts

```
color = signal('#f637e3');
progress = signal(65);
increase() { this.progress.update(p => Math.min(p + 10, 100)); }
```

localhost:4200

Obojen tekst



progress() = 65 → width: 65%

- ▶ Jedinice: `.px`, `.%`, `.em`, `.rem`...
- ▶ Za dinamičke vrednosti (širina, boja iz podataka)
- ▶ Za statičke stilove → CSS klasa je bolji izbor

Event binding – dugme i (click)

Smer je obrnut: događaj iz HTML-a poziva kod u TS klasi.

events.html

```
<button (click)="increment()">+1</button>
<button (click)="count.set(0)">Reset</button>

<input (input)="onInput($event)" (keyup.enter)="save()">
<p>Uneli ste: {{ name() }}</p>
```

events.ts

```
count = signal(0);
name = signal('');

increment() { this.count.update(c => c + 1); }
onInput(e: Event) { this.name.set((e.target as HTMLInputElement).value); }
save() { console.log('Čuvam', this.name()); }
```

Događaj

Kada se desi

(click)	klik mišem
(input)	svaka promena u polju
(change)	potvrđena promena
(keyup.enter)	pritisnut Enter
(submit)	slanje forme
(mouseenter)	miš uđe u element

- ▶ **\$event** – objekat događaja (npr. **e.target.value**)
- ▶ Desna strana je **naredba** (poziv metode, **set...**)

Sve zajedno: „like” dugme

like.ts

```
likes = signal(0);
max = 10;
liked = computed(() => this.likes() > 0);

like() { this.likes.update(n => Math.min(n + 1, this.max)); }
```

like.html

```
<button (click)="like()"
  [disabled]="likes() >= max"
  [class.liked]="liked()"
  [attr.aria-label]="'Lajkova: ' + likes()">
  ♥ {{ likes() }}
</button>
<div class="bar" [style.width.%"="likes() * 10"></div>
```

**(click)**

event

[disabled]

property

[class.liked]

class

[attr.aria-label]

attribute

{{ likes() }}

interpolacija

[style.width.%]

style

Hvala na pažnji!