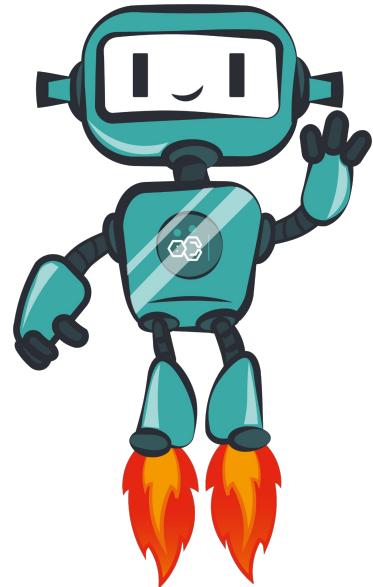
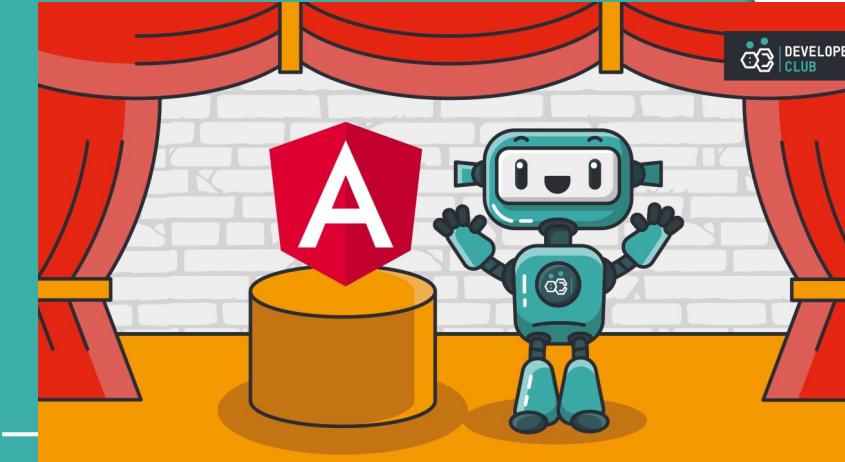


BDD#13 <http://bit.ly/bdd-13>

The Renaissance of Angular: Philosophical Reflections of v17



Cilj predavanja



- Software Craftsmanship prioritet #1
- React postaje Angular, Angular postaje...React?
- Kritički pogled na hvalospev Angularove renesanse
- Diskusija novih stvari
- Benchmarking web aplikacija, JS benchmark projekat

Marcus Vitruvius



- Marcus Vitruvius Pollio (/vɪ'tru:viəs 'poliə/; c. 80–70 BC – after c. 15 BC) - Vitruvius, De architectura (On architecture, published as Ten Books on Architecture)
- Ovo znanje je dete prakse i teorije. Praksa je kontinuirano i redovno vežbanje gde se ručni rad obavlja sa bilo kojim potrebnim materijalom prema dizajnu crteža. Teorija, s druge strane, je sposobnost da se demonstrira i objasni produkcija spremnosti na principima proporcije.

React postaje Angular



- Redovna tema na Podkastu IT Tipa

Podkast IT Tipa sa Sebom i Nikolom



Sve što ste hteli da zнате о softverskom inženjerstvu,
a niste smeli da pitate!

[Najnovije epizode](#)

Slušajte nas na:







NETFLIX

RxJS + Redux + React = Amazing

Side Effect Management with RxJS

```
const autoCompleteEpic = (action$, store) =>
  action$.ofType('QUERY')
    .debounceTime(500)
    .switchMap(action =>
      ajax(`https://api.github.com/search/users?q=${value}`)
        .map(payload => ({
          type: 'QUERY_FULFILLED',
          payload
        }))
    );
  );
```

-RxJS fun: <https://thinkrx.io/>

sebastian-veed-io / [sample-demo.js](#)
sample vs throttle

sample-demo.js

```
1 const { rxObserver, palette } = require('api/v0.3');
2 const { merge, timer, from } = require('rxjs');
3 const { map, zip, throttle, throttleTime, takeUntil, sample } = rxObserver;
4
5 // stream for coloring
6 const palette$ = from(palette);
7
8 // generate a colorized marble stream
9 const source$ = merge(timer(0, 50)).pipe(
10   zip(palette$, Marbles),
11   map(setCurrentTime),
12   takeUntil(timer(1000))
13 );
14
15 source$.subscribe(rxObserver('User triggering actions'));
16
17 source$.pipe(
18   throttle(100, undefined),
19   map(setCurrentTime)
20 ).subscribe(rxObserver('autosave with throttle'));
21
22 source$.pipe(
23   throttleTime(100),
24   map(setCurrentTime)
25 ).subscribe(rxObserver('autosave with throttleTime (100)'));
26
27 source$.pipe(
28   throttleTime(100, undefined, { leading: true, trailing: true }),
29   map(setCurrentTime)
30 ).subscribe(rxObserver('autosave with throttleTime (100) -- leading: true + trailing: true'));
31
32 source$.pipe(
33   throttleTime(100, undefined, { leading: false, trailing: true }),
34   map(setCurrentTime)
35 ).subscribe(rxObserver('autosave with throttleTime (100) -- leading: false + trailing: true'));
36
37 source$.pipe(
38   throttleTime(100, undefined, { leading: false, trailing: false }),
39   map(setCurrentTime)
40 ).subscribe(rxObserver('autosave with throttleTime (100) -- leading: false + trailing: false'));
41
42 source$.pipe(
43   sampleTime(100),
44   map(setCurrentTime)
45 ).subscribe(rxObserver('autosave with sampleTime(100)'));
```

Execution time is limited to 1000ms

The diagram shows a timeline from 0ms to 1000ms. It displays five sets of colored circles representing events. The first set, labeled 'User triggering actions', has events at 0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, and 950 ms. The subsequent four sets show how these events are processed by different RxJS operators:

- autosave with throttle:** Events are grouped into 10-second windows. The first window contains events at 0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, and 950 ms.
- autosave with throttleTime (100) default:** Events are grouped into 100ms windows. The first window contains events at 0, 100, 200, 250, 400, 550, 700, and 850 ms.
- autosave with throttleTime (100) -- leading: true + trailing: true:** Events are grouped into 100ms windows, including the start of the next window. The first window contains events at 0, 100, 200, 250, 350, 400, 500, 550, 650, 700, 800, 850, and 950 ms.
- autosave with throttleTime (100) -- leading: false + trailing: true:** Events are grouped into 100ms windows, excluding the start of the previous window. The first window contains events at 100, 200, 350, 400, 500, 650, 700, 800, and 950 ms.
- autosave with sampleTime(100):** Events are sampled every 100ms. The first window contains events at 100, 200, 300, 400, 500, 600, 700, 800, and 900 ms.

-RxJS marble diagram tests

Given a hot source, test multiple subscribers that subscribe at different times:

```
testScheduler.run(({ hot, expectObservable }) => {
  const source = hot('--a---a---a---a---a---');
  const sub1 = '      --^-----!';
  const sub2 = '      -----^-----!';
  const expect1 = '  --a---a---a---';
  const expect2 = '  -----a---a---a-';

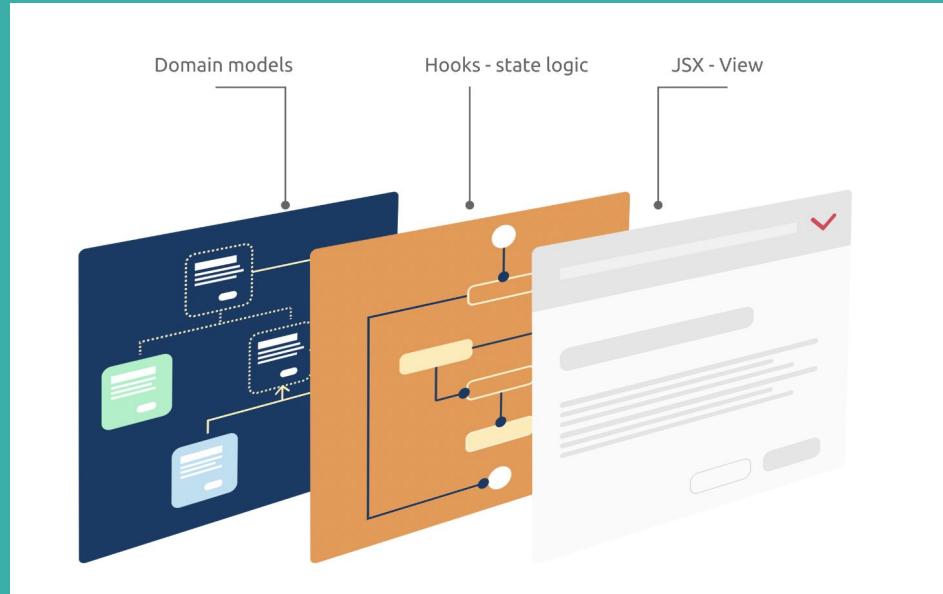
  expectObservable(source, sub1).toBe(expect1);
  expectObservable(source, sub2).toBe(expect2);
});
```

Martin Fowler blog



- Headless components

<https://martinfowler.com/articles/headless-component.html>



Angular postaje React



```
@Component({
  selector: "counter",
  standalone: true,
  template: `
    <div>
      <p>The count is: {{ count() }}</p>
      <p>The doubled count is: {{ computedDouble() }}</p>
      <button (click)="increment()">+</button>
      @for (item of items; track item) {
        <span>{{ item }}</span>
      }
    </div>
  `,
})
export class CounterComponent {
  items = ['what', 'is', 'this', 'life'] as const;
  count = signal(0);
  computedDouble = computed(() => this.count() * 2);

  increment() {
    this.count.update((value) => value + 1);
  }
}
```

```
const Counter = () => {
  const [count, setCount] = useState(0);
  const [computedDouble, setComputedDouble] = useState(0);

  const increment = () => {
    setCount(count + 1);
  };

  useEffect(() => {
    setComputedDouble(count * 2);
  }, [count]);

  return (
    <div>
      <p>The count is: {count}</p>
      <p>The doubled count is: {computedDouble}</p>
      <button onClick={increment}>+</button>
      {items.map((item, index) => (
        <span key={index}>{item}</span>
      ))}
    </div>
  );
};
```

Angular postaje React



→ <https://www.youtube.com/@JoshuaMorony>

The screenshot shows the homepage of Joshua Morony's YouTube channel. At the top, there is a banner with the text "Advanced training for **Angular** developers who want to create **NEXT LEVEL** native web applications." Below the banner, there is a row of icons representing various technologies: Ionic, Angular, Nest.js, Nx, Capacitor, React Native, React, and Gatsby. The channel's profile picture is a yellow circle containing a portrait of Joshua Morony. His name, "Joshua Morony", is displayed in white text. Below his name, it says "@JoshuaMorony · 64.8K subscribers · 307 videos". A bio snippet reads: "I think the web is cool and like to use it wherever I can. Check out my advanced Ionic tutori... >". A "Subscribed" button with a bell icon is visible. At the bottom, there are navigation links for "Home", "Videos", "Playlists", "Community", and a search bar.

Angular postaje React



```
// sources
private checklistsLoaded$ = this.storageService.loadChecklists();
add$ = new Subject<AddChecklist>();
edit$ = new Subject<EditChecklist>();
remove$ = this.checklistItemService.checklistRemoved$;

// state
private state = signal<ChecklistsState>({
  checklists: [],
  loaded: false,
  error: null,
});

// selectors
checklists = computed(() => this.state().checklists);
loaded = computed(() => this.state().loaded);
error = computed(() => this.state().error);
```

```
constructor() {
  // reducers
  this.checklistsLoaded$.pipe(takeUntilDestroyed()).subscribe({
    next: (checklists) =>
      this.state.update((state) => ({
        ...state,
        checklists,
        loaded: true,
      })),
    error: (err) => this.state.update((state) => ({ ...state, error: err })),
  });

  this.add$.pipe(takeUntilDestroyed()).subscribe((checklist) =>
    this.state.update((state) => ({
      ...state,
      checklists: [...state.checklists, this.addIdToChecklist(checklist)],
    }))
  );
}
```

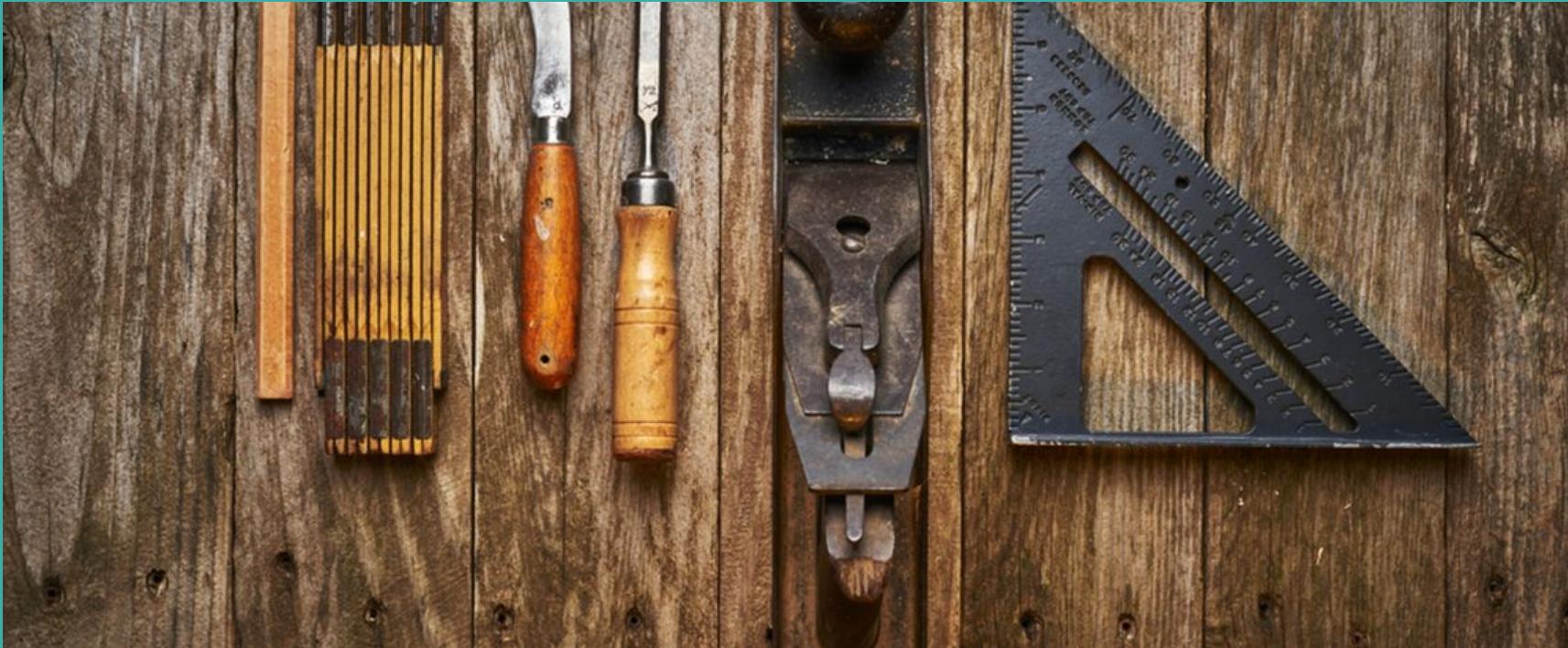
Angular postaje React



DEVELOPER'S
CLUB

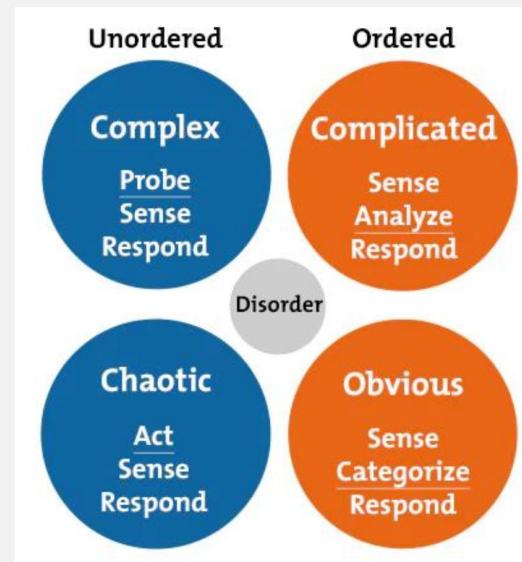
```
state = signalSlice({
  initialState: this.initialState,
  sources: [this.sources$],
  actionSources: {
    add: (state, action$: Observable<AddChecklist>) =>
      action$.pipe(
        map((checklist) => ({
          checklists: [
            ...state().checklists,
            this.addIdToChecklist(checklist),
          ],
        }))
      ),
    },
  );
});
```

Angular postaje React



FRED BROOKS: NO SILVER BULLET, 1986

- **Essential complexity**
 - complexity of **domain**
 - cannot be avoided
 - cannot be changed
- **Accidental complexity**
 - complexity of **implementation**
 - problems engineers create and fix



Special Angular Event



DEVELOPER'S
CLUB

https://www.youtube.com/watch?v=Wq6GpTZ7AX0&ab_channel=Angular

- DevEx / DX
- Performance

November 6
10am Pacific

goo.gle/angular-event

Say hello *`{again}`*
to Angular

Special Angular Event



→ On page playground

The screenshot shows a developer's environment for an Angular application. On the left, a code editor displays `app/app.component.ts` with the following code:

```
3
4  @Component({
5    selector: 'app-root',
6    template: `
7      Welcome to Angular!
8      <p>The count is: {{ count() }}</p>
9      <p> double is {{ doubleCount() }} </p>
10     <button (click)="increment()"></button>
11
12     <ng-container *ngFor="let item of items; track by item">
13       {{ item }}
14     </ng-container>
15   `,
16   standalone: true,
17 })
18 export class AppComponent {
19   items = ['Hello', 'World'] as const;
```

Below the code editor are three tabs: Preview, Console, and Terminal. The Preview tab shows "Welcome to Angular!". The Console tab shows "The count is: 5" and "double is 10". The Terminal tab shows "+ Hello World".

The main content area of the playground shows the "Learn Angular" introduction page. It features sections for "Welcome to the Angular tutorial", "How to use this tutorial", and "Ready to explore more of Angular?". Each section has a "Reveal answer" button at the top.

At the bottom of the playground, there is a footer with the text "hen playground zaista radi xd".

Special Angular Event



DEVELOPER'S
CLUB

Angular Playground

main.ts

game.html

game.css

+

Cela Angular aplikacija



DEVELOPER'S
CLUB

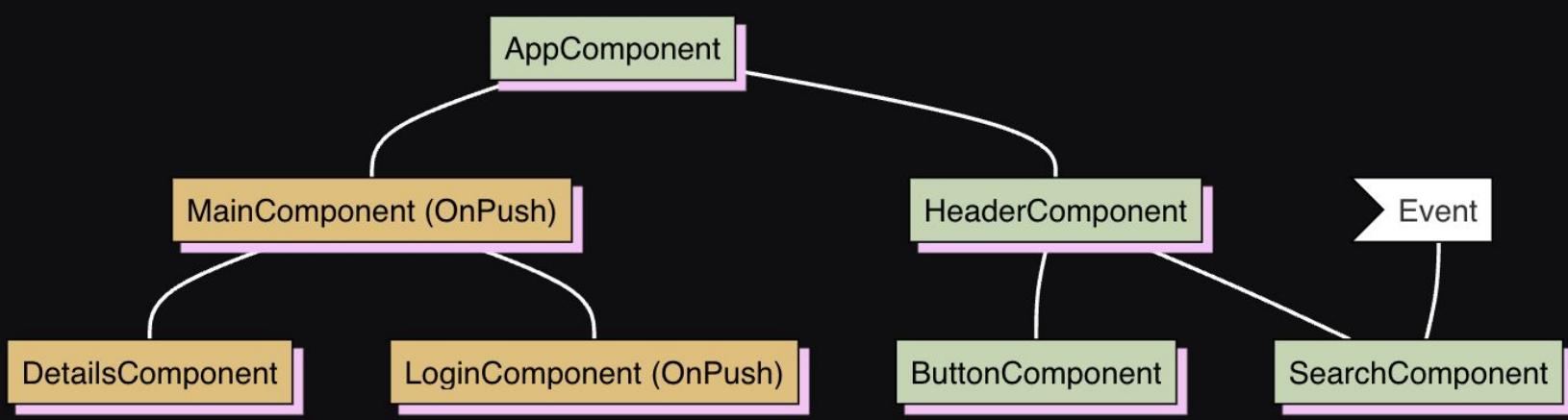
Main.ts →

```
@Component({
  selector: "counter",
  standalone: true,
  template: `
    <div>
      <p>The count is: {{ count() }}</p>
      <button (click)="increment()">+</button>
      @for (item of items; let i = $index; track item) {
        | <span attr.data-seb="{{i}}">{{ item }}
      }
    </div>
  `,
})
export class CounterComponent {
  items = ['what', 'is', 'this', 'life'] as const;
  count = signal(0);

  increment() {
    this.count.update((value) => value + 1);
  }
}

bootstrapApplication(CounterComponent);
```

Local change detection

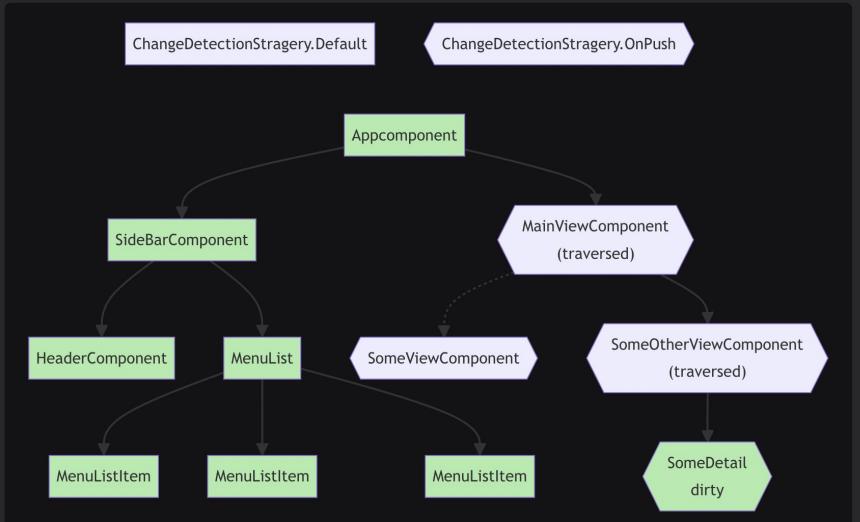


- <https://angular.dev/best-practices/skipping-subtrees>
- Local Change detection almost done

Local change detection



As of 17.0, this local change detection is only available when using `Signals . markAncestorsForTraversal` is a private API.



- <https://jeanmeche.github.io/angular-change-detection/>
- `markAncestorsForTraversal()` `markViewDirty()`

Trigger global actions

selfTimeout: 1 Click

Control dirty check coloring

Clear dirty check coloring automatically

iPlay with component input - current value: n/a

trigger change merge input object (rel-change) mutate input object (no rel-change) via Observable (on rel)

Add

Have you
Black Fr

You can
browse th
combine R
VPA

Enjoy con
Through D

Get

Default

inputChanges attached

Input value:
object prop:
observable:
local signal: 0

Click

inputChanges attached
Input value:
object prop:
observable:
local signal: 0

Click

OnPush

Default

inputChanges attached

Input value:
object prop:
observable:
local signal: 0

Click

OnPush

inputChanges attached

Input value:
object prop:
observable:
local signal: 0

Click

Default

inputChanges attached

Input value:
object prop:
observable:
local signal: 0

Click

OnPush

inputChanges ignored

Input value:
object prop:
observable:
local signal: 0

Click

Default

inputChanges attached

Input value:
object prop:
observable:
local signal: 0

Click

OnPush

inputChanges attached

Input value:
object prop:
observable:
local signal: 0

Click

Default

inputChanges attached

Input value:
object prop:
observable:
local signal: 0

Click

OnPush

inputChanges attached

Input value:
object prop:
observable:
local signal: 0

Click

Default

inputChanges attached

Input value:
object prop:
observable:
local signal: 0

Click

OnPush

inputChanges attached

Input value:
object prop:
observable:
local signal: 0

Click

Default

inputChanges attached

Input value:
object prop:
observable:
local signal: 0

Click

OnPush

inputChanges attached

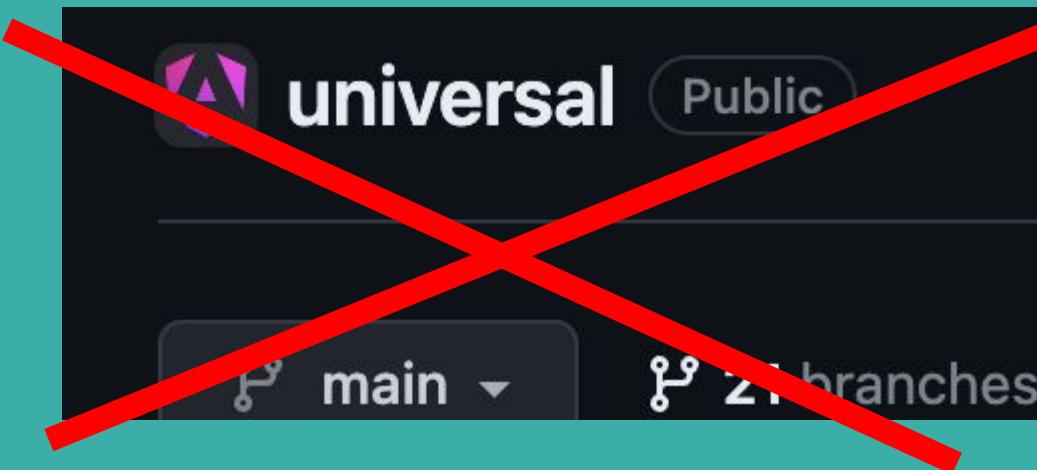
Input value:
object prop:
observable:
local signal: 0

Click

SSR je production ready

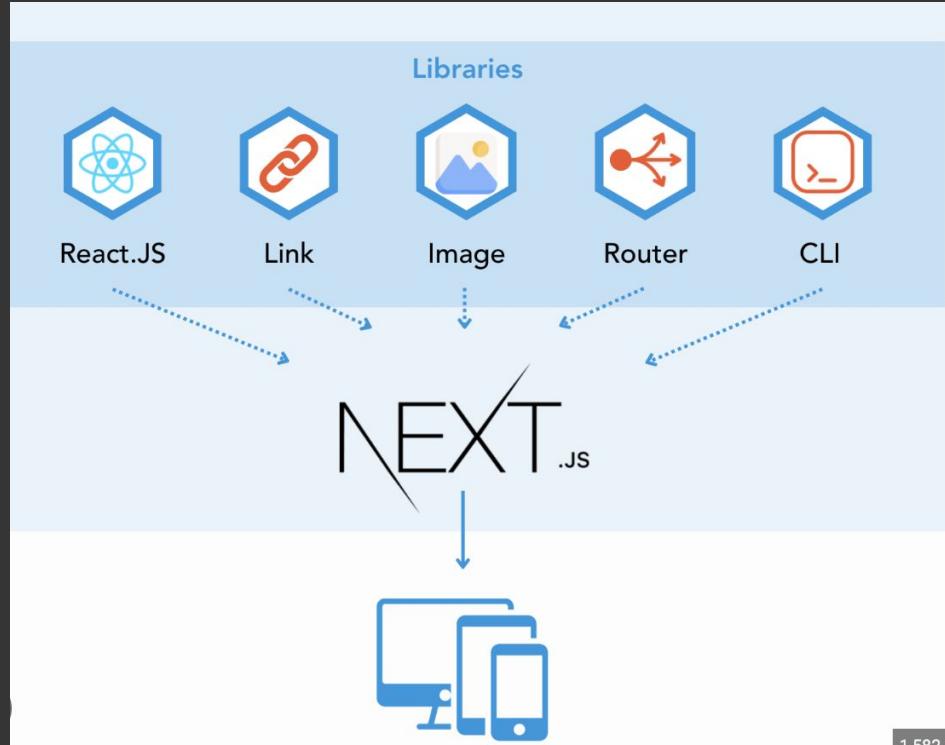


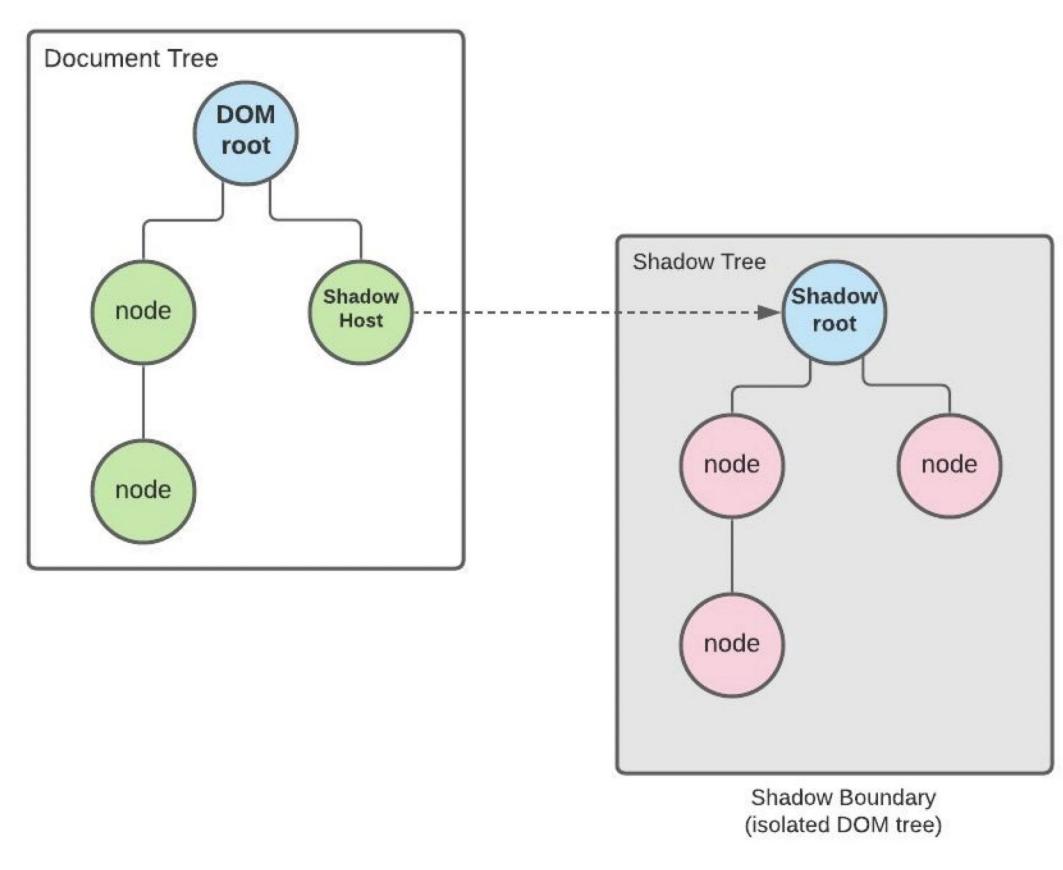
- ng new city-app --ssr
- pre-rendering + hydration



```
import {  
  bootstrapApplication,  
  provideClientHydration,  
} from '@angular/platform-browser';  
...  
  
bootstrapApplication(RootCmp, {  
  providers: [provideClientHydration()]  
});
```

<https://web.dev/articles/rendering-on-the-web>





```
styleUrl: ['my-style.css'],
templateUrl: ['my-template.html']
```

```
styleUrl: 'my-style.css',
templateUrl: 'my-template.html'
```

Design tokens - soon



What's a design token?

Design tokens represent the small, repeated design decisions that make up a design system's visual style. Tokens replace static values, such as hex codes for color, with self-explanatory names.

A Material Design token consists of 2 parts:

- 1 A code-like name, such as `md.ref.palette.secondary90`
- 2 An associated value, such as `#E8DEF8`

The token's value can be one of several things: A color, a typeface, a measurement, or even another token.

 → 

Example of a reference token and its associated color value

Component Authoring



```
@Component({
  selector: "max-pamet",
  standalone: true,
  template: `

    <div *ngIf="pametOK; else pametNijeOk">
      MAX_PAMET
    </div>
    <ng-template #pametNijeOk>
      NEDOVOLJNO_PAMETI
    </ng-template>

    @if (pametOK) {
      MAX_PAMET
    } @else {
      NEDOVOLJNO_PAMETI
    }
  `,
})
```

Component Authoring



The diagram illustrates the structure of the `Counter.cshtml` file. It is divided into three main sections: **Page Routing**, **Page HTML**, and **Code Section**. A large curly brace on the right side groups the **Page HTML** and **Code Section**.

```
1 @page "/counter"
2
3 <h1>Counter</h1>
4
5 <p>Current count: @currentCount</p>
6
7 <button onclick="@IncrementCount">Click me</button>
8
9 @functions {
10     int currentCount = 0;
11
12     void IncrementCount()
13     {
14         currentCount++;
15     }
16 }
17
```

- Page Routing:** Points to the `@page` directive at line 1.
- Page HTML:** Groups the **Page HTML** section (lines 3-7) and the **Code Section**.
- Event Binding:** Points to the `onclick="@IncrementCount"` attribute on the button at line 7.
- Code Section:** Groups the **Code Section** (lines 9-16).

Component Authoring



```
body>
    <c:set var="age" scope="request" value="${21}" />

    <c:if test="${age >= 18}">
        <p>You are an adult.</p>
    </c:if>

    <c:if test="${age < 18}">
        <p>You are not an adult.</p>
    </c:if>
    ...

```

Component Authoring



```
<div [ngSwitch]="userType">
  <pro-dashboard *ngSwitchCase="pro"/>
  <basic-dashboard *ngSwitchCase="basic"/>
  <free-dashboard *ngSwitchDefault/>
</div>

@switch (userType) {
  @case ('pro') { <pro-dashboard/> }
  @case ('basic') { <basic-dashboard/> }
  @default { <free-dashboard/> }
```

Component Authoring



```
<div *ngFor="let user of users; trackBy: trackById">
  <user-card [user]="user" />
</div>
```

```
<div *ngIf="users.length === 0">
  Empty list of users
</div>
```

```
trackById(index: number, user: any): number {
  return user.id;
}
```

Component Authoring



DEVELOPER'S
CLUB

```
@for (user of users; track user.id) {  
    <user-card [user]="user" />  
} @empty {  
    <div>Empty list of users </div>  
}
```

Defer



```
<div #triggerZaLoadMegaKomponente>Ja postojim</div>

<!-- KLIK -->
@defer (when interaction) {
    <moja-mega-komponenta />
}

<!-- trigger druga comp -->
@defer (on viewport(triggerZaLoadMegaKomponente)) {
    <moja-mega-komponenta />
}

@defer (when condition) {
}

@placeholder (minimum 2s) {
    <span>Dok se ne klikne ovo stoji</span>
}

@loading (after 200ms; minimum 700ms) {
    <span>Kliknuto, loading...</span>
}
```

Defer testing



DEVELOPER'S
CLUB

```
const componentFixture = TestBed.createComponent(ComponentA);
// Retrieve the list of all defer block fixtures and get the first block.
const deferBlockFixture = (await componentFixture.getDeferBlocks())[0];

// Render loading state and verify rendered output.
await deferBlockFixture.render(DeferBlockState.Loading);
expect(componentFixture.nativeElement.innerHTML).toContain('Loading');

// Render final state and verify the output.
await deferBlockFixture.render(DeferBlockState.Complete);
expect(componentFixture.nativeElement.innerHTML).toContain('large works!');
```

Defer pre v17



```
class DeferRucnoKomponenta {  
  
    async deferRucno() {  
        const { MojaMegaKomponenta } = await import(`./moja-mega-komponenta.component`);  
        const factory = this.resolver.resolveComponentFactory(MojaMegaKomponenta);  
        this.sadrzajZaNgTemplate = this.vcr.createComponent(factory);  
    }  
}
```

@Input transform

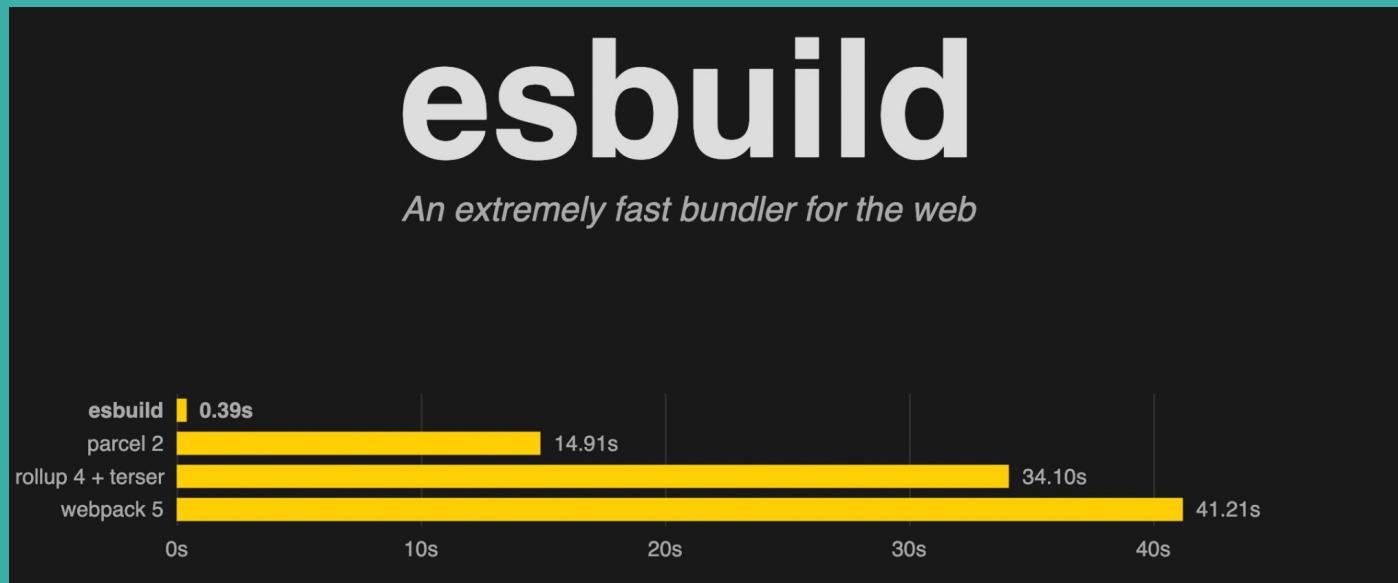


```
    ,  
    ,  
    <typical-expander expanded />  
}  
)  
export class CounterComponent {  
  @Input({ transform: booleanAttribute }) expanded: boolean = false;
```

BUILDER improvements



- Webpack nije deprecated, za sada



Benchmark

- BLAZING FAST?

Duration

[None](#) [All](#)

- create rows
- replace all rows
- partial update
- select row
- swap rows
- remove row
- create many rows
- append rows to large table
- clear rows

Startup

[None](#) [All](#)

- consistently interactive
- total kilobyte weight

Memory

[None](#) [All](#)

- ready memory
- run memory
- update every 10th row for 1k rows (5 cycles)
- creating/clearing 1k rows (5 cycles)
- run memory 10k



DEVELOPER'S
CLUB

Benchmark



```
async init(browser: Browser, page: Page) {
    await checkElementExists(page, "#run");
    for (let i = 0; i < config.WARMUP_COUNT; i++) {
        await clickElement(page, "#run");
        await checkElementContainsText(page, "tbody>tr:nth-of-type(1)>td:nth-of-type(1)", (i * 100));
        await clickElement(page, "#clear");
        await checkElementNotExists(page, "tbody>tr:nth-of-type(1000)>td:nth-of-type(1)");
    }
}
async run(browser: Browser, page: Page) {
    await clickElement(page, "#run");
    await checkElementContainsText(page, "tbody>tr:nth-of-type(1000)>td:nth-of-type(1)", 1000);
}
}());
```

Benchmark



Name Duration for...	angular-cf-signals-v17.0.0-rc.0	react-hooks-v18.2.0
Implementation notes		
Implementation link	code	code
create rows creating 1,000 rows (5 warmup runs).	46.7 ± 0.5 (1.20)	49.6 ± 0.5 (1.27)
replace all rows updating all 1,000 rows (5 warmup runs).	52.9 ± 0.8 (1.31)	52.3 ± 0.5 (1.30)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 4 x CPU slowdown.	22.7 ± 0.7 (1.16)	25.1 ± 0.8 (1.29)
select row highlighting a selected row. (5 warmup runs). 4 x CPU slowdown.	7.5 ± 0.2 (2.44)	6.4 ± 0.2 (2.07)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	25.5 ± 0.6 (1.10)	161.0 ± 1.5 (6.94)

Memory allocation in MBs ± 95%

Name	angular-cf-signals-v17.0.0-rc.0	react-hooks-v18.2.0
ready memory Memory usage after page load.	1.5 (2.85)	1.0 (2.04)
run memory Memory usage after adding 1,000 rows.	5.3 (2.62)	4.7 (2.34)
update every 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	5.4 (2.49)	5.3 (2.43)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	2.2 (3.76)	1.8 (3.08)
run memory 10k Memory usage after adding 10,000 rows.	35.0 (2.47)	34.4 (2.42)
geometric mean of all factors in the table	2.80	2.44

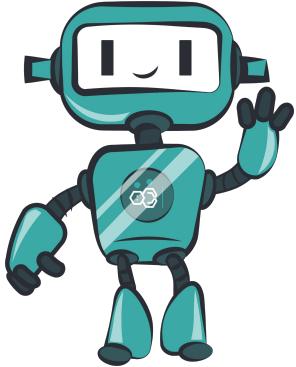
Known issues and notes

- 634 [Issue]: The HTML structure for the implementation is not fully correct.
- 772 [Note]: Implementation uses manual DOM manipulations
- 796 [Note]: Implementation uses explicit requestAnimationFrame calls
- 800 [Note]: View state on the model
- 801 [Note]: Implementation uses manual event delegation
- 1139 [Note]: Implementation uses runtime code generation
- 1261 [Note]: Manual caching of (v)dom nodes

Benchmark



```
, div {
  <table class="table table-hover table-striped test-data">
    <tbody>
      @for (item of data(); track item.id)
      {
        <tr [class.danger]="item.id === selected()">
          <td class="col-md-1">{{item.id}}</td>
          <td class="col-md-4">
            <a href="#" (click)="selected.set(item.id)">{{item.label}}</a>
          </td>
          <td class="col-md-1"><a href="#" (click)="delete(item.id)"><span class="glyphicon glyphicon-remove" aria-hidden="true"></span></a></td>
          <td class="col-md-6"></td>
        </tr>
      }
    </tbody>
  </table>
  <span class="preloadicon glyphicon glyphicon-remove" aria-hidden="true"></span>
```



Hvala!
Postani Član:
developersclub.rs