Angular, you are here

NG Poland 2024

Mark Techson
Angular DevRel
Google

Alex Rickabaugh
Angular Framework
Google

Pawel Kozlowski
Angular Framework
Google

Fuzzy pictures from the future

You trusted us

We move with purpose

Angular is growing.

You are happier 🎉

Beyond the Angular community?

Developer interest is in Angular is up 7%

*State of JavaScript Survey since 2021

Developers want more.

Pawel Kozlowski

NG Poland 2024

Reactivity

reactive framework

signal based reactivity

zoneless

Optional zone.js. Ready for experiments.

Zoneless is possible. In production.

Ready for experiments!

Switching to zoneless is straightforward:

- Remove zone.js imports
- Add the zoneless provider the application bootstrap
- Check and validate your application

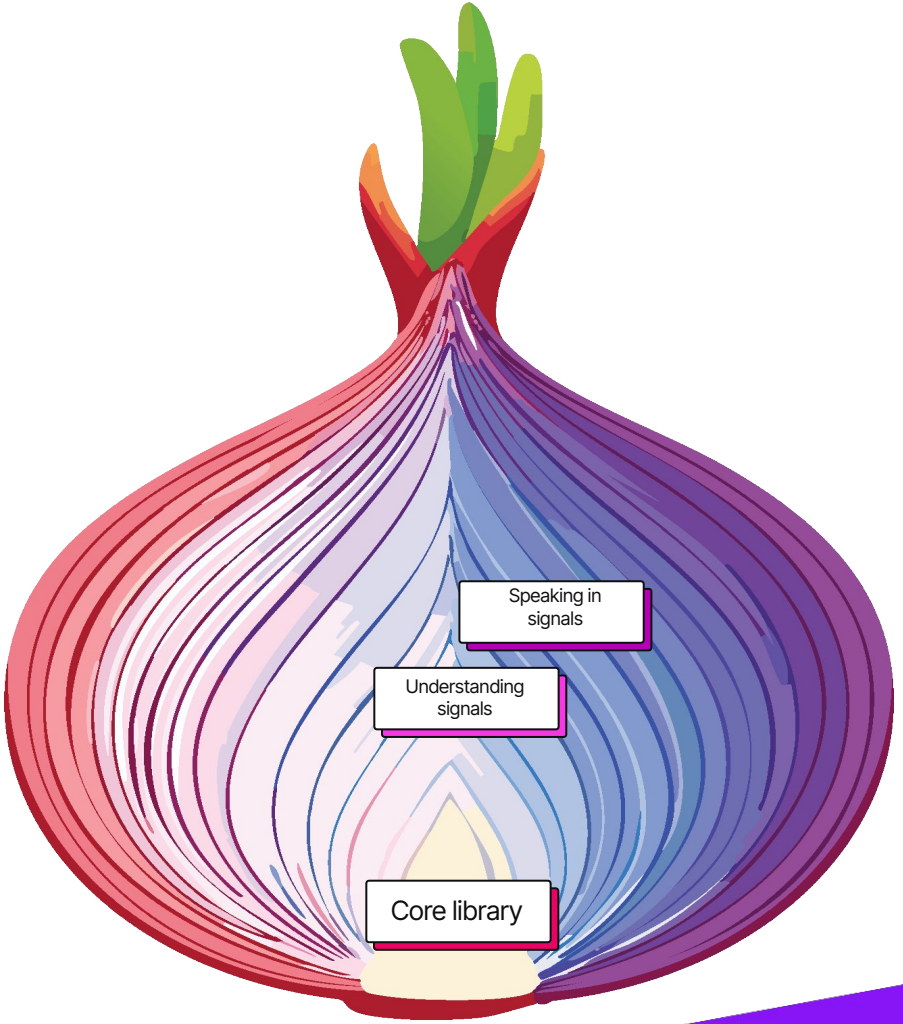
main.ts example.ts code.ts

```
//  import 'zone.js/dist/zone';
import {
 provideExperimentalZonelessChangeDetection
} from '@angular/core';
import { bootstrapApplication } from
'@angular/platform-browser';

bootstrapApplication(App, {
  providers: [
provideExperimentalZonelessChangeDetection()
  ]
});
```

Editor Preview Both

Reactivity. Powered by signals.


Speaking in signals
Understanding signals
Core library

Angular speaks signals!

Reactive inputs, model and queries:

- participate in the reactive graph;
- improved type safety;
- reviewed and simplified API surface;

main.ts example.ts code.ts

```
1  export class UserProfile {
2
3    first = input<string>();
4    last  = input.required<string>();
5
6
7    fullName = computed(() =>
8      this.first() ?
9        `${this.first()} ${this.last()}`:
10       this.last()
11   );
12 }
13
14
15
16
17
18
```

Editor Preview Both

Reactive inputs, model, queries and outputs are stable in Angular v19.

~90%

of inputs, queries and outputs migrated.

Automatically.

```
$ ng generate @angular/core:signals

// or run individual, focused migrations
@angular/core:signal-input-migration
@angular/core:signal-queries-migration
@angular/core:output-migration
```


Experiment
Developer preview
Stable + automated migration

New reactive primitives:

`afterRenderEffect`, `linkedSignal`

afterRenderEffect

Combines effect + afterRender hook:

- effect track reactive dependencies
- runs when it is safe to do changes to the DOM

main.ts example.ts code.ts

```
export class App {
  firstItem = viewChild<ElementRef>('item');

  constructor() {
    afterRenderEffect(() => {
 this.firstItem()?.nativeElement.focus();
    });
  }
}
```

Editor Preview Both

linkedSignal

State that depends on another reactive state:

- It is a writable signal
- Initialized and reset based on the dependencies change.
- Describes a relationship without using effects.

main.ts example.ts code.ts

```
1  export class App {
2
3    options = signal([
4
5      'apple',
6
7      'banana',
8
9      'fig'
10   ]);
11
12
13   choice = linkedSignal(
14
15     () => this.options()[0]
16
17   );
18 }
```

Editor Preview Both

signals meet async data: `resource`

resource

Asynchronous data meet signals:

- Asynchronously loads data in response to dependencies change
- Exposes loaded data as signal
- Loading and error states are signals too
- There is a `rxResource` version

```
@Component({
  template: `
    @if (user.isLoading()) {
      Loading....
    } @else {
      <user-profile [user]="user.value()" />
    }`
})
export class App {
  userId = input<number>();

  user = resource({
    request: () => this.userId(),
    loader: async ({ request: id }) =>
      await this.userService.getUser(id),
  });
}
```

`httpResource` is an obvious candidate!

resource wrapping HTTPClient:

- Reusing existing interceptors
- Testing story is here as well
- Simplified API

main.ts example.ts code.ts

```
1  export class App {
2    userId = input<number>();
3
4    user = httpResource(() =>
5      `/data/user/${this.userIdid()}`
6    );
7
8
9  }
10
11
12
13
14
15
16
17
18
```

Editor Preview Both

`httpResource` is an obvious candidate!

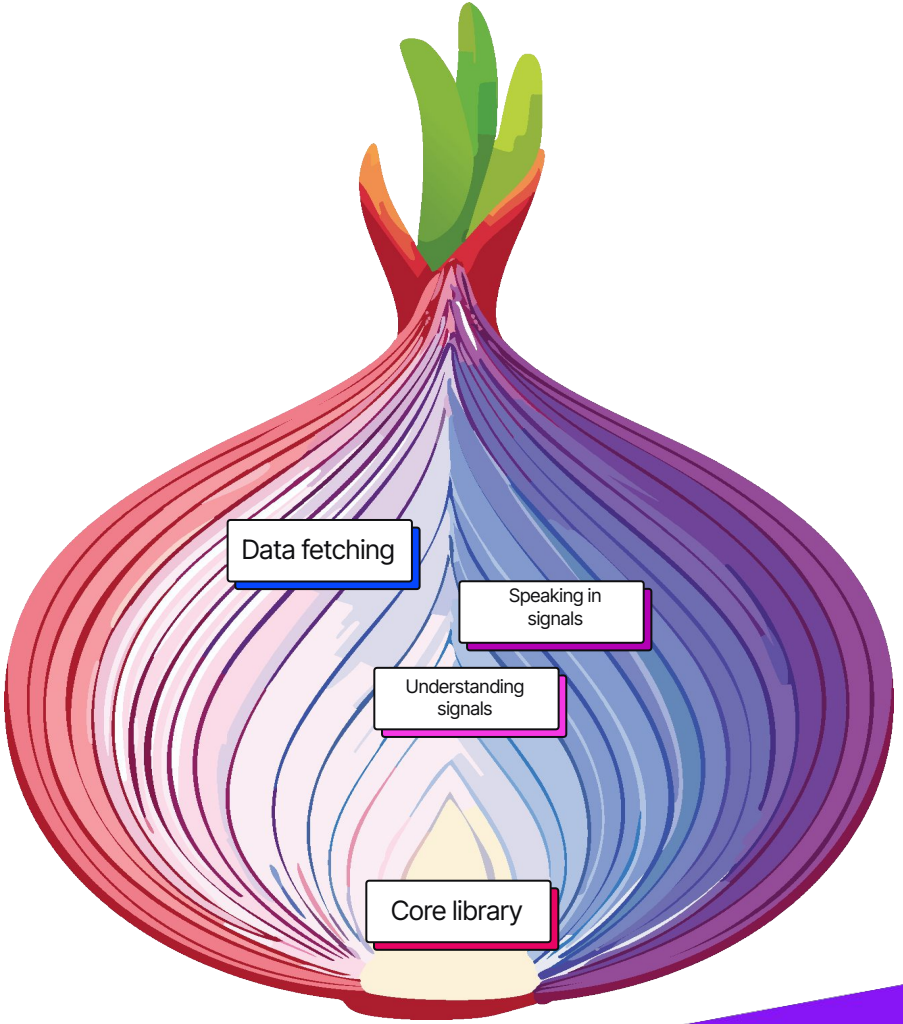
resource wrapping HTTPClient:

- Reusing existing interceptors
- Testing story is here as well
- More advanced API version

main.ts example.ts code.ts

```
1  export class App {
2    userId = input<number>();
3
4    user = httpResource(() => ({
5      method: 'GET',
6      url: `/data/${id()}`,
7      headers: {
8          'X-Page': this.page(),
9      }
10   }));
11
12 }
13
14
15
16
17
18
```

Editor Preview Both


Data fetching
Speaking in signals
Understanding signals
Core library

Expanding signals support in Angular packages: elements and upgrade.

What about forms?

Forms reimagined. Built on signals.

Unified.

Flexible and easier to use

Interoperable.

```
@Component({
 template: `
   <input [field]="form.fields.first">
   <input [field]="form.fields.last">

`})
class UserProfile {
  // Data stored in signals
  data = signal({
    first: 'Pawel',
    last: 'K',
  });

  // Form state derived from data
  form = signalForm(this.data);
}
```

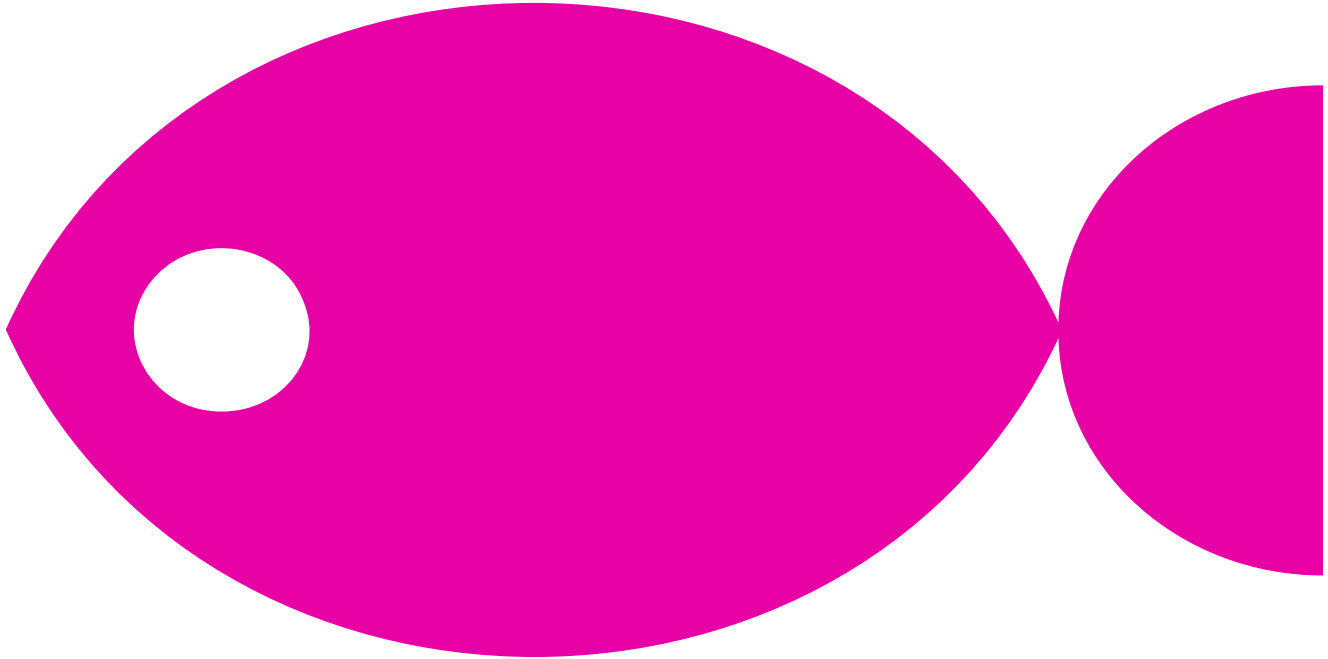
Editor Preview Both

Complex validation with computed signals

Composition of nested forms

Simplified authoring for custom controls

What about router?



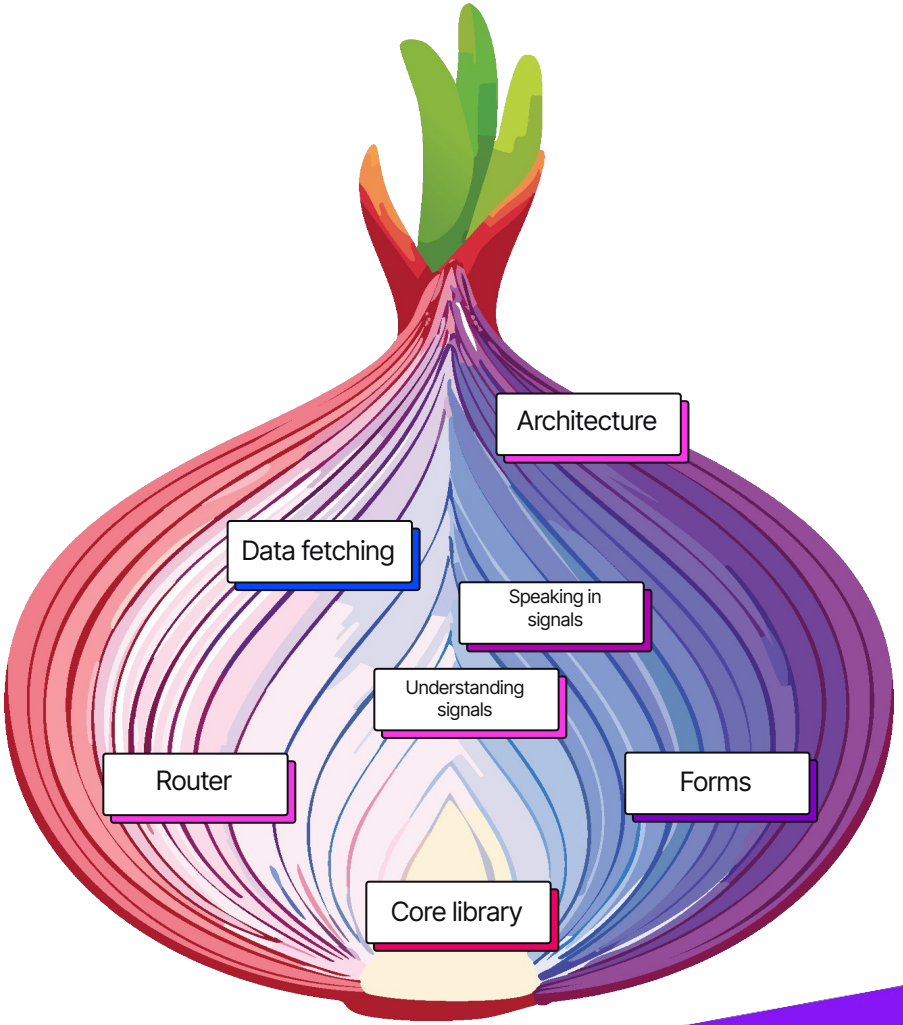
Router coordinates data fetching.

Rendering becomes synchronous.

Resource ties them together.

Great for users

Great for developers


Architecture
Data fetching
Speaking in signals
Understanding signals
Router
Forms
Core library

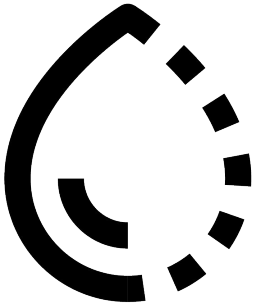
Mark Techson

NG Poland 2024 - @marktechson

Last year...

I was at home.

We were investigating things



Google to merge Angular and Wiz frameworks

News

Apr 01, 2024 • 2 mins

Development Tools | JavaScript | Software Development

▲ Google: Angular and Wiz Are Merging (twitter.com/sarah_edo)

104 points by tosh 7 months ago | hide | past | favorite | 169 comments

r/Angular2 • 8 mo. ago
lab34fr

Angular and Wiz! are merging

Hi,
Regarding the fusion of Angular and Wiz!, what exciting features do you expect it will bring us?

https://twitter.com/sarah_edo/status/1770478763253379488?t=kztJ_cGR3mk3XT2Mxf7CiQ

https://youtube.com/watch?v=nIBseTi6RVk&t=2400

26 | 19 | Share

- Built by Google to power **scalable applications**
- Excellent with **server-side-rendering**
- Optimized for **lazy-loading** code

convergence

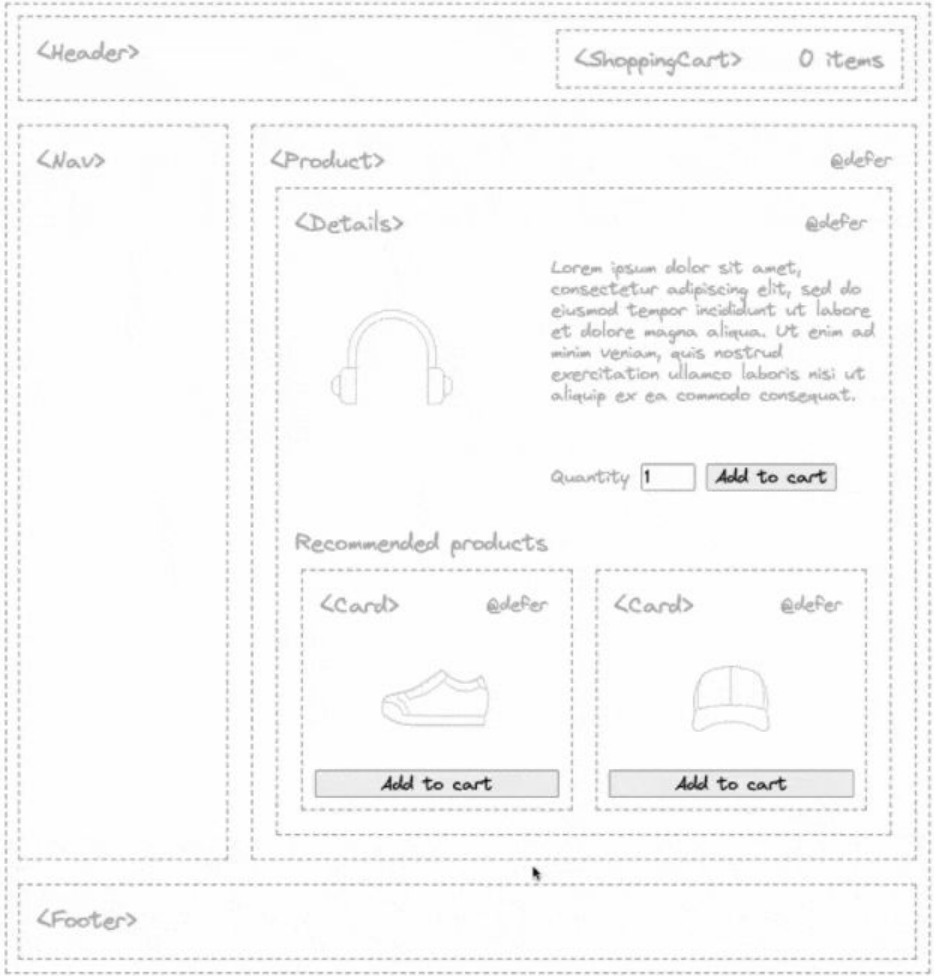
This is a long term effort.

But what about now?

The good stuff.

withEventReplay

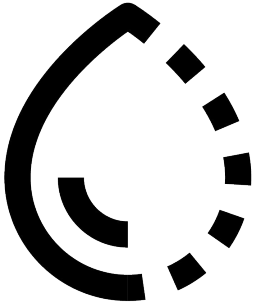
DEMO


<Header>
<ShoppingCart> 0 items
<Nav>
<Product>
@defer
<Details>
@defer
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Quantity 1
Add to cart
Recommended products
<Card>
@defer
Add to cart
<Card>
@defer
Add to cart
<Footer>

```
import {provideClientHydration, withEventReplay}
from '@angular/platform-browser';

bootstrapApplication(App, {
  providers: [
    provideClientHydration(withEventReplay())
  ]
});
```

`withEventReplay` enabled by default



Incremental Hydration

Angular Store

localhost:4200

Angular Store

Cart (0)

Welcome to the Angular Store

An Ecommerce Angular template

Search for a product

Merchandise

View more →

Angular T-shirt

MERCHANDISE

$19.00 ~~$25.00~~ 24% OFF

Angular Sweatshirt

MERCHANDISE

$39.00

Angular Stickers

MERCHANDISE

$5.99

Angular Mug

MERCHANDISE

$9.00

Google I/O T-shirt

MERCHANDISE

$25.00

Technology

View more →

HYDRATION CONTROL PANEL

Hydrated components: 0

Fetched resources: 0 kB

Fetch delay: 0 ms

All requests will be delayed during hydration

Incremental hydration:

`@defer` is the foundation

```
<button #loadBtn>Load Movies</button>

@defer (on interaction(loadBtn)) {
  <recommended-movies />
}
```

We're extending `@defer`

Introducing `hydrate`

```
@defer(on interaction                    ) {
  <recommended-movies />
}
```

Hydration is an initial load optimization

```
@defer(on interaction; hydrate on idle) {
  <recommended-movies />
}
```

`hydrate` triggers

- **Same triggers you know and love:** `Interaction, Immediate, Idle, Viewport, Hover, Timer, When [condition]`
- **One new trigger:** `never`

Wait a minute.

https://slide-from-2023-without-mark

We're investigating... streaming

Incremental hydration is just the beginning.

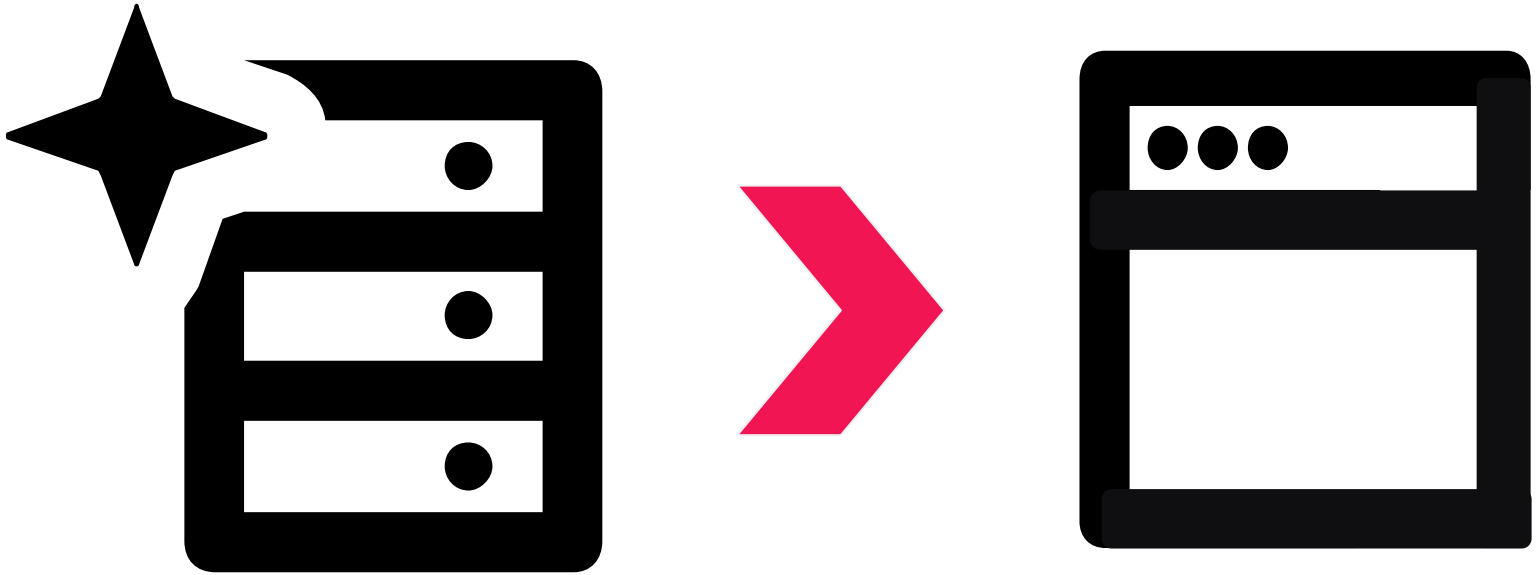
`hydrate` triggers

- **Same triggers you know and love:** `Interaction`, `Immediate`, `Idle`, `Viewport`, `Hover`, `Timer`, `When`
- **One new trigger:** `never`

The future

- Continue our work on streaming responses
- Islands - integration with Astro, for example
- Incremental hydration is the foundation

The 20 year rule.



Alex Rickabaugh

NG Poland 2024

Authoring Format

We incrementally improve Angular, including component authoring

We value your time and energy

1. Unnecessary complexity beyond the platform
2. Multiple approaches to the same problem
3. Special rules that must be remembered

v14: standalone

v17: control flow & @defer

v18.1: @let

What’s next?

Selectorless templates

TypeScript in bindings

Lexical scopes

Selectorless templates

Standalone: directly reference what you use

```
@Component({
  imports: [UserAvatar],
  template: `
    <h3>User profile</h3>
    <div class="avatar">
      <user-avatar />
    </div>
  `,
})
export class UserProfile {}
```

```
@Component({
  imports: [UserAvatar],
  template: `
    <h3>User profile</h3>
    <div class="avatar">
      <user-avatar />
    </div>
  `,
})
export class UserProfile {}
```

```
@Component({
  imports: [UserAvatar],
  template: `
    <h3>User profile</h3>
    <div class="avatar">
      <user-avatar />
    </div>
  `,
})
export class UserProfile {}
```

```
@Component({
  selector: 'user-avatar',
  template: '...',
})
export class UserAvatar {}
```

Selector Matching

Host Element

Imports are doubled

Components have two names

“Selectors” aren’t real CSS

Can't see where directives are in a template

Selectorless: actually reference what you use

```
@Component({
  template: `
    <h3>User profile</h3>
    <div class="avatar">
      <UserAvatar />
    </div>
  `,
})
export class UserProfile {}
```

Direct Reference

```
<input @NgModel( [value]="name" required )>
```

Expressions

```
{{ (user as RegisteredUser).email }}
```

Angular expressions aren’t TypeScript

- Type-cast support (`a as b`) – only `$any(a)`
- Generic support (`fn<T>()`)
- Literal notation (`1e5`)
- Arrow functions
- Postfix operators
- `in` operator
- `instanceof` operator
- Object literal capabilities (spread, etc)
- Optional chaining semantics are different
- Bitwise OR (`a | b`)
- `typeof`

- Type-cast support (`a as b`) – only `$any(a)`
- Generic support (`fn<T>()`)
- Literal notation (`1e5`)
- Arrow functions
- Postfix operators
- `in` operator
- `instanceof` operator
- Object literal capabilities (spread, etc)
- Optional chaining semantics are different
- Bitwise OR (`a | b`)
- ~~`typeof`~~ Thanks, Matthieu!

New TypeScript features aren't supported immediately

Angular expressions should be plain TypeScript expressions #43485

Open

petebacondarwin opened on Sep 17, 2021 edited by JoostK · Edits

Which @angular/* package(s) are relevant/releated to the feature request?

compiler

Description

This issue is a container for all the issues that have requested new expression syntax features.
The aim is to collect all the features into one place so that we can track the work.
The plan is to do an holistic analysis of the syntax and decide things like:

1. How close to ECMAScript syntax we want to stay
2. What are the security implications of exposing particular syntaxes
3. How to incorporate Angular specific syntax, such as pipes and structural directive microsyntax.

The result will be a project proposal for the changes that we would like to make to the syntax in the short to medium term.

What about pipes?

JavaScript is considering a built-in pipeline operator

```
data |> json
```

computed() provides an alternative

Can we reduce the friction of using utility functions directly?

Lexical Scoping for Templates

```
import {UserKind} from ‘./user’;

@Component({template: `
  @if (user.kind === UserKind.Admin) { … }
`})
export class UserProfile {
  UserKind = UserKind; // expose the enum on `this.`
  …
}
```

```typescript
let count = 3;

function addToCount(step: number) {
  return count + step;
}
```

```
<p>Name: {{ name }}
```

```
<p>Name: {{ this.name }}
```

Extending Angular's lexical scoping to include JavaScript

```
import {UserKind} from ‘./user’;

@Component({template: `
  @if (user.kind === UserKind.Admin) { … }
`})
export class UserProfile {
  …
}
```

Selectorless components/directives

Plain TypeScript expressions

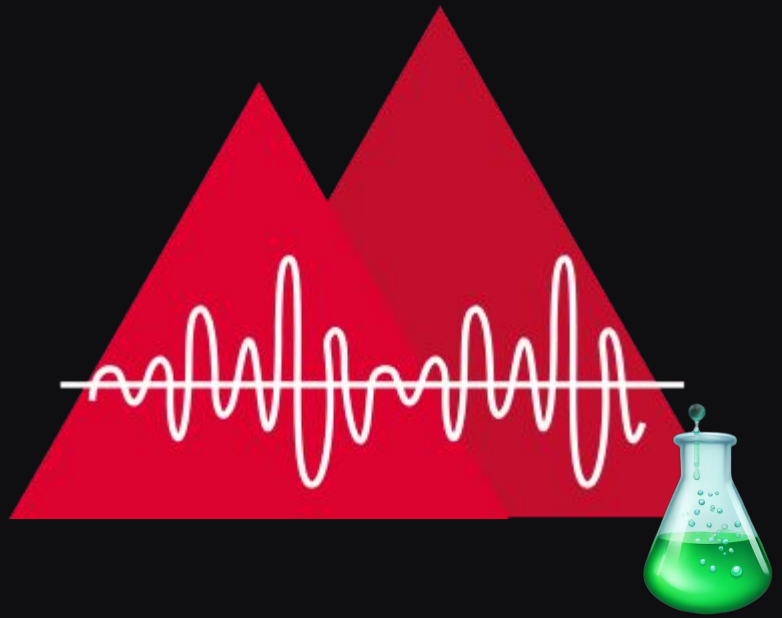
Lexical scoping

Host vs template bindings

Content projection limitations

Inputs not available during constructor

What if we took a more holistic approach?



```
<script lang="ts">
  const isChecked = model(false);

  function toggle() {
    isChecked.update(c => !c);
  }
</script>

<template>
  <input [checked]="isChecked" (click)="toggle()">
</template>
```

You may have seen some of our early brainstorming

```
function MyCheckbox({
  isChecked = model(false),
}) {

  function toggle() {
    isChecked.update(c => !c);
  }

  return ng`
    <input
      checked={isChecked}
      on:click={toggle()} >
  `;
}
```

```
<script>
  const isChecked =
    model(false);

  function toggle() {
    isChecked.update(c => !c);
  }
</script>

<input
  checked={isChecked}
  on:click={toggle()} >
```

```
@Component
class MyCheckbox {
  constructor(
    private isChecked = model(false)
  ) {}

  toggle() {
    this.isChecked.update(c => !c);
  }

  template = ng`
      <input
      checked={this.isChecked}
      on:click={this.toggle()} >`;
}
```

You may have seen some of our early brainstorming

```
function MyCheckbox({
  isChecked = model(false),
}) {

  function toggle() {
    isChecked.update(c => !c);
  }

  return ng`
    <input
      checked={isChecked}
      on:click={toggle()} >
  `;
}
```

```
<script>
  const isChecked =
    model(false);

  function toggle() {
    isChecked.update(c => !c);
  }
</script>

<input
  checked={isChecked}
  on:click={toggle()} >
```

```
@Component
class MyCheckbox {
  constructor(
    private isChecked = model(false)
  ) {}

  toggle() {
    this.isChecked.update(c => !c);
  }

  template = ng`
      <input
      checked={this.isChecked}
      on:click={this.toggle()} >`;
}
```

You may have seen some of our early brainstorming

```
function MyCheckbox({
  isChecked = model(false),
}) {

  function toggle() {
    isChecked.update(c => !c);
  }

  return ng`
    <input
      checked={isChecked}
      on:click={toggle()} >
  `;
}
```

```
<script>
  const isChecked =
    model(false);

  function toggle() {
    isChecked.update(c => !c);
  }
</script>

<input
  checked={isChecked}
  on:click={toggle()} >
```

```
@Component
class MyCheckbox {
  constructor(
    private isChecked = model(false)
  ) {}

  toggle() {
    this.isChecked.update(c => !c);
  }

  template = ng`
      <input
      checked={this.isChecked}
      on:click={this.toggle()} >`;
}
```

You may have seen some of our early brainstorming

```
function MyCheckbox({
  isChecked = model(false),
}) {

  function toggle() {
    isChecked.update(c => !c);
  }

  return ng`
    <input
      checked={isChecked}
      on:click={toggle()} >
  `;
}
```

```
<script>
  const isChecked =
    model(false);

  function toggle() {
    isChecked.update(c => !c);
  }
</script>

<input
  checked={isChecked}
  on:click={toggle()} >
```

```
@Component
class MyCheckbox {
  constructor(
    private isChecked = model(false)
  ) {}

  toggle() {
    this.isChecked.update(c => !c);
  }

  template = ng`
      <input
      checked={this.isChecked}
      on:click={this.toggle()} >`;
}
```

Is it worth it?

Developers ❤️ authoring in Angular

Conclusion

We build Angular together

We move forward together

Deliver web apps with confidence

There's more to this story

There's tooling updates

Tooling Updates

- New migrations available for signals
- New Language service tooling for migrating code in your apps

There's Material updates

Material Updates

- New Components: 2d drag and drop, datepicker
- New theming guides
- New schematics and new APIs

Learn more at the v19 developer event

Thank you for being a part of this journey

Thank you

NG Poland 2024

Mark Techson
Angular DevRel
Google

Alex Rickabaugh
Angular Framework
Google

Pawel Kozlowski
Angular Framework
Google