WARMTEBOUW.

Angular Advanced
Introduction, Architecture

Peter Kassenaar
info@kassenaar.com

Peter Kassenaar

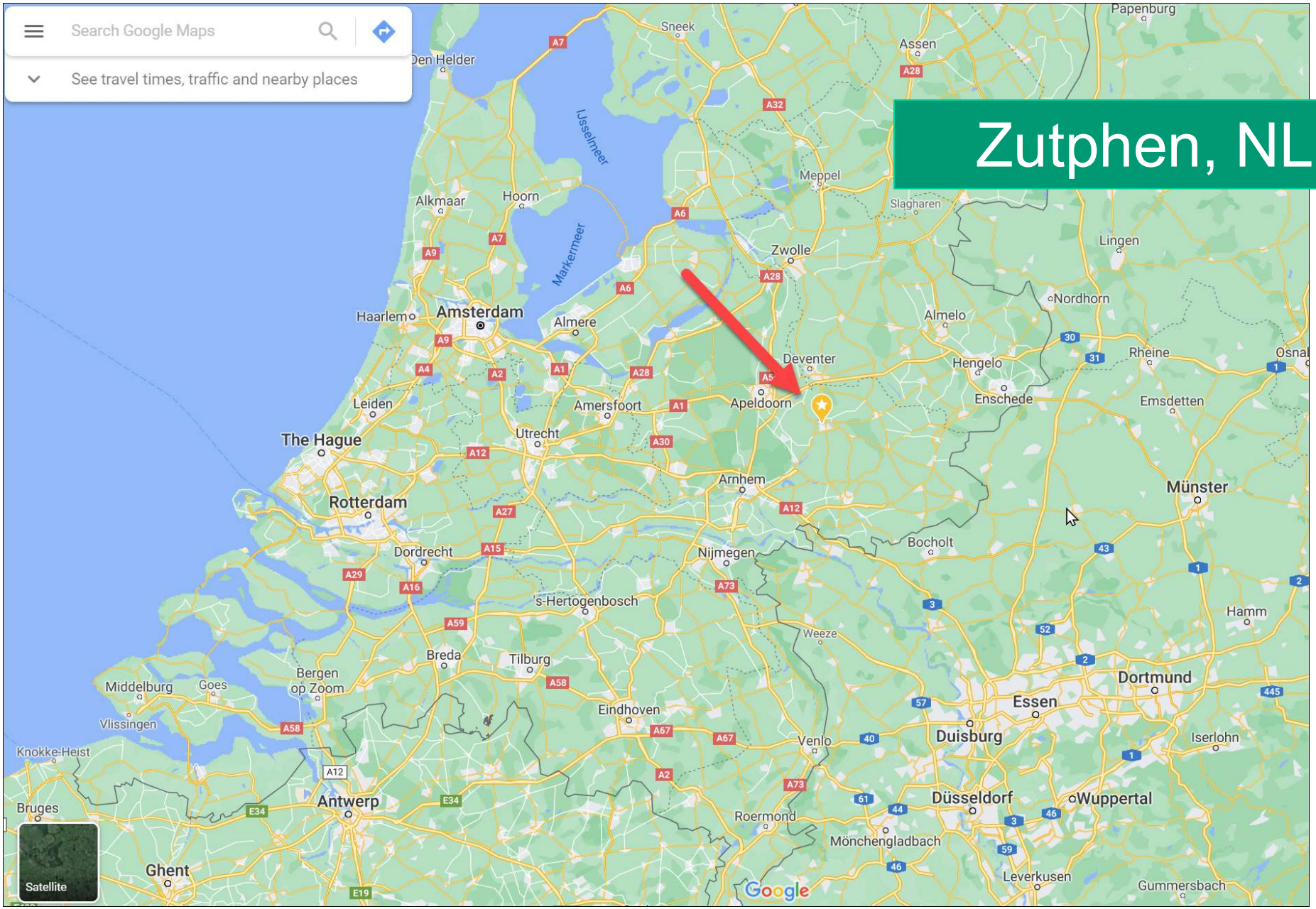
- Trainer, author, developer – since 1996
- Specialty: *"Everything Frontend"*
- JavaScript, ES6, Angular, NodeJS, TypeScript, jQuery, Vue.js, React, Flutter, Ionic

www.kassenaar.com

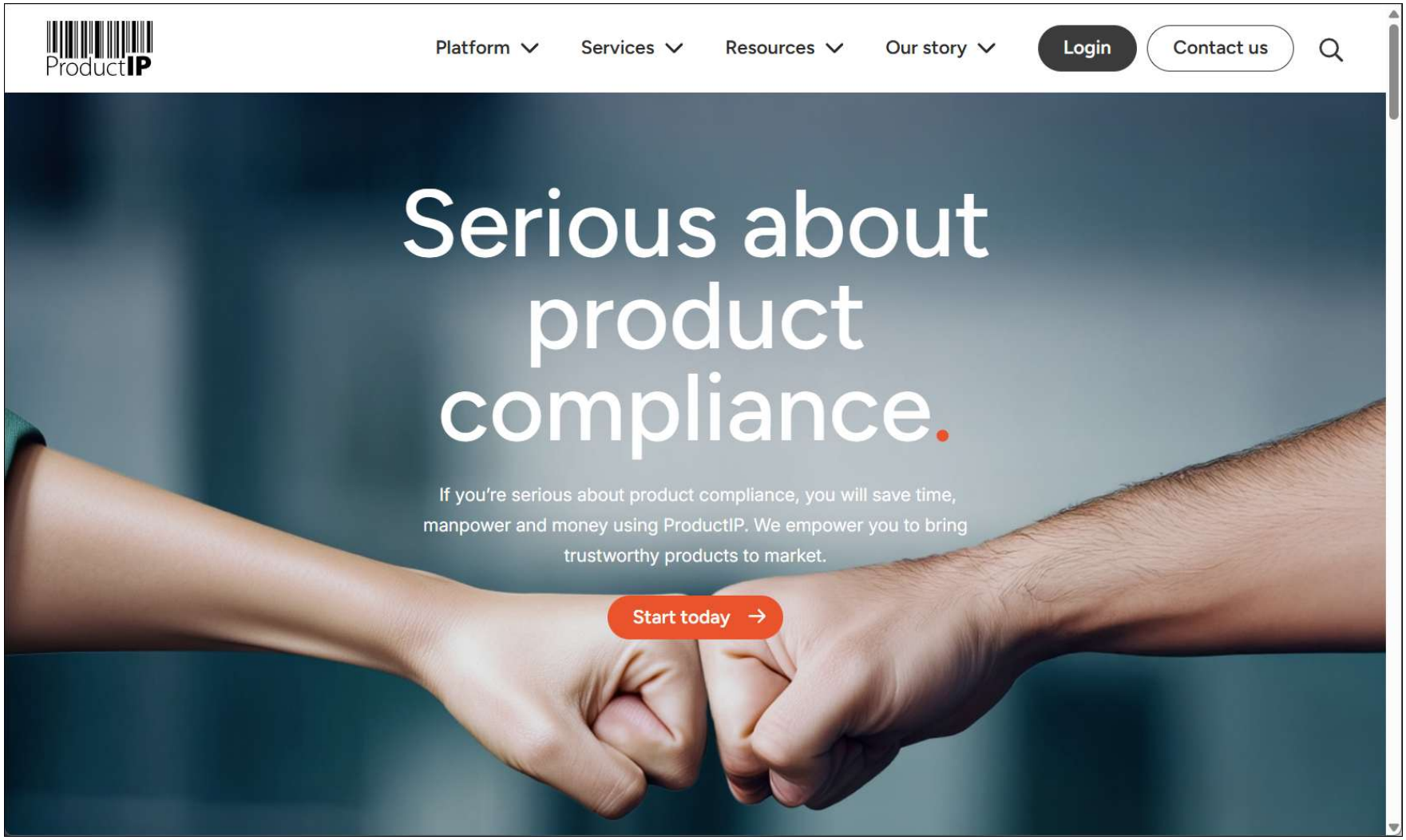
info@kassenaar.com

VANDUUREN MEDIA

ING OHRA euricom A DIMENSION DATA COMPANY sanoma delta lloyd zenito BETERE ZEKERHEID VOOR ONDERNEMERS Atos woonbron OBERON INTERACTIVE ROC West-Brabant the eforum FACTORY


Search Google Maps
See travel times, traffic and nearby places
Zutphen, NL
Amsterdam
Rotterdam
The Hague
Utrecht
Arnhem
Nijmegen
Apeldoorn
Deventer
Zwolle
Amersfoort
Eindhoven
Tilburg
Breda
Antwerp
Ghent
Bruges
Roermond
Venlo
Duisburg
Essen
Dortmund
Düsseldorf
Wuppertal
Münster
Enschede
Almelo
Hengelo
Satellite
Google

Senior frontend developer ProductIP, (50%)



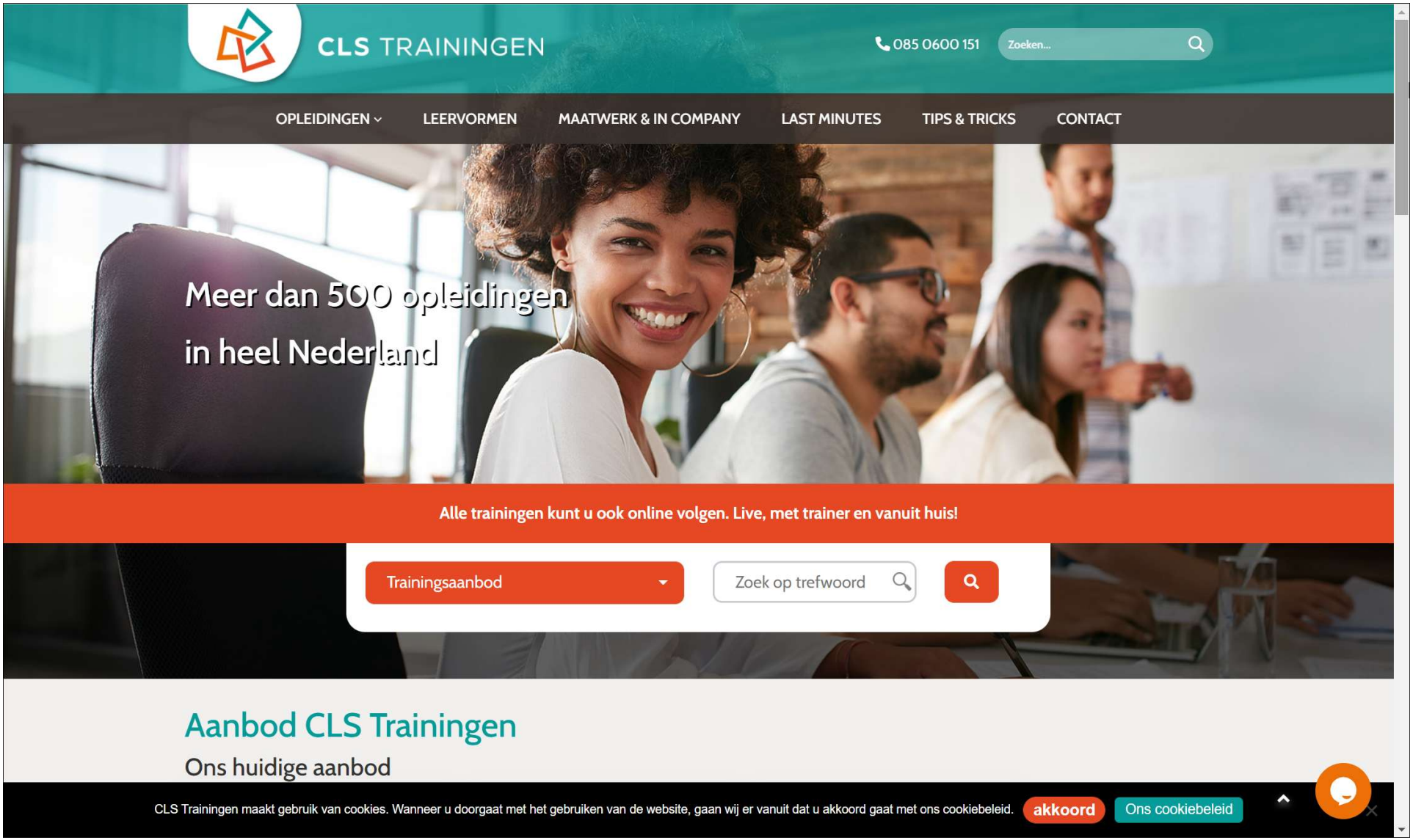
www.productip.com

peterkassenaar

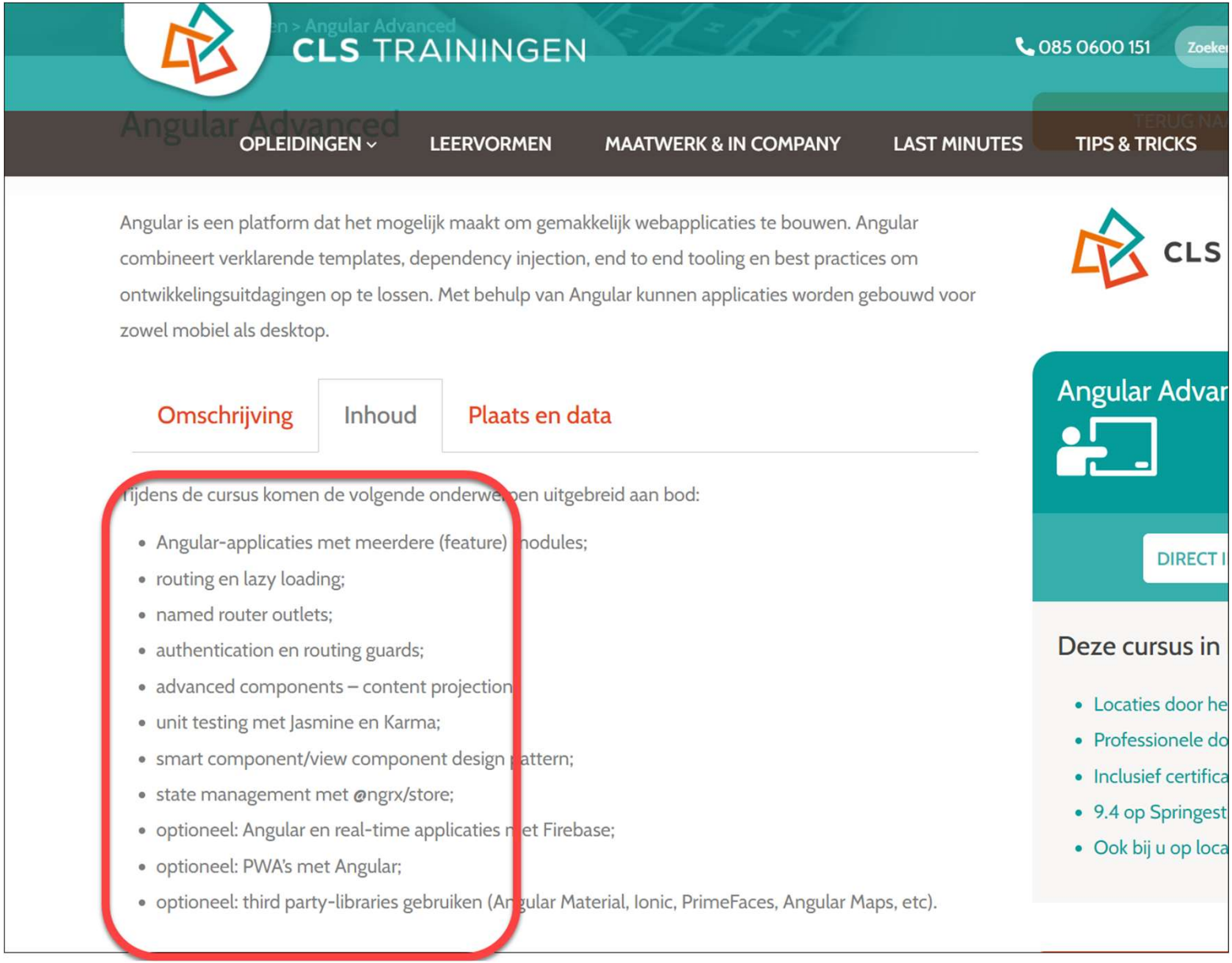
PeterKassenaar

PeterKassenaar

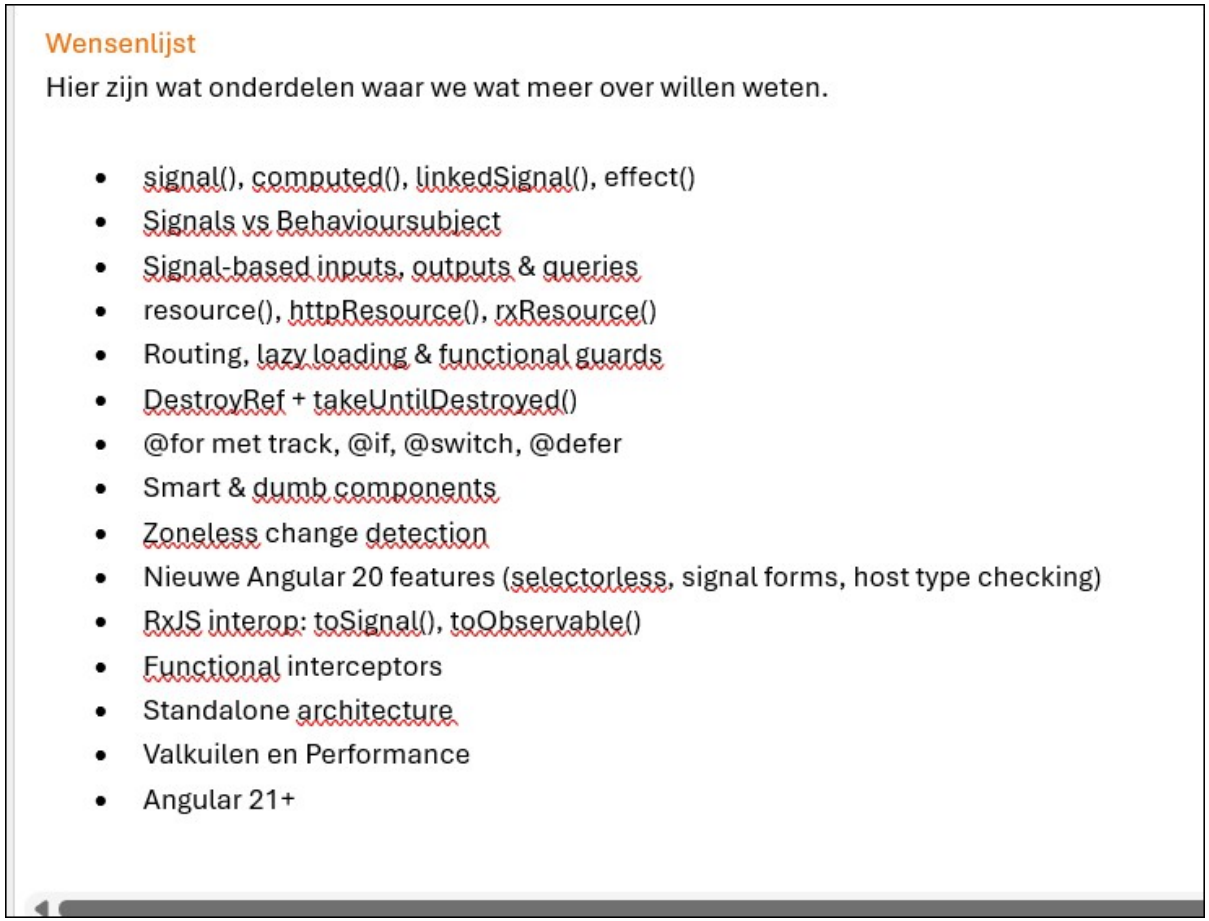
pkas06


CLS TRAININGEN
085 0600 151
Zoeken...
OPLEIDINGEN
LEERVORMEN
MAATWERK & IN COMPANY
LAST MINUTES
TIPS & TRICKS
CONTACT
Meer dan 500 opleidingen
in heel Nederland
Alle trainingen kunt u ook online volgen. Live, met trainer en vanuit huis!
Trainingsaanbod
Zoek op trefwoord
Aanbod CLS Trainingen
Ons huidige aanbod
CLS Trainingen maakt gebruik van cookies. Wanneer u doorgaat met het gebruiken van de website, gaan wij er vanuit dat u akkoord gaat met ons cookiebeleid.
akkoord
Ons cookiebeleid

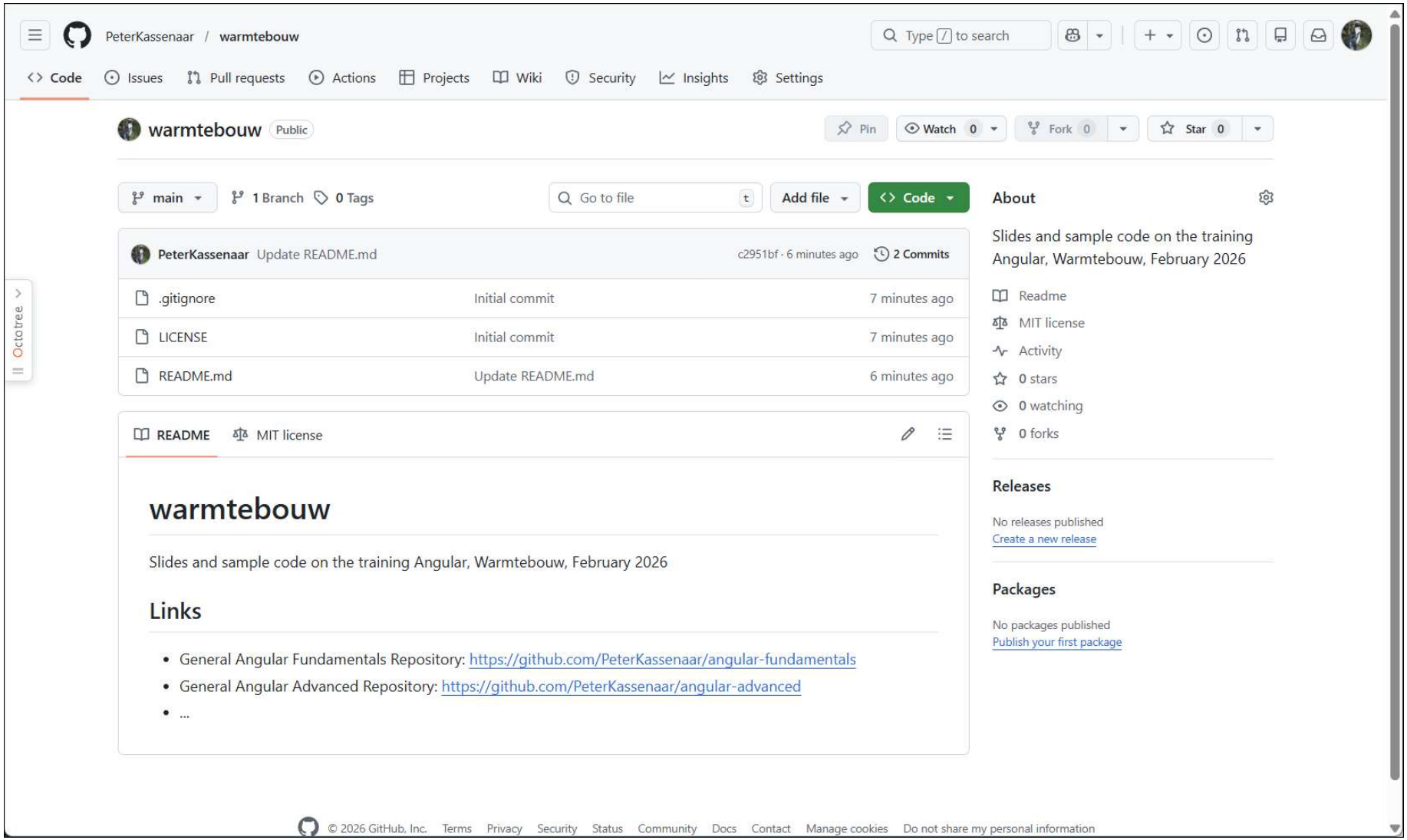
https://cls.nl/


CLS TRAININGEN
085 0600 151
Angular Advanced
OPLEIDINGEN
LEERVORMEN
MAATWERK & IN COMPANY
LAST MINUTES
TIPS & TRICKS
Angular is een platform dat het mogelijk maakt om gemakkelijk webapplicaties te bouwen. Angular combineert verklarende templates, dependency injection, end to end tooling en best practices om ontwikkelingsuitdagingen op te lossen. Met behulp van Angular kunnen applicaties worden gebouwd voor zowel mobiel als desktop.
CLS
Omschrijving
Inhoud
Plaats en data
Tijdens de cursus komen de volgende onderwerpen uitgebreid aan bod:
Angular-applicaties met meerdere (feature) modules;
routing en lazy loading;
named router outlets;
authentication en routing guards;
advanced components – content projection
unit testing met Jasmine en Karma;
smart component/view component design pattern;
state management met @ngrx/store;
optioneel: Angular en real-time applicaties met Firebase;
optioneel: PWA's met Angular;
optioneel: third party-libraries gebruiken (Angular Material, Ionic, PrimeFaces, Angular Maps, etc).
Deze cursus in
Locaties door he
Professionele do
Inclusief certifica
9.4 op Springest
Ook bij u op loca

Warmtebouw “Wish List”



github.com/PeterKassenaar/warmtebouw



About you…

Introduce yourself briefly

Knowledge of Angular, (mobile/web-) apps?

How long have you worked with Angular yet?

Tell us a little bit about your projects.

What are your expectations of this course?

Agenda – 25, 26 February 2026

~09:00 start – Morning session

~ 10:00, 11:00 Short Break

~12:00 Lunch

~12:45 Afternoon session

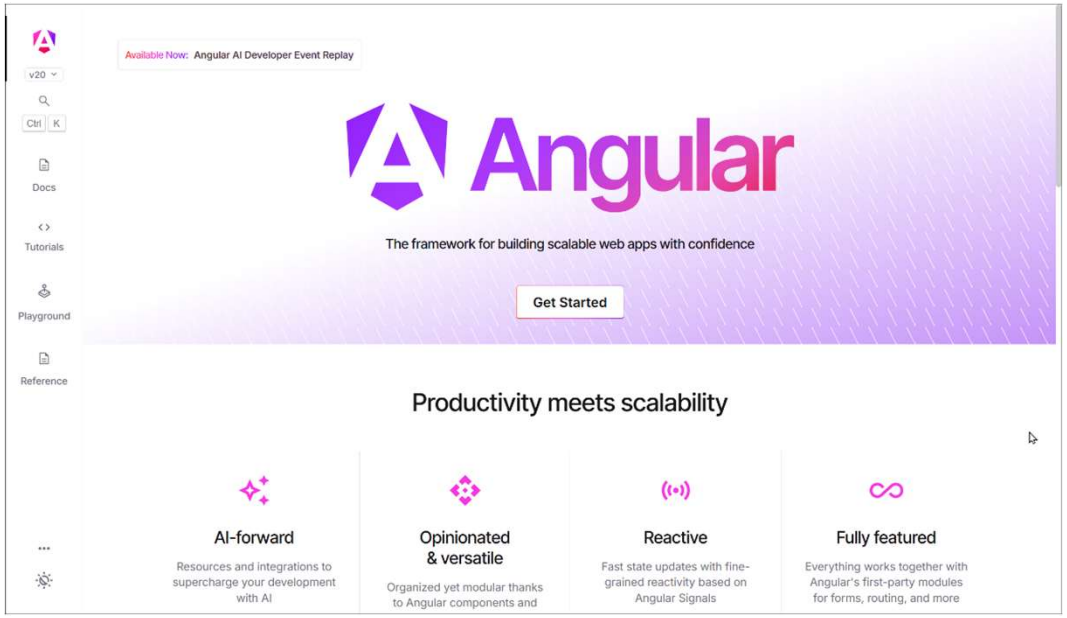
~ 14:00, 15:00 Break

~16:00-16:15 - end

Last day: if possible, wrap up a bit early

Material

Software	(Angular + Editor + Browser + libraries)
Handouts	(PDF, Github)
Workshops	(in the presentations)
Websites	(online)



angular.dev/

Short recap

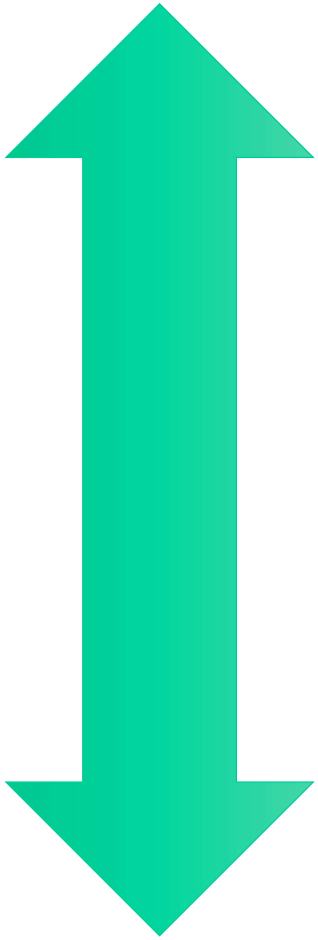
- Assumed familiar: Fundamentals
  - Concepts, context & architecture
  - Angular CLI basics
  - Components, Data binding
  - Services
  - Live API’s
  - Component communication / event buses
  - Routing
    - Basics
    - Routing Parameters

“Advanced”

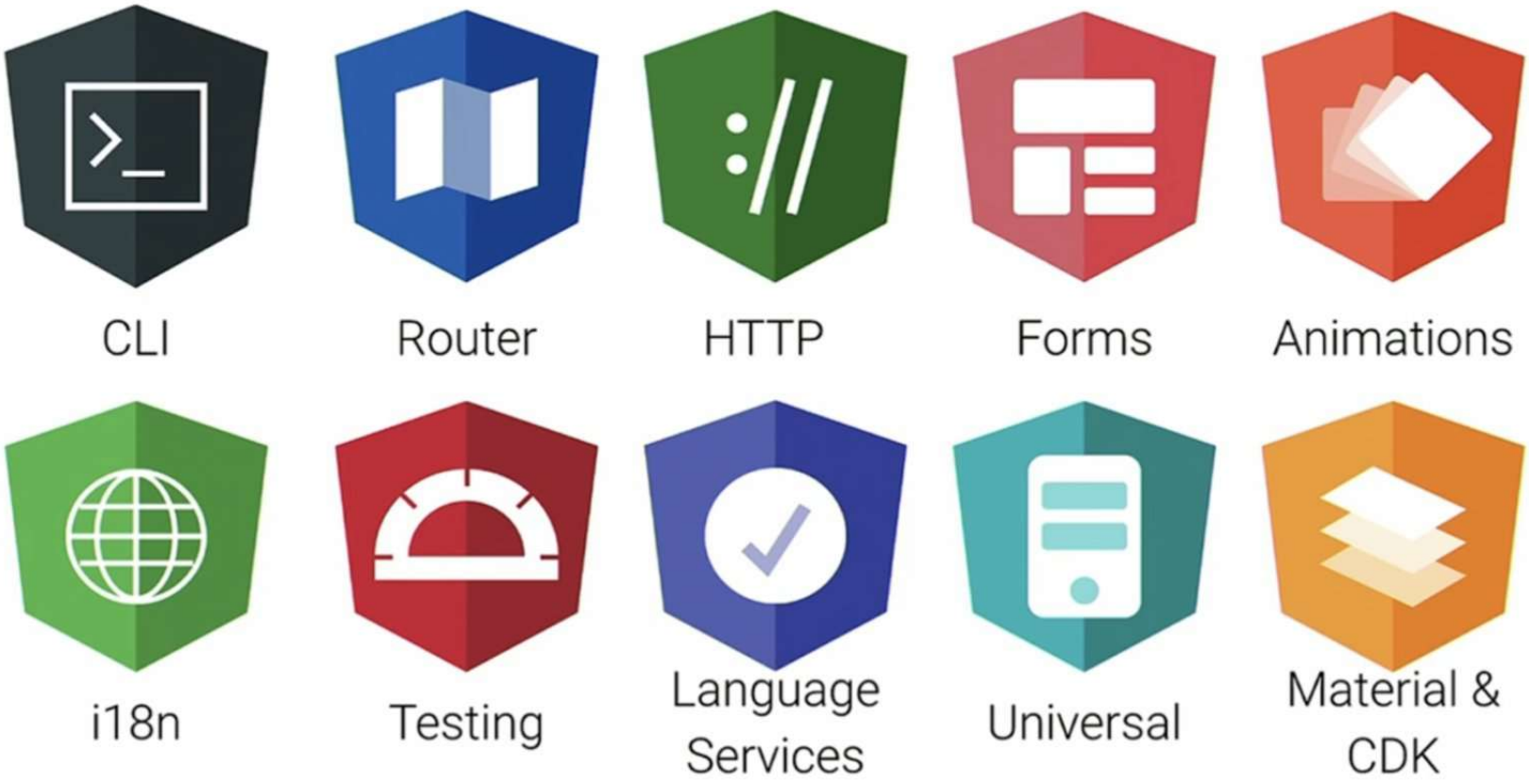
Broadening?

or…

deepening?



Why devs and enterprises like Angular...



Agenda - 2 days - Thematic

- Introduction
- Day 1: Angular New Features + refresher
  - Standalone components and module-less defaults
  - Template syntax: `@for`, `@if-then-else`, `@switch()`
  - Signals
    - Types of signals, computed, input/output
  - Dependency Injection
  - New `provideHttpClient()`
  - Components: Smart/View ~, content projection

Agenda - 2 days- Thematic

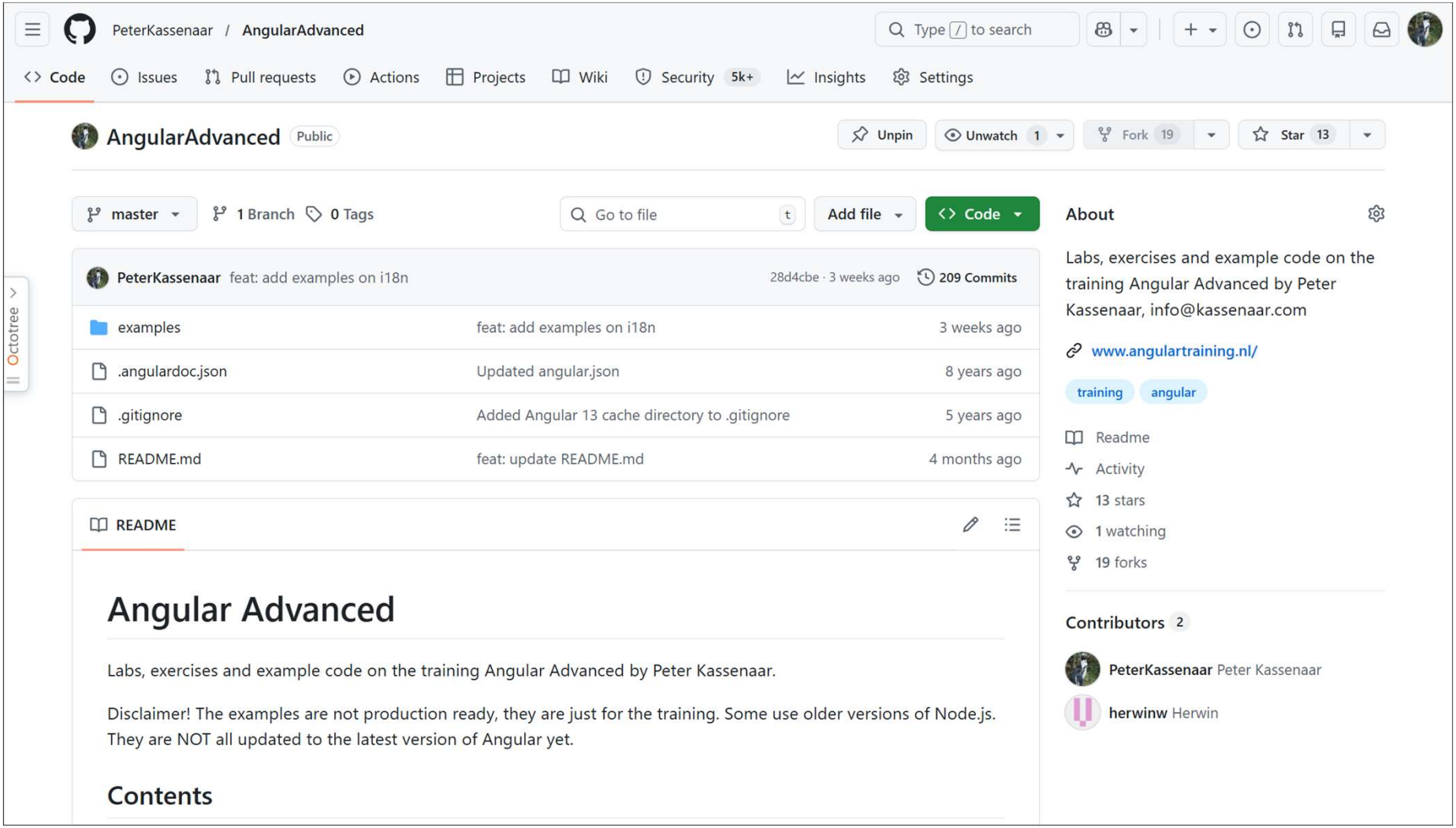
- Day 2: Miscellaneous
  - Routing, Lazy Loading, Guards
  - Unsubscribing with `DestroyRef`
  - RXJS Patterns
  - …
- Overall : Best practices on coding & architecture

Labs and example code

1. Labs/Exercises
    - In the PDF's in the Github-repo. But: feel free to deviate. Adapt to suit your own needs! (hobby, work, current projects)

2. Example code
    - Executions of the exercises, small projects (`npm install`, `npm start`)
    - Work in progress – let me know of additions/errors!
    - github.com/PeterKassenaar/AngularAdvanced

Generic 'Advanced' Github repo



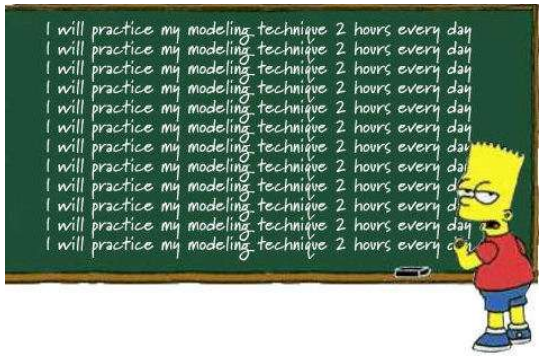
https://github.com/PeterKassenaar/AngularAdvanced

Overall process

…

On Workshops…

- … are designated with a slide like this 👇
- Are between the talks with theory
- Are NOT written out line-by-line. You have to think for yourself
- The time during the training may be too short to finish all the workshops
  - Do them in your own time
  - At least you know the concepts the workshop is about
  - Choice – we have more to discuss, I let that prevale
- The example code often (but not always!) contains the ‘solution’ to the workshop. Use it! It’s there for you
  - But of course it would be nice if you can work with your own project/data



Questions?

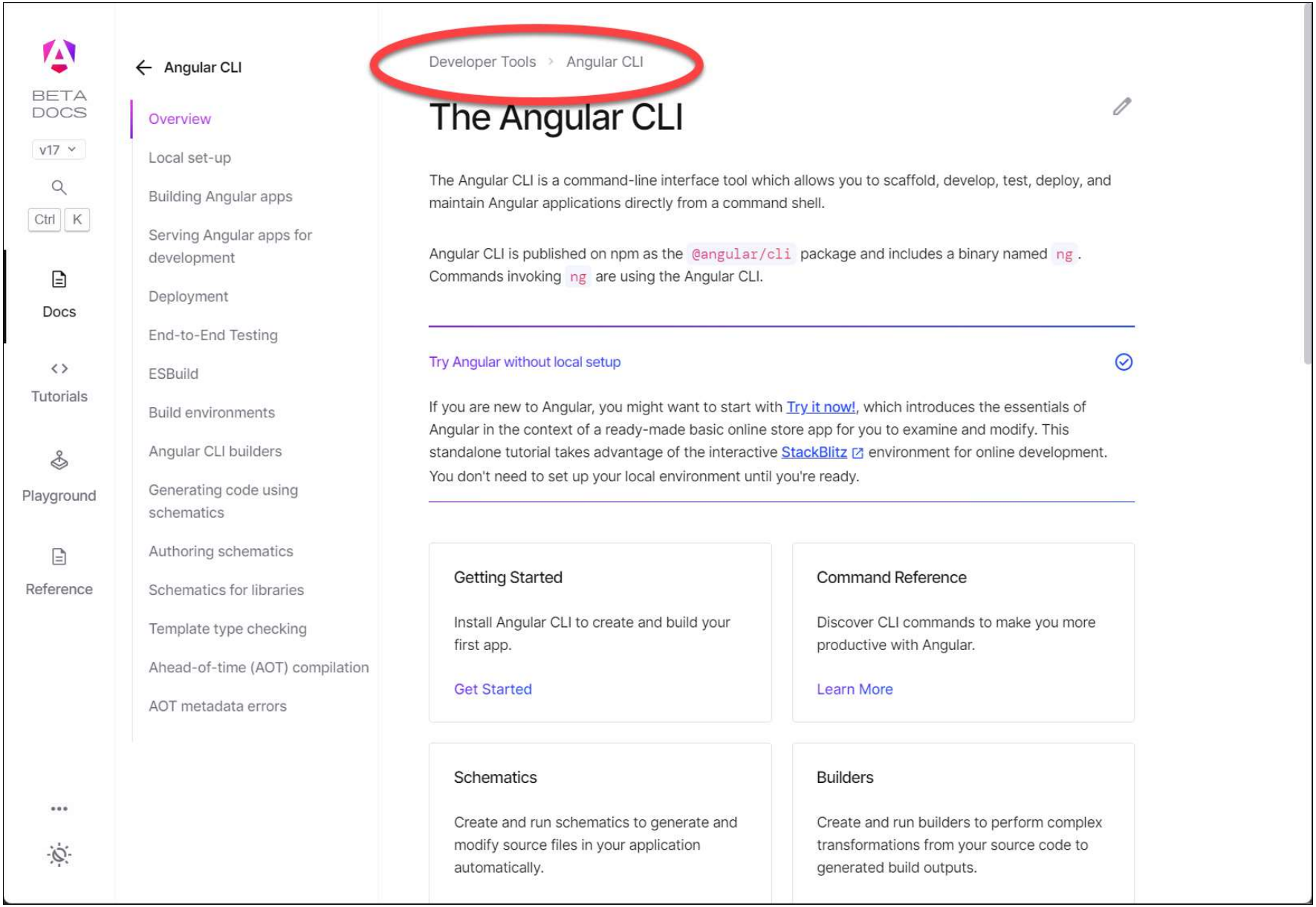
Angular CLI

Scaffolding new projects, new options

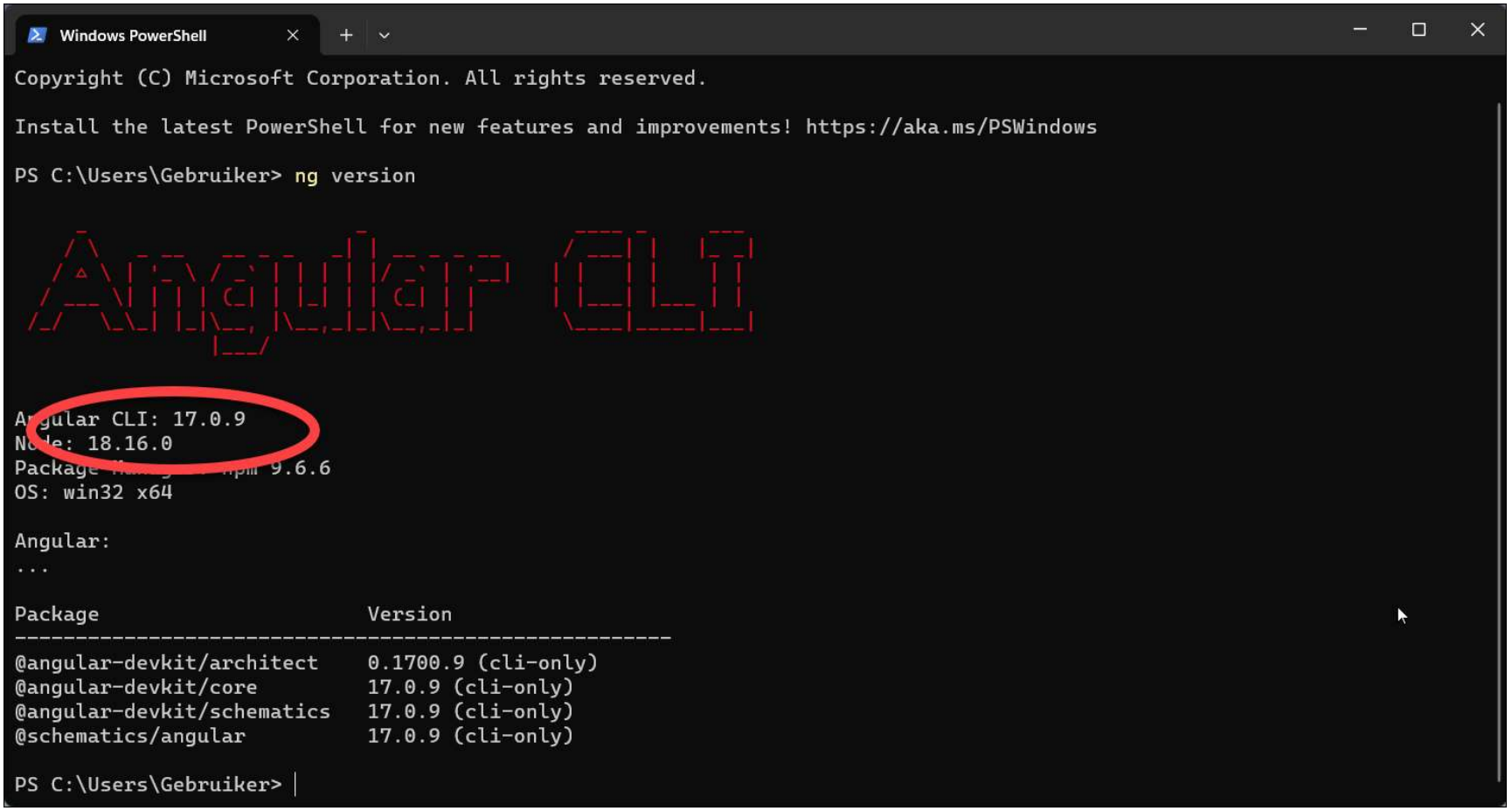
Angular CLI

- Command Line Interface for
  - Scaffolding (`ng new`),
  - Developing (`ng generate`)
  - Testing (`ng test`)
  - Deploying applications (`ng deploy`)
- Local installation
  - SO: no online environment like Stackblitz or CodeSandbox

```
npm install -g @angular/cli
```



https://angular.dev/tools/cli



Current version? `ng version`

Latest version: make sure you have at least Node 20+

New in Angular CLI v.17+

- Modern output format using ESM
  - dynamic import expressions to support lazy module loading.
- Faster build-time performance
  - both initial builds and incremental rebuilds.
- Newer JavaScript ecosystem
  - tools as `esbuild` and Vite.
- Integrated SSR and prerendering capabilites

ESBuild – extremely fast builder

esbuild

Try in the browser

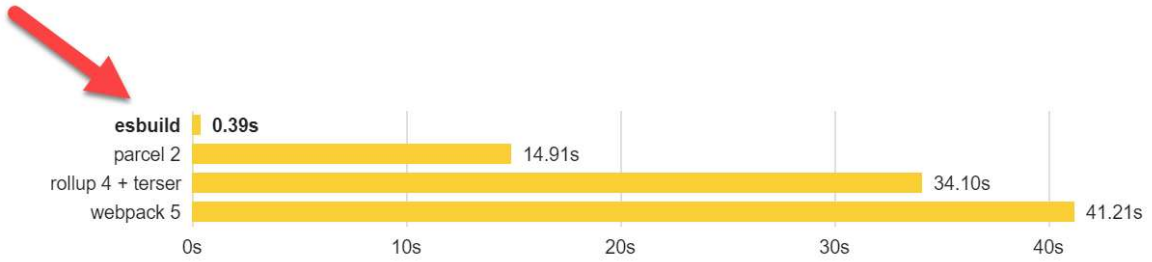
Getting Started
- Install esbuild
- Your first bundle
- Build scripts
- Bundling for the browser
- Bundling for node
- Simultaneous platforms
- Using Yarn Plug'n'Play
- Other ways to install

API
- Overview
- General options
- Input
- Output contents
- Output location
- Path resolution
- Transformation
- Optimization
- Source maps
- Build metadata
- Logging

Content Types

esbuild

An extremely fast bundler for the web



Above: the time to do a production bundle of 10 copies of the three.js library from scratch using default settings, including minification and source maps. More info here.

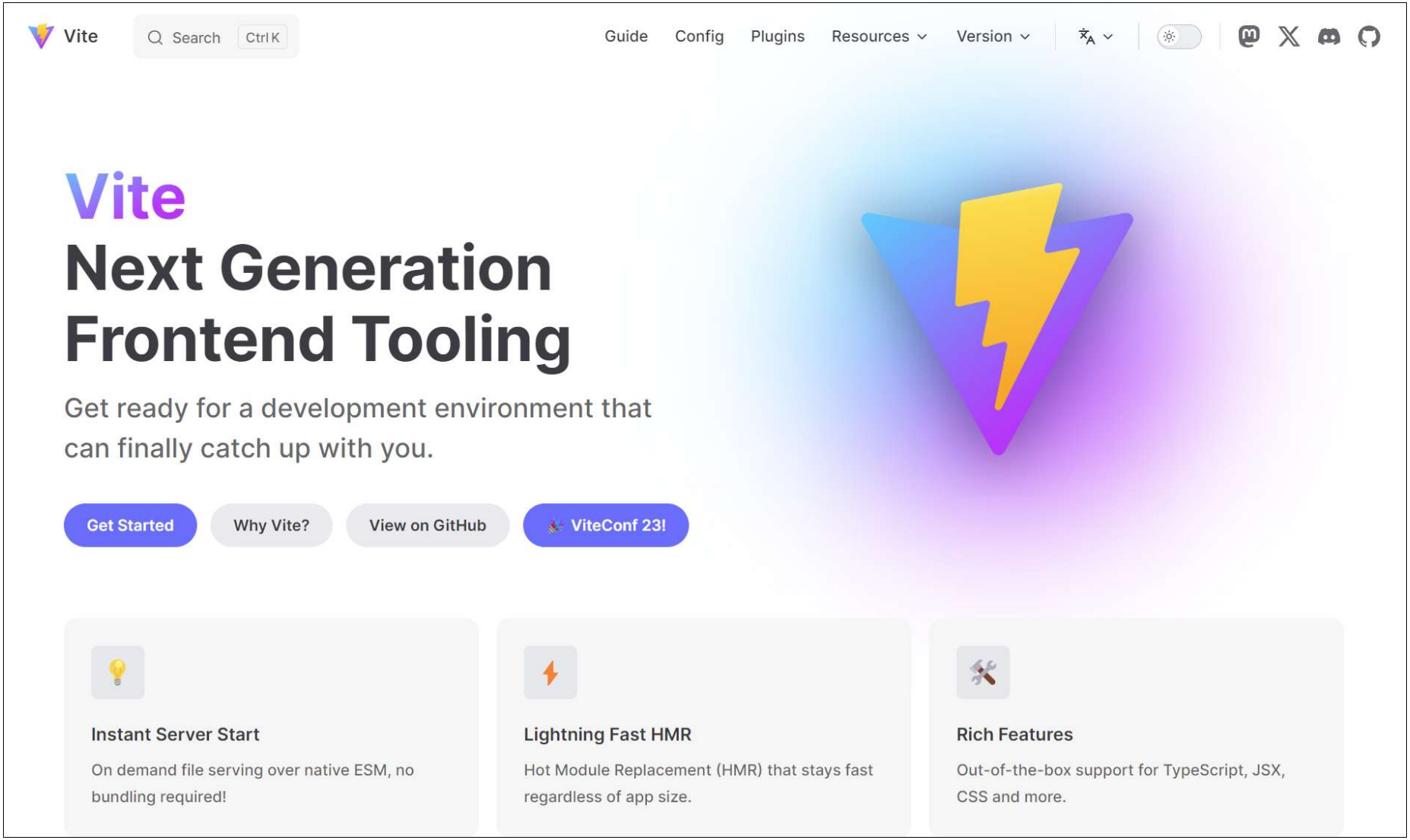
Our current build tools for the web are 10-100x slower than they could be. The main goal of the esbuild bundler project is to bring about a new era of build tool performance, and create an easy-to-use modern bundler along the way.

Major features:

- Extreme speed without needing a cache

https://esbuild.github.io/

Vite.js – modern bundler



https://vitejs.dev/

New in CLI

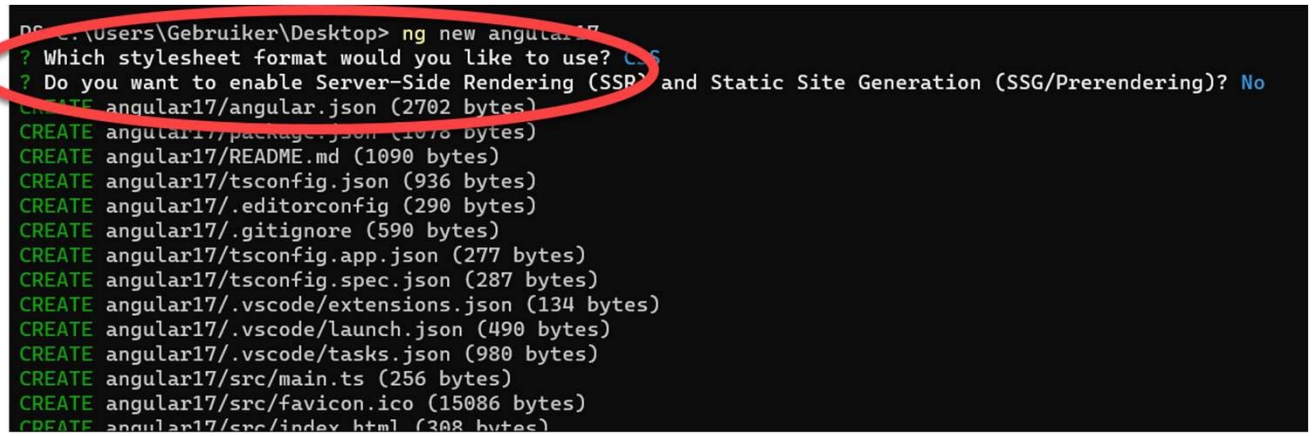
- Option syntax: Unix/POSIX conventions
- Boolean options:
  - `--some-option` sets `--some-option` to `true`.

    Alternative: `--some-option=true`
  - `--no-some-option` sets `--some-option` to false.

    Alternative: `--some-option=false`
  - Example: `ng new --no-create-application`
- Array options:
  - Space separated or repeated
  - `--option value1 value2 …`
  - `--option value1 --option value2 …`

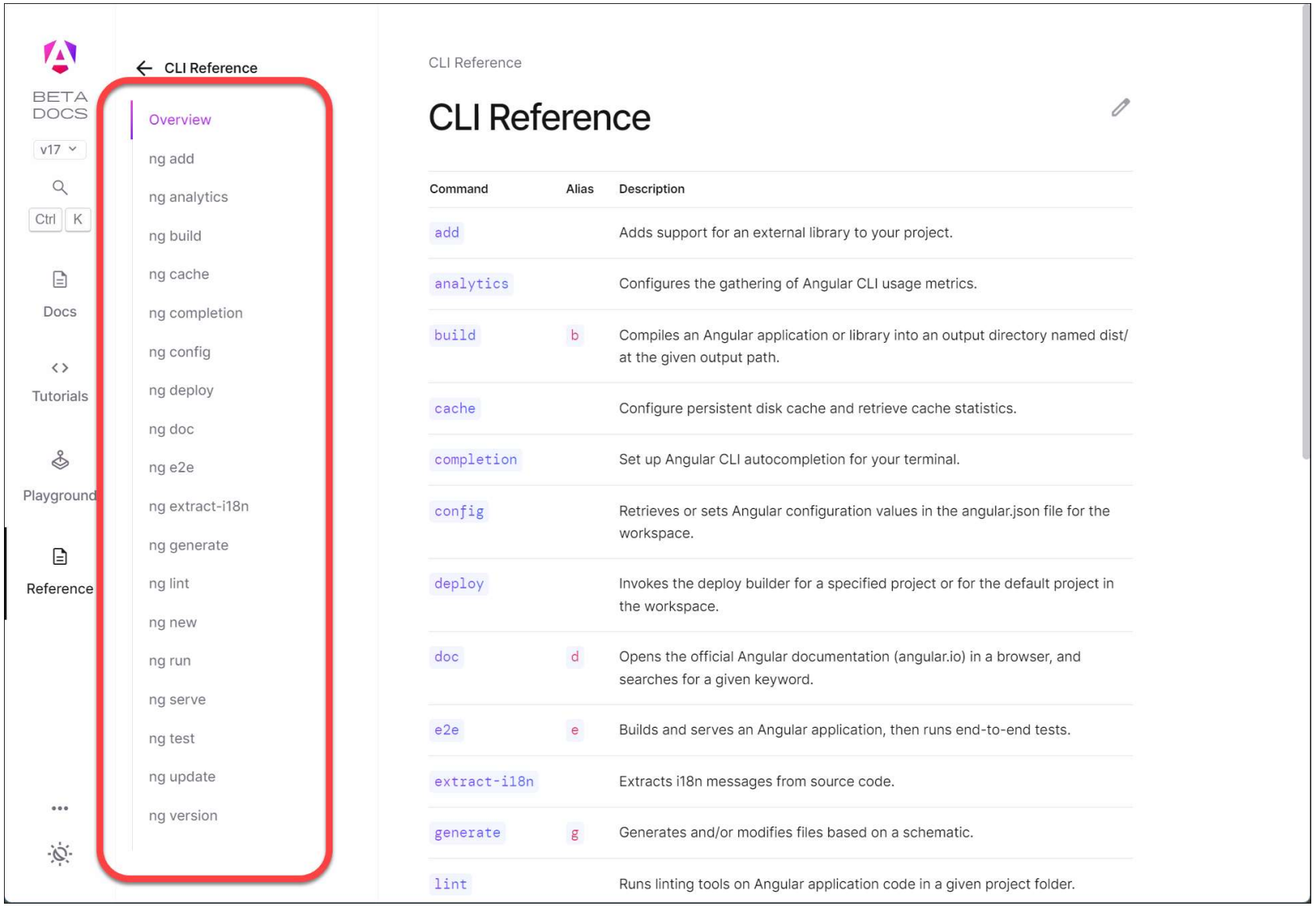
Creating a new application – `ng new`

- Creating an application: like before, using `ng new application-name`
- New options
  - NO more `Routing Y/N` – now enabled by default
  - Picking a `CSS variant` – same
  - `Enable pre-rendering SSG / SSR` (default: `No`)
  - Picking an AI option to store defaults (`gemini.md`, `claude.md`, etc).




Demo

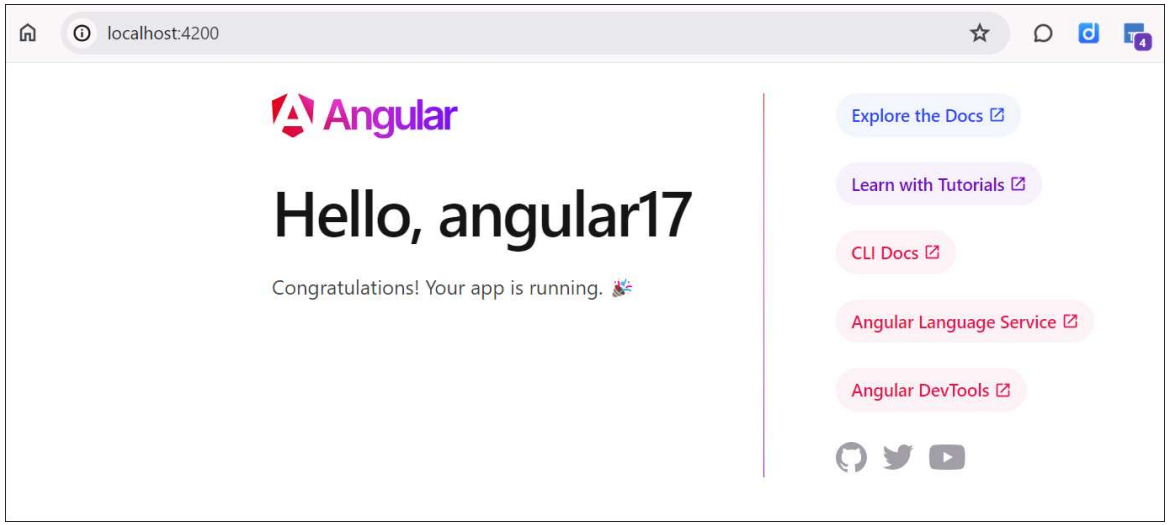
CLI reference

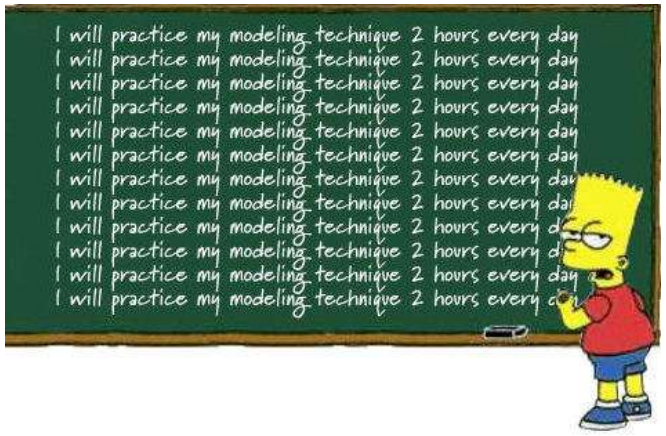


https://angular.dev/cli

Workshop

- Install angular CLI – make sure you have the latest version
  - `ng version`
- Create a new application, using the default values
- Open the application in your editor and run it
  - `ng serve –open`
- Check components, see what has changed already





Standalone Components

Towards an `ngModule`-less future

Standalone components

- Traditionally: ngModule-based components
  - All components belong to an `@ngModule()`.
  - An `@ngModule()` acts as a container for similar functionality (Customers, Products, Login, and so on)
- Modern applications: standalone components
  - Components don’t have to belong to a module
  - You can mix & match!
  - Better performance
  - More component-level imports
  - In preview since Angular 13+, default in Angular 17+

What are standalone components?

Regular Angular Components with the `standalone : true` option

Angular 19+? No `standalone:true` option, but standalone is default!

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

@Component({
  selector: 'app-hello',
  standalone: true,
  imports: [],
  templateUrl: './hello.component.html',
  styleUrl: './hello.component.css'
})
export class HelloComponent {

}
```

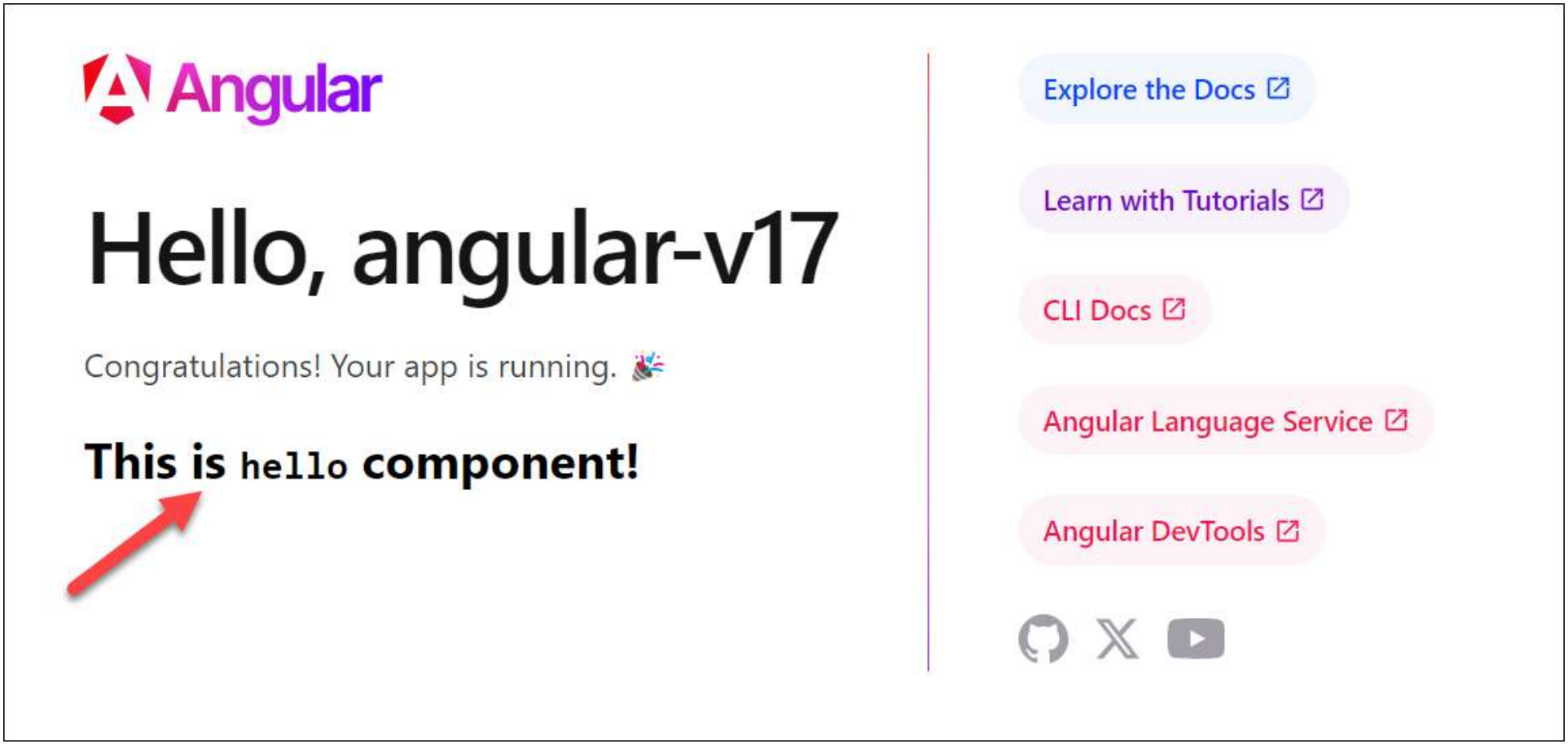
Using standalone components

Simply import them in the component where you want to use it

```
import { Component } from '@angular/core';
import {HelloComponent} from "./hello/hello.component";

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [ HelloComponent],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
}
```

```
<h1>Hello, {{ title }}</h1>
<p>Congratulations! Your app is running. 🎉</p>
<app-hello/>
```


Angular
Hello, angular-v17
Congratulations! Your app is running.
This is hello component!
Explore the Docs
Learn with Tutorials
CLI Docs
Angular Language Service
Angular DevTools

Note

- Use `import {…} from …`
  - Otherwise it will NOT work – but NO ERROR will be thrown!
  - The browser just sees an unknown tag and renders nothing
  - Most IDE's handle this automatically

```
1   import { Component } from '@angular/core';
2   import {HelloComponent} from "./hello/hello.component";
3
    1+ usages   • PeterKassenaar
```

Using Standard Directives

- Standard directives are made available as standalone directives
- We have to `import` them in the component

```
<h2 *ngIf="username">
  Username: <code>{{ username }}</code>
</h2>
```

```
import { Component } from '@angular/core';
import {NgClass, NgFor, NgIf} from "@angular/common";

@Component({
  selector: 'app-hello',
  standalone: true,
  imports: [NgIf, NgClass, NgFor],
  templateUrl: './hello.component.html',
  styleUrl: './hello.component.css'
})
export class HelloComponent {
  username = 'info@kassenaar.com';
}
```

But...new control flow syntax

- NO imports are needed when using the new control flow syntax, like `@if`, `@for`, and so on.
- So this will work out of the box:

```
@if (username) {
  <h2>
    Username: <code>{{ username }}</code>
  </h2>
}
```

Standalone Pipes, Directives

- The same is true for standalone Pipes, Directives and more
  - Just add the `standalone: true` flag in Angular 19-

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'capitalize',
  standalone: true
})
export class CapitalizePipe implements PipeTransform {

  transform(value: string):string {
    return value.toUpperCase();
  }
}
```

```
@Component({
  selector: 'app-hello',
  standalone: true,
  imports: [CapitalizePipe],
  templateUrl: './hello.component.html',
  styleUrl: './hello.component.css'
})
```

Why Standalone components?

- Removing the need for NgModules – no extra complexity
  - Handy for beginners
- But: "I have to import everything in that component. Really?"
  - Yes – but IDE's handle that automatically for you
- Now, components can be lazy loaded instead of complete modules
  - More modularization, less monolithic
  - Performance!

> *The main benefit of standalone components is that they make it trivial to develop a fully lazy-loaded application, or migrate an existing application and make it fully lazy-loaded.*

Mixing and matching

- No need for a 'big bang' when refactoring
- Using standalone components in NgModule based applications
- Standalone components act as other modules, so `import` them:

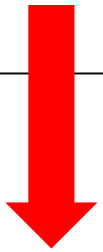
```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import {HelloComponent} from "../hello/hello.component";

@NgModule({
  declarations: [classicComponent],
  imports: [
    CommonModule,
    HelloComponent
  ]
})
export class CustomerModule { }
```

Using NgModules in standalone components

- Export the component from the module as usual
- Import the module in the component as (now) usual

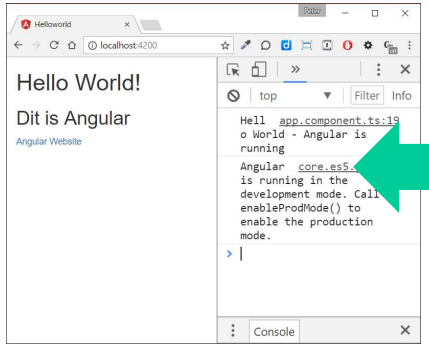
```
@NgModule({
  declarations: [
    CustomerComponent
  ],
  imports: [
    CommonModule,
  ],
  exports:[
    CustomerComponent
  ]
})
export class CustomerModule {
```



```
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [HelloComponent, CustomerModule],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
```

Combined NgModules/Standalone

Classic Angular apps



main.ts / bootstrapper

ngModule / root module

AppComponent

Other components

Other components

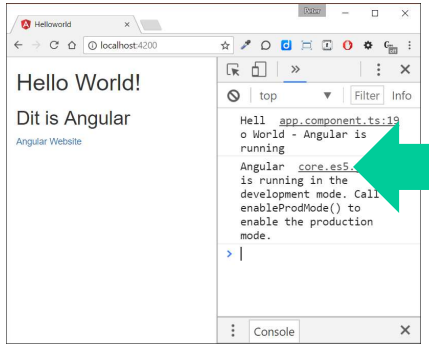
Other components

Services

3rd party modules

Other modules

Modern Angular apps



main.ts / bootstrapper

AppComponent

Services

3rd party libs

Other components

Other components

Other components

Other components

(Note: no `ngModule` anymore, as this is now handled by standalone components)

Workshop

- Open repo `ngx-new-features` and run it (`npm install`, `npm start`)
- Study the standalone components structure and architecture
- Going old school: create a new module
- Create a new component inside this new module and give it some UI.
- Include the module in the Main Standalone Component and show it besides other components
- Then the other way: Include the 'old' module in the component and show it's contents
  - You can mix and match!
  - No big bang needed