Angular Advanced Module - Monorepo's

Peter Kassenaar
info@kassenaar.com

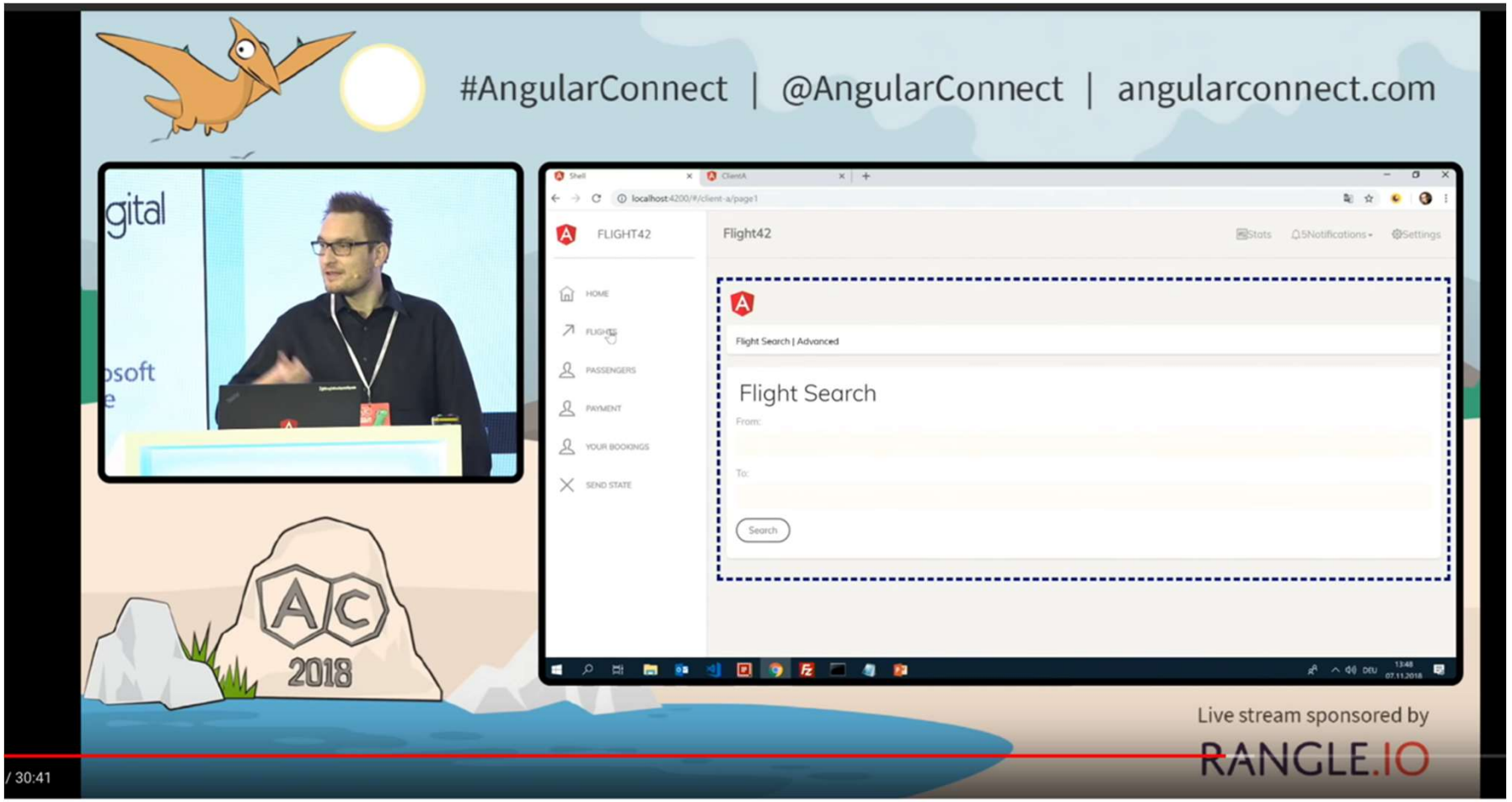
Angular in the Enterprise

- When? With (really) big(ger) applications
- Multiple solutions, examples:
  - Monorepo approach:

    https://github.com/PeterKassenaar/ng-monorepo
  - Micro-app approach:

    https://github.com/PeterKassenaar/ng-microfrontends

Manfred Steyer – Angular Connect



https://www.youtube.com/watch?v=YU-fMRs-ZYU

Code: https://github.com/PeterKassenaar/angular-microapp

Enterprise applications – multiple options

- There are always *multiple solutions*
- There is NOT one solution that is 'the best'

- Options:

1. NPM packages

   Publish your own packages/libraries to `npm`, so others can `npm install` them

2. Monorepo

   Multiple projects in one code base, optionally sharing code

3. Micro-apps / micro-frontends

   Multiple applications, not sharing code, optionally different techniques/frameworks

So basically...

NPM packages

Monorepo

Micro apps

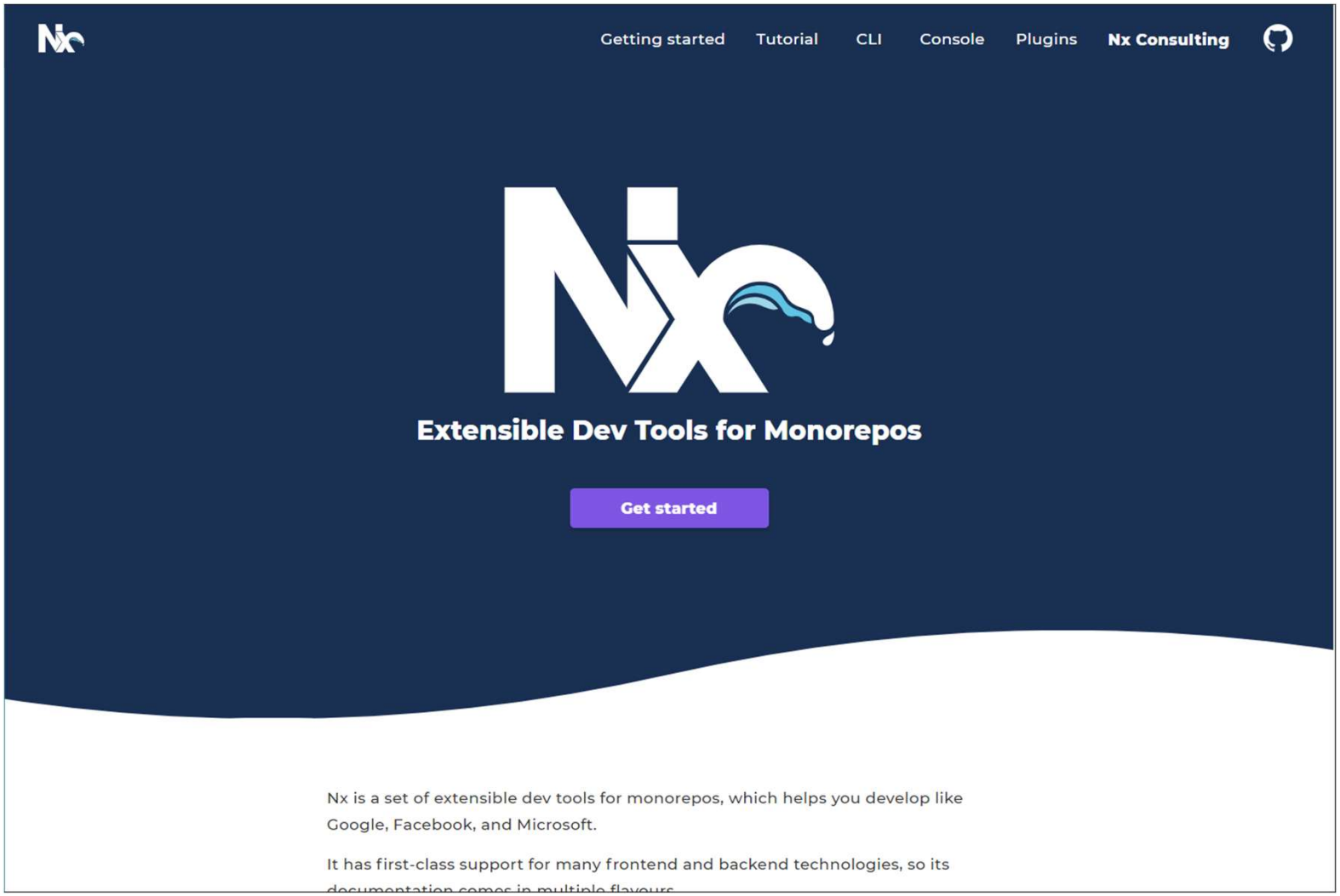
Monorepos

Multiple projects in one solution, optionally sharing code

What is a monorepo

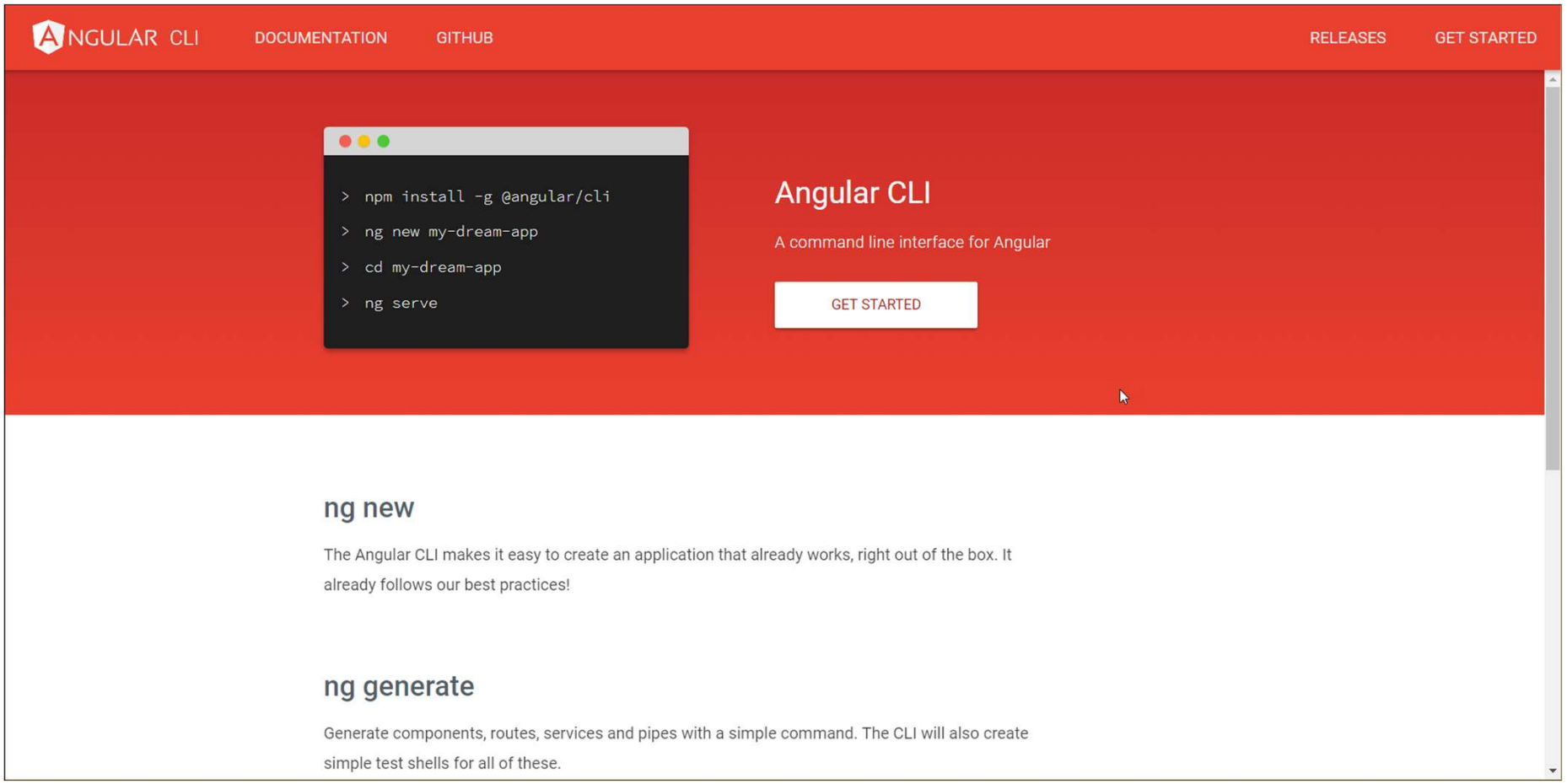
- Often, bigger applications are not only split up into modules, but also in other applications
- Applications generally share a lot of code
  - Sharing components
  - Sharing services
  - Sharing pipes, other logic, etc…
- One (opiniated) solution – create a so called *monorepo*
- There are tools that also cover this
  - Nx Framework – free, open source, https://nx.dev/
  - Angular CLI as of V6.0.0+, https://cli.angular.io/

Nrwl extensions to create a monorepo



https://nx.dev/

Angular CLI – as of V6.0+

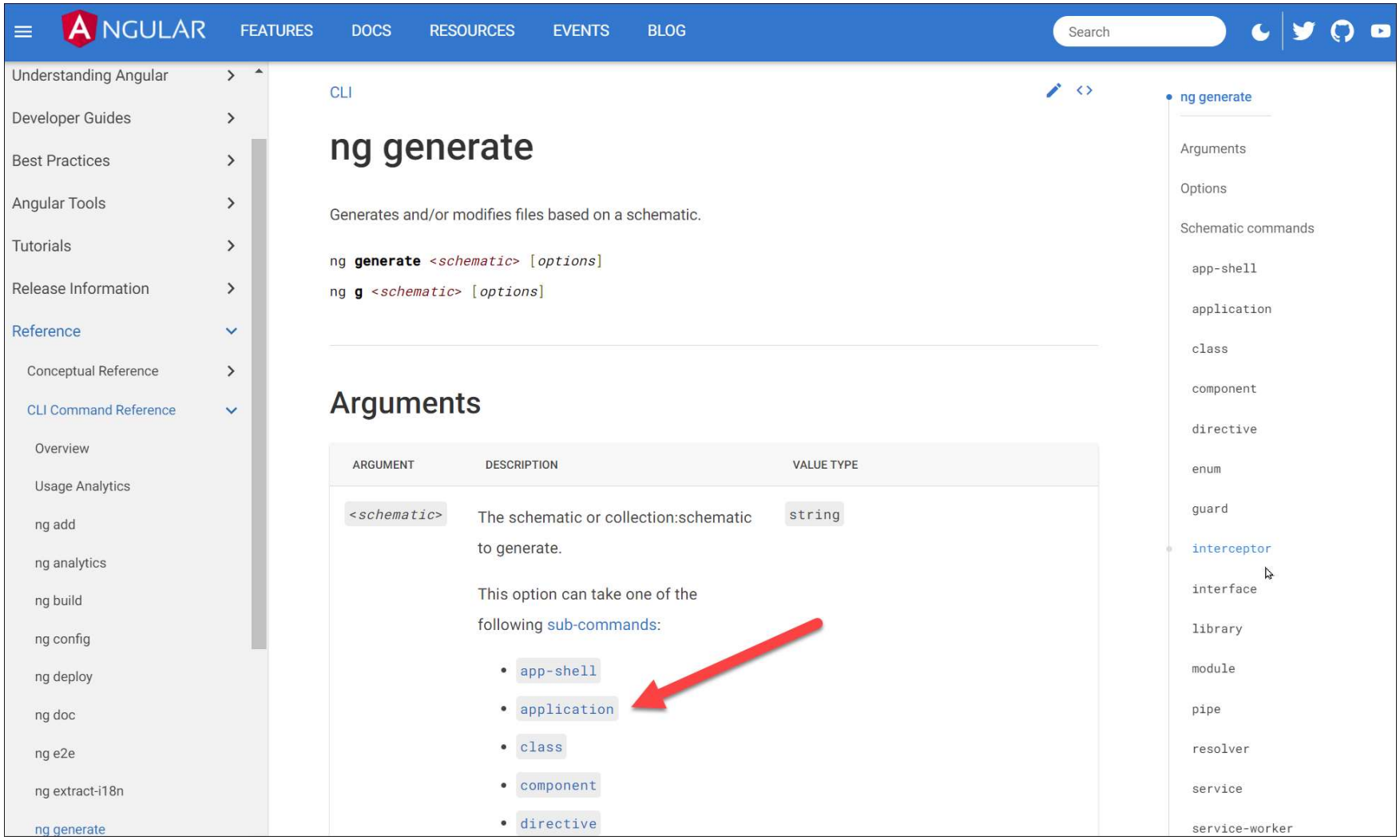


Which one to use?

- This is highly *opiniated*
- The work (roughly) the same
  - Creating a workspace
  - Creating applications and libraries inside that workspace
- Nx is also available for React
- Angular CLI is limited to (duh…) Angular
  - But…Nx is a wrapper around Angular CLI

mostly - Personal Preference!

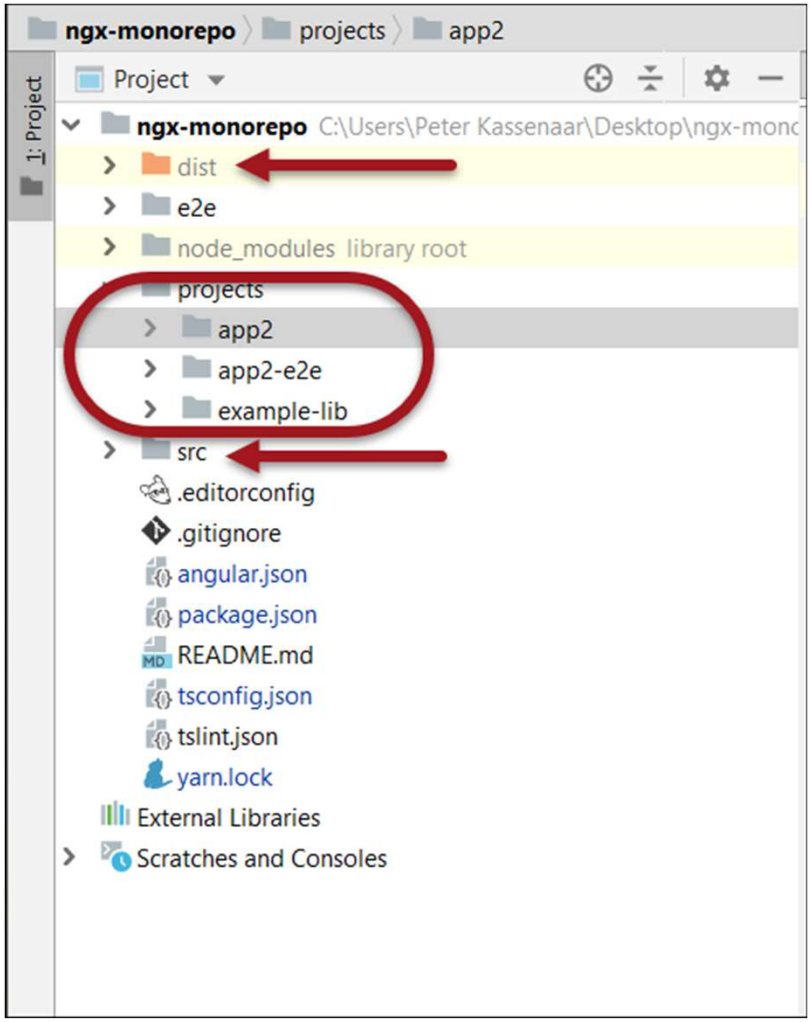
We'll be using Angular CLI in this demo



Inside an Angular monorepo

- One `/node_modules`
- One main `package.json`
- `angular.json`, describing all the projects in the workspace
- Optional - One root app, in `/src` folder
- One `/projects` folder, containing:
  - A folder for every project
  - Libraries
  - Applications
  - This folder is created upon the first creation of a `library` or `application`
- `/dist` folder, holding the compiled Angular Packages that needs to be shared

Structure/architecture

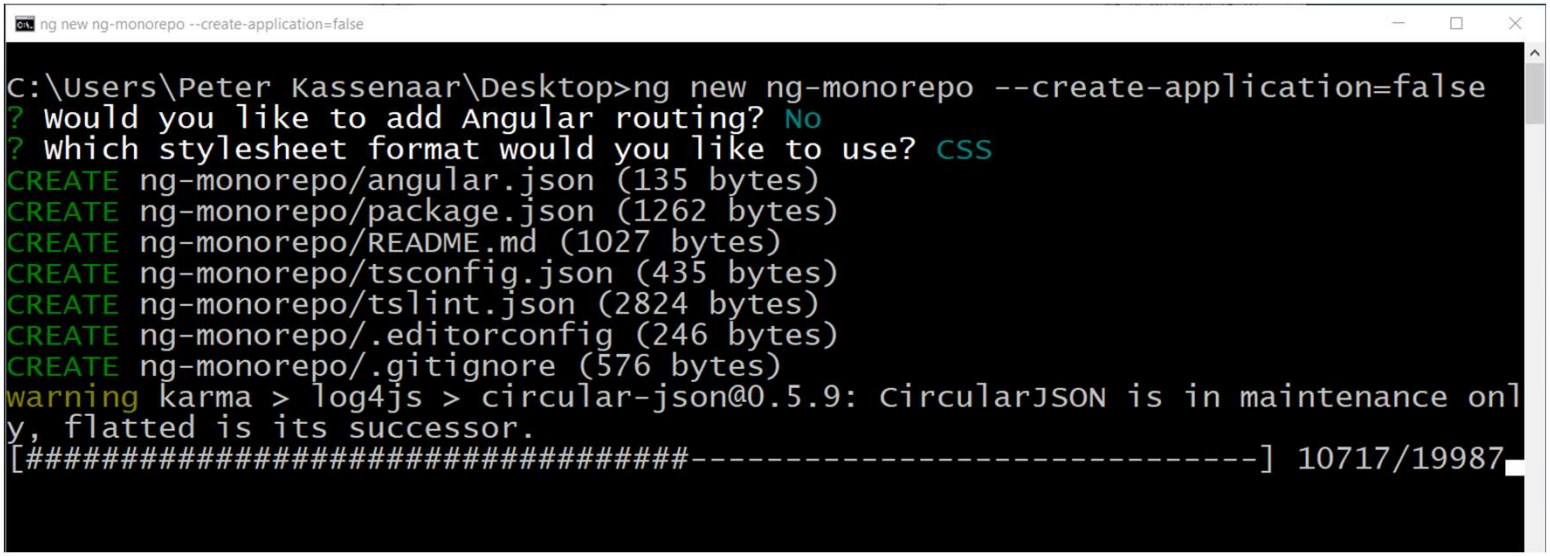


Credits: Angular in Depth

https://blog.angularindepth.com/creating-a-library-in-angular-6-87799552e7e5

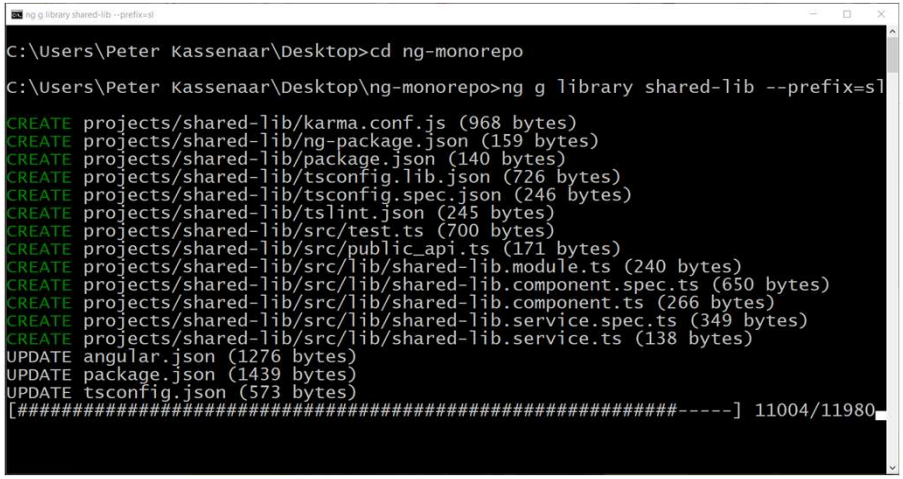
High level overview of the steps

1. Create the container monorepo using `ng new <mono-repo-name>`
   - We call this folder a *workspace*
2. Create *without* the main app: `ng new <name> --create-application=false`!
   - https://blog.angularindepth.com/angular-workspace-no-application-for-you-4b451afcc2ba



Create (shared) lib

- Generate the (shared) library
  - `cd ng-monorepo`
  - `ng generate library shared-lib --prefix=sl` (or some other prefix)
  - Angular also updates the global `angular.json`, `package.json` and `tsconfig.json`
- Always use custom prefixes on libraries and projects
  - distinguish components and services!



angular.json

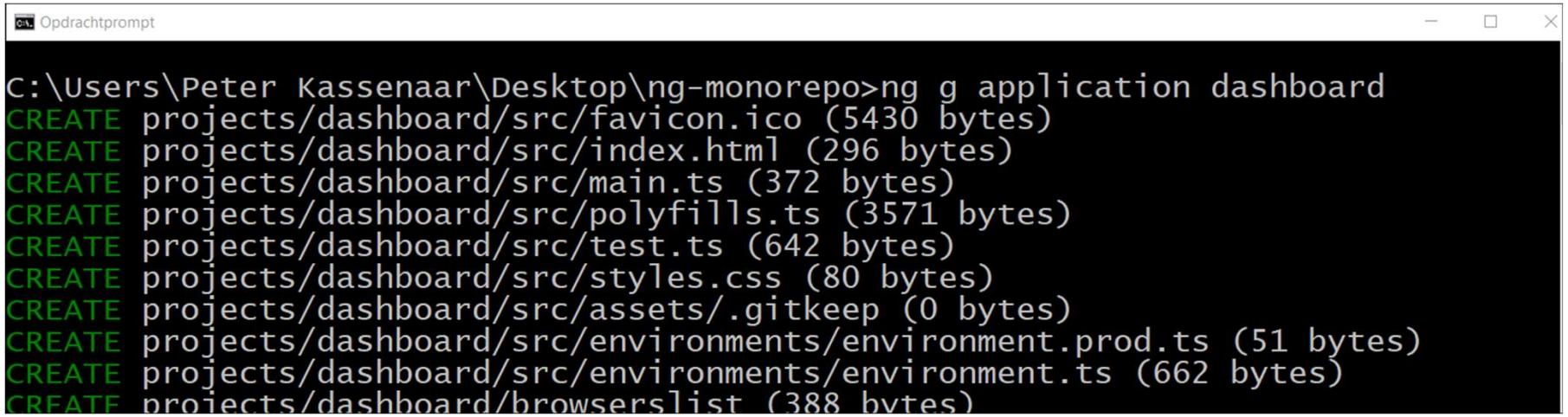
```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "shared-lib": {"root": "projects/shared-lib"...}
  },
  "defaultProject": "shared-lib"
}
```

```
    "lib": [
      "es2018",
      "dom"
    ],
    "paths": {
      "shared-lib": [
        "dist/shared-lib"
      ],
      "shared-lib/*": [
        "dist/shared-lib/*"
      ]
    }
  }
}
```

`tsconfig.json` – added paths, so we can import shared stuff easily later on

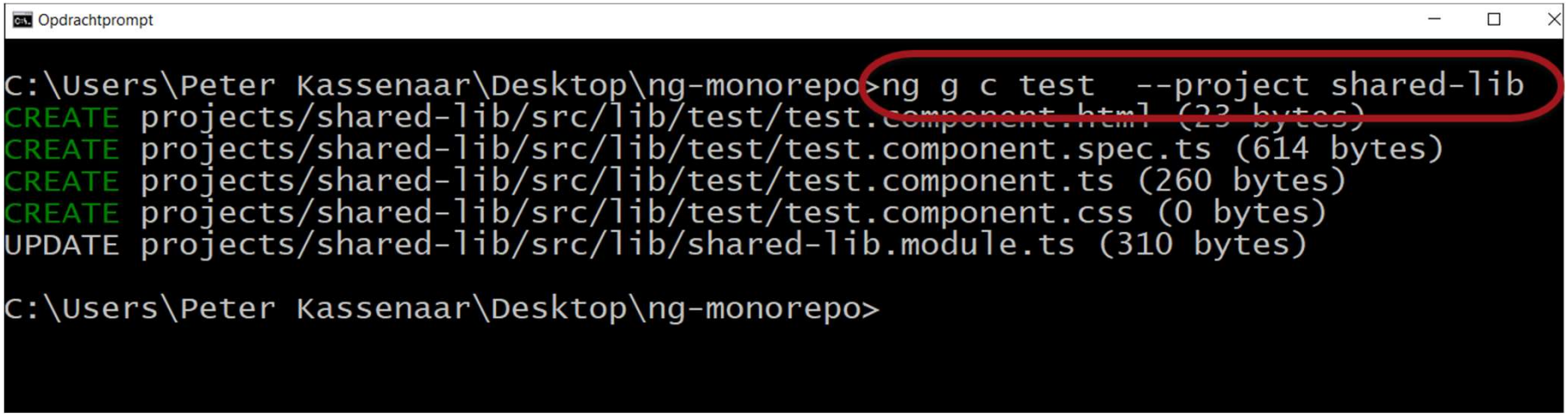
Create first app in the monorepo

- Generate the (first) application
  - `ng generate application <application-name>`
  - Again, `angular.json` and `package.json` are updated with the new project



Create a shared component

- Of course you can create multiple components
  - `ng generate component <component-name> --project shared-lib`
  - Use the `--project` flag to tell the CLI what project you want to add the component to
- Give the component some UI



Export the component

- Add it to the exports array of the `shared-lib.module.ts`

  `exports: […, TestComponent]`

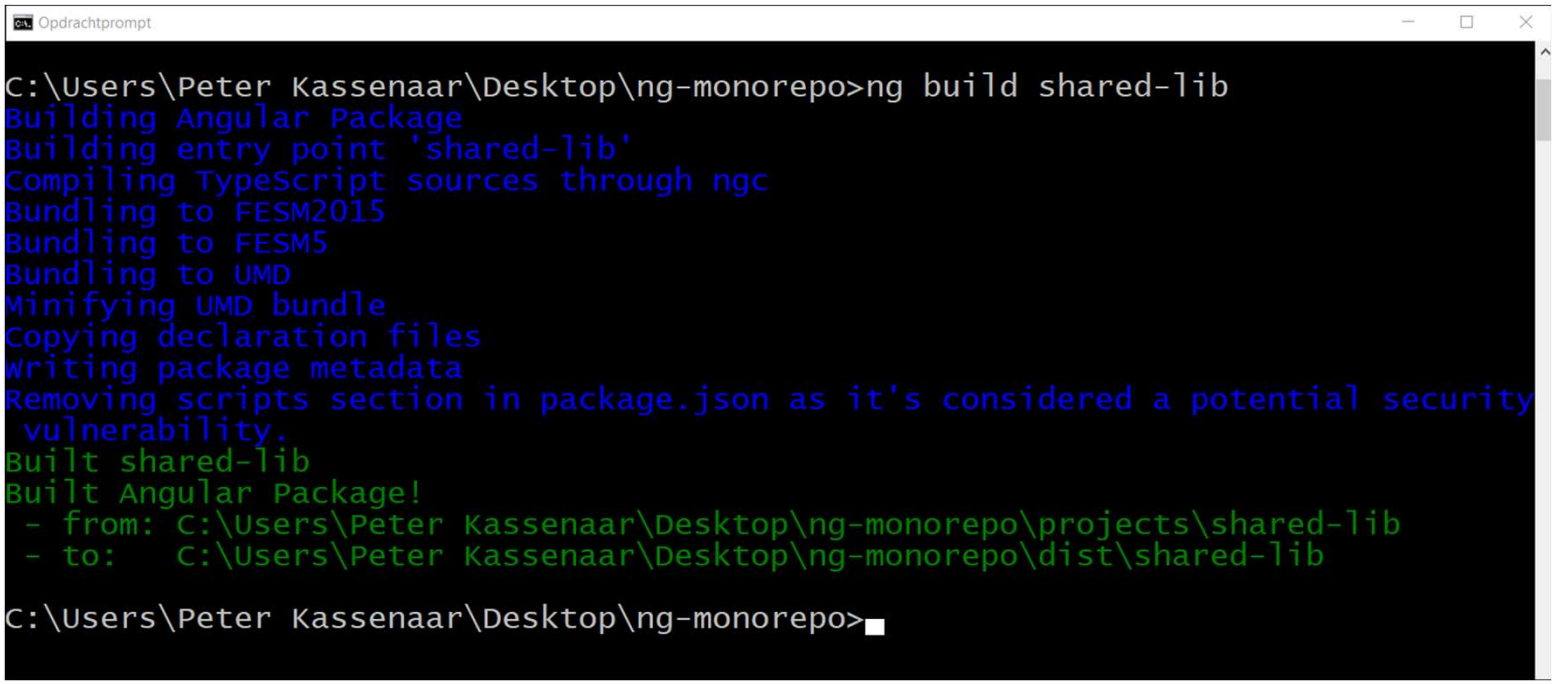
- Update the `public_api.ts` file to make the class available
  - Don't forget!

```
/*
 * Public API Surface of shared-lib
 */

export * from './lib/shared-lib.service';
export * from './lib/shared-lib.component';
export * from './lib/shared-lib.module';
export * from './lib/test/test.component';
```

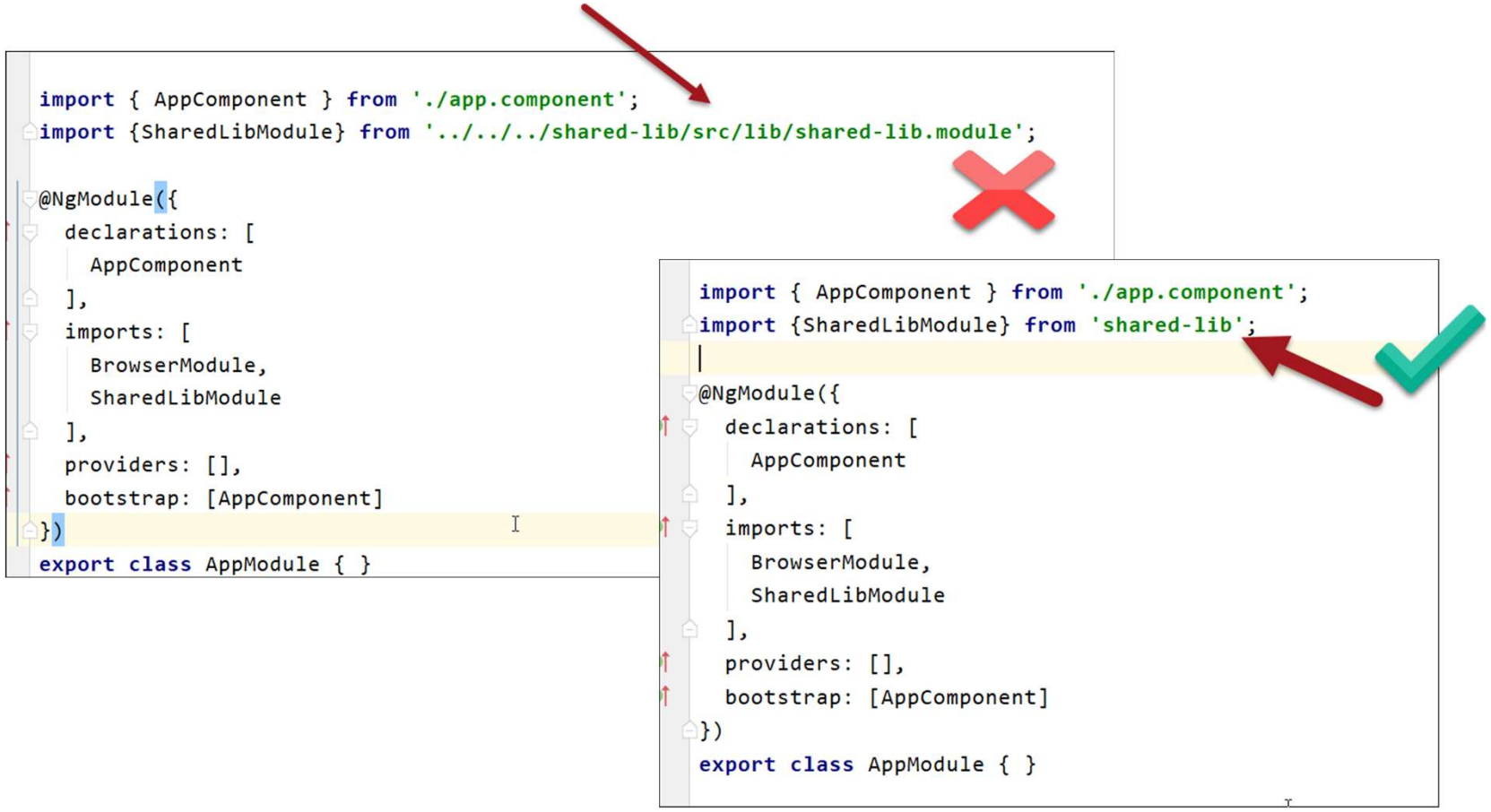
Build the library

- In order to use the component(s) from a shared library, it must be build
  - `ng build shared-lib`
  - A `\dist` folder is created
  - This folder is already mentioned in `tsconfig.json`. Verify this!



Using the library in the application

- Import the library module in the application `app.module.ts`
  - in our example: `dashboard.module.ts`
  - Remove the path your IDE might add to the AutoImport



```
import { AppComponent } from './app.component';
import {SharedLibModule} from '../../../shared-lib/src/lib/shared-lib.module';

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

```
import { AppComponent } from './app.component';
import {SharedLibModule} from 'shared-lib';

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

Using the shared component

- Use the selector of the component from the shared library as normal

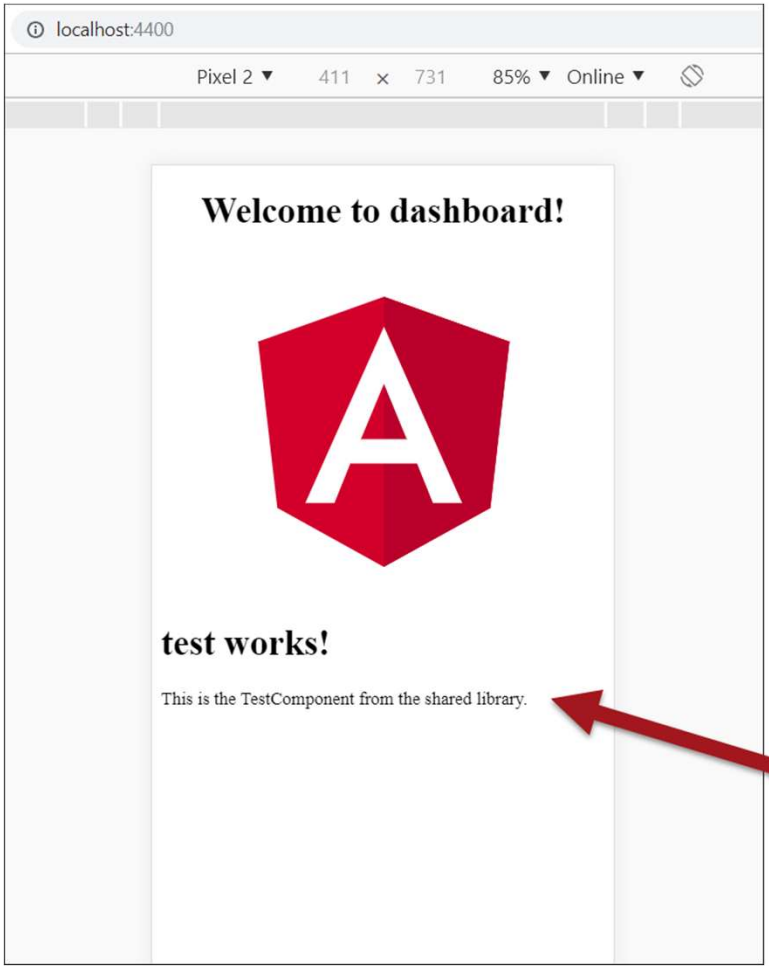
```
<div style="text-align:center">
   …
</div>
<sl-test></sl-test>
```

Running the application

- Use the `ng serve` command
  - Use the `--project=<application-name>` to open the correct project
  - You can use additional flags as needed

- If you update your library contents, you need to rebuild it!
  - Write a script for that. For example

```
"scripts": {
   …
  "build_lib": "ng build shared-lib"
},
```



Exporting a service

- Create a service the regular way
  - `ng generate service <service-name> --project=<project-name>`
- You need to make sure to export your service from the module
- But is only needs to be loaded once!
  - Use a `.forRoot()` method on the module

Creating a `.forRoot()`

```
@NgModule({
  declarations: [SharedLibComponent, TestComponent],
  imports: [
  ],
  exports: [SharedLibComponent, TestComponent]
})
export class SharedLibModule {
  static forRoot(): ModuleWithProviders {
    return {
      ngModule: SharedLibModule,
      providers: [ UserService ]
    };
  }
}
```

- Remember to export the service from `public_api.ts`
- Remember to rebuild the library

Using the shared service

- In your application, update the module to use `SharedLibModule.forRoot()`
- Inject the Service in the component where you want to use it
- Use the servicemethods as normal

```
@NgModule({
  …
  imports: [
    SharedLibModule.forRoot()
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Component

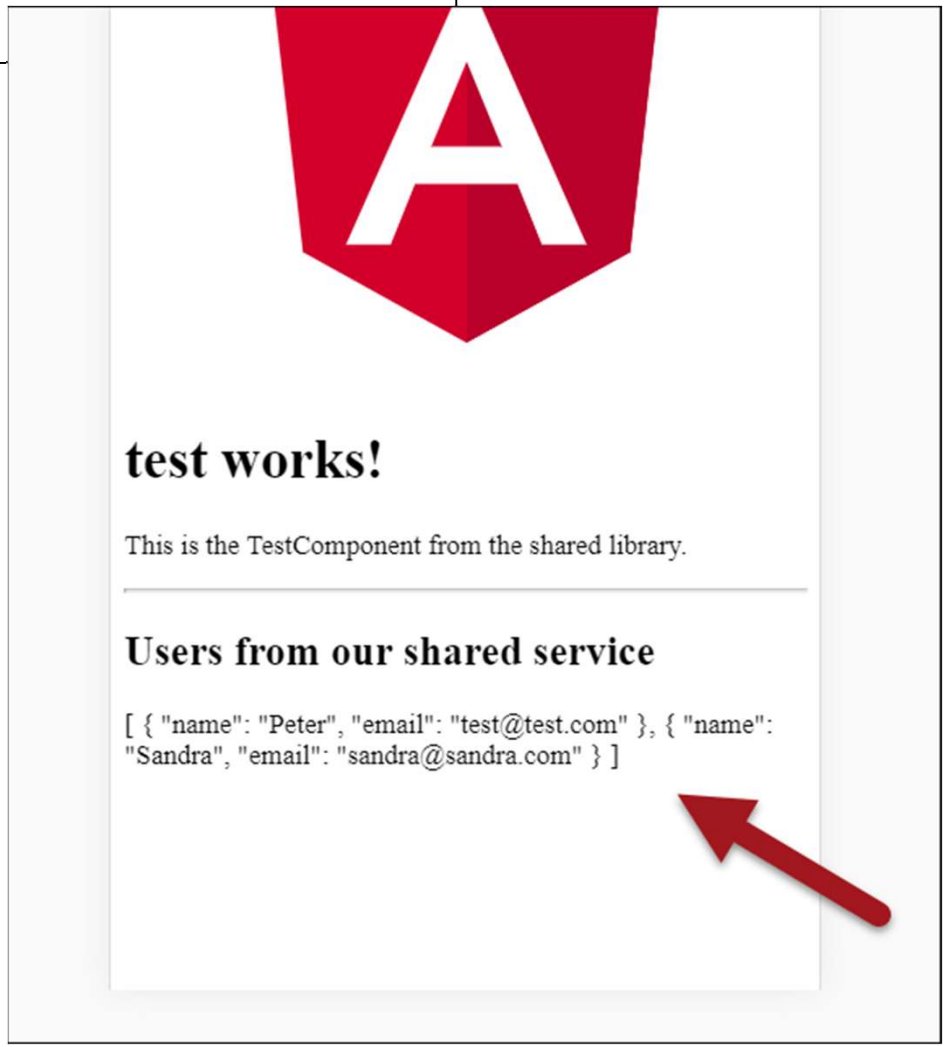
```
import {Component, OnInit} from '@angular/core';
import {User, UserService} from 'shared-lib';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  title = 'dashboard';
  users: User[];

  constructor(private userService: UserService) {
  }

  ngOnInit(): void {
    this.users = this.userService.getUsers();
  }
}
```

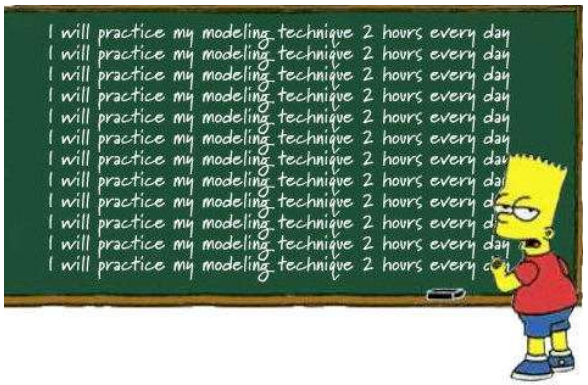
```
<hr>
<h2>Users from our shared service</h2>
{{ users | json }}
```



Workshop

- Create a new application inside the Monorepo
- Import the shared library
- Use/show the `<sl-test>` component from the shared lib inside your new project
- Build and run the new project
- Optional: create a new shared component in the lib.
  - Export / use the new component inside your project
  - Use the exported shared service from the library

Example: github.com/PeterKassenaar/ng-monorepo



Verdict on monorepo

- PRO:
  - Same version of Angular
  - Clear responsibilities per project/application
  - Share code and share services
- CON
  - Harder to use shared state / store
  - All the pro's, are actually also cons! (depending on your project)
  - Checkout: you always get the complete monorepo- unless working with *Github subtrees*

More info

Info on npm packages, monorepo’s and micro apps

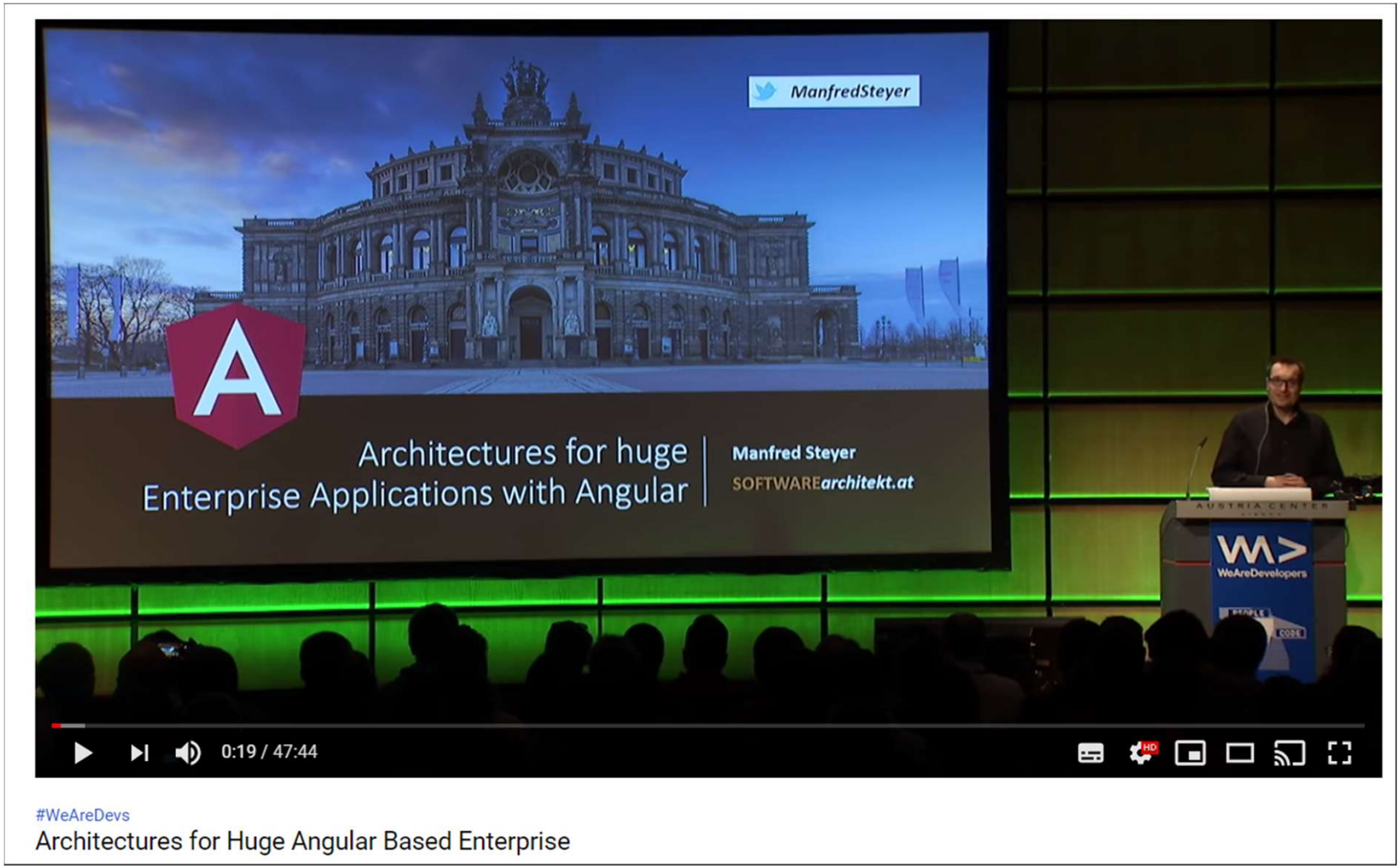
Info on...

NPM packages

Monorepo

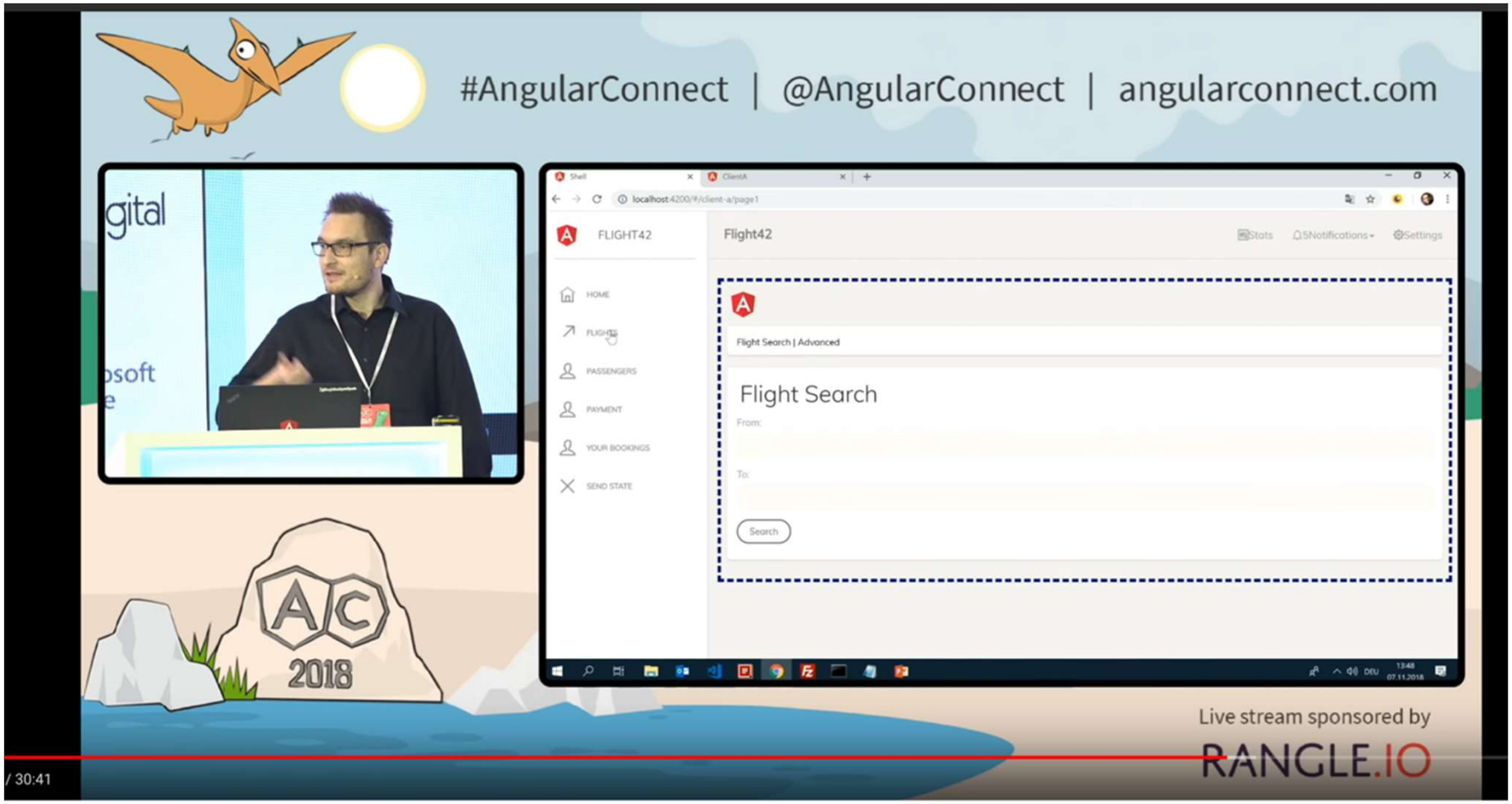
Micro apps

Talks on Angular Monorepo's



- https://www.youtube.com/watch?v=q4XmAy6_ucw

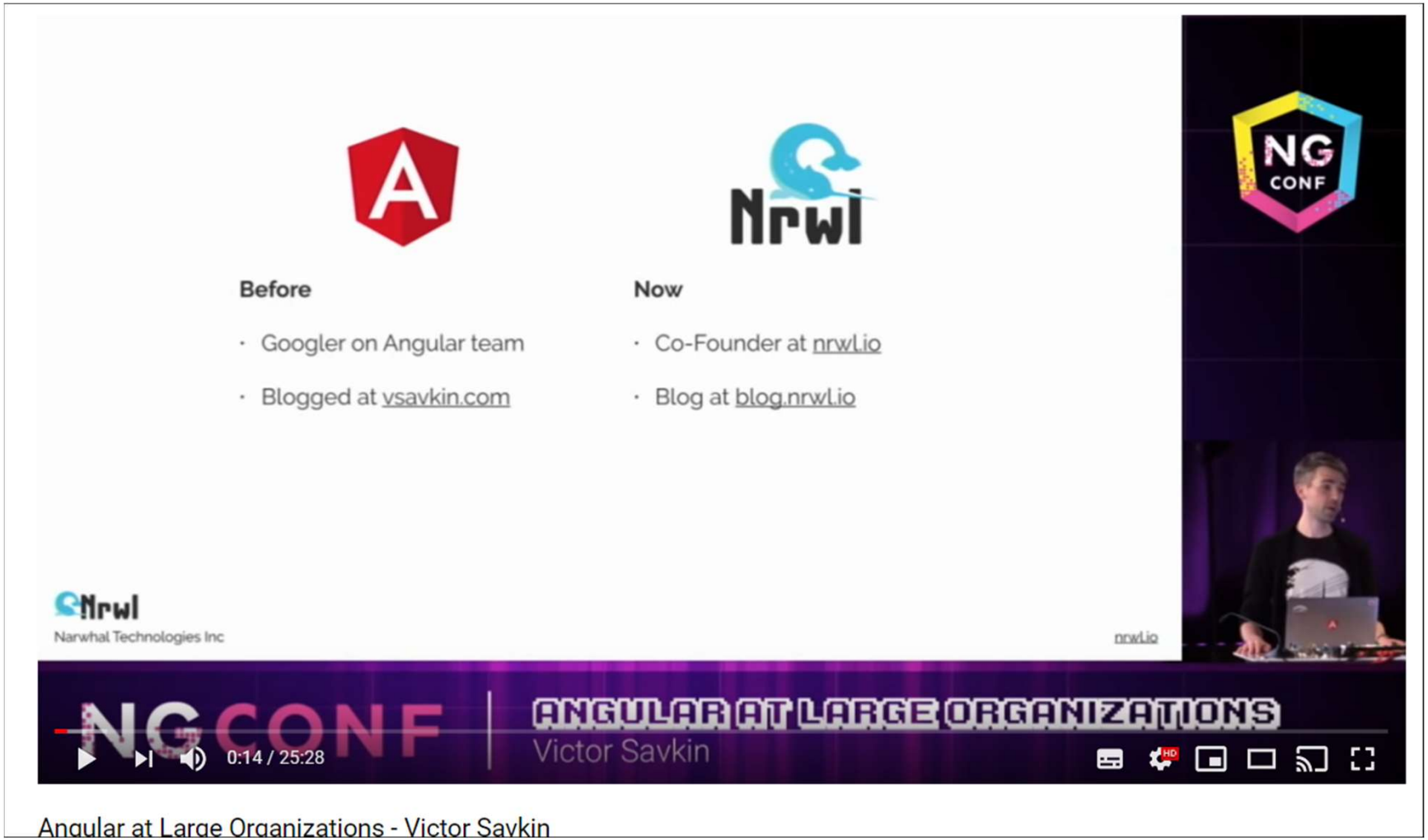
Manfred Steyer – Angular Connect



https://www.youtube.com/watch?v=YU-fMRs-ZYU

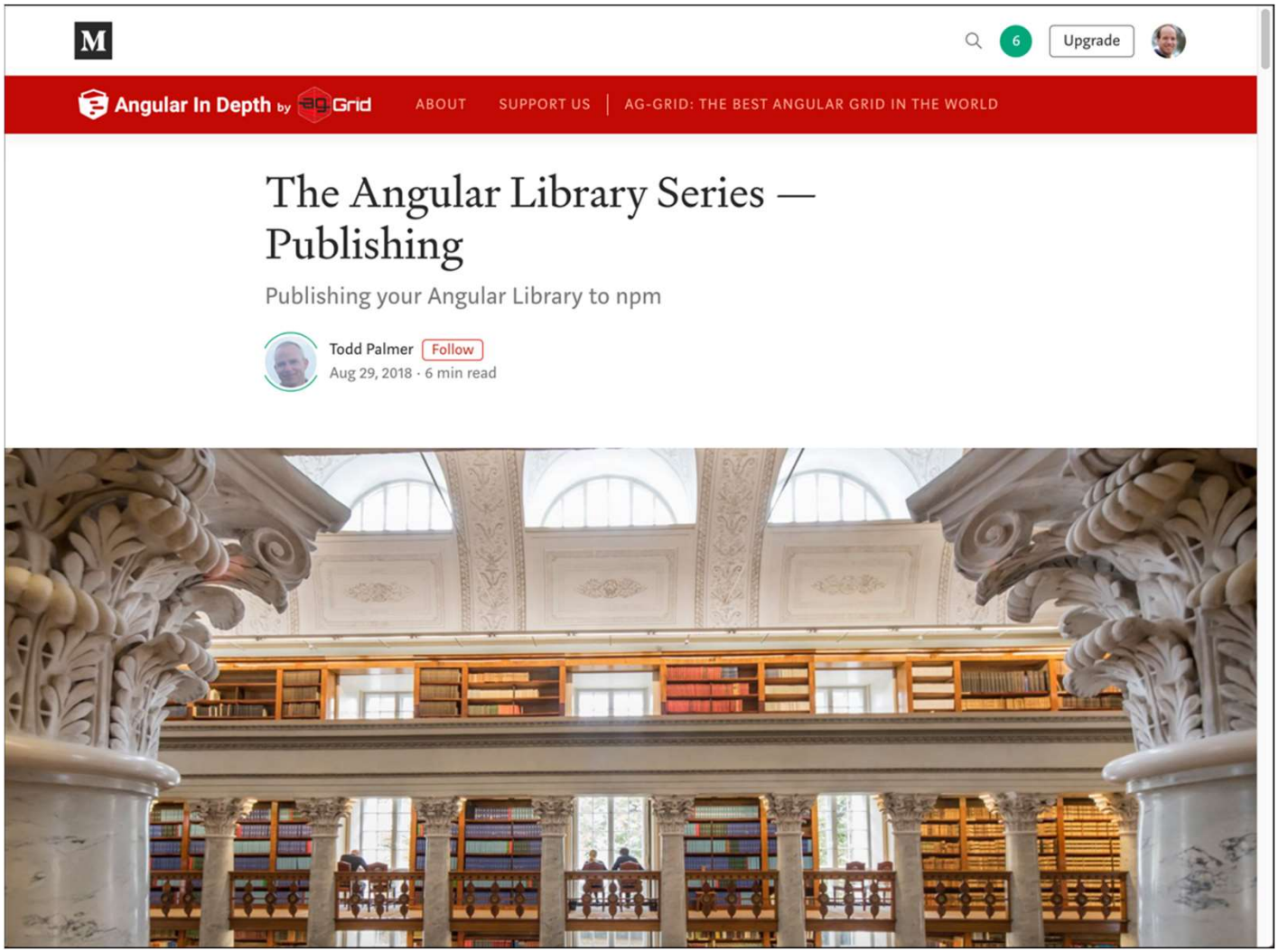
Code: https://github.com/PeterKassenaar/angular-microapp

Victor Savkin – creator of Nx



- https://www.youtube.com/watch?v=piQ0EZhtus0

Publishing your library to npm



https://blog.angularindepth.com/the-angular-library-series-publishing-ce24bb673275

Implementing micro apps in Angular

https://medium.com/bb-tutorials-and-thoughts/how-to-implement-micro-frontend-architecture-with-angular-e6828a0a049c