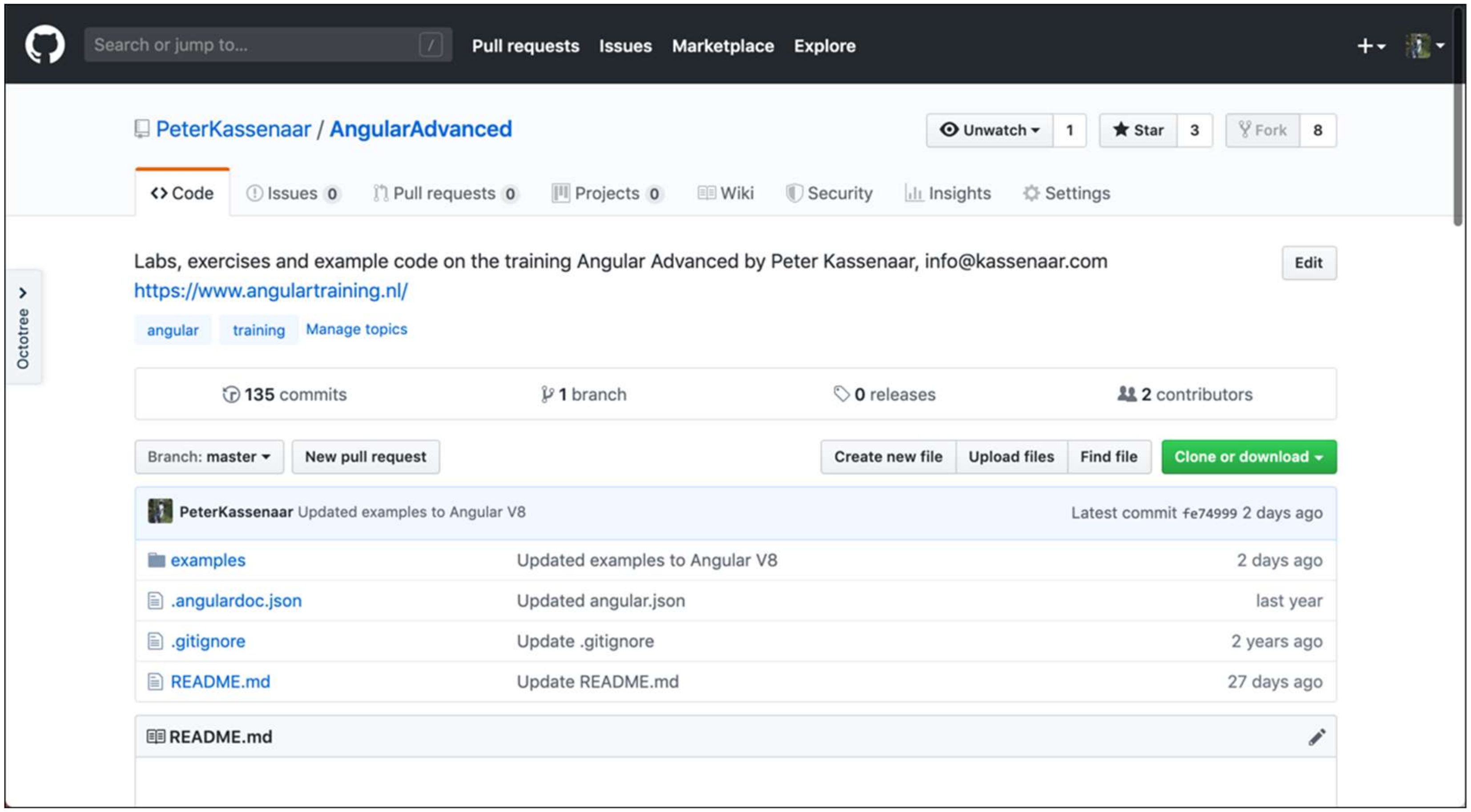
Angular Advanced Introduction, Architecture

Peter Kassenaar
info@kassenaar.com

Generic 'Advanced' Github repo



PeterKassenaar / AngularAdvanced

Labs, exercises and example code on the training Angular Advanced by Peter Kassenaar, info@kassenaar.com
https://www.angulartraining.nl/

angular training Manage topics

135 commits | 1 branch | 0 releases | 2 contributors

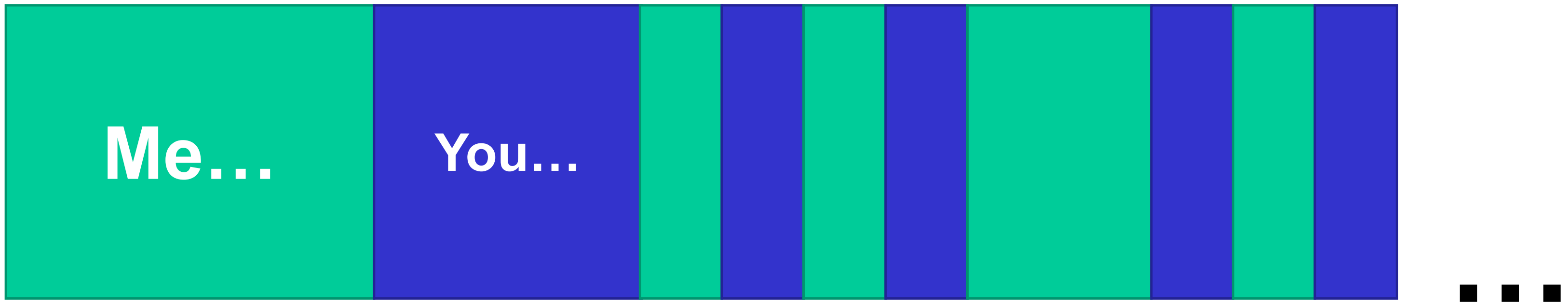
PeterKassenaar Updated examples to Angular V8 — Latest commit fe74999 2 days ago

examples	Updated examples to Angular V8	2 days ago
.angulardoc.json	Updated angular.json	last year
.gitignore	Update .gitignore	2 years ago
README.md	Update README.md	27 days ago

README.md

https://github.com/PeterKassenaar/AngularAdvanced

Overall process

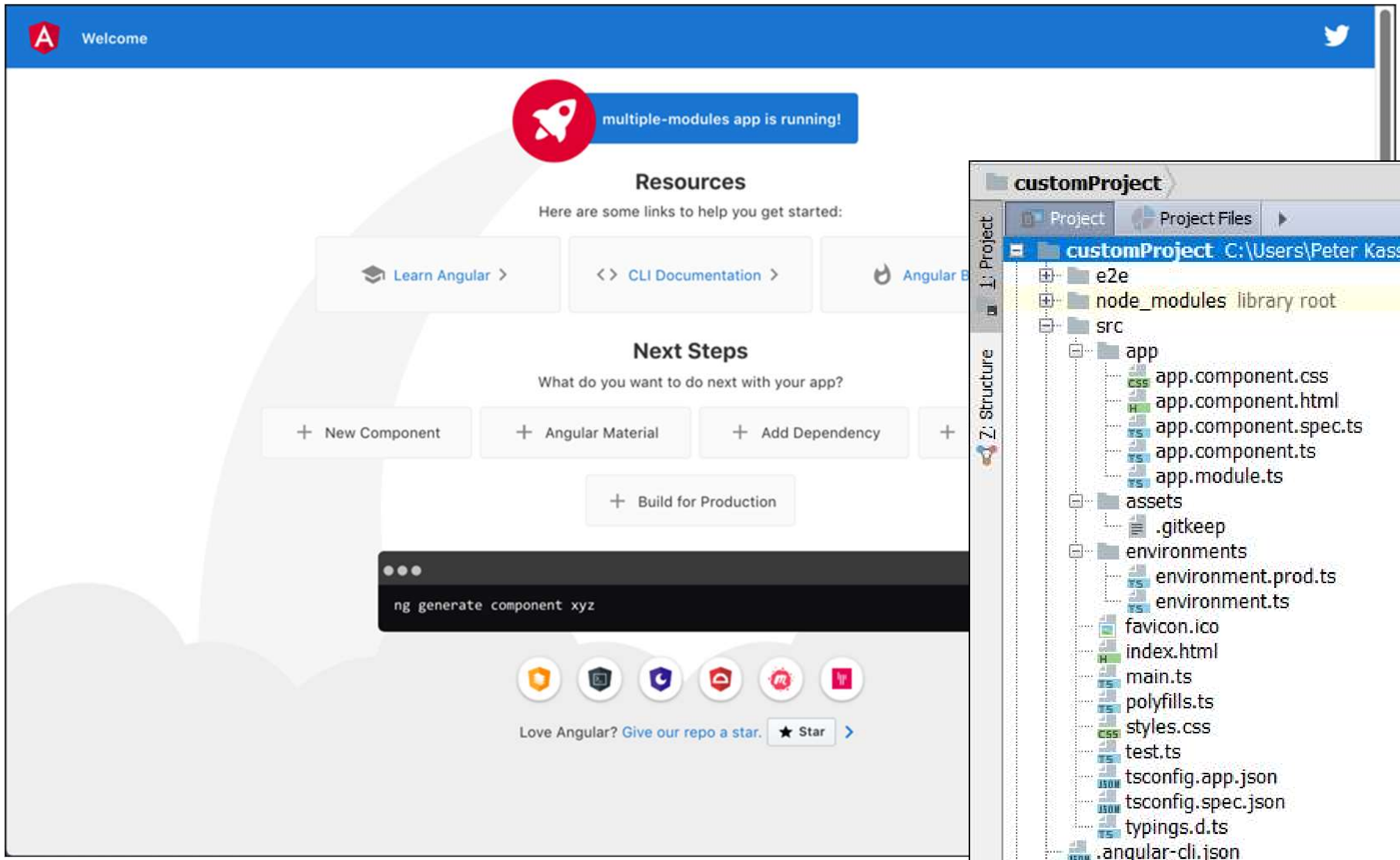


Questions?

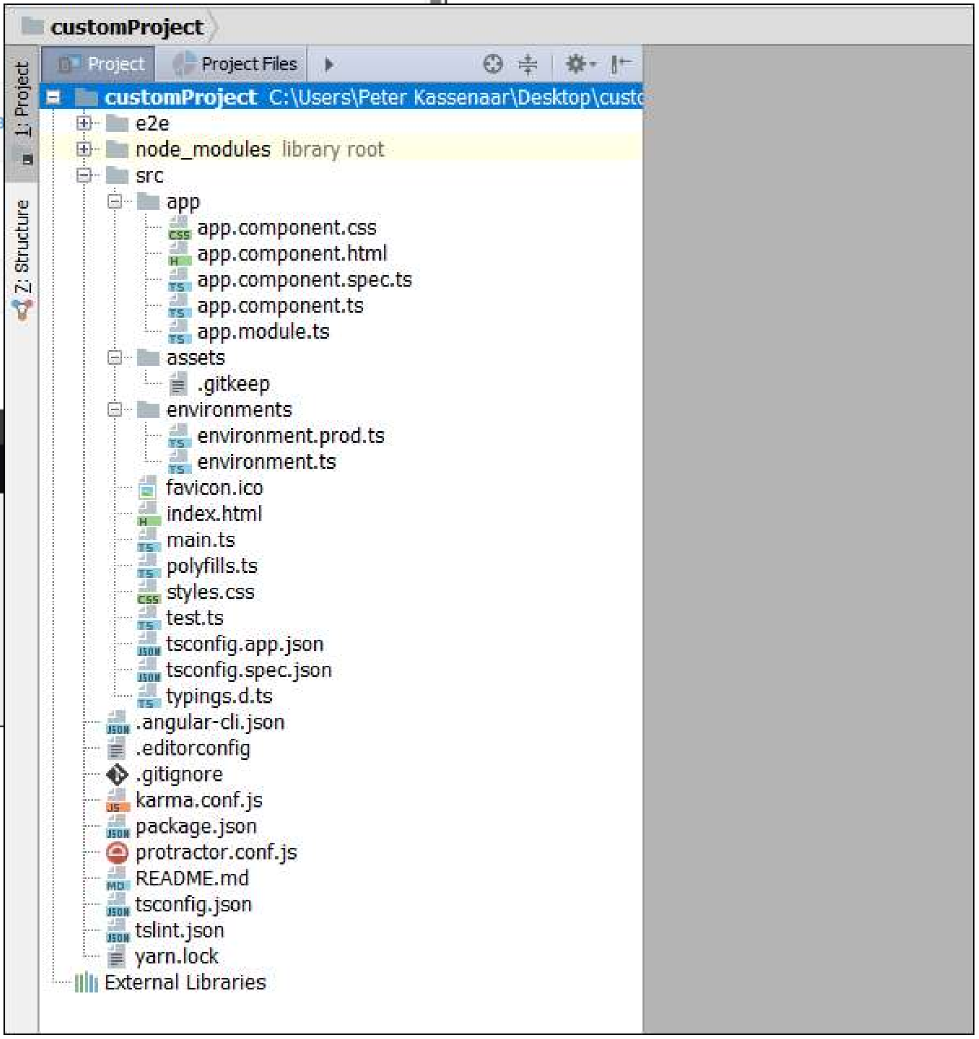
Multiple modules

Splitting your application into separate, reusable modules

Default application – 1 module


Welcome
multiple-modules app is running!
Resources
Here are some links to help you get started:
Learn Angular >
<> CLI Documentation >
Angular B
Next Steps
What do you want to do next with your app?
+ New Component
+ Angular Material
+ Add Dependency
+
+ Build for Production
Love Angular? Give our repo a star.
★ Star

```
ng generate component xyz
```



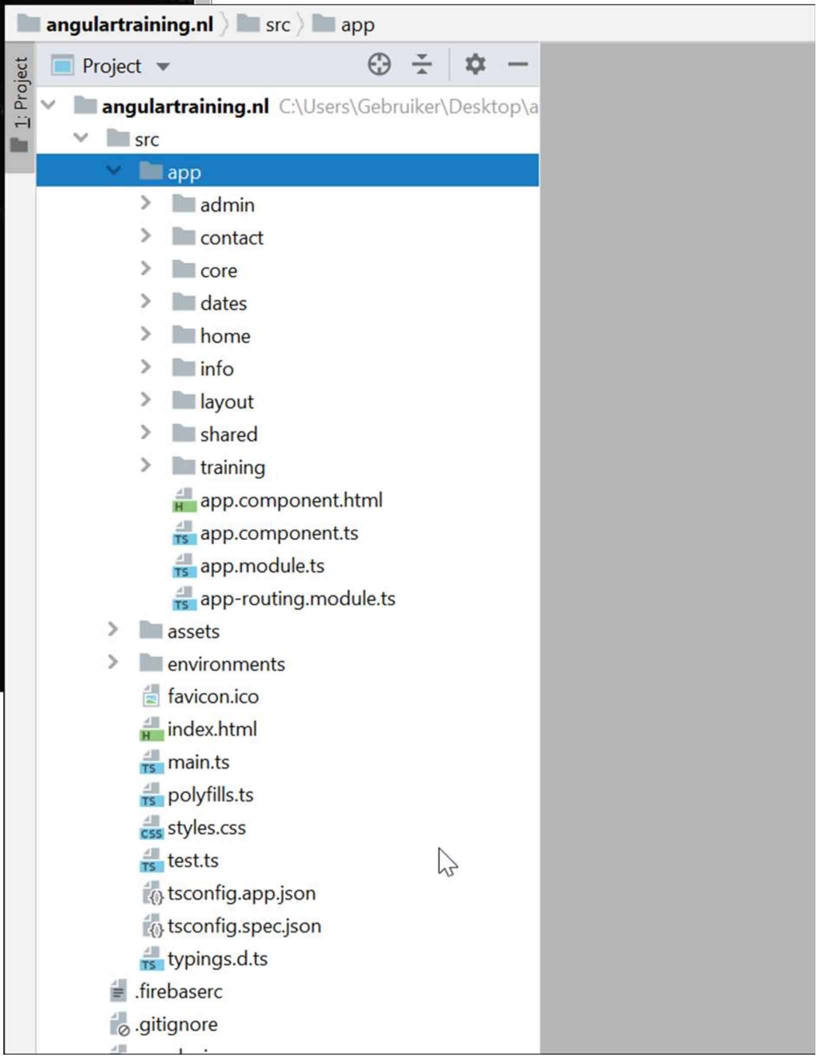
```
customProject  C:\Users\Peter Kassenaar\Desktop\custo
  e2e
  node_modules  library root
  src
    app
      app.component.css
      app.component.html
      app.component.spec.ts
      app.component.ts
      app.module.ts
    assets
      .gitkeep
    environments
      environment.prod.ts
      environment.ts
    favicon.ico
    index.html
    main.ts
    polyfills.ts
    styles.css
    test.ts
    tsconfig.app.json
    tsconfig.spec.json
    typings.d.ts
  .angular-cli.json
  .editorconfig
  .gitignore
  karma.conf.js
  package.json
  protractor.conf.js
  README.md
  tsconfig.json
  tslint.json
  yarn.lock
External Libraries
```

(228 MB)

Bigger applications – multiple modules

World-class Angular training in Dutch and English

Live classrooms - focused on today's developers

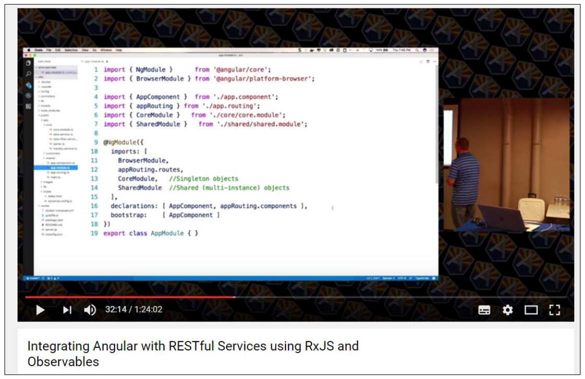


```
angulartraining.nl 〉 src 〉 app

angulartraining.nl  C:\Users\Gebruiker\Desktop\a
  src
    app
      admin
      contact
      core
      dates
      home
      info
      layout
      shared
      training
      app.component.html
      app.component.ts
      app.module.ts
      app-routing.module.ts
    assets
    environments
    favicon.ico
    index.html
    main.ts
    polyfills.ts
    styles.css
    test.ts
    tsconfig.app.json
    tsconfig.spec.json
    typings.d.ts
  .firebaserc
  .gitignore
```

Angular Modules - naming

- Divide your app into *logical* and often *reusable* pieces of code
- Keyword : code organization

- Use one `AppModule` - the root of your app
- Use one `CoreModule` - containing all *singletons* in your app
- Use one `SharedModule` - containing all shared resources, possible multiple instances
- Use additional modules *per feature*
- https://www.youtube.com/watch?v=YxK4UW4UfCk

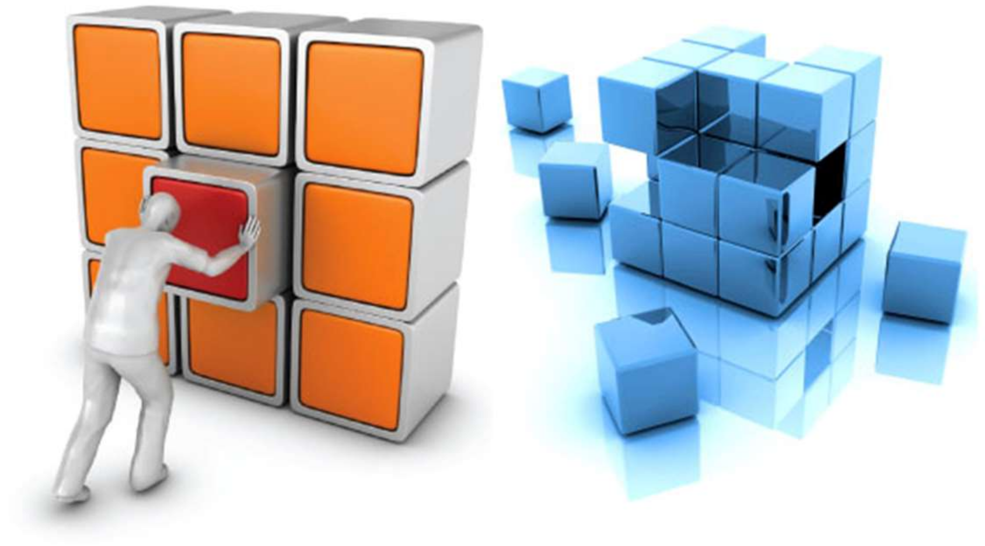


```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }  from './app.component';
import { appRouting } from './app.routing';
import { CoreModule }   from './core/core.module';
import { SharedModule }   from './shared/shared.module';

@NgModule({
  imports: [
    BrowserModule,
    appRouting.routes,
    CoreModule,   //Singleton objects
    SharedModule  //Shared (multi-instance) objects
  ],
  declarations: [ AppComponent, appRouting.components ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Integrating Angular with RESTful Services using RxJS and Observables

Application – multiple Modules – why?

- *Reuse* of Components, Pipes, Routes and Services etc. over different apps
- *Wrap* each set of logical related components, services, etc. in its own module.

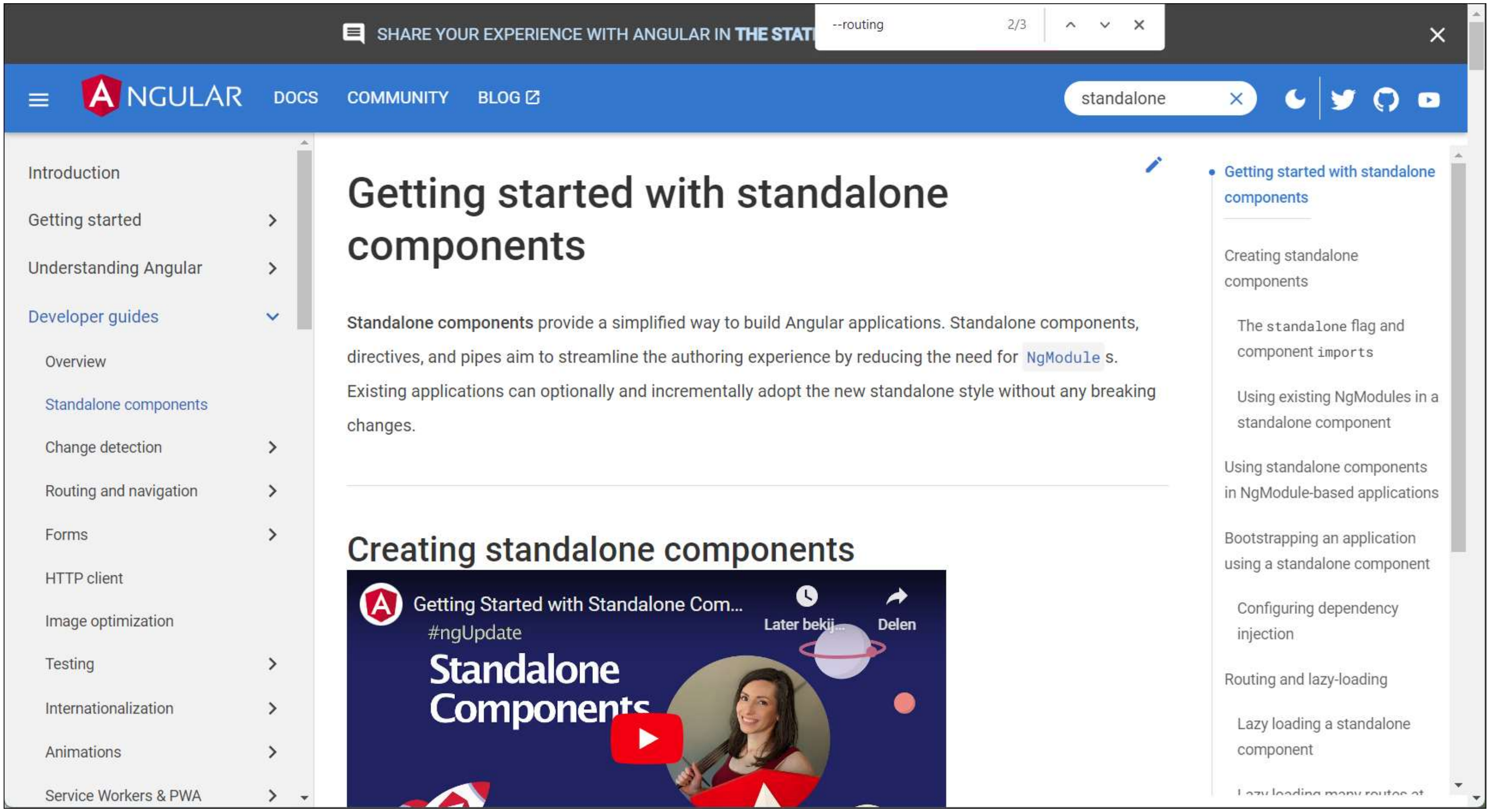


Since Angular 15 – Standalone Components

- Applications and components without an `ngModule`

"Standalone components provide a simplified way to build Angular applications. Standalone components, directives, and pipes aim to streamline the authoring experience by reducing the need for `NgModules`. Existing applications can optionally and incrementally adopt the new standalone style without any breaking changes."

More info on standalone components



SHARE YOUR EXPERIENCE WITH ANGULAR IN THE STAT

--routing 2/3

NGULAR DOCS COMMUNITY BLOG

standalone

Introduction
Getting started
Understanding Angular
Developer guides
Overview
Standalone components
Change detection
Routing and navigation
Forms
HTTP client
Image optimization
Testing
Internationalization
Animations
Service Workers & PWA

Getting started with standalone components

Standalone components provide a simplified way to build Angular applications. Standalone components, directives, and pipes aim to streamline the authoring experience by reducing the need for `NgModule` s. Existing applications can optionally and incrementally adopt the new standalone style without any breaking changes.

Creating standalone components

- Getting started with standalone components
- Creating standalone components
  - The `standalone` flag and component `imports`
  - Using existing NgModules in a standalone component
- Using standalone components in NgModule-based applications
- Bootstrapping an application using a standalone component
  - Configuring dependency injection
- Routing and lazy-loading
  - Lazy loading a standalone component

https://angular.io/guide/standalone-components

Verdict (personal, opinion!)

We’re NOT using standalone components in this course

- Standalone components might come in handy, in new applications from scratch
- Due to confusion, personally I don’t see it implemented *‘optionally and incrementally’* in existing apps
- It is mostly done (IMO) to mimic Vue, React et all.
- otherwise – little advantages, IMO if you already know Angular

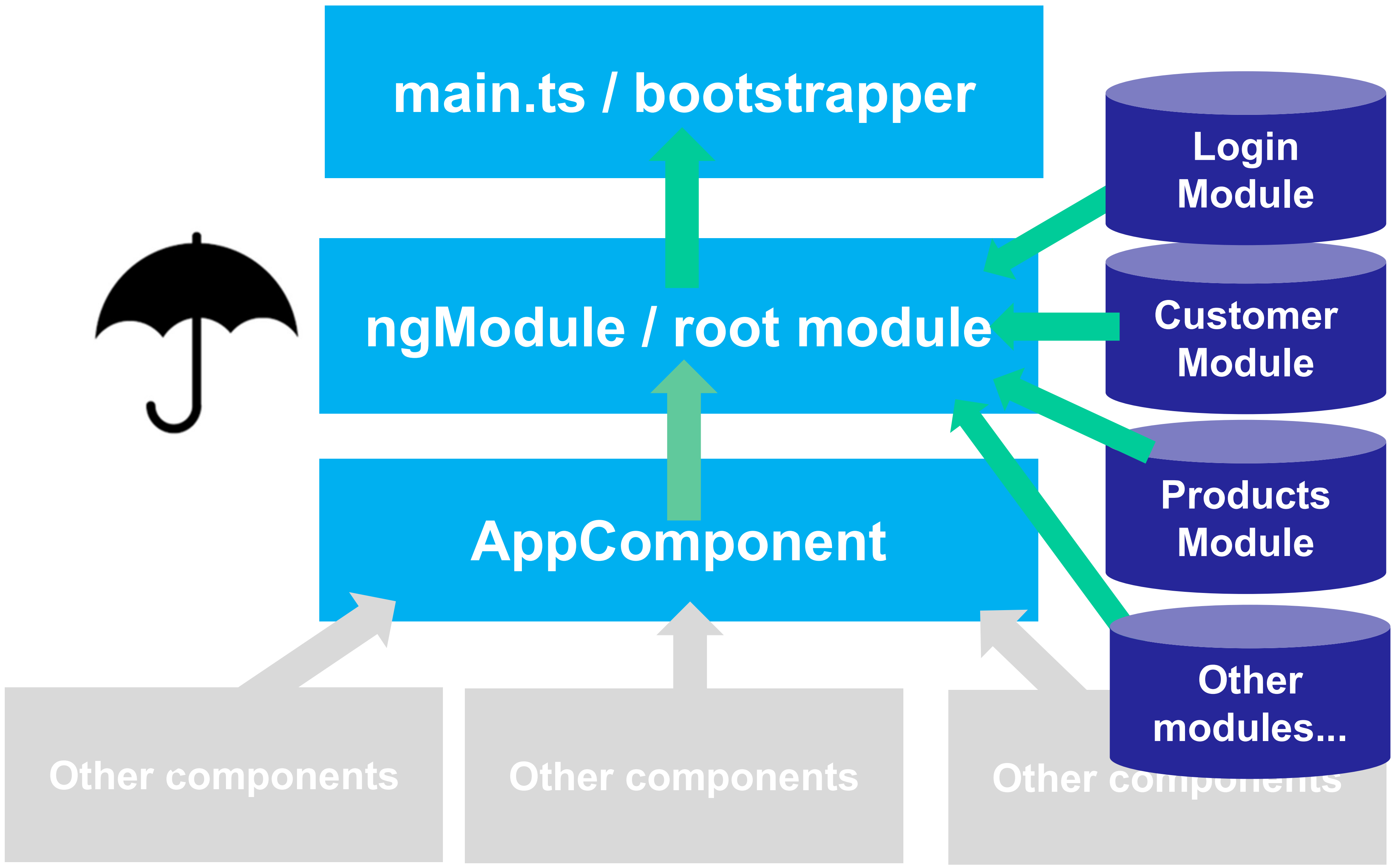
But, if you want more information…

- https://blog.angular.io/angular-v15-is-now-available-df7be7f2f4c8
- https://www.angulararchitects.io/aktuelles/angulars-future-without-ngmodules-lightweight-solutions-on-top-of-standalone-components/
- https://netbasal.com/angular-standalone-components-welcome-to-a-world-without-ngmodule-abd3963e89c5

Angular's Future Without NgModules – Part 1: Lightweight Solutions Using Standalone Components

Standalone Components make the future of Angular applications more lightweight. We don't need NgModules anymore. Instead, we just use EcmaScript modules.

Article by Manfred Steyer, GDE | 01.05.2022 | share


main.ts / bootstrapper
ngModule / root module
AppComponent
Login Module
Customer Module
Products Module
Other modules...
Other components
Other components
Other components

Steps

1. Create a new module
    - Optional: test first with `--dry-run`
    - `ng generate module customers --dry-run`
2. Create component(s) inside that module
    - Again: test first with `--dry-run`
    - `ng generate component customers --module customers --dry-run`
3. Apply UI, logic, etc. to your component
4. Export your component inside `customers.module.ts`
    - `exports : [CustomersComponent],`
    - Otherwise it can’t be used in other components!
5. Provide new module to `app.module.ts`
    - `imports: [CustomersModule]`

Optional : SharedModule

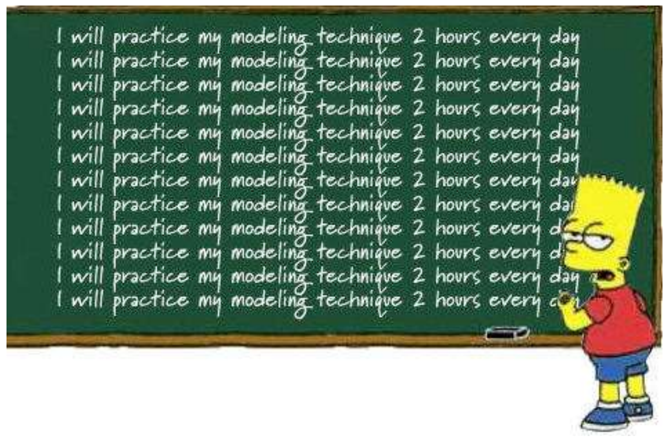
- Reuse components in multiple modules? Use a `SharedModule`
  - `ng g m shared` – shorthand notation
- Create components inside `SharedModule`
- Import `SharedModule` in other modules
- It doesn't have to be in `AppModule` if you don't use it directly!
- It *does* not add size to module bundles

Demo

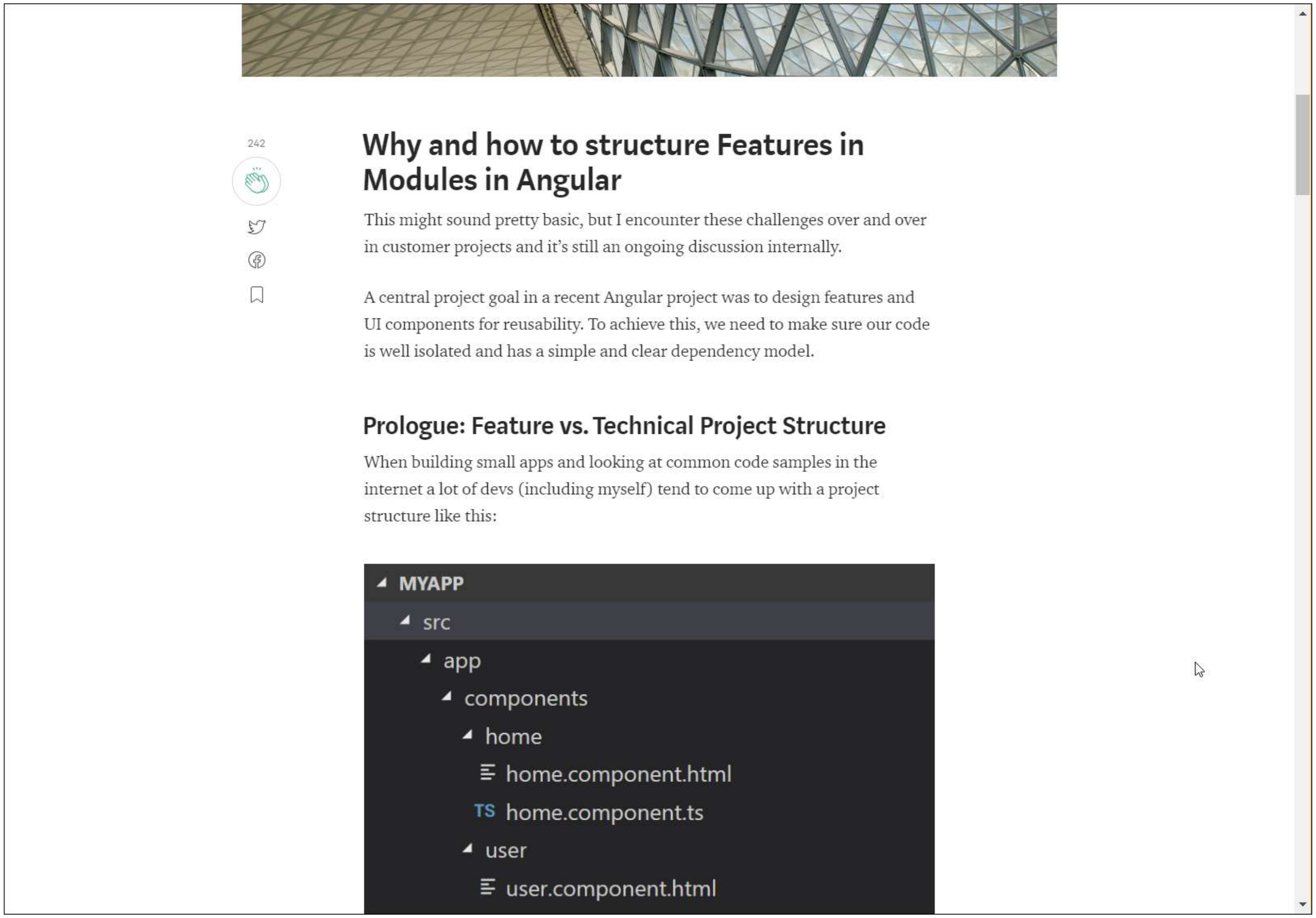
Workshop

- Open `../100-multiple modules.` (`npm install`, `npm start`)
- Create a new module
- Create a new component inside this new module and give it some UI.
- Include the module in the Main Module and show it besides other modules
- Include the Search Component in your own module

- *OR:*
- Add Multiple Modules from scratch to your own application, using the steps described in this module.



How to structure feature modules



242

Why and how to structure Features in Modules in Angular

This might sound pretty basic, but I encounter these challenges over and over in customer projects and it's still an ongoing discussion internally.

A central project goal in a recent Angular project was to design features and UI components for reusability. To achieve this, we need to make sure our code is well isolated and has a simple and clear dependency model.

Prologue: Feature vs. Technical Project Structure

When building small apps and looking at common code samples in the internet a lot of devs (including myself) tend to come up with a project structure like this:

```
MYAPP
  src
    app
      components
        home
          home.component.html
          home.component.ts
        user
          user.component.html
```

https://medium.com/@philippbauknecht/why-and-how-to-structure-features-in-modules-in-angular-d5602c6436be