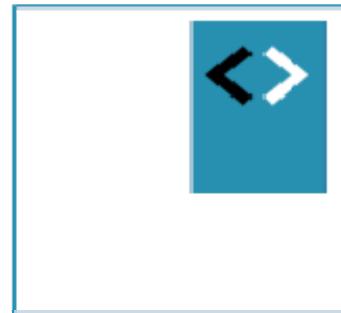




Angular 2

Module 10 – Multiple Modules



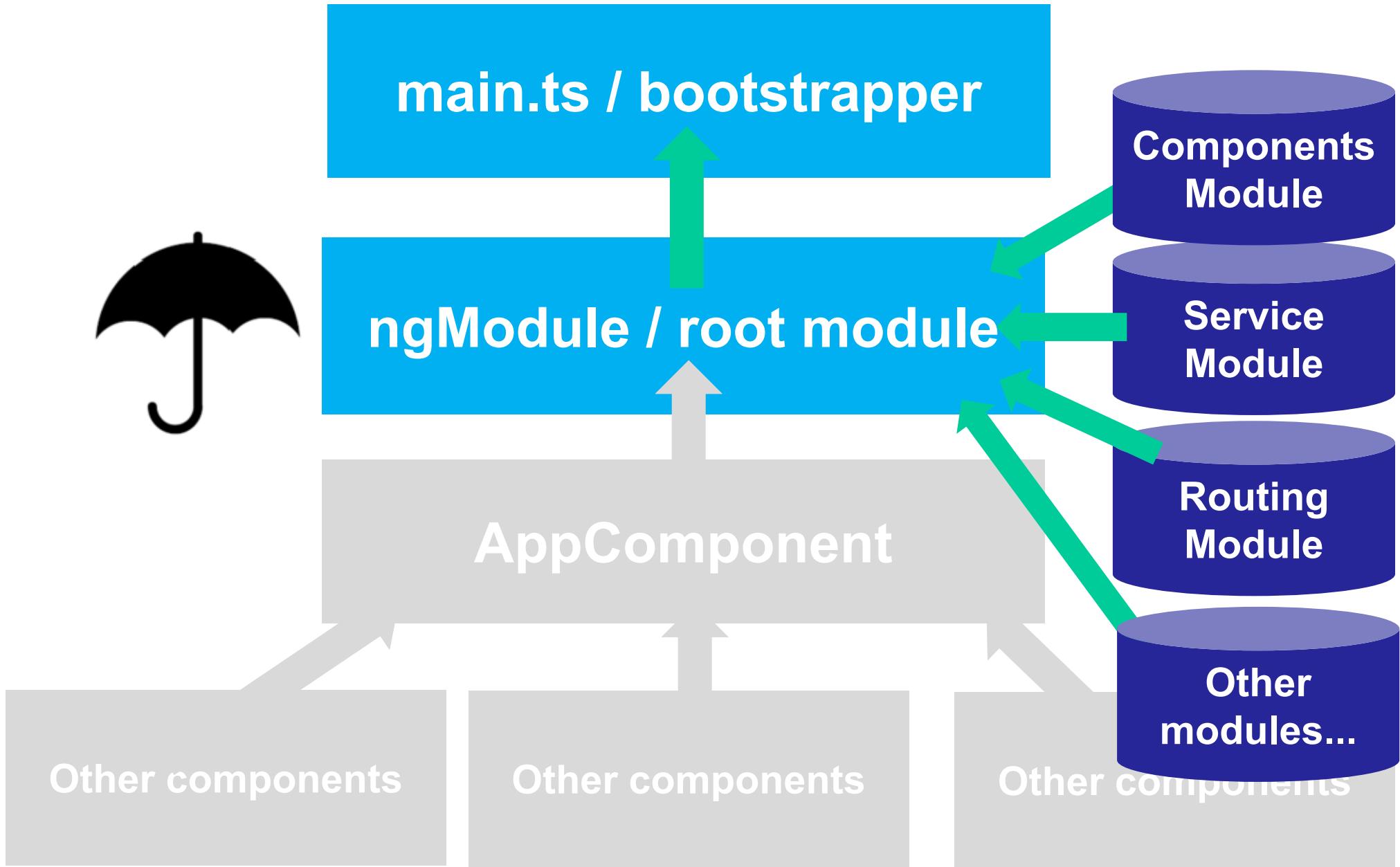
Peter Kassenaar –
info@kassenaar.com

Modules

- Introduced in Angular 2-rc.5
- Successor of Angular 1 `angular.module('myApp', [...]).`
- Divide your app into *logical* and often *reusable* pieces of code
- Keyword : code organization
- Recommendation (as of January 2017) by John Papa/Dan Wahlin for larger projects:
 - 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>

Application – multiple Modules

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



This example:

- *Layer based* modules
 - Combine routes, combine services, combine components, and so on.
- In Real Life Applications: *Feature based* modules (we'll discuss that later)
 - Place all components that belong to the same subject in their own module

Example 1: Routing in its own module

```
// app.routing.module.ts
// Routing in its own module - now imported in main module app.module.ts

// 1. Import Router stuff
import {NgModule} from '@angular/core';
import {RouterModule, Routes} from '@angular/router';

// 2. Import Components
import {AppComponent} from "./app.component";
import ...

// 3. Routing table
const AppRoutes: Routes = [...]

// 4. Declare *new* NgModule!
@NgModule({
  imports: [RouterModule.forRoot(AppRoutes)],
  exports: [RouterModule]
})
export class AppRoutingModule {
}
export const routingComponents = [
  AppComponent,
  CityAddComponent,
  CityDetailComponent,
  CanDeactivateComponent
];
```

Import all stuff needed by this module

Define routes

Create new Module. Import & Export enhanced RouterModule

Export Components to be routed to. No need to re-import in main module

Edit Main Module

```
// app.module.ts
// 1. Import common Angular stuff
import {NgModule}      from '@angular/core';

// 2. Routing - in its own module
import {AppRoutingModule, routingComponents} from './app.routing.module';

// 3. Components for the app. Only the MainComponent remains.
import {MainComponent} from './MainComponent';

@NgModule({
  imports     : [
    BrowserModule,
    ...
    AppRoutingModule, // ALL routes inside their own module
  ],
  declarations: [
    MainComponent,
    // Components are now bundled in the routing module
    routingComponents
  ],
  ...
})
export class AppModule {
```

Import New Module

Import in Main Module

Declarations are already
exported in Routing Module

Example 2: Services in own module

```
// app.services.module.ts
import {NgModule} from '@angular/core';
import {CommonModule} from '@angular/common';

// 2. Import Services & Guards
import {CityService} from "./city.service";
import ...;

// 3. Declare module. Import and Export CommonModule,
// decorated with providers: [...] array
@NgModule({
  imports : [CommonModule],
  exports : [CommonModule],
  providers: [
    CityService,
    AuthService,
    CanActivateViaAuthGuard,
    CanDeactivateGuard
  ]
})

export class AppServicesModule { }
```



Import and re-export
CommonModule, enhanced with
module-specific stuff. In this
case providers [...]

Edit Main Module

```
// app.module.ts
// Import Services - in its own module
import {AppServicesModule} from "./app.services.module";

import ...

@NgModule({
  imports      : [
    AppRoutingModule, // ALL routes inside their own module
    AppServicesModule, // ALL services inside their own module
  ],
  declarations: [
    MainComponent,
    // Components are now bundled in the routing module
    routingComponents
  ],
  // providers   : [
  //   Not neccesary anymore, as these are defined in their own module
  // ],
  bootstrap    : [
    MainComponent
  ]
})
export class AppModule {
```

No more providers [...], as this
now lives inside the
AppServiceModule

Example 3: Components in own module

```
// Login.module.ts
// 1. Import NgModule stuff
import {NgModule} from '@angular/core';
import {CommonModule} from '@angular/common';

// 2. Import Services & Guards
import {LoginComponent} from "./login.component";

// 3. Declare module. Import and Export CommonModule and components
// inside this module (in this case only the LoginComponent).
@NgModule({
  imports      : [CommonModule],
  exports      : [
    CommonModule,
    LoginComponent
  ],
  declarations: [LoginComponent]
})
export class LoginModule {
```

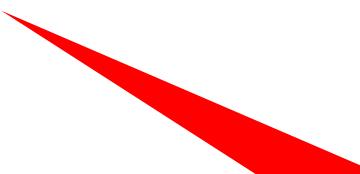


Import and re-export
CommonModule, enhanced with
Components,
see declarations: [...]

Edit Main Module

```
// app.module.ts
...
// 4. Import Login module w/ Login component
import {LoginModule} from './login.module';
...

@NgModule({
  imports      : [
    ...,
    AppRoutingModule, // ALL routes inside their own module
    AppServicesModule, // ALL services inside their own module
    LoginModule        // Login Module
  ],
  ...
  bootstrap    : [
    MainComponent
  ]
})
export class AppModule {
```



Login Module imported. No
need to repeat LoginComponent
in declarations: [...]

More on NgModules

The screenshot shows a blog post titled "Introducing Angular Modules - Root Module" by John Papa. The post discusses the @NgModule decorator and its purpose in organizing applications. The page includes navigation links like HOME, SPEAKING, VIDEOS, ARTICLES, ABOUT, and CONTACT, as well as social sharing icons for Twitter, Facebook, Google+, LinkedIn, and StumbleUpon. The date of the post is September 5, 2016. On the right side, there is a sidebar featuring a bio for John Papa and a "Most Recent" section with links to other posts.

HOME SPEAKING VIDEOS ARTICLES ABOUT CONTACT

Angular Modules

Introducing Angular Modules - Root Module

05 SEPTEMBER 2016

The `@NgModule` is a new decorator that has recently been added in Angular 2. `NgModule` defines an Angular Module, which (from the official docs) are defined as "Angular Modules help organize an application into cohesive blocks of functionality."

John

Hi, I'm John Papa. I author this blog, create courses for Pluralsight and am a Google Developer Expert and Microsoft Regional Director. I speak at events and I train technology thought leaders →

Most Recent

[Introducing Angular Modules - Feature Modules](#)

[Introducing Angular Modules - Routing Module](#)

<https://johnpapa.net/introducing-angular-modules-root-module/>

Official Documentation

The screenshot shows the Angular official documentation website. The top navigation bar includes links for FEATURES, DOCS, EVENTS, NEWS, and GET STARTED. On the left, a sidebar menu lists DOCS HOME, CORE DOCUMENTATION (which is selected), QUICKSTART, GUIDE, API REFERENCE, ADDITIONAL DOCUMENTATION (selected), TUTORIAL, ADVANCED, and several sub-sections under ADVANCED: Angular Modules (NgModule) (selected), Animations, Attribute Directives, Browser support, Component Styles, and Hierarchical Injectors. At the bottom of the sidebar is a link for Angular for TypeScript. The main content area features a large title 'ANGULAR MODULES (NGMODULE)' and a sub-section titled 'Define application modules with @NgModule'. It explains that Angular Modules help organize an application into cohesive blocks of functionality. Below this, it describes what an Angular Module is and how it works. A 'Table of Contents' section at the bottom lists 'Angular modularity'.

SEARCH DOCS...

DOCS HOME

CORE DOCUMENTATION

QUICKSTART

GUIDE

API REFERENCE

ADDITIONAL DOCUMENTATION

TUTORIAL

ADVANCED

Angular Modules (NgModule)

Animations

Attribute Directives

Browser support

Component Styles

Hierarchical Injectors

Angular for TypeScript

FEATURES DOCS EVENTS NEWS GET STARTED

ANGULAR MODULES (NGMODULE)

Define application modules with `@NgModule`

Angular Modules help organize an application into cohesive blocks of functionality.

An Angular Module is a *class* adorned with the `@NgModule` decorator function. `@NgModule` takes a metadata object that tells Angular how to compile and run module code. It identifies the module's *own* components, directives and pipes, making some of them public so external components can use them. It may add service providers to the application dependency injectors. And there are more options covered here.

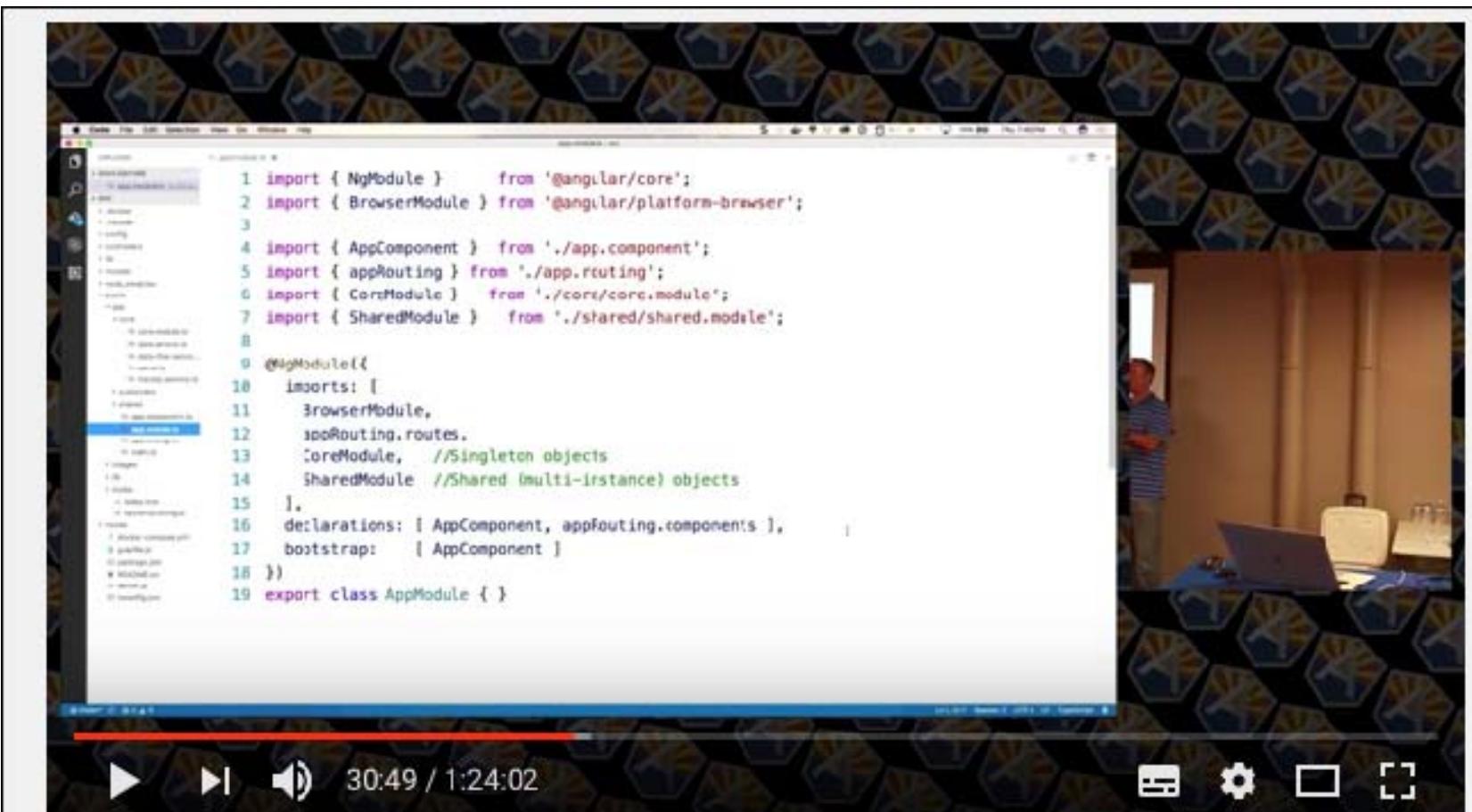
This page explains how to [create `NgModule`](#) classes and how to load them, either immediately when the application launches or later, as needed, via the [Router](#).

Table of Contents

- [Angular modularity](#)

<https://angular.io/docs/ts/latest/guide/ngmodule.html>

Dan Wahlin on core module & shared module



Integrating Angular with RESTful Services using
RxJS and Observables

<https://www.youtube.com/watch?v=YxK4UW4UfCk&t=2035s>