Angular 2
Module 10 – Multiple Modules

Peter Kassenaar –
info@kassenaar.com

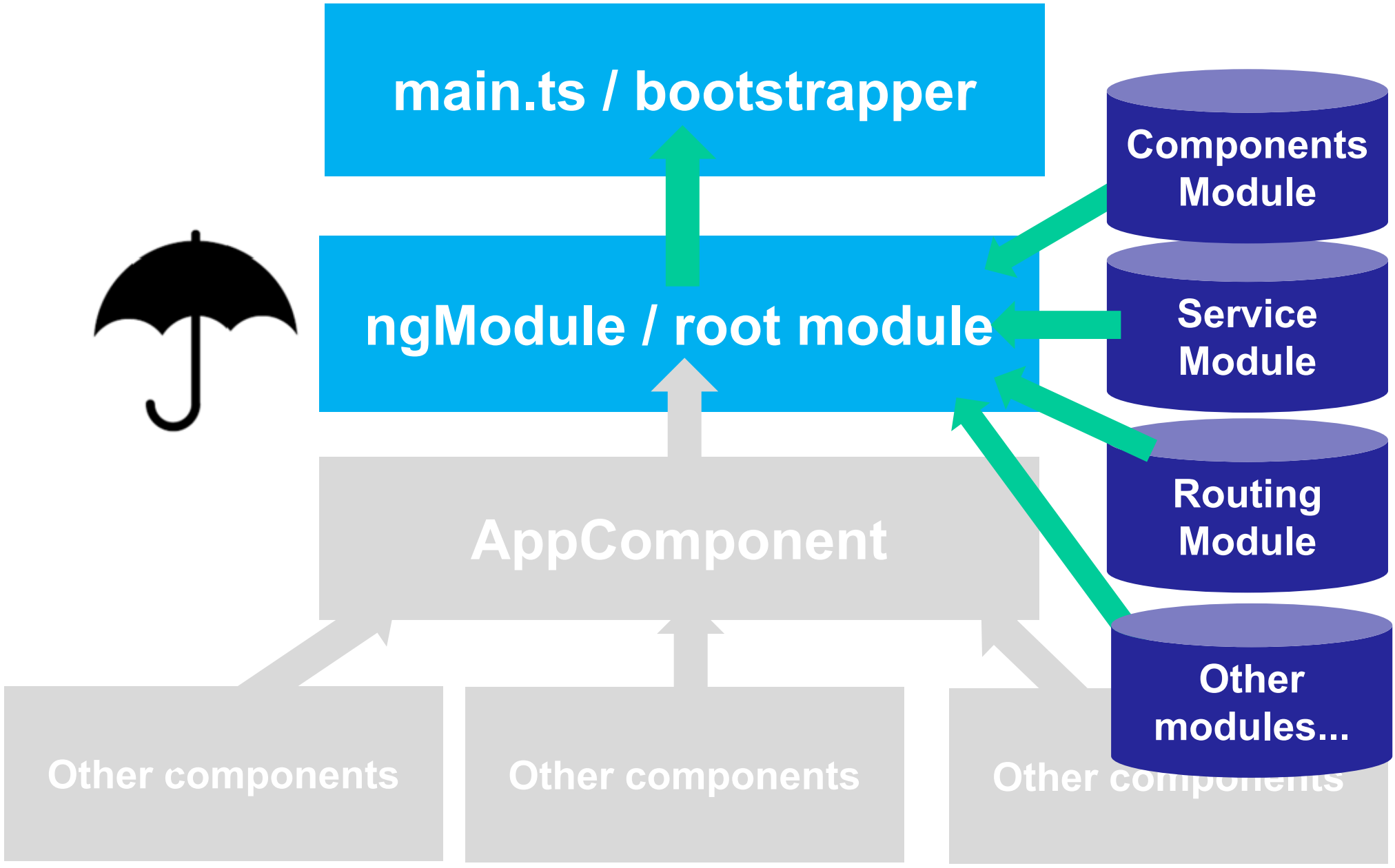
Modules

- 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.


main.ts / bootstrapper
ngModule / root module
AppComponent
Other components
Other components
Other components
Components Module
Service Module
Routing Module
Other modules...

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 it's 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 commmon 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 mod
   ],
   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: […]

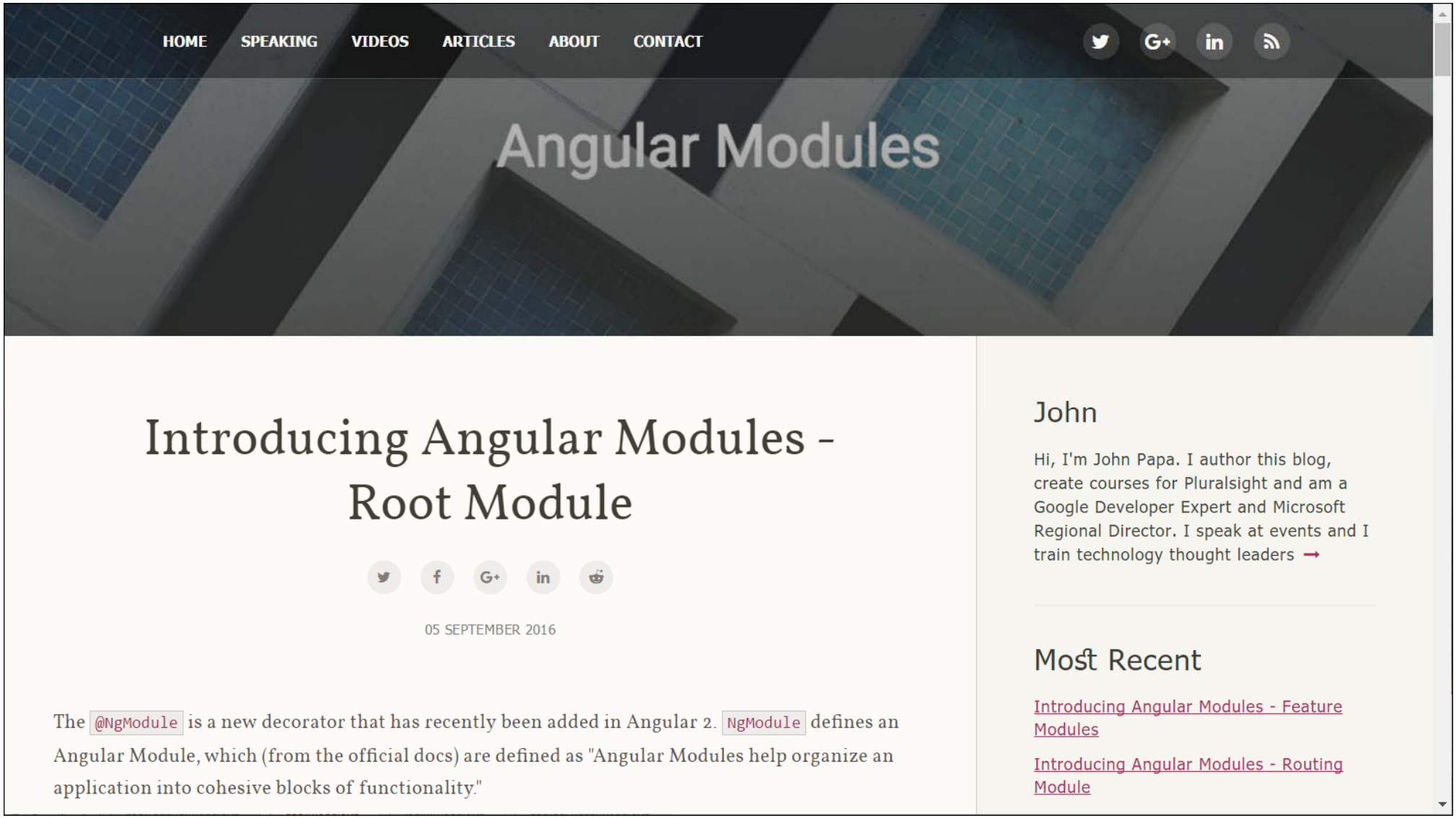
Edit Main Module

```
// 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

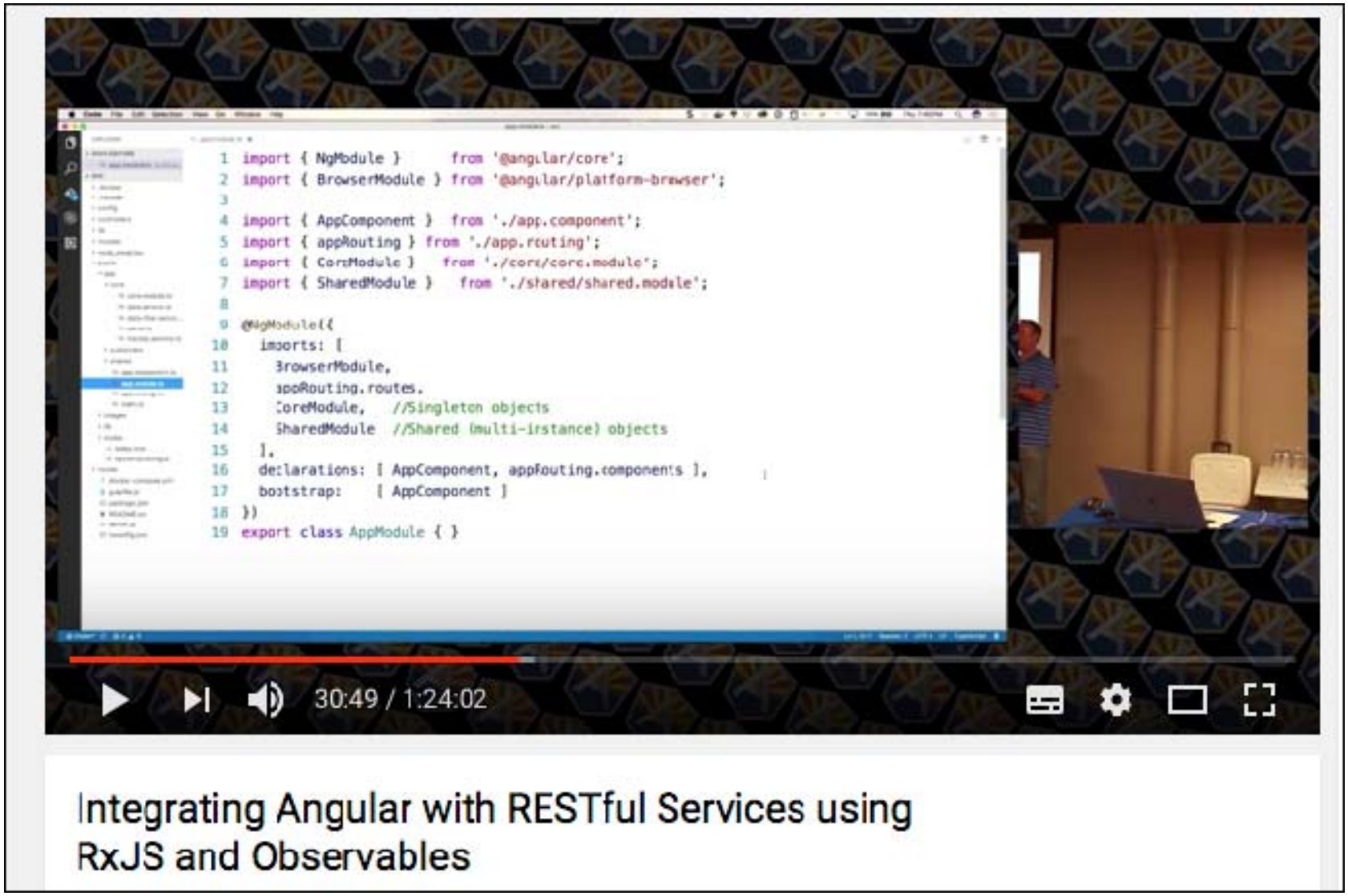


https://johnpapa.net/introducing-angular-modules-root-module/

Official Documentation

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

Dan Wahlin on core module & shared module



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