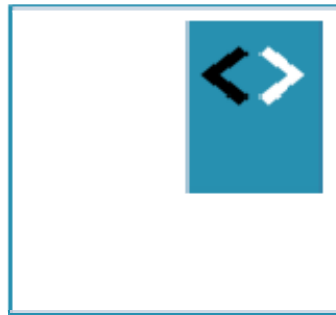




---

# Angular 2

## Module 10 – Multiple Modules



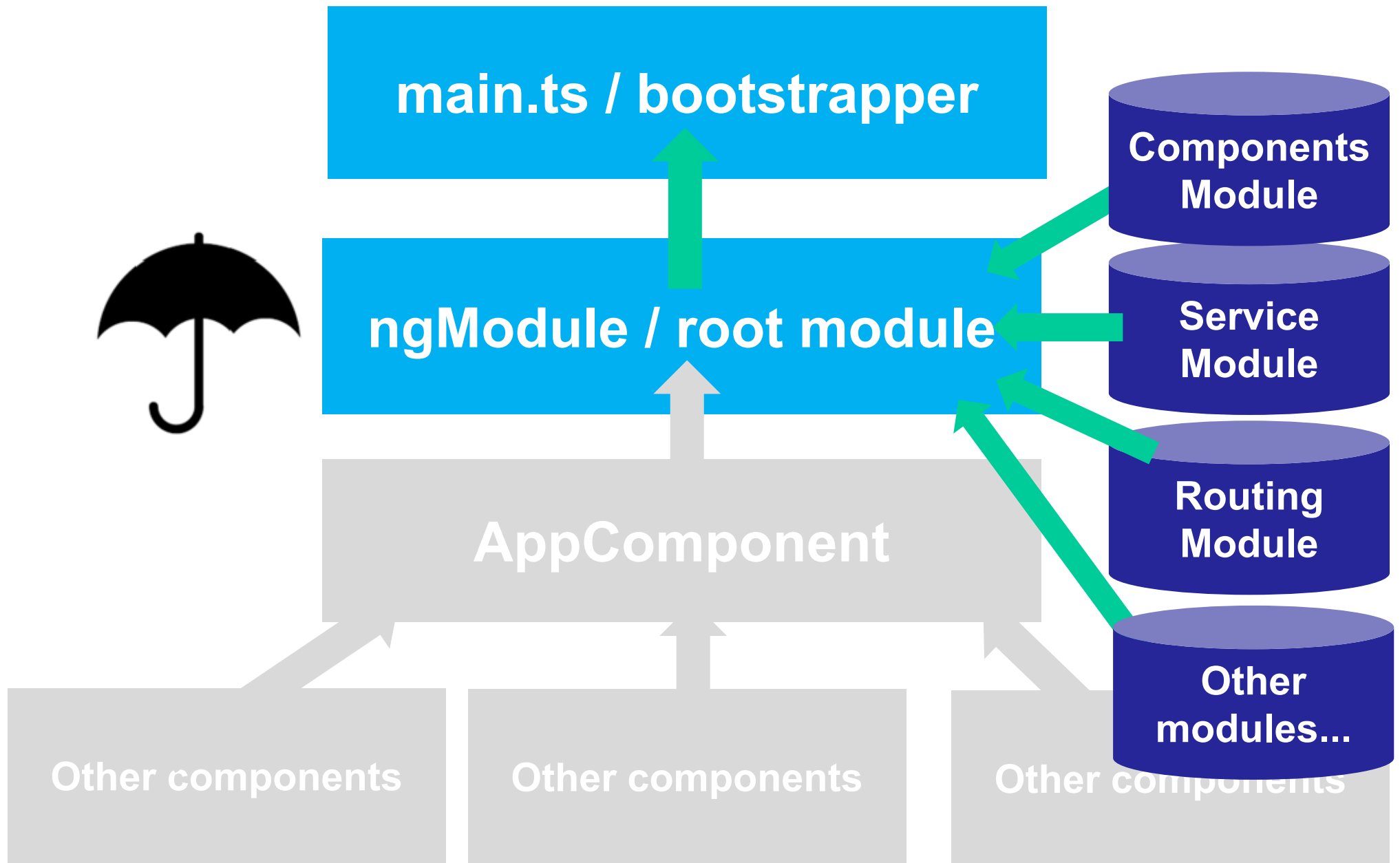
Peter Kassenaar –  
[info@kassenaar.com](mailto:info@kassenaar.com)

# 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 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';
```

Import all stuff needed by this module

```
// 2. Import Components
```

```
import {AppComponent} from './app.component';
```

```
import ...
```

Define routes

```
// 3. Routing table
```

```
const AppRoutes: Routes = [...]
```

```
// 4. Declare *new* NgModule!
```

```
@NgModule({  
  imports: [RouterModule.forRoot(AppRoutes)],  
  exports: [RouterModule]
```

Create new Module. Import & Export enhanced RouterModule

```
})
```

```
export class AppRoutingModule {  
}
```

```
export const routingComponents = [  
  AppComponent,  
  CityAddComponent,  
  CityDetailComponent,  
  CanDeactivateComponent
```

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';
```

Import New Module

```
// 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 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 necessary anymore, as these are defined in their own module
  // ],
  bootstrap: [
    MainComponent
  ]
})
export class AppModule {
}
```

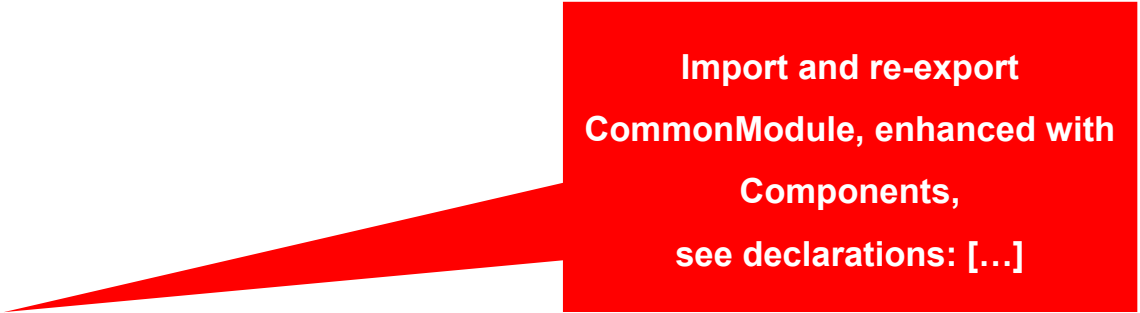
No more providers [...], as this  
now lives inside the  
AppServiceModule

## 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    : [
```

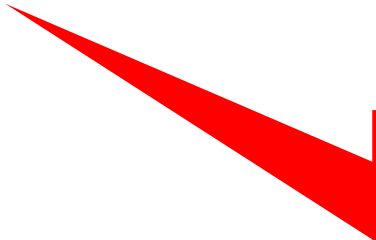
```
    MainComponent
```

```
  ]
```

```
})
```

```
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



The screenshot shows the Angular official documentation website. The top navigation bar is blue with the Angular logo on the left and links for FEATURES, DOCS, EVENTS, NEWS, and GET STARTED on the right. A search bar is located on the left side of the page. The left sidebar contains a list of navigation links: DOCS HOME, CORE DOCUMENTATION, QUICKSTART, GUIDE, API REFERENCE, ADDITIONAL DOCUMENTATION, TUTORIAL, ADVANCED, and a section for Angular Modules (NgModule) which is currently selected. Below this section, there are links for Animations, Attribute Directives, Browser support, Component Styles, and Hierarchical Injectors. At the bottom of the sidebar, there is a link for Angular for TypeScript. The main content area has a blue header with the title 'ANGULAR MODULES (NGMODULE)'. Below the header, the text reads 'Define application modules with @NgModule'. A paragraph explains that Angular Modules help organize an application into cohesive blocks of functionality. Another paragraph states that an Angular Module is a class adorned with the @NgModule decorator function, which takes a metadata object to tell Angular how to compile and run module code. It identifies the module's own components, directives, and pipes, making some of them public for external use. It also mentions adding service providers to the application dependency injectors. A final paragraph explains how to create NgModule classes and how to load them, either immediately at application launch or later via the Router. A 'Table of Contents' section is visible at the bottom, with a link to 'Angular modularity'.

ANGULAR

FEATURES DOCS EVENTS NEWS GET STARTED

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

## 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.

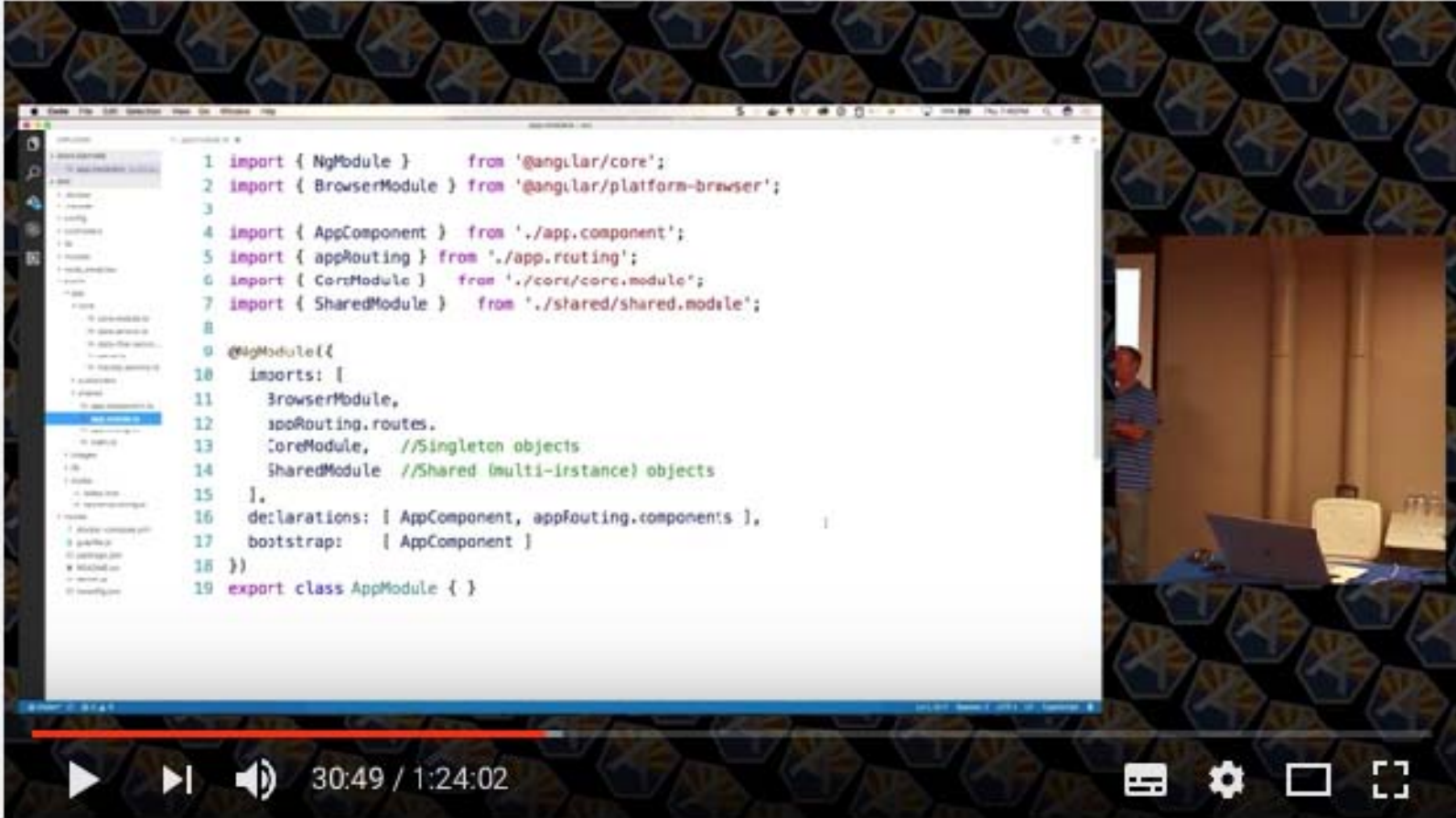
This page explains how to **create** **NgModule** classes and how to load them, either immediately when the application launches or later, as needed, via the **Router**.

### Table of Contents

- [Angular modularity](#)

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

# Dan Wahlin on core module & shared module



The screenshot shows a video player interface. The main content is a code editor displaying the configuration for an Angular application's main module, `AppModule`. The code is as follows:

```
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3
4 import { AppComponent } from './app.component';
5 import { AppRoutingModule } from './app.routing';
6 import { CoreModule } from './core/core.module';
7 import { SharedModule } from './shared/shared.module';
8
9 @NgModule({
10   imports: [
11     BrowserModule,
12     AppRoutingModule,
13     CoreModule, //Singleton objects
14     SharedModule //Shared (multi-instance) objects
15   ],
16   declarations: [ AppComponent, AppRoutingModule ],
17   bootstrap: [ AppComponent ]
18 })
19 export class AppModule { }
```

The video player controls at the bottom show a progress bar at 30:49 / 1:24:02. The video title is "Integrating Angular with RESTful Services using RxJS and Observables".

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