"THIS GUIDE IS PERFECT FOR NEW AND EXPERIENCED ANGULAR DEVELOPERS."

CUSTOM ANGULAR MODULES

LEARN HOW TO BUILD ANGULAR MODULES THE RIGHT WAY

BY MATT VAUGHN

Custom Angular Modules

Matt Vaughn

2017-09-20

Contents

Chapter 1

Introduction

1.1 Welcome

Welcome to the `Custom Angular Modules` guide. This is not just another book about Angular. However, this book is not about teaching you the basics of Angular or using Typescript. There are many resources out there that you can access. The purpose of this book is to provide you with guidance on how to use Angular Modules the right way. You need to know how to use this fundamental element of Angular effectively so that you do not waste any time and resources by doing it the wrong way. Rewriting and/or refactoring applications is an expensive process. Your application or startup doesn't have to be another failure statistic - the Angular framework provides you with the tools you need. Let's learn how to use one of them to create a killer app.

1.2 No Excuse to Not Build It Right the First Time

You should be able to build the application right the first time so that you can create amazing solutions for your current projects, but also for your future projects. If you need to leverage and reuse code that you have already developed wthout doing the dredded copy/paste, you need to *learn how to create custom Angular Modules the right way.* Thanks for reading this guide. You will find complete and detailed instructions on how to create custom modules.

Before we get started though, you need to know and understand how Angular Modules can work for you to create amazing applications. The solutions you develop need to be of the highest quality - your career and reputation depends on it. I want you to be successful. Ask these questions about your code:

- Are my components and services *testable* using testing frameworks?
- Are my components and services *extensible* and allow me to add features easily.
- Are my components and services *maintainable* to allow developers to be efficient.
- Are my components and services *sharable* to allow for efficient code reuse.

Don't be too sad if you answered `no` to any of the questions above. It is not entirely your fault. When new tools and frameworks are released there is a lot of excitement. You get the typical `Hello World` applications and tutorials from all of the rock star coders out there. When we are trying to learn new things, we gravitate towards these tutorials, ebooks, videos, and blogs. However, not many individuals are providing guidance on how to architect solutions that are:

- testable
- extensible
- maintainable
- sharable

No worries, this guide will teach you a valuable and essential lesson - how to create Angular Modules. Modules help us to keep things simple, use single responsibility and seperation of concerns principles.

1.3 What You Will Learn

As mentioned previously, the focus of this book is to provide guidance on understanding and using Angular Modules effectively. If you are new to Angular and/or Typescript, please see the Resources chapter of this book.

Whether your an experienced or new Angular developer, it should be your goal to know:

- Why modules are important in your application architecture and design.
- How to create custom modules that can be developed as their own Angular libraries.
- How to use your custom modules in other Angular applications.
- How to create different types of modules that take care of application concerns, like:
  - Core and shared elements of your application.
  - UI and application components.
  - Infrastructure services (i.e., Logging, Http, Security) and domain feature services that are specific to the solution.

This guide will provide you with additional resources so that you are successful in implementing your own modules. You will have links to GitHub.com projects that contains source code and examples. In addition, there will be a set of video courses that walk you through the basics and the advanced topics of Angular modules.

Chapter 2

What's the Problem with My Modules?

If you are not sure, you may already have a problem.

Angular is built from the ground up with modularity in mind. When you examine the structure of an Angular application, you see that it is a composite of many modules. If it were a pyramid, you would have the `app.module` as the tip and all of the required core modules, modules you import from the framework, npm packages loading modules, and perhaps your custom feature modules.

```
"Technology should enable, not disable, right?"
```

The Angular team has given the developer community an awesome framework. However, having a garage full of the best tools is of no use if you do not know how to use them the right way.

2.1 How do Angular Modules work?

Angular applications really only need one module to be a functional single page application. However, would you buy a house with only 4 walls - I guess it's kind of funtional, but not very useful. Different rooms in a house provide the ability to organize and enable specialized functionality for each room. The kitchen is very different from the laundry room - and you would rather sleep in a bedroom instead of the garage, right?

It is the same with Angular Modules, you can use them to organize your code and provide specific features. Code modularity is very much in line with the Singe Responsibility Principle (SR). How many modules do you have in your application?

The angular.io site has more details on what a module is and how they are used in a basic application. However, there doesn't appear to be much guidance on how to implement modules or how to create your own custom Angular modules. The purpose of the book is to provide some guidance, not rules, on how to take advantage of Angular's modularity. We will promote software development principles that make sense and that also allow you to create solutions that are testable, extensible, and very maintainable. The Angular team has given us a lot of features - however, learning to use the most basic ones to their capacity will serve us well.

2.2 How Many Modules Do I Need?

Now that we have some basic understanding on how modules work, you might have some questions:

- Do I need more than one module?
- Do I need to create and use custom modules?
- What are the different types of modules that I should consider using?

Chapter 3

Why Create Angular Modules?

This section of the book will provide some reasons and motivation to use Angular modules more fully. We've read or participated in developer discussions about modularity and the benefits. Fundamentally, it is a great concept. However, sometimes not so simple to implement in the real world. It takes effort to understand and design applications. Modules give you the mechanism to organize the `things` of your application into containers. So that when combined or available from a specified module container it makes sense. Modules also provide you with some context of the feature set; in terms of how it is used and what it does.

Modules are also very much in line with object-oriented programming designs and patterns. These structures emphasize separating features of a program, or features that can be distributed from distinct modules. A module is a self-contained structure of related things. Modules also help in the decomposition of applications into smaller pieces of functionality (SoC - separation of concerns).

3.1 Modularity, Modularity, and Modularity

The whole notion of modules is to create reusable items to improve efficiency and minimize maintenance of source code. The base element of an Angular application is a Module. Each Angular application contains an app.module that is the entry point of the entire application.

Therefore, you are already familiar with using modules. Also, it is very common to create additional modules in an Angular applications. However, what if you want to create Angular modules that can be reused in more than a single Angular application? Or, would you like to create custom modules that can be published to NPM and shared with the world?

Think of modules as containers of related things. Even if you never create a custom module that is distributable, you can still dramatically improve the design of you existing Angular applications by creating modules to encapsulate and organize specific feature sets. Angular modules let you decide what items within the module are publicly available. For example, if you create a module that contains an Angular Service, it can act like an API that provides end-points to the application to perform different actions. You get to use and implement some nice architectural patterns that provide:

- A service facade pattern to provide end-points of functionality.
- Abstract the implementation details to private members of the module.
- Create a highly-testable service using specification tests.
- Encapsulation of many items used to implement the feauture set.

3.2 Reusable Angular Libraries

You are already taking advantage of reusable Angular libraries in the form of modules. The Angular team has encapsulated many of its feature sets into distinct reusable libraries: modules. You simply reference and import these modules into your application. The Angular team has created the framework with infrastructure elements with amazing reusability - the `@NgModule` mechanism. They use it internally to create many of the usefule feature sets of Angular, such as:

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

3.3 Reuse Things Already Developed

You probably already have some candidates (Services, Components, etc.) in mind that can be shared and reused in your applications.

Do not duplicate your code by copying to other places in your applications. Bad things will find you eventually!

Lt. Joe Kenda is a detective that had an amazing career in Colorado Springs. He has solved literally hundreds of murder crimes. He is pragmatic and he doesn't make mistakes. If Joe Kenda was a programmer, he wouldn't cut and past code from another app.

For example, my team is now finishing up the second of two enterprise Angular applications within 9 months. During the development of the second application we noted that there were some things in the first application that we wished we could use in the new application. And now that we are finishing up the second application, there are even more `elements` that

we want to make reusable for future projects. The best way to do this is to create reusable packages - custom Angular modules.

Our current application takes advantage of using different kinds of modules throughout the application. Now, we are going a step further by creating modules that can be distributed and reused on more than a single application. This is the next level of modularity. Do it now, because you might not get the chance for a `do-over`!

Chapter 4

Scope of Solution

The scope or context of this information is creating custom modules that contain reusable `elements` for Angular web applications. This guide will demonstrate how to create a reusable module that contains either services or components.

Just today our team designed a custom submit button that changes state and shows a spinny icon while submitting to the back end of the application. The problem with most buttons is when do you enable the button to be submittable (when the form is valid, right?). And after you have clicked the button, you want to restrict users from re-clicking the Submit button.

Our new Submit button `component` can now be used by all forms in different applications. But only, if we include it in our custom UI module. Developing shared module packages allows you to create something that can be used everywhere, but maintained in a single location. And now, when you want to add more features to the component, add the feature and bump the version number of the module. You now also have backward compatibility for other applications using the older version. Nice!

You will need to determine how you manage versions of the module if you are not publishing to https://www.npmjs.com/. NPMjs already has built-in version management for your NPM packages/modules.

Chapter 5

Tools

The examples demonstrated will use Visual Studio Code and other tools listed below to create the solution. Your development environment should have the following installed and configured.

- node.js and npm
  - Download at: https://nodejs.org/en/download/
- Typescript
  - Install using npm: `npm install -g typescript`

5.1 Angular CLI

You are probably familiar with the Angular CLI tool - a command line interface for Angular. More information at: https://cli.angular.io/. It has many different features that allow you to create new applications, build and service apps, testing, linting, and formatting. You can also use to generate (or scaffold) modules, services, components, classes, directives, interfaces and more. You can get a lot of mileage out of tools like this.

```
npm install -g @angular/cli
```

Chapter 6

Getting Started

Use the following steps to create a new module project. The instructions provide enough details for you to create a module of (2) different types. The Module Resources section has information on creating (2) different types of modules - each has a different responsibility within the architecture of the application.

1. UI Modules
2. Service Modules

6.1 Angular Application Folders

Within Visual Studio Code, open the Powershell Terminal. Change to a directory where you want to create the module project. Create a folder for the project.

```
mkdir ng-common
```

Now that we have a folder that will contain our module solution, open the folder using the option from the File menu.

6.1.1 Angular Folders

By convention, Angular web applications use a specific folder structure. We also want to take advantage of the `angular-cli` tool to generate code for our project. Therefore, we will create a `src\app` folder structure for the Angular members of the project.

6.2 Configuration Files

6.2.1 index.ts

This file is really the most important element of the solution. It will allow you to publicly expose (can I say that in technical documentation?) or allow clients to find the specified module of the package - which is very important to Angular applications.

The only project member we need to expose is the module itself. The `module` will actually define what elements of the module are publicly visible - more on that later.

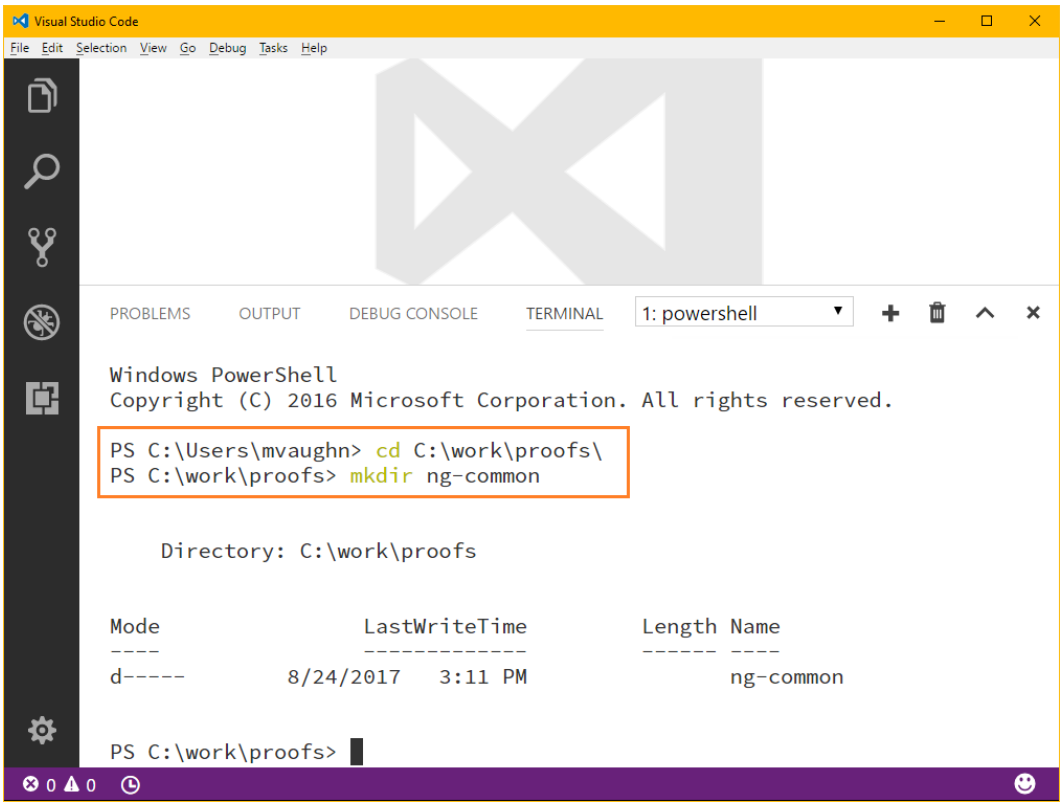

Figure 6.1:

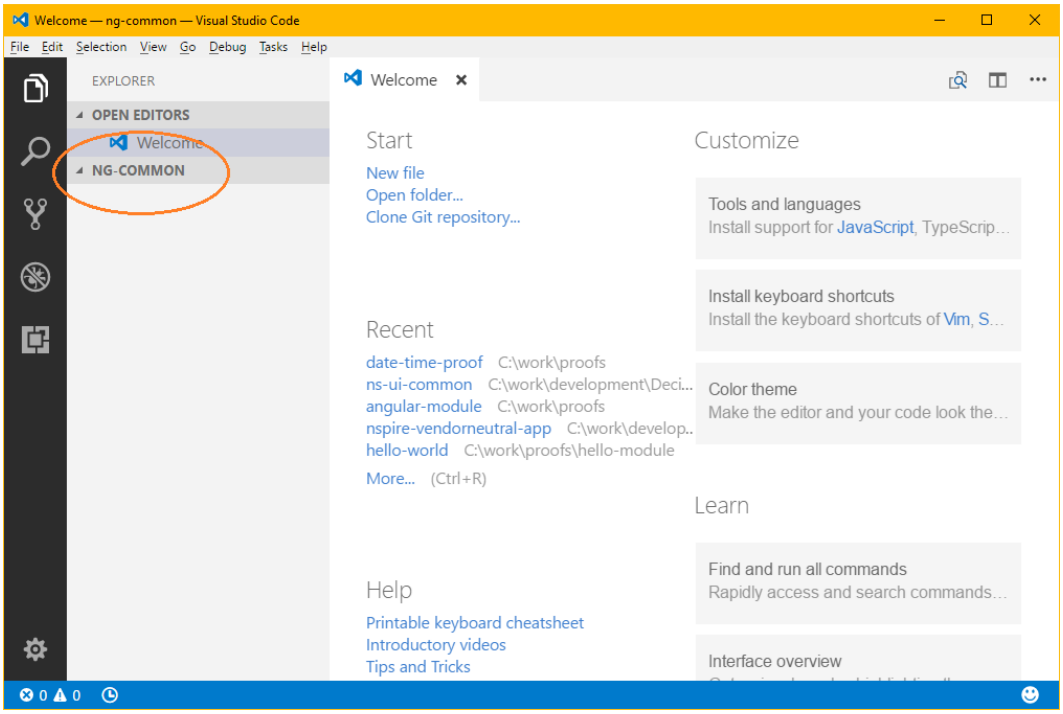

Figure 6.2:

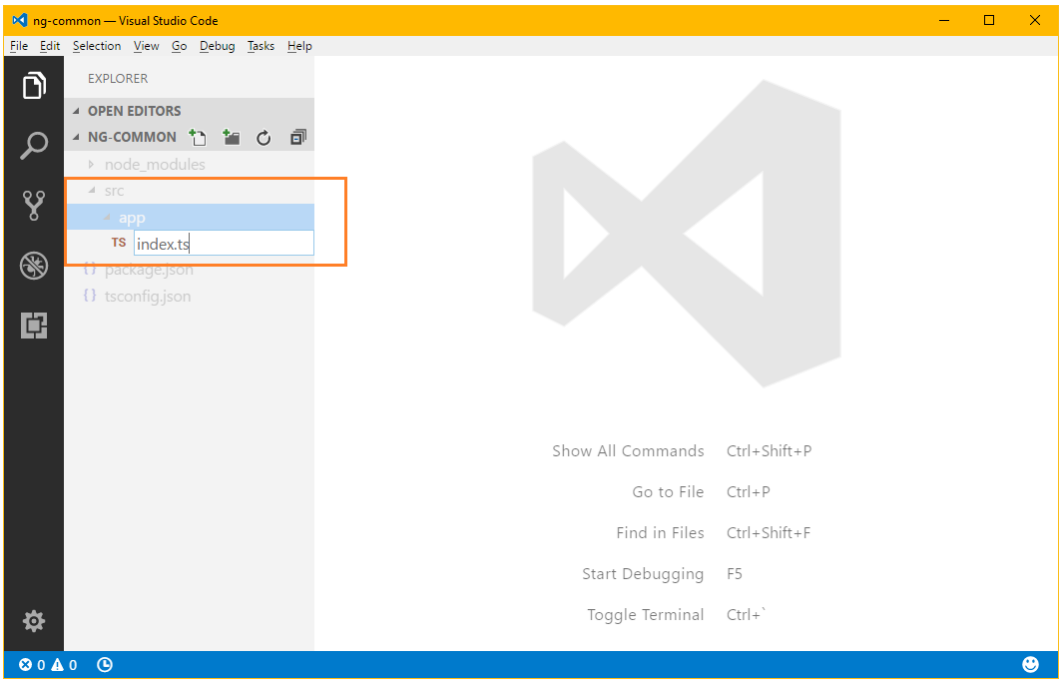

Figure 6.3:

6.2.2 package.json

We will use a `package.json` file to set name, version, and also to reference package dependencies required for the module.

Open the Powershell Terminal and run the following command. This will create the `package.json` file.

```
npm init
```

The `npm init` command creates a new `package.json` file after guiding through a few prompts for information about the package. This file is required when you want to use and or publish your package.

The complete/revised full version of the `package.json` file.

```
{
  "name": "StarterModule",
  "version": "2.1.1",
  "description": "asdf",
  "main": "index.js",
  "scripts": {
    "transpile": "ngc",
    "build": "npm run transpile"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/buildmotion/angular-module-starter.git"
  },
  "author": "",
  "license": "MIT",
```

```
  "bugs": {
    "url": "https://github.com/buildmotion/angular-module-starter/issues"
  },
  "homepage": "https://github.com/buildmotion/angular-module-starter#readme",
  "devDependencies": {
    "@angular/cli": "^1.4.1",
    "@angular/common": "^4.4.0-RC.0",
    "@angular/compiler": "^4.4.0-RC.0",
    "@angular/compiler-cli": "^4.4.0-RC.0",
    "@angular/core": "^4.4.0-RC.0",
    "rxjs": "^5.0.1"
  }
}
```

6.2.3 tsconfig.json

The next section is very important. Important. Typically, you would compile a Typescript project using `tsc` - most of these applications contain a `tsconfig.json` file that contains the configuration for the build process. We are going to do much the same. However, we are going to use the `ngc` - the ng compiler tool with its own configuration. The `ngc` is located at: `.\node_modules\.bin\ngc`.

Use the command below to create a new `tsconfig.json` file in the root of the project. We will need to update the configuration using the code snippet below.

```
tsc --init
```

Open the `tsconfig.json` file and update the configuration using the configuration in the snippet below.

```
{
  "compilerOptions": {
    "baseUrl": ".",
    "declaration": true,
    "experimentalDecorators": true,
    "inlineSources": true,
    "lib": [
      "es2015",
      "dom"
    ],
    "module": "es2015",
    "moduleResolution": "node",
    "noImplicitAny": true,
    "outDir": "dist",
    "paths": {
      "@angular/core": [
        "node_modules/@angular/core"
      ],
      "rxjs/*": [
        "node_modules/rxjs/*"
      ]
    },
    "rootDir": "src/app",
    "skipLibCheck": true,
```

```
    "sourceMap": true,
    "strictNullChecks": true,
    "stripInternal": true,
    "target": "es5",
  },
  "files": [
    "./src/app/index.ts"
  ],
  "angularCompilerOptions": {
    "strictMetadataEmit": true
  }
}
```

For better build automation, you can update your `package.json` file to include the code below. Notice that we are indicating that the transpile should be performed by the `ngc` - the Angular compiler tool. This tool is located at .

```
"transpile": "ngc",
"build": "npm run transpile"
```

Full sample of the `package.json` configuration.

```
{
  "name": "StarterModule",
  "version": "2.1.1",
  "description": "",
  "main": "index.js",
  "scripts": {
    "transpile": "ngc",
    "build": "npm run transpile"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/buildmotion/angular-module-starter.git"
  },
  "author": "",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/buildmotion/angular-module-starter/issues"
  },
  "homepage": "https://github.com/buildmotion/angular-module-starter#readme",
  "devDependencies": {
    "@angular/cli": "^1.4.1",
    "@angular/common": "^4.4.0-RC.0",
    "@angular/compiler": "^4.4.0-RC.0",
    "@angular/compiler-cli": "^4.4.0-RC.0",
    "@angular/core": "^4.4.0-RC.0",
    "rxjs": "^5.0.1"
  }
}
```

Use the Powershell terminal to build with the command below. This will build and put the output files into the `dist` folder.

```
npm run build
```

6.2.4 Compiler Options

There are quite a few compiler options available. Update the new `tsconfig.json` file with the following configuration shown below. Find more information about options here: Typescript Compiler Options

- **declaration**: Generates corresponding .d.ts file.
- **emitDecoratorMetadata**: Emit design-type metadata for decorated declarations in source.
- **experimentalDecorators**: Enables experimental support for ES decorators.
- **inlineSources**: Emit the source alongside the sourcemaps within a single file; requires –inlineSourceMap or –sourceMap to be set.
- **mapRoot**: Specifies the location where debugger should locate map files instead of generated locations. Use this flag if the .map files will be located at run-time in a different location than the .js files. The location specified will be embedded in the sourceMap to direct the debugger where the map files will be located.
- **module**: Specify module code generation.
- **moduleResolution**: Determine how modules get resolved. Either "Node" for Node.js/io.js style resolution, or "Classic". See Module Resolution documentation for more details.
- **noEmitOnError**: Do not emit outputs if any errors were reported.
- **noImplicitAny**: Raise error on expressions and declarations with an implied any type.
- **outDir**: Redirect output structure to the directory.
- **rootDir**: Specifies the root directory of input files. Only use to control the output directory structure with –outDir.
- **sourceMap**: Generates corresponding .map file.
- **target**: Specify ECMAScript target version.

6.3 angular-cli.json

Create/Add a new file called `angular-cli.json` in the root of the project folder. Add the following contents.

You may need to update the `project.name` value to the name of the current module project you are working on.

```
{
    "project": {
        "version": "1.0.0",
        "name": "ng-common"
    },
    "apps": [
        {
            "tsconfig": "tsconfig.json",
            "mobile": false,
            "root": "src",
            "prefix": "app"
        }
    ],
    "defaults": {
        "styleExt": "css",
        "prefixInterfaces": false,
        "lazyRoutePrefix": "+"
```

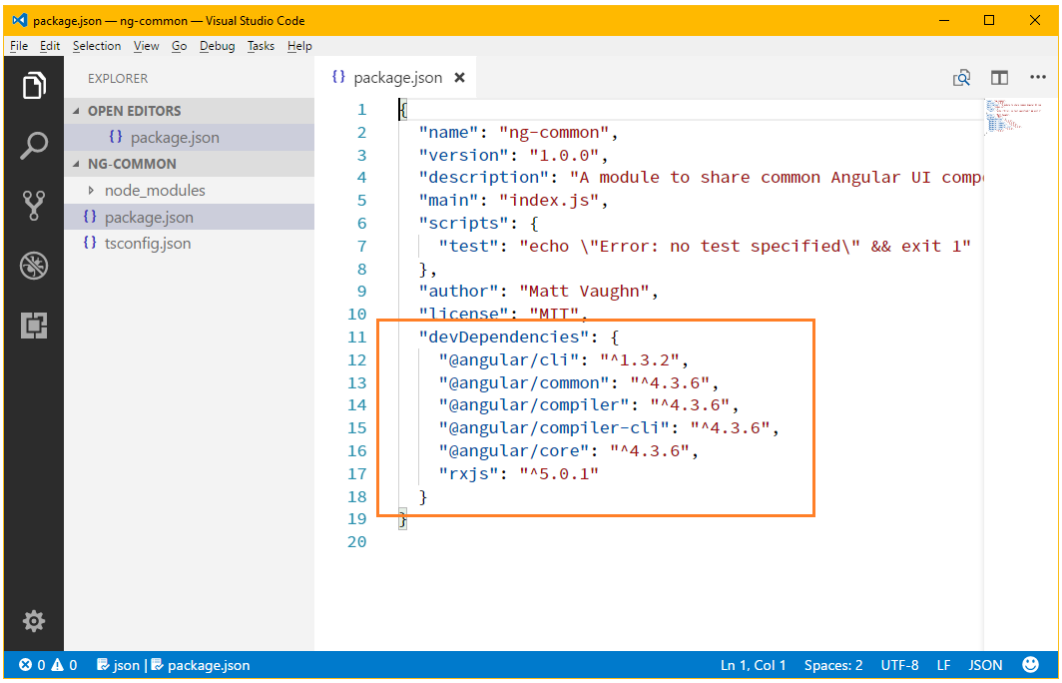

Figure 6.4:

```
    }
}
```

6.4 Install NPM Packages

Since our module is for sharing common Angular components, we will install the following packages. Use the Powershell Terminal to run the following commands.

```
npm install --save --save-dev rxjs@^5.0.1
npm install --save --save-dev @angular/cli@latest
npm install --save --save-dev @angular/core
npm install --save --save-dev @angular/common
npm install --save --save-dev @angular/compiler
npm install --save --save-dev @angular/compiler-cli
```

Note, that the packages are only installed in the `devDependencies` section of the `packages.json` file. We only require the elements for development - we expect that the client Angular application will have the required package references in the `dependencies` of its packages.json file.

Chapter 7

Create a Module

Now, the moment that we have been waiting for. Create a module that will provide access to all of the shared members (components, services, etc.) we'll create later.

Run the following command to create a new module with the name `ngCommonModule`. Note that you do not have to include the `Module` suffix. The angular-cli tool will do that for you by convention.

```
syntax: ng generate module <NAME-OF-YOUR-MODULE>
example: ng generate module ngCommon
```

7.1 Module Configuration

We will use the `@NgModule` declaration to add imported items to the `imports`, `declarations`, and `exports`

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

\@NgModule({
  imports: [
    CommonModule
  ],
  declarations: []
})
export class NgCommonModule { }
```

7.1.1 Module Name

The default name of the module is created when the `package.json` file is created. Now that we have a module and the client applications will reference the module by name, you will need to update the `name` setting in the `package.json` file before publishing the module.

```
old: ng-common
new: NgCommonModule
```

Note: Later we will copy the `package.json` file to the output directory before publishing the module. The names should match in both files.

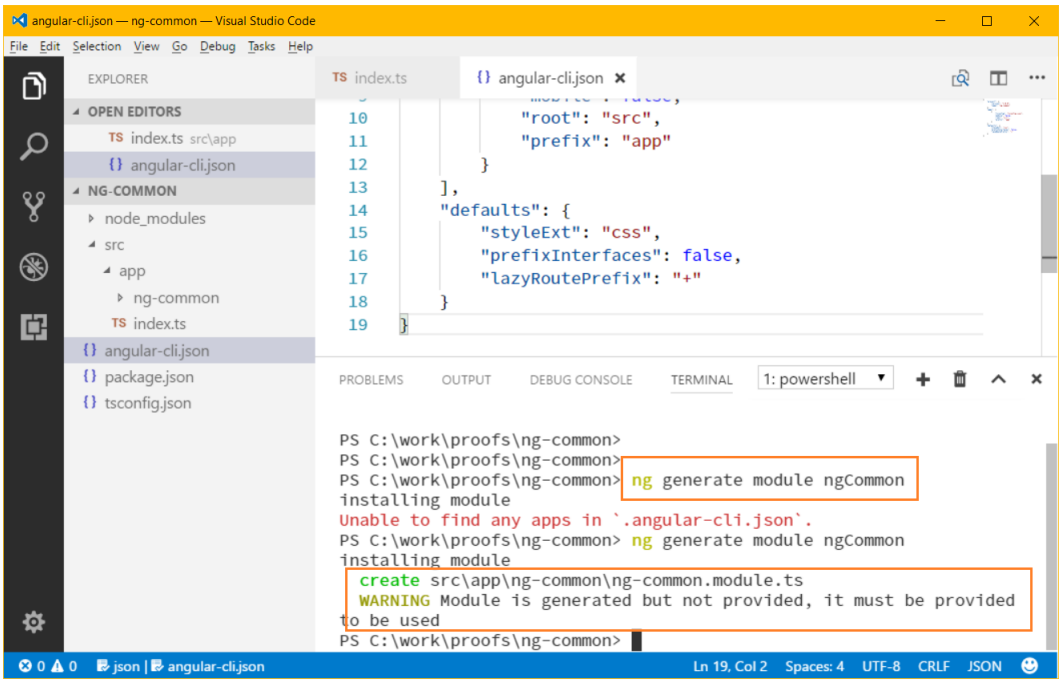

Figure 7.1:

```
{
  "name": "NgCommonModule",
  "version": "1.1.1",
  "description": "A module to share common Angular UI components.",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Matt Vaughn",
  "license": "MIT",
  "devDependencies": {
    "@angular/cli": "^1.3.2",
    "@angular/common": "^4.3.6",
    "@angular/compiler": "^4.3.6",
    "@angular/compiler-cli": "^4.3.6",
    "@angular/core": "^4.3.6",
    "rxjs": "^5.0.1"
  }
}
```

7.1.2 Export the Module

Add an `export` for the module in the `index.ts` file. When the module project is compiled, this will create an `index.js` file that is referenced as the `main` value in the package.json file.

```
export * from './ng-common/ng-common.module';
```

***NOTE**: Adding the `export` item for the module is required. Otherwise, the Angular client*

application will throw an exception stating that the ngCommonModule is not an @ngModule - basically, it cannot find it due to not being exported and referenced in the ***index.js****.*

7.2 Module Resources

Now that we have a structure and process to create a module we are in a better place. Now you need to start thinking about the contents of the module.

Principle: A module should only contain related and like items that as a composite make up the module contents. A module should not be a **"junk drawer"**_ that contains non-related items. _

For example, in our current Angular applications we have service and ui modules. The *UI modules* contain components, routes, pipes, directives, and things related to the UI or display of content. We also have *service modules* that contain Angular services, business logic code, HTTP services, rules and business actions. These services basically contain the *business logic* of the application to enable a clear separation of concerns between the UI/UX and the business end of the application.

My hope is that we can abstract the *service modules* into separate and reusable modules. One scenario is that the service modules have the potential for reuse in other web and hybrid mobile applications. Another benefit, is that the unit and/or specification testing of service modules is simplified and targets/context of this type of testing is much different that UI testing.

The next two sections talk about *ui modules* and *service modules*. My opinion is that modules of these types should not be combined into a single module

7.2.1 UI Modules: Components and Angular "Things"

Use the `angular-cli` to create a component. Note that we are using the path `ng-common\helloWorld` - this allows the CLI to find and update the `ngCommonModule` with the component configuration (Why write code when you don't have to?):

1. import
2. declaration
3. export

```
ng generate component ng-common\helloWorld
```

7.2.1.1 Declare and Export Components

If you want your components to be publicly exposed, or I should say available (sounds better), you will need to add the component to the `exports` array of the @ngModule.

Exporting, allows the component to be publicly available to applications that import/use this module.

```
\@NgModule({
  imports: [
    CommonModule
  ],
  declarations: [HelloWorldComponent],
  exports: [
    HelloWorldComponent
```

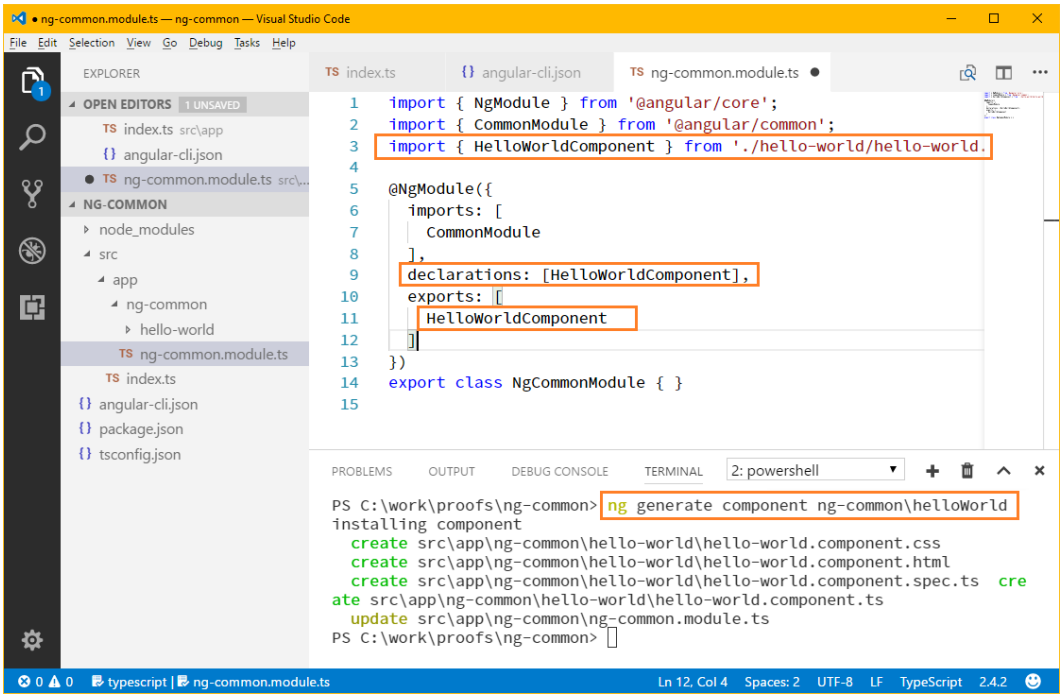

Figure 7.2:

```
  ]
})
```

7.2.1.2 HelloWorldComponent

The `HelloWorldComponent` is created in its own folder using the same conventions we are used to when working with an Angular application. We have:

- a `component` class with an `@Component` decorator.
- an HTML file that will render when the `selector` is used.
- a CSS file for styling the HTML.
- a `specification test` file to write specification tests.

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-hello-world',
  templateUrl: './hello-world.component.html',
  styleUrls: ['./hello-world.component.css']
})
export class HelloWorldComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

7.2.1.3 What to do about HTML and CSS files?

In a typical Angular application, a component references the HTML and CSS files in the `@Component` decorator. For shared module packages (which are not Angular applications) it is best to include the HTML and CSS as inline text in the **?** decorator. Otherwise, you will need to implement a more elaborate build scheme that will bundle and/or inline the file contents for you.

My opinion is that the Angular CLI should have the ability and feature to create a stand alone module without requiring the context of being within an Angular application. There are requests for this feature - however, not much activity from the team managing the Angular CLI tool. It is nice that we ***CAN*** use the CLI to create feature and service modules in existing Angular applications. But the point of even having the concept of modules is that they can be reused and shared. If they are stuck in the muck of a single application you have the temptation of copying the folders/files into a new project. And if you do that (I'm guilty of this from time to time), beware trouble is lurking and it will find you!

7.2.2 Service Modules: Services, Business Providers, Actions, and Rules...

Use the information in this section if you building a *service module* - a module that is basically an entry point into your business logic with a service API-like abstraction. The service APIs provide a facade pattern implementation that delegate the real concrete business logic in other classes below the service.

Each consumer of the service (most likely an Angular Component) will only have direct access to the service of the module. Access to things internal should not be allowed. For example, you do not want the components to directly interact with or have access to:

- Business Providers
- Business Actions*
- Business Rules
- Http Providers/Services that interact with the web services or Web APIs

Our web team has current standards, patterns, and practices for implementing business logic in our Angular applications - we take advantage of the angular-actions npm package along with the angular-rules-engine. These packages elevate your business lgoic to the next level with a robust processing framwork for business logic, authorization, business rule process, data validation, and interaction with other services like Angular HTTP.

Chapter 8

Compiling and Publishing the Module

We now have a module with something that can be reused - everyone wants the awesome `HelloWorldComponent`, right? You get the point - you can now create as many reusable components as you want.

Let's compile something so we can use the module. Typically within Visual Studio Code you can use the command below to run the build task.

```
SHIFT+CTRL B
```

However, when we use this command, the tool asks you to select the specified build task. Usually, the `tsc: build - tsconfig.json` option from your `tsconfig.json` file. This will work every time.

8.1 Alternate: Configure a Default Build Task

Use the `CTRL + P` command to show the command pallete. Enter `task` to display the `Tasks` options. Select the `Tasks: Configure Default Build Task`.

Select the option `tsc: build -tsconfig.json` as your option. This will configure the task to use the `tsconfig.json` when building the code with the `tsc` Typescript compiler.

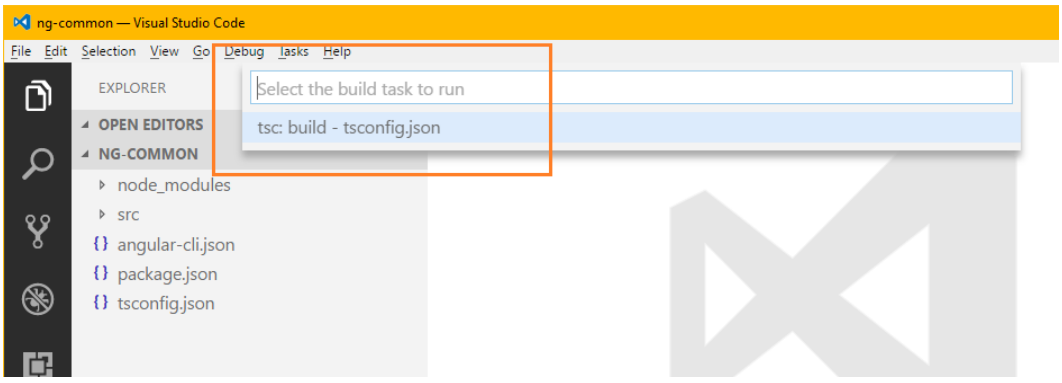

Figure 8.1:

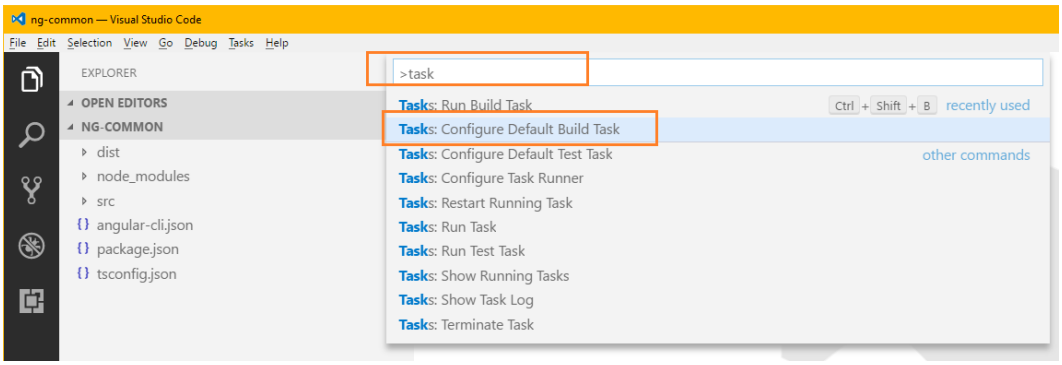

Figure 8.2:

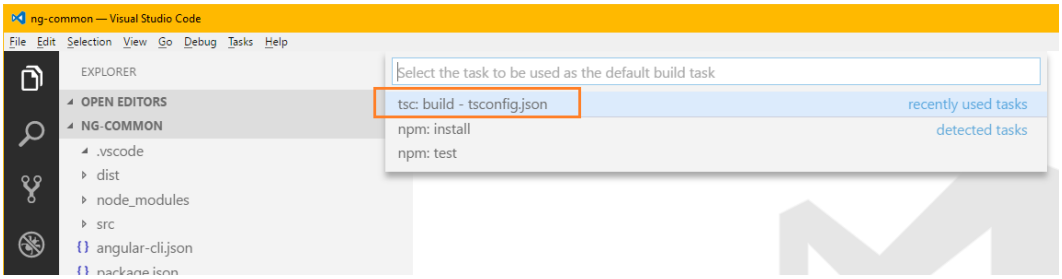

Figure 8.3:

This will create a `tasks.json` file in the .vscode folder of your project.

```
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
            "type": "typescript",
            "tsconfig": "tsconfig.json",
            "problemMatcher": [
                "$tsc"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            }
        }
    ]
}
```

8.2 Compiled Output

The compiled output of the build is in the `dist` folder of the project. The output directory is configured in the `tsconfig.json.outDir` setting.

8.3 Distribution package.json Configuration

In order for other applications to use the module package, the module must contain a `package.json` file. You can copy the `package.json` file from the root of the project to the `dist` folder.

The initial file should have a version of `1.0.0`. You may want to modify the version of the module package when new features, components, or bug fixes occur. You can use the `npm` command to update the version:

```
Syntax: npm version [patch|minor|major]
npm version patch
npm version minor
npm version major
```

The following shows the usage of the npm command and the changes in the version of the module package. How you use versioning is up to you and/or your team. Please remember, that the version and how your package is referenced by clients will have an impact on the applications that will use your module.

Make sure that you are running the `npm version` command from the `dist` folder so that the correct `package.json` file is updated. Or, you may want to update the source file and have an XCOPY process that copies the file to the `dist` folder during builds - they should probably be kept in sync.

```
PS C:\work\proofs\ng-common>
PS C:\work\proofs\ng-common>
PS C:\work\proofs\ng-common> cd .\dist\
PS C:\work\proofs\ng-common\dist> npm version patch
v1.0.1
PS C:\work\proofs\ng-common\dist> npm version minor
v1.1.0
PS C:\work\proofs\ng-common\dist> npm version major
v2.0.0
```

Chapter 9

Using the Module

In order to use the module, you will need to create or configure an existing Angular web application. The most common way to use a package module is to install it using `npm install <myModuleName>`.

```
npm install angular-rules-engine@0.0.27 --save --save-dev
```

This will work if you publish the package to an npm repository like https://www.npmjs.com/. For modules that you want to share with the world - this is a great way to do it.

For example, our team uses the angular-rules-engine npm package developed and maintained by one of team members (me).

If you do not need to or do not want to make your module publicly available, you can publish the contents of the `dist` folder from the module project to a location accessible via file path/network share to the Angular application under consideration.

For example, our team will publish modules to an `ng-resources` folder. This folder can be version controlled and updated when and if newer versions are made available. You can also create folders that have `version` information so that you can maintain and use more than one version of the module.

9.1 Update package.json with Reference to Module

You can add a reference to the `devDependecies` and/or the `dependencies` settings in the applications `package.json` file.

```
"NgCommonModule": "../ng-resources/ng-common"
```

You will need to run the `npm install` command so that the module is found and is added to the applications `node_modules` folder.

```
npm install
```

If the `install` command did its job, you will the name and version of the referenced module package in the Powershell Terminal window. If you increment the version of the module and publish, you can run the same `npm install` command - this will allow the client application to pick up the new version and update the `node_modules` with the correct version. If you do not see the reference to the updated version in the output window, you may need to delete/remove the module folder from `node_modules` and run the `npm install` command again.

```
PS C:\work\proofs\help> npm install
help@0.0.0 C:\work\proofs\help
`-- NgCommonModule@1.0.0
```

9.1.1 Update the Application Module References

The shared module should be imported by the client application.

References:

```
import { NgCommonModule } from 'NgCommonModule/ng-common/ng-common.module';
import { HelloWorldComponent } from 'NgCommonModule/ng-common/hello-world/hello-world.component'
```

app.module.ts

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

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

import { NgCommonModule } from 'NgCommonModule/ng-common/ng-common.module';
import { HelloWorldComponent } from 'NgCommonModule/ng-common/hello-world/hello-world.component'

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

9.1.2 Add the Component to the HTML

You can add the `selector` of any component made available by the shared module. In the client application, update a component's HTML file with the following selector.

```
<app-hello-world></app-hello-world>
```

9.1.3 Running the App

You should be able to run the application and see the HTML from the shared component. This is a simple example - however, the power of being able to distribute and share an Angular module as a library is awesome.

```
ng serve
```

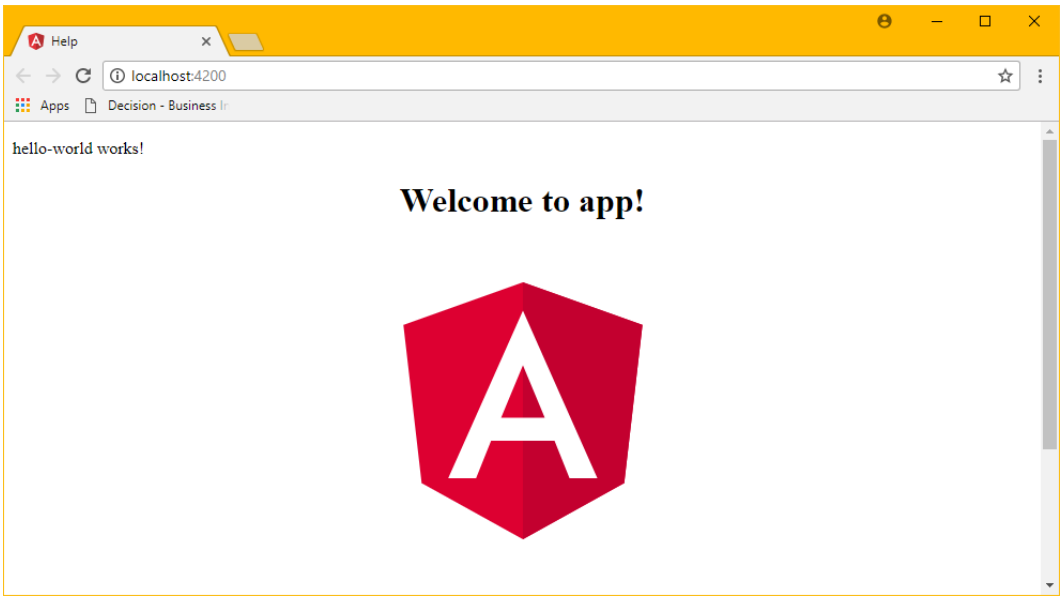
Help
localhost:4200
Apps
Decision - Business
hello-world works!
Welcome to app!

Figure 9.1:

Chapter 10

Resources

10.1 Source Code

10.1.1 angular-module-starter

The source code for the Angular Module Starter is available on Github.com. You can use this Angular Custom Module project to start building your own modules or as a reference. * https://github.com/buildmotion/angular-module-starter

10.2 Books

10.2.1 Rangle's Angular 2 Training Book

http://info.rangle.io/angular-2-training-book

10.3 Web

10.3.1 Rangle.io Angular 2 Training

https://angular-2-training-book.rangle.io/

10.3.2 Angular.io Quickstart

angular.io QuickStart

10.3.3 dofactory.com - Design Patterns

http://www.dofactory.com

10.4 Design Patterns

10.4.1 Facade Design Pattern

- https://en.wikipedia.org/wiki/Facade_pattern
- http://www.dofactory.com/net/facade-design-pattern

10.5 Tools

10.5.1 Angular CLI

https://cli.angular.io/

10.6 Testing

10.6.1 Specification Tests

https://angular.io/guide/testing

Bibliography