Custom Angular Modules

Presented by matt vaughn

Contact

- Matt Vaughn
  - Email: matt.vaughn@buildmotion.com
  - Web: www.angularlicio.us || www.angularlicious.com
  - Github: https://github.com/**buildmotion**
- Presentation, Code Samples and Resources
  - https://github.com/buildmotion/**custom-angular-modules**
    - Custom Angular Module
    - (2) client Angular applications using different versions of the module.
    - Powerpoint Presentation

Presentation, Code Samples and Resources

- https://github.com/buildmotion/**custom-angular-modules**
  - Custom Angular Module (source, dist)
  - (2) client Angular applications using different versions of the module.
    - Basic logging
    - Logging with configuration
  - Powerpoint Presentation
  - Guide (markdown/PDF)

Introduction

What we will learn:

- 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 different application concerns.

What’s wrong with ~~me~~ my modules?

Houston, we have a problem.

- If you application has only one module (i.e., app.module)…
  - you ~~might~~ already have a problem.
- No “junk drawer” modules.
  - Modules with everything and a kitchen sink.
- Application with modules, components, services, or other things copied from another application.
- Code copied to more than one application.

Software Transmitted Defects

- Code copied from one location to one or more different locations.
  - Code has defects.
  - Every place that the code is copied to now has the defect.
- Code needs to be extended.
  - The code will need to be updated in all places where it was copied.
- There is a cure for Software Transmitted Defects.
  - Practice safe programming – DO NOT COPY CODE.

Goals and Strategy

What are we trying to achieve?

Goals

- Great software solutions.
- Efficient with our resources (time and people)
- High *quality* software.
- Share your amazing solutions.

Strategy

- Reuse high quality components and services.
- Leverage Angular tools and elements.
- Better code
  - Consistency
  - Extensibility
  - Maintainability: Decoupling, Interface
  - Shareable/Reusable
  - Scope, Public/Private
- Practice and Principles
  - DRY: Do not repeat yourself.

How can we do this with Angular?

Answer: Modules

- What is a module?
  - A module is a collection of related things that work together. Code cluster.
  - Allows an application to be organized into cohesive blocks of functionalities.
  - Allows an application module to be extended by capabilities of external libraries (i.e., other module packages).
  - Allows applications to be composed by modules.
  - A reusable library.
- Javascript Modules
  - commonJS, AMD, or UMD
    - Help manage dependencies, scope
  - Modules need to be exposed(**export**) to others for use and be accessible(**import**).

Module Pros:

- Pros:
  - Improve efficiency.
  - Minimize code maintenance.
  - Better code ***organization***.
  - Share and distribute → Reusable libraries.

Module Cons:

- Cons:
  - Takes thought, design, analysis to determine [what] belongs in a module.
  - Development approach is different.
  - May take awhile to stabilize the module.
  - Managing dependencies.
  - A suite of modules that are inter-dependent require package and version updates when a dependency is updated.

Custom Angular Module Types

- Component
- Service
- Component/Service
- Infrastructure
- Framework
- Feature

Know Your Angular Module

Hello Module.

Angular Application Modules :: Quick Overview

- **Root** Module: app.module.ts
- **Route** Module: manage application routes
- **Shared** Module: shared.module.ts
- **Core** Module: core.module.ts
- **Feature** Modules
  - UI
  - Service

Root Module: app.module.ts

- Purpose
  - Responsible for initializing the application's modules (loading), and bootstrapping the top-level component (i.e., app.component).
  - Try to keep the concern to initializing the application.
- Contents
  - Common application-level components (NotAuthorizedComponent, PageNotFoundComponent, ErrorComponent, etc.).

Core Module: core.module.ts

- Purpose
  - Part of the application initialization process.
  - To import and reference modules, components, and services that are part of the application’s domain.
  - Only a single-instance of the core.module should be loaded by the application.
- Contents
  - Application specific modules and/or services.

Shared Module: shared.module.ts

- Purpose
  - Responsible for importing and referencing common Angular and 3rd-party modules, common components and/or services.
  - Import and use by other feature modules in the application.
    - Not imported by AppModule or CoreModule.
  - Use to hold the common components, directives, and pipes and share them with the modules that need them.
- Content Samples:

```
import { NgModule } from '@angular/core';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';
import { RouterModule } from '@angular/router';
import { Observable } from 'rxjs/Observable';
```

Feature Module: <my-feature>.module.ts

- Purpose
  - Use to implement a domain feature of the application. The module contains services and owns components with templates.
  - A feature module delivers a cohesive set of functionality focused on an application business domain, user workflow, facility (forms, http, routing), or collection of related utilities.
- Types
  - UI (ng front end)
    - Components, Directives, Pipes, constants
  - Service (ng back end)
    - Service (API)
    - Business Logic Layer
    - Models
    - HttpServices

Different Modules for Different Purposes

- Domain specific providing services, workflow or utilities for the specified application.
- Common services like logging or http.
- Frameworks for processing business and validation rules; business actions.
- Common components (Alerts, Modals, etc.) used by many applications.
- Infrastructure concerns – base classes for components, services, business actions, and HTTP services, etc.

Module Purpose Drives the Design

- Understanding the module purpose will drive the design and implementation of the module.
  - Requires thought and analysis.
- Helps determine how a module is organized.
- Helps determine the contents of the module.
- Emerging Modules
  - Application features, components, services, infrastructure, frameworks change over time.
  - Evaluate what things need to be re-organized or refactored to a module.

Environment Setup

Tools and Stuff

npm and node.js

- Where to get it?
  - https://nodejs.org/en/download/
  - Installs both node and npm.
- Version
  - LTS (long-term support) version
- Resources
  - https://docs.npmjs.com/
  - https://docs.npmjs.com/cli/install

Typescript

- Where to get it?
  - `npm install -g typescript@'>=2.4.2 <2.5.0'`
- Version
  - Depends on version of other developer tools/modules.
    - @angular/cli, @angular/compiler, @angular/compiler-cli, @angular/core
- Resources
  - https://www.typescriptlang.org/
  - https://www.typescriptlang.org/docs/home.html

Angular

- Where to get it?

```
npm install @angular/common@latest
npm install @angular/compiler@latest
npm install @angular/compiler-cli@latest
npm install @angular/core@latest
npm install @angular/forms@latest
npm install @angular/http@latest
npm install @angular/platform-browser@latest
npm install @angular/platform-browser-dynamic@latest
npm install @angular/platform-server@latest
npm install @angular/router@latest
npm install @angular/animations@latest
```

- Version
  - Depends on version of other tools and modules.
- Resources
  - https://angular.io/
  - https://angular.io/docs
  - https://angular.io/resources

Angular CLI

- Where to get it?
  - npm install -g @angular-cli@latest
- Version
  - Depends on version of other tools and modules.
- Resources
  - https://cli.angular.io/

Technical Implementation

Setup & Configuration

Configuration Files

- package.json
- tsconfig.json
- angular-cli.json
- rollup.config.js (optional)
- package.json (dist)
- License
- README.md

Technical Implementation

Module Contents

Module Contents

- index.ts
  - Entry point to the Angular Module
  - Exports all members that are public
  - Contain at the minimum an export of your module.
- <YOUR_CUSTOM>.module.ts
  - Angular Example: HttpModule

Build Process

Transpile

- To tsc or not to tsc?
  - Not!
- Use the ngc
  - .\node_modules\.bin\ngc .\tsconfig.json
- Configure Default Build Task

```
"scripts": {
"transpile": "ngc",
"build": "npm run transpile"
}
```

UMD (Universal Module Definition)

- If you want to provide UMD formatted module
  - UMD bundler
    - https://www.npmjs.com/package/rollup
  - Minify the Script
    - https://www.npmjs.com/package/uglify-js

Deployment

Safe Storage

- Code Repository
- Package Manager (npm)
  - Public
  - Private
- Versioning
  - npm

Using Custom Modules

See the Code

- Install the module
- Import the module/service
- Provide the service
  - ngAppOne
- Configuration for Modules/Services
  - ngAppDos

Resources

https://github.com/buildmotion

- references for each module typescript
- reference application
- starter kit for Angular Module

https://angularlicio.us

- blog
- book: Custom Angular Modules
- quick guide PDF
- podcasts

https://angularlicious.teachable.com

- Video Tutorials
- PDF guides

Principles

- DRY (Don’t Repeat Yourself)
- SOLID Principles
- Object-Oriented Principles
  - Objects, classes
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Design Patterns

References

- Brief history of JavaScript Modules
- 10-Minute Module Primer
- https://eloquentjavascript.net/10_modules.html
- What are Javascript modules?
- Core and Shared modules.
- What kinds of modules should I have and how should I use them?
- NEW!!: Angular Universal