Custom Angular Libraries Course

A course on developing, testing, integrating, and publishing custom Angular libraries.

- Custom Angular Libraries Course
  - Getting Started with Libraries
    - Generate a Library Project
    - Workspace Updates When a Library is Generated
    - Library Project Source (`src`)
    - Build an Angular Library
    - Adding a Service to Angular Libraries
    - Using the Library
    - Using Custom Libraries in an Angular Workspace
  - Adding Configuration to a Library
    - Configuration Model
    - Configure Module Configuration
    - Add Service to the Library
    - AppModule Configuration
    - Use the LoggingService
  - Publishing Custom Libraries
    - Update the Package Information
    - Build the Library for Distribution/Publishing
    - Git Repository
    - Emulate Publishing with NPM Link

Getting Started with Libraries

Create a new workspace for developing libraries. This sample will use the Angular 6/Nrwl.io Nx workspace. The Nx workspace is an enhanced environment that extends some of the capabilities of the default Angular 6 workspace.

```
create-nx-workspace getting-started-with-libs --npm-scope=angularlicious
```

The following files are created:

```
CREATE getting-started-with-libs/.prettierrc (25 bytes)
CREATE getting-started-with-libs/angular.json (307 bytes)
CREATE getting-started-with-libs/karma.conf.js (1012 bytes)
CREATE getting-started-with-libs/nx.json (205 bytes)
CREATE getting-started-with-libs/package.json (2485 bytes)
CREATE getting-started-with-libs/README.md (1874 bytes)
CREATE getting-started-with-libs/tsconfig.json (423 bytes)
CREATE getting-started-with-libs/tslint.json (2307 bytes)
CREATE getting-started-with-libs/.editorconfig (245 bytes)
CREATE getting-started-with-libs/.gitignore (503 bytes)
CREATE getting-started-with-libs/apps/.gitkeep (1 bytes)
CREATE getting-started-with-libs/libs/.gitkeep (0 bytes)
CREATE getting-started-with-libs/tools/tsconfig.tools.json (254 bytes)
CREATE getting-started-with-libs/tools/schematics/.gitkeep (0 bytes)
```

Generate a Library Project

Use the following command to generate a new project of type `library`. There are (2) project types supported by the Angular Workspace.

```
ng generate lib logging --publishable
```

> Note the use of the `--publishable` option. This is an exclusive feature of the `Nx` Workspace from https://www.nrwl.io.

The output of the CLI command:

```
CREATE libs/logging/karma.conf.js (484 bytes)
CREATE libs/logging/ng-package.json (157 bytes)
CREATE libs/logging/package.json (184 bytes)
CREATE libs/logging/tsconfig.lib.json (740 bytes)
CREATE libs/logging/tsconfig.spec.json (259 bytes)
CREATE libs/logging/tslint.json (269 bytes)
CREATE libs/logging/src/test.ts (700 bytes)
CREATE libs/logging/src/index.ts (55 bytes)
CREATE libs/logging/src/lib/logging.module.ts (252 bytes)
CREATE libs/logging/src/lib/logging.module.spec.ts (431 bytes)
UPDATE angular.json (1392 bytes)
UPDATE package.json (2614 bytes)
UPDATE nx.json (248 bytes)
UPDATE tsconfig.json (502 bytes)
```

Workspace Updates When a Library is Generated

The following `devDependencies` are added to the workspace to support building the library using `ng-packagr`.

```
"@angular-devkit/build-ng-packagr": "~0.10.0",
"ng-packagr": "^4.2.0",
"tsickle": ">=0.29.0",
"tslib": "^1.9.0",
```

tsconfig.json

```
"paths": {
    "@angularlicious/logging": [
    "libs/logging/src/index.ts"
    ]
}
```

angular.json

```
 "projects": {
    "logging": {
      "root": "libs/logging",
      "sourceRoot": "libs/logging/src",
      "projectType": "library",
      "prefix": "angularlicious",
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-ng-packagr:build",
          "options": {
            "tsConfig": "libs/logging/tsconfig.lib.json",
            "project": "libs/logging/ng-package.json"
          }
        },
        "test": {
          "builder": "@angular-devkit/build-angular:karma",
          "options": {
            "main": "libs/logging/src/test.ts",
            "tsConfig": "libs/logging/tsconfig.spec.json",
            "karmaConfig": "libs/logging/karma.conf.js"
          }
        },
        "lint": {
          "builder": "@angular-devkit/build-angular:tslint",
          "options": {
            "tsConfig": [
              "libs/logging/tsconfig.lib.json",
              "libs/logging/tsconfig.spec.json"
            ],
            "exclude": [
              "**/node_modules/**"
            ]
          }
        }
      }
    }
  },
```

Anatomy of an Angular Library

Notice the package contains a `peerDependencies` section. The `dependencies` and/or the `devDependencies` are contained in the workspace `package.json`. You will need to specify any `peer` dependencies using this section in the library source - this will provide consumers of your library a message regarding the dependencies when the `npm install` the library (if published).

package.json

```
{
    "name": "@angularlicious/logging",
    "version": "0.0.1",
        "peerDependencies": {
        "@angular/common": "^6.0.0-rc.0 || ^6.0.0",
        "@angular/core": "^6.0.0-rc.0 || ^6.0.0"
        }
}
```

ng-package.json

```
{
  "$schema": "../../node_modules/ng-packagr/ng-package.schema.json",
  "dest": "../../dist/libs/logging",
  "lib": {
    "entryFile": "src/index.ts"
  }
}
```

tsconfig.lib.json

```
{
  "extends": "../../tsconfig.json",
  "compilerOptions": {
    "outDir": "../../dist/out-tsc/libs/logging",
    "target": "es2015",
    "module": "es2015",
    "moduleResolution": "node",
    "declaration": true,
    "sourceMap": true,
    "inlineSources": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "importHelpers": true,
    "types": [],
    "lib": [
      "dom",
      "es2018"
    ]
  },
  "angularCompilerOptions": {
    "annotateForClosureCompiler": true,
    "skipTemplateCodegen": true,
    "strictMetadataEmit": true,
    "fullTemplateTypeCheck": true,
    "strictInjectionParameters": true,
    "enableResourceInlining": true
  },
  "exclude": [
    "src/test.ts",
    "**/*.spec.ts"
  ]
}
```

Library Project Source (`src`)

All libraries need an entry point. The `index` barrel file contains a list of all items in the library that are accessible by consumers of the library. If it *ain't* in this file, no one will see it. By default the template names this file `index`.

index.ts

```
export * from './lib/logging.module';
```

The `@NgModule` for the library.

logging.module.ts

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

@NgModule({
  imports: [CommonModule]
})
export class LoggingModule {}
```

Build an Angular Library

Build a specified library by using the `npm run build` command with the `--project` option to specify the project that you want to build. It is going to kick-off the `ng-build` for the specified project.

```
npm run build --project=logging
```

The configuration for the project in the `angular.json` contains the `build` information. Notice, that the library project is setup to use the `@angular-devkit/build-ng-packagr:build` package with the command to `build`. Remember, the library project contains additional `ng-packagr` configuration information.

```
"build": {
    "builder": "@angular-devkit/build-ng-packagr:build",
    "options": {
    "tsConfig": "libs/logging/tsconfig.lib.json",
    "project": "libs/logging/ng-package.json"
    }
},
```

The `ng-packagr` outputs the build to the specified `outDir` as defined in the library's `ng-package.json` file. It uses the Angular Package Format guidelines to create builds for different types of consumers. Things to note:

- uses the npm scope name for the entry point
- uses the `ngc` compiler
- creates bundles for different consumer types
  - FESM2015
  - FESM5
  - UMD with minification
- cleans up the `package.json` file - required to publishing to npm
- outputs to destination directory

```
Building Angular Package
Building entry point '@angularlicious/logging'
Compiling TypeScript sources through ngc
Bundling to FESM2015
Bundling to FESM5
Bundling to UMD
Minifying UMD bundle
Copying declaration files
Writing package metadata
Removing scripts section in package.json as it's considered a potential security
vulnerability.
Built @angularlicious/logging
Built Angular Package!
 - from: D:\development\github\custom-angular-libraries-course\getting-started-
with-libs\libs\logging
 - to:   D:\development\github\custom-angular-libraries-course\getting-started-
with-libs\dist\libs\loggingng
```

> Use the `ngc` compiler against the library directly to get more precise error messages if you require more details.

Adding a Service to Angular Libraries

Add a new `service` to the library project. Angular services are `@Injectable` - therefore, the library should be responsible for providing a service for any applications. Consumers of the library should do this.

```
ng generate service -help
ng generate service --name=logging --project=logging --spec=false --dry-run
```

The new service should be added to the library's `index.ts` file (manifest) to indicate it is accessible to consumers.

index.ts

```
export * from './lib/logging.module';
export * from './lib/logging.service';
```

The CLI generates a service for the library. However, it isn't doing much at the moment. Empty constructor, no methods, and no public properties.

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class LoggingService {

  constructor() { }
}
```

The easiest implementation for demonstration purposes is to add a `log()` method to `console.log` a string message sent in by consumers of the library.

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class LoggingService {

  constructor() { }

  public log(message: string) {
    console.log(message);
  }
}
```

Using the Library

Create a new `application` project to consume/use the `library` project.

```
ng generate application --help # to see options for application project
ng generate application logging-consumer --routing --style=scss --dry-run # remote
--dry-run to add/update workspace files
```

```
CREATE apps/logging-consumer-e2e/protractor.conf.js (752 bytes)
CREATE apps/logging-consumer-e2e/tsconfig.e2e.json (247 bytes)
CREATE apps/logging-consumer-e2e/src/app.e2e-spec.ts (312 bytes)
CREATE apps/logging-consumer-e2e/src/app.po.ts (219 bytes)
CREATE apps/logging-consumer/browserslist (388 bytes)
CREATE apps/logging-consumer/karma.conf.js (493 bytes)
CREATE apps/logging-consumer/tsconfig.app.json (229 bytes)
CREATE apps/logging-consumer/tsconfig.spec.json (292 bytes)
CREATE apps/logging-consumer/tslint.json (269 bytes)
CREATE apps/logging-consumer/src/favicon.ico (5430 bytes)
CREATE apps/logging-consumer/src/index.html (324 bytes)
CREATE apps/logging-consumer/src/main.ts (372 bytes)
CREATE apps/logging-consumer/src/polyfills.ts (3234 bytes)
CREATE apps/logging-consumer/src/test.ts (642 bytes)
CREATE apps/logging-consumer/src/styles.scss (80 bytes)
CREATE apps/logging-consumer/src/assets/.gitkeep (0 bytes)
CREATE apps/logging-consumer/src/environments/environment.prod.ts (51 bytes)
CREATE apps/logging-consumer/src/environments/environment.ts (662 bytes)
CREATE apps/logging-consumer/src/app/app.module.ts (485 bytes)
CREATE apps/logging-consumer/src/app/app.component.html (602 bytes)
CREATE apps/logging-consumer/src/app/app.component.spec.ts (1103 bytes)
CREATE apps/logging-consumer/src/app/app.component.ts (232 bytes)
CREATE apps/logging-consumer/src/app/app.component.scss (0 bytes)
UPDATE angular.json (5665 bytes)
UPDATE package.json (2614 bytes)
UPDATE nx.json (352 bytes)
```

Using Custom Libraries in an Angular Workspace

If you are consuming libraries that are part of the Angular Workspace project list, you do not need to go through the process of building, publishing, and installing the libraries for a consumer application. The Angular Workspace simplifies this developer workflow. Typically, we would attempt to install a desired package from npm - this package may or may not have a scope. I like scopes. The sample libraries in this workspace use the scope name `angularlicious`.

When we want to use the `logging` library with the scope name, we use `@angularlicious/logging`. This is familiar to common usage of packages. However, in the workspace, we can reference the packages using the scope and library name without *installing* any packages. Since the library projects are already part of the Angular Workspace it is not necessary.

Benefits:

- Tree Shaking
- Simplified Developer Workflow

How is this possible? When `library` projects are generated. The workspace `tsconfig.json` is updated with path mappings in the `path` section. It basically points the scoped library name to the actual entry point of the library source code .

```
  "paths": {
       "@angularlicious/logging": [
         "libs/logging/src/index.ts"
       ]
     }
```

This allows the consumer application to point to the library using the scoped name - in our example it is `@angularlicious/logging`. However, during the build process of a consumer (i.e., projects with type `application` or `library`), the builder will point to the correct location of the library. The build process will use the only source from library that is used by the consumer (i.e., tree shaking).

Some benefits:

- smaller bundles
  - only includes code that is actually used by the consumer of the library
- use scoped name as if it were a published library
- consistent usage if the library is either published or used within an Angular workspace.
- provides a shared resource within the workspace
- has the capability to be published to a private/public npm repository

Adding Configuration to a Library

Add a new library to the workspace. This library project will be enabled to provide configuration to your library (module). Many

```
ng generate library logging-with-config --publishable
```

```
CREATE libs/logging-with-config/karma.conf.js (496 bytes)
CREATE libs/logging-with-config/ng-package.json (169 bytes)
CREATE libs/logging-with-config/package.json (196 bytes)
CREATE libs/logging-with-config/tsconfig.lib.json (752 bytes)
CREATE libs/logging-with-config/tsconfig.spec.json (271 bytes)
CREATE libs/logging-with-config/tslint.json (269 bytes)
CREATE libs/logging-with-config/src/test.ts (700 bytes)
CREATE libs/logging-with-config/src/index.ts (67 bytes)
CREATE libs/logging-with-config/src/lib/logging-with-config.module.ts (262 bytes)
CREATE libs/logging-with-config/src/lib/logging-with-config.module.spec.ts (483
bytes)
UPDATE angular.json (6836 bytes)
UPDATE package.json (2690 bytes)
UPDATE nx.json (405 bytes)
UPDATE tsconfig.json (606 bytes)
```

Configuration Model

Add a configuration class.

```
ng generate class loggingConfig --project=logging-config --spec=false --dry-run
```

```
export class LoggingConfig {
    name: string
}
```

> Remember to add the new item to the barrel file (index.ts)

```
export { LoggingConfig } from './lib/logging-config';
```

Configure Module Configuration

Configure the library's module to accept configuration.

- import the configuration class
- add static method `forRoot`
- provide the configuration
  - creates an injectable scoped to the module

```
import { NgModule, ModuleWithProviders } from '@angular/core';
import { CommonModule } from '@angular/common';
import { LoggingConfig } from './logging-config';

@NgModule({
  imports: [CommonModule]
})
export class LoggingWithConfigModule {
  static forRoot(config: LoggingConfig): ModuleWithProviders {
    return {
      ngModule: LoggingWithConfigModule,
      providers: [
        {
          provide: LoggingConfig,
          useValue: config
        }
      ]
    }
  }
}
```

Add Service to the Library

```
import { Injectable } from "@angular/core";
import { LoggingConfig } from "./logging-config";


@Injectable()
export class LoggingWithConfigService {

    name: string;

    constructor(config: LoggingConfig) {
        this.name = config.name;
    }

    /**
     * Use to log information to the console.
     */
    log(message: string) {
        console.log(`${this.name}: ${message} at ${new
Date(Date.now()).toLocaleTimeString()}`);
    }
}
```

> Remember to add the new member to the barrel.

```
export { LoggingWithConfigService } from './lib/logging-with-config.service';
```

AppModule Configuration

Update the application module to load the module with configuration.

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

import { AppComponent } from './app.component';
import { NxModule } from '@nrwl/nx';
import { RouterModule } from '@angular/router';
import { LoggingWithConfigModule } from '@angularlicious/logging-with-config';
import { LoggingWithConfigService } from '@angularlicious/logging-with-config';

const config = {
  name: 'NG-APP-CONFIG'
}

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    NxModule.forRoot(),
    RouterModule.forRoot([], { initialNavigation: 'enabled' }),
    LoggingWithConfigModule.forRoot(config)
  ],
  providers: [
    LoggingWithConfigService
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Use the LoggingService

Using the service from the custom library.

- add imports
- use DI to inject the service - component constructor
- use service in component

```
import { Component, OnInit } from '@angular/core';
import { LoggingWithConfigService } from '@angularlicious/logging-with-config';

@Component({
  selector: 'angularlicious-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {

  title = 'logging-consumer';

  constructor(
    private loggingService: LoggingWithConfigService
  ) {
    this.loggingService.log(`Running constructor from AppComponent`);
  }

  ngOnInit(): void {
    this.loggingService.log(`Running ngOnInit from AppComponent`);
  }
}
```

Publishing Custom Libraries

Preparing the library for publishing requires building for different consumers

Update the Package Information

The `package.json` file contains information about the library. The information is used by the package repository (i.e., see npm).

- name
- version: update the version using npm commands from the `dist` folder before publishing to npm
  - More information at: https://docs.npmjs.com/about-semantic-versioning
  - npm version commands: https://docs.npmjs.com/cli/version
- repository
- license
- peerDependencies

> Note: that the `package.json` doesn't contain `devDependencies` or `dependencies` sections.

Here's is a sample of the @angularlicious/actions published package.

```
{
  "name": "@angularlicious/actions",
  "version": "6.0.0",
  "description": "@angularlicious/actions is a framework to build amazing business
logic. It compliments the @angularlicious/rules-engine.",
  "license": "MIT",
  "keywords": [
    "angular",
    "typescript",
    "business logic",
    "business actions",
    "business rules",
    "angularlicious",
    "validation",
    "javascript rules",
    "Matt Vaughn",
    "Angularlicious.com",
    "Angularlicious"
  ],
  "author": "Matt Vaughn",
  "repository": {
    "type": "git",
    "url": "git+https://github.com/angularlicious/angularlicious.git"
  },
  "peerDependencies": {
    "@angularlicious/rules-engine": "^6.0.0"
  }
}
```

Build the Library for Distribution/Publishing

```
ng build --project=logging-with-config
```

The `angular.json` configuration for the library project contains information about the specified builder.

- using `ng-package.json` and `build-ng-packagr`

```
"build": {
  "builder": "@angular-devkit/build-ng-packagr:build",
  "options": {
    "tsConfig": "libs/logging-with-config/tsconfig.lib.json",
    "project": "libs/logging-with-config/ng-package.json"
  }
},
```

The output of the build from `ng-packagr`:

```
ng build --project=logging-with-configBuilding Angular Package
Building entry point '@angularlicious/logging-with-config'
Compiling TypeScript sources through ngc
Bundling to FESM2015
Bundling to FESM5
Bundling to UMD
Minifying UMD bundle
Copying declaration files
Writing package metadata
Removing scripts section in package.json as it's considered a potential security
vulnerability.
Built @angularlicious/logging-with-config
Built Angular Package!
 - from: D:\development\github\custom-angular-libraries-course\getting-started-
with-libs\libs\logging-with-config
 - to:   D:\development\github\custom-angular-libraries-course\getting-started-
with-libs\dist\libs\logging-with-config
```

Publish to NPM

Publishing to npmjs.com requires that you have an account. Publishing packages with public access is free. However, there is a small fee to use private repositories.

1. Create an account
2. Determine if your account will use a scope

- How to contribute packages to the NPM registry.
- (How to Create a package.json file.
- (Create node.js Modules
- (Package README files
- (Publishing Scoped Packages

If you are publishing a public package with a scope.

> Use command: `npm publish --access public`

Git Repository

Part of you publish workflow should include pushing the same version to your Git repository.

Emulate Publishing with NPM Link

Use the `npm link` command to emulate the process of installing an npm package. The `link` command will push the package to the `node_modules` folder for the workspace.

```
npm link ./dist/libs/logging-with-config
```