Angular Monorepo Setup

Here's some notes on how to develop Angular in a Monorepo, with a specific (non-global) Angular version.

What this means is, we will install the desired version of Angular, specifically for the application, so that multiple versions can exist on our development machine, without interference.

What is A MonoRepo?

A monorepo provides the ability to contain multiple projects in one workspace.

Good references for this pattern: Monorepo Pattern: Setting up Angular workspace for multiple applications in one single repository

Angular Workspaces: Multi-Application Projects

Benefits

Several aspects of a monorepo are different from a single-project structure. Specifically, lots of elements are now shared. This means there are common config files, such as: tsconfig.json, package.json, angular.json, etc. These files would normally sit at the project level, but are now shared by all projects in a monorepo.

The monorepo gets created with a shared node_modules folder. This saves space by not requiring several gigs of disk for each project's copy of referenced libraries. This has the benefit of saving disk space, AND the greater benefit of ensuring consistent dependency versioning when using components between projects.

The monorepo has a shared dist folder, where all library and app builds are published. No direct benefit comes from this. It's just nice to have things in one spot.

A good benefit for a monorepo is that it can leverage a shared assets folder. This is good for projects that use the same icons and images, allowing them all to use a single set of icon and image files, preventing duplication and folder sync problems (for traditional project structures).

Benefits to Library Development

The best benefit for a monorepo is being able to create multiple projects for developing a component library, who all share the same dependencies and rigid pathing. Specifically, this allows you to create multiple projects for a single purpose, like the following:

- Dev Project - A project for developing new components that will be part of a published library
- Library Project - to hold the pristine components. This project is the one that gets directly built and published
- Testing Project - a project that contains all the logic for unit or integration testing of the library

Doing this for each library or app you create, allows you to minimize pollution of your main app or library.

Downside

The downside to using a monorepo, is a little extra complexity in setup (covered by this guide) and the need to explicitly “enter” a project: specifically, navigating into a project folder for editing, calling it by name during builds, etc…

Creating a MonoRepo

Here are steps to create a MonoRepo that uses a local Angular CLI version.

There’s a little bit of indirection with first creating a monorepo. And, that has to do with the creation of the root workspace.
The root workspace is the container of the shared node_modules, projects, Dist, and common config.

Considerations for New Projects

1. See this page for Angular version compatibility, before choosing the version you will use:

Developing in Multiple Angular Versions

2. Once you decide on the version of Angular, you need to ensure your development host includes NVM, to manage your Node.js version.

>

> Follow the instructions, here, to setup NVM: Node.js Version Support

The above link includes instructions for how to setup and swap in the desired version of Node.js.

3. Once NVM is installed and you've had it install the Angular-compatible Node.js version, you can continue.

4. Tell NVM to swap in the compatible version of Node.js.
For example, Angular 17 works with Node.js v20.18.3:

```
nvm use 20.18.3
```

Angular 19 works with this Node.js version:

```
nvm use 22.14.0
```

5. Before creating your project, make sure your development host is using the correct NPM repository.
Currently, we have a NPM repository running on the house build server, that holds all build packages.
As well, it proxies all package requests to the public NPM repository.
So, be sure that your dev host is configured to use it.

> NOTE: See this page for details: Private NPM Repository

If needed, use this command to point all projects to it:

```
npm set registry https://npmrepo.ogsofttech.com:4873
```

To confirm that the house NPM repository is configured, use this:

```
npm config get registry
```

If setup, it will respond with this:



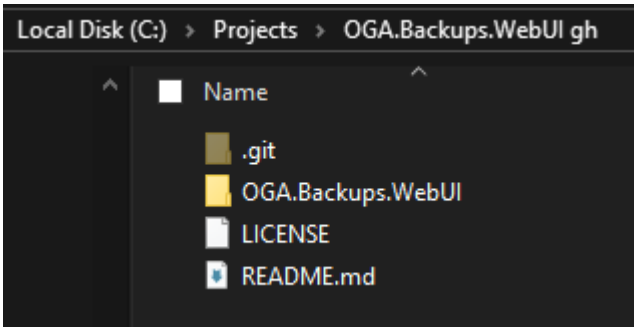
6. Create a baseline GitHub project, and note its url

For example, we have a GH project at: LeeWhite187/OGA.Backups.WebUI

7. Create a folder in your dev host and checkout the GH project to it.

8. Create a subfolder inside the checkout root with the same name as the GH project.

NOTE: This folder will be adjacent to the checked out LICENSE and README.md, like this:



The created folder will hold the dev workspace, assets folder, library projects, etc.

9. Open a command window, and navigate to the created subfolder.

10. Execute this command in the folder, to create monorepo workspace, with an Angular 17 intent:

```
npx @angular/cli@17 new dev-workspace --create-application=false
```

> NOTE: If the above hangs, you may need to update your NPM version, with this:
> `npm install -g npm`

> NOTE: Also, if the above hangs, you can run the command with the '--skip-install' switch, and execute the 'npm install' command separately, to troubleshoot what's wrong.
> Here's what the same command looks like with the skip install switch:
> `npx @angular/cli@17 new dev-workspace --create-application=false --skip-install`
>
> Or, for Angular 19.2.3:
> `npx @angular/cli@19 new dev-workspace --create-application=false --skip-install`

> NOTE: If the above command continues to fail, at the npm install, then the house NPM package service may be down.
>
> You can check on the NPM package repository, here: https://npmrepo.ogsofttech.com:4873/

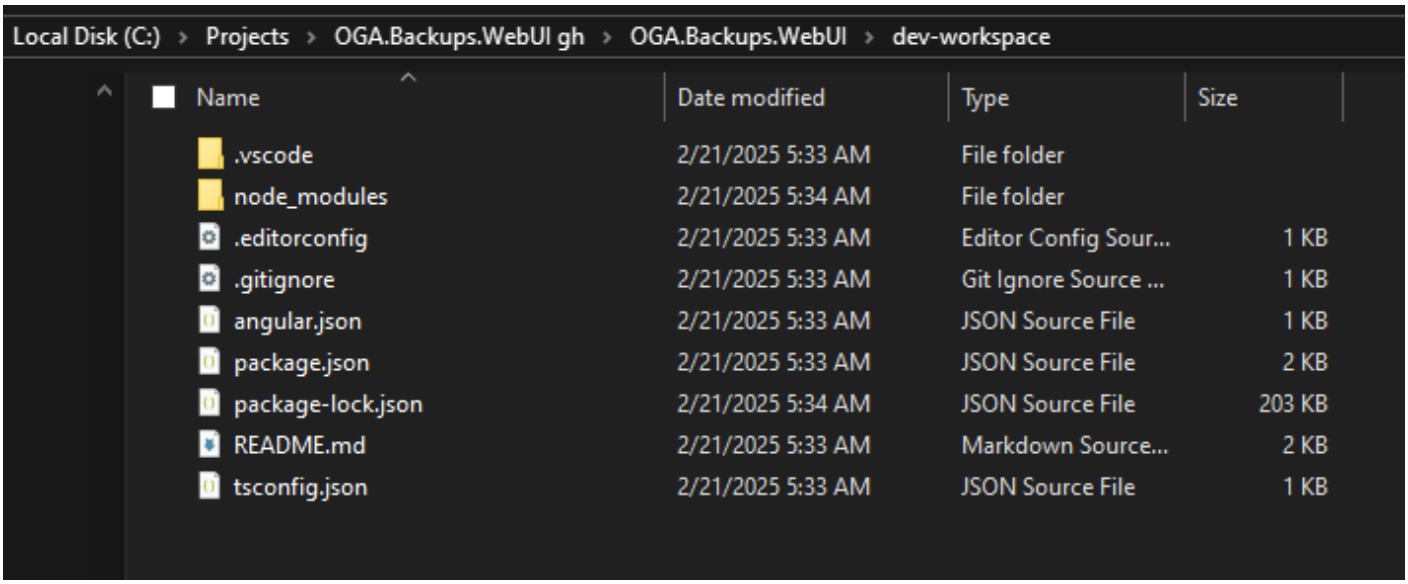
The above creates a root monorepo workspace named, 'dev-workspace', without an application, and preps it for a local Angular 17 (or 19) CLI.

> NOTE: The above command creates a subfolder inside the working directory, with the workspace name.

> NOTE: Specify a different angular cli version if needed.

NPX will churn for a bit, downloading packages and populating the workspace.

11. Once finished, change to the workspace folder (with a cd command).
The workspace will look something like this:



12. Once your command prompt is changed to the workspace folder, you can install the desired Angular version for the monorepo, with this:

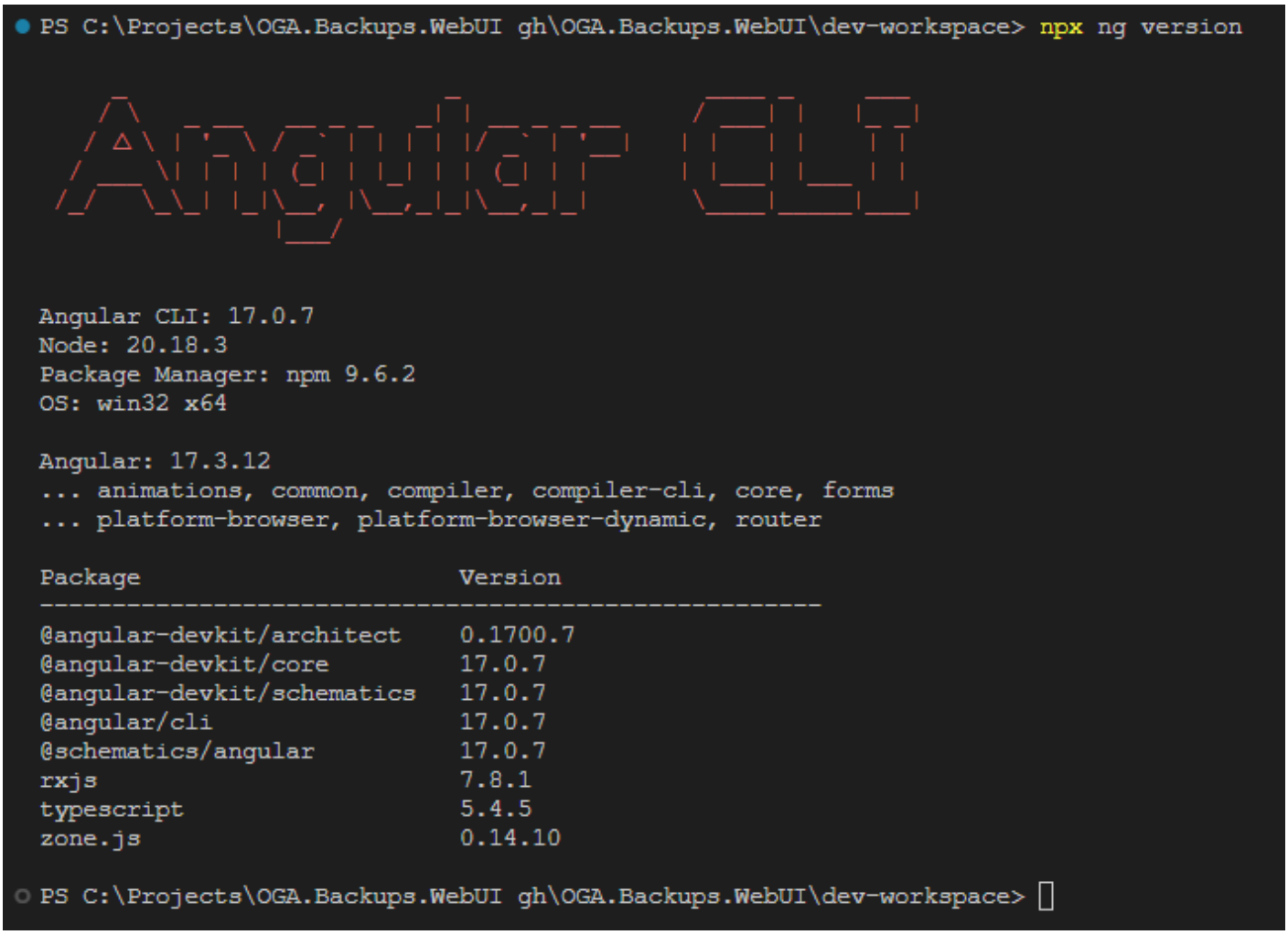
`npm install --save-dev @angular/cli@17.3.10`

For a new Angular 19 project, use this:
`npm install --save-dev @angular/cli@19.2.3`

You can open the workspace folder in VSCode, and continue.

With the workspace is open in a command line, you can verify the local Angular version of the monorepo, with this:

```
npx ng version
```



13. Open the README.md file in the workspace root, and add these lines to it, as a reminder of things you must do each time it is checked out:

> On checkout, you will need to do the following things:
>
> Tell NVM to swap in the needed Node.js version for the project:
>
> nvm use 20.18.3
>
> Pull down any packages needed for building, with this:
>
> npm install

The above includes the command to swap in the Node.js version, compatible with the project's Angular version.
And, the other downloads packages for building.

14. Update the README.md to list usage of npx ng, instead of ng.
This is required, since we are using a local Angular version.

Doing so, means the following changes to ng command usage:

- 'ng serve' becomes 'npx ng serve'
- 'ng generate' becomes 'npx ng generate'
- 'ng build' becomes 'npx ng build'

- 'ng test' becomes 'npx ng test'
- 'ng e2e' becomes 'npx ng e2e'
- 'ng help' becomes 'npx ng help'

15. Enforce local CLI usage, by adding a file to the workspace root, called: .npmrc.
Give it this content:

```
engine-strict=true
```

This will force npm to use the locally installed Angular CLI.

16. Open package.json, and update the scripts to use 'npx ng', instead of 'ng'.
Like this:

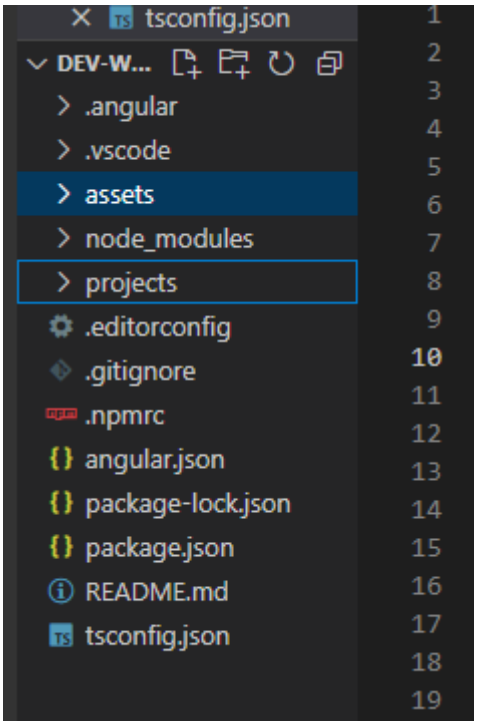
```
  "scripts": {
    "ng": "ng",
    "start": "npx ng serve",
    "build": "npx ng build",
    "watch": "npx ng build --watch --configuration development",
    "test": "npx ng test"
  },
```

17. In order for the projects of your workspace to reference any shared assets or libraries, you must set the baseUrl in the tsconfig.json file, like this:

```
/* To learn more about this file see: https://angular.io/config/tsconfig. */
{
  "compileOnSave": false,
  "compilerOptions": {
    "baseUrl": "./",
    "outDir": "./dist/out-tsc",
    "forceConsistentCasingInFileNames": true,
  ...
```

NOTE: We set the 'baseUrl' property to "./".

18. Read this page to create a shared asset folder in your workspace: Angular Shared Asset Library
But, the asset folder, itself can be created in the workspace root, like this:



The assets folder will hold all shared assets, such as icons, images, and such.

19. It's a good idea to create the projects folder, as well. This is hold all of the applications and libraries of the monorepo.
Create it inside the workspace root, same as the assets folder, above.

Adding Projects

Here's now to create apps and libraries in the monorepo.

Create an App

You can generate applications in the monorepo with this:

> NOTE: Make sure the command window is at the workspace root, when executing this.

```
npx ng generate application admin-app --no-standalone
```

> NOTE: We are using 'npx' instead of 'ng'. This is required

> NOTE: If you are working in Angular 17 or later, and don’t want standalone components, add this: '`--no-standalone`'

The generate command will prompt for the normal application creation things.
Say NO to routing, and choose the SCSS style sheet type.

Common Config

When using a monorepo, there is a common angular.json file. It retains much of the normal content of a single project angular.json file, but instead, has an array for all project entries.

When using a monorepo, there will be cascaded tsconfig.json files. A common tsconfig.json for the workspace, that sets workspace-global things such as the dist folder path (outDir). And, a tsconfig.json in each project that “extends” the common tsconfig, adding paths and shortcuts that are unique to each project.

App Updates for MonoRepo Workspace

For each app created in a monorepo, there are several things that have to be done, so it plays nicely with other apps, across the shared workspace.

- We have to increase the app’s build budget values in the angular.json file.
- We need to promote the app’s tsconfig file to be a “tsconfig.json” file.
- We need to update the paths in the app’s tsconfig.json, because the root tsconfig doesn’t list them.
- We need to update the application’s assets property in the angular.json file.
- We need to copy over in app files and app config from another app project.
- Each app needs its own app library folder, which contains commonly named components that are used by the Common-Library.

Refer to this page for how to do these things: Current Angular App Development

Once your app is setup, you can run it with this: `npx ng serve -- open`

Create a Library

A Monorepo is an ideal pattern for library development, in that the workspace can contain a project for the library, another project for testing the library, and any scratch work projects, and usage demonstrations.

Here’s the corresponding article for standing up a library project: Angular Library Development

> Sidebar: See this Tutorial for Angular Library development: Step-by-step guide to creating your first library in Angular

Creating a library project in your monrepo is slightly different, with this command:

```
npx ng generate library some-new-library --no-standalone
```

> NOTE: Execute the above while at the workspace root.

Once created, the library can be built with: `npx ng build`

Additional Considerations

Here are some special things done to projects in a monorepo so that all apps share components and libraries: Current Angular App Development

See this page for additional workspace setup: Angular Workspace Additional Setup

Initial Checkout

NOTE: Each time you checkout this project from GH, you will need to run the following to set the Node.js version and download packages needed for build and testing:

```
nvm use 20.18.3
npm install
```

The above command swaps in the Node.js version, and reads the `package.json` and `package-lock.json` files and installs everything needed, including the locally-specified Angular CLI version.

> NOTE: If you follows the above setup instructions, these are also in the workspace README.md.

Revision #21
Created 28 January 2025 22:11:34 by glwhite
Updated 4 August 2025 15:52:29 by glwhite