Reusability

Agenda

Workspaces & Libraries

Monorepos using nrwl Nx

Understanding & Intro to Schematics

Web Components

Angular Elements

Dynamic Component Loading

Angular Workspaces

Workspaces

Workspaces allow the development of multiple

- ng Applications and
- ng Libraries

... in a single workspace (folder) created by using the "ng new" CLI Command

Applications & Libs are registered in angular.json

Encourages Modular Angular Development supporting the Micro*-Pattern

- ngElements also supported

Workspaces in angular.json

```
my-workspace/
 ...          (workspace-wide config files)
 projects/      (generated applications and libraries)
   my-first-app/ --(an explicitly generated application)
    ...        --(application-specific config)
    e2e/        ----(corresponding e2e tests)
      src/     ----(e2e tests source)
      ...      ----(e2e-specific config)
    src/        --(source and support files for application)
   my-lib/      --(a generated library)
    ...        --(library-specific config)
    src/        --source and support files for library)
```

Angular Libraries

Why Angular Libraries

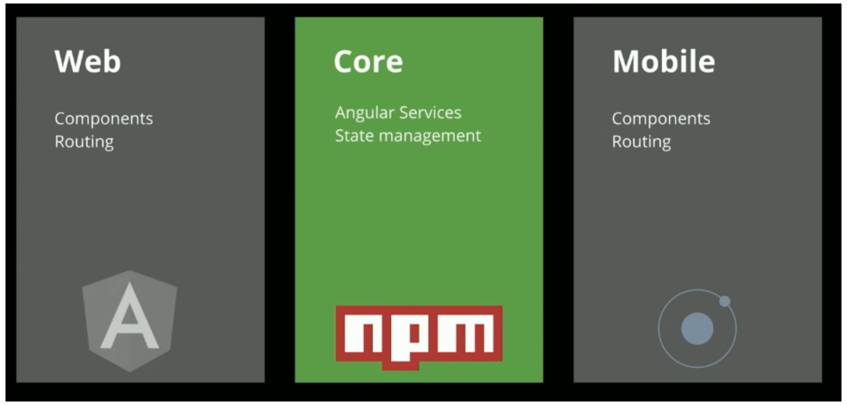
Re-use functionality already implemented in other projects

Allow code sharing between

- Angular - Angular
- Angular - Mobile (Ionic, NativeScript)
  - Code Sharing between Angular, PWA & Mobile covered in a seperate class

Use it for:

- Utilities
- Re-Usable UI Components (Nav, ...)



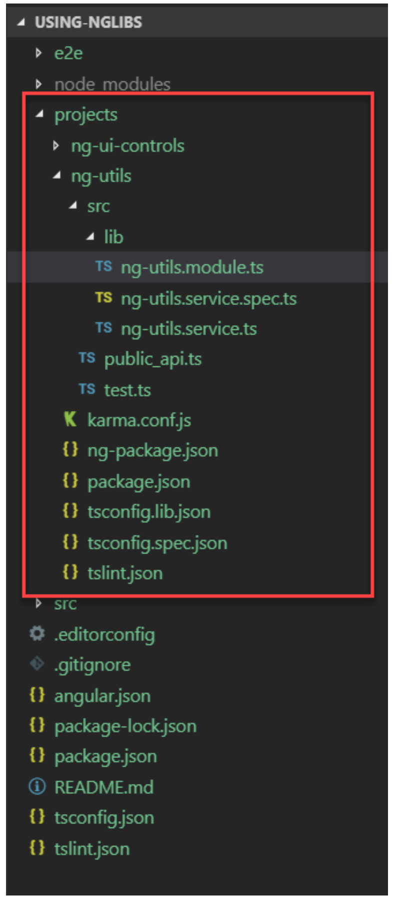
Creating Libraries

Creation:

- ng generate library ng-utils --prefix=my
- ng generate library ng-ui-controls --prefix=my

Adds:

- Project folder in /projects/<projectname>
- package.json, default module
- Sample Component, Service, ...
- public_api.ts -> exports artifacts in the lib
- ng-package.json -> config file for ng-packagr



```
USING-NGLIBS
  e2e
  node_modules
  projects
    ng-ui-controls
    ng-utils
      src
        lib
          ng-utils.module.ts
          ng-utils.service.spec.ts
          ng-utils.service.ts
        public_api.ts
        test.ts
      karma.conf.js
      ng-package.json
      package.json
      tsconfig.lib.json
      tsconfig.spec.json
      tslint.json
  src
  .editorconfig
  .gitignore
  angular.json
  package-lock.json
  package.json
  README.md
  tsconfig.json
  tslint.json
```

What to put in Libs

We use Shared Libs to implements

- Types
- Business Logic
- Utilities
- Constants
- Themes
- Auth-Systems
- ...

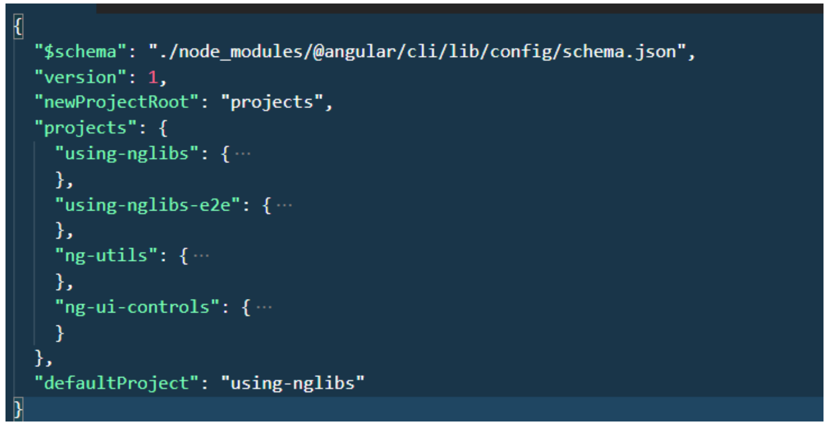
Config Files

angular.json

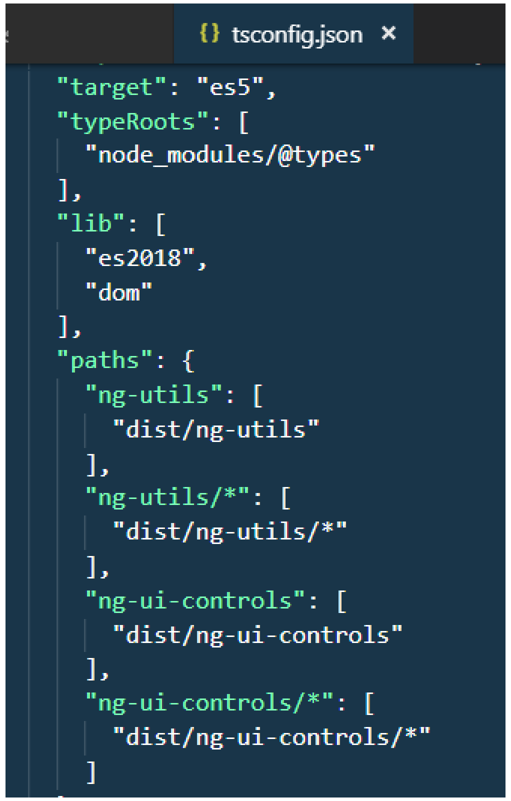
- Reflects the structure of your Workspace
- Adds a default project

tsconfig.json

- Referencs Libs in your "main" project



```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "using-nglibs": {…
    },
    "using-nglibs-e2e": {…
    },
    "ng-utils": {…
    },
    "ng-ui-controls": {…
    }
  },
  "defaultProject": "using-nglibs"
}
```



```
"target": "es5",
"typeRoots": [
  "node_modules/@types"
],
"lib": [
  "es2018",
  "dom"
],
"paths": {
  "ng-utils": [
    "dist/ng-utils"
  ],
  "ng-utils/*": [
    "dist/ng-utils/*"
  ],
  "ng-ui-controls": [
    "dist/ng-ui-controls"
  ],
  "ng-ui-controls/*": [
    "dist/ng-ui-controls/*"
  ]
```

Sections of angular.json

root

- Points to our library project's root folder.

sourceRoot

- Points to root of our library's actual source code.

projectType

- pecifies this is a library as opposed to our other two projects which are of type: application.

prefix

- Identifier that we will use in the selectors of our components

architect

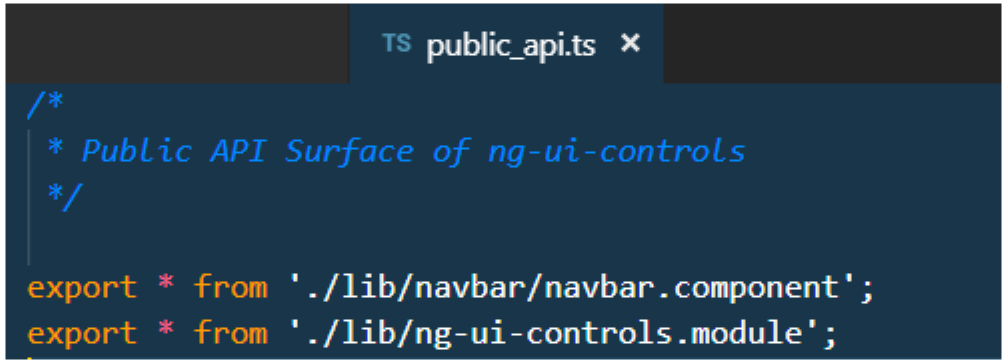
- Instructions for Angular CLI how to handles build, test, and lint

Scaffolding & Exporting Artifacts

In order to Scaffold Artifacts like Components, Services, ... use the CLI:

- Change Path to Lib, or
- ng generate component navbar --project=ng-ui-controls

Export Module & Artifacts in public_api.ts



public_api.ts

```
/*
 * Public API Surface of ng-ui-controls
 */

export * from './lib/navbar/navbar.component';
export * from './lib/ng-ui-controls.module';
```

Building Libraries

The lib's package.json contains peerDependencies

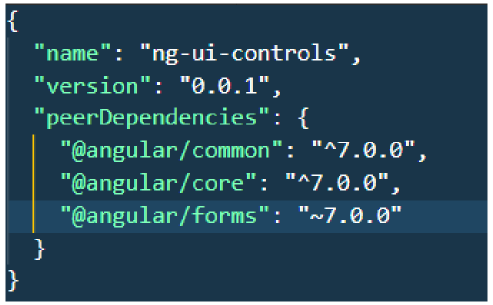
Building Libs requires the name of the lib to build

- ng build ng-ui-controls
- -- watch is supported

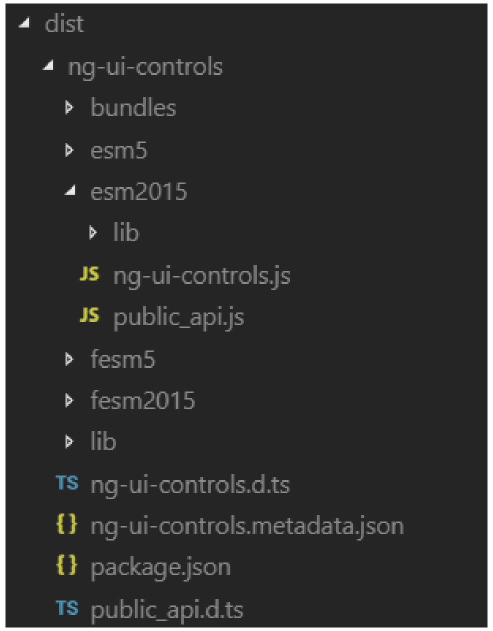
Output is generated to dist/<project> folder

Several Module Formats will be generated to ensure compatibilty

- Typings (*.d.ts) are also generated



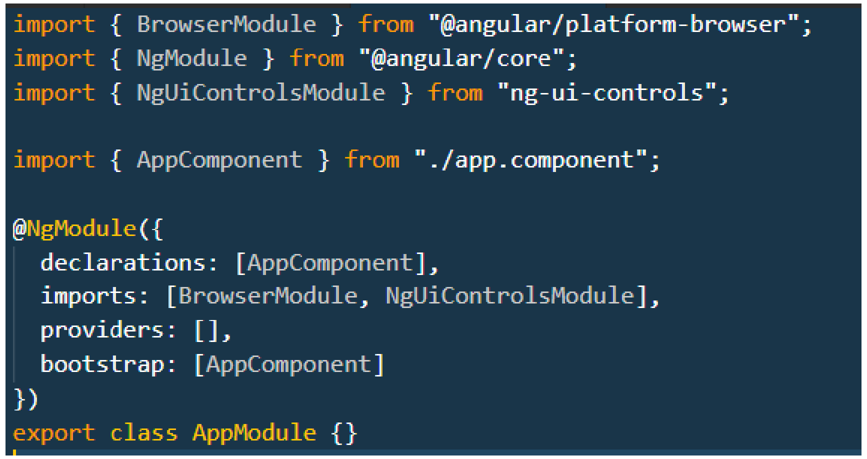
```
{
  "name": "ng-ui-controls",
  "version": "0.0.1",
  "peerDependencies": {
    "@angular/common": "^7.0.0",
    "@angular/core": "^7.0.0",
    "@angular/forms": "~7.0.0"
  }
}
```



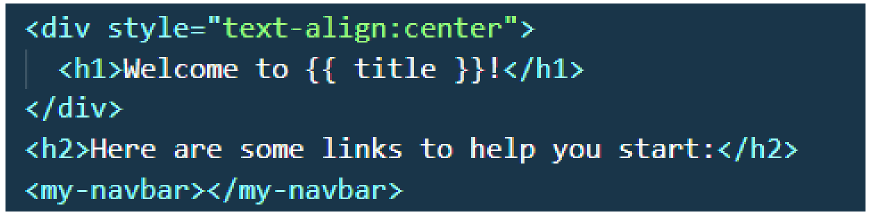
```
dist
  ng-ui-controls
    bundles
    esm5
    esm2015
      lib
      ng-ui-controls.js
      public_api.js
    fesm5
    fesm2015
    lib
    ng-ui-controls.d.ts
    ng-ui-controls.metadata.json
    package.json
    public_api.d.ts
```

Using Libraries

- Can be published to NPM, or
- Used locally
  - Import Artifact
  - Use in Project



```
import { BrowserModule } from "@angular/platform-browser";
import { NgModule } from "@angular/core";
import { NgUiControlsModule } from "ng-ui-controls";

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

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



```
<div style="text-align:center">
  <h1>Welcome to {{ title }}!</h1>
</div>
<h2>Here are some links to help you start:</h2>
<my-navbar></my-navbar>
```

Publish Libraries using NPM

NPM Overview

When publishing NPM packages package.json needs at least

- Name
- Version
- peerDependencies

Modules

Are highly self-contained with distinct functionality, allowing them to be shuffled, removed, or added as necessary, without disrupting the system as a whole

Why use Modules

- Maintainablity -> Working in a Team on the same code
- Namespacing
- Reusability

Module Loaders

Module Loaders facilitate implementing and using Modules

Module Loader implementations are:

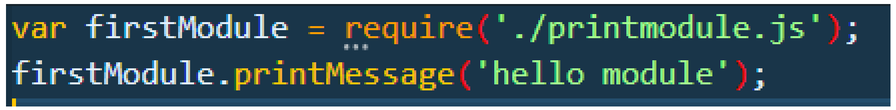
- System JS -> Require JS
- Common JS
- ESM (Ecma Script 6 Modules)

CommonJS

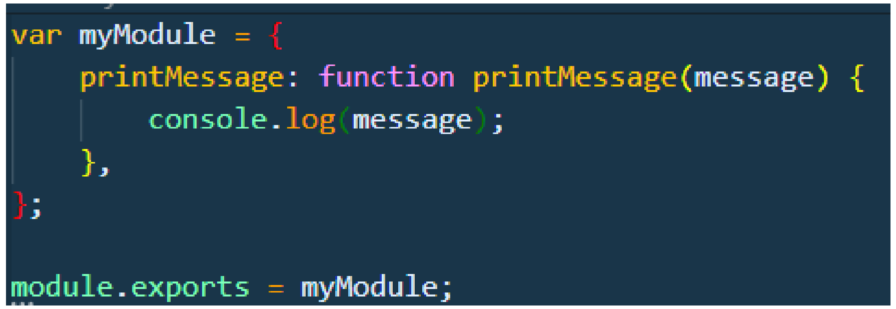
Specification for Exporting & Importing Objects

- Defines two Keywords
  - Exports
  - Require

Module Loader that Node.js used by default



```
var firstModule = require('./printmodule.js');
firstModule.printMessage('hello module');
```

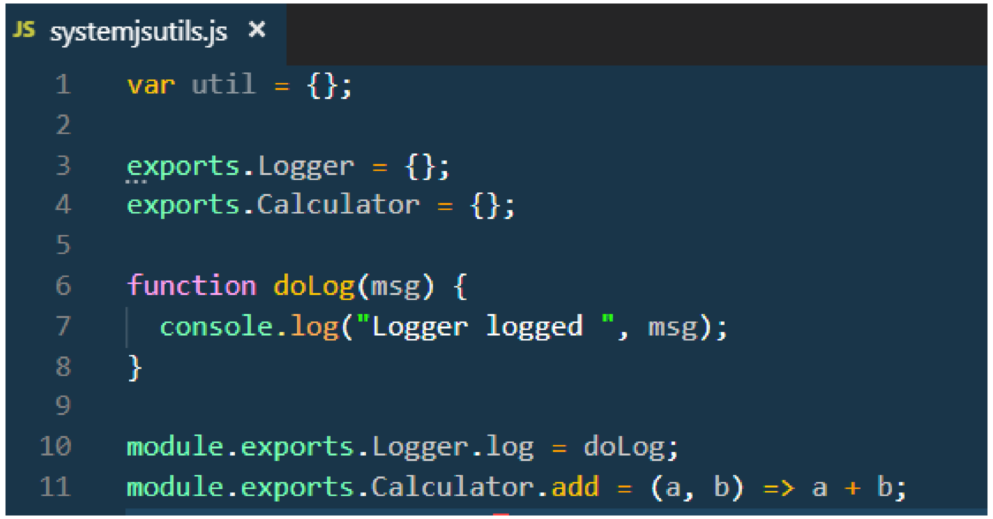


```
var myModule = {
    printMessage: function printMessage(message) {
        console.log(message);
    },
};

module.exports = myModule;
```

System JS

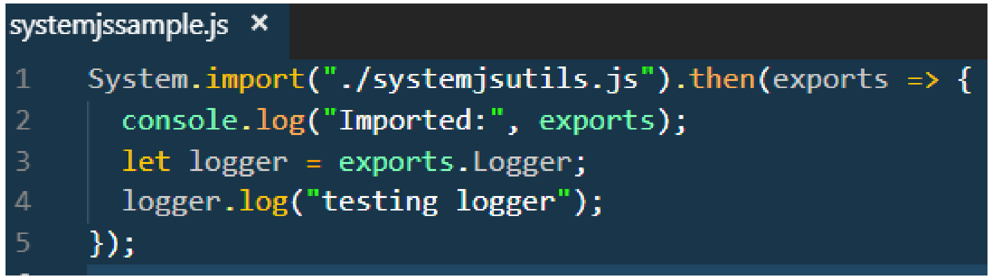
Module Loader compatible with older browsers

Published @ https://github.com/systemjs/systemjs



systemjsutils.js

```js
var util = {};

exports.Logger = {};
exports.Calculator = {};

function doLog(msg) {
  console.log("Logger logged ", msg);
}

module.exports.Logger.log = doLog;
module.exports.Calculator.add = (a, b) => a + b;
```



systemjssample.js

```js
System.import("./systemjsutils.js").then(exports => {
  console.log("Imported:", exports);
  let logger = exports.Logger;
  logger.log("testing logger");
});
```

INTEGRATIONS

ECMAScript Modules (ESM)

Formerly called ES 6 Modules – JavaScript Standard Modules

Consists of named and default exports, support Aliases (x as y)

Consist of two statements

- export / export default
- import

Support cyclic dependencies

- A depends on B, B depends on A

Publish to NPMJS

npm-pack

- Creates a tarball from a package

npm-publish

- Publish a Package
- Can retract for 48 hours

Monorepos using nrwl Nx

What is a Monorepo

A Monorepo (syllabic abbreviation of monolithic repository) is a software development strategy where code for many projects are stored in the same repository

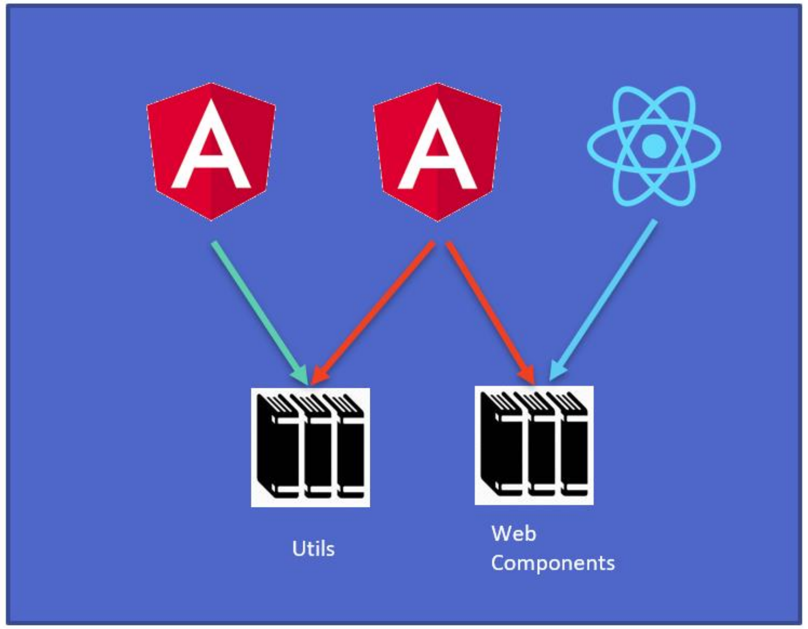
An Update is an "all-or-nothing" process

- Helps to avoid version hell

Advantages:

- Ease of code reuse
- Simplified dependency management
- Easier Code Refactoring

Monorepos are used by Google, Microsoft, FB, Uber, ...



What is Nx

Nx is a set of Extensible Dev Tools for Monorepos

Developed by Narwhal Technologies (nrwl.io)

- Former Angular Core Team Members

With Nx, you can:

- Use modern tools like Cypress, Jest, Prettier, TypeScript, and others with Zero Config
- Use computation cache + code change analysis
  - Efficient build in large projects with many references
- Extend using custom Schematics

Use Cases

Projects with Front End and API

- ie Angular & Express | NestJS
- Shared Model, Interfaces implemented in TS

Share a common Design System / UI Elements

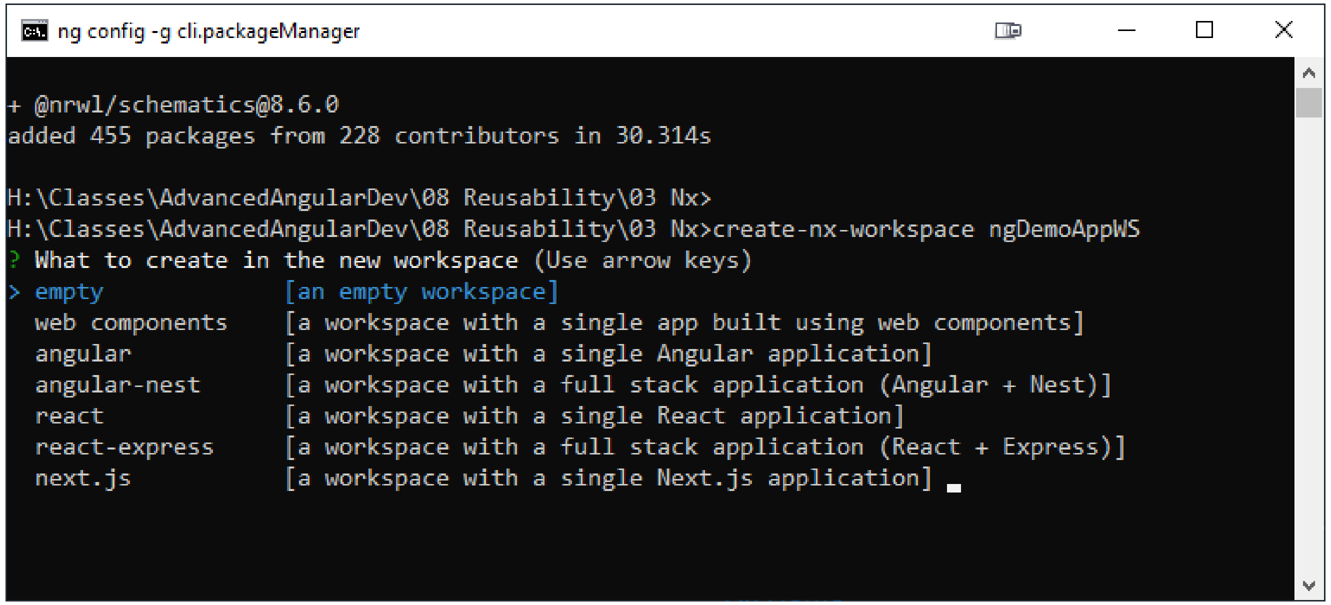
Share Libs between several Apps

- Can use different JavaScript Tech Stacks
  - Typescript
  - Node
  - SPA Frameworks like Angular, React ...

Getting started

Install:

- npm i -g @nrwl/schematics @nrwl/cli
- npx create-nx-workspace ngDemoAppWS



ng config -g cli.packageManager

```
+ @nrwl/schematics@8.6.0
added 455 packages from 228 contributors in 30.314s

H:\Classes\AdvancedAngularDev\08 Reusability\03 Nx>
H:\Classes\AdvancedAngularDev\08 Reusability\03 Nx>create-nx-workspace ngDemoAppWS
? What to create in the new workspace (Use arrow keys)
> empty               [an empty workspace]
  web components      [a workspace with a single app built using web components]
  angular             [a workspace with a single Angular application]
  angular-nest        [a workspace with a full stack application (Angular + Nest)]
  react               [a workspace with a single React application]
  react-express       [a workspace with a full stack application (React + Express)]
  next.js             [a workspace with a single Next.js application]
```

Workspace Structure

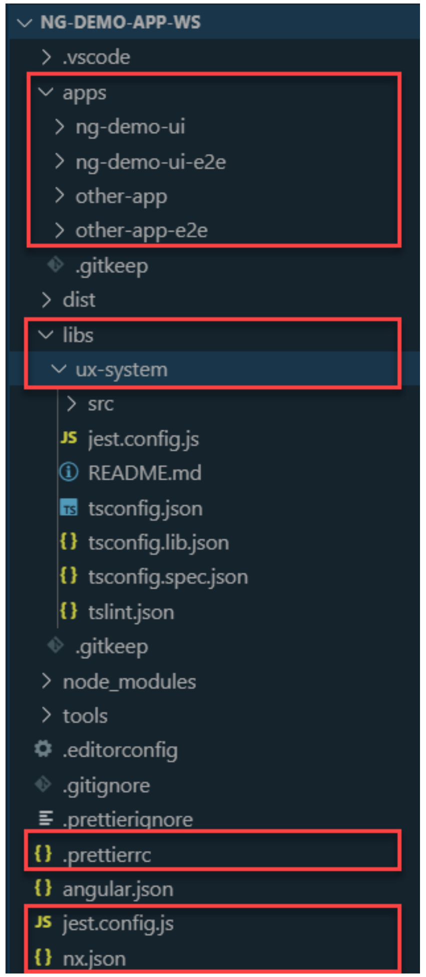
Practically nx re-organizes Angulars Workspaces

- Enables use of React, NestJS, other JS Libs

Often used together with Angular Console VS Code Extension

Divdes into tree main sections

- Apps
- Libs
- Tools -> Schematics



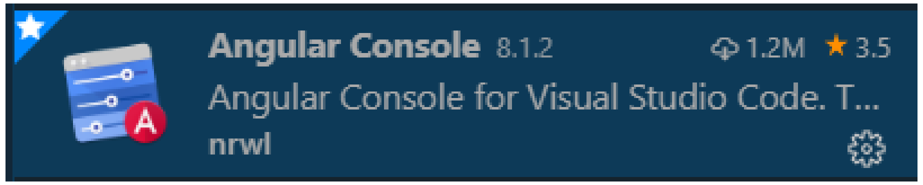
```
NG-DEMO-APP-WS
  > .vscode
  v apps
    > ng-demo-ui
    > ng-demo-ui-e2e
    > other-app
    > other-app-e2e
    .gitkeep
  > dist
  v libs
    v ux-system
      > src
      jest.config.js
      README.md
      tsconfig.json
      tsconfig.lib.json
      tsconfig.spec.json
      tslint.json
    .gitkeep
  > node_modules
  > tools
  .editorconfig
  .gitignore
  .prettierignore
  .prettierrc
  angular.json
  jest.config.js
  nx.json
```

Angular Console

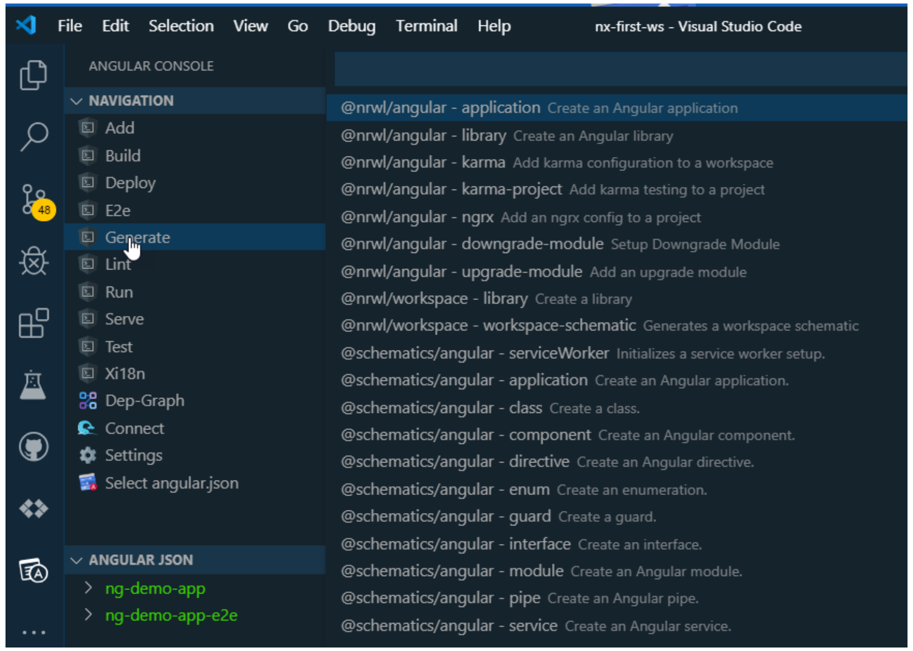
Graphical User Interface for common Angular CLI commands

Lifts the burdon of looking up command line params

Nx compatible - fits with folder layout



Angular Console 8.1.2 1.2M ★ 3.5
Angular Console for Visual Studio Code. T...
nrwl



File Edit Selection View Go Debug Terminal Help nx-first-ws - Visual Studio Code

ANGULAR CONSOLE

NAVIGATION
- Add
- Build
- Deploy
- E2e
- Generate
- Lint
- Run
- Serve
- Test
- Xi18n
- Dep-Graph
- Connect
- Settings
- Select angular.json

ANGULAR JSON
- ng-demo-app
- ng-demo-app-e2e

```
@nrwl/angular - application Create an Angular application
@nrwl/angular - library Create an Angular library
@nrwl/angular - karma Add karma configuration to a workspace
@nrwl/angular - karma-project Add karma testing to a project
@nrwl/angular - ngrx Add an ngrx config to a project
@nrwl/angular - downgrade-module Setup Downgrade Module
@nrwl/angular - upgrade-module Add an upgrade module
@nrwl/workspace - library Create a library
@nrwl/workspace - workspace-schematic Generates a workspace schematic
@schematics/angular - serviceWorker Initializes a service worker setup.
@schematics/angular - application Create an Angular application.
@schematics/angular - class Create a class.
@schematics/angular - component Create an Angular component.
@schematics/angular - directive Create an Angular directive.
@schematics/angular - enum Create an enumeration.
@schematics/angular - guard Create a guard.
@schematics/angular - interface Create an interface.
@schematics/angular - module Create an Angular module.
@schematics/angular - pipe Create an Angular pipe.
@schematics/angular - service Create an Angular service.
```

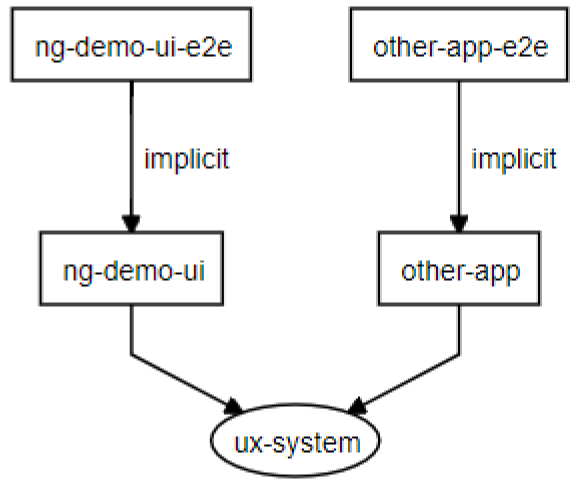
Nx Dependency Graph

Nx automatically creates a Dependency Graph

- nx dep-graph

Nx enables Change Detection between Projects and their Dependencies

- Knows what to rebuild - Visualizes Changes



Angular Schematics

Angular Schematics

A schematic is a template-based code generator that supports complex logic

It is a set of instructions for transforming a software project by generating or modifying code

Schematics are packaged into collections and installed with npm

Actually you are using Schematics all the time when using:

- Angular CLI
- ng add ...

When to use Schematics

Use Schematics when you want to:

- Add Files
  - Scaffolding, Templating
- Update Files
  - Implement a custom ng add ... for your lib
- Extend existing Schematics
- Automate Tasks you find yourself doing over and over
  - Add Jest Config to your project

Common Tasks

Register with NgModules, Components, ...

Update Constructors

Modify configuration

Install Tasks

- ng add

Provide Migrations for code update

Generate any kind of specifiy files

- *.ts, *.scss, *.html
- *.md
- ...

Who uses Schematics?

Angular

- Framework
- CLI

Material

NgRx

Nx

YOU!!!

Base API Elements

Tree

- A staging area for changes / virual Filesystem
- Only commited if all schematics rules run successful

Rule

- A function that is applied to a tree in SchematicContext
- Tree is immutable -> Rule returns a new tree

Schema / Options

- Provide Params to Schematics
- Options can use prompts -> Think of ng new: routing, style or Matierial Theme

Templates

- Parametized Files that acts as a template for things we want to scaffold

Virtual FileSystem / Engine

Virtual FileSystem

- Uses node to create a host
- Represents your filesystem
- Staged operations to your filesystem

Engine

- Controls the creation of a collection and execution of schematic -> Applies changes -> Commit them
- Workflows and scheduled tasks -> Package installation
- Uses the virtual filesystem as a host

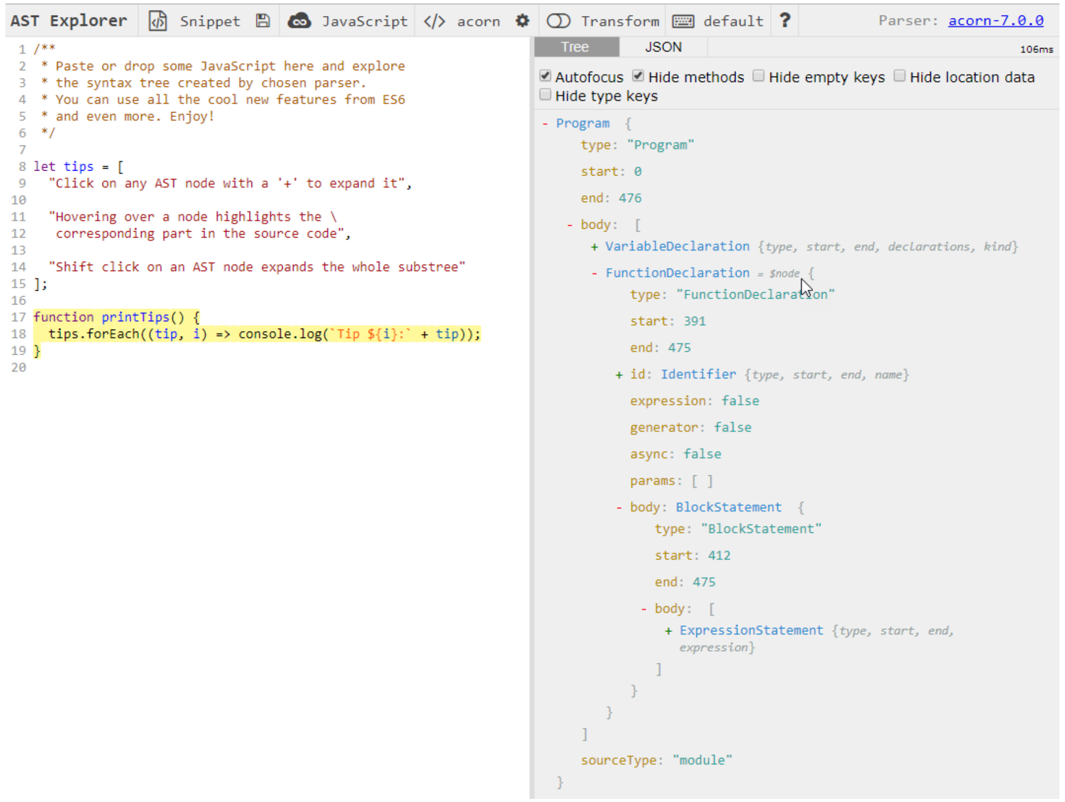
AST

An abstract syntax tree (AST) is a tree representation of the abstract syntactic structure of source code written in a programming language

AST is provided by the TypeScript compiler

AST Explorer helps understanding AST

- https://astexplorer.net/



☑ Autofocus ☑ Hide methods ☐ Hide empty keys ☐ Hide location data
☐ Hide type keys

```javascript
/**
 * Paste or drop some JavaScript here and explore
 * the syntax tree created by chosen parser.
 * You can use all the cool new features from ES6
 * and even more. Enjoy!
 */

let tips = [
  "Click on any AST node with a '+' to expand it",

  "Hovering over a node highlights the \
   corresponding part in the source code",

  "Shift click on an AST node expands the whole substree"
];

function printTips() {
  tips.forEach((tip, i) => console.log(`Tip ${i}:` + tip));
}
```

```
- Program  {
    type: "Program"
    start: 0
    end: 476
  - body:  [
     + VariableDeclaration {type, start, end, declarations, kind}
     - FunctionDeclaration = $node {
          type: "FunctionDeclaration"
          start: 391
          end: 475
        + id: Identifier {type, start, end, name}
          expression: false
          generator: false
          async: false
          params: [ ]
        - body: BlockStatement  {
             type: "BlockStatement"
             start: 412
             end: 475
           - body:  [
              + ExpressionStatement {type, start, end,
                expression}
             ]
          }
       }
    ]
    sourceType: "module"
  }
```

Getting Started.

Install Schematics CLI

- npm i -g @angular-devkit/schematics-cli

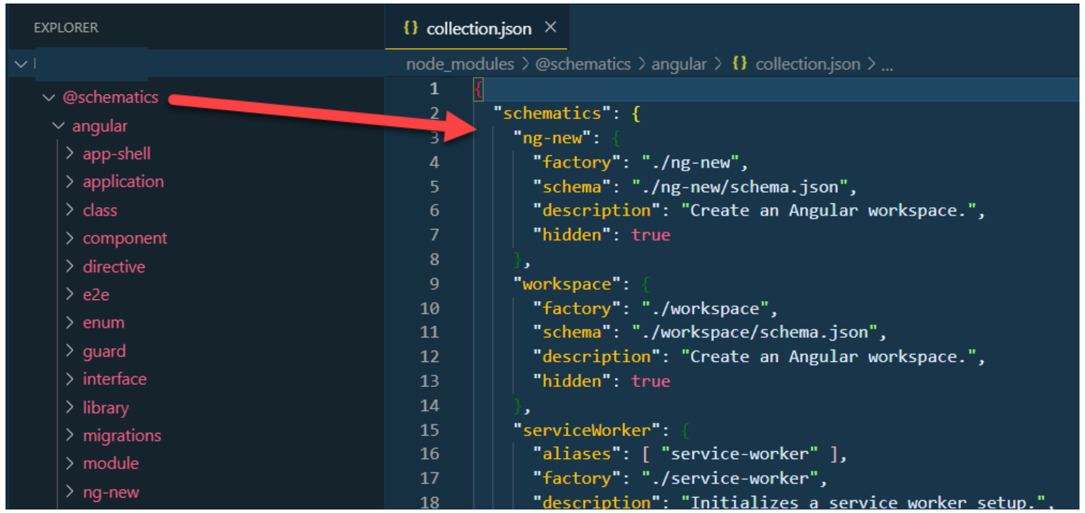
Create new Schematics

- schematics blank --name=hello

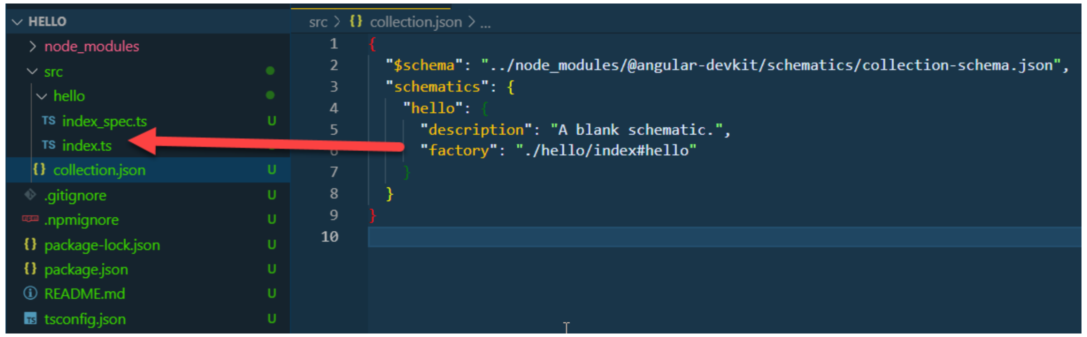
Add a simple rule to it

Add a second Schematic to the same collection

- schematics blank --name=helloparam



```
{
  "schematics": {
    "ng-new": {
      "factory": "./ng-new",
      "schema": "./ng-new/schema.json",
      "description": "Create an Angular workspace.",
      "hidden": true
    },
    "workspace": {
      "factory": "./workspace",
      "schema": "./workspace/schema.json",
      "description": "Create an Angular workspace.",
      "hidden": true
    },
    "serviceWorker": {
      "aliases": [ "service-worker" ],
      "factory": "./service-worker",
      "description": "Initializes a service worker setup.",
```



```
{
  "$schema": "../node_modules/@angular-devkit/schematics/collection-schema.json",
  "schematics": {
    "hello": {
      "description": "A blank schematic.",
      "factory": "./hello/index#hello"
    }
  }
}
```

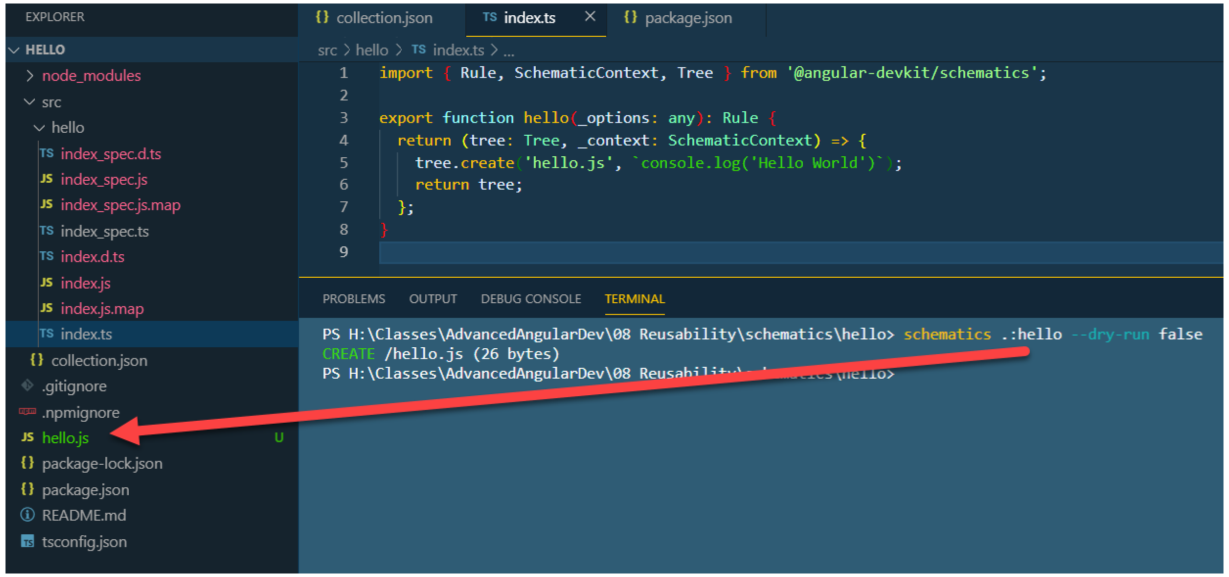
Build & Run Schematics

Build & Run your Schematics

- npm run build
- schematics .:<package-name> [--name <schematics>]
- schematics .:hello
- "." is a pointer to collection.json
  - you can also use a path

Schematics run in debug mode by default

- same as --dry-run



```ts
import { Rule, SchematicContext, Tree } from '@angular-devkit/schematics';

export function hello(_options: any): Rule {
  return (tree: Tree, _context: SchematicContext) => {
    tree.create('hello.js', `console.log('Hello World')`);
    return tree;
  };
}
```

```
PS H:\Classes\AdvancedAngularDev\08 Reusability\schematics\hello> schematics .:hello --dry-run false
CREATE /hello.js (26 bytes)
PS H:\Classes\AdvancedAngularDev\08 Reusability\schematics\hello>
```

Schema

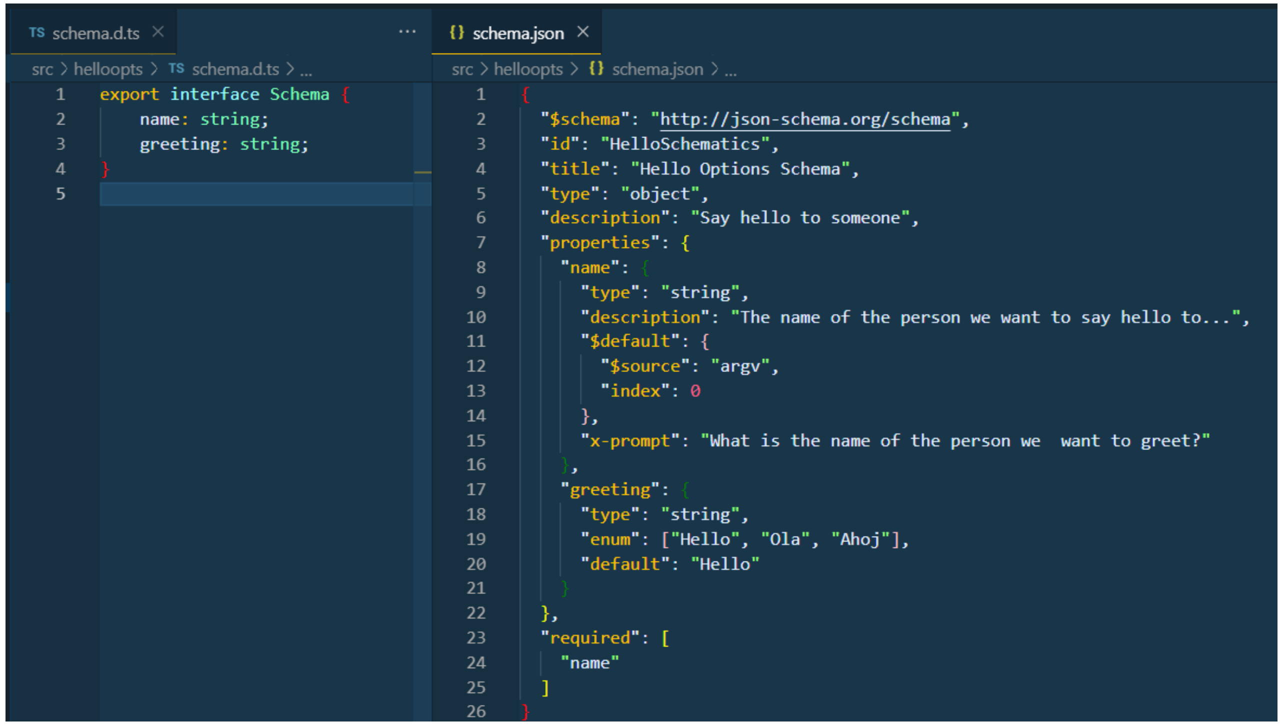
Schema define the params using JSON and TypeScript

Support many data types

- Boolean
- String
- List / Prompts

Missing params can be collected using

- x-prompt



TS schema.d.ts
src > helloopts > TS schema.d.ts > ...
{} schema.json
src > helloopts > {} schema.json > ...

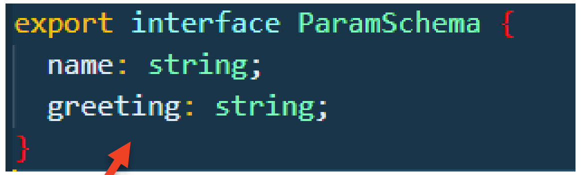
```
export interface Schema {
    name: string;
    greeting: string;
}
```

```
{
  "$schema": "http://json-schema.org/schema",
  "id": "HelloSchematics",
  "title": "Hello Options Schema",
  "type": "object",
  "description": "Say hello to someone",
  "properties": {
    "name": {
      "type": "string",
      "description": "The name of the person we want to say hello to...",
      "$default": {
        "$source": "argv",
        "index": 0
      },
      "x-prompt": "What is the name of the person we  want to greet?"
    },
    "greeting": {
      "type": "string",
      "enum": ["Hello", "Ola", "Ahoj"],
      "default": "Hello"
    }
  },
  "required": [
    "name"
  ]
}
```

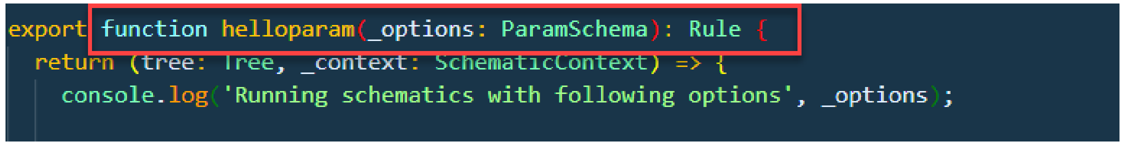
Typed Schema

Typing the Schema is not a Requirement but it helps a lot!

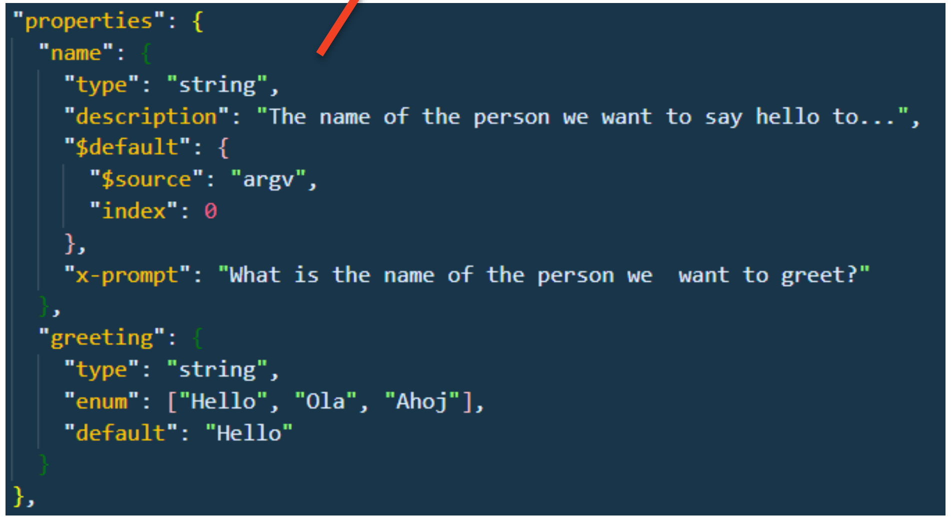
- Code easier to read
- Removes helper vars!



```
export interface ParamSchema {
  name: string;
  greeting: string;
}
```



```
export function helloparam(_options: ParamSchema): Rule {
  return (tree: Tree, _context: SchematicContext) => {
    console.log('Running schematics with following options', _options);
```

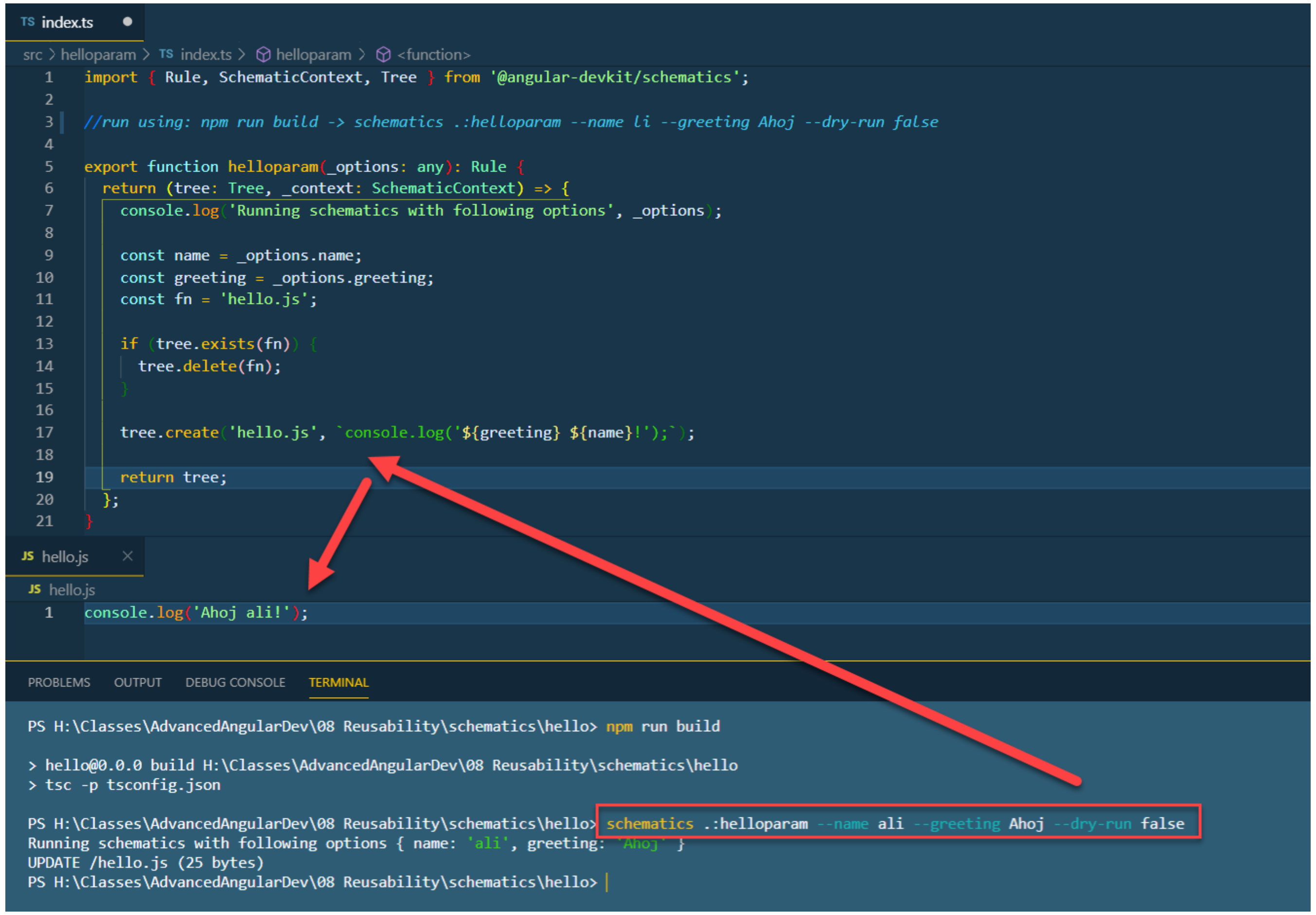


```
"properties": {
  "name": {
    "type": "string",
    "description": "The name of the person we want to say hello to...",
    "$default": {
      "$source": "argv",
      "index": 0
    },
    "x-prompt": "What is the name of the person we  want to greet?"
  },
  "greeting": {
    "type": "string",
    "enum": ["Hello", "Ola", "Ahoj"],
    "default": "Hello"
  }
},
```

File Generation

Files can be created using:

- tree.create
- ng generate
- Templates
  - Static
  - Dynamic



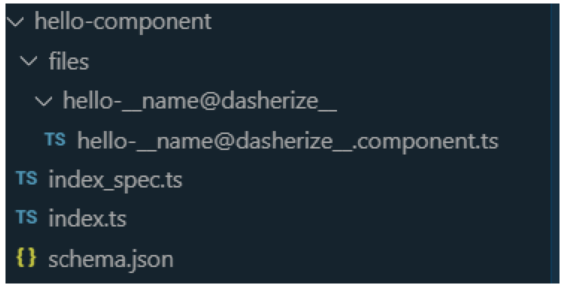
```
import { Rule, SchematicContext, Tree } from '@angular-devkit/schematics';

//run using: npm run build -> schematics .:helloparam --name li --greeting Ahoj --dry-run false

export function helloparam(_options: any): Rule {
  return (tree: Tree, _context: SchematicContext) => {
    console.log('Running schematics with following options', _options);

    const name = _options.name;
    const greeting = _options.greeting;
    const fn = 'hello.js';

    if (tree.exists(fn)) {
      tree.delete(fn);
    }

    tree.create('hello.js', `console.log('${greeting} ${name}!');`);

    return tree;
  };
}
```

```
console.log('Ahoj ali!');
```

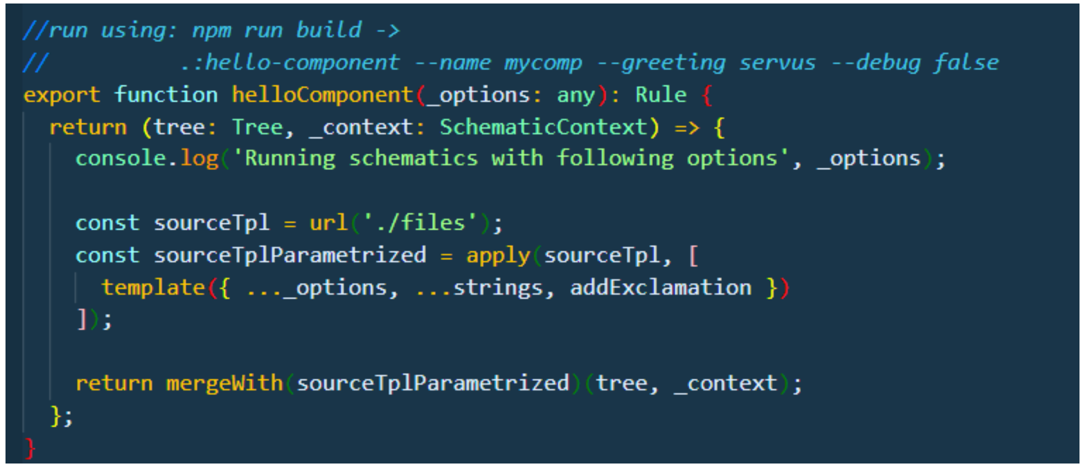
```
PS H:\Classes\AdvancedAngularDev\08 Reusability\schematics\hello> npm run build

> hello@0.0.0 build H:\Classes\AdvancedAngularDev\08 Reusability\schematics\hello
> tsc -p tsconfig.json

PS H:\Classes\AdvancedAngularDev\08 Reusability\schematics\hello> schematics .:helloparam --name ali --greeting Ahoj --dry-run false
Running schematics with following options { name: 'ali', greeting: 'Ahoj' }
UPDATE /hello.js (25 bytes)
PS H:\Classes\AdvancedAngularDev\08 Reusability\schematics\hello>
```

Using Templates

- Template Files will be generated in the ./files folder
  - folder name is fixed
  - excluded from compilation in ts.config
- Importante Charactes / Conventions
  - Filenames
    - __ (double underscore): seperates variables from normal strings
    - @ and dasherize apply variable
      - hello-__name@dasherize__
  - In Code
    - <%= varname %>



```
hello-component
  files
    hello-__name@dasherize__
      hello-__name@dasherize__.component.ts
  index_spec.ts
  index.ts
  schema.json
```

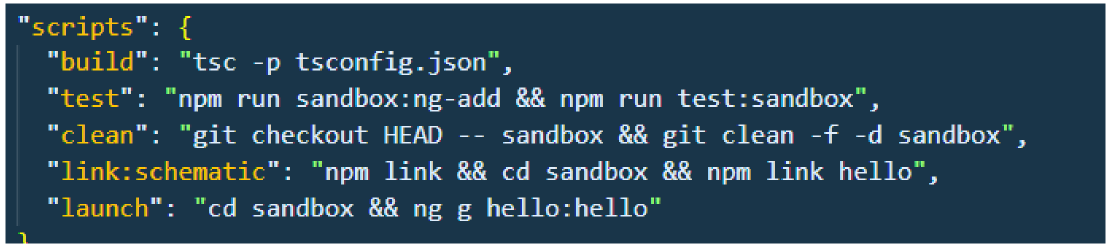


```
//run using: npm run build ->
//          .:hello-component --name mycomp --greeting servus --debug false
export function helloComponent(_options: any): Rule {
  return (tree: Tree, _context: SchematicContext) => {
    console.log('Running schematics with following options', _options);

    const sourceTpl = url('./files');
    const sourceTplParametrized = apply(sourceTpl, [
      template({ ..._options, ...strings, addExclamation })
    ]);

    return mergeWith(sourceTplParametrized)(tree, _context);
  };
}
```

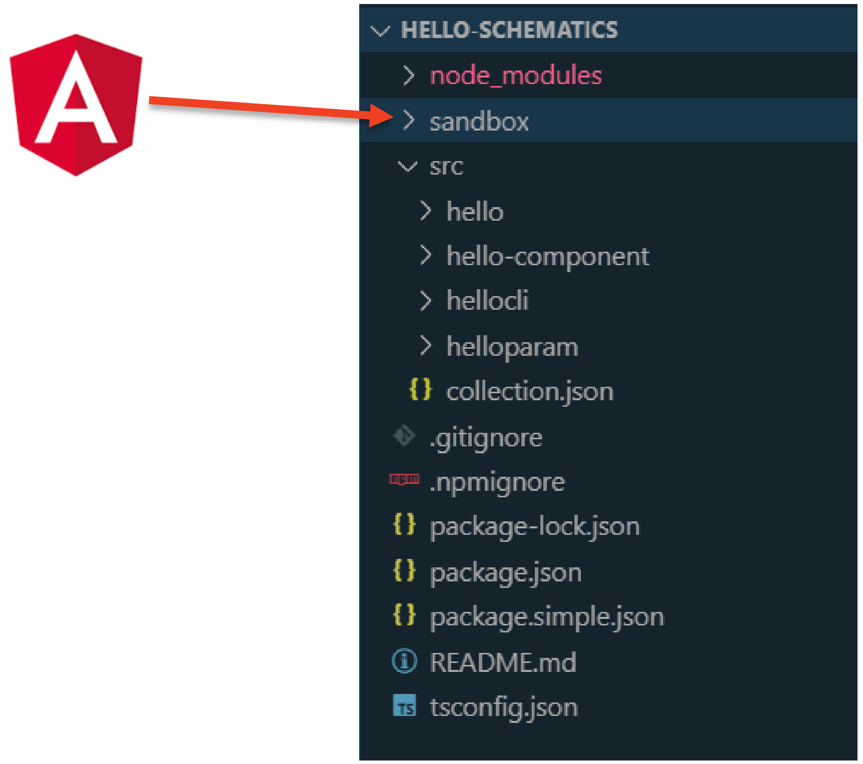
Sandbox

A Sandbox is a version contolled (Angular) project where you can test your Schematics

- It is part of the Schematics project
  - Create using ng new sandbox
- Easy to reset (because of version control)
- Schematics are registered in the Sandbox



```
"scripts": {
  "build": "tsc -p tsconfig.json",
  "test": "npm run sandbox:ng-add && npm run test:sandbox",
  "clean": "git checkout HEAD -- sandbox && git clean -f -d sandbox",
  "link:schematic": "npm link && cd sandbox && npm link hello",
  "launch": "cd sandbox && ng g hello:hello"
}
```



```
HELLO-SCHEMATICS
  > node_modules
  > sandbox
  v src
    > hello
    > hello-component
    > hellocli
    > helloparam
    collection.json
  .gitignore
  .npmignore
  package-lock.json
  package.json
  package.simple.json
  README.md
  tsconfig.json
```

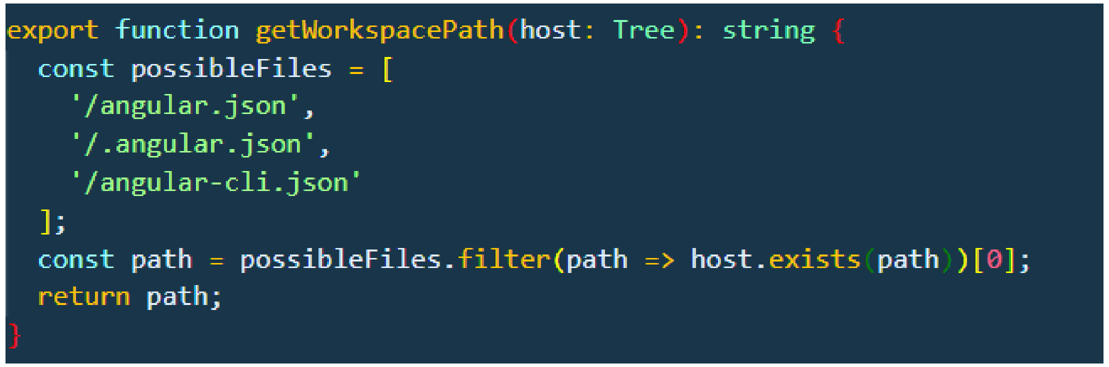
Utils

Creating Schematics often requires the same repeating activities

- getSourcePath
- getWorkspace, getWorkspaceConfig
- addPackageJsonDependency, removePackageJsonDependency
- ...

Create your own Utils Class

- Typically store in "utils" folder
- Extend it over time



```
export function getWorkspacePath(host: Tree): string {
  const possibleFiles = [
    '/angular.json',
    '/.angular.json',
    '/angular-cli.json'
  ];
  const path = possibleFiles.filter(path => host.exists(path))[0];
  return path;
}
```

Web Components

Monolithic Applications

A monolithic Application describes a single-tiered Software application in which the user interface and data access code are combined into a single program from a single platform

Might be implemented using Layers

- Layers are tightly coupled

Problems related to this approach are

- Hard to scale
- Bound to a specific Framework / Technologie (Java, .NET, SharePoint, PHP ...)
- Cost Intensive to migrate / replace

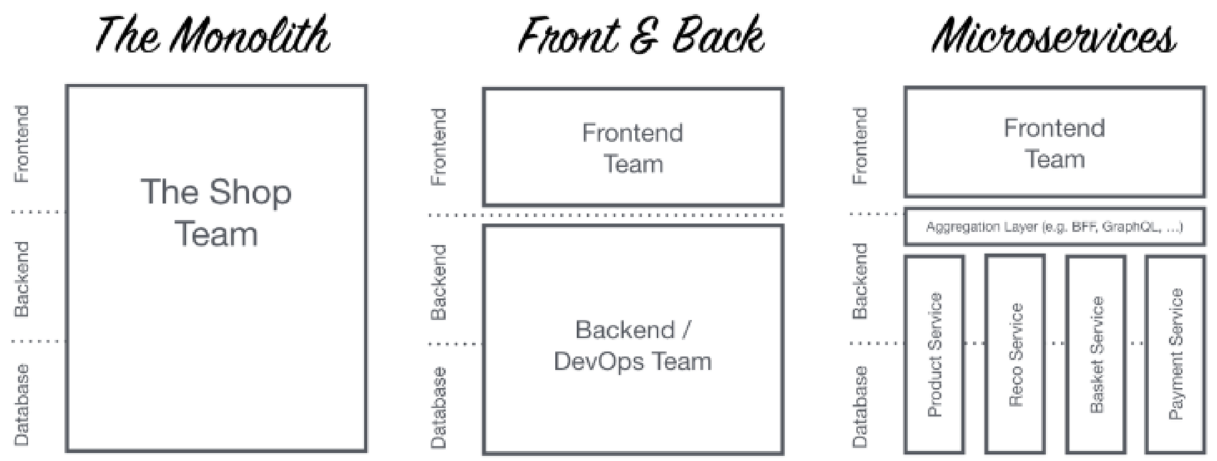
Micro Frontends

Monolithic Applications have been replaced by the Front- & Backend Architecture

- Often still resulting in Frontend Monoliths with the same disadvantages

Micro Frontends are the next step in this Evolution. They are:

- Small Plug & Play Portions of the UI
- Typically hosted by a SPA
- Easy to test & replace
- Can use any Technology - Cloud Ready
- Use Web Component Standard



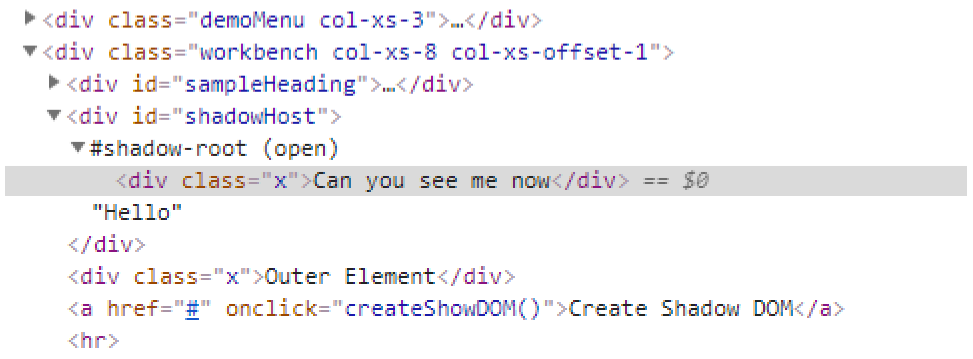
Shadow DOM

Solves the problems related to using Components from multible lib in on HTML Solutions

- Ie. same CSS Style defined in two libs

Implements a seperate DOM Tree for a defined part of the DOM

- Behaves like a seperate DOM Tree



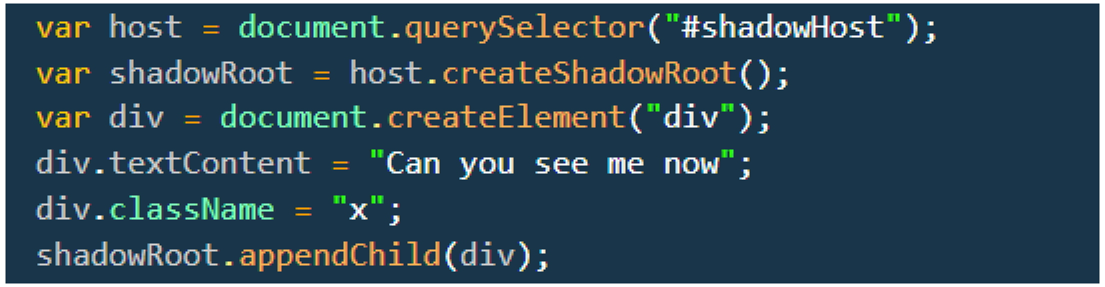
```
▶<div class="demoMenu col-xs-3">…</div>
▼<div class="workbench col-xs-8 col-xs-offset-1">
  ▶<div id="sampleHeading">…</div>
  ▼<div id="shadowHost">
    ▼#shadow-root (open)
        <div class="x">Can you see me now</div> == $0
      "Hello"
  </div>
  <div class="x">Outer Element</div>
  <a href="#" onclick="createShowDOM()">Create Shadow DOM</a>
  <hr>
```

Steps to create a Shadow DOM

Pick a Tag to host the Shadow DOM

Create a Shadow Root

Insert Content into that Shadow Root

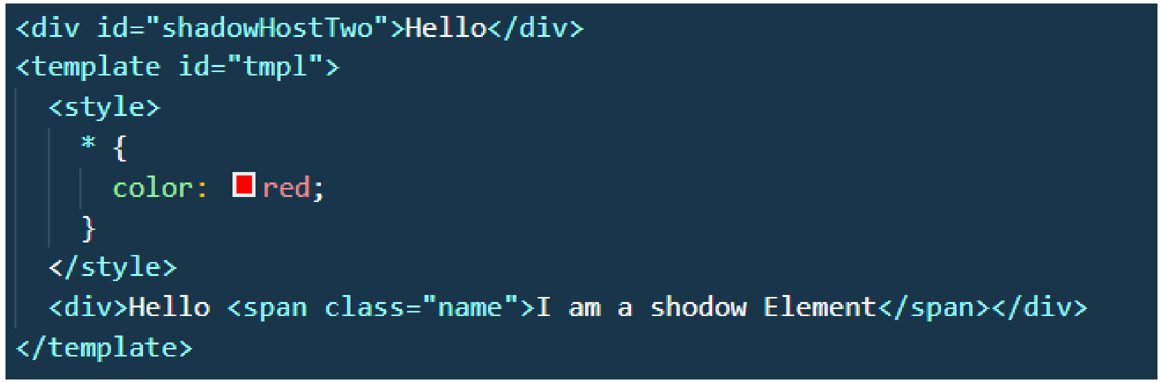


```
var host = document.querySelector("#shadowHost");
var shadowRoot = host.createShadowRoot();
var div = document.createElement("div");
div.textContent = "Can you see me now";
div.className = "x";
shadowRoot.appendChild(div);
```

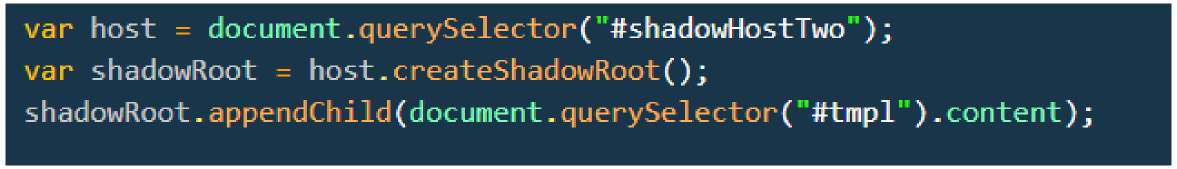
HTML Template Tag

The template element holds HTML code without displaying it

The content can be visible and rendered later by using JavaScript



```
<div id="shadowHostTwo">Hello</div>
<template id="tmpl">
  <style>
    * {
      color: 🟥red;
    }
  </style>
  <div>Hello <span class="name">I am a shodow Element</span></div>
</template>
```



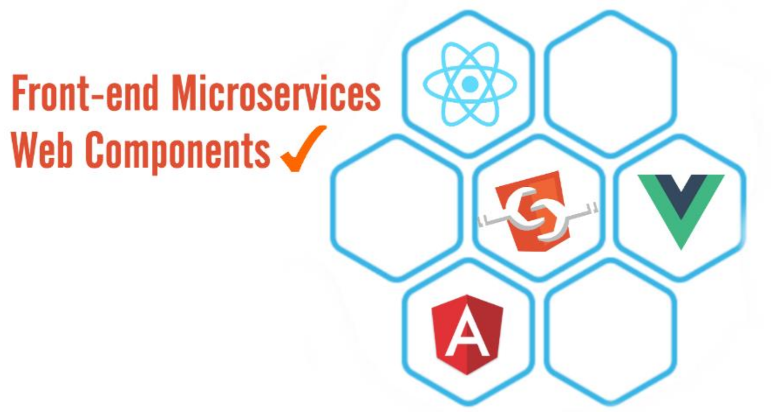
```
var host = document.querySelector("#shadowHostTwo");
var shadowRoot = host.createShadowRoot();
shadowRoot.appendChild(document.querySelector("#tmpl").content);
```

Web Components

Web components are a set of web platform APIs that allow you to create new custom, reusable, encapsulated HTML tags to use in web pages and web apps

They rely on the following Specifications

- Shadow DOM
- HTML Templates
- Custom Elements
- ES Modules



When to use:

When you want to share Code between Apps that are implemented in different Frameworks / Technology

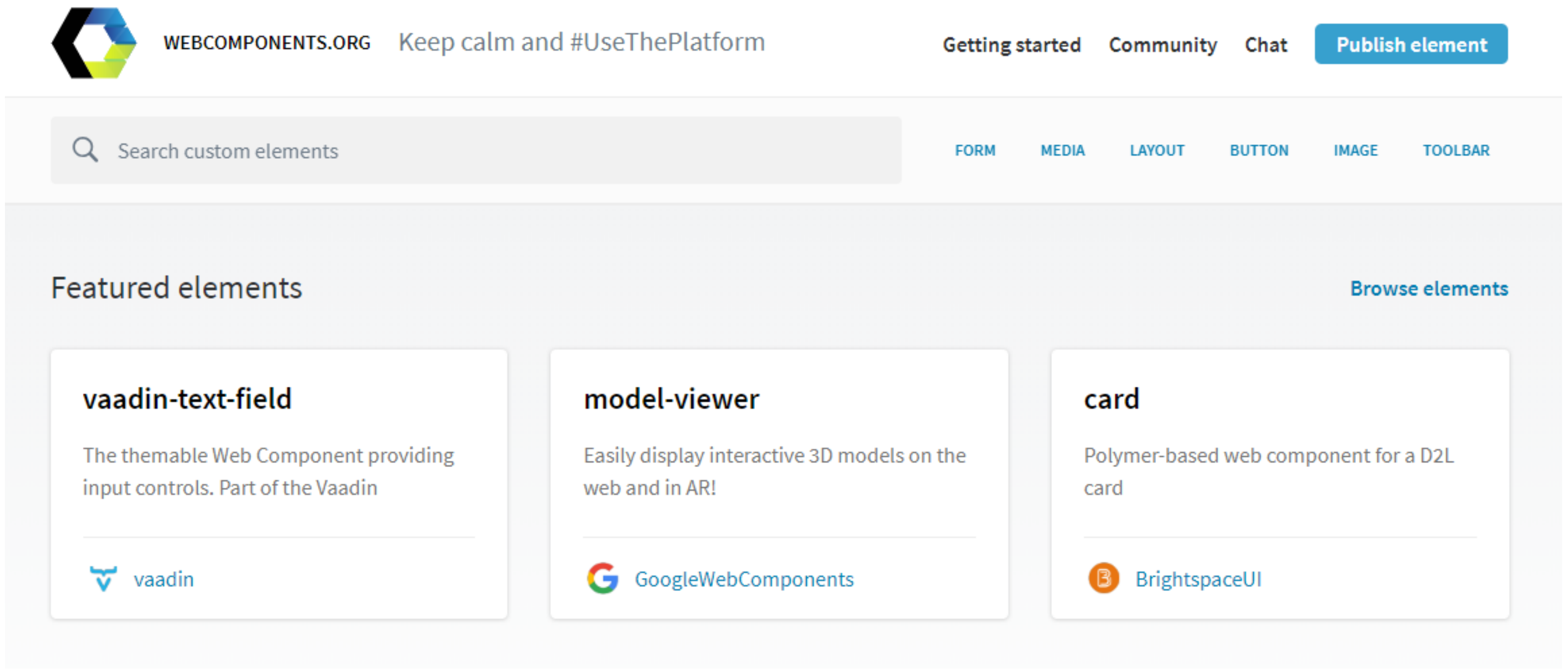
Web Components created by ngElement can be hosted using:

- Pure HTML
- SPA Frameworks: React, Vue.js
- SharePoint / Office 365
- ...

www.webcomponents.org

Site that promotes the use / implementation of Web Components

Getting Starte aviailable



Featured elements

Browse elements

vaadin-text-field

The themable Web Component providing input controls. Part of the Vaadin

vaadin

model-viewer

Easily display interactive 3D models on the web and in AR!

GoogleWebComponents

card

Polymer-based web component for a D2L card

BrightspaceUI

How do I use a web component

Components provide new HTML elements that you can use in your web pages and web applications

Using a custom element is as simple as importing it, and using the new tags in an HTML document

How do I define a new HTML element?

Web Components can be created

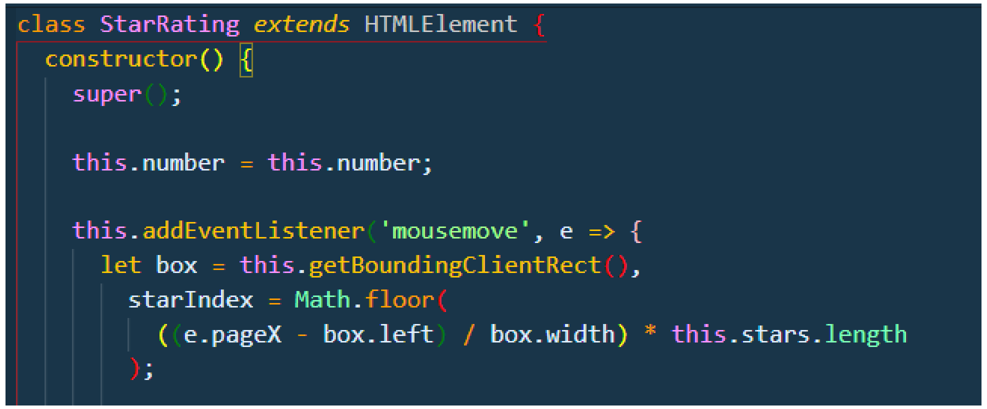
- Manually
- Using a Framework
  - Angualr -> Angular Elements
  - React -> React Web Component
  - Polymer
  - Stencil
  - ...

Steps to Implement

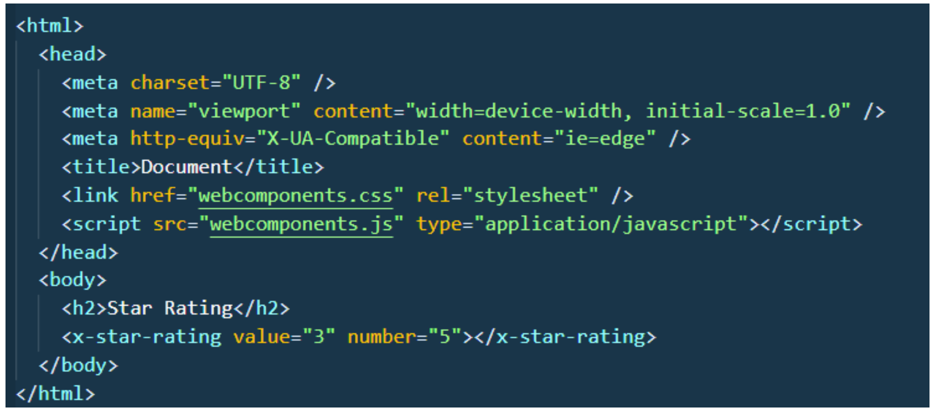
Create Class inheriting vom HTMLElement

Call super() in constructor

Export Custom Element using: customElements.define(„TAG“, CLASS-Name);



```
class StarRating extends HTMLElement {
  constructor() {
    super();

    this.number = this.number;

    this.addEventListener('mousemove', e => {
      let box = this.getBoundingClientRect(),
        starIndex = Math.floor(
          ((e.pageX - box.left) / box.width) * this.stars.length
        );
```



```
<html>
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Document</title>
    <link href="webcomponents.css" rel="stylesheet" />
    <script src="webcomponents.js" type="application/javascript"></script>
  </head>
  <body>
    <h2>Star Rating</h2>
    <x-star-rating value="3" number="5"></x-star-rating>
  </body>
</html>
```

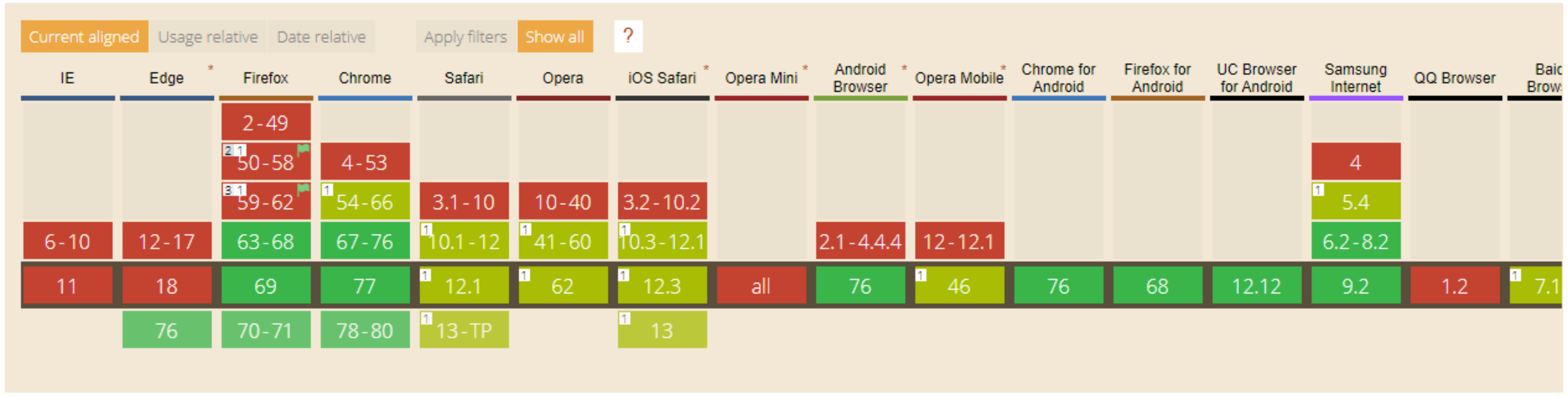
Compatibility

Not all browsers support Web Components

- the "usual suspects" :-)

Polifills available

- npm install @webcomponents/webcomponentsjs



Current aligned | Usage relative | Date relative | Apply filters | Show all | ?

IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini
		2-49					
		[2] [1] 50-58	4-53				
		[3] [1] 59-62	[1] 54-66	3.1-10	10-40	3.2-10.2	
6-10	12-17	63-68	67-76	[1] 10.1-12	[1] 41-60	[1] 10.3-12.1	
11	18	69	77	[1] 12.1	[1] 62	[1] 12.3	all
	76	70-71	78-80	[1] 13-TP		[1] 13	

Android Browser	Opera Mobile	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baic Brows
					4		
					[1] 5.4		
2.1-4.4.4	12-12.1				6.2-8.2		
76	[1] 46	76	68	12.12	9.2	1.2	[1] 7.1

Angular Elements

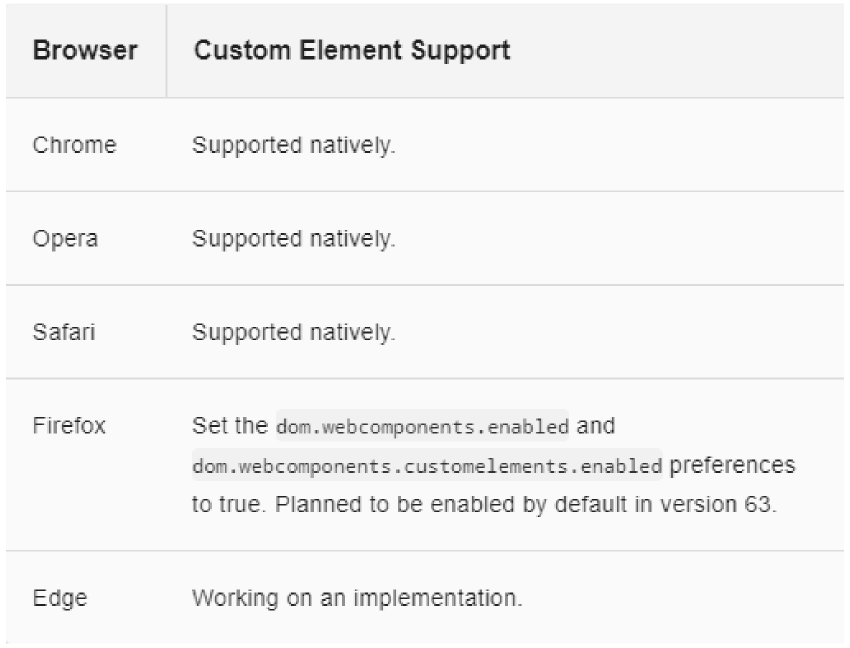
What is Angular Elements

Angular elements are Angular components packaged as custom elements

Defines new HTML elements in a framework-agnostic way

Uses Web Component Standard

Allows implementing Micro Frontends



Browser	Custom Element Support
Chrome	Supported natively.
Opera	Supported natively.
Safari	Supported natively.
Firefox	Set the `dom.webcomponents.enabled` and `dom.webcomponents.customelements.enabled` preferences to true. Planned to be enabled by default in version 63.
Edge	Working on an implementation.

Angular Elements Big Picture

Create an Angular Project - Use Technology you already know

Add Angular Elements

Implement Functionality

Export as a single File

Use it!

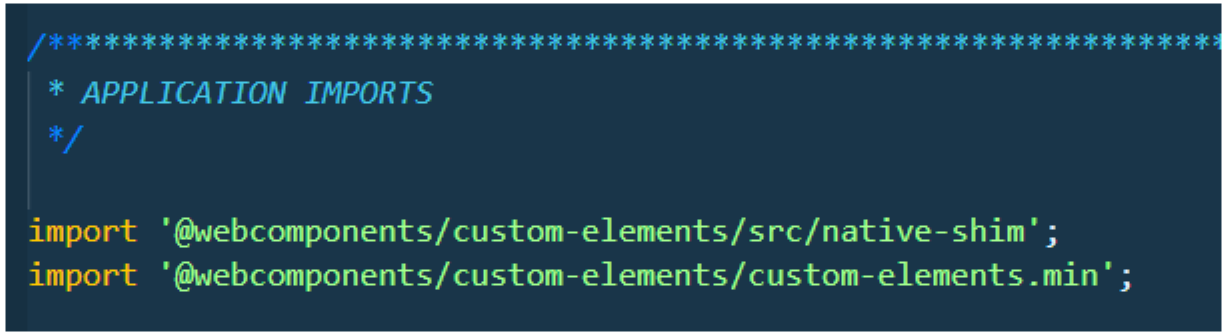
Polyfills

Two Polyfills required:

- @webcomponents/webcomponentsjs
- @webcomponents/custom-elements

Actually three:

- document-register-element.js added by ng add elements



```
/******************************************************************
 * APPLICATION IMPORTS
 */

import '@webcomponents/custom-elements/src/native-shim';
import '@webcomponents/custom-elements/custom-elements.min';
```

ngx-build-plus

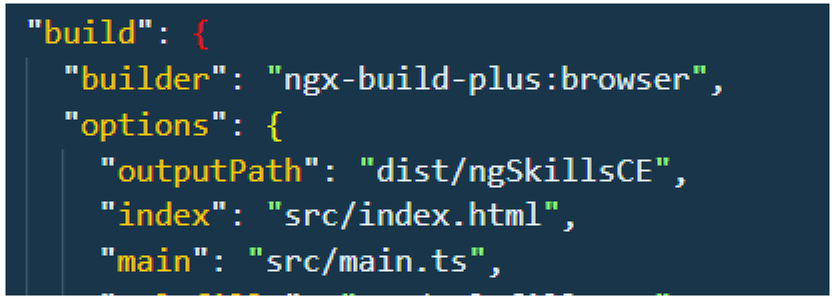
Lib by Austrian Angular GDE Manfred Steyer

Extends Angular Build process with methods used by Elements

Changes builder setting in angular.json

Extends CLI with new Flags

- --single-bundls
- --keep-polyfills



```
"build": {
  "builder": "ngx-build-plus:browser",
  "options": {
    "outputPath": "dist/ngSkillsCE",
    "index": "src/index.html",
    "main": "src/main.ts",
```

Getting Started

1. Create Project: `ng new nge-skills`
2. Add Polyfills:
   - npm install -S @webcomponents/webcomponentsjs @webcomponents/custom-elements
3. Add document-register-element: `npm i document-register-element@1.8.1`
4. Create your Component: `ng g c skills-list -v Native`
5. Add it to AppComponent & Implement your Custom Element
6. Uncomment AppComponent -> Build -> Test

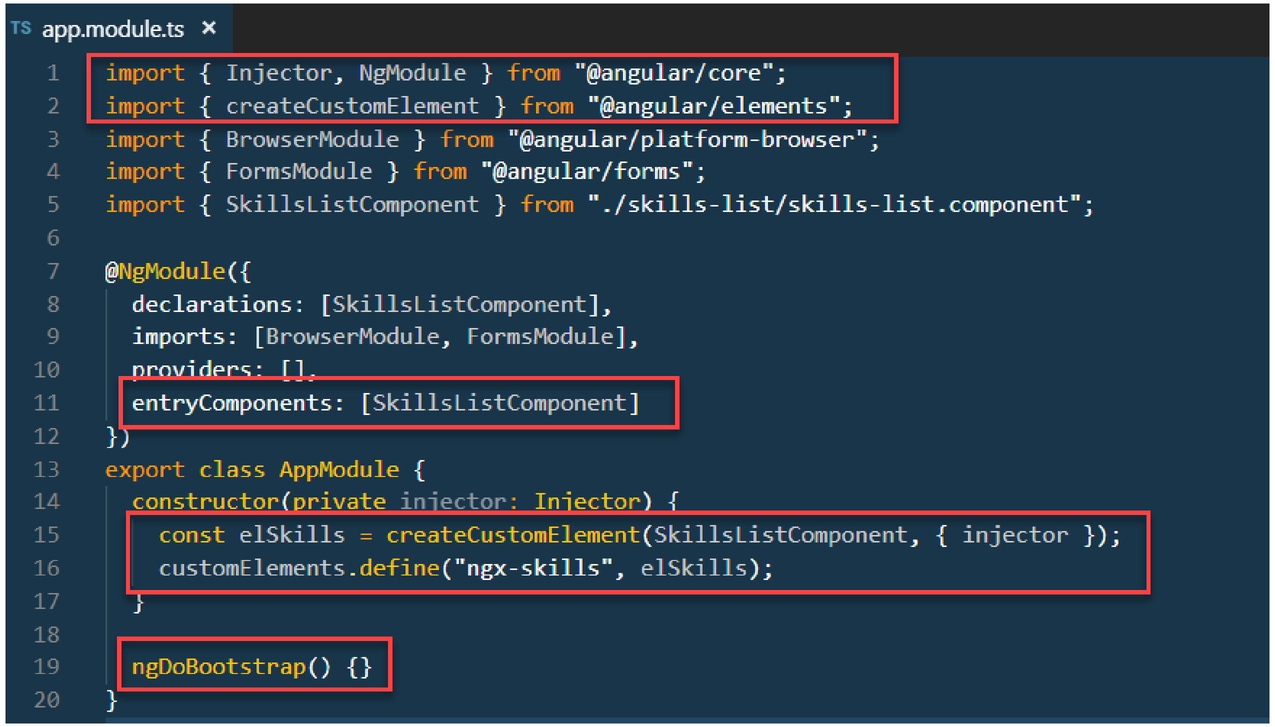
Modify App Module

Remove References to AppComonent from App Module

Set your Component as entryComponent

Create your Custom Element &

bootstrap it



app.module.ts ×

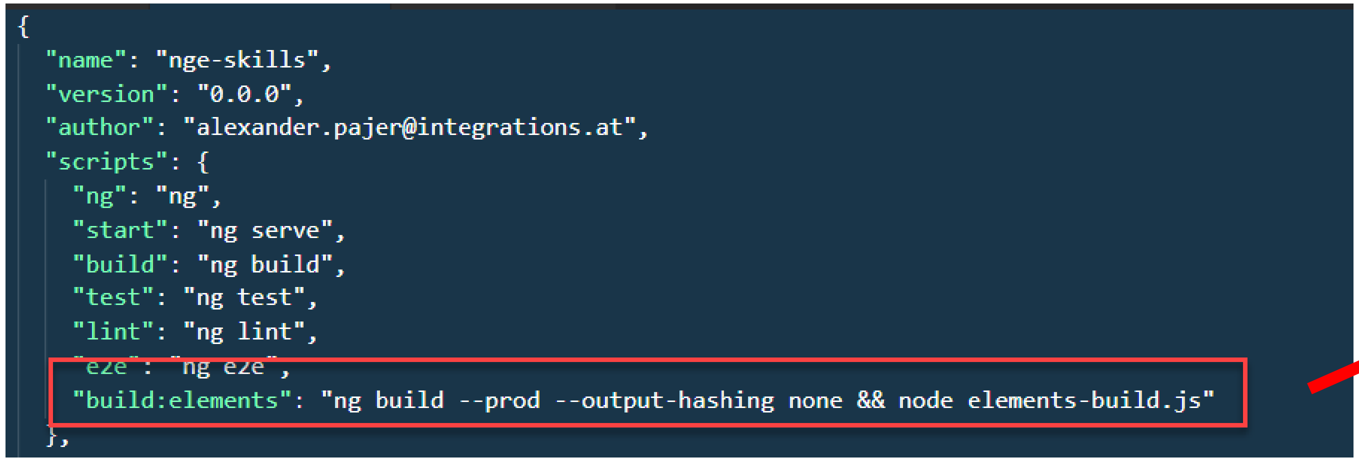
```typescript
import { Injector, NgModule } from "@angular/core";
import { createCustomElement } from "@angular/elements";
import { BrowserModule } from "@angular/platform-browser";
import { FormsModule } from "@angular/forms";
import { SkillsListComponent } from "./skills-list/skills-list.component";

@NgModule({
  declarations: [SkillsListComponent],
  imports: [BrowserModule, FormsModule],
  providers: [],
  entryComponents: [SkillsListComponent]
})
export class AppModule {
  constructor(private injector: Injector) {
    const elSkills = createCustomElement(SkillsListComponent, { injector });
    customElements.define("ngx-skills", elSkills);
  }

  ngDoBootstrap() {}
}
```

INTEGRATIONS

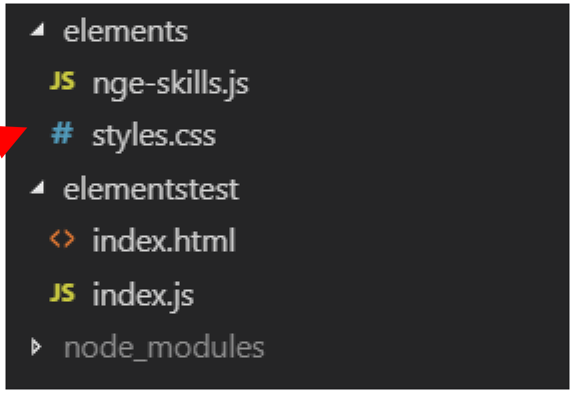
Building

Add a custom npm build script

Combine the output into one single file using elements-build.js



```
{
  "name": "nge-skills",
  "version": "0.0.0",
  "author": "alexander.pajer@integrations.at",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
    "build:elements": "ng build --prod --output-hashing none && node elements-build.js"
  },
```

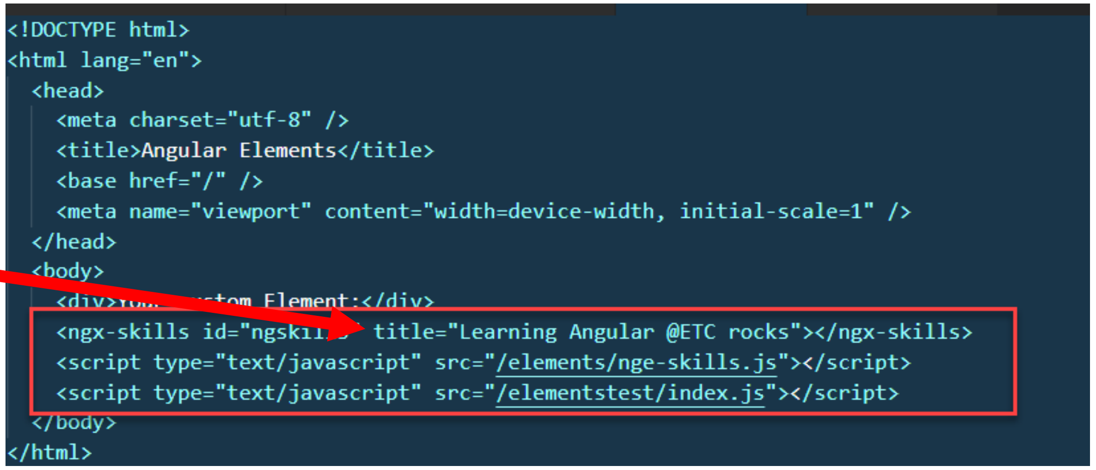


```
◢ elements
    nge-skills.js
    styles.css
◢ elementstest
    index.html
    index.js
▸ node_modules
```

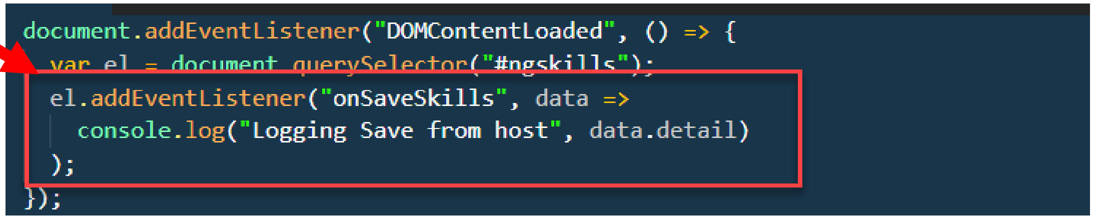
Use Elements

Once you have build and concated your Custom Element you can use it by

- Use the custom tag
- Reference the *.js file
- Pass Params using Attributes
- Hook Event Handler using Code



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Angular Elements</title>
    <base href="/" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
  </head>
  <body>
    <div>Your Custom Element:</div>
    <ngx-skills id="ngskills" title="Learning Angular @ETC rocks"></ngx-skills>
    <script type="text/javascript" src="/elements/nge-skills.js"></script>
    <script type="text/javascript" src="/elementstest/index.js"></script>
  </body>
</html>
```



```
document.addEventListener("DOMContentLoaded", () => {
  var el = document.querySelector("#ngskills");
  el.addEventListener("onSaveSkills", data =>
    console.log("Logging Save from host", data.detail)
  );
});
```

Dynamic Component Loading