

illionx



Global Knowledge.®

# Angular Advanced Module – Schematics

Peter Kassenaar –  
[info@kassenaar.com](mailto:info@kassenaar.com)

WORLDWIDE LOCATIONS

BELGIUM CANADA COLOMBIA DENMARK EGYPT FRANCE IRELAND JAPAN KOREA MALAYSIA MEXICO NETHERLANDS NORWAY QATAR  
SAUDI ARABIA SINGAPORE SPAIN SWEDEN UNITED ARAB EMIRATES UNITED KINGDOM UNITED STATES OF AMERICA



# What are Schematics?

Creating your own schema to create components, services, and so on.

# Contents

- What are Schematics
- Use the Schematics CLI tool to create a new project
- Use the Schematics CLI tool to debug your newly created schematic project
- Using chain and externalSchematic to compose rules and call other schematics
- Use Angular CLI to call your schematics

# Credits – Angular blog

The screenshot shows a blog post on the Angular blog. At the top, there's a navigation bar with icons for 'M' and 'A', a search bar, a notifications icon (6), an 'Upgrade' button, and a user profile picture. The main title is 'Schematics — An Introduction'. Below it, the author is 'Hans' (with a 'Follow' button), the date is 'Feb 1, 2018 · 8 min read', and social sharing icons for Twitter, LinkedIn, Facebook, and a bookmark are present. The post content discusses Schematics as a workflow tool for the modern web, mentioning its ability to apply transforms like creating new components or fixing dependency changes. It also mentions adding new configuration options or frameworks to existing projects. A section titled 'Goals' explains the mission of the Angular CLI, which is to improve development productivity by providing a more powerful and generic facility for CLI scaffolding. The goals listed are: 1. Ease of use and development, 2. Intuitive concepts, 3. Natural feel, and 4. Synchronous development.

## Schematics — An Introduction

Hans [Follow](#)  
Feb 1, 2018 · 8 min read

Schematics is a workflow tool for the modern web; it can apply transforms to your project, such as create a new component, or updating your code to fix breaking changes in a dependency. Or maybe you want to add a new configuration option or framework to an existing project.

### Goals

The mission of the Angular CLI is to improve your development productivity. We came to a point where we needed a more powerful and generic facility to support the CLI scaffolding, and we settled on 4 primary goals:

1. **Ease of use and development.** It had to have simple concepts for developers that were intuitive and feel natural. Also, the code developed needed to be synchronous to make it easier to develop.

<https://blog.angular.io/schematics-an-introduction-dc1dfbc2a2b2>

# What are schematics?

*“Schematics is a **workflow tool** for the modern web; it can **apply transforms** to your project, such as create a new component, or updating your code to fix breaking changes in a dependency. Or maybe you want to **add a new configuration option** or framework to an existing project.”*

# 4 goals

1. Ease of use – by using a schematic, your users can generate code more quickly
2. Extensibility and reuseability – extend and reuse existing schemas and compose new functionality
3. Atomicity – remove side effects (such as creating a file) from the workflow
4. Asynchronicity – work in synchronous and asynchronous environments

# Understanding Schematics

*"In a schematic, you **don't actually perform** any direct actions on the filesystem. Rather, you describe what transformation you would like to apply to a Tree."*

*The Tree is a data structure that contains a **base** (a set of files that already exists) and a **staging area** (a list of changes to be applied to the base).*



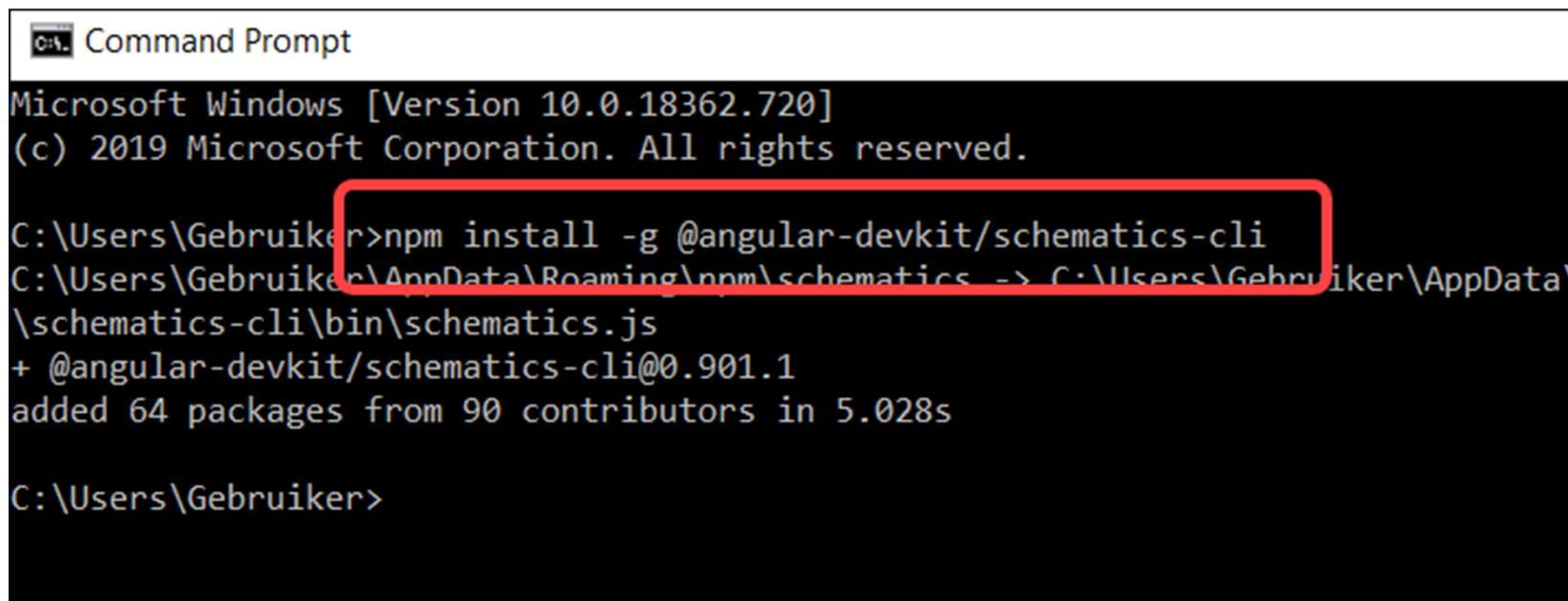
# Creating your first Schematic

Creating your own schema to create components, services, and so on.

# 1. Install Schematics tool globally

Install the schematics executable, which you can use to create a blank Schematics project

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



```
Command Prompt
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Gebruiker>npm install -g @angular-devkit/schematics-cli
C:\Users\Gebruiker\AppData\Roaming\npm\schematics -> C:\Users\Gebruiker\AppData\
\schematics-cli\bin\schematics.js
+ @angular-devkit/schematics-cli@0.901.1
added 64 packages from 90 contributors in 5.028s

C:\Users\Gebruiker>
```

## 2. Create blank schematics project

schematics blank --name=my-component

cd into the project and install dependencies

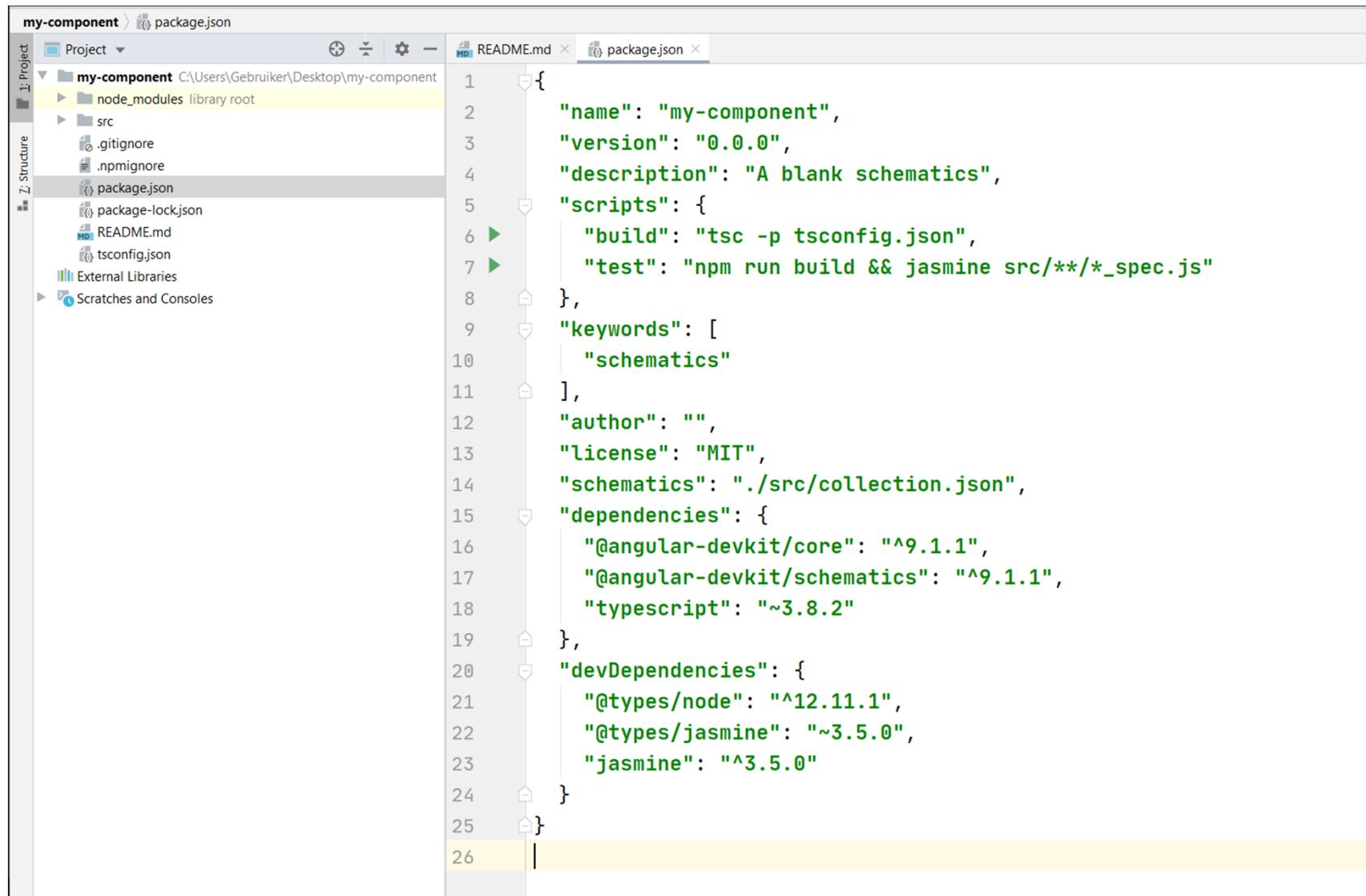
cd my-component

npm install (if necessary)

```
C:\Users\Gebruiker\Desktop>schematics blank --name=my-component
CREATE my-component/package.json (50 bytes)
CREATE my-component/README.md (639 bytes)
CREATE my-component/tsconfig.json (656 bytes)
CREATE my-component/.gitignore (191 bytes)
CREATE my-component/.npmignore (64 bytes)
CREATE my-component/src/collection.json (231 bytes)
CREATE my-component/src/my-component/index.ts (318 bytes)
CREATE my-component/src/my-component/index_spec.ts (474 bytes)
✓ Packages installed successfully.

C:\Users\Gebruiker\Desktop>
```

# Open project in editor



The screenshot shows a code editor interface with the following details:

- Project Bar:** Shows the project name "my-component" and the path "C:\Users\Gebruiker\Desktop\my-component".
- Structure View:** Shows the project structure with files like node\_modules, src, .gitignore, .npmignore, package.json, package-lock.json, README.md, tsconfig.json, External Libraries, and Scratches and Consoles.
- Editor Tabs:** The "package.json" tab is active, showing the JSON content of the package.json file.
- Code Content:** The package.json file contains the following JSON code:

```
{  
  "name": "my-component",  
  "version": "0.0.0",  
  "description": "A blank schematics",  
  "scripts": {  
    "build": "tsc -p tsconfig.json",  
    "test": "npm run build && jasmine src/**/*_spec.js"  
  },  
  "keywords": [  
    "schematics"  
  ],  
  "author": "",  
  "license": "MIT",  
  "schematics": "./src/collection.json",  
  "dependencies": {  
    "@angular-devkit/core": "^9.1.1",  
    "@angular-devkit/schematics": "^9.1.1",  
    "typescript": "~3.8.2"  
  },  
  "devDependencies": {  
    "@types/node": "^12.11.1",  
    "@types/jasmine": "~3.5.0",  
    "jasmine": "^3.5.0"  
  }  
}
```

# Schematic Collections

- Schematic Collections are **sets of named schematics**
  - Publish your schematics
  - Users `npm install` and use your schematic
- Our project holds the `my-component` schematic
  - It is described in `./src/collection.json`
  - This file holds the description of our collection

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

# The factory field

- Note the factory field.
  - The factory field uses a **string reference** to point to a JavaScript function
  - in our case the exported function `myComponent` in the file `my-component/index.js`.
  - It represents the *RuleFactory*.

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

// You don't have to export the function as default.
// You can also have more than one rule factory per file.
export function myComponent(_options: any): Rule {
  return (tree: Tree, _context: SchematicContext) => {
    return tree;
  };
}
```

# Rules

- A Rule is a function that takes a Tree and returns another Tree.
- Rules are the core of Schematics!
- Rules are the ones making changes to your projects, calling external tools, and implementing logic.
- RuleFactory are functions that create a Rule.

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

// You don't have to export the function as default.
// You can also have more than one rule factory per file.
export function myComponent(_options: any): Rule {
  return (tree: Tree, _context: SchematicContext) => {
    return tree;
  };
}
```

# The options argument

- The options argument is an object that can be seen as the **input** of the factory.
- From the CLI, it is the **command line arguments** the user passed.
- From another schematic, it's the **options that were passed in** by that schematic.
- Options is *always* an object, of type `any`.
- In the current situation, the function takes in a tree and returns it unaltered – *it is your task to write logic here!*

```
return (tree: Tree, _context: SchematicContext) => {
  return tree;
};
```

# Let's make our rule do something

```
export function myComponent(_options: any): Rule {  
  return (tree: Tree, _context: SchematicContext) => {  
    tree.create(_options.name || 'hello', 'world');  
    return tree;  
  }  
}
```



With this new line, we're creating a file in the root of the schematic's Tree, named either after the name option (or 'hello' by default), containing the string world.

# Working with Tree

- By default, Angular CLI will pass the root of your Angular project as the Tree,
- Any schematic can pass in a different Tree to other schematics.
  - You can **create empty trees**,
  - **Scope** a Tree to a directory of a parent Tree,
  - **Merge** two trees or **branch** them (making a copy of it).
- Tree changing methods:
  - `Tree.create`,
  - `Tree.delete`,
  - `Tree.rename`,
  - `Tree.overwrite`



# Running your Schematic

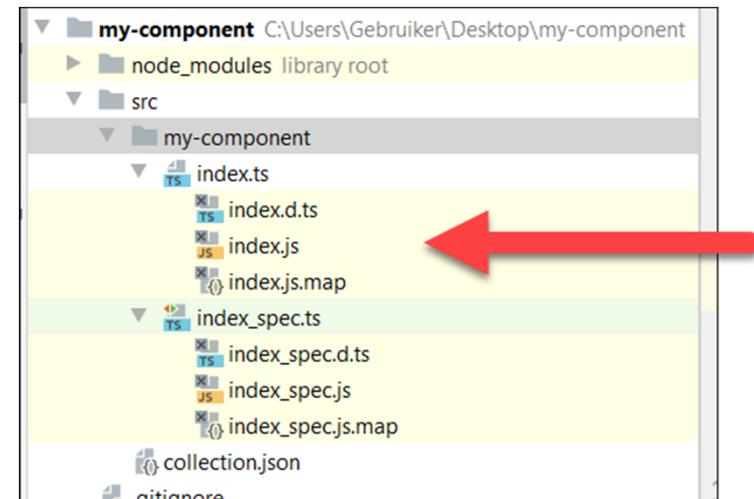
First build your schematic, then run it using the  
schematics command line tool

# Building the schematic

- Build the schematic, then run it using the schematics tool.
- The current path acts as the schematics collection

npm run build

```
PS C:\Users\Gebruiker\Desktop\my-component> npm run build  
  
> my-component@0.0.0 build C:\Users\Gebruiker\Desktop\my-component  
> tsc -p tsconfig.json  
  
PS C:\Users\Gebruiker\Desktop\my-component>
```



# Running the schematic

- By default schematics run in **debug mode**
- **No changes** to the file system are made
- Be careful however when testing this!
  - `--dry-run=false`
  - Only disable dry runs when necessary

Schematics `.:my-component --name=test`

```
PS C:\Users\Gebruiker\Desktop\my-component> schematics .:my-component --name=test
CREATE test (5 bytes)
PS C:\Users\Gebruiker\Desktop\my-component> schematics .:my-component
CREATE hello (5 bytes)
PS C:\Users\Gebruiker\Desktop\my-component>
```

# Composing schematics

- We can call **other schematics** from our schematic and thus compose a workflow
- In this example, we'll call the component schematic from the Angular collection to add a component to the application
- then add a header to every TypeScript files added by the Schematic.

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

const licenseText = `
/*
 * @license
 * Copyright Kassenaar IT Services (C) 2020. All Rights Reserved.
 *
 * Use of this source code is governed by an MIT-style license.
 * See for more information: https://opensource.org/licenses/MIT
 */
`;

export function ts HeaderComponent(_options: any): Rule {
  return chain([
    // Don't forget to add @schematics/angular to your dependencies in your package.json!
    externalSchematic('@schematics/angular', 'component', _options),
    (tree: Tree, _context: SchematicContext) => {
      tree.getDir(_options.sourceDir)
        .visit(filePath => {
          if (!filePath.endsWith('.ts')) {
            return;
          }
          const content = tree.read(filePath);
          if (!content) {
            return;
          }
          // Prevent from writing license to files that already have one.
          if (content.indexOf(licenseText) == -1) {
            tree.overwrite(filePath, licenseText + content);
          }
        });
      return tree;
    },
  ])
}
```

# What's going on here?

- Call and return `chain()`
  - `Chain()` is a `RuleFactory` that can chain multiple rules together
- Call `externalSchematic` to import the requested schematic
- Pass the tree and loop over every file. Add the header to the `.ts`-files
- Documentation: <https://angular.io/guide/schematics>

The screenshot shows a section of the Angular documentation titled "Generating code using schematics". The left sidebar lists various topics under "TECHNIQUES", including "Security", "Internationalization (i18n)", "Accessibility", "Service Workers & PWA", "Server-side Rendering", "Upgrading from AngularJS", and "Angular Libraries". The main content area starts with a brief introduction to what a schematic is, followed by a detailed explanation of how the schematic collection can be used to customize projects. Below this, there is a heading "Schematics for the Angular CLI" with a corresponding section of text.

INTRODUCTION

GETTING STARTED >

SETUP

FUNDAMENTALS >

TECHNIQUES >

- Security
- Internationalization (i18n)
- Accessibility
- Service Workers & PWA >
- Server-side Rendering
- Upgrading from AngularJS >
- Angular Libraries >

## Generating code using 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.

The schematic collection can be a powerful tool for creating, modifying, and maintaining any software project, but is particularly useful for customizing Angular projects to suit the particular needs of your own organization. You might use schematics, for example, to generate commonly-used UI patterns or specific components, using predefined templates or layouts. You can use schematics to enforce architectural rules and conventions, making your projects consistent and inter-operative.

## Schematics for the Angular CLI

Schematics are part of the Angular ecosystem. The [Angular CLI](#) uses schematics to apply transforms to a web-app project. You can modify these schematics, and define new ones to do things like update your code to fix breaking changes in a dependency, for example, or to add a new configuration option or framework to an existing project.

Schematics that are included in the `@schematics/angular` collection are run by default by the commands `ng generate` and `ng add`. The package contains named schematics that configure the options that are available to the CLI for `ng generate` sub-commands, such as `ng generate component` and `ng generate service`. The subcommands for `ng generate` are shorthand for the corresponding schematic. You can specify a particular schematic (or collection of

# Testing your schematic with Angular CLI

- Ideally you'll publish your schematic to NPM where others can `npm install` it
- For now we'll use npm symbolic links to link to our schematic
- First, create a new project
  - `ng new my-project`

# Linking to the schematic

- Inside the project, link the Schematics we just build (again, instead of npm installing it).
  - `npm link $PATH_TO_SCHEMATIC_PROJECT`
  - E.g `npm link ../my-component`

```
C:\Users\Gebruiker\Desktop>cd my-project

C:\Users\Gebruiker\Desktop\my-project>npm link ../my-component
npm WARN my-component@0.0.0 No repository field.

audited 136 packages in 0.619s

2 packages are looking for funding
  run `npm fund` for details

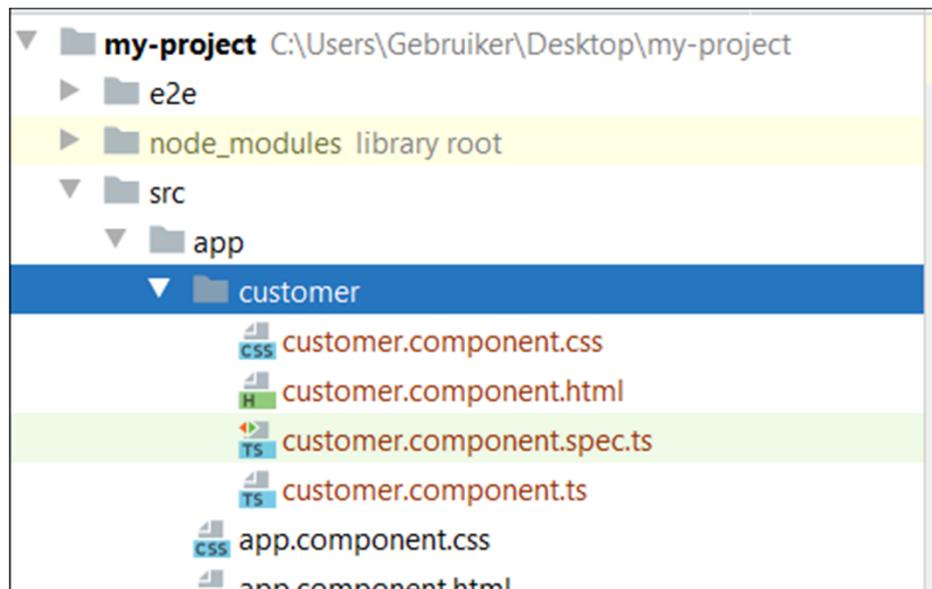
found 0 vulnerabilities

C:\Users\Gebruiker\AppData\Roaming\npm\node_modules\my-component -> C:\Users\Gebruiker\Desktop\my-component
C:\Users\Gebruiker\Desktop\my-project\node_modules\my-component -> C:\Users\Gebruiker\AppData\Roaming\npm\node_modules\my-component -> C:\Users\Gebruiker\Desktop\my-component

C:\Users\Gebruiker\Desktop\my-project>
```

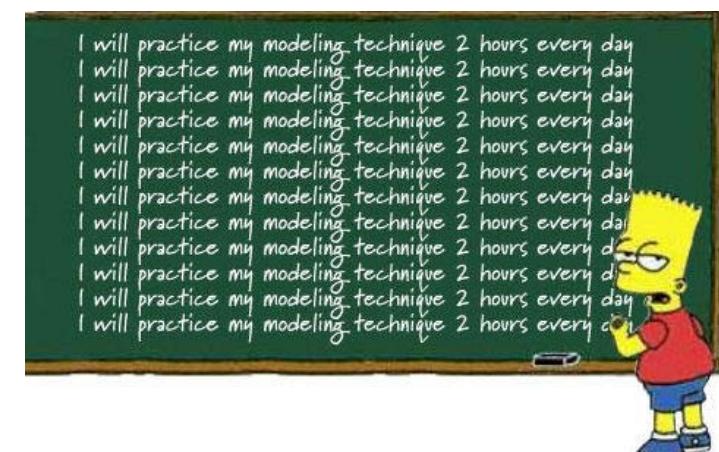
# Using the schematic

```
ng generate my-component:ts-header-component --name=Customer
```



# Workshop

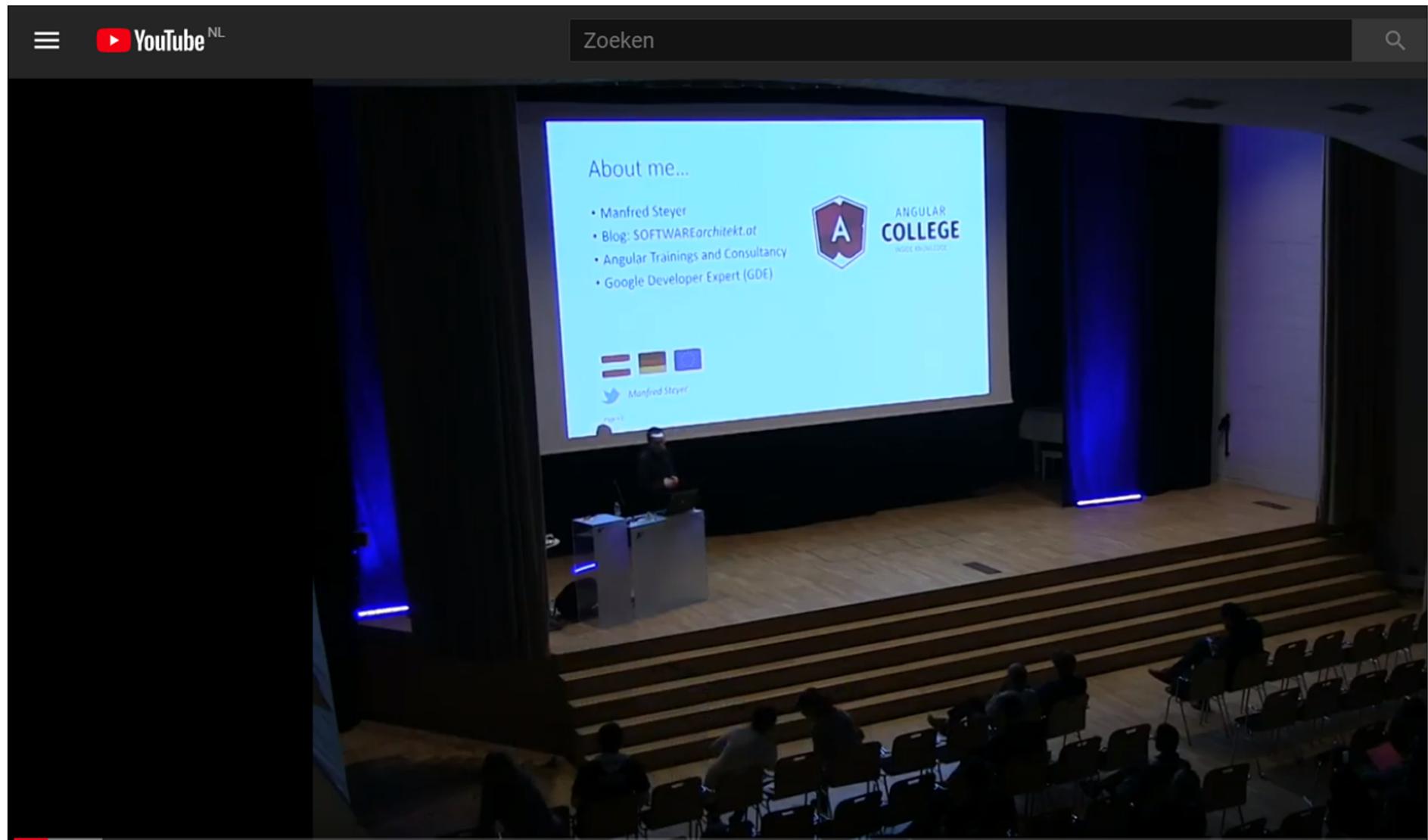
- Build a custom Schematic, using the steps in this presentation
- Your schematic should output a file hello.js, which does a `console.log('hello world')`.
- ...TBD...



# More info



<https://medium.com/@tomastrajan/total-guide-to-custom-angular-schematics-5c50cf90cdb4>



<https://www.youtube.com/watch?v=JAt1FSwhnWk>