ilionx

Global Knowledge®

Angular Advanced Module – Schematics

Peter Kassenaar –
info@kassenaar.com

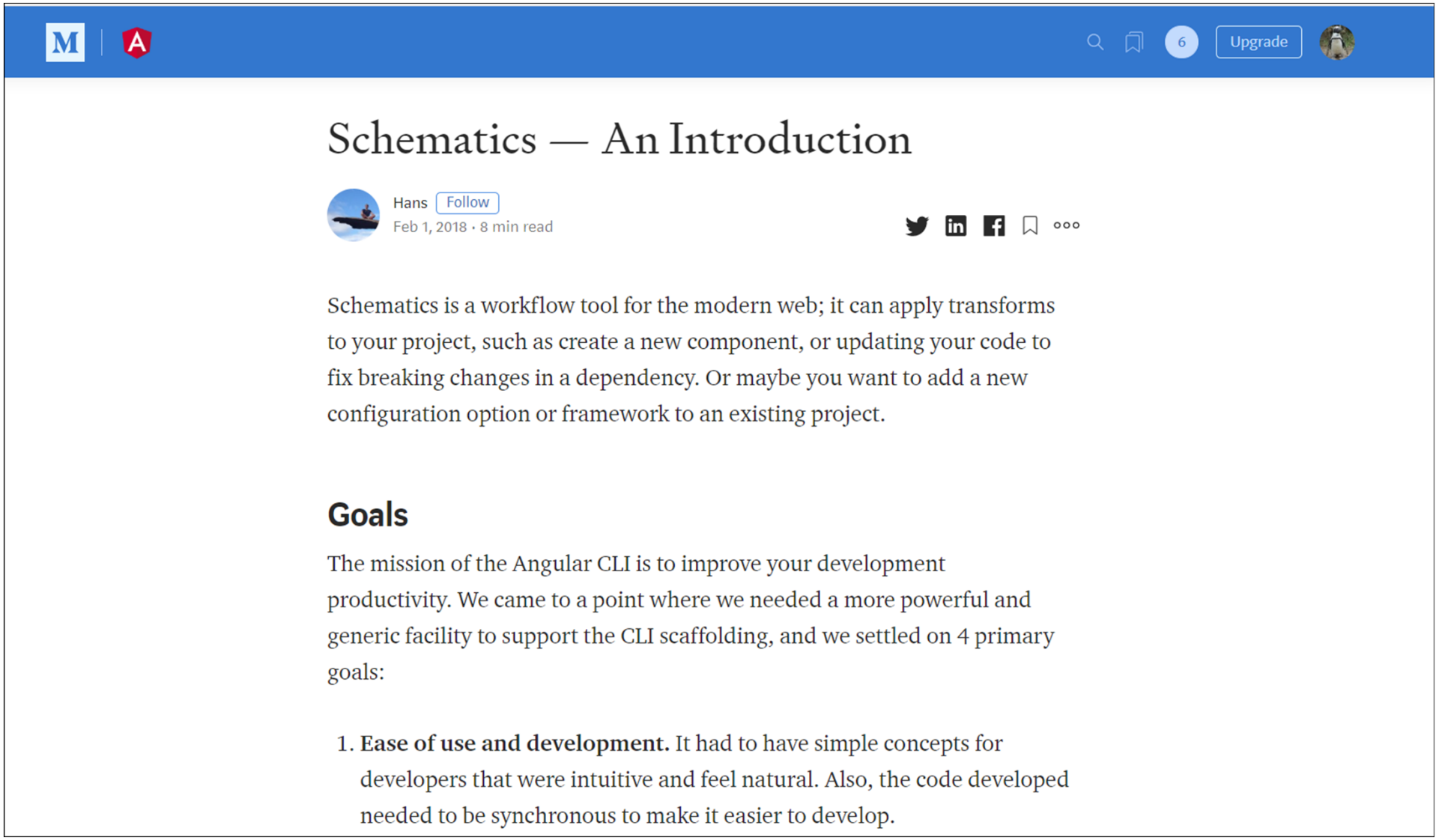
What are Schematics?

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



Upgrade

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."

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

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 an asynchronous environments

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

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).

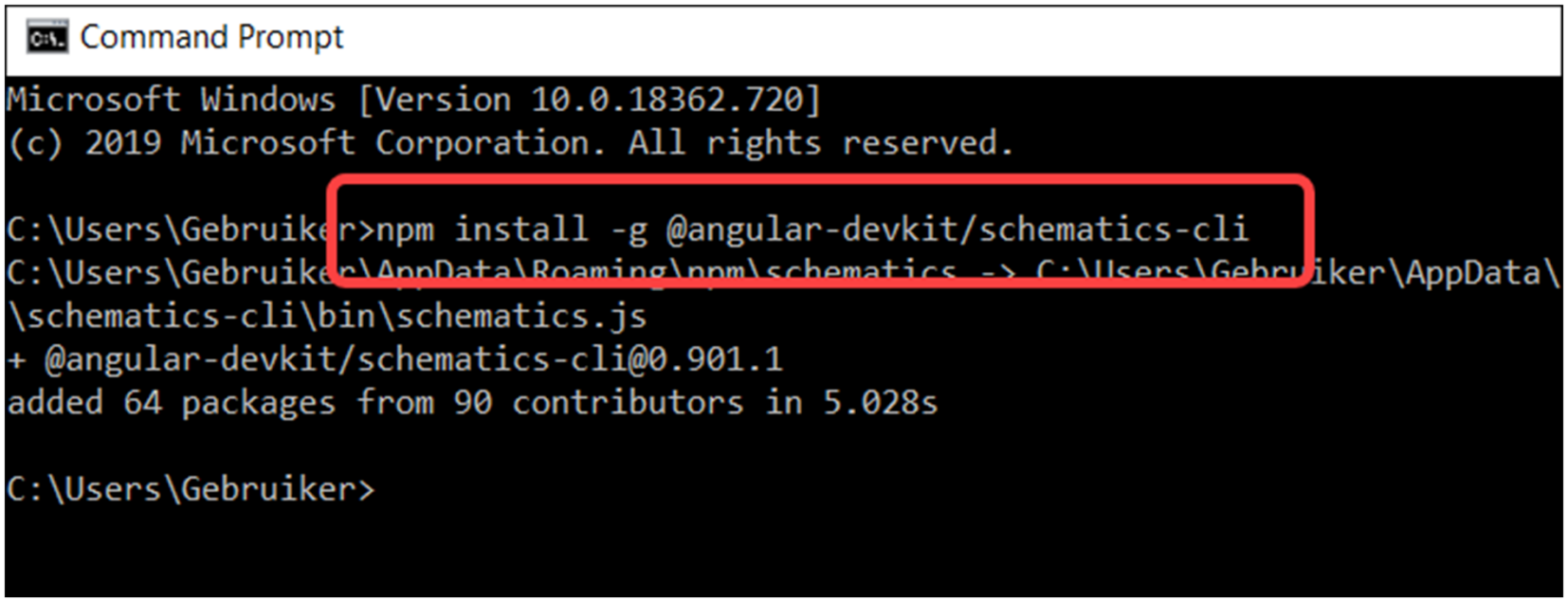
Creating your first Schematic

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
```



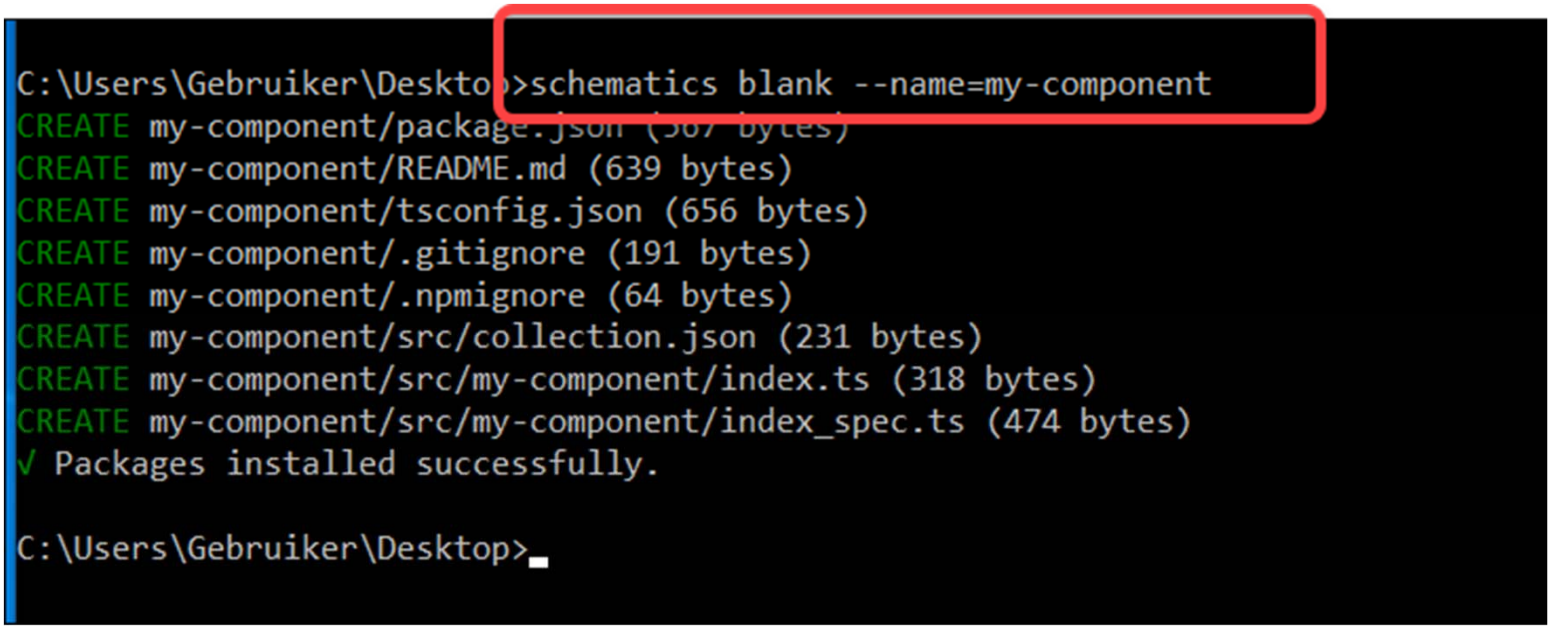
2. Create blank schematics project

`schematics blank --name=my-component`

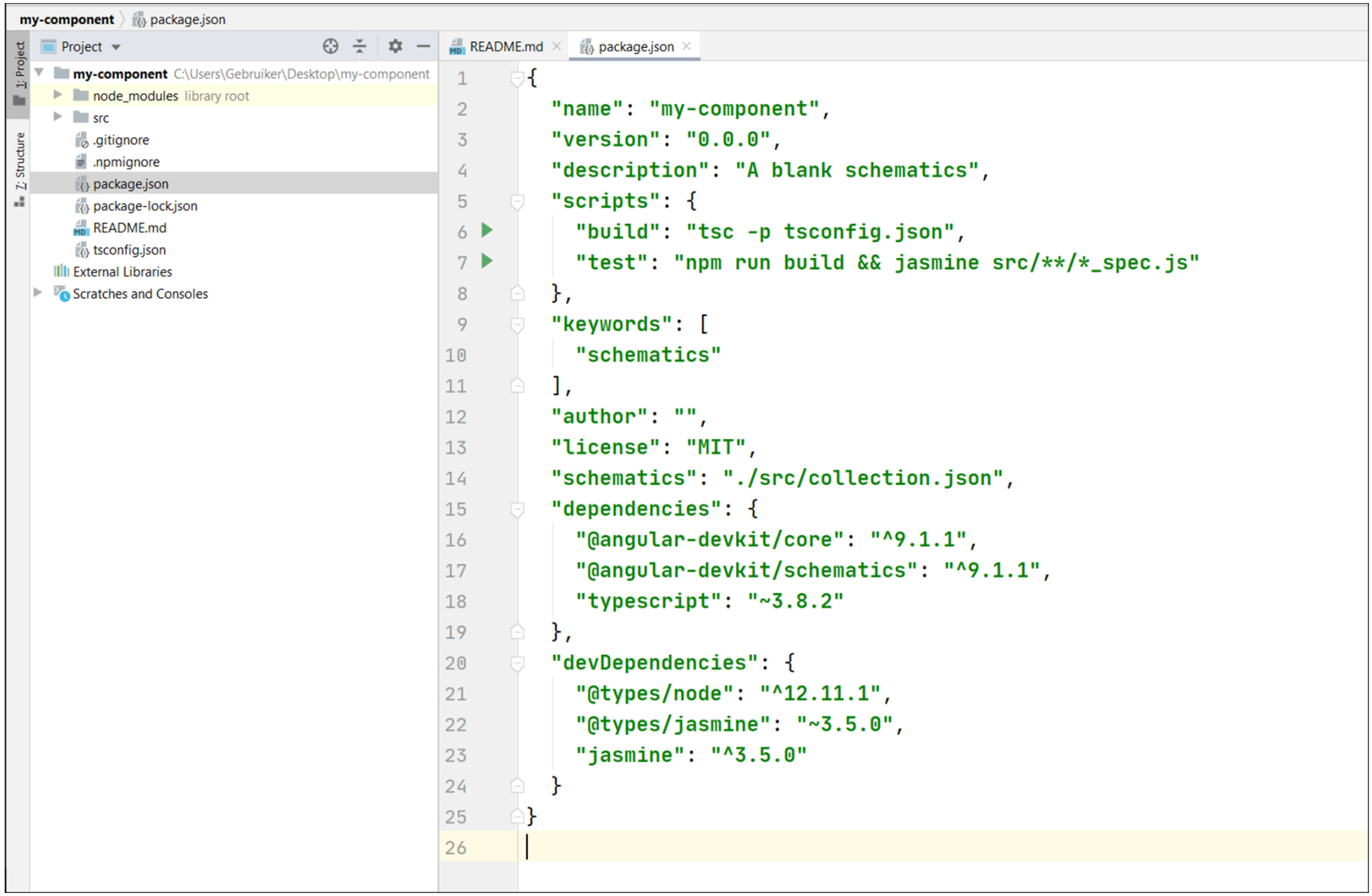
cd into the project and install dependencies

`cd my-component`

`npm install` (if necessary)



Open project in editor



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

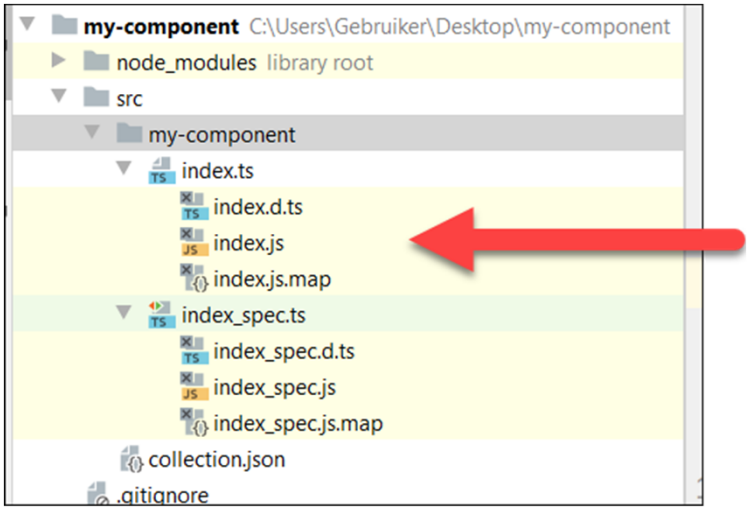
First build your schematic, then run it using the `schematics` command line tool

Building the schematic

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

`npm run build`

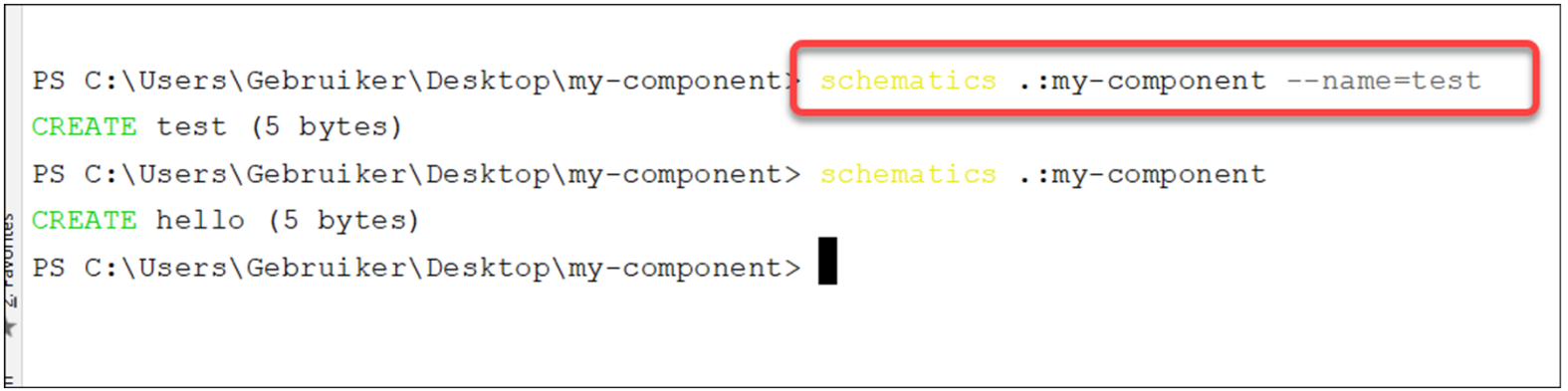
```
PS C:\Users\Gebruiker\Desktop\my-component> npm run build

> 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`



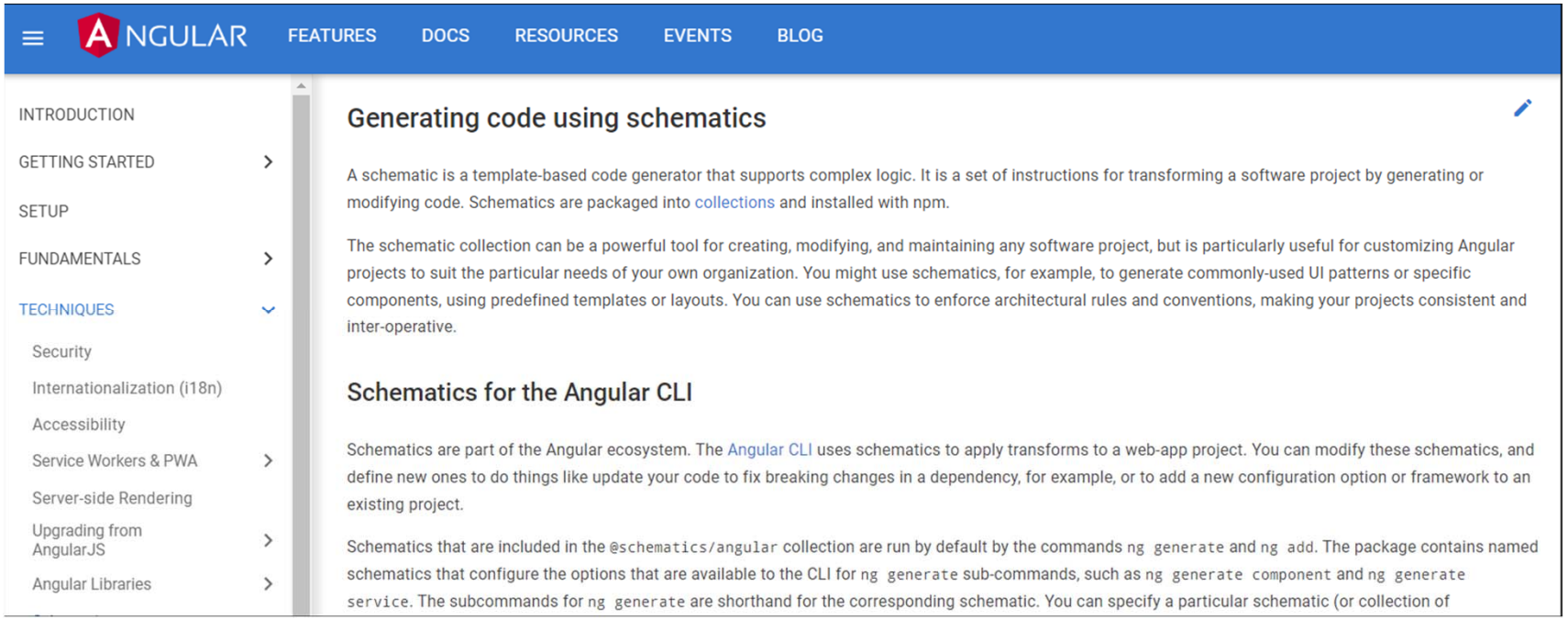
Composing schematics

- 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 tsHeaderComponent(_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



Testing your schematic with Angular CLI

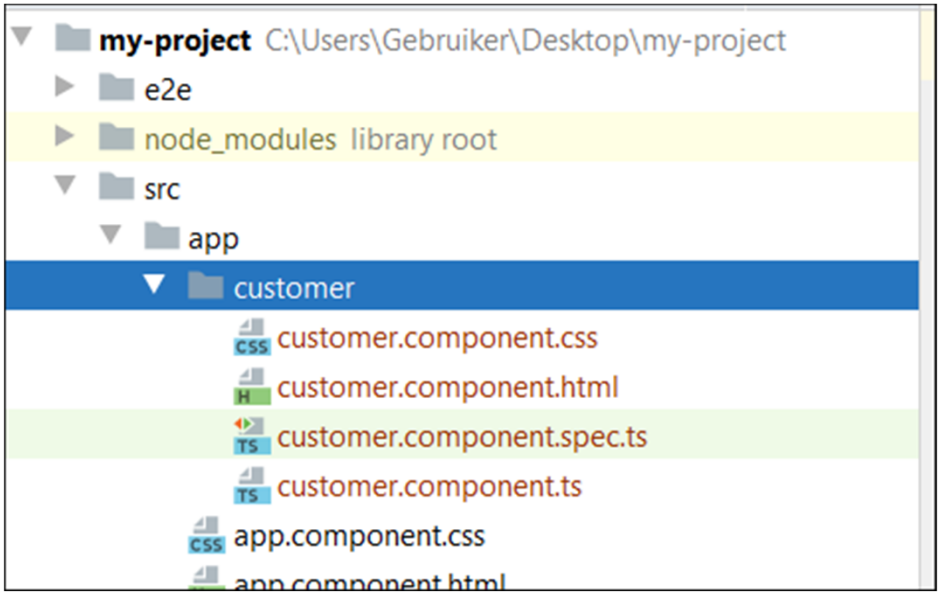
- Ideally you'll publish your schematic to NPM where others can `npm install` it
- For now we'll use npm symbolic links to link to our `schematic`
- First, create a new project
  - `ng new my-project`

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`

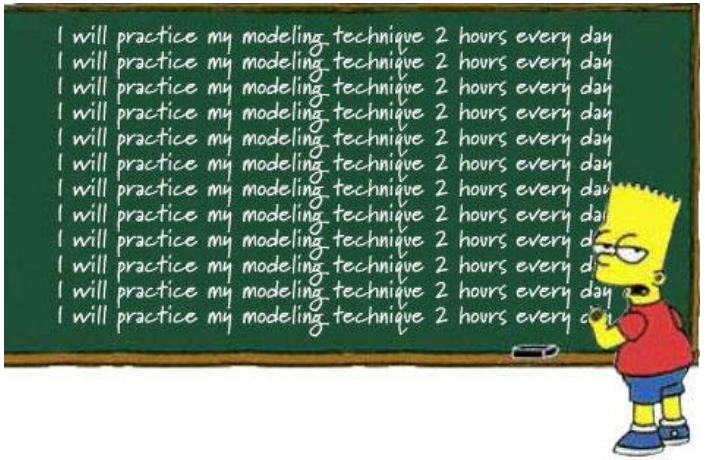
Using the schematic

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



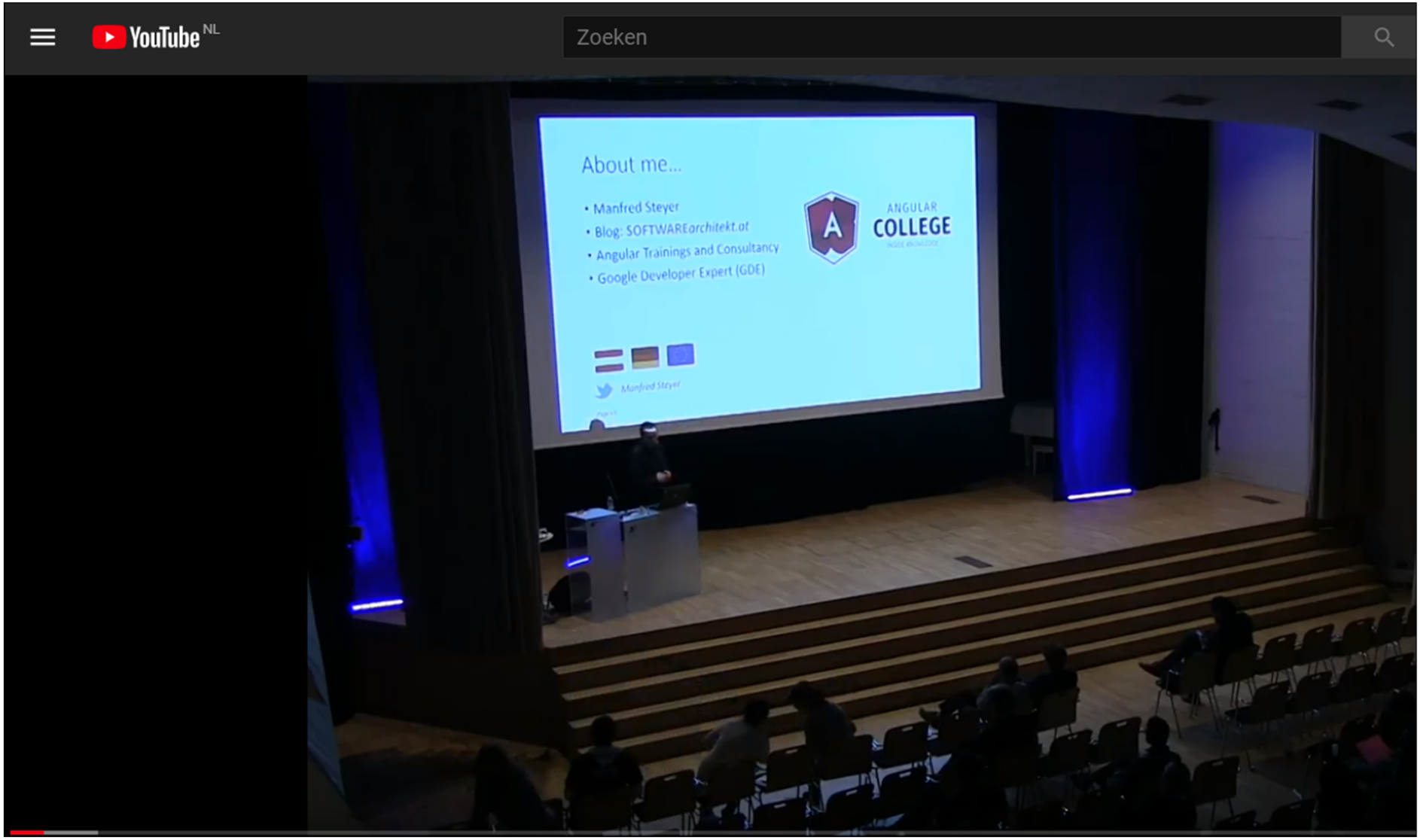
Workshop

- Build a custom Schematic, using the steps in this presentation
- Your schematic should output a file `hello.js`, wich 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