ANGULAR SCHEMATICS

AUTOMATE YOUR ANGULAR WORKSPACE

HI, MY NAME IS ORJAN.

Competence Lead Frontend Architecture
Ordina Belgium

@orjandesmet

orjandesmet

THE PROBLEM STATEMENT

Code isn't always completely DRY

Work isn't always completely DRY

EXAMPLE 1

Shared logic is moved to a library

Initialisation of said library in your app is still needed

New app -> library might be forgotten

Updated library with breaking change?

EXAMPLE 2

Custom reusable component

Specialised wrappers for this component

1 Wrapper with if/else/switch or multiple wrappers?

What if you find a bug?

TODAY'S WORKSHOP

- What are workspace schematics
- Build your first (and second) schematic

WHAT ARE SCHEMATICS

WHAT THEY CAN DO

Help you to create, move, delete, edit... files automatically inside your Angular workspace.

Help you to stay organised by automating tasks that you would normally do manually.

Endless possibilities... (of course)

ANGULAR CLI EXTENSION

Using **ng generate**, **ng add** and **ng update**.

EXAMPLES

```
ng generate component --name profile --inlineStyle --inlineTemplate
```

```
ng add @angular/material
```

YOUR FIRST SCHEMATIC: HELLO WORLD!

INITIATING A SCHEMATICS PROJECT

```
npm install -g @angular-devkit/schematics-cli
schematics blank --name=my-first-schematic
```

Creates a workspace with a blank schematic
(or only a blank schematic, if already in a workspace)

SCHEMATICS WORKSPACE

src-folder contains actual schematics source and *collection.json* Each schematic has a sub-folder with **index.ts** and **index_spec.ts** and is represented in *collection.json* with a description and a pointer to its factory.

package.json has a property *schematics*, which point to collection.json

INSPECTING THE SOURCE

```
export function myFirstSchematic(_options: any): Rule {
    return (tree: Tree, _context: SchematicContext) => {
        return tree;
    };
}
```

tree is the projection of the file system.
_context is the context the schematic is running in.
_options are the parameters passed by the CLI.

The function returns a **Rule**, a function that can return Tree, Observable<Tree>, Rule, Promise<Rule>, void, Promis<void>.

CREATE A FILE

Update the schematic to create a file:

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

RUN YOUR SCHEMATIC

```
npm run build
schematics .:my-first-schematic
```

.: indicates the schematic collection's package.json is in this directory.

Add parameter **--debug=false** to actually run the schematic.

USING PARAMETERS

```
tree.create('hello.txt', `Hello ${_options.name || 'FamiliarNameMissing'}!`);
```

```
npm run build
schematics .:my-first-schematic --name=YourNameHere
```

A SECOND SCHEMATIC

A bit more advanced schematic to quickly add one or more RxJS Store to your project. This way we can prevent having to manually copy and paste the source and update names in files.

After each step you can run the following to test the results of your schematic.

```
npm run build
schematics .:rxjs-storage
```

STORE SOURCE CODE

Too much to copy and paste here, so go to this link: https://github.com/ordina-jworks/my-first-schematic/raw/master/store-source.zip

Contains of model, state, service and test file.

1. START A NEW SCHEMATIC

```
schematics blank --name=rxjs-storage
```

2. USING TEMPLATE FILES

1. Unpack the downloaded files into **src/rxjs-storage/files**
2. Create a function in index.ts, called **copyFiles**, which returns a Rule and merges the template files in the source tree

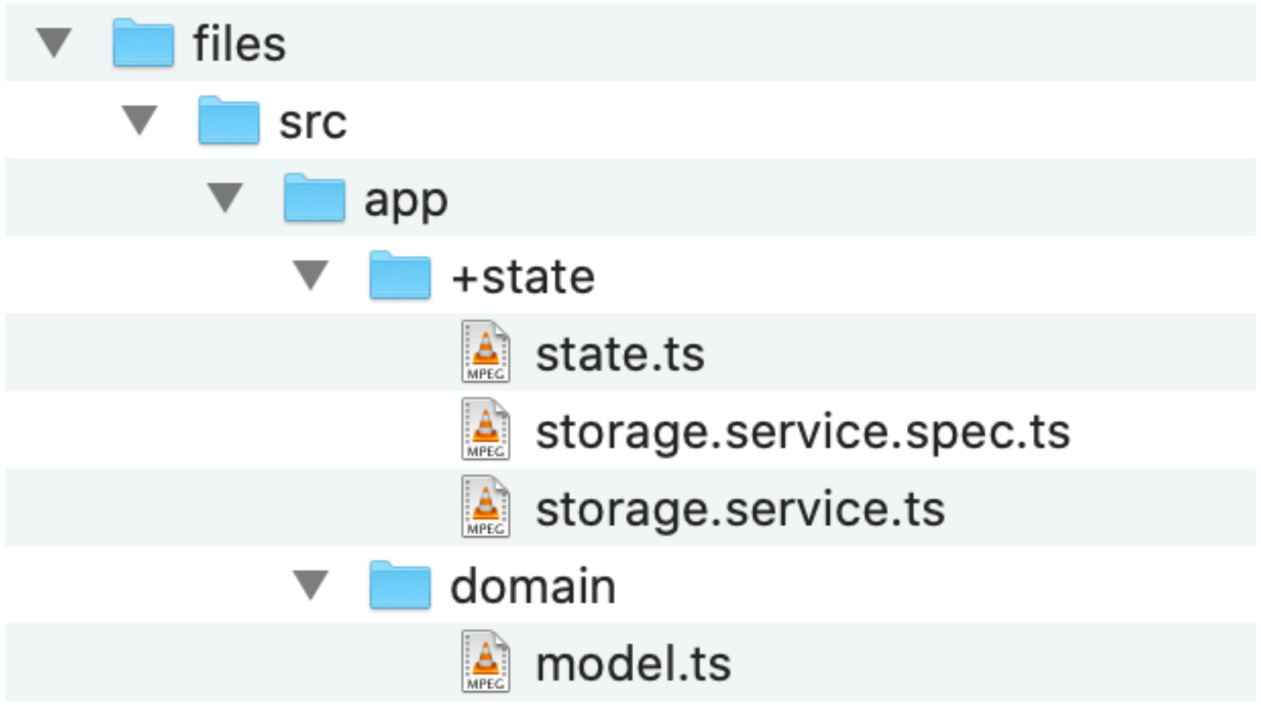
```
import { mergeWith, Rule, url } from '@angular-devkit/schematics';

export function rxjsStorage(_options: any): Rule {
    return copyFiles();
}

function copyFiles(): Rule {
    const sourceTemplates = url('./files');
    return mergeWith(sourceTemplates);
}
```

3. TEMPLATE FILE TREE

This previous step will copy your files into the root of your tree. We'll want them in a sub-folder instead, so create sub-folders in your **files**-folder:



```
files
  src
    app
      +state
        state.ts
        storage.service.spec.ts
        storage.service.ts
      domain
        model.ts
```

Don't forget that you may need to update your **import**s in the source files!

4. APPLYING PARAMETERS TO OUR TEMPLATES

We can modify our template files while they are being copied using parameters.

```
import { strings } from '@angular-devkit/core';
import { apply, mergeWith, Rule, template, url } from '@angular-devkit/schematics
';

function copyFiles(_options: any): Rule {
    const sourceTemplates = url('./files');
    const sourceParameterizedTemplates = apply(sourceTemplates, [
        template({
            ..._options,
            ...strings
        })
    ]);
    return mergeWith(sourceParameterizedTemplates);
}
```

_options are our CLI parameters, and **strings** are a couple of functions to modify strings.

4.A. USING TEMPLATE FUNCTIONS INSIDE TEMPLATES

This allows us to use JS code inside our templates, like this:

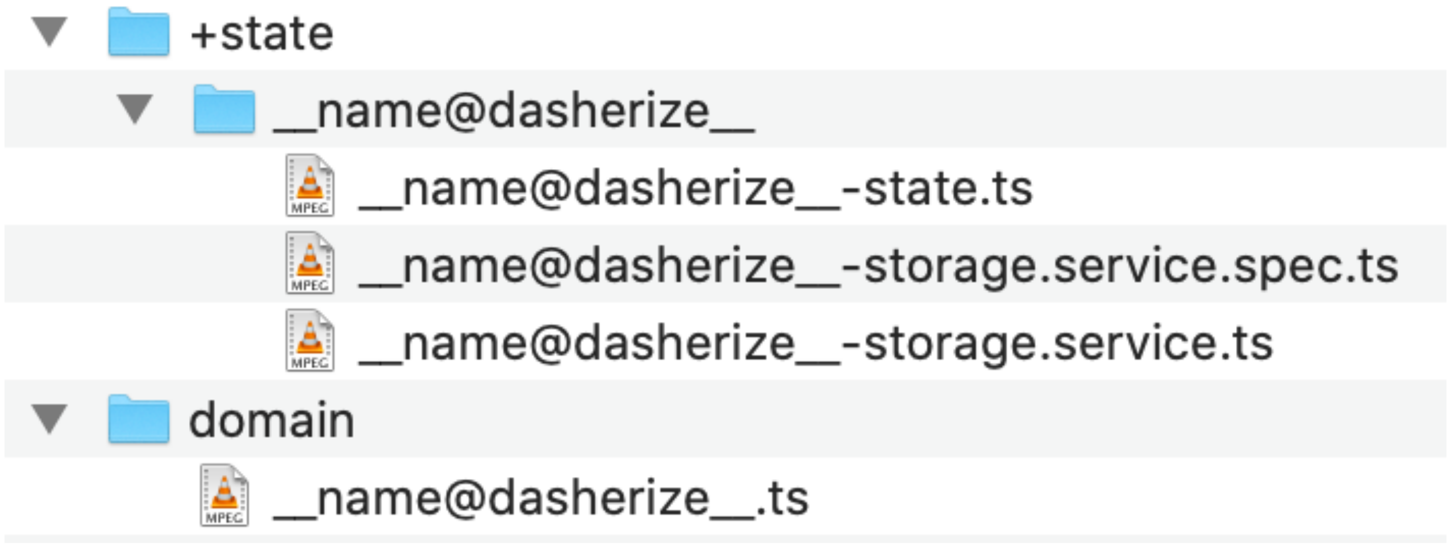
```
import { <%= classify(name) %> } from './../domain/model';

export interface <%= classify(name) %>State {
    list: <%= classify(name) %>[];
    selectedId: string | number | null;
}

export const initialState: <%= classify(name) %>State = {
    list: [],
    selectedId: null
};
```

4.B. USING TEMPLATE FUNCTIONS FOR FILE NAMES

This also allows us to update the file and directory names with 2 _ at both sides, like this:



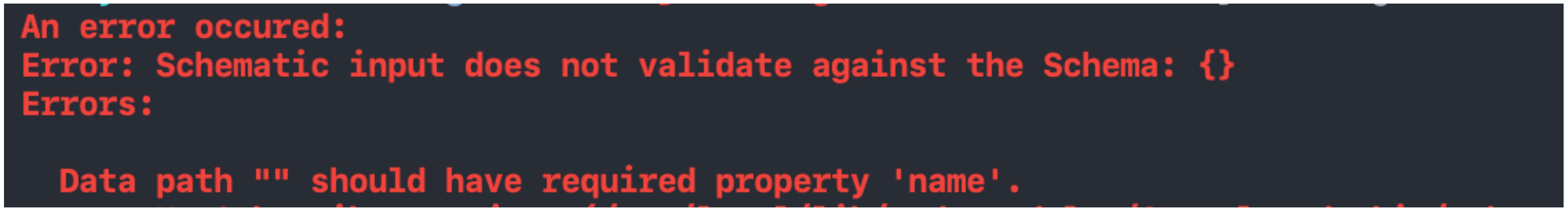
```
+state
    __name@dasherize__
        __name@dasherize__-state.ts
        __name@dasherize__-storage.service.spec.ts
        __name@dasherize__-storage.service.ts
domain
    __name@dasherize__.ts
```

Don't forget that you may need to update your **import**s in the source files!

5. REQUIRED PARAMETER

You might have noticed a difficult to read error message when running the schematic without **--name=YourNameHere**.

Add a schema for a better error message:



```
An error occured:
Error: Schematic input does not validate against the Schema: {}
Errors:

  Data path "" should have required property 'name'.
```

5.A. SCHEMA

Create a schema.json file in your schematic directory:

```
{
    "$schema": "http://json-schema.org/schema",
    "id": "RxJSStorageSchematics",
    "title": "RxJS Storage Options Schema",
    "type": "object",
    "description": "Adds an RxJS storage to your application.",
    "properties": {
        "name": {
            "type": "string",
            "description": "The name of the entity in the storage.",
            "$default": {
                "$source": "argv",
                "index": 0
            }
        }
    },
    "required": ["name"]
}
```

5.B. REFERENCING THE SCHEMA

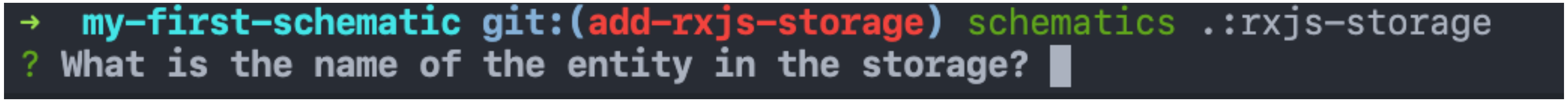
Add the **schema** property in collection.json

```
"schema": "./rxjs-storage/schema.json"
```

5.C. PROMPTING THE REQUIRED PROPERTY

Add a **x-prompt** property to the schema under "name"

```
"x-prompt": "What is the name of the entity in the storage?"
```



```
➜  my-first-schematic git:(add-rxjs-storage) schematics .:rxjs-storage
? What is the name of the entity in the storage?
```

6. CONDITIONAL TEMPLATES

You can use JS in your templates:

```
export interface <%= classify(name) %> {
    id: string | number;
    name: string;
<% if (props && props.length) {
    for (const prop of props) { %>
    <%= camelize(prop[0]) %>: <%= (prop[1] || 'any') %>;
<% }
} %>
}
```

Just make sure that every property is defined:

```
const sourceParameterizedTemplates = apply(sourceTemplates, [
    template({
        ..._options,
        ...strings,
        props: _options.props.split(',')
            .filter((prop: string) => !!prop.trim())
```

```
                .map((prop: string) => prop.split(':').map(value => value.trim()))
        })
]);
```

UNIT TESTING

```
let appTree: UnitTestTree;
beforeEach(() => {
    appTree = Tree.empty()
});
```

```
it('should add the files to the app tree', () => {

    const tree = runner.runSchematic('rxjs-storage', { name: 'myTest', props: 'la
titude:string,longitude:string'  }, appTree.branch());

    const appFiles = tree.files.filter(file => file.startsWith('/src/app'));
    expect(appFiles).toContain('/src/app/+state/my-test/my-test-state.ts');
    const fileContent = tree.read('/src/app/domain/my-test.ts')?.toString();
    expect(fileContent).toContain('latitude: string;');
    expect(fileContent).toContain('longitude: string;');
});
```

ADDING SUPPORT FOR ANGULAR CLI

PROJECT DIRECTORY

Angular uses **angular.json** to determine project directory. We'll make use of that.

GETTING THE PROJECT DIRECTORY BY PROJECT NAME

```
import { SchematicsException } from '@angular-devkit/schematics';
import { parseName } from '@schematics/angular/utility/parse-name';
import { buildDefaultPath } from '@schematics/angular/utility/project';

function getPath(tree: Tree, _options: Schema) {
    const workspaceConfigBuffer = tree.read('angular.json');
    if (!workspaceConfigBuffer) {
        throw new SchematicsException("Not an Angular CLI workspace");
    }
    const workspaceConfig = JSON.parse(workspaceConfigBuffer.toString());
    const project = workspaceConfig.projects[_options.project || workspaceConfig.
defaultProject];
    const defaultProjectPath = buildDefaultPath(project);
    return parseName(defaultProjectPath, _options.name);
}
```

```
ng generate ..:rxjs-storage subfolder/myData/entity-name
```

files/* -> src/app/**subfolder/my-data**/*

MOVING THE TEMPLATE FILES INTO THE PROJECT DIRECTORY

```
function copyFiles(_options: Schema): Rule {
    return (tree: Tree, context: SchematicContext) => {
        const { name, path } = getPath(tree, _options);
        const sourceTemplates = url('./files');
        const sourceParameterizedTemplates = apply(sourceTemplates, [
            template({
                ..._options,
                ...strings,
                props: _options.props.split(',')
                    .filter((prop: string) => !!prop.trim())
                    .map((prop: string) => prop.split(':').map(value => value.tri
m())),
                name
            }),
            move(path)
        ]);
        return mergeWith(sourceParameterizedTemplates)(tree, context);
    }
}
```

Make sure your **files**-directory no longer contains the sub-directories **src/app**.

TESTING?

Your workspace is not an Angular workspace -> Error when running schematic

Create an Angular project using **ng new example** and run the schematic there using:

```
ng generate ..:rxjs-storage --name=my-name
```

UNIT TESTING

Update beforeEach:

```
beforeEach(async () => {
    appTree = runner.runExternalSchematic(
        '@schematics/angular',
        'workspace',
        { name: 'test-workspace', version: '9' }
    );
    appTree = await runner.runExternalSchematicAsync(
        '@schematics/angular',
        'application',
        { name: 'test-app' },
        appTree
    ).toPromise();
});
```

NG-ADD SUPPORT

ng add installs package and runs ng-add schematic.

Just create a schematic named **ng-add** and publish your package.

NG-UPDATE SUPPORT

Add to package.json:

```
"ng-update": {
    "migrations": "./src/migrations.json"
}
```

Content of migration.json

```
{
    "$schema": "./node_modules/@angular-deckit/schematics/collection-schema.json"
,
    "schematics": {
        "migration-v8": {
            "version": "8",
            "description": "My schematic to update from previous versions to vers
ion 8",
            "factory": "./ng-update/index#updateToV8"
        }
        }
    }
```

THANKS FOR WATCHING!

Now get creative!