Overview

Many of the documentation pages contain snippets of code examples. These snippets are extracted from real working example applications, which are stored in sub-folders of the aio/content/examples/ folder. Each example can be built and run independently. Each example also provides tests (mostly e2e and occasionally unit tests), which are run as part of our CircleCI `test_docs_examples*` jobs, to verify that the examples continue to work as expected, as changes are made to the core Angular libraries.

In order to build, run and test these examples independently, you need to install dependencies into their sub-folder. Also there are a number of common boilerplate files that are needed to configure each example's project. These common boilerplate files are maintained centrally to reduce the amount of effort if one of them needs to change.

> ***Note for Windows users***
>
> *Setting up the examples involves creating some symbolic links (see here for details). On Windows, this requires to either have Developer Mode enabled (supported on Windows 10 or newer) or run the setup commands as administrator.*

Boilerplate overview

As mentioned above, many of the documentation pages contain snippets extracted from real example applications. To achieve that, all those applications need to contain some basic boilerplate, such as a `node_modules/` folder, a `package.json` file with scripts and dependencies, etc.

There are also different project types, each with its own boilerplate. For example, there are projects based on the Angular CLI, projects that use AngularJS, Custom Elements, i18n, server-side rendering, etc. (See the example configuration section below for more info on how to specify the project type.)

To avoid having to maintain the boilerplate in each example, we use the example-boilerplate-js script to provide a set of files that works across all the examples of a specific type.

Boilerplate files

Inside shared/boilerplate/ there is a sub-folder with boilerplate files for each of the different project types.

Currently, the following project types are supported:

- `cli` : For example apps based on the Angular CLI. This is the default type and is used in the majority of the examples.
- `cli-ajs` : For CLI-based examples that also use AngularJS (but not via `@angular/upgrade`).
- `elements` : For CLI-based examples that also use `@angular/elements` .
- `getting-started` : For the "Getting started" tutorial. Essentially the same as `cli` but with custom CSS styles.
- `i18n` : For CLI-based examples that also use internationalization.
- `schematics` : For CLI-based examples that include a library with schematics.
- `service-worker` : For CLI-based examples that also use `@angular/service-worker` .
- `systemjs` : For non-CLI legacy examples using SystemJS. This is deprecated and only used in few examples.
- `testing` : For CLI-based examples that are related to unit testing.
- `universal` : For CLI-based examples that also use `@nguniversal/express-engine` for SSR.

There are also the following special folders:

- `common` : Contains files used in many examples. (See the next section for info on how to exclude common files in certain examples.)

The `example-config.json`

Each example is identified by an `example-config.json` configuration file in its root folder. This configuration file indicates what type of boilerplate this example needs and how to test it. For example:

```
{
  "projectType": "cli"
}
```

The file is expected to contain a JSON object with zero or more of the following properties:

- `projectType: string` : One of the supported project types (see above). Default: `"cli"`
- `useCommonBoilerplate: boolean` : Whether to include common boilerplate from the common/ folder. Default: `true`
- `"overrideBoilerplate": string[]` : A list of paths to boilerplate files that are overridden by custom files in this example. Commonly this is used when a boilerplate file is referenced in a guide and so needs to have doc-regions added. When adding such overrides, ensure that the file is "unignored" by adding an appropriate negation pattern to the `content/examples/.gitignore` file.

SystemJS-only properties:

- `build: string` : The npm script to run in order to build the example app. Default: `"build"`
- `run: string` : The npm script to run in order to serve the example app (so that e2e test can be run against it). Default `"serve:e2e"`

CLI-only properties:

- `tests: object[]` : An array of objects, each specifying a test command. This can be used to run multiple test commands in series (for example, to run unit and e2e tests). The commands are specified as `{cmd: string, args: string[]}` and must be in a format that could be passed to Node.js' `child_process.spawn(cmd, args)` . You can use a special `{PORT}` placeholder, that will be replaced with the port on which the app is served during the actual test. Default:

  ```
  [
    {
      "cmd": "yarn",
      "args": [
        "e2e",
        "--configuration=production",
        "--protractor-config=e2e/protractor-puppeteer.conf.js",
        "--no-webdriver-update",
        "--port={PORT}"
      ]
    }
  ]
  ```

An empty `example-config.json` file is equivalent with `{"projectType": "cli"}` .

A `node_modules/` to share

With all the boilerplate files in place, the only missing piece is the installed packages. For that we have shared/package.json, which contains **all** the packages needed to run any example type.

Upon installing these dependencies, a shared/node_modules/ folder is created. This folder will be **symlinked** into each example. So it is not a copy like the other boilerplate files.

End-to-end tests

End-to-end infrastructure is slightly different between CLI- and SystemJS-based examples.

For CLI-based examples, create an `app.e2e-spec.ts` file inside the `e2e/` folder. This will be picked up by the default testing command (see the example configuration section above). If you are using a custom test command, make sure e2e specs are picked up (if applicable).

For SystemJS-based examples, create an `e2e-spec.ts` file inside the example root folder. These apps will be tested with the following command (and an optional `outputFile` to receive log messages):

```
yarn protractor [--params.outputFile=path/to/logfile.txt]
```

`example-boilerplate.js`

The example-boilerplate.js script manages the dependencies for all examples.

- `example-boilerplate.js add` : create the `node_modules/` symlinks and copy the necessary boilerplate files into example folders.
- `example-boilerplate.js remove` : remove all the boilerplate files from examples. It uses `git clean -xdf` to do the job. It will remove all files that are not tracked by git, **including any new files that you are working on that haven't been staged yet.** So, be sure to commit your work before removing the boilerplate.
- `example-boilerplate.js list-overrides` : print out a list of all example files that override boilerplate files. This is useful when updating the boilerplate files to a new version of Angular.

`run-example-e2e.mjs`

The run-example-e2e.mjs script will find and run the e2e tests for all examples. Although it only runs e2e tests by default, it can be configured to run any test command (for CLI-based examples) by using the `tests` property of the example-config.json file. It is named `*-e2e` for historical reasons, but it is not limited to running e2e tests.

See aio/README.md for the available command-line options.

Running the script will create an `aio/protractor-results.txt` file with the results of the tests.

`create-example.js`

The create-example.js script creates a new example under the `aio/content/examples` directory.

You must provide a new name for the example. By default the script will place basic scaffold files into the new example (from shared/example-scaffold). But you can also specify the path to a separate CLI project, from which the script will copy files that would not be considered "boilerplate". See the Boilerplate overview for more information.

Updating example dependencies

With every major Angular release, we update the examples to be on the latest version. See UPDATING.md for instructions.