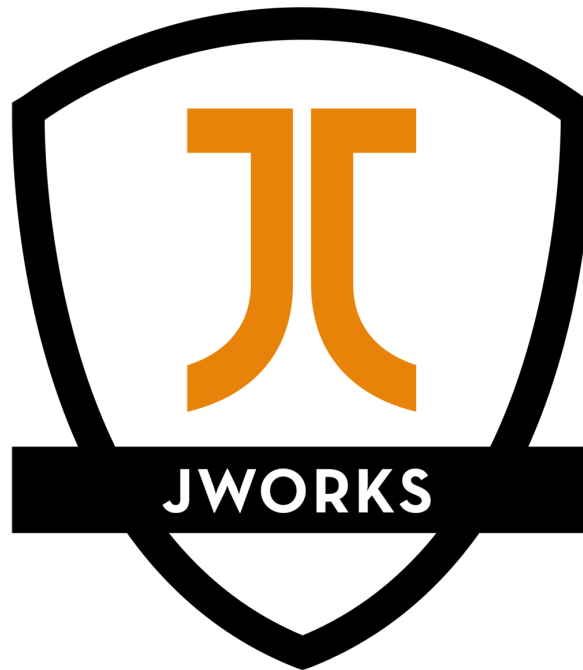


FRONTEND TOOLS



POWERED BY ORDINA

HI, MY NAME IS DIMITRI.

Frontend Developer @ Ordina Belgium

<https://github.com/dimidekerf>

WHAT ARE WE GOING TO TALK ABOUT?

- Package managers
- Build tools
- Linting
- Testing
- Generators

PACKAGE MANAGERS



NPM

Node Package Manager installs and manages dependencies in your project powered by the largest software registry

<https://www.npmjs.com/>

NPM PACKAGE

- Directory with files
- package.json file which holds metadata
- Mostly small packages to solve a certain problem

PACKAGE.JSON

Manage local npm packages the easy way

- List packages for your project
- Specify version of each package
- Share your build with others

PACKAGE.JSON

Create one by running `npm init`

```
{
  "name": "MyProject", // Required
  "version": "1.0.0", // Required
  "description": "Showing off my npm skills",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "dependencies": {
    // Your dependencies
    "@angular/core": "^4.4.4"
  },
  "devDependencies": {
    // Your dev dependencies
  }
}
```

NPM CLI

Command Line Interface to install packages, managing versions and dependencies

NPM CLI

Install the CLI by installing Node.js and check if npm works

```
$ npm -v
```

Update the CLI globally

```
$ npm install npm@latest -g
```

INSTALL PACKAGES

```
$ npm install @angular/core
```

INSTALL FLAGS

Dependencies with `--save`

```
$ npm install @angular/core --save
```

Dev dependencies with `--save-dev`

```
$ npm install @angular/compiler-cli --save-dev
```

Global dependencies with `-g`

```
$ npm install @angular/core -g
```

SAVE OR SAVE-DEV?

- Save: packages where your app relies on (Angular)
- Save-dev: packages for development only (Building, Testing)

UPDATE PACKAGES

Update your local packages

```
$ npm update
```

Check outdated versions of local packages

```
$ npm outdated
```

DELETE PACKAGES

```
$ npm uninstall @angular/core
```


NODE_MODULES

npm will place all required packages in node_modules folder at the root of your project

SEMANTIC VERSIONING

Standard to communicate which kind of changes are in that release

- Patch releases: ~1.0.0 -> 1.0.7
- Minor releases: ^1.0.0 -> 1.5.0
- Major releases: * -> 2.0.0

SCOPED PACKAGES

Puts packages under a scope with the '@'-sign, such as
@angular/core



BOWER

Package manager which installs versions of packages and their dependencies for your project

<https://bower.io/>

- Currently only maintained
- Look for alternatives



YARN

Fast, reliable and secure dependency management

<https://yarnpkg.com/en/>

FAST

Yarn runs tasks in parallel and uses a cache which results in faster installation of packages

RELIABLE

Lockfile prevents other versions of packages to be installed

SECURE

Yarn uses checksums for packages to be installed

YARN COMMANDS

Initialize a project

```
$ yarn init
```

Install all packages

```
$ yarn
```

```
or
```

```
$ yarn install
```

YARN COMMANDS

Install a package

```
$ yarn add @angular/core
```

Install a dev package

```
$ yarn add @angular/compiler-cli --dev
```

Update a package

```
$ yarn upgrade @angular/core
```

Uninstall a package

```
$ yarn remove @angular/core
```

TRYOUTS

Let's try out some commands we've just learned

BUILD TOOLS

WHY?

Automate your workflow to speed up development

REPETITIVE TASKS

- Compilation
- Minification
- Testing
- Linting
- Live reloading

WE'RE LAZY... BUILD TOOLS TO THE RESCUE!

Watch for file changes and run certain tasks for us



GRUNT

GRUNT

Grunt is a well known task runner with hundreds of plugins which can be run from cmd

<https://gruntjs.com/>

SETUP GRUNT

Install the grunt-cli

```
$ npm install -g grunt-cli
```

Create a Gruntfile.js file to configure Grunt

GRUNTFILE

```
module.exports = function(grunt) {  
  grunt.initConfig({  
    uglify: {  
      build: {  
        'build/app.min.js': ['src/app.js']  
      }  
    }  
  });  
  grunt.loadNpmTasks('grunt-contrib-uglify'); // Load Uglify plugin  
  grunt.registerTask('default', ['uglify']); // Configure default task  
};
```

GRUNTFILE

- Wrapper function
- Project and task configuration
- Loading Grunt plugins
- Register tasks

WRAPPER FUNCTION

Encapsulates the Grunt configuration

```
module.exports = function(grunt) {  
  // Grunt configuration  
};
```

INITIALIZE CONFIGURATION

```
module.exports = function(grunt) {  
  grunt.initConfig({  
  
    });  
};
```


PROJECT AND TASK CONFIGURATION

Configure Grunt plugins to your specific needs. The configuration object often has the same name as the plugin

```
module.exports = function(grunt) {  
  grunt.initConfig({  
    // Configure the uglify plugin with a build target  
    uglify: {  
      build: {  
        'build/app.min.js': ['src/app.js']  
      }  
    }  
  });  
};
```

LOADING GRUNT PLUGINS

```
module.exports = function(grunt) {  
  grunt.initConfig({  
    // Tasks...  
  });  
  
  // Load Uglify plugin  
  grunt.loadNpmTasks('grunt-contrib-uglify');  
};
```

REGISTER GRUNT TASKS

Register tasks with Grunt, optionally let other tasks run first by specifying them inside the array behind the task name

```
module.exports = function(grunt) {  
  grunt.initConfig({  
    // Tasks...  
  });  
  // Load Uglify plugin  
  grunt.loadNpmTasks('grunt-contrib-uglify');  
  
  // Configure default task, run uglify task first  
  grunt.registerTask('default', ['uglify']);  
};
```

Default task executes when running `grunt` in cmd

BRINGING IT ALL TOGETHER

```
module.exports = function(grunt) { // Wrapper function
  grunt.initConfig({
    uglify: { // Configure uglify plugin
      build: {
        'build/app.min.js': ['src/app.js']
      }
    }
  });
  grunt.loadNpmTasks('grunt-contrib-uglify'); // Load Uglify plugin
  grunt.registerTask('default', ['uglify']); // Register default task
};
```

TRYOUTS

Let's setup a Grunt configuration



GULP

Faster task runner with easier setup compared to Grunt

<https://gulpjs.com/>

SETUP GULP

- Install the gulp-cli
- Create a gulpfile.js fome
- Run the gulp command in cmd

```
$ npm install -g gulp-cli
```

```
$ gulp
```


GULP API

- gulp.src
- pipe
- gulp.dest
- gulp.task
- gulp.watch

GULPFILE

```
var gulp = require('gulp');
var rename = require('gulp-rename');
var uglify = require('gulp-uglify');

gulp.task('default', function() {
  return gulp.src('src/app.js')
    .pipe(gulp.dest('build/'))
    .pipe(uglify())
    .pipe(rename({ extname: '.min.js' }))
    .pipe(gulp.dest('build/'));
});
```

GULP & PLUGINS

Load in Gulp and the plugins we're going to use

```
var gulp = require('gulp');  
var rename = require('gulp-rename');  
var uglify = require('gulp-uglify');
```

TASK

Define a Gulp task which we can run afterwards

```
gulp.task('default', function() { // Default task runs when entering gulp in cmd
  // Return a Gulp pipeline here
});
```

MULTIPLE TASKS

Run another Gulp task before executing current

```
gulp.task('default', ['clean'], function() { // Trigger the clean task first
  // Return a Gulp pipeline here
});

gulp.task('clean', function () {
  return del(['dist']);
});
```

SOURCE FILE

Tell Gulp which file(s) to start from

```
gulp.src('src/app.js') // Single file
```

```
gulp.src(['src/app.js', 'src/vendor.js']) // Multiple files
```

```
gulp.src(['src/*.js']) // Wildcard selector
```

PIPES & PLUGINS

Use a pipe to stream the source to the destination, while plugins will perform modifications on that stream

```
.pipe(gulp.dest('build/'))  
.pipe(uglify())  
.pipe(rename({ extname: '.min.js' })))  
.pipe(gulp.dest('build/'))
```

USEFUL PLUGINS

- gulp-concant
- gulp-uglify
- gulp-rename
- gulp-typescript
- A lot more are available on npm

DESTINATION FILE

Write the output stream to the given directory

```
gulp.dest('build/')
```

BRINGING IT ALL TOGETHER

```
var gulp = require('gulp');
var rename = require('gulp-rename');
var uglify = require('gulp-uglify');

gulp.task('default', function() {
  return gulp.src('src/app.js')
    .pipe(gulp.dest('build/'))
    .pipe(uglify())
    .pipe(rename({ extname: '.min.js' }))
    .pipe(gulp.dest('build/'));
});
```

LIVE RELOADING

Reload webpage when code changes occur

```
var browserSync = require('browser-sync');
var reload = browserSync.reload;

gulp.task('browser-sync', function() {
  browserSync({
    server: {
      baseDir: "./"
    }
  });
});

gulp.task('default', function () {
  gulp.watch('js/*.js').on('change', reload);
});
```

TRYOUTS

Let's setup a Gulp configuration



SIMILARITIES

- Tasks runners using Node.js
- Plugins to perform tasks
- Large community

DIFFERENCES

- Accomplishing tasks
- Speed
- Code vs Configuration

ACCOMPLISHING TASKS

Grunt plugin does often many tasks at once, while a Gulp plugin does one task very well

SPEED

Gulp runs tasks faster by using streams, performing tasks in memory and running tasks in parallel

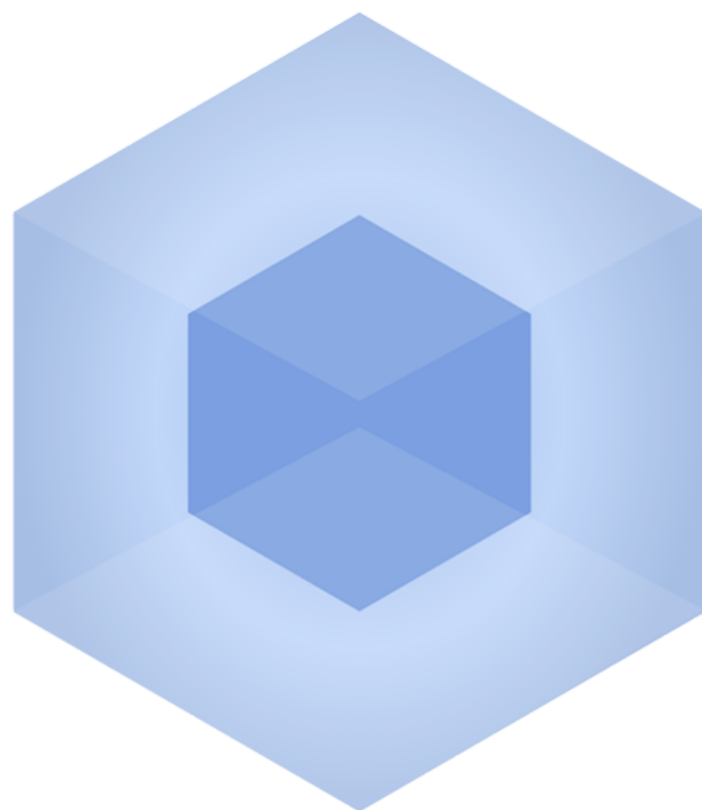
CODE VS CONFIGURATION

Grunt has a JSON configuration schema, whereas Gulp relies on streams writing in JavaScript

WHICH ONE SHOULD I PICK?

Depends on your preference and experience in both

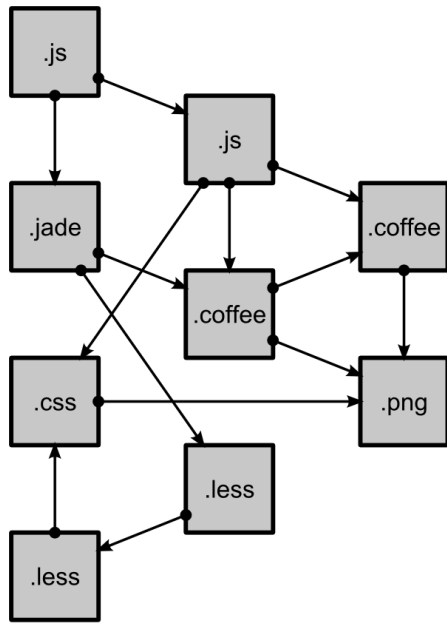
**BUT THERE IS A NEW KID ON THE
BLOCK...**



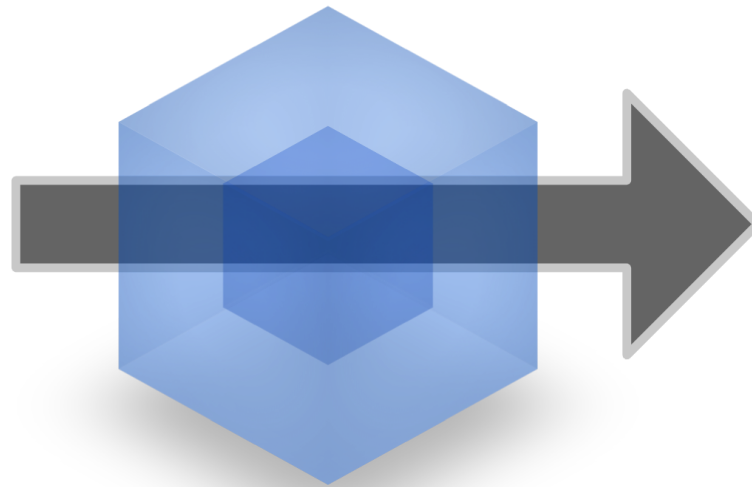
WEBPACK

Module bundler by building a dependency graph of all modules required by your application

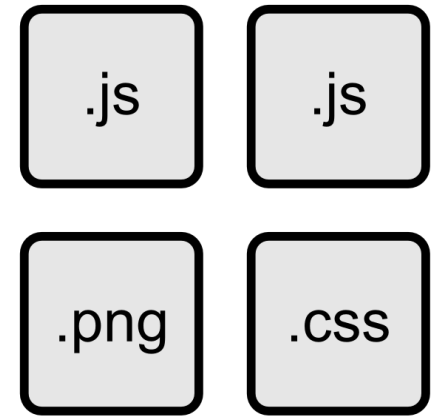
<https://webpack.js.org/>



modules
with dependencies



webpack
MODULE BUNDLER



static
assets

FOUR CORE CONCEPTS

- Entry
- Output
- Loaders
- Plugins

ENTRY

- Entry point of your application
- Webpack follows dependency graph starting from Entry to know what to bundle

```
entry: './app/index.js'
```

OUTPUT

Output where Webpack will place the bundled code

```
output: {  
  path: path.resolve(__dirname, 'dist'),  
  publicPath: 'http://www.my-cdn.com',  
  filename: 'bundle.js'  
}
```

LOADERS

- Every file is a module
- But Webpack only understands JavaScript...
- Loaders will transform files into modules

```
module: {  
  rules: [  
    { test: /\.html$/, use: 'html-loader' }  
  ]  
}
```

SOME USEFUL LOADERS

- HTML loader
- TS & TSLint loader
- SASS, CSS, Style loader
- File loader

PLUGINS

- Apply actions on chunks of bundled modules
- Many plugins provided by Webpack
- Customizable via options

```
plugins: [  
  new webpack.optimize.UglifyJsPlugin(),  
  new HtmlWebpackPlugin({template: './src/index.html'})  
]
```

SOME USEFUL PLUGINS

- UglifyJsPlugin
- HotModuleReplacementPlugin
- CommonsChunkPlugin
- More available at: <https://webpack.js.org/plugins/>

WEBPACK CONFIGURATION

```
const webpack = require('webpack');
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');

const config = {
  entry: './app/index.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'bundle.js'
  },
  module: {
    rules: [
      { test: /\.html$/, use: 'html-loader' }
    ]
  },
  plugins: [
    new webpack.optimize.UglifyJsPlugin(),
    new HtmlWebpackPlugin({template: './src/index.html'})
  ]
};
```

WEBPACK DEV SERVER

- Node server to host your application
- Enables features like Live reloading and Hot Module Replacement

SOURCEMAPS

- Mapping between original and transformed source code
- Different types of sourcemaps
 - Higher quality, but slower build speeds
 - Development vs production

HOT MODULE REPLACEMENT

- Replaces modules when application is running
- Watch code or style changes without refreshing the browser
- Speeds up development process

```
const config = {
  ...
  devServer: {
    hot: true,
    inline: true
  },
  plugins: [
    new webpack.HotModuleReplacementPlugin(),
  ]
  ...
};

module.exports = config;
```

TREE SHAKING

Eliminates unused code from bundle by only including used code

TRYOUTS

Configure Webpack to bundle an app

LINTING

LINTING

Run a program that will analyse code for for readability, maintainability, and functionality errors

WHY SHOULD I USE IT?

- Instantly feedback
- Safer code
- Same coding style in team

JSLINT

Checks JavaScript code following the standards defined by Douglas
Crockford

JSHINT

More flexible than JSLint, community-driven

TSLINT

Static analysis tool for TypeScript code

HOW TO LINK MY CODE?

Usually a step inside the build proces of your app, for example when running tests or building the app

INTEGRATION WITH EDITOR

Most code editors recognize lint configurations and show warnings when violating lint standards

TESTING

WHY?

Check if your code works as expected and also stays working that way

TYPES OF TESTING

- Unit testing
- Intergration testing
- E2E testing

UNIT TESTING

- Code that tests a unit
- Module, Class, method
- Isolate the unit by mocking

MOCKING?

Provide a mocked instance for the dependencies of the unit

UNIT-TESTABLE CODE

- Seperate code to make it unit-testable
- Little dependecies and no I/O operations

LETS GO TESTING!

But first, some tooling...



KARMA

- Test runner that runs your tests in a browser
- Encourages Test-Driven Development
- Developed by the Angular team

TEST FRAMEWORKS

Combination of protocols, rules, standards and guidelines to write automated tests



JASMINE

- Behaviour-driven development framework
- Built-in assertions allows for fluent syntax
- Built-in spy library



MOCHA

- Behaviour-driven development framework
- Does not have assertion library built in
 - Chai
 - expect.js
- Does not have spy library built in
 - Sinon

BUT WHICH ONE IS THE BEST?

- Depends on your preference
- Let's stick with Jasmina for now

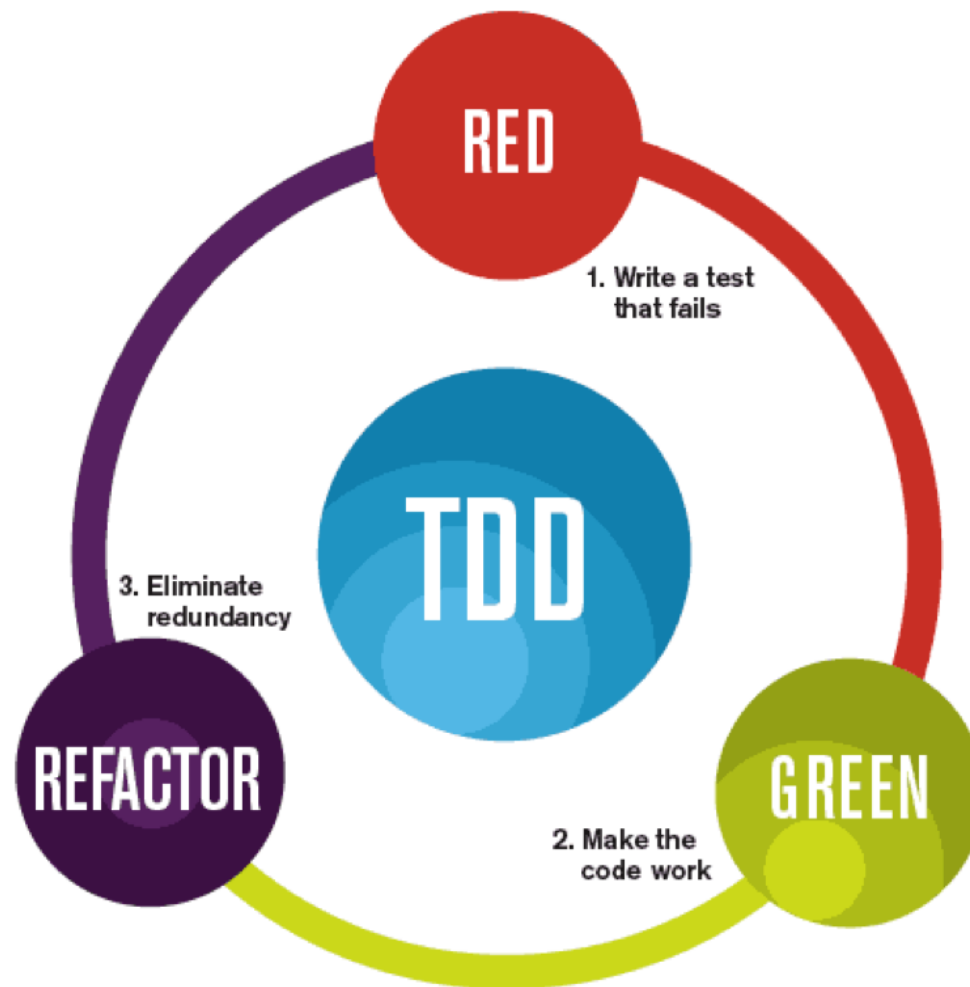
TEST DRIVEN DEVELOPMENT

Repetition of short development cycle

- Phase 1: Write test
- Phase 2: ...
- Phase 3: Profit!

TEST DRIVEN DEVELOPMENT

1. Add a test which defines a new function
2. Run the test and see it failing
3. Write necessary code to let test pass
4. Run all tests to check if everything passes
5. Refactor code



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

BEHAVIOUR DRIVEN DEVELOPMENT

Describe the behaviour of the app and provide a test

TEST EXAMPLE

```
describe('Calculator', function () {  
  let calculator: Calculator;  
  
  beforeEach(() => {  
    calculator = new Calculator();  
  })  
  
  it('should calculate the sum', () => {  
    expect(calculator.add(2, 2)).toEqual(4);  
  })  
})
```


DESCRIBE

Suite describing the part you want to test of your app

```
describe('Calculator', function() {  
    ...  
})
```

BEFOREEACH

Runs a code block before each tests, ideal to initiate a class, etc...

```
beforeEach(() => {  
    calculator = new Calculator();  
})
```

AFTEREACH

Runs a code block after each tests, can be used to reset variables

```
afterEach(() => {  
  // Reset variables to initial state  
})
```

IT

Spec describing what the code should do, located inside the suite

```
it('should calculate the sum', () => {  
  ...  
})
```

MATCHER

Check if the result is what we expected

```
expect(calculator.add(2, 2)).toEqual(4);
```

Use not keyword to inverse the matcher

```
expect(calculator.add(2, 2)).not.toEqual(5);
```

MORE MATCHERS

- toBeDefined()
- toBeTruthy()
- toContain()
- nothing()
- ...

Jasmine matchers

CUSTOM MATCHERS

Add your own matcher to make your tests more readable

```
beforeEach(() => {
  jasmine.addMatchers({
    toBeFour: () => {
      return this.actual === 4;
    }
  });
})

it('should calculate the sum', () => {
  expect(calculator.add(2, 2)).toBeFour();
})
```

SPY

Stub any function and track calls to it

SPY

```
describe('Calculator', function() {
  const calculator;

  beforeEach(() => {
    calculator = new Calculator();
    spyOn(calculator, 'add');
  })

  it('should have called sum', () => {
    calculator.add(2, 2);

    expect(calculator.add).toHaveBeenCalled();
    expect(calculator.add).toHaveBeenCalledWith(2, 2);
  })
})
```

TEST COVERAGE

Check amount of code covered by tests

- Coverage is displayed by percentage
- Some thrive to 100% code coverage
- Not most important, write meaningful tests

ISTANBUL

- Code coverage tool for JavaScript
- Generates reports which highlights covered lines of code

100% Statements 35/35 100% Branches 8/8 100% Functions 8/8 100% Lines 33/33

File ▾												Statements ▾		Branches ▾		Functions ▾		Lines ▾	
calculator/		<div></div>		100%		6/6		100%		0/0		100%		3/3		100%		5/5	

100% Statements 6/6

100% Branches 0/0

100% Functions 3/3

100% Lines 5/5

```
1 1× export class Calculator {
2
3 1×   public add(value1: number, value2: number): number {
4 1×       const sum = value1 + value2;
5 1×       return sum
6       }
7
8 1× }
```

INTEGRATION TESTING

Testing more than one unit, but not all of them at once

E2E TESTING

Test the whole application like a user would in an automated way,
where all parts of the app are working together

HOW MANY E2E TESTS?

- Functionality is already tested, check if everything works together
- E2E Tests are slow
- Some tests tend to be flaky

WHAT'S NEEDED?

- Tool to automate our tests
- A browser



PROTRACTOR

- E2E test framework for Angular
- Automatically waits until page is loaded

SETUP

Install Protractor globally

```
$ npm install -g protractor
```

This installs two tools, Protractor and Webdriver-manager.
Update the driver first

```
$ webdriver-manager update
```

Startup the webdriver afterwards

```
$ webdriver-manager start
```

CONFIGURATION

Protractor needs a config file to specify where tests are located, which browser to use,...

```
exports.config = {  
  framework: 'jasmine',  
  seleniumAddress: 'http://localhost:4444/wd/hub',  
  specs: ['./e2e/**/*.spec.js']  
}
```

OUR FIRST E2E TEST

```
describe('Protractor Demo', function() {  
  it('should have a title', function() {  
    browser.get('http://www.my-app.com/'); // browser is provided by Protractor  
    expect(browser.getTitle()).toEqual('My App');  
  });  
});
```

INTERACTION WITH BROWSER

```
describe('Protractor Google Demo', function() {  
  it('should login the user', function() {  
    browser.get('http://www.my-app.com/'); // browser is provided by Protractor  
    element(by.id('username')).sendKeys('User 1'); // element as well  
    element(by.id('password')).sendKeys('pass');  
    element(by.tagName('button')).click();  
    expect(element(by.id('welcome')).isDisplayed()).toBeThruthy();  
    expect(element(by.id('welcome')).getText()).toEqual('Welcome User 1');  
  });  
});
```

GENERATORS



YEOMAN

YEOMAN

Generating complete projects using plugins

<http://yeoman.io/>

SETUP

Install Yeoman globally on your system

```
$ npm install -g yo
```

Choose a **generator**

```
$ npm install -g generator-awesome
```

BASIC SCAFFOLDING

Run the generator with the `yo` command

```
$ yo awesome
```



ANGULAR CLI

"The Angular CLI makes it easy to create an application that already works, right out of the box. It already follows our best practices!"

Angular CLI team

INSTALL

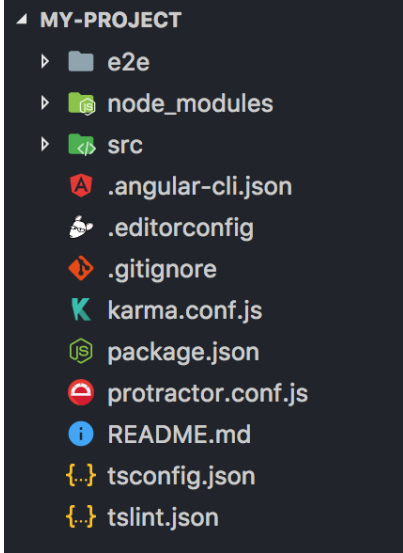
Install Angular CLI globally which enables `ng` commands in terminal

```
$ npm install -g @angular/cli
```

NEW PROJECT

Start a project by just typing the following command

```
$ ng new my-project
```



▲ MY-PROJECT

- e2e
- node_modules
- src
- ▲ .angular-cli.json
- 📄 .editorconfig
- 📄 .gitignore
- 📄 karma.conf.js
- 📄 package.json
- 📄 protractor.conf.js
- 📄 README.md
- 📄 tsconfig.json
- 📄 tslint.json

NEW PROJECT

The `npm new my-project` command will generate a project for you with everything set up!

- Development
- Unit & E2E testing
- Build for production

DEVELOPMENT

Serve the application locally with auto-reloading enabled

```
$ ng serve
```

UNIT TESTS

Test your code in a TDD way!

```
ng test
```

If you like a code coverage report, pass the --cc flag with it

```
$ ng test --c
```

E2E TESTS

Test the flow of your app by providing tests inside the e2e subfolder

```
$ ng e2e
```

BUILD

Build your app

```
$ ng build
```

DEV VS PRODUCTION BUILD

Generates sourcemaps, useful for debuggin

```
$ ng build --dev
```

Enables unused code elimination and AOT

```
$ ng build --prod
```

RECAP

- Create new project

```
$ ng new my-project
```

- Serving the application

```
$ ng serve
```

- Running unit tests

```
$ ng test
```

- Running end-to-end tests

```
$ ng e2e
```

- Building the application

```
$ ng build
```

GENERATE

Angular CLI can also generate components, services, modules, etc...
for you with the `ng generate` command!

COMPONENT

```
$ ng g component myComponent
```

Generates the Angular component, template, stylesheet and unit test spec file. The module will get updated with this new component.

SERVICE

```
$ ng g service myService
```

Generates the Angular service and a spec file for testing. Don't forget to import your new service in the module!

PIPE

```
$ ng g pipe myPipe
```

Generates the Angular pipe and a spec file. The pipe will be automatically imported in the module

INTERFACE

```
$ ng g interface myInterface
```

Generates an empty interface in an interface file

ENUM

```
$ ng g enum myEnum
```

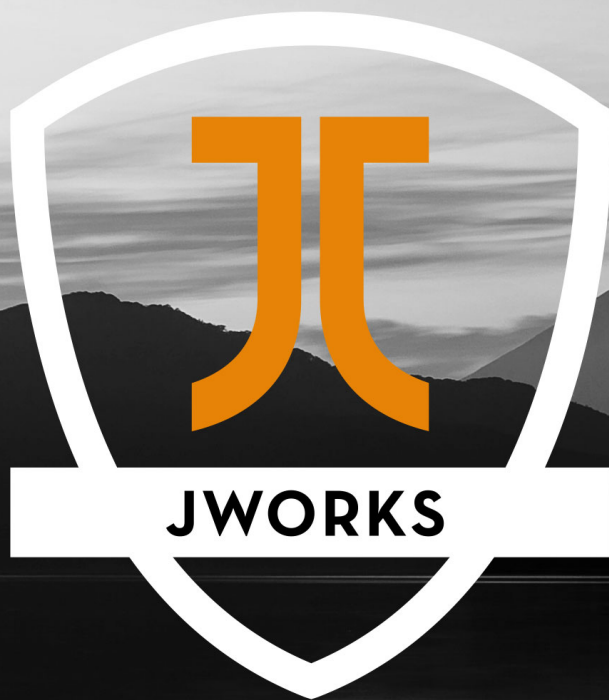
Generates an empty enum in a enum file

TRYOUTS

Create a new ng project and generate some blueprints

FINAL NOTES

Lots of other tools are available! Familiarize yourself with more options and find out which you prefer



POWERED BY  ORDINA