

Take Your Sitecore Project To The Next Level With Node.js

Presented by Anastasiya Flynn,
Sitecore Architect at  oarke



Agenda

This presentation will cover common tasks that Sitecore architects and developers encounter in projects, and I will show you how to leverage server-side JavaScript to solve these tasks.



Demo – Creating a Node Module



Setup

Automate the setup of a complex Sitecore project with **Yeoman**

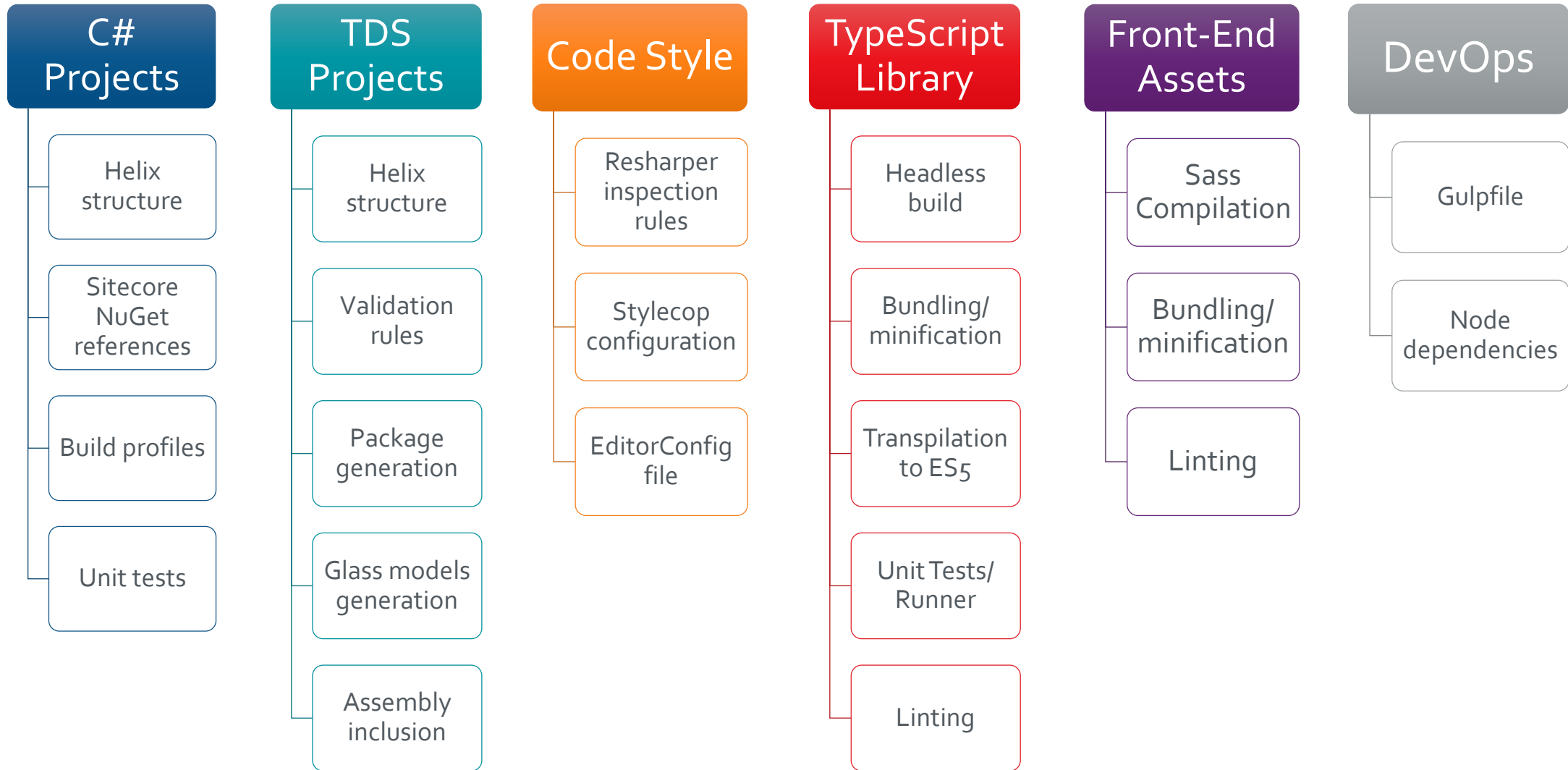


Setup

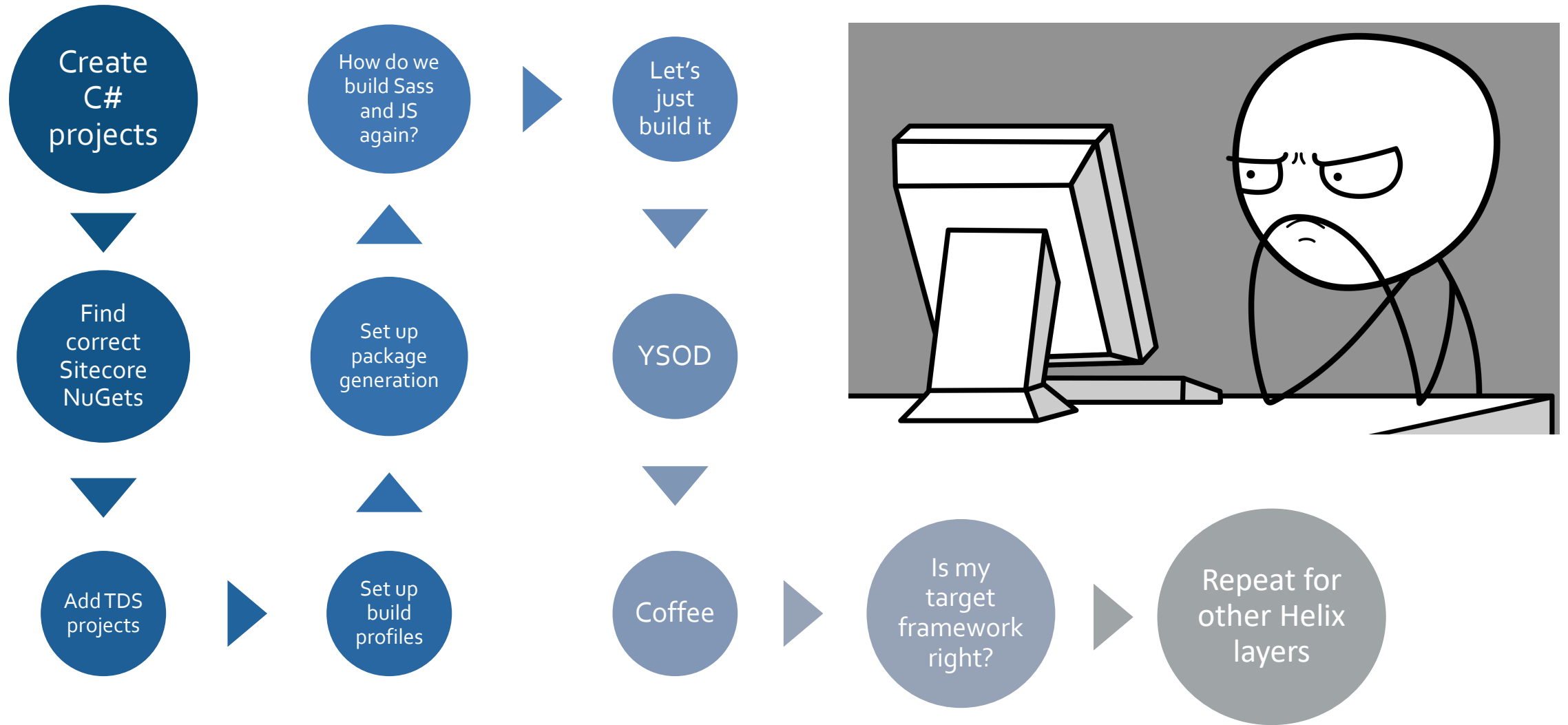
Development

Testing

New Solution Checklist



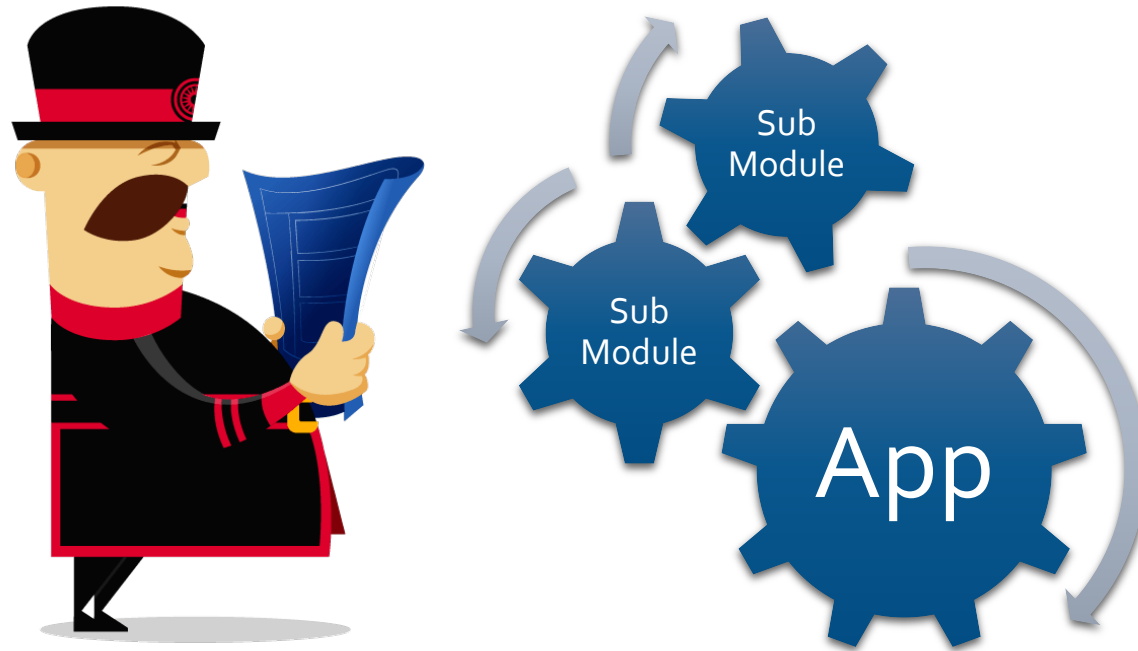
New Solution Checklist



Automate Project Setup With Yeoman

<http://yeoman.io/>

“Yeoman is a generic scaffolding system... Yeoman by itself doesn't make any decisions. Every decision is made by generators.”



Yeoman Generator – Setting Up



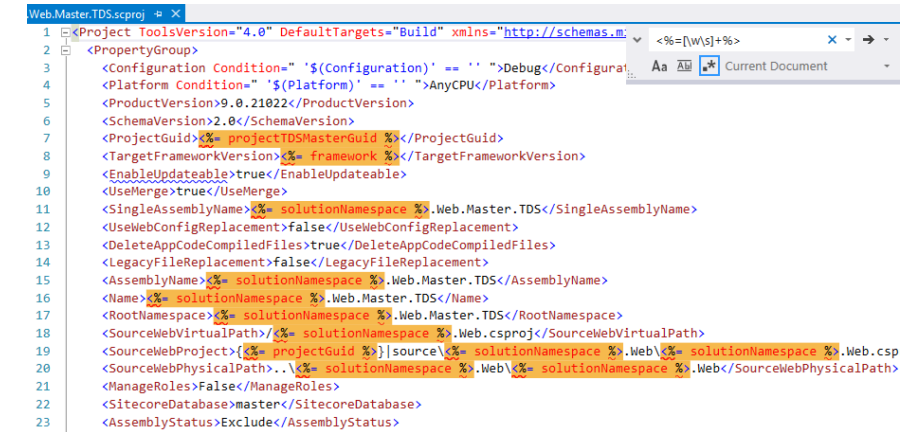
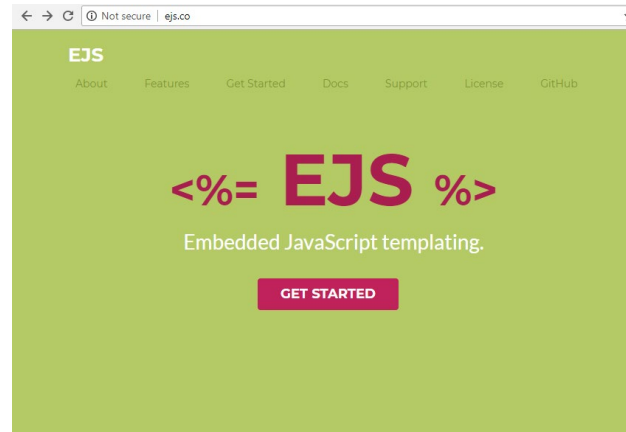
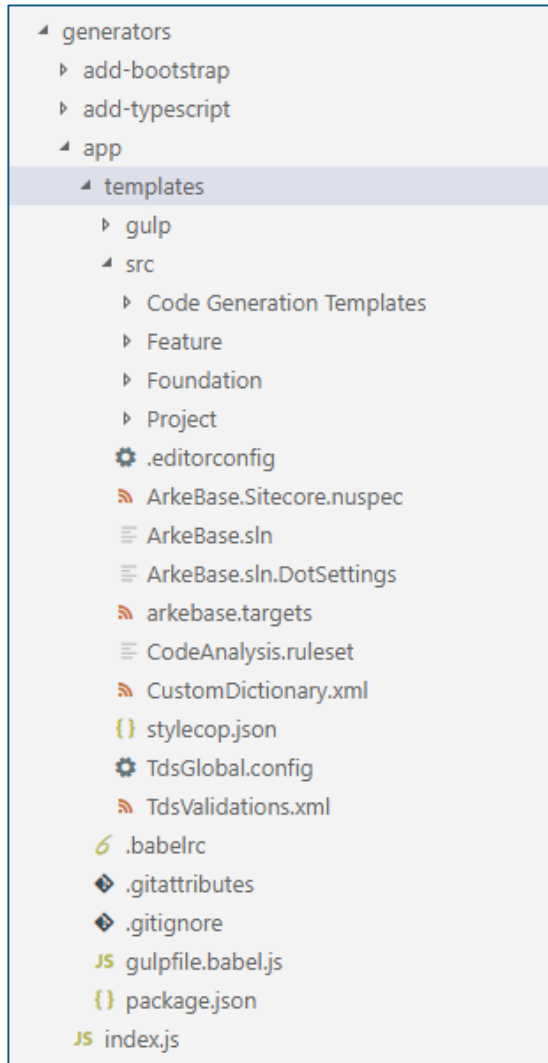
Yeoman – Setting Up

/package.json

The screenshot shows a VS Code editor with the Explorer sidebar on the left and the package.json file open in the main editor. The Explorer sidebar shows a project structure with folders for 'generator-flynn', 'add-bootstrap', 'add-typescript', and 'app'. The 'package.json' file is selected in the Explorer. The main editor displays the contents of 'package.json', which includes fields for name, version, description, author, license, keywords, devDependencies, and files. A blue arrow points to the 'generators' field in the 'files' array. To the right of the code editor, there is a diagram of three interlocking gears labeled 'add-bootstrap', 'add-typescript', and 'app'. Arrows indicate a clockwise flow from 'add-bootstrap' to 'add-typescript' to 'app'.

```
1 {
2   "name": "generator-flynn",
3   "version": "1.0.0",
4   "description": "Anastasiya Flynn's Sitecore Solution Generator",
5   "author": "Anastasiya Flynn",
6   "license": "ISC",
7   "keywords": [
8     "yeoman-generator",
9     "sitecore",
10    "helix"
11  ],
12   "devDependencies": {
13     "gulp-rename": "1.2.2",
14     "uuid": "^3.3.2",
15     "yeoman-generator": "^3.1.1"
16  },
17   "files": [
18     "generators"
19  ]
20 }
```

Yeoman – Template Variables



Solution
name

Solution
namespace

MVC area
name

Local
Sitecore URL

Sitecore
version

.NET
framework

Yeoman – Generator Lifecycle

/generators/app/index.js



The screenshot shows a VS Code editor with the Explorer sidebar on the left and the index.js file open in the editor. The Explorer sidebar shows a project structure with folders like 'generator-flynn', 'generators', 'app', and 'lib'. The 'index.js' file is selected. The editor shows the following code:

```
1  const Generator = require("yeoman-generator");
2
3  module.exports = class extends Generator {
4    ... constructor(args, opts) {
5      ... super(args, opts);
6    }
7
8    ... initializing() {
9      ... this.log("Generating Sitecore project");
10     ... }
11
12     ... _privateHelper() {
13
14     ... }
15   };
16
```

Initializing

Configuring

Writing

Install

Prompting

Default

Conflicts

End

/generators/app/index.js

```
EXPLORER: GENERATOR-FLYNN
├─ generators
│   └─ add-bootstrap
│   └─ add-typescript
├─ app
│   └─ templates
└─ JS index.js
  └─ lib
    └─ node_modules
      └─ .gitignore
      └─ package-lock.json
      └─ package.json
      └─ yarn.lock

JS index.js
1  const Generator = require("yeoman-generator"),
2    prompter = require("../lib/prompter"),
3    writer = require("../lib/writer");
4
5  module.exports = class extends Generator {
6    constructor(args, opts) {
7      super(args, opts);
8    }
9
10   initializing() {
11     this.log("Generating Sitecore project");
12
13     // Initialize an object to track project-specific settings that change each
14     // time the generator runs
15     this.settings = {};
16
17     // Escape single backslash with double-backslash for usage in strings
18     this.settings.solutionRoot = this.destinationPath().replace(/\\/g, "\\");
19   }
20
21   async prompting() {
22     // Prompt user for project-specific settings
23     await prompter.getSolutionInfo.call(this);
24     await prompter.getFrameworkInfo.call(this);
25   }
26
27   configuring() {
28     // Save all the settings that have been provided to config file
29     Object.keys(this.settings).forEach((key) => this.config.set(key, this.settings[key]));
30   }
31
32   writing() {
33     writer.writeConfigFiles.call(this);
34
35     // Copy Helix layers to destination, dynamically transforming files based on
36     // project-specific settings as needed
37     writer.writeTemplates.call(this);
38   }
39
40   install() {
41     // Install node modules in destination
42     this.yarnInstall();
43   }
44
45   end() {
46     this.log("Finished generating Sitecore project");
47   }
48 }
```

Initializing

- Log Message
- Initialize settings object



Prompting

- Prompt user for project-specific settings



Configuring

- Save all the settings that have been provided to config file



Writing

- Write settings to config file on disk
- Copy templates to destination, transforming content using project-specific settings



Install

- Install node modules in destination



End

- Log Message

/generators/app/index.js

/generators/lib/prompter.js

EXPLORER: GENERATOR-FLYNN

- generators
 - add-bootstrap
 - add-typescript
 - app
 - templates
 - lib
 - node_modules
 - .gitignore
 - package-lock.json
 - package.json
 - yarn.lock

index.js

```
1 const Generator = require("yeoman-generator"),
2   prompter = require("../lib/prompter"),
3   writer = require("../lib/writer");
4
5 module.exports = class extends Generator {
6   constructor(args, opts) {
7     super(args, opts);
8   }
9
10  initializing() {
11    this.log("Generating Sitecore project");
12
13    // Initialize an object to track project-specific settings
14    // time the generator runs
15    this.settings = {};
16
17    // Escape single backslash with double-backslash for usage
18    this.settings.solutionRoot = this.destinationPath().replace(/\\/g, '\\\\');
19  }
20
21  async prompting() {
22    // Prompt user for project-specific settings
23    await prompter.getSolutionInfo.call(this);
24    await prompter.getFrameworkInfo.call(this);
25  }
26
27  configuring() {
28    // Save all the settings that have been provided to config
29    Object.keys(this.settings).forEach((key) => this.config.set(key, this.settings[key]));
30  }
31
32  writing() {
33    writer.writeConfigFiles.call(this);
34
35    // Copy Helix layers to destination, dynamically transform
36    // project-specific settings as needed
37    writer.writeTemplates.call(this);
38  }
39
40  install() {
41    // Install node modules in destination
42    this.yarnInstall();
43  }
44
45  end() {
46    this.log("Finished generating Sitecore project");
47  }
48 };
```

prompter.js

```
1 /* ...
2
3 function getSolutionInfo() {
4   return this.prompt([
5     {
6       type: "input",
7       name: "solutionName",
8       message: "Your solution name",
9       default: this.appname // Default to current folder name
10     }, {
11       type: "input",
12       name: "solutionNamespace",
13       message: "Your .NET namespace base",
14       default: this.appname.replace(".", "")
15     }, {
16       type: "input",
17       name: "areaName",
18       message: "Your MVC area name base",
19       default: this.appname.replace(".", "")
20     }, {
21       type: "input",
22       name: "frontEndAssetPath",
23       message: "Your path to front-end assets directory (relative to the Project-level web project)",
24       default: "/Assets"
25     }, {
26       type: "input",
27       name: "localUrl",
28       message: "Domain of your local Sitecore instance",
29       required: true
30     }, {
31       type: "list",
32       name: "sitecoreVersion",
33       message: "Your Sitecore version",
34       choices: [
35         { name: "8.2 rev. 171121 (8.2 Update-6)", value: "8.2.171121" },
36         { name: "8.2 rev. 180406 (8.2 Update-7)", value: "8.2.180406" },
37         { name: "9.0 rev. 171219 (9.0 Update-1)", value: "9.0.171219" }
38       ],
39       required: true,
40       store: true
41     }
42   ]).then((answers) => {
43     Object.keys(answers).forEach((key) => this.settings[key] = answers[key]);
44   });
45 }
46
47 function getFrameworkInfo() {
48   /* ...
49
50   function getScriptInfo() {
51     /* ...
52
53   module.exports = {
54     getSolutionInfo: getSolutionInfo,
55     getFrameworkInfo: getFrameworkInfo,
56     getScriptInfo: getScriptInfo
57   };
58 };
```

/generators/app/index.js

```
EXPLORER: GENERATOR-FLYNN
generators
├─ add-bootstrap
├─ add-typescript
├─ app
│   └─ templates
└─ JS index.js
lib
node_modules
.gitignore
package-lock.json
package.json
yarn.lock

JS index.js x
1  const Generator = require("yeoman-generator"),
2    prompter = require("../lib/prompter"),
3    writer = require("../lib/writer");
4
5  module.exports = class extends Generator {
6    constructor(args, opts) {
7      super(args, opts);
8    }
9
10   initializing() {
11     this.log("Generating Sitecore project");
12
13     // Initialize an object to track project-specific settings
14     // time the generator runs
15     this.settings = {};
16
17     // Escape single backslash with double-backslash
18     this.settings.solutionRoot = this.destinationPath(
19     );
20
21   async prompting() {
22     // Prompt user for project-specific settings
23     await prompter.getSolutionInfo.call(this);
24     await prompter.getFrameworkInfo.call(this);
25   }
26
27   configuring() {
28     // Save all the settings that have been provided to
29     Object.keys(this.settings).forEach((key) => this.config.set(
30     ));
31
32   writing() {
33     writer.writeConfigFiles.call(this);
34
35     // Copy Helix layers to destination, dynamically to
36     // project-specific settings as needed
37     writer.writeTemplates.call(this);
38   }
39
40   install() {
41     // Install node modules in destination
42     this.yarnInstall();
43   }
44
45   end() {
46     this.log("Finished generating Sitecore project");
47   }
48 }
```

/generators/lib/prompter.js

```
JS prompter.js x
1  /* ...
12
13  function getSolutionInfo() { ...
14  }
15
16  function getFrameworkInfo() {
17    const choices = [];
18    switch (this.settings.sitecoreVersion) {
19      case "8.2.171121":
20      case "8.2.180406":
21        choices.push({ name: ".NET 4.6.1", value: "4.6.1" });
22        choices.push({ name: ".NET 4.6", value: "4.6" });
23        choices.push({ name: ".NET 4.5.2", value: "4.5.2" });
24        break;
25      case "9.0.171219":
26        choices.push({ name: ".NET 4.7", value: "4.7" });
27        choices.push({ name: ".NET 4.6.2", value: "4.6.2" });
28        break;
29    };
30    return this.prompt([
31      {
32        type: "list",
33        name: "framework",
34        message: "Your .NET framework",
35        choices: choices,
36        required: true,
37        store: true
38      }
39    ]).then((answers) => {
40      this.settings.framework = "v" + answers.framework;
41      this.settings.frameworkShorthand = "net" + this.settings.framework.replace(/\.\/g, "");
42    });
43  }
44
45  /* ...
46
47  function getTypeScriptInfo() { ...
48  }
49
50  module.exports = {
51    getSolutionInfo: getSolutionInfo,
52    getFrameworkInfo: getFrameworkInfo,
53    getTypeScriptInfo: getTypeScriptInfo
54  }
55
56  111
57  112
```

/generators/app/index.js

```
EXPLORER: GENERATOR-FLYNN
├─ generators
│  ├─ add-bootstrap
│  ├─ add-typescript
│  └─ app
│     └─ templates
├─ JS index.js
├─ lib
├─ node_modules
├─ .gitignore
├─ {} package-lock.json
├─ {} package.json
└─ yarn.lock

JS index.js
1  const Generator = require("yeoman-generator"),
2    prompter = require("../lib/prompter"),
3    writer = require("../lib/writer");
4
5  module.exports = class extends Generator {
6    constructor(args, opts) {
7      super(args, opts);
8    }
9
10   initializing() {
11     this.log("Generating Sitecore project");
12
13     // Initialize an object to track project-specific settings that change each
14     // time the generator runs
15     this.settings = {};
16
17     // Escape single backslash with double-backslash for usage in strings
18     this.settings.solutionRoot = this.destinationPath().replace(/\\/g, "\\");
19   }
20
21   async prompting() {
22     // Prompt user for project-specific settings
23     await prompter.getSolutionInfo.call(this);
24     await prompter.getFrameworkInfo.call(this);
25   }
26
27   configuring() {
28     // Save all the settings that have been provided to config file
29     Object.keys(this.settings).forEach((key) => this.config.set(key, this.settings[key]));
30   }
31
32   writing() {
33     writer.writeConfigFiles.call(this);
34
35     // Copy Helix layers to destination, dynamically transforming files based on
36     // project-specific settings as needed
37     writer.writeTemplates.call(this);
38   }
39
40   install() {
41     // Install node modules in destination
42     this.yarnInstall();
43   }
44
45   end() {
46     this.log("Finished generating Sitecore project");
47   }
48 };
```

Initializing

- Log Message
- Initialize settings object



Prompting

- Prompt user for project-specific settings



Configuring

- Save all the settings that have been provided to config file



Writing

- Write settings to config file on disk
- Copy templates to destination, transforming content using project-specific settings



Install

- Install node modules in destination



End

- Log Message

/generators/app/index.js

```
EXPLORER: GENERATOR-FLYNN
generators
├─ add-bootstrap
├─ add-typescript
├─ app
│  └─ templates
└─ JS index.js
lib
node_modules
.gitignore
package-lock.json
package.json
yarn.lock

JS index.js x
1  const Generator = require("yeoman-generator"),
2    prompter = require("../lib/prompter"),
3    writer = require("../lib/writer");
4
5  module.exports = class extends Generator {
6    constructor(args, opts) {
7      super(args, opts);
8    }
9
10   initializing() {
11     this.log("Generating Sitecore project");
12
13     // Initialize an object to track project-specific settings that change each
14     // time the generator runs
15     this.settings = {};
16
17     // Escape single backslash with double-backslash for usage in strings
18     this.settings.solutionRoot = this.destinationPath().replace(/\\/g, "\\");
19   }
20
21   async prompting() {
22     // Prompt user for project-specific
23     await prompter.getSolutionInfo.call(this);
24     await prompter.getFrameworkInfo.call(this);
25   }
26
27   configuring() {
28     // Save all the settings that have to be saved
29     Object.keys(this.settings).forEach((key) => {
30       this.settings[key] = this.prompter.get(key);
31     });
32
33   writing() {
34     writer.writeConfigFiles.call(this);
35
36     // Copy Helix layers to destination,
37     // project-specific settings as needed
38     writer.writeTemplates.call(this);
39   }
40
41   install() {
42     // Install node modules in destination
43     this.yarnInstall();
44   }
45
46   end() {
47     this.log("Finished generating Sitecore project");
48   }
49 };

JS writer.js x
1  const guid = require("uuid");
2  //gulp-rename@1.2.2 is required, See https://github.com/yeoman/yo/issues/577
3  const rename = require("gulp-rename");
4
5  function writeTemplates(src, dist) { ...
6  }
7
8  function writeConfigFiles() {
9    // Any file that only has an extension needs to be copied explicitly
10    this.fs.copyTpl(this.templatePath(".gitignore"), this.destinationPath(".gitignore"), this.settings);
11    this.fs.copyTpl(this.templatePath(".gitattributes"), this.destinationPath(".gitattributes"));
12    this.fs.copyTpl(this.templatePath(".babelrc"), this.destinationPath(".babelrc"));
13    this.fs.copyTpl(this.templatePath("./src/.editorconfig"), this.destinationPath("./src/.editorconfig"));
14  }
15
16  function generateGuids() { ...
17  }
18
19  module.exports = {
20    writeTemplates: writeTemplates,
21    writeConfigFiles: writeConfigFiles,
22    generateGuids: generateGuids
23  };
24
```

/generators/lib/writer.js

```
JS writer.js x
1  const guid = require("uuid");
2  //gulp-rename@1.2.2 is required, See https://github.com/yeoman/yo/issues/577
3  const rename = require("gulp-rename");
4
5  function writeTemplates(src, dist) { ...
6  }
7
8  function writeConfigFiles() {
9    // Any file that only has an extension needs to be copied explicitly
10    this.fs.copyTpl(this.templatePath(".gitignore"), this.destinationPath(".gitignore"), this.settings);
11    this.fs.copyTpl(this.templatePath(".gitattributes"), this.destinationPath(".gitattributes"));
12    this.fs.copyTpl(this.templatePath(".babelrc"), this.destinationPath(".babelrc"));
13    this.fs.copyTpl(this.templatePath("./src/.editorconfig"), this.destinationPath("./src/.editorconfig"));
14  }
15
16  function generateGuids() { ...
17  }
18
19  module.exports = {
20    writeTemplates: writeTemplates,
21    writeConfigFiles: writeConfigFiles,
22    generateGuids: generateGuids
23  };
24
```


/generators/app/index.js

/generators/lib/writer.js

EXPLORER: GENERATOR-FLYNN

- generators
 - add-bootstrap
 - add-typescript
 - app
 - templates
 - JS index.js
 - lib
 - node_modules
 - .gitignore
 - package-lock.json
 - package.json
 - yarn.lock

```
1 const Generator = require("yeoman-generator"),
2   prompter = require("../lib/prompter"),
3   writer = require("../lib/writer");
4
5 module.exports = class extends Generator {
6   constructor(args, opts) {
7     super(args, opts);
8   }
9
10  initializing() {
11    this.log("Generating Sitecore project");
12
13    // Initialize an object to track project-specific
14    // time the generator runs
15    this.settings = {};
16
17    // Escape single backslash with double-backslash
18    this.settings.solutionRoot = this.destinationPath
19  }
20
21  async prompting() {
22    // Prompt user for project-specific settings
23    await prompter.getSolutionInfo.call(this);
24    await prompter.getFrameworkInfo.call(this);
25  }
26
27  configuring() {
28    // Save all the settings that have been provided
29    Object.keys(this.settings).forEach((key) => this.
30  }
31
32  writing() {
33    writer.writeConfigFiles.call(this);
34
35    // Copy Helix layers to destination, dynamically
36    // project-specific settings as needed
37    writer.writeTemplates.call(this);
38  }
39
40  install() {
41    // Install node modules in destination
42    this.yarnInstall();
43  }
44
45  end() {
46    this.log("Finished generating Sitecore project");
47  }
48 };
```

```
1 const guid = require("uuid");
2 //gulp-rename@1.2.2 is required, See https://github.com/yeoman/yo/issues/577
3 const rename = require("gulp-rename");
4
5 function writeTemplates(src, dist) {
6   this.registerTransformStream(rename((path) => {
7     path.basename = path.basename.replace(/ArkeAreaName/g, this.settings.areaName);
8     path.basename = path.basename.replace(/ArkeBase/g, this.settings.solutionNamespace);
9     path.basename = path.basename.replace(/arkebase/g, this.settings.solutionNamespace.toLowerCase());
10
11     path.dirname = path.dirname.replace(/ArkeAreaName/g, this.settings.areaName);
12     path.dirname = path.dirname.replace(/ArkeBase/g, this.settings.solutionNamespace);
13     path.dirname = path.dirname.replace(/arkebase/g, this.settings.solutionNamespace.toLowerCase());
14   }));
15
16   const templateData = Object.assign(generateGuids(), this.settings);
17
18   if (src && dist) {
19     this.fs.copyTpl(this.templatePath(src), this.destinationPath(dist), templateData);
20   } else {
21     this.fs.copyTpl(this.templatePath(), this.destinationPath(), templateData);
22   }
23 }
24
25 function writeConfigFiles() {
26
27   function generateGuids() {
28     return {
29       projectGuid: guid.v4(),
30       projectTDSCoreGuid: guid.v4(),
31       projectTDSMasterGuid: guid.v4(),
32       projectUnitTestGuid: guid.v4(),
33       featureGuid: guid.v4(),
34       featureTDSCoreGuid: guid.v4(),
35       featureTDSMasterGuid: guid.v4(),
36       featureUnitTestGuid: guid.v4(),
37       foundationGuid: guid.v4(),
38       foundationTDSCoreGuid: guid.v4(),
39       foundationTDSMasterGuid: guid.v4(),
40       cmBundleGuid: guid.v4(),
41       cdBundleGuid: guid.v4()
42     };
43   }
44
45   module.exports = {
46     writeTemplates: writeTemplates,
47     writeConfigFiles: writeConfigFiles
48   };
49 }
```

/generators/app/index.js

```
EXPLORER: GENERATOR-FLYNN
├─ generators
│  ├─ add-bootstrap
│  ├─ add-typescript
│  └─ app
│     └─ templates
├─ JS index.js
├─ lib
├─ node_modules
├─ .gitignore
├─ {} package-lock.json
├─ {} package.json
└─ yarn.lock

JS index.js
1  const Generator = require("yeoman-generator"),
2    prompter = require("../lib/prompter"),
3    writer = require("../lib/writer");
4
5  module.exports = class extends Generator {
6    constructor(args, opts) {
7      super(args, opts);
8    }
9
10   initializing() {
11     this.log("Generating Sitecore project");
12
13     // Initialize an object to track project-specific settings that change each
14     // time the generator runs
15     this.settings = {};
16
17     // Escape single backslash with double-backslash for usage in strings
18     this.settings.solutionRoot = this.destinationPath().replace(/\\/g, "\\");
19   }
20
21   async prompting() {
22     // Prompt user for project-specific settings
23     await prompter.getSolutionInfo.call(this);
24     await prompter.getFrameworkInfo.call(this);
25   }
26
27   configuring() {
28     // Save all the settings that have been provided to config file
29     Object.keys(this.settings).forEach((key) => this.config.set(key, this.settings[key]));
30   }
31
32   writing() {
33     writer.writeConfigFiles.call(this);
34
35     // Copy Helix layers to destination, dynamically transforming files based on
36     // project-specific settings as needed
37     writer.writeTemplates.call(this);
38   }
39
40   install() {
41     // Install node modules in destination
42     this.yarnInstall();
43   }
44
45   end() {
46     this.log("Finished generating Sitecore project");
47   }
48 }
```

Initializing

- Log Message
- Initialize settings object



Prompting

- Prompt user for project-specific settings



Configuring

- Save all the settings that have been provided to config file



Writing

- Write settings to config file on disk
- Copy templates to destination, transforming content using project-specific settings



Install

- Install node modules in destination



End

- Log Message

Yeoman – Exposing Generator For Testing



Yeoman – Running Main Generator



/generators/add-bootstrap/index.js

```
1  const Generator = require("yeoman-generator"),
2  ...writer = require("../lib/writer");
3
4  module.exports = class extends Generator {
5
6  ...constructor(args, opts) {
7  ...super(args, opts);
8  ...}
9
10 → initializing() {
11  ...this.log("Generating Sitecore project");
12
13  ...// Read previously defined settings
14  ...this.settings = this.config.getAll();
15  ...}
16
17 → writing() {
18  ...const pkgJson = {
19  ...dependencies: {
20  ..."bootstrap": "4.1.2",
21  ..."jquery": "^3.3.1",
22  ..."popper.js": "^1.14.4"
23  ...}
24  ...};
25
26  ...// Extend or create package.json file in destination path
27  ...this.fs.extendJSON(this.destinationPath("package.json"), pkgJson);
28
29  ...// Copy template files, dynamically transforming files based on project-specific settings as needed
30  ...writer.writeTemplates.call(this);
31  ...writer.writeTemplates.call(
32  ...this,
33  ..."../Assets",
34  ...`${this.src}/Project/${this.settings.solutionNamespace}.Web/${this.settings.frontEndAssetPath}`);
35  ...}
36
37 → install() {
38  ...this.yarnInstall();
39  ...}
40 }
```

Initializing

- Log Message
- Read previously defined settings



Writing

- Update Node module dependency declarations
- Copy templates to destination, transforming content using project-specific settings



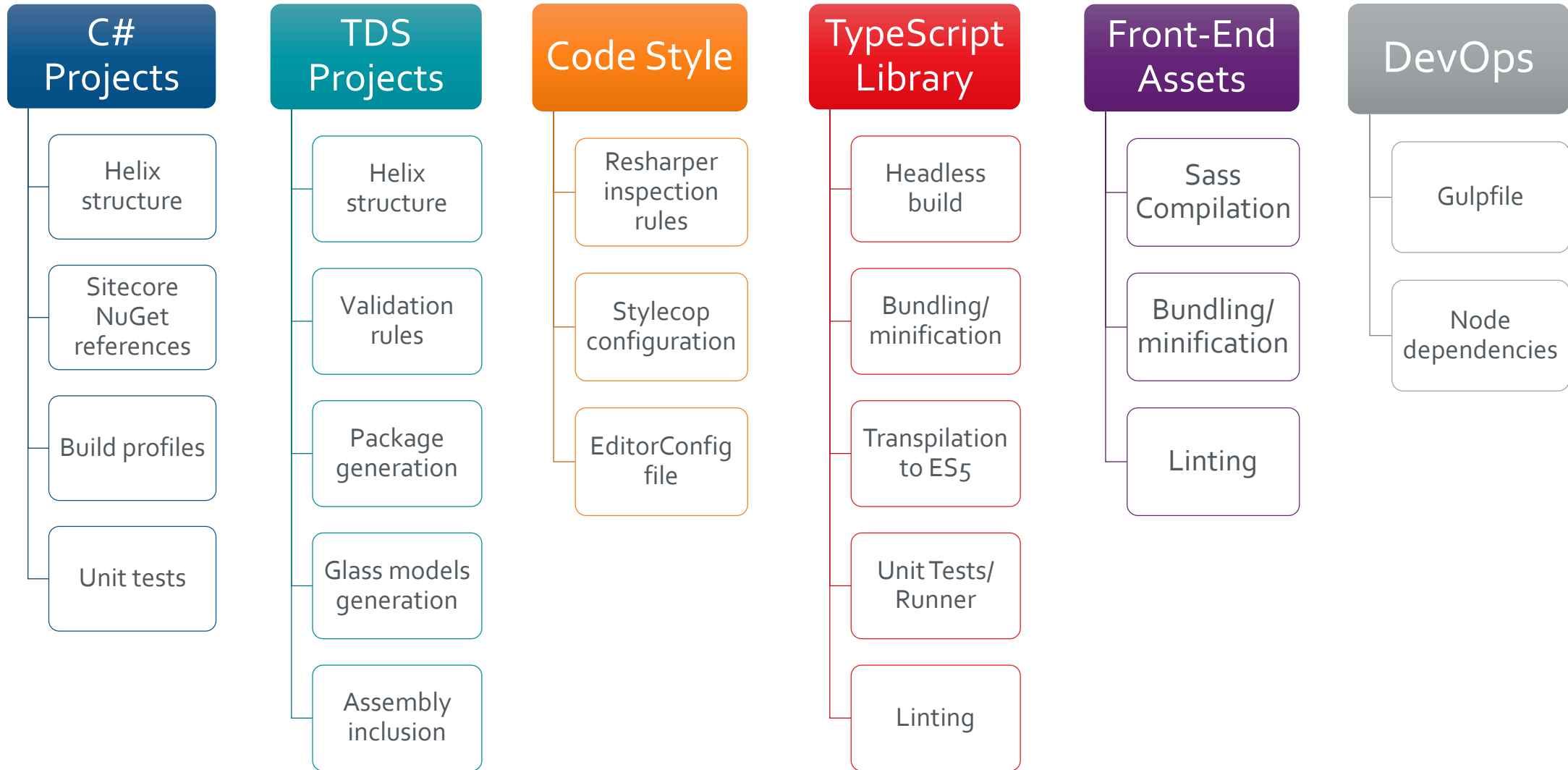
Install

- Install new Node module dependencies

Yeoman – Running Sub-Generator



New Solution Checklist – Conquered!



Development

Use **Webpack** to

- Helix-ify the JavaScript layer
- Build against a headless CMS
- Perform on-the-fly compilation



Setup

Development

Testing

Project (C#)



Feature (C#)

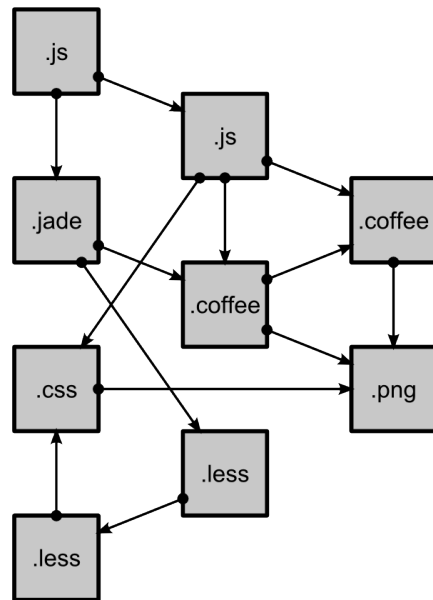
JavaScript/TypeScript Library



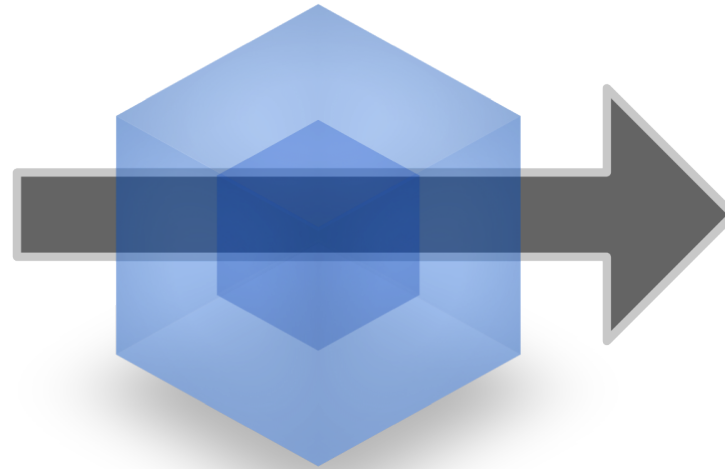
Foundation (C#)

What is Webpack?

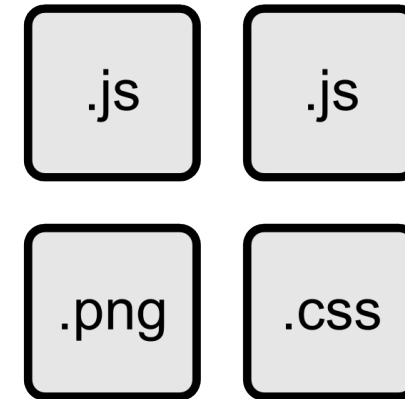
<https://webpack.js.org/>



modules
with dependencies



webpack
MODULE BUNDLER



static
assets

Webpack - Configuration

> yarn add webpack --dev

> yarn add webpack-cli --dev

/webpack.config.json

```
webpack.config.js •
1  const path = require('path');
2
3  module.exports = {
4    entry: './path/to/my/entry/file.js',
5    output: {
6      path: path.resolve(__dirname, 'dist'),
7      filename: 'my-first-webpack.bundle.js'
8    },
9    module: {
10     rules: [
11       { test: /\.txt$/, use: 'raw-loader' }
12     ]
13   }
14 };
```

/package.json

```
package.json •
1  {
2    "name": "flynn-typescript-library",
3    "version": "1.0.0",
4    "description": "",
5    "scripts": {
6      "build": "yarn install && webpack"
7    },
8    "devDependencies": { ...
30   }
31 }
32
```

Project

Feature

Group - Common

Gallery

Carousel

Group - Product

Add To
Cart

Customer
Reviews

Related
Products

Foundation

Component
Factory

Event
Manager

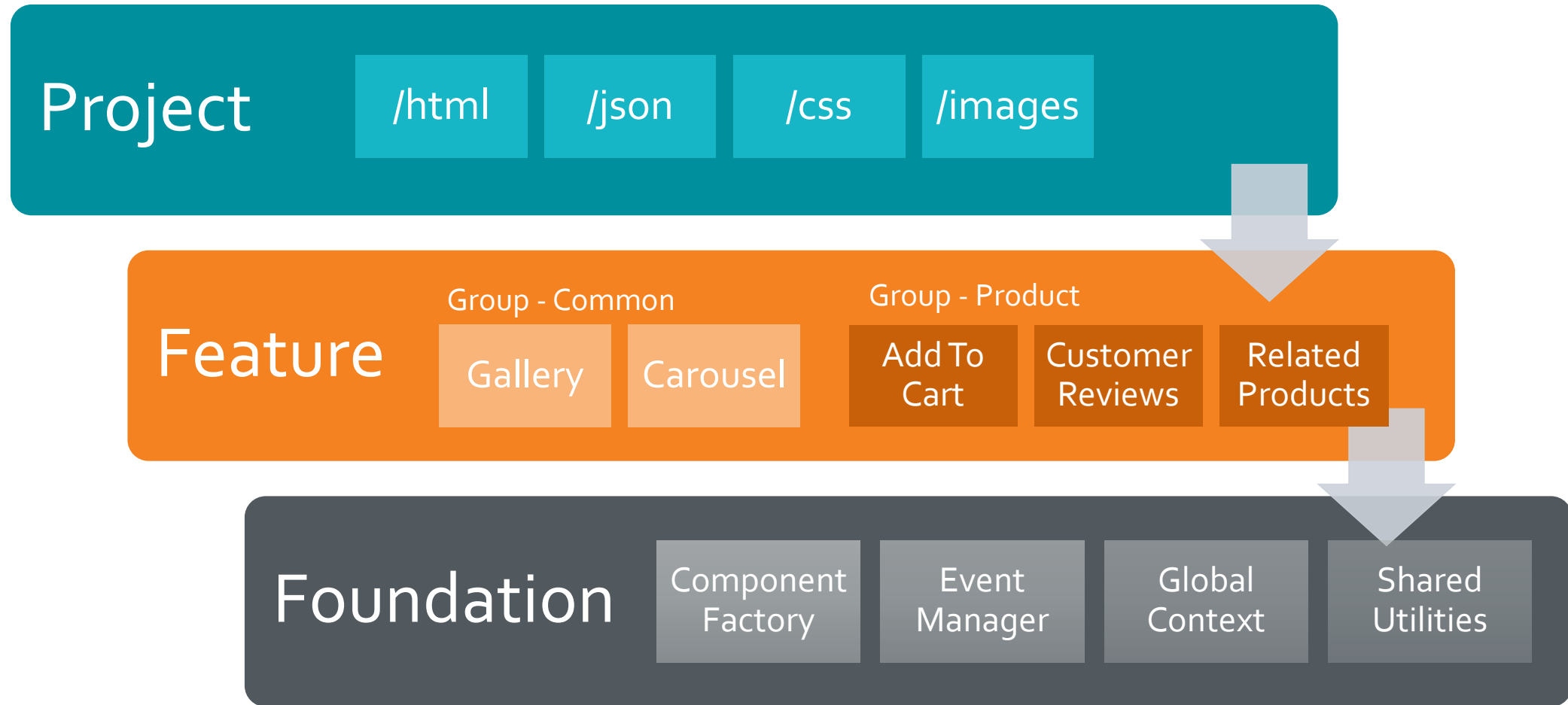
Global
Context

Shared
Utilities

Webpack – Demo 'npm run build'



TypeScript Helix



Webpack Dev Server - Configuration

<https://webpack.js.org/configuration/dev-server/>

> yarn add webpack-dev-server --dev

/webpack.config.json

```
webpack.config.js
1  var path = require('path');
2
3  module.exports = {
4    //...
5    devServer: {
6      contentBase: path.join(__dirname, 'dist'),
7      compress: true,
8      port: 9000
9    }
10  };
```

/package.json

```
package.json
1  {
2    "name": "flynn-typescript-library",
3    "version": "1.0.0",
4    "description": "",
5    "scripts": {
6      "setup": "node setup.js",
7      "build": "yarn install && webpack",
8      "dev": "npm run setup && webpack-dev-server"
9    },
10   "devDependencies": { ...
32   }
33 }
34
```

Webpack – Demo 'npm run dev'



Dev Server – Ajax?



TypeScript Helix – Best practice patterns

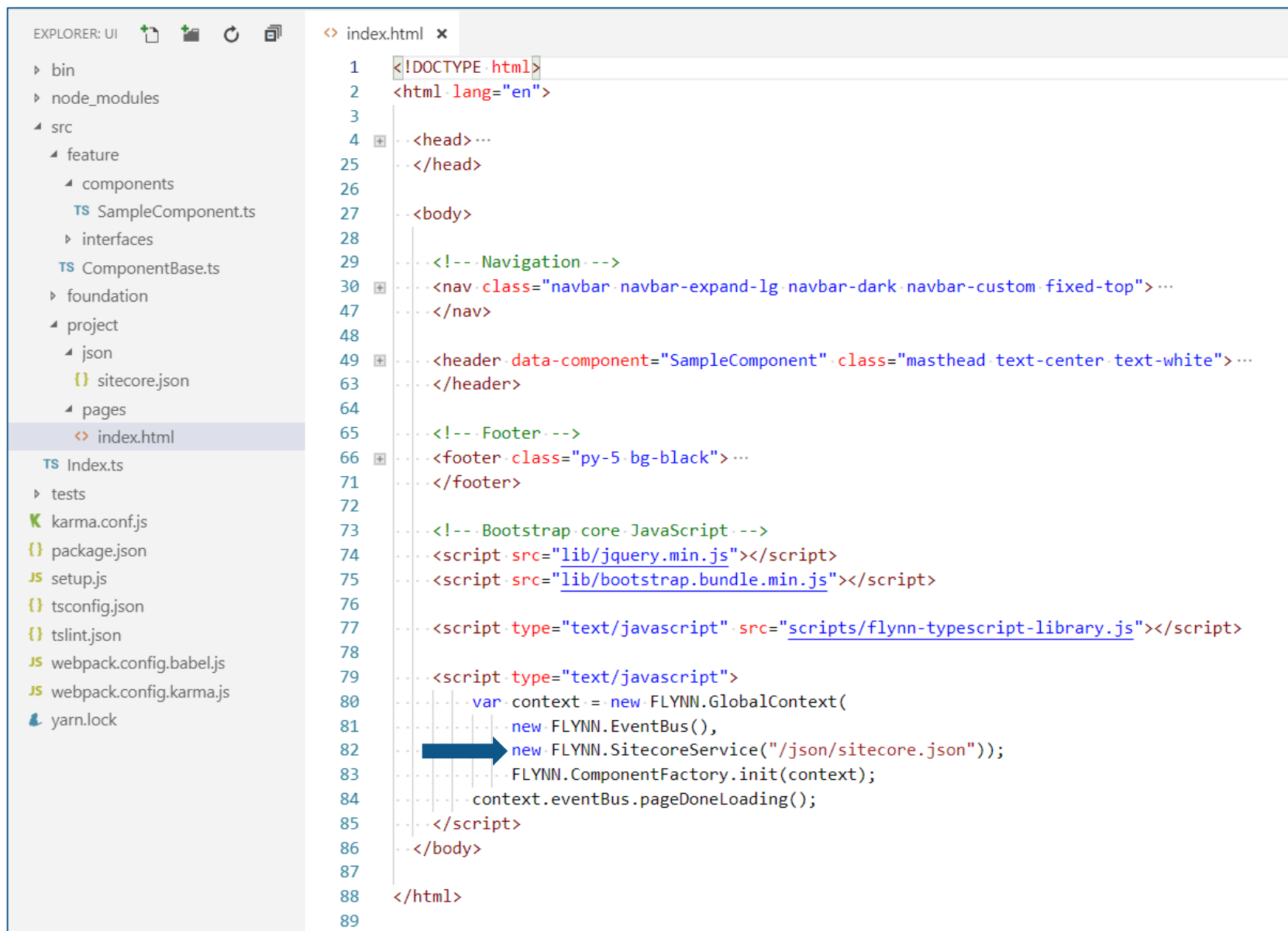
Server interaction

- Encapsulate all Ajax functionality in an injectable helper
- Never hardcode server endpoints

UI component classes

- Accept server interaction helpers as parameters in constructors
- Don't talk to the server directly

/src/project/pages/index.html



```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head> ...
25 </head>
26
27 <body>
28
29 <!-- Navigation -->
30 <nav class="navbar navbar-expand-lg navbar-dark navbar-custom fixed-top"> ...
47 </nav>
48
49 <header data-component="SampleComponent" class="masthead text-center text-white"> ...
63 </header>
64
65 <!-- Footer -->
66 <footer class="py-5 bg-black"> ...
71 </footer>
72
73 <!-- Bootstrap core JavaScript -->
74 <script src="lib/jquery.min.js"></script>
75 <script src="lib/bootstrap.bundle.min.js"></script>
76
77 <script type="text/javascript" src="scripts/flynn-typescript-library.js"></script>
78
79 <script type="text/javascript">
80     var context = new FLYNN.GlobalContext(
81     new FLYNN.EventBus(),
82     new FLYNN.SitecoreService("/json/sitecore.json"));
83     FLYNN.ComponentFactory.init(context);
84     context.eventBus.pageDoneLoading();
85 </script>
86 </body>
87
88 </html>
89
```

EXPLORER: UI

bin

node_modules

src

feature

components

TS SampleComponent.ts

interfaces

TS ComponentBase.ts

foundation

project

json

sitecore.json

pages

index.html

TS Index.ts

tests

karma.conf.js

package.json

setup.js

tsconfig.json

tslint.json

webpack.config.babel.js

webpack.config.karma.js

yarn.lock

index.html

1

2

3

4

25

26

27

28

29

30

47

48

49

63

64

65

66

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

<!DOCTYPE html>

<html lang="en">

<head>...

</head>

<body>

<!-- Navigation -->

<nav class="navbar navbar-expand-lg navbar">

</nav>

<header data-component="SampleComponent" c<

</header>

<!-- Footer -->

<footer class="py-5 bg-black">...

</footer>

<!-- Bootstrap core JavaScript -->

<script src="lib/jquery.min.js"></script>

<script src="lib/bootstrap.bundle.min.js">

<script type="text/javascript" src="script

<script type="text/javascript">

var context = new FLYNN.GlobalContext(

new FLYNN.EventBus(),

new FLYNN.SitecoreService("/json/sitecore.json");

FLYNN.ComponentFactory.init(context);

context.eventBus.pageDoneLoading();

</script>

</body>

</html>

using System.Web.Mvc;

namespace Flynn.Sitecore.Web.App_Start

{

public class RouteConfig

{

public const string ProxyRoute = "ApiRoute";

public override void RegisterArea(RouteCollection routes)

{

routes.MapRoute(

ProxyRoute,

"api/{controller}/{action}/{id}",

new { id = UrlParameter.Optional });

}

}

<script type="text/javascript" src="/Assets/dist/lib/jquery.min.js"></script>

<script type="text/javascript" src="/Assets/dist/lib/bootstrap.bundle.min.js"></script>

<script type="text/javascript" src="/Assets/dist/scripts/flynn-typescript-library.js"></script>

<script type="text/javascript">

var context = new FLYNN.GlobalContext(

new FLYNN.EventBus(),

new FLYNN.SitecoreService("@Url.RouteUrl(RouteConfig.ProxyRoute)");

FLYNN.ComponentFactory.init(context);

context.eventBus.pageDoneLoading();

</script>

Headless CMS – Sample Component

```
1 import { ComponentBase } from "../../ComponentBase";
2 import { ComponentFactory } from "../../foundation/services/ComponentFactory";
3 import { IContext } from "../../foundation/interfaces/IContext";
4 import { ICard } from "../../interfaces/ICard";
5
6 export class SampleComponent extends ComponentBase {
7     ...private readonly cta: HTMLElement;
8     ...private readonly cardDeck: HTMLElement;
9
10    ...// The value of element is the HTML element with the data-component property
11    ...constructor(public element: HTMLElement, public context: IContext) {
12        ...super(context);
13
14        ...this.cardDeck = this.element.querySelector(".card-deck");
15        ...this.cta = this.element.querySelector(".btn-primary");
16        ...this.cta.addEventListener("click", (ev: Event) => {
17            ...ev.preventDefault();
18            ...this.callToAction();
19        ...});
20    ...}
21
22    ...public callToAction() {
23        ...this.context.sitecoreService.getSitecoreData("Card", "GetCards")
24        ...    .then((cards: ICard[]) => this.showCards(cards));
25    ...}
26
27    ...public showCards(cards: ICard[]) {
28        ...cards.forEach((card) => this.appendCard(card));
29    ...}
30
31    ...public appendCard(card: ICard) {
32        ...const html = `<div class="card text-white ${card.priority} mb-3" style="max-width: 18rem;">
33        ...    <div class="card-body">
34        ...        <h5 class="card-title">${card.title}</h5>
35        ...        <p class="card-text">${card.description}</p>
36        ...    </div></div>`;
37        ...this.cardDeck.insertAdjacentHTML("beforeend", html);
38    ...}
39 }
40
41 // Registers component with ComponentFactory so that it can be initialized
42 ComponentFactory.registerComponent("SampleComponent", SampleComponent);
43
```

Testing

Unit test JavaScript with
Jasmine and Karma



Setup

Development

Testing

Why Unit Test JavaScript?

Unit tests are valuable because they:

- Document the code
 - Shows how code is expected to work
 - Shows how code is expected to be used
- Enforce good class design
 - Classes have to be broken up into granular functions (units)
 - Classes have to use dependency injection
- Enforce good Sitecore component design
 - UI component classes must be autonomous and self-contained

Jasmine - Setup

<https://jasmine.github.io>

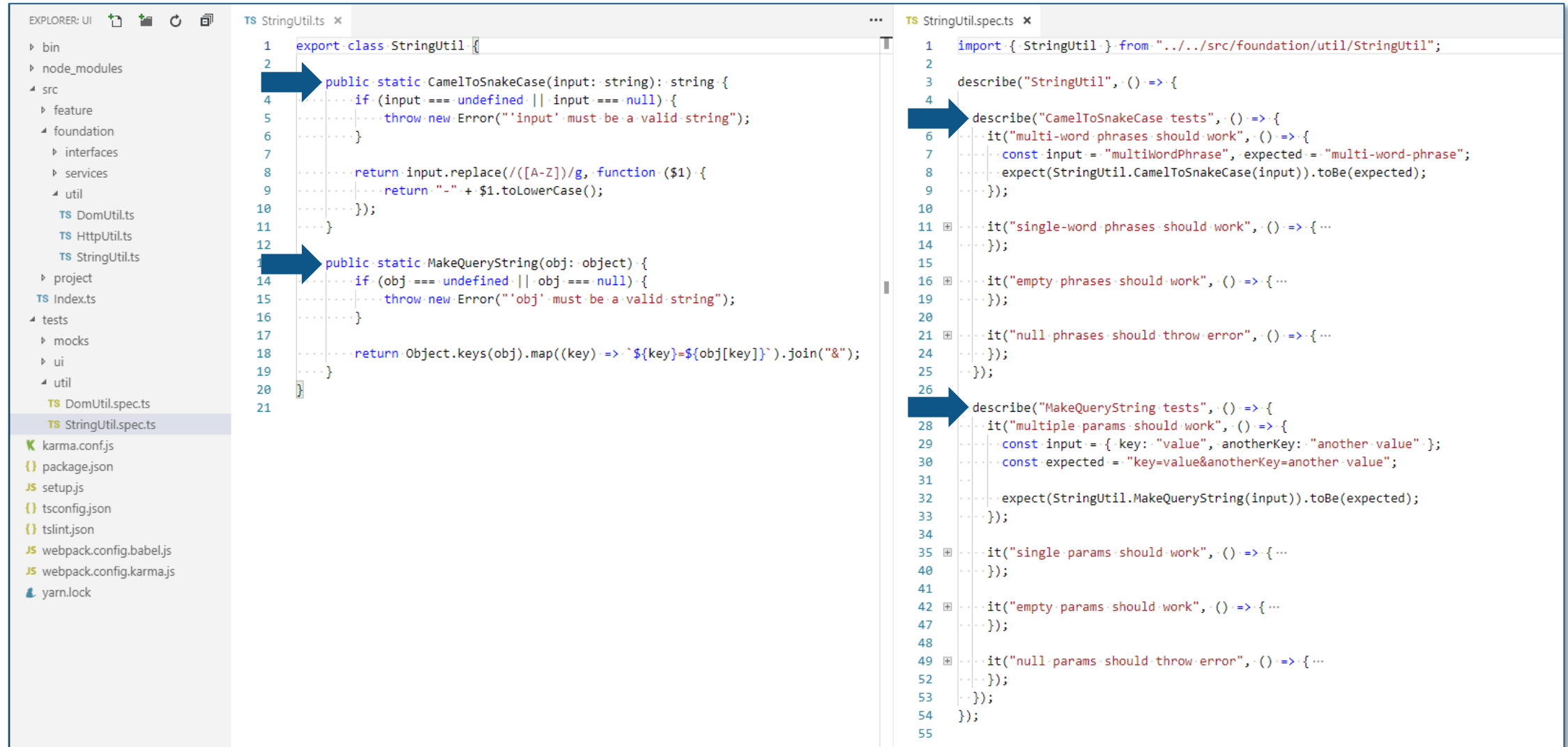
> yarn add jasmine --dev

> yarn add jasmine-core --dev

> yarn add @types/jasmine --dev

```
TS example.spec.ts •
1 describe("A suite is just a function", function() {
2   ... var a;
3
4   ... it("and so is a spec", function() {
5     ... a = true;
6
7     ... expect(a).toBe(true);
8   ... });
9 });
```


Jasmine - Implementation



The image shows a VS Code editor with two files open: `StringUtil.ts` and `StringUtil.spec.ts`. The left sidebar shows the Explorer view with the project structure. The `StringUtil.ts` file contains two static methods: `CamelToSnakeCase` and `MakeQueryString`. The `StringUtil.spec.ts` file contains Jasmine tests for these methods. Blue arrows point to the start of each method in the implementation and the start of each test suite in the spec file.

```
EXPLORER: UI
  bin
  node_modules
  src
    feature
    foundation
      interfaces
      services
    util
      TS DomUtil.ts
      TS HttpUtil.ts
      TS StringUtil.ts
    project
  TS Index.ts
  tests
    mocks
    ui
    util
      TS DomUtil.spec.ts
      TS StringUtil.spec.ts
  karma.conf.js
  package.json
  setup.js
  tsconfig.json
  tslint.json
  webpack.config.babel.js
  webpack.config.karma.js
  yarn.lock

TS StringUtil.ts
1 export class StringUtil {
2   public static CamelToSnakeCase(input: string): string {
3     if (input === undefined || input === null) {
4       throw new Error("'input' must be a valid string");
5     }
6     return input.replace(/[A-Z]/g, function ($1) {
7       return "-" + $1.toLowerCase();
8     });
9   }
10  public static MakeQueryString(obj: object): string {
11    if (obj === undefined || obj === null) {
12      throw new Error("'obj' must be a valid string");
13    }
14    return Object.keys(obj).map((key) => `${key}=${obj[key]}`).join("&");
15  }
16 }

TS StringUtil.spec.ts
1 import { StringUtil } from "../../src/foundation/util/StringUtil";
2
3 describe("StringUtil", () => {
4   describe("CamelToSnakeCase tests", () => {
5     it("multi-word phrases should work", () => {
6       const input = "multiWordPhrase";
7       const expected = "multi-word-phrase";
8       expect(StringUtil.CamelToSnakeCase(input)).toBe(expected);
9     });
10    it("single-word phrases should work", () => {
11      const input = "singleWord";
12      const expected = "single-word";
13      expect(StringUtil.CamelToSnakeCase(input)).toBe(expected);
14    });
15    it("empty phrases should work", () => {
16      const input = "";
17      const expected = "";
18      expect(StringUtil.CamelToSnakeCase(input)).toBe(expected);
19    });
20    it("null phrases should throw error", () => {
21      expect(() => StringUtil.CamelToSnakeCase(null)).toThrowError();
22    });
23  });
24  describe("MakeQueryString tests", () => {
25    it("multiple params should work", () => {
26      const input = { key: "value", anotherKey: "another value" };
27      const expected = "key=value&anotherKey=another value";
28      expect(StringUtil.MakeQueryString(input)).toBe(expected);
29    });
30    it("single params should work", () => {
31      const input = { key: "value" };
32      const expected = "key=value";
33      expect(StringUtil.MakeQueryString(input)).toBe(expected);
34    });
35    it("empty params should work", () => {
36      const input = {};
37      const expected = "";
38      expect(StringUtil.MakeQueryString(input)).toBe(expected);
39    });
40    it("null params should throw error", () => {
41      expect(() => StringUtil.MakeQueryString(null)).toThrowError();
42    });
43  });
44 }
```

TypeScript Helix – Best practice patterns (continued)

Server interaction

- Encapsulate all Ajax functionality in an injectable helper
- Never hardcode server endpoints

UI component classes

- Accept server interaction helpers as parameters in constructors
 - Don't talk to the server directly
-
- Accept interface injection, not concrete classes

Unit Testing UI Components – Sample Component

```
1 import { ComponentBase } from "../../ComponentBase";
2 import { ComponentFactory } from "../../foundation/services/ComponentFactory";
3 import { IContext } from "../../foundation/interfaces/IContext";
4 import { ICard } from "../interfaces/ICard";
5
6 export class SampleComponent extends ComponentBase {
7     ...private readonly cta: HTMLElement;
8     ...private readonly cardDeck: HTMLElement;
9
10    ...// The value of element is the HTML element with the data-component property
11    ...constructor(public element: HTMLElement, public context: IContext) {
12        ...super(context);
13
14        ...this.cardDeck = this.element.querySelector(".card-deck");
15        ...this.cta = this.element.querySelector(".btn-primary");
16        ...this.cta.addEventListener("click", (ev: Event) => {
17            ...ev.preventDefault();
18            ...this.callToAction();
19        ...});
20    ...}
21
22    ...public callToAction() {
23        ...this.context.sitecoreService.getSitecoreData("Card", "GetCards")
24        ...then((cards: ICard[]) => this.showCards(cards));
25    ...}
26
27    ...public showCards(cards: ICard[]) {
28        ...cards.forEach((card) => this.appendCard(card));
29    ...}
30
31    ...public appendCard(card: ICard) {
32        ...const html = `<div class="card text-white ${card.priority} mb-3" style="max-width: 18rem;">
33        ...<div class="card-body">
34        ...<h5 class="card-title">${card.title}</h5>
35        ...<p class="card-text">${card.description}</p>
36        ...</div></div>`;
37        ...this.cardDeck.insertAdjacentHTML("beforeend", html);
38    ...}
39 }
40
41 // Registers component with ComponentFactory so that it can be initialized
42 ComponentFactory.registerComponent("SampleComponent", SampleComponent);
43
```

TypeScript Helix – Best practice patterns (continued)

Server interaction

- Encapsulate all Ajax functionality in an injectable helper
- Never hardcode server endpoints

UI component classes

- Accept server interaction helpers as parameters in constructors
 - Don't talk to the server directly
 - Accept interface injection, not concrete classes
-
- Accept a DOM element in the constructor
 - Scope all actions to it's DOM element

Unit Testing UI Components – Sample Component

```
1 import { ComponentBase } from "../ComponentBase";
2 import { ComponentFactory } from "../../foundation/services/ComponentFactory";
3 import { IContext } from "../../foundation/interfaces/IContext";
4 import { ICard } from "../interfaces/ICard";
5
6 export class SampleComponent extends ComponentBase {
7     ...private readonly cta: HTMLElement;
8     ...private readonly cardDeck: HTMLElement;
9
10    ...// The value of element is the HTML element with the data-component property
11    ...constructor public element: HTMLElement public context: IContext) {
12        ...super(context);
13
14        ...this.cardDeck = this.element.querySelector(".card-deck");
15        ...this.cta = this.element.querySelector(".btn-primary");
16        ...this.cta.addEventListener("click", (ev: Event) => {
17            ...ev.preventDefault();
18            ...this.callToAction();
19        ...});
20    ...}
21
22    ...public callToAction() {
23        ...this.context.sitecoreService.getSitecoreData("Card", "GetCards")
24        ...then((cards: ICard[]) => this.showCards(cards));
25    ...}
26
27    ...public showCards(cards: ICard[]) {
28        ...cards.forEach((card) => this.appendCard(card));
29    ...}
30
31    ...public appendCard(card: ICard) {
32        ...const html = `<div class="card text-white ${card.priority} mb-3" style="max-width: 18rem;">
33        ...<div class="card-body">
34        ...<h5 class="card-title">${card.title}</h5>
35        ...<p class="card-text">${card.description}</p>
36        ...</div></div>`;
37        ...this.cardDeck.insertAdjacentHTML("beforeend", html);
38    ...}
39 }
40
41 // Registers component with ComponentFactory so that it can be initialized
42 ComponentFactory.registerComponent("SampleComponent", SampleComponent);
43
```

EXPLORER: UI

bin

node_modules

src

tests

mocks

TS PageLayoutMock.ts

ui

common

TS SampleComponent.spec.ts

util

karma.conf.js

package.json

JS setup.js

tsconfig.json

tslint.json

JS webpack.config.babel.js

JS webpack.config.karma.js

yarn.lock

TS SampleComponent.spec.ts

```
1 import { SampleComponent } from "../../src/feature/components/SampleComponent";
2 import { PageLayoutMock } from "../../mocks/PageLayoutMock";
3 import { ComponentFactory, IComponentDeclaration } from "../../src/foundation/services/ComponentFactory";
4
5 describe("SampleComponent", () => {
6     ...function makePage(): PageLayoutMock {
7         ...return new PageLayoutMock("/scripts/json/sitecore.json");
8     }
9
10    ...describe("constructor tests", () => {
11        ...it("can create new SampleComponent instance", () => {
12            ...// Arrange
13            ...const page = makePage();
14            ...const html = `
15            <div>
16            ...<div id='myDiv' data-component='SampleComponent'>
17            ...<a class='btn-primary'></a>
18            ...<div class='card-deck'></div>
19            ...</div>
20            </div>`;
21            ...document.body.innerHTML = html;
22            ...const div = document.getElementById("myDiv");
23
24            ...// Act
25            ...page.afterPageLoad();
26            ...const result = ComponentFactory.loadedComponents
27            ...    .find((decl: IComponentDeclaration) => decl.element === div);
28
29            ...// Assert
30            ...expect(result).not.toBeUndefined();
31            ...expect(result.component instanceof SampleComponent).toBe(true);
32            ...});
33
34            ...it("SampleComponent throws error when require child elements are missing", () => { ...
43            ...});
44            ...});
45
46            ...describe("callToAction tests", () => { ...
72            ...});
73            ...});
74
```

Karma - Setup

<http://karma-runner.github.io>

> yarn add karma --dev

> yarn add karma-jasmine --dev

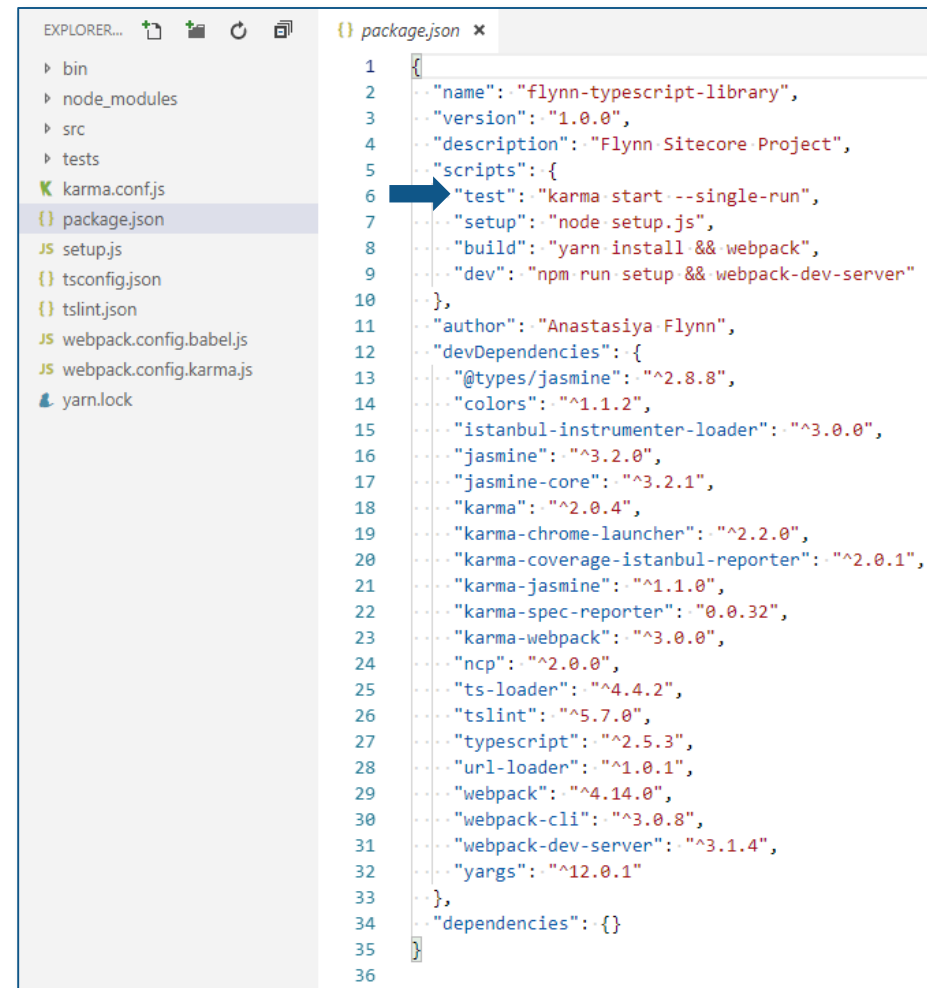
> yarn add karma-chrome-launcher --dev

> yarn add karma-coverage-istanbul-reporter --dev

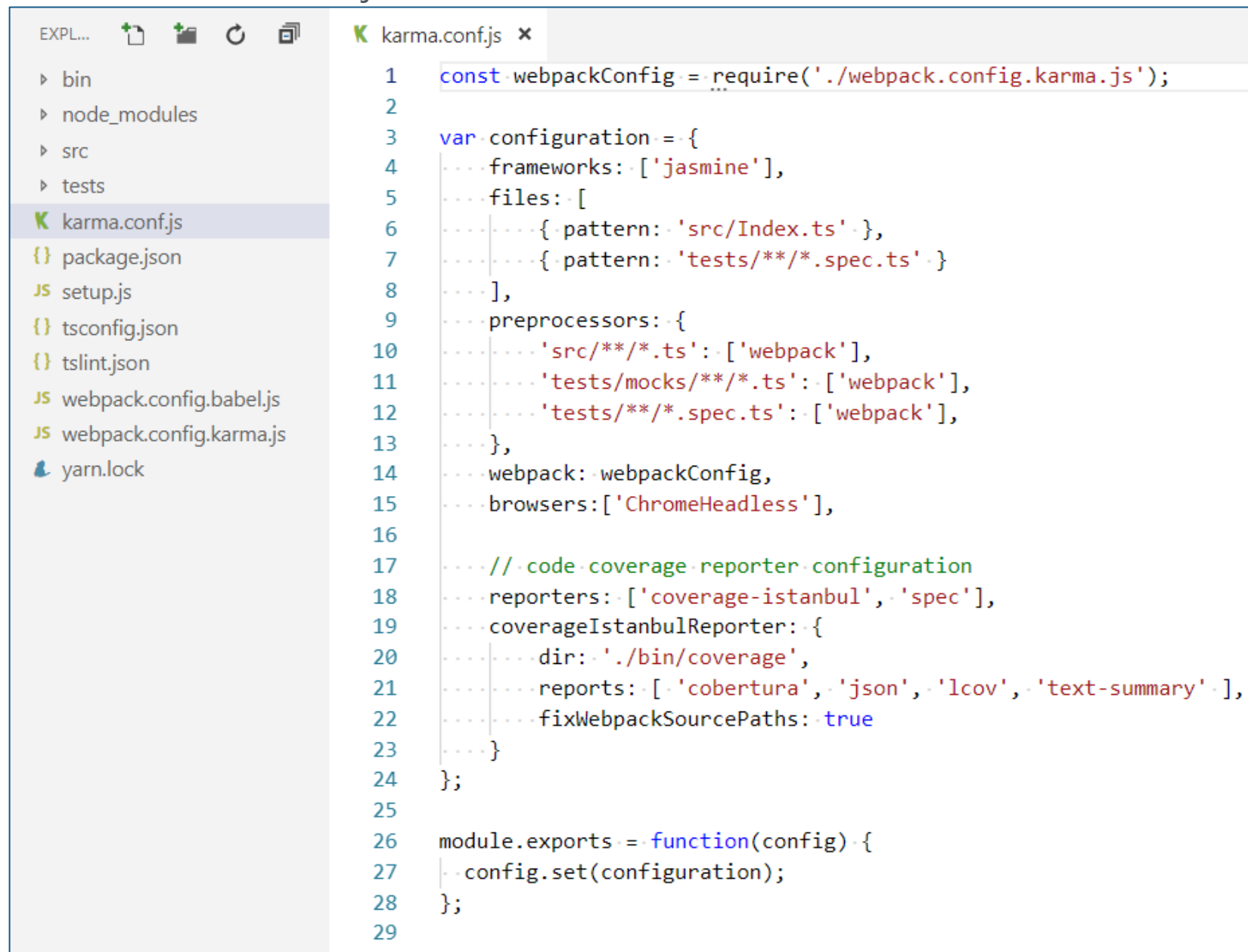
> yarn add karma-spec-reporter --dev

> yarn add karma-webpack --dev

/package.json



```
1 {
2   "name": "flynn-typescript-library",
3   "version": "1.0.0",
4   "description": "Flynn Sitecore Project",
5   "scripts": {
6     "test": "karma start --single-run",
7     "setup": "node setup.js",
8     "build": "yarn install && webpack",
9     "dev": "npm run setup && webpack-dev-server"
10  },
11   "author": "Anastasiya Flynn",
12   "devDependencies": {
13     "@types/jasmine": "^2.8.8",
14     "colors": "^1.1.2",
15     "istanbul-instrumenter-loader": "^3.0.0",
16     "jasmine": "^3.2.0",
17     "jasmine-core": "^3.2.1",
18     "karma": "^2.0.4",
19     "karma-chrome-launcher": "^2.2.0",
20     "karma-coverage-istanbul-reporter": "^2.0.1",
21     "karma-jasmine": "^1.1.0",
22     "karma-spec-reporter": "0.0.32",
23     "karma-webpack": "^3.0.0",
24     "ncp": "^2.0.0",
25     "ts-loader": "^4.4.2",
26     "tslint": "^5.7.0",
27     "typescript": "^2.5.3",
28     "url-loader": "^1.0.1",
29     "webpack": "^4.14.0",
30     "webpack-cli": "^3.0.8",
31     "webpack-dev-server": "^3.1.4",
32     "yargs": "^12.0.1"
33  },
34   "dependencies": {}
35 }
```



```
1 const webpackConfig = require('./webpack.config.karma.js');
2
3 var configuration = {
4   ... frameworks: ['jasmine'],
5   ... files: [
6     ... { pattern: 'src/Index.ts' },
7     ... { pattern: 'tests/**/*.spec.ts' }
8   ],
9   ... preprocessors: {
10     ... 'src/**/*.ts': ['webpack'],
11     ... 'tests/mocks/**/*.ts': ['webpack'],
12     ... 'tests/**/*.spec.ts': ['webpack'],
13   },
14   ... webpack: webpackConfig,
15   ... browsers: ['ChromeHeadless'],
16
17   ... // code coverage reporter configuration
18   ... reporters: ['coverage-istanbul', 'spec'],
19   ... coverageIstanbulReporter: {
20     ... dir: './bin/coverage',
21     ... reports: ['cobertura', 'json', 'lcov', 'text-summary'],
22     ... fixWebpackSourcePaths: true
23   }
24 };
25
26 module.exports = function(config) {
27   config.set(configuration);
28 };
29
```


The image shows a Visual Studio Code editor with three panels. The left panel is the Explorer view showing a file tree with folders like `bin`, `node_modules`, `src`, and `tests`, and files like `karma.conf.js`, `package.json`, `setup.js`, `tsconfig.json`, `tslint.json`, `webpack.config.babel.js`, `webpack.config.karma.js`, and `yarn.lock`. The middle panel shows the `karma.conf.js` file with the following code:

```
1 const webpackConfig = require('./webpack.config.karma.js');
2
3 var config = {
4   // ...
5   files: [
6     // ...
7     {
8       // ...
9     },
10    ],
11    // ...
12    webpack: {
13      // ...
14    },
15    // ...
16    // ...
17    // ...
18    // ...
19    // ...
20    // ...
21    // ...
22    // ...
23    // ...
24  };
25
26 module.exports = function(config) {
27   config.set(configuration);
28 };
29
```

The right panel shows the `webpack.config.karma.js` file with the following code:

```
1 const webpackConfig = require('./webpack.config.babel.js');
2
3 // Extend base webpack config with istanbul-instrumenter-loader
4 // configuration for code coverage reporting during karma run.
5 webpackConfig.devtool = "inline-source-map";
6
7 webpackConfig.module.rules.push({
8   enforce: "post",
9   test: /\.ts$/,
10   loader: 'istanbul-instrumenter-loader',
11   query: {
12     esModules: true
13   },
14   exclude: [
15     'node_modules',
16     /\.spec\.ts$/
17   ],
18 });
19
20 module.exports = webpackConfig;
21
```

A blue arrow points from the `require('./webpack.config.karma.js')` line in `karma.conf.js` to the `webpack.config.karma.js` file in the Explorer view.

Jasmine/Karma – Demo





Get the code

<https://bitbucket.org/anastasiya29/yeoman-sitecore-generator>



FOR DISCUSSION PURPOSES ONLY. Sitecore Confidential and Proprietary. © 2018 Sitecore Corporation A/S. All rights reserved. Sitecore® and Own the Experience® are registered trademarks of Sitecore Corporation A/S. All other brand and product names are the property of their respective owners.

Let's continue the conversation on
Twitter @AnastasiyaFlynn

