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

Presented by Anastasiya Flynn,
Sitecore Architect at Parke

sitecore
SYMPOSIUM
ORLANDO

#SitecoreSYM @AnastasiyaFlynn

sitecore SYMPOSIUM ORLANDO

arke

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**

New Solution Checklist

C# Projects

- Helix structure
- Sitecore NuGet references
- Build profiles
- Unit tests

TDS Projects

- Helix structure
- Validation rules
- Package generation
- Glass models generation
- Assembly inclusion

Code Style

- Resharper inspection rules
- Stylecop configuration
- EditorConfig file

TypeScript Library

- Headless build
- Bundling/ minification
- Transpilation to ES5
- Unit Tests/ Runner
- Linting

Front-End Assets

- Sass Compilation
- Bundling/ minification
- Linting

DevOps

- Gulpfile
- Node dependencies

New Solution Checklist

Create C# projects

Find correct Sitecore NuGets

Add TDS projects

Set up build profiles

Set up package generation

How do we build Sass and JS again?

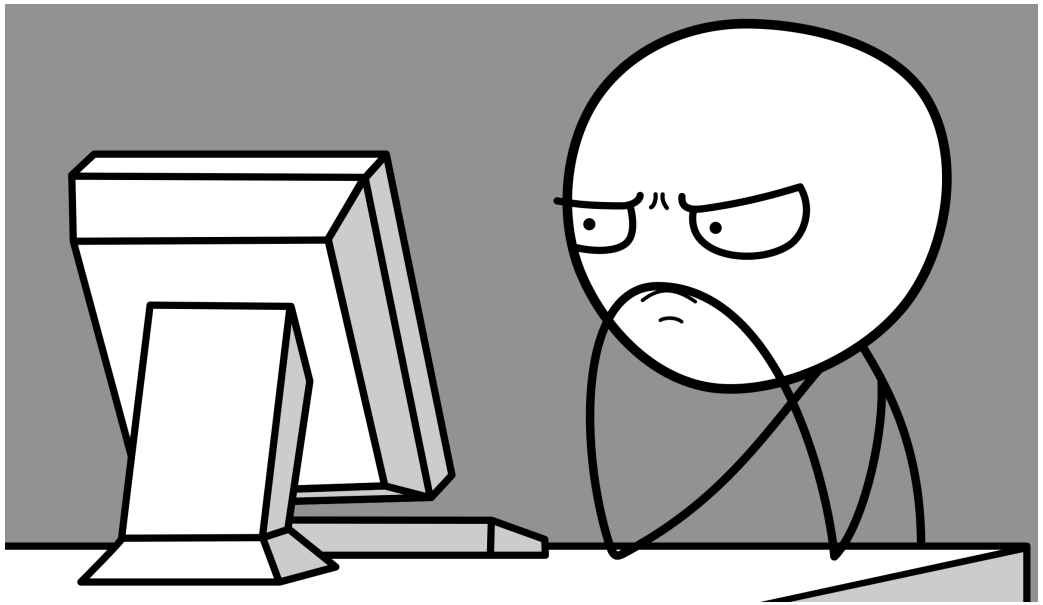
Let's just build it

YSOD

Coffee

Is my target framework right?

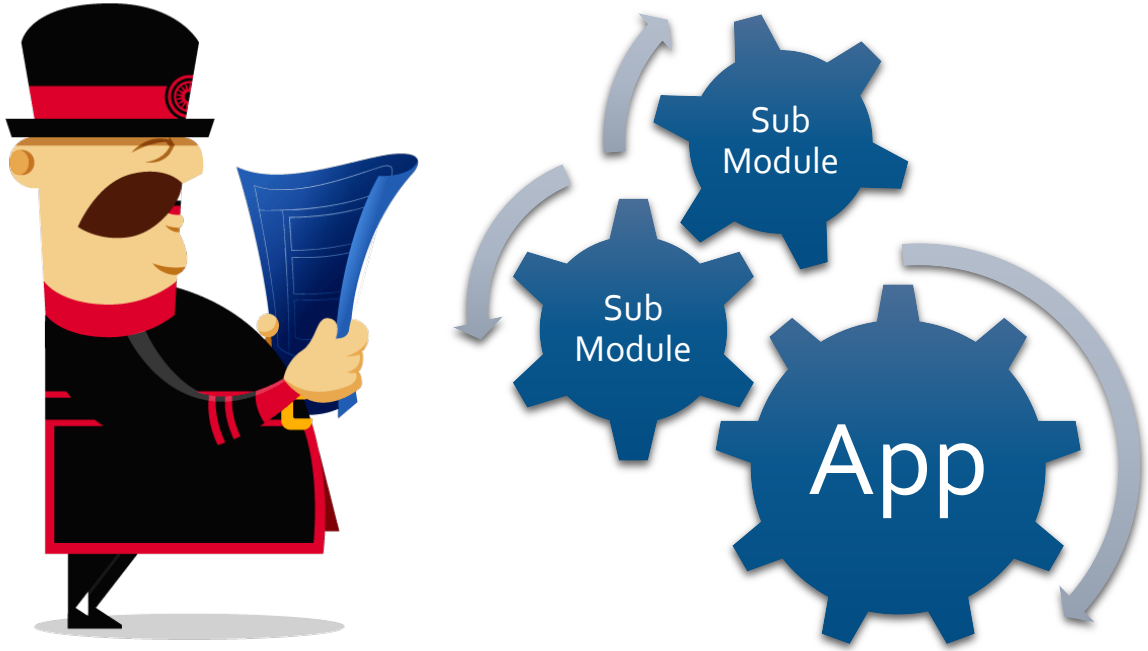
Repeat for other Helix layers



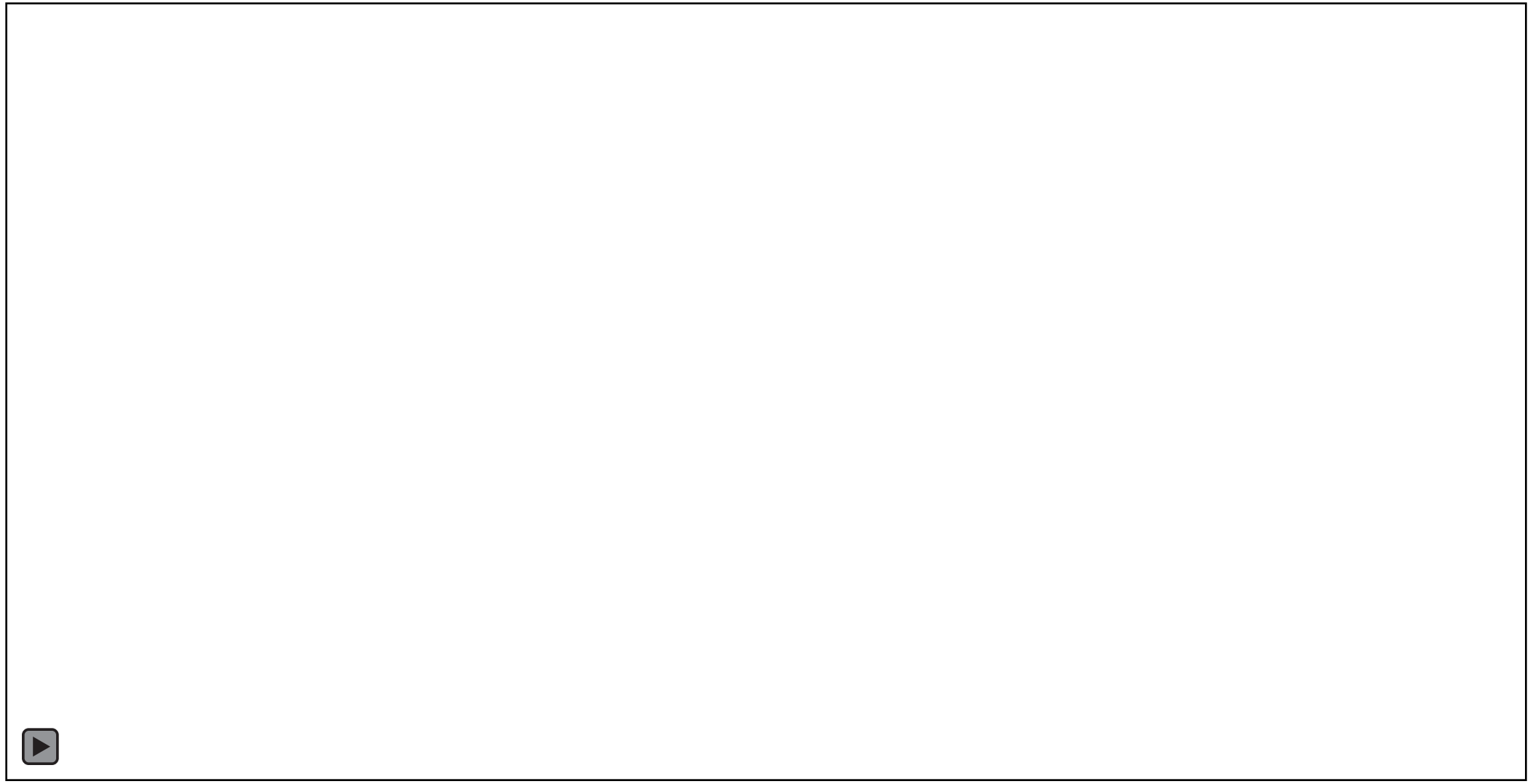
Automate Project Setup With Yeoman

http://yeoman.io/

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

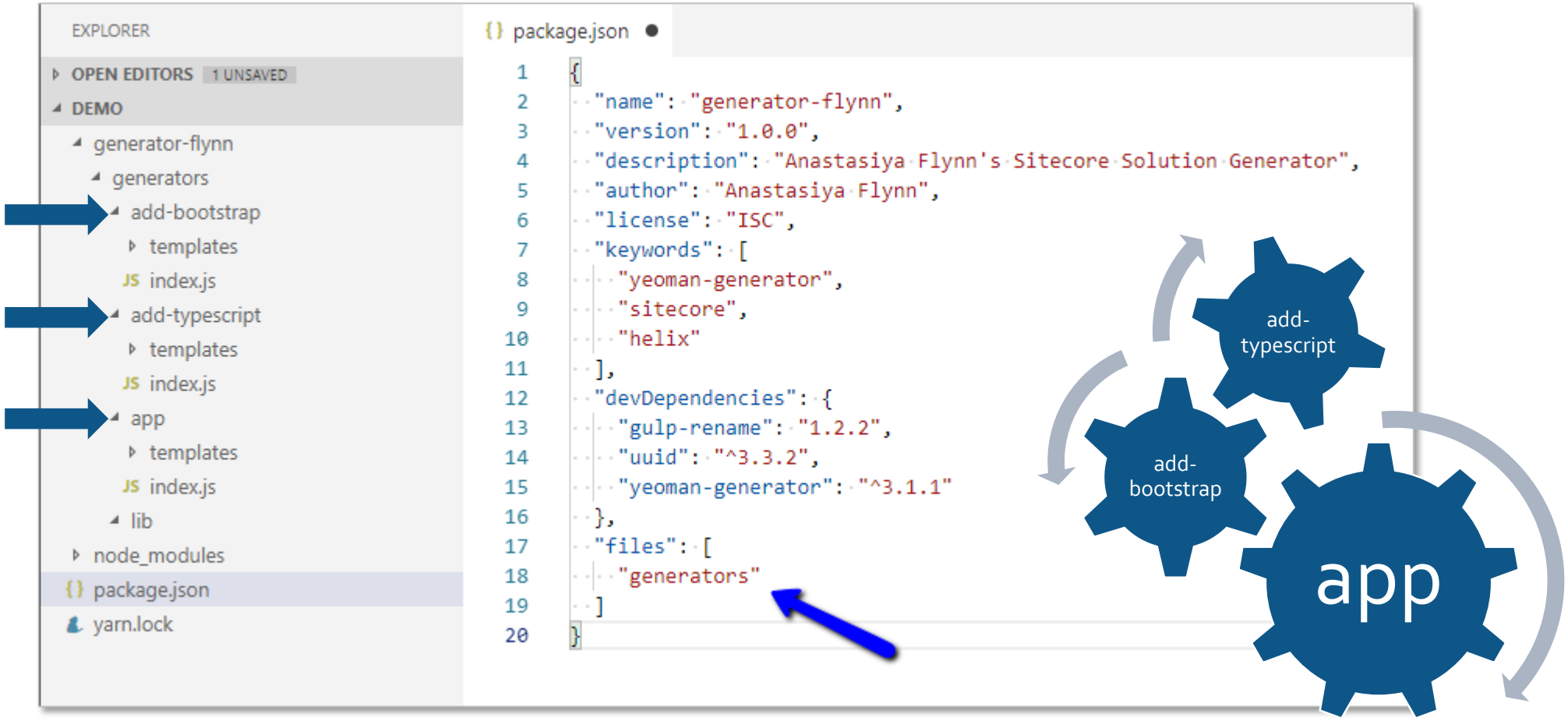


Yeoman Generator – Setting Up

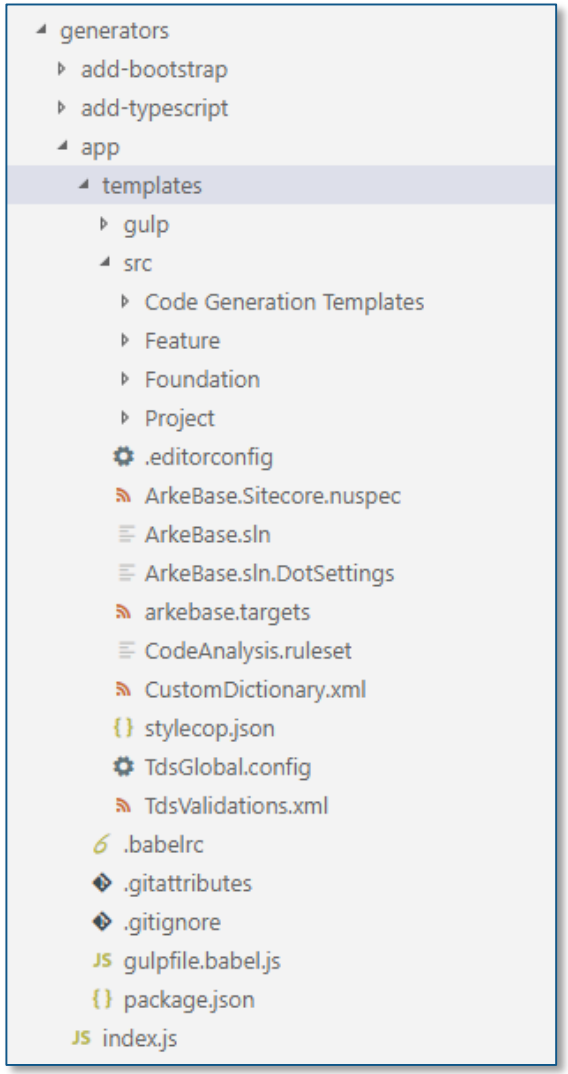


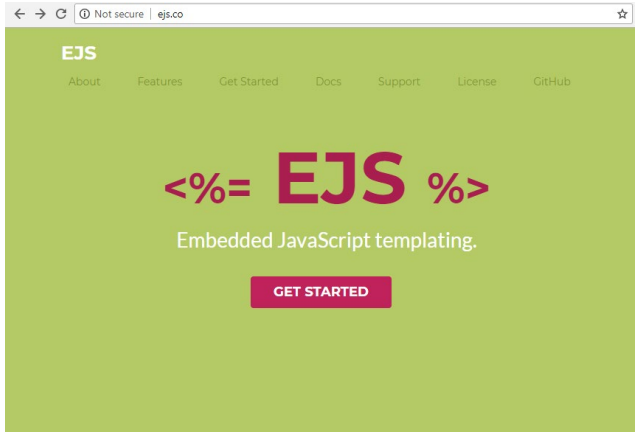
Yeoman – Setting Up

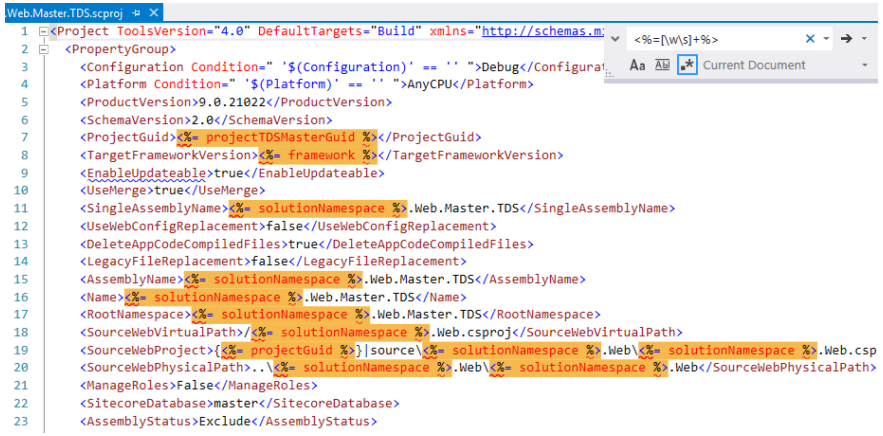
/package.json



Yeoman – Template Variables







Solution name

Solution namespace

MVC area name

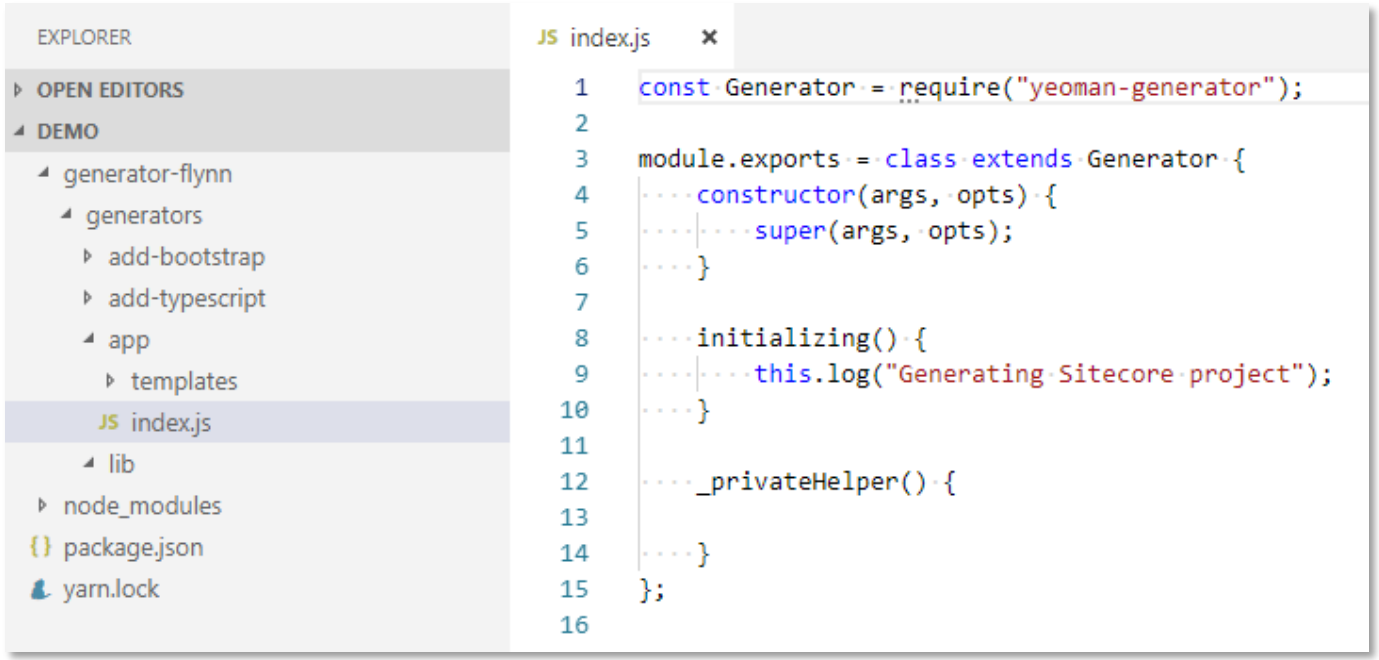
Local Sitecore URL

Sitecore version

.NET framework

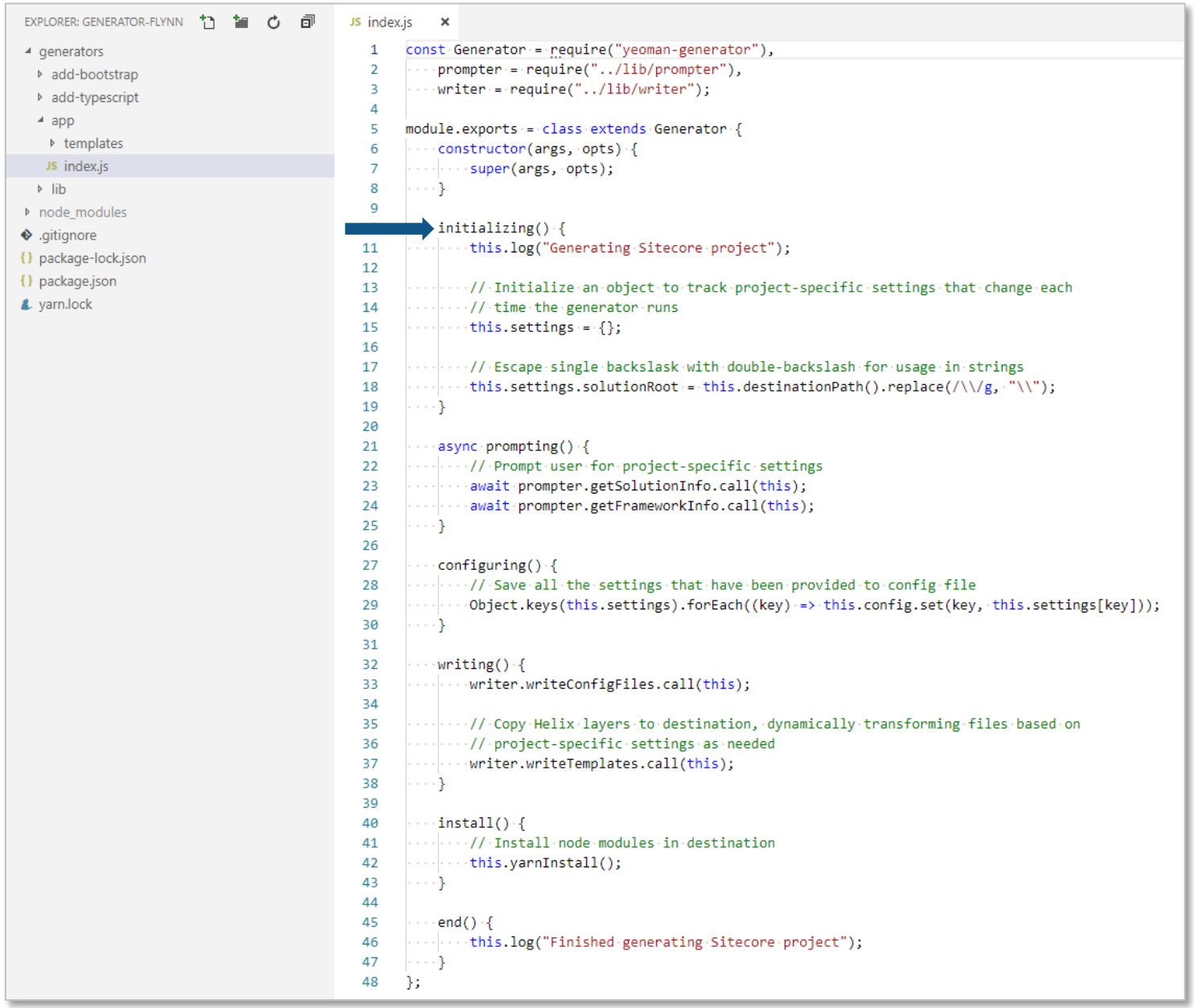
Yeoman – Generator Lifecycle

/generators/app/index.js



Initializing → Prompting → Configuring → Default → Writing → Conflicts → Install → End

/generators/app/index.js



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

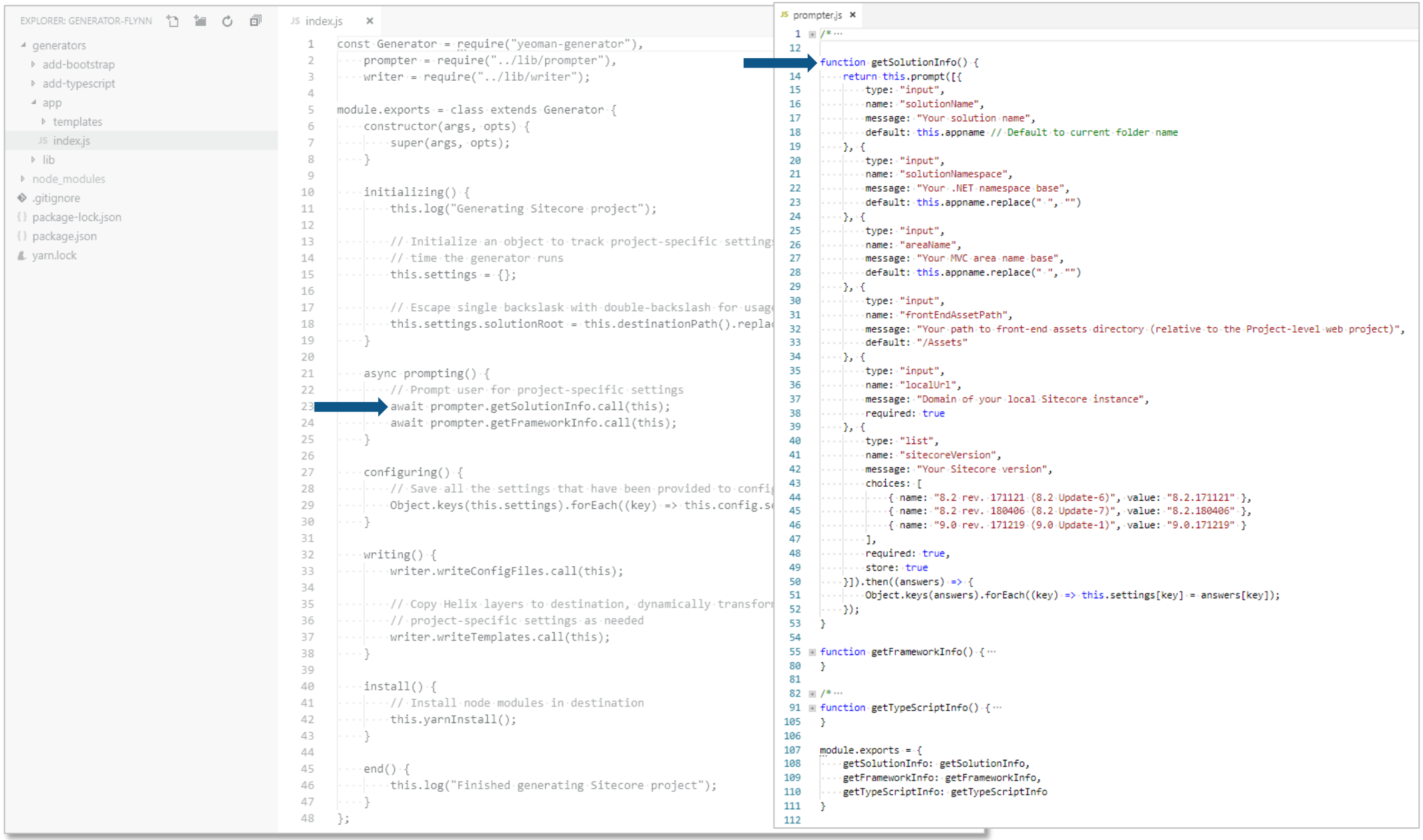
Writing
- Write settings to config file on disk
- Copy templates to destination, transforming content using project-specific settings

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
    index.js
  lib
node_modules
.gitignore
package-lock.json
package.json
yarn.lock
```

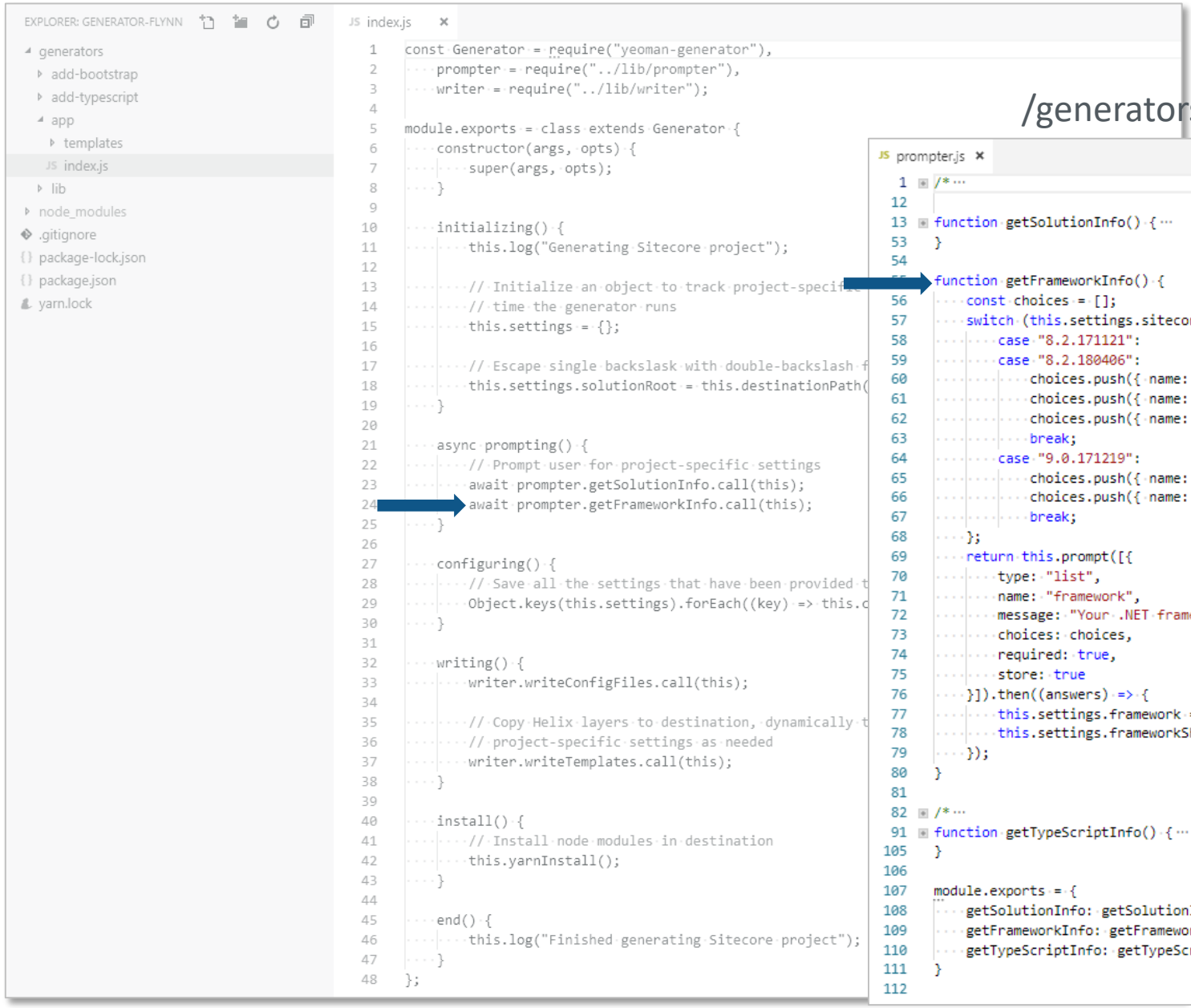
index.js

```
const Generator = require("yeoman-generator"),
    prompter = require("../lib/prompter"),
    writer = require("../lib/writer");

module.exports = class extends Generator {
    constructor(args, opts) {
        super(args, opts);
    }

    initializing() {
        this.log("Generating Sitecore project");

        // Initialize an object to track project-specific setting
        // time the generator runs
        this.settings = {};

        // Escape single backslask with double-backslash for usag
        this.settings.solutionRoot = this.destinationPath().repla
    }

    async prompting() {
        // Prompt user for project-specific settings
        await prompter.getSolutionInfo.call(this);
        await prompter.getFrameworkInfo.call(this);
    }

    configuring() {
        // Save all the settings that have been provided to confi
        Object.keys(this.settings).forEach((key) => this.config.s
    }

    writing() {
        writer.writeConfigFiles.call(this);

        // Copy Helix layers to destination, dynamically transfor
        // project-specific settings as needed
        writer.writeTemplates.call(this);
    }

    install() {
        // Install node modules in destination
        this.yarnInstall();
    }

    end() {
        this.log("Finished generating Sitecore project");
    }
};
```

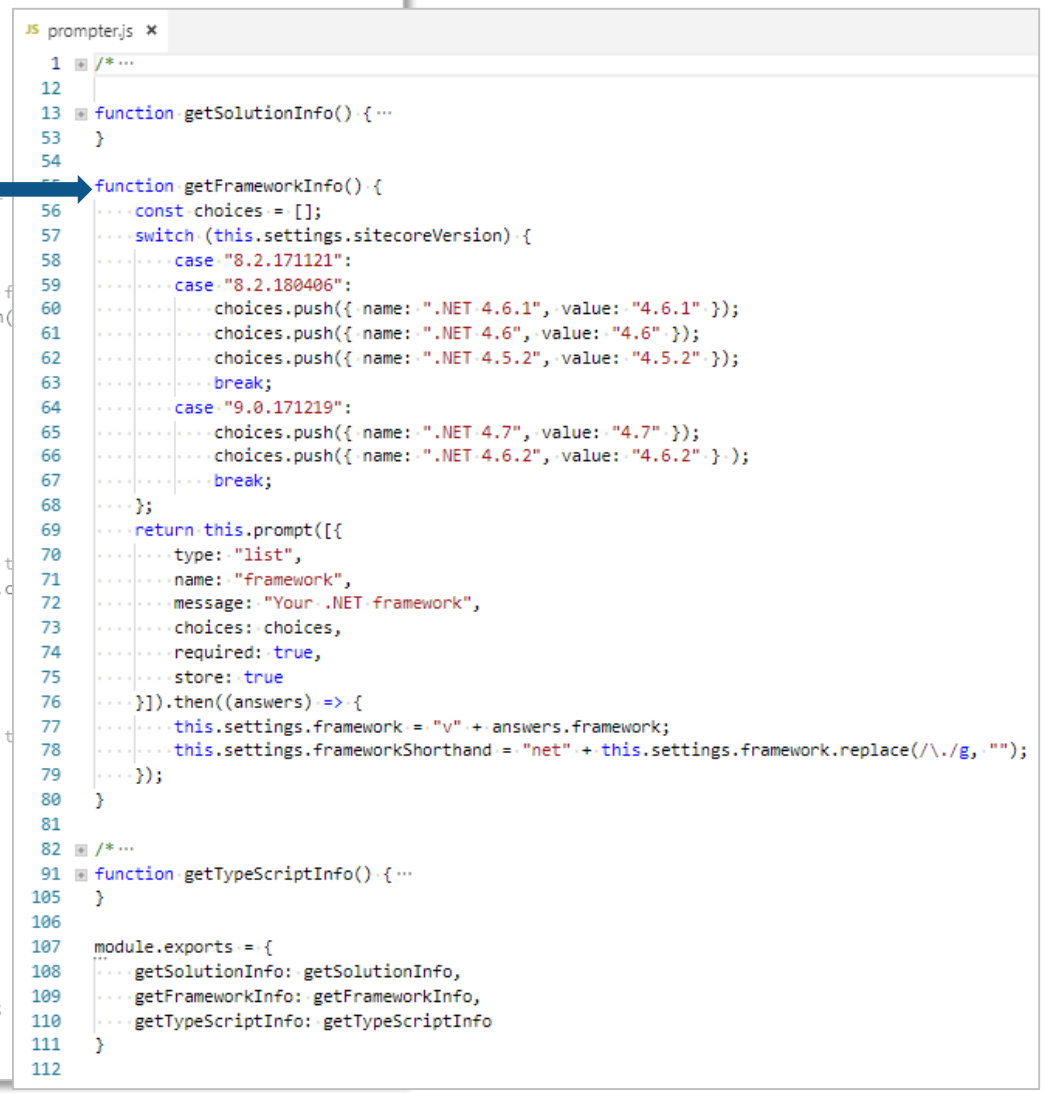
prompter.js

```
/*...

function getSolutionInfo() {
    return this.prompt([{
        type: "input",
        name: "solutionName",
        message: "Your solution name",
        default: this.appname // Default to current folder name
    }, {
        type: "input",
        name: "solutionNamespace",
        message: "Your .NET namespace base",
        default: this.appname.replace(" ", "")
    }, {
        type: "input",
        name: "areaName",
        message: "Your MVC area name base",
        default: this.appname.replace(" ", "")
    }, {
        type: "input",
        name: "frontEndAssetPath",
        message: "Your path to front-end assets directory (relative to the Project-level web project)",
        default: "/Assets"
    }, {
        type: "input",
        name: "localUrl",
        message: "Domain of your local Sitecore instance",
        required: true
    }, {
        type: "list",
        name: "sitecoreVersion",
        message: "Your Sitecore version",
        choices: [
            { name: "8.2 rev. 171121 (8.2 Update-6)", value: "8.2.171121" },
            { name: "8.2 rev. 180406 (8.2 Update-7)", value: "8.2.180406" },
            { name: "9.0 rev. 171219 (9.0 Update-1)", value: "9.0.171219" }
        ],
        required: true,
        store: true
    }]).then((answers) => {
        Object.keys(answers).forEach((key) => this.settings[key] = answers[key]);
    });
}

function getFrameworkInfo() {...
}

/*...
function getTypeScriptInfo() {...
}

module.exports = {
    getSolutionInfo: getSolutionInfo,
    getFrameworkInfo: getFrameworkInfo,
    getTypeScriptInfo: getTypeScriptInfo
}
```

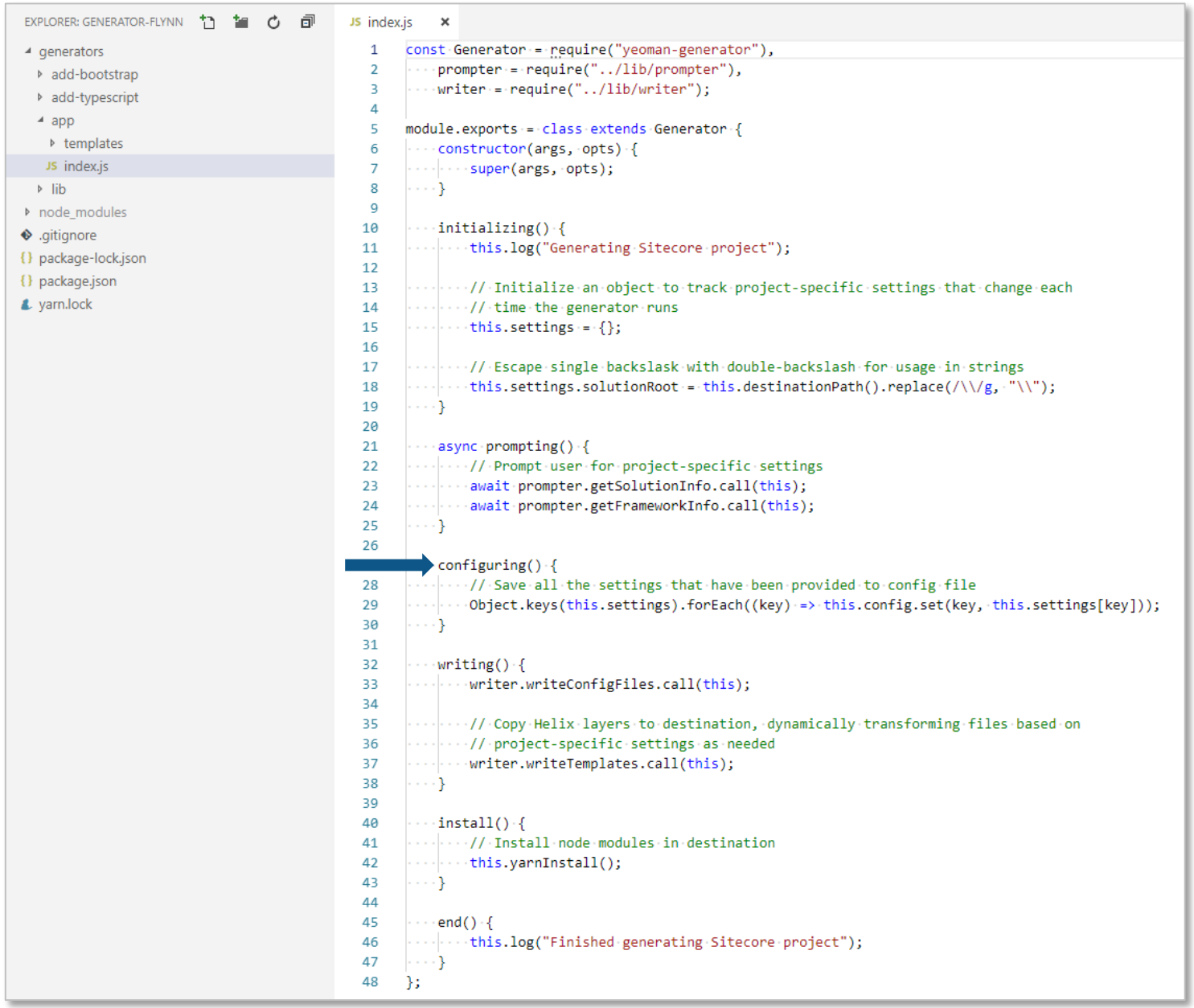
/generators/app/index.js



/generators/lib/prompter.js



/generators/app/index.js



Initializing
- Log Message
- Initialize settings object

Prompting
- Prompt user for project-specific settings

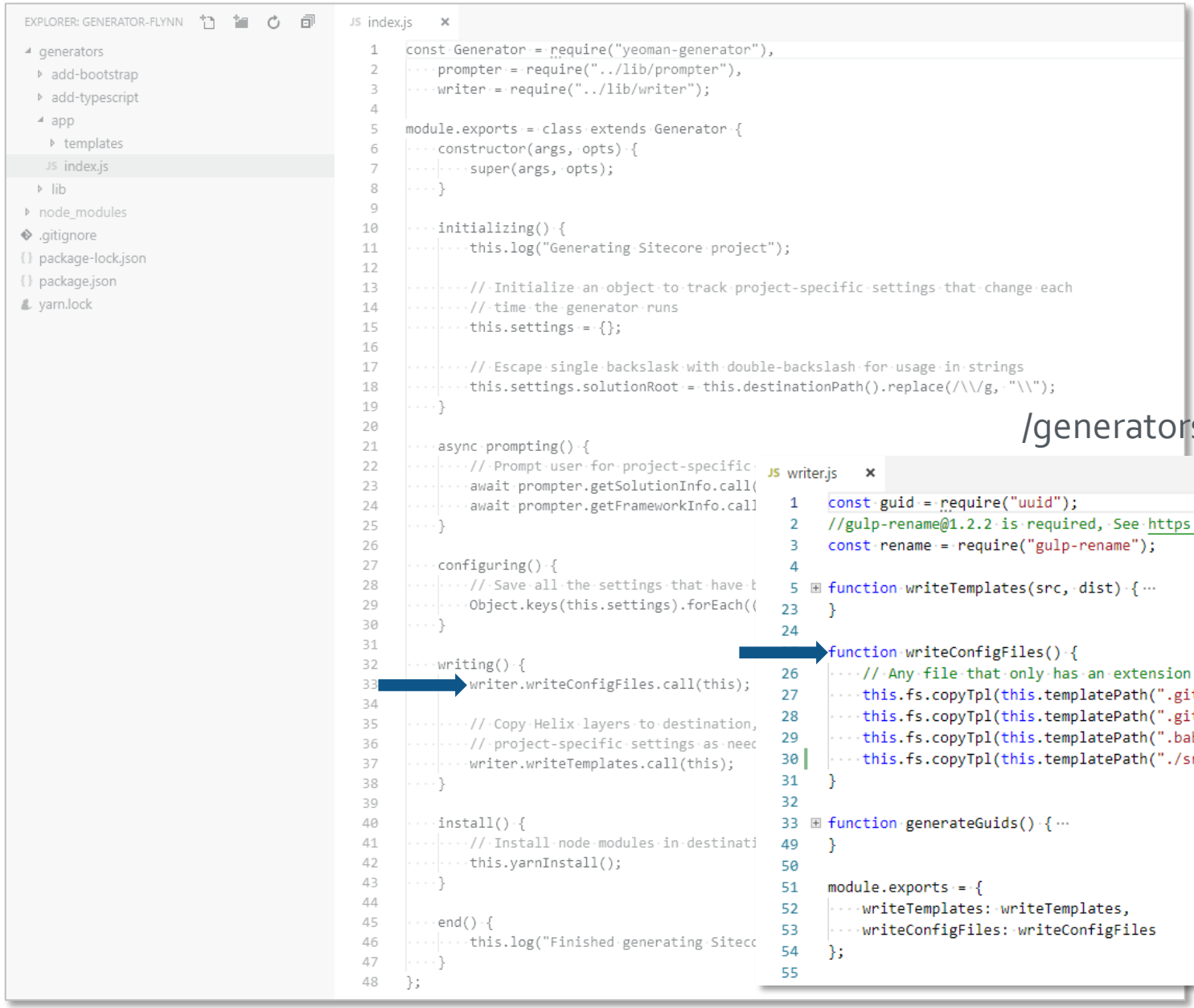
Configuring
- Save all the settings that have been provided to config file

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

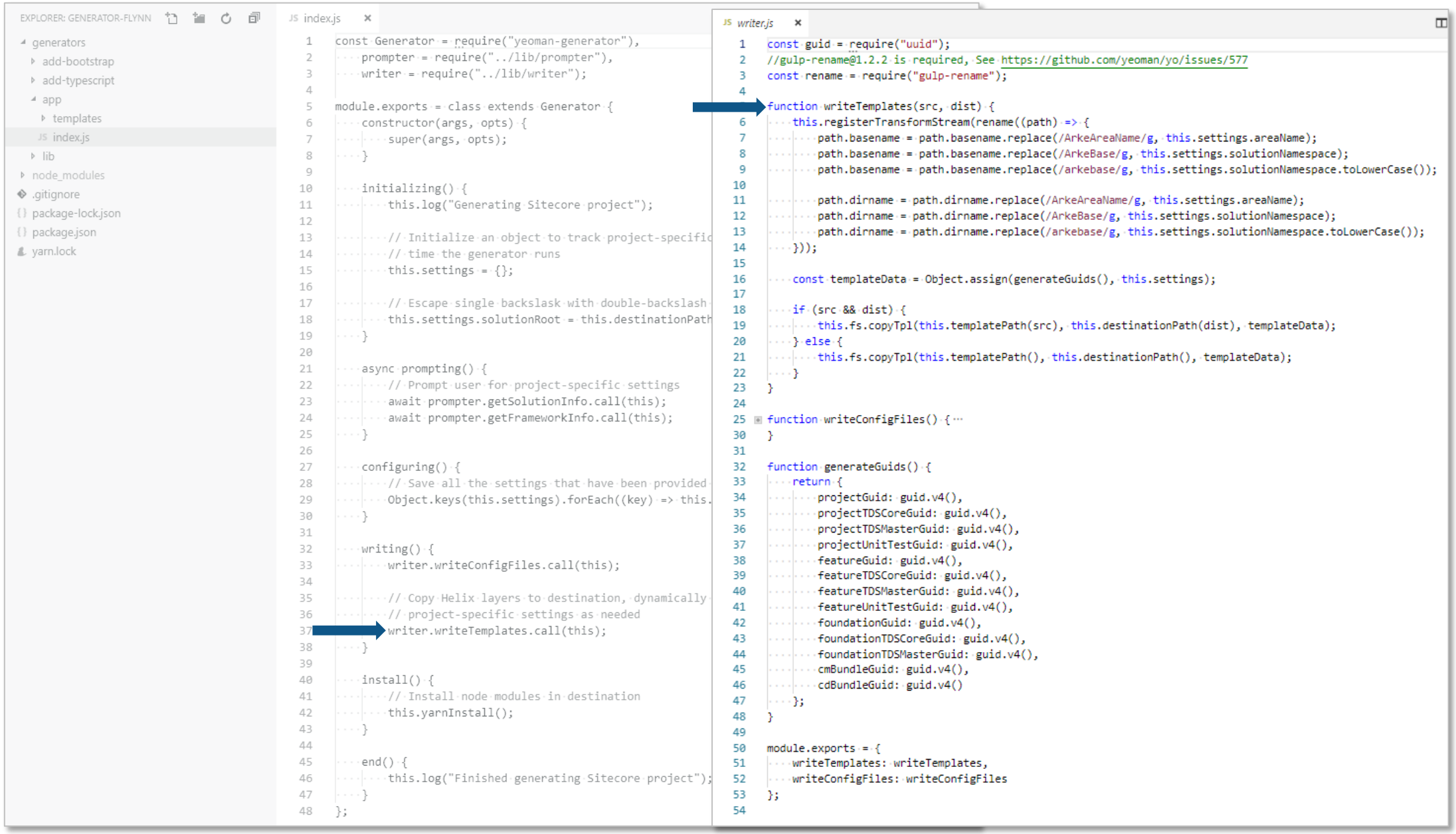
/generators/app/index.js



/generators/lib/writer.js

/generators/app/index.js

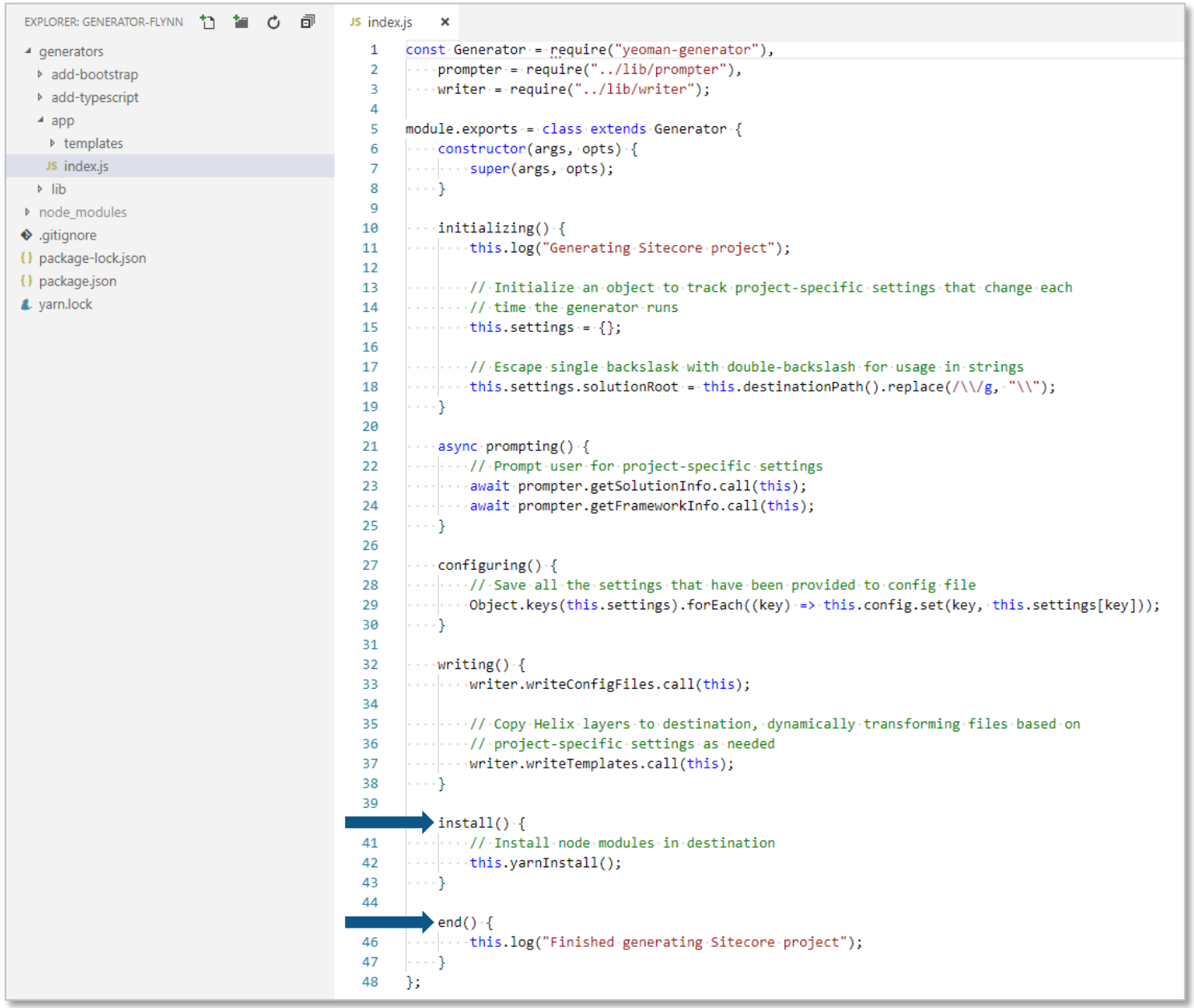
/generators/lib/writer.js




```
const Generator = require("yeoman-generator"),
    prompter = require("../lib/prompter"),
    writer = require("../lib/writer");

module.exports = class extends Generator {
    constructor(args, opts) {
        super(args, opts);
    }

    initializing() {
        this.log("Generating Sitecore project");

        // Initialize an object to track project-specific
        // time the generator runs
        this.settings = {};

        // Escape single backslask with double-backslash
        this.settings.solutionRoot = this.destinationPath
    }

    async prompting() {
        // Prompt user for project-specific settings
        await prompter.getSolutionInfo.call(this);
        await prompter.getFrameworkInfo.call(this);
    }

    configuring() {
        // Save all the settings that have been provided
        Object.keys(this.settings).forEach((key) => this.
    }

    writing() {
        writer.writeConfigFiles.call(this);

        // Copy Helix layers to destination, dynamically
        // project-specific settings as needed
        writer.writeTemplates.call(this);
    }

    install() {
        // Install node modules in destination
        this.yarnInstall();
    }

    end() {
        this.log("Finished generating Sitecore project");
    }
};
```

```
const guid = require("uuid");
//gulp-rename@1.2.2 is required, See https://github.com/yeoman/yo/issues/577
const rename = require("gulp-rename");

function writeTemplates(src, dist) {
    this.registerTransformStream(rename((path) => {
        path.basename = path.basename.replace(/ArkeAreaName/g, this.settings.areaName);
        path.basename = path.basename.replace(/ArkeBase/g, this.settings.solutionNamespace);
        path.basename = path.basename.replace(/arkebase/g, this.settings.solutionNamespace.toLowerCase());

        path.dirname = path.dirname.replace(/ArkeAreaName/g, this.settings.areaName);
        path.dirname = path.dirname.replace(/ArkeBase/g, this.settings.solutionNamespace);
        path.dirname = path.dirname.replace(/arkebase/g, this.settings.solutionNamespace.toLowerCase());
    }));

    const templateData = Object.assign(generateGuids(), this.settings);

    if (src && dist) {
        this.fs.copyTpl(this.templatePath(src), this.destinationPath(dist), templateData);
    } else {
        this.fs.copyTpl(this.templatePath(), this.destinationPath(), templateData);
    }
}

function writeConfigFiles() {…
}

function generateGuids() {
    return {
        projectGuid: guid.v4(),
        projectTDSCoreGuid: guid.v4(),
        projectTDSMasterGuid: guid.v4(),
        projectUnitTestGuid: guid.v4(),
        featureGuid: guid.v4(),
        featureTDSCoreGuid: guid.v4(),
        featureTDSMasterGuid: guid.v4(),
        featureUnitTestGuid: guid.v4(),
        foundationGuid: guid.v4(),
        foundationTDSCoreGuid: guid.v4(),
        foundationTDSMasterGuid: guid.v4(),
        cmBundleGuid: guid.v4(),
        cdBundleGuid: guid.v4()
    };
}

module.exports = {
    writeTemplates: writeTemplates,
    writeConfigFiles: writeConfigFiles
};
```

/generators/app/index.js



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

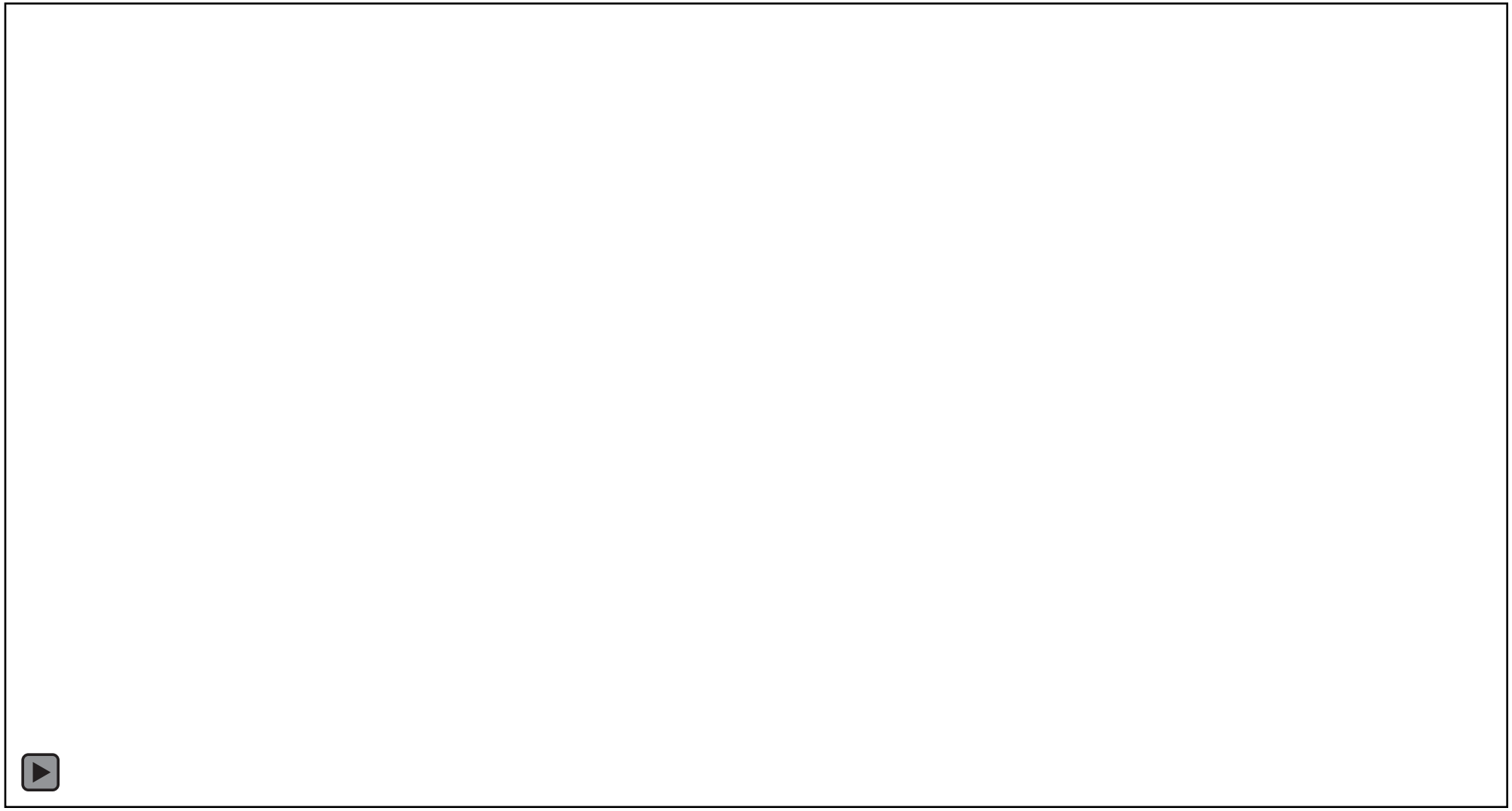
Writing
- Write settings to config file on disk
- Copy templates to destination, transforming content using project-specific settings

Install
- Install node modules in destination

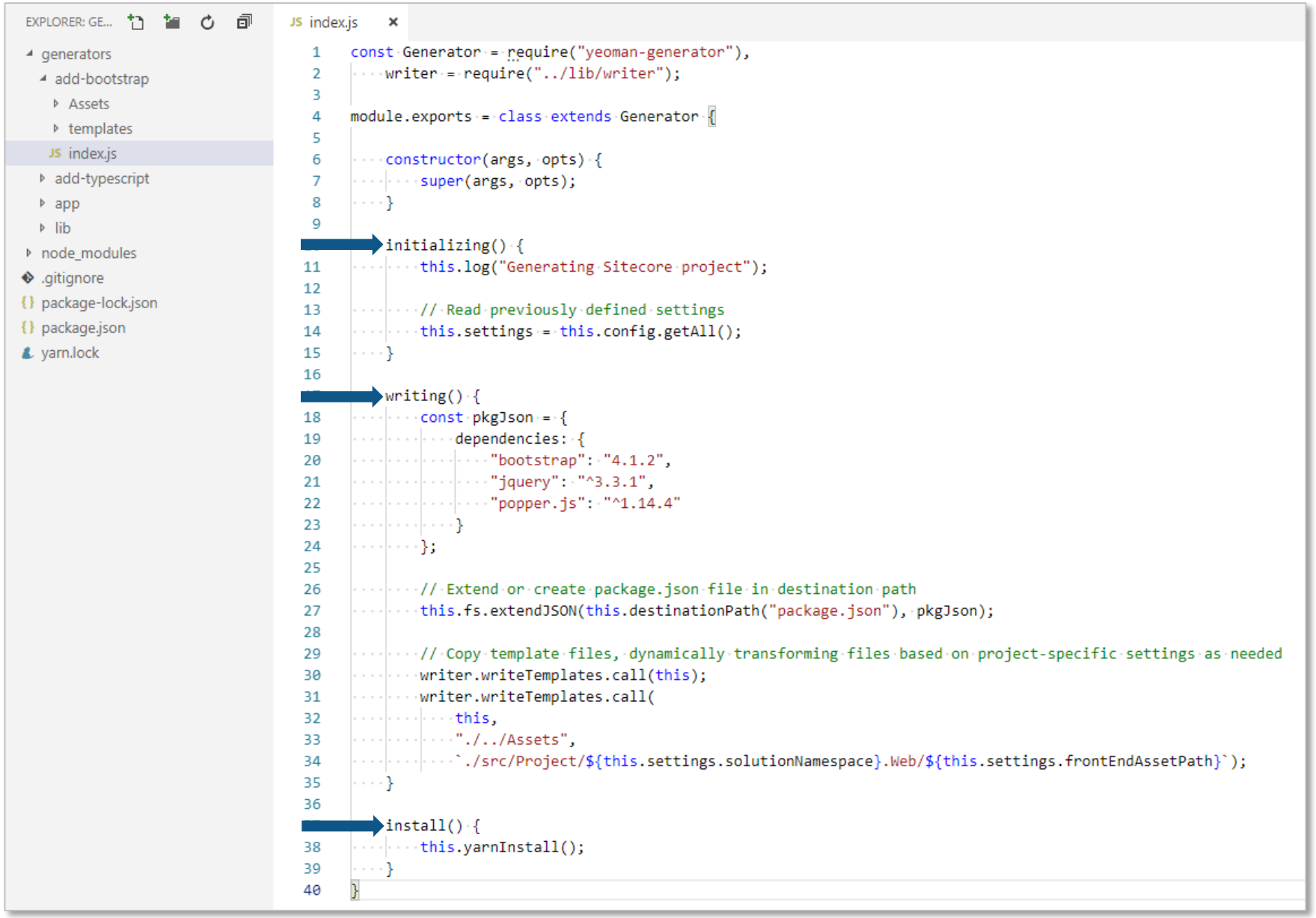
End
- Log Message

Yeoman – Exposing Generator For Testing

Yeoman – Running Main Generator



/generators/add-bootstrap/index.js



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!

- **C# Projects**
  - Helix structure
  - Sitecore NuGet references
  - Build profiles
  - Unit tests
- **TDS Projects**
  - Helix structure
  - Validation rules
  - Package generation
  - Glass models generation
  - Assembly inclusion
- **Code Style**
  - Resharper inspection rules
  - Stylecop configuration
  - EditorConfig file
- **TypeScript Library**
  - Headless build
  - Bundling/ minification
  - Transpilation to ES5
  - Unit Tests/ Runner
  - Linting
- **Front-End Assets**
  - Sass Compilation
  - Bundling/ minification
  - Linting
- **DevOps**
  - Gulpfile
  - Node dependencies

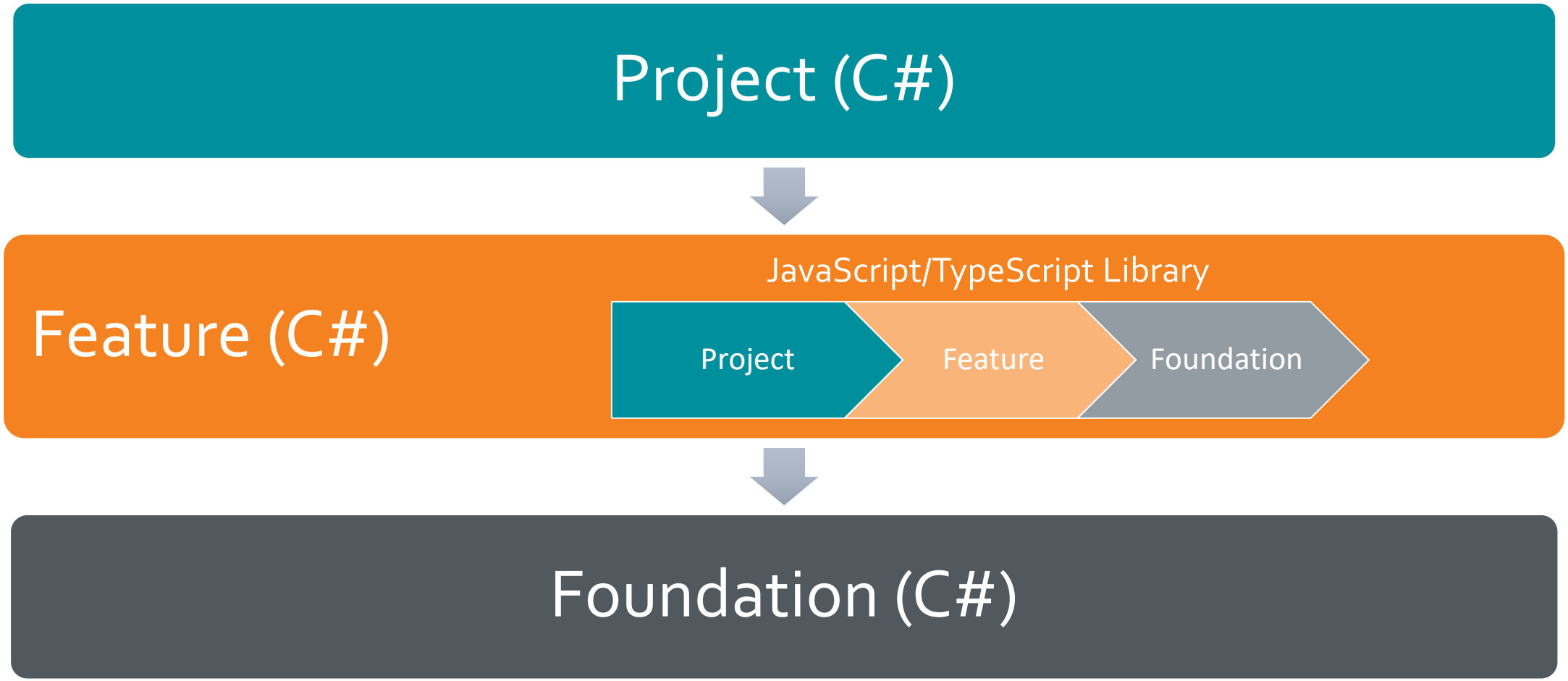
Development

Use **Webpack** to

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

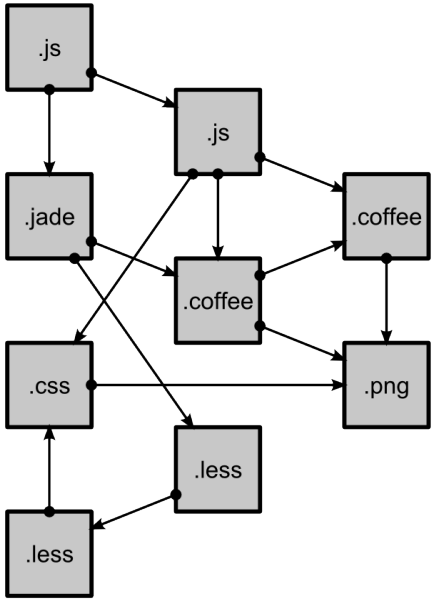
Setup → Development → Testing

TypeScript Helix

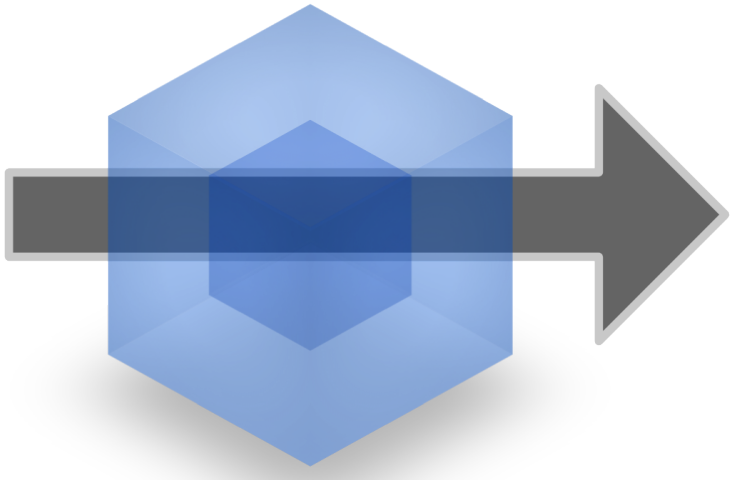


What is Webpack?

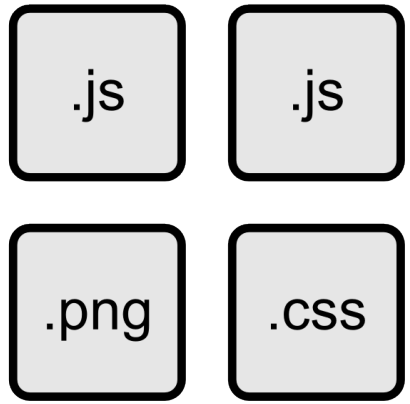
https://webpack.js.org/



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

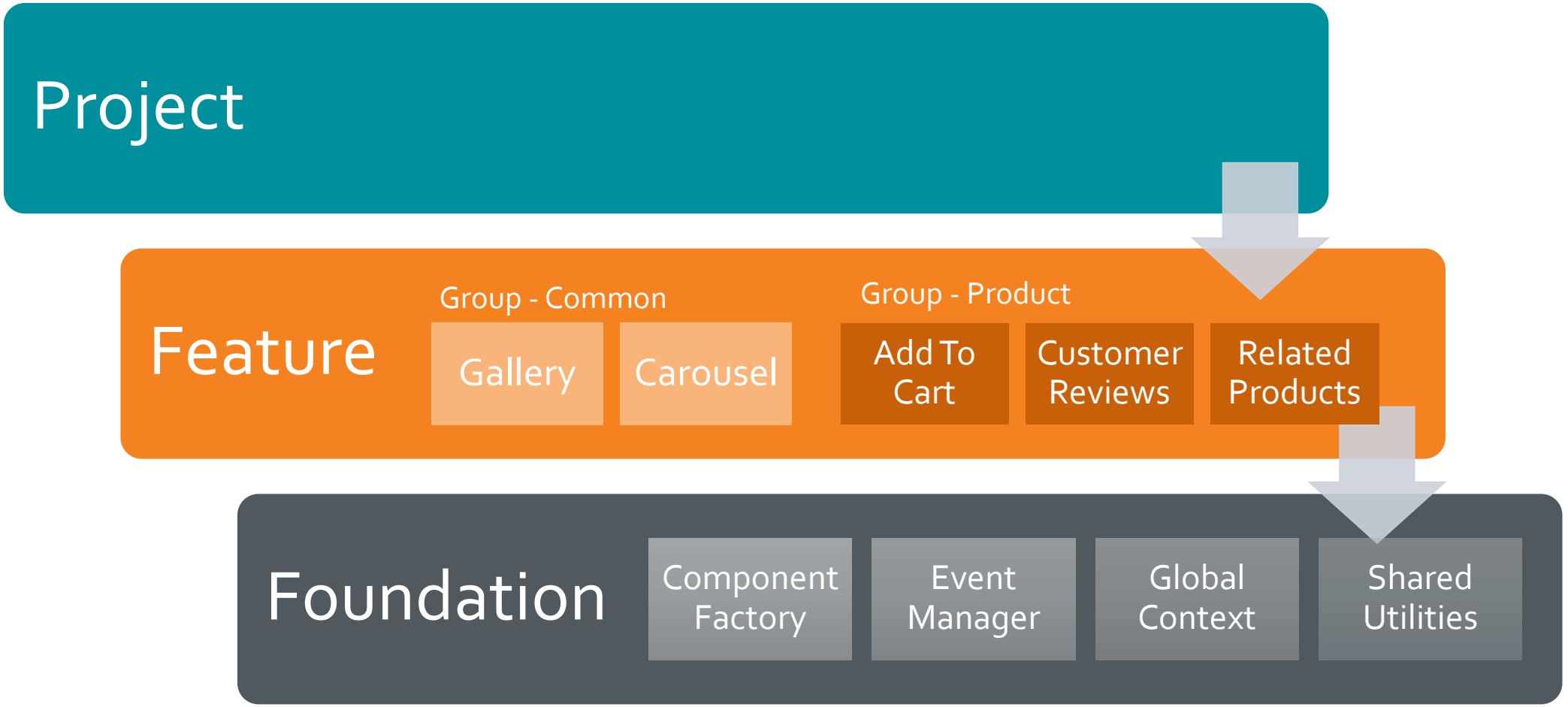
```
const path = require('path');

module.exports = {
    entry: './path/to/my/entry/file.js',
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: 'my-first-webpack.bundle.js'
    },
    module: {
        rules: [
            { test: /\.txt$/, use: 'raw-loader' }
        ]
    }
};
```

/package.json

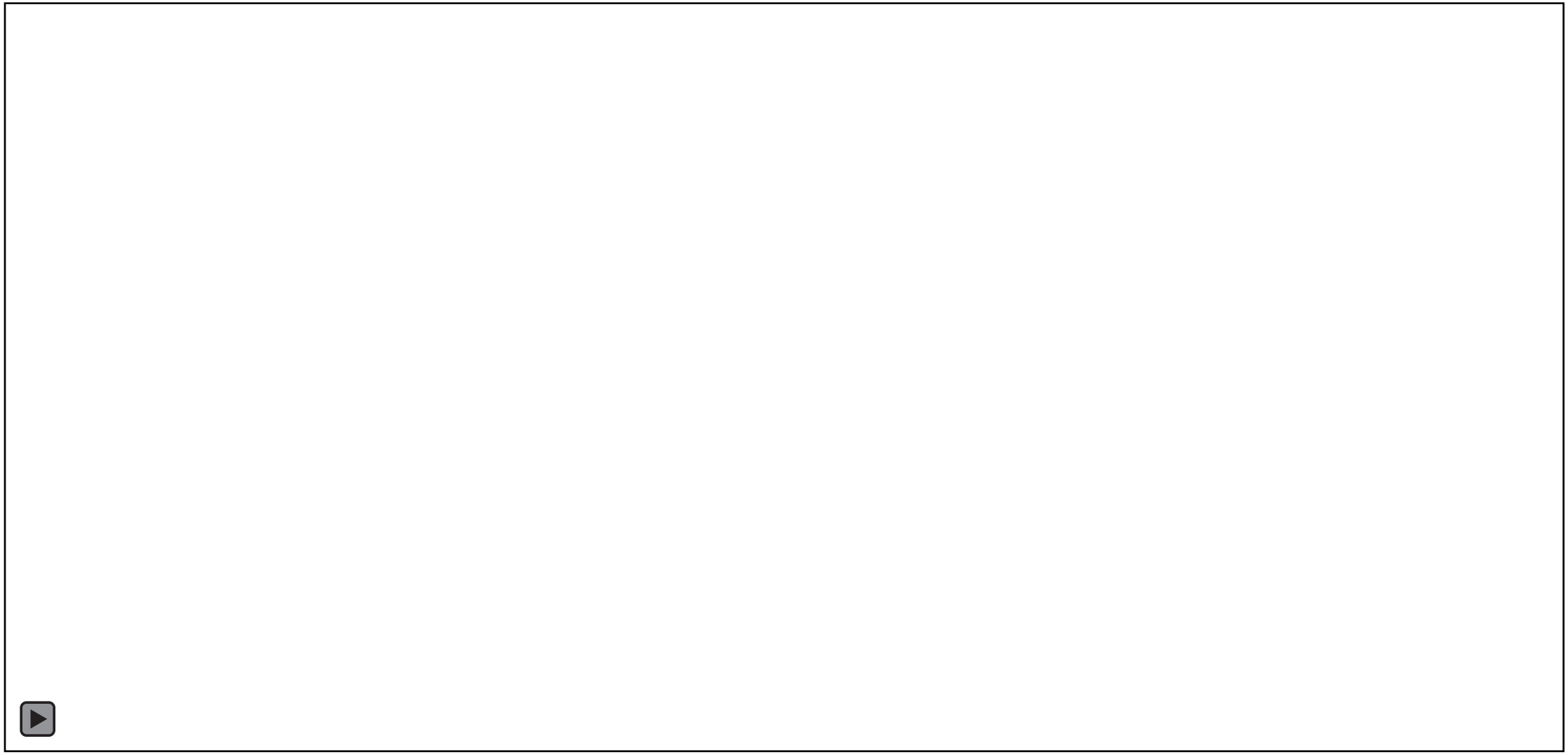
package.json

```
{
  "name": "flynn-typescript-library",
  "version": "1.0.0",
  "description": "",
  "scripts": {
    "build": "yarn install && webpack"
  },
  "devDependencies": {…
  }
}
```

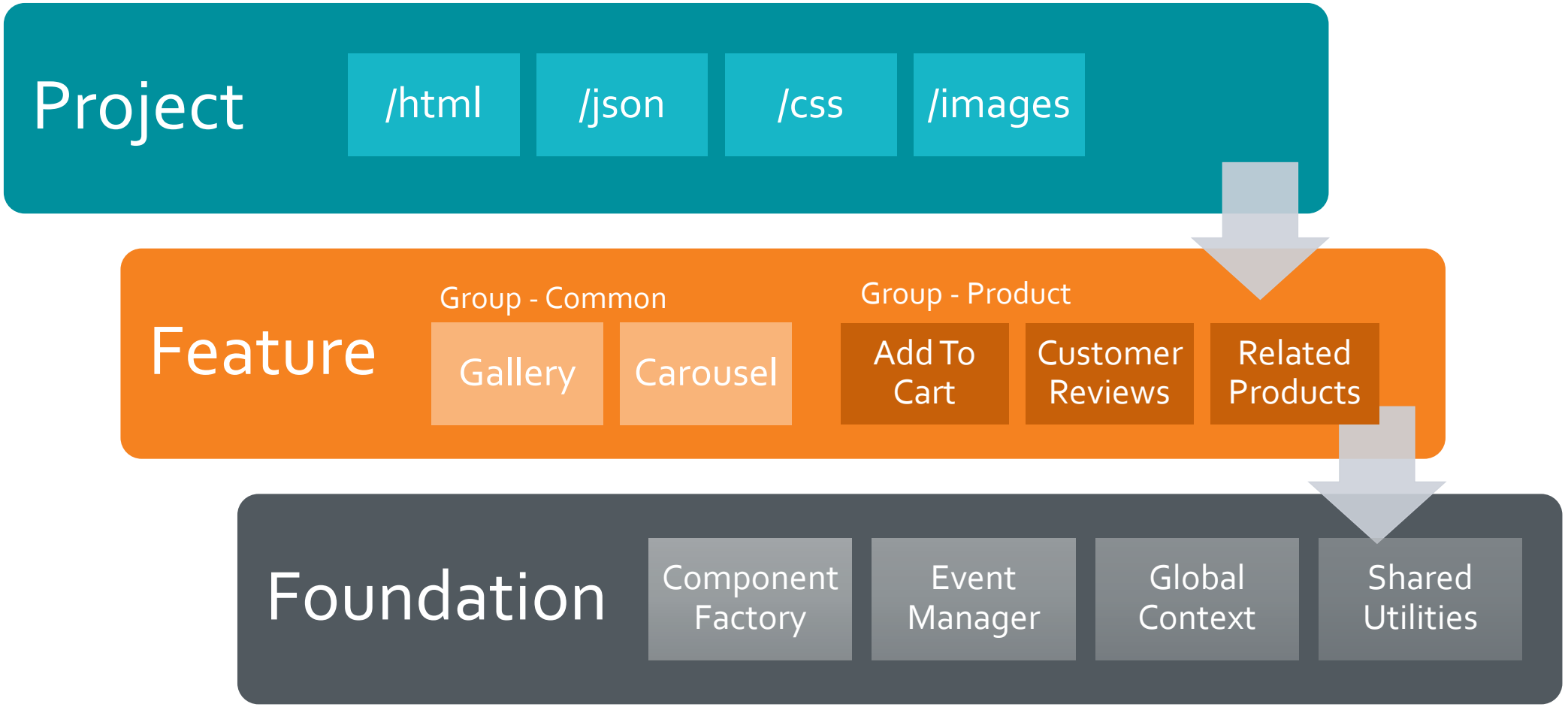
TypeScript Helix



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

```
var path = require('path');

module.exports = {
  //...
  devServer: {
    contentBase: path.join(__dirname, 'dist'),
    compress: true,
    port: 9000
  }
};
```

/package.json

package.json

```
{
  "name": "flynn-typescript-library",
  "version": "1.0.0",
  "description": "",
  "scripts": {
    "setup": "node setup.js",
    "build": "yarn install && webpack",
    "dev": "npm run setup && webpack-dev-server"
  },
  "devDependencies": {…
  }
}
```

Webpack – Demo 'npm run dev'

Dev Server – Ajax?

TypeScript Helix – Best practice patterns

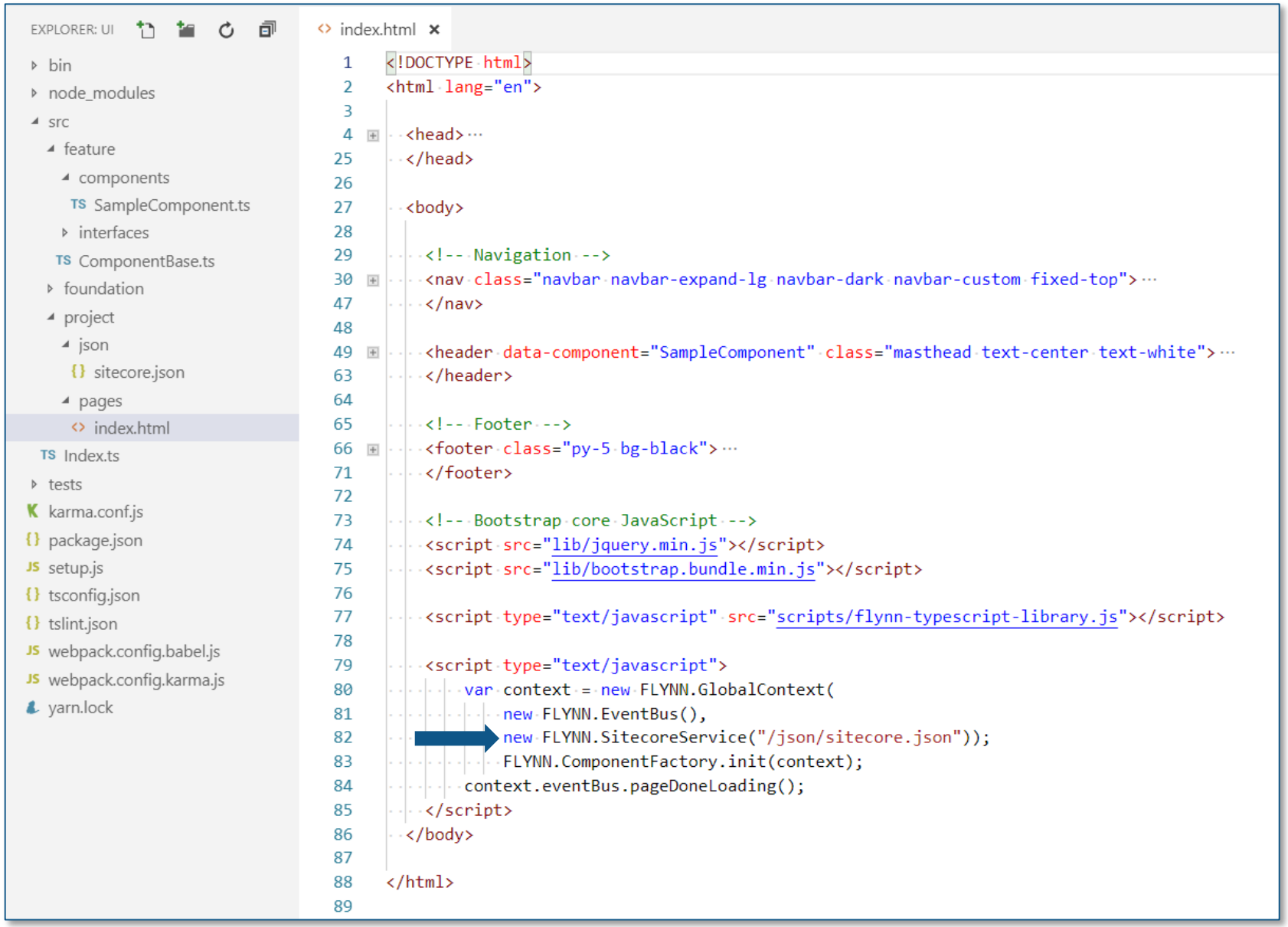
Server interaction

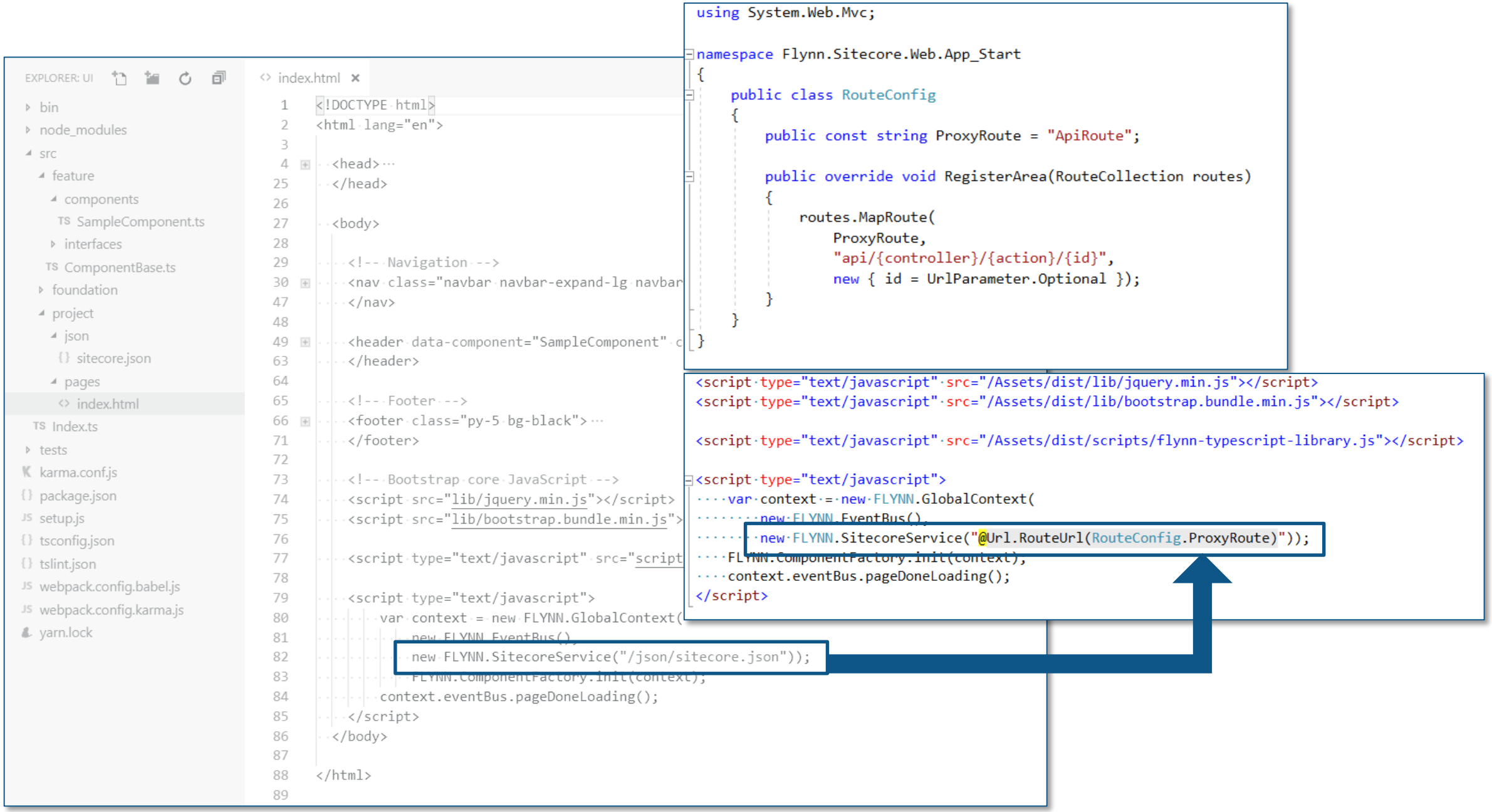
- Encapsulate all Ajax functionality in an injectable helper
- Never hardcode server endpoints

UI component classes

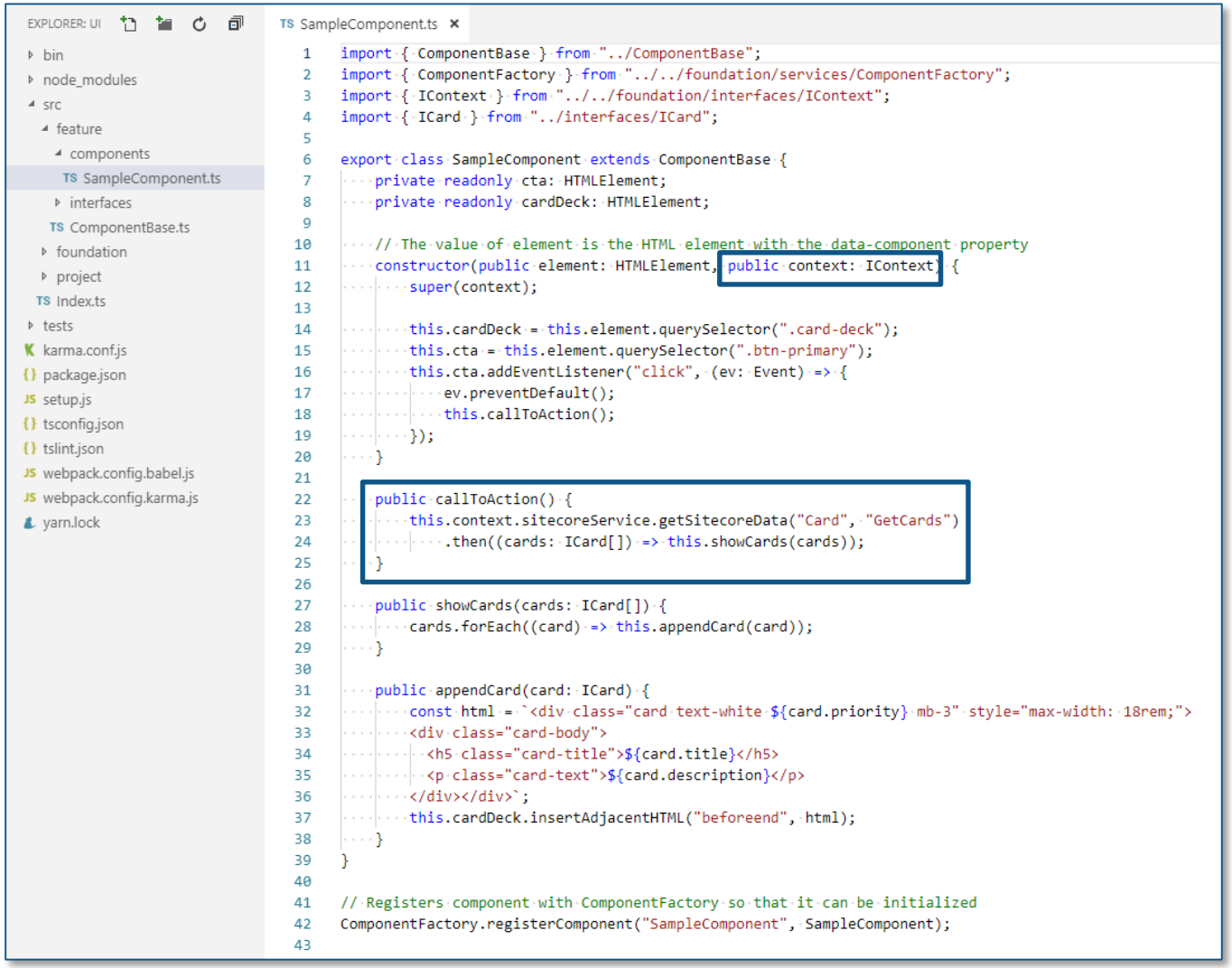
- Accept server interaction helpers as parameters in constructors
- Don’t talk to the server directly

/src/project/pages/index.html





Headless CMS – Sample Component



sitecore SYMPOSIUM ORLANDO

arke

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

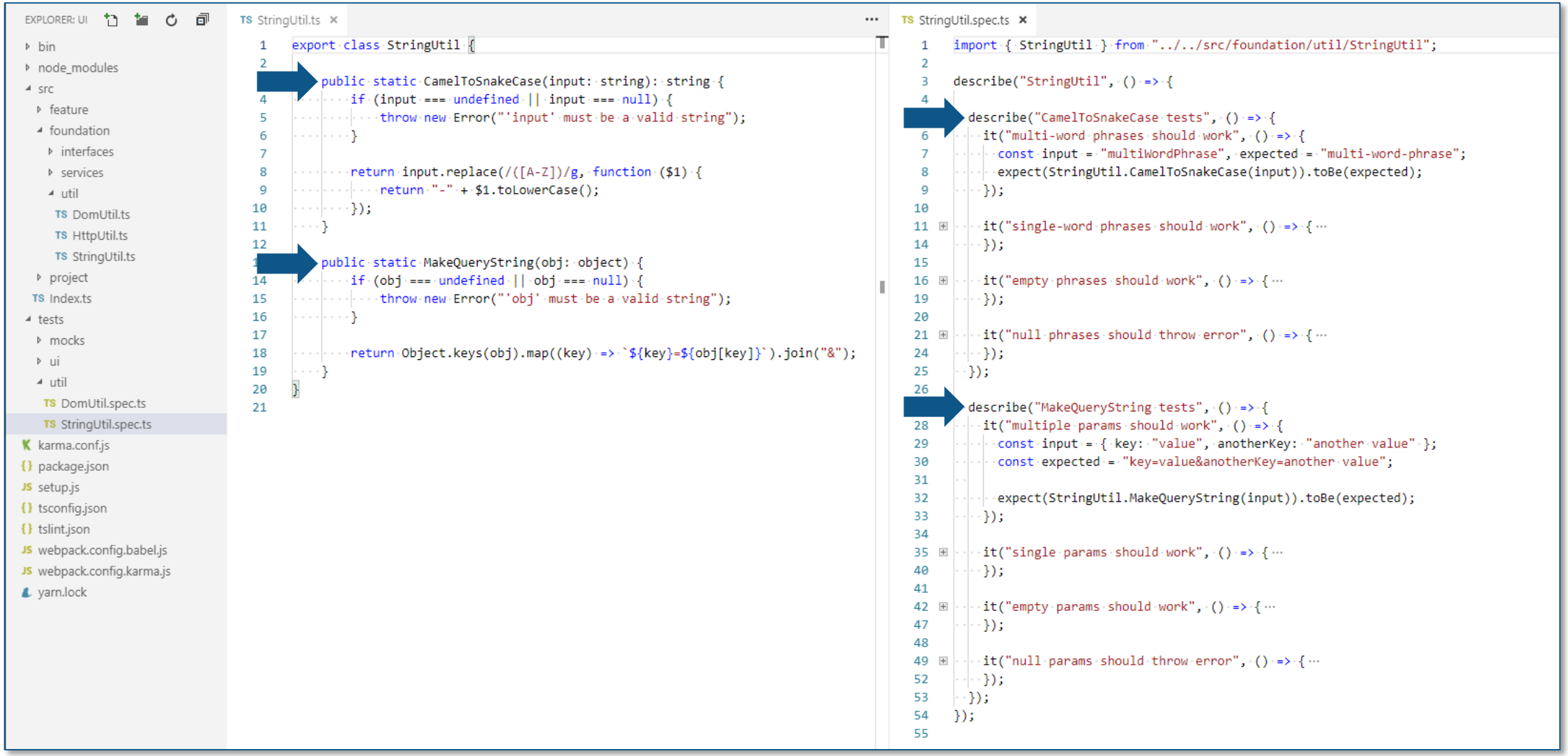
> yarn add jasmine-core --dev

> yarn add @types/jasmine --dev

example.spec.ts

```ts
describe("A suite is just a function", function () {
    var a;

    it("and so is a spec", function () {
        a = true;

        expect(a).toBe(true);
    });
});
```

Jasmine - Implementation



TypeScript Helix – Best practice patterns (continued)

Server interaction

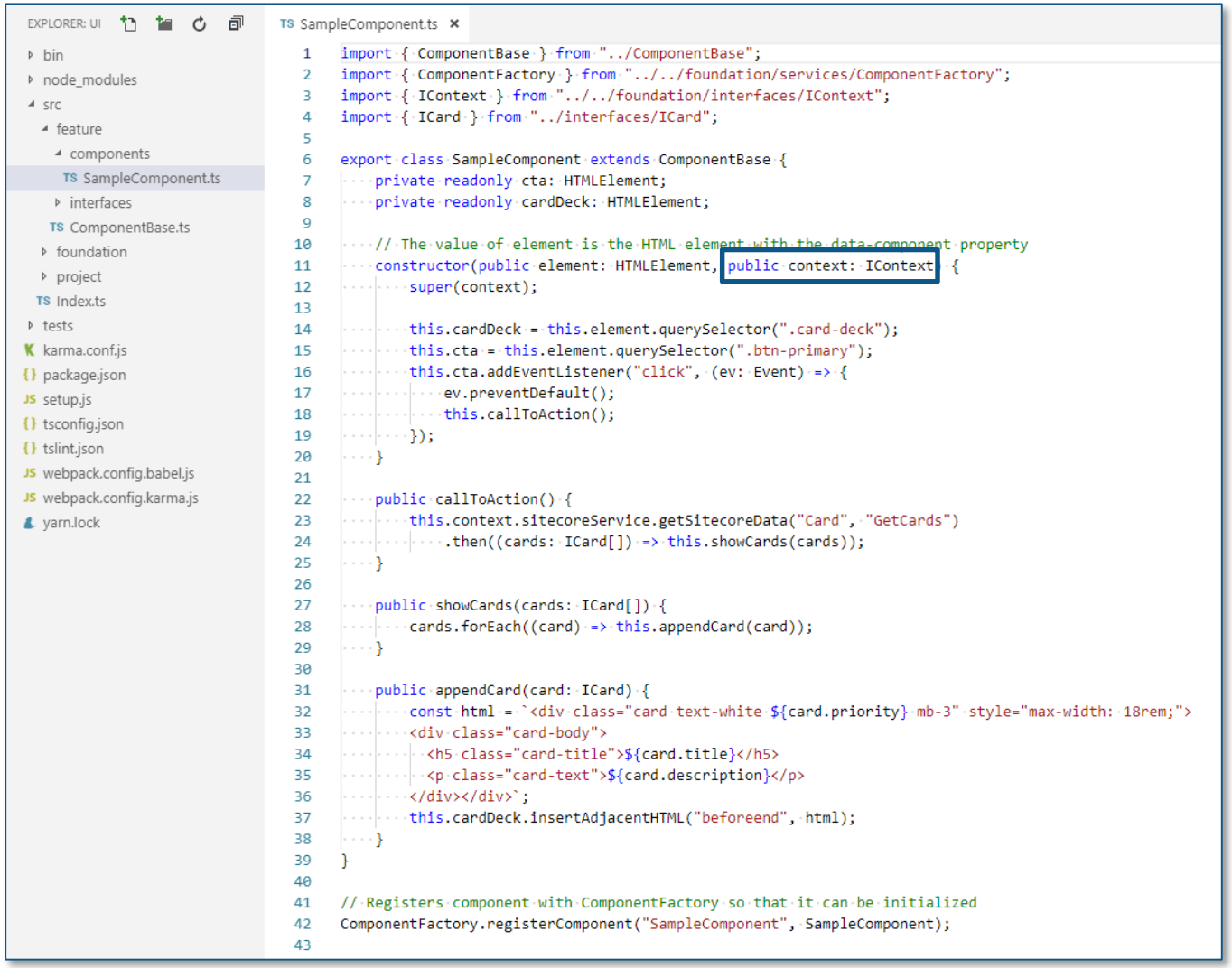
- Encapsulate all Ajax functionality in an injectable helper
- Never hardcode server endpoints

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



TypeScript Helix – Best practice patterns (continued)

Server interaction

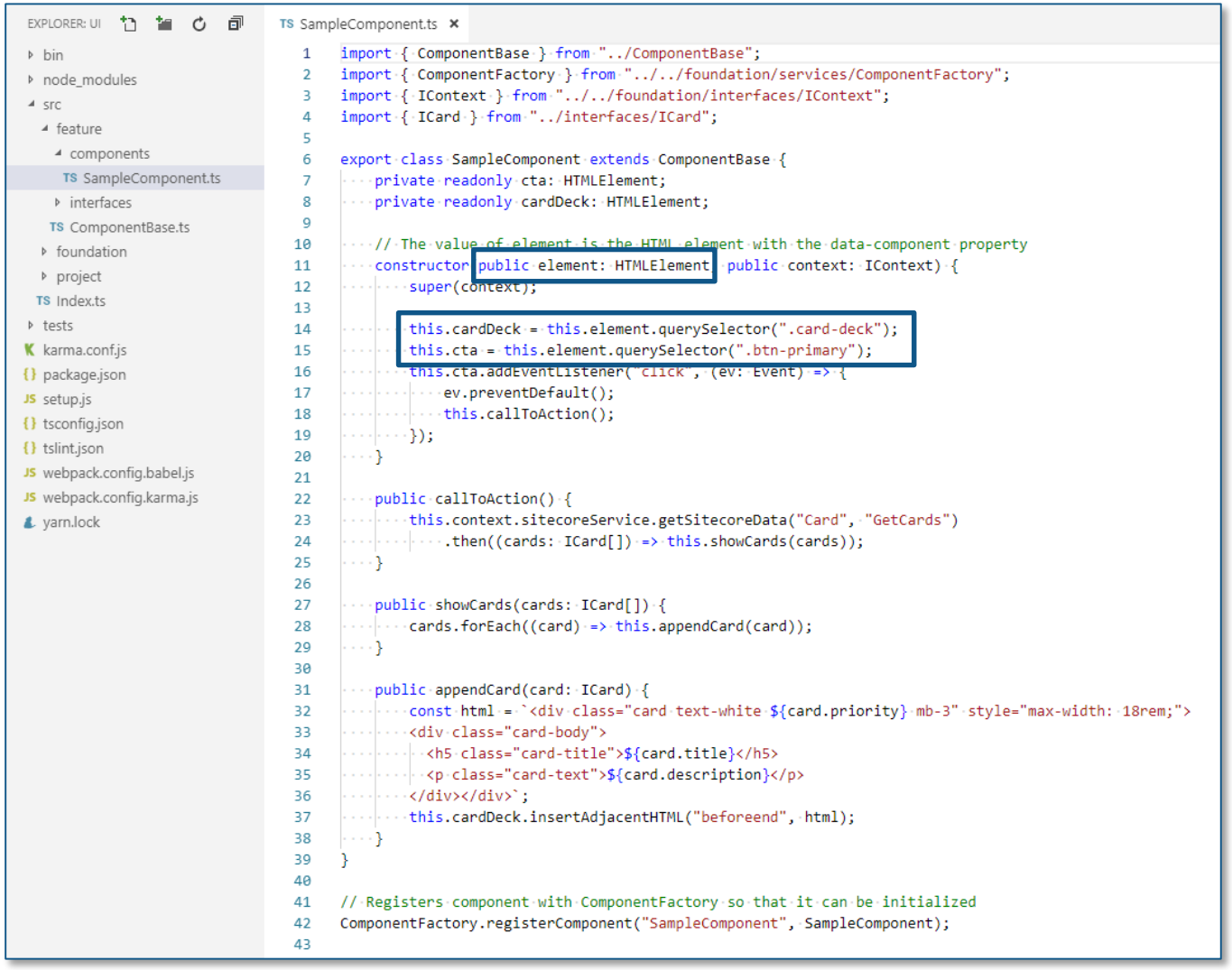
- Encapsulate all Ajax functionality in an injectable helper
- Never hardcode server endpoints

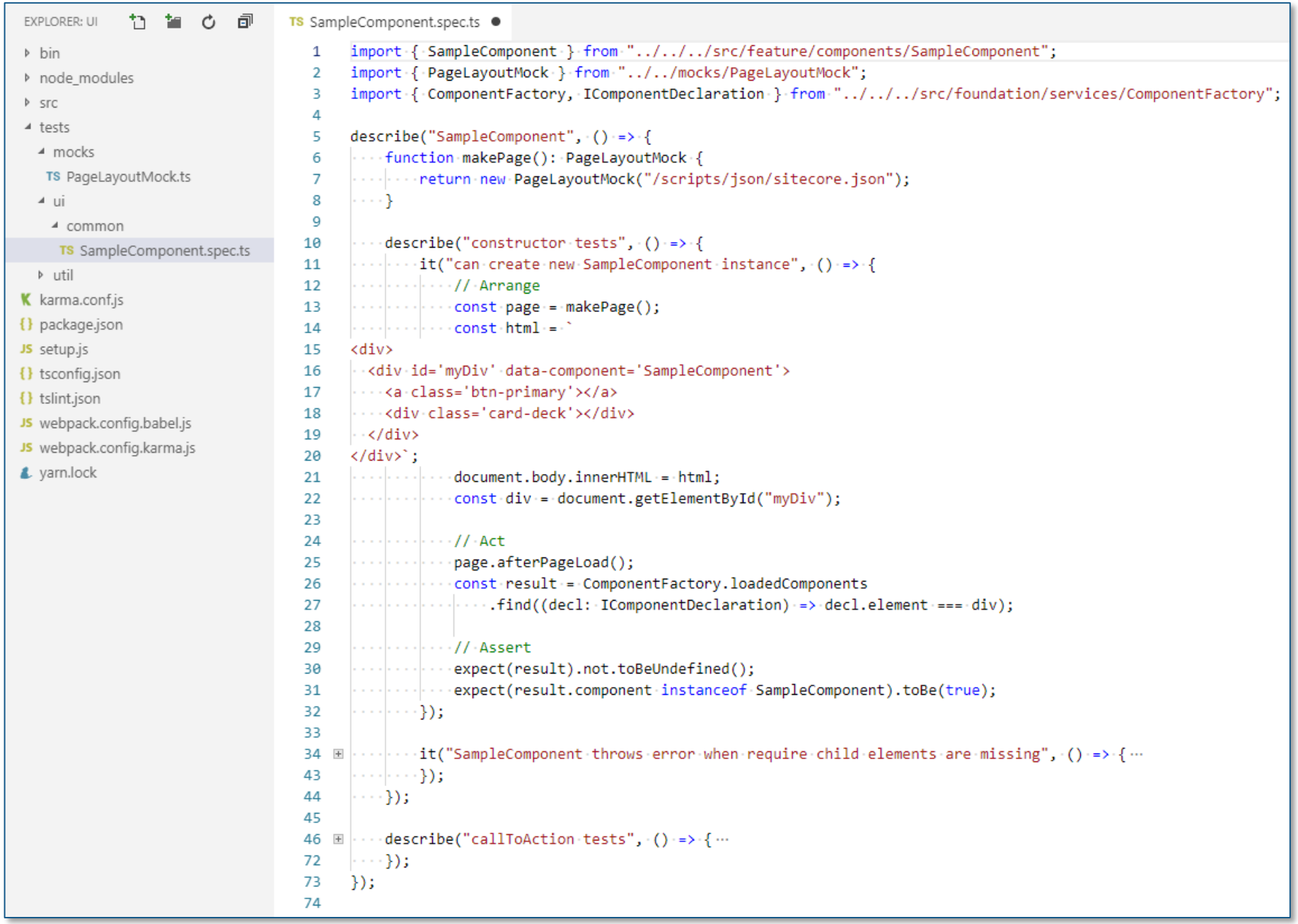
UI component classes

- 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





Karma - Setup

http://karma-runner.github.io

> yarn add karma --dev

> yarn add karma-jasmine -- dev

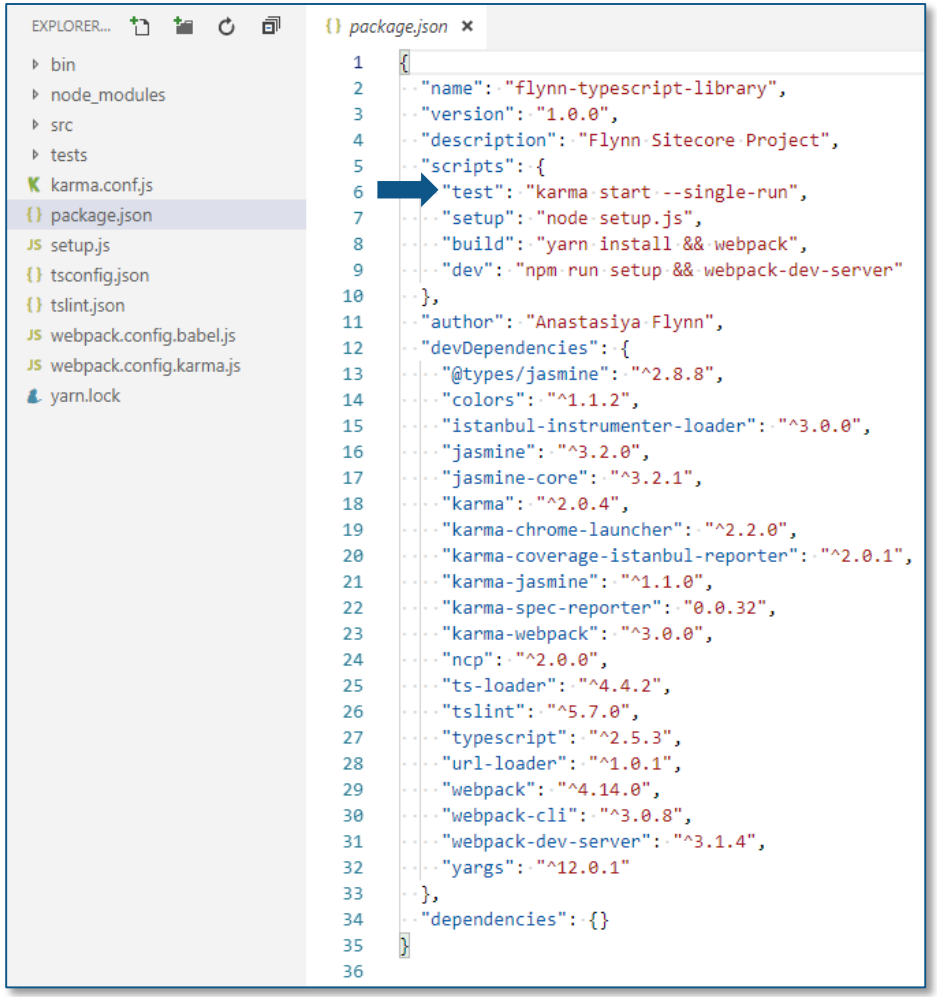
> yarn add karma-chrome-launcher --dev

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

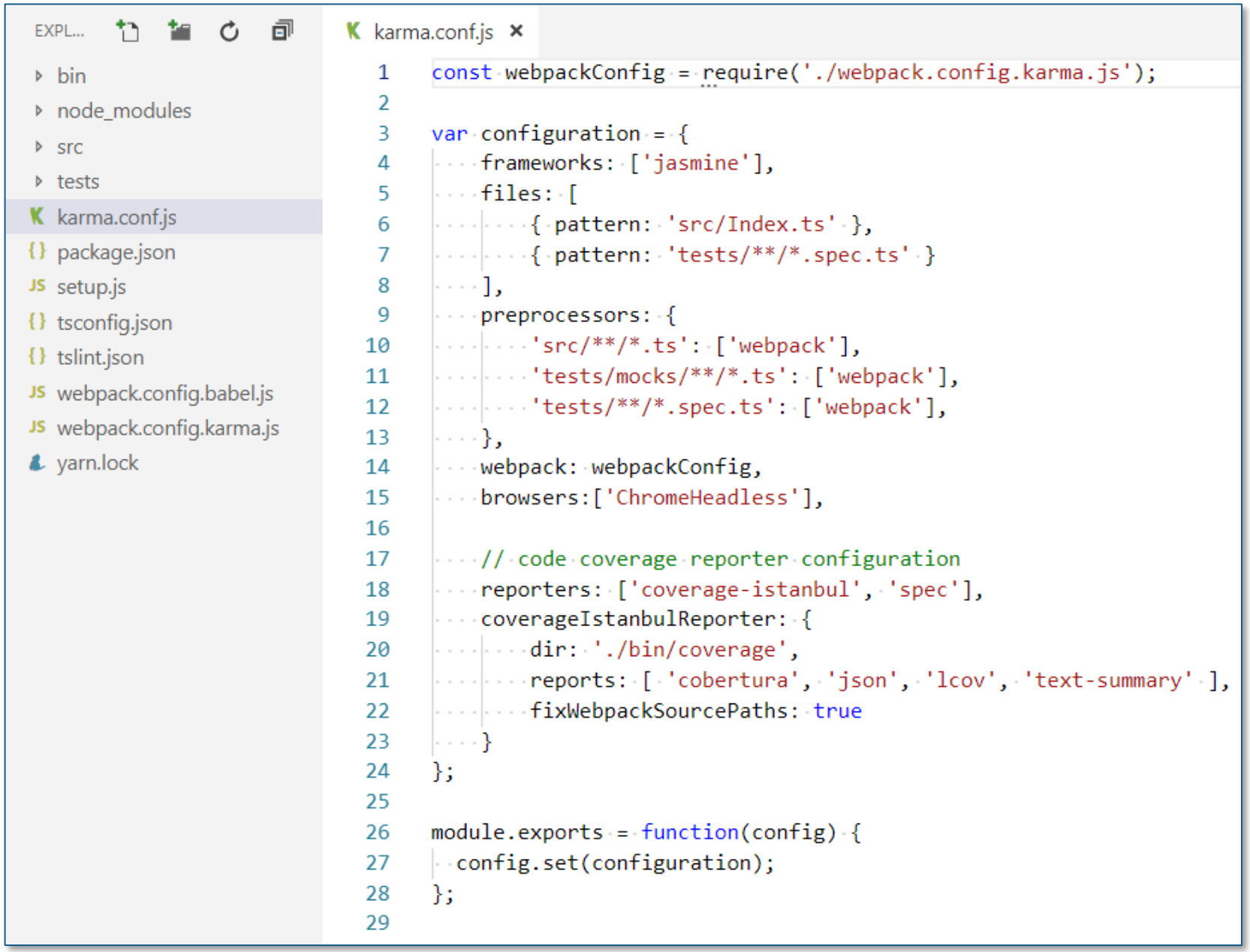
> yarn add karma-spec-reporter --dev

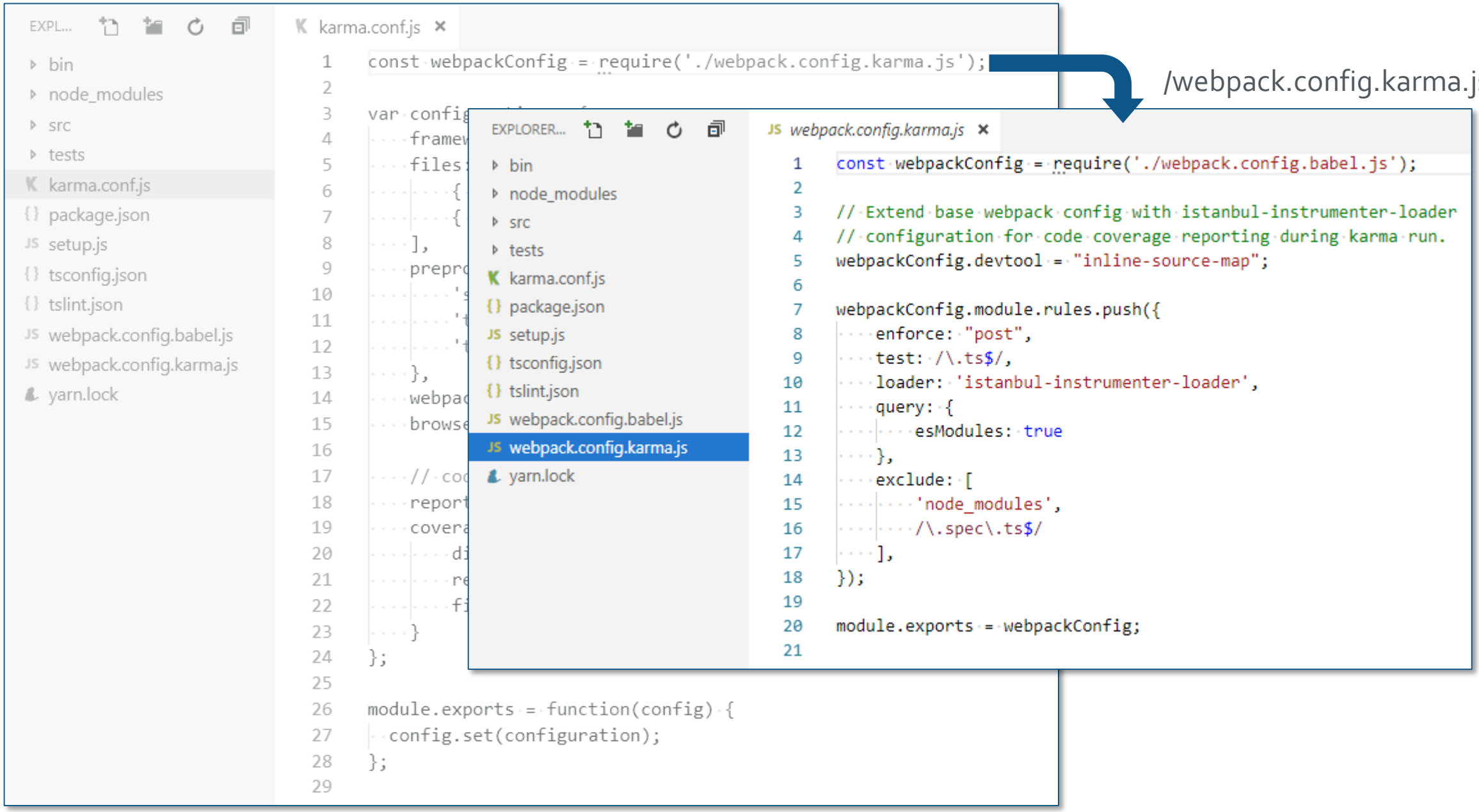
> yarn add karma-webpack --dev

/package.json

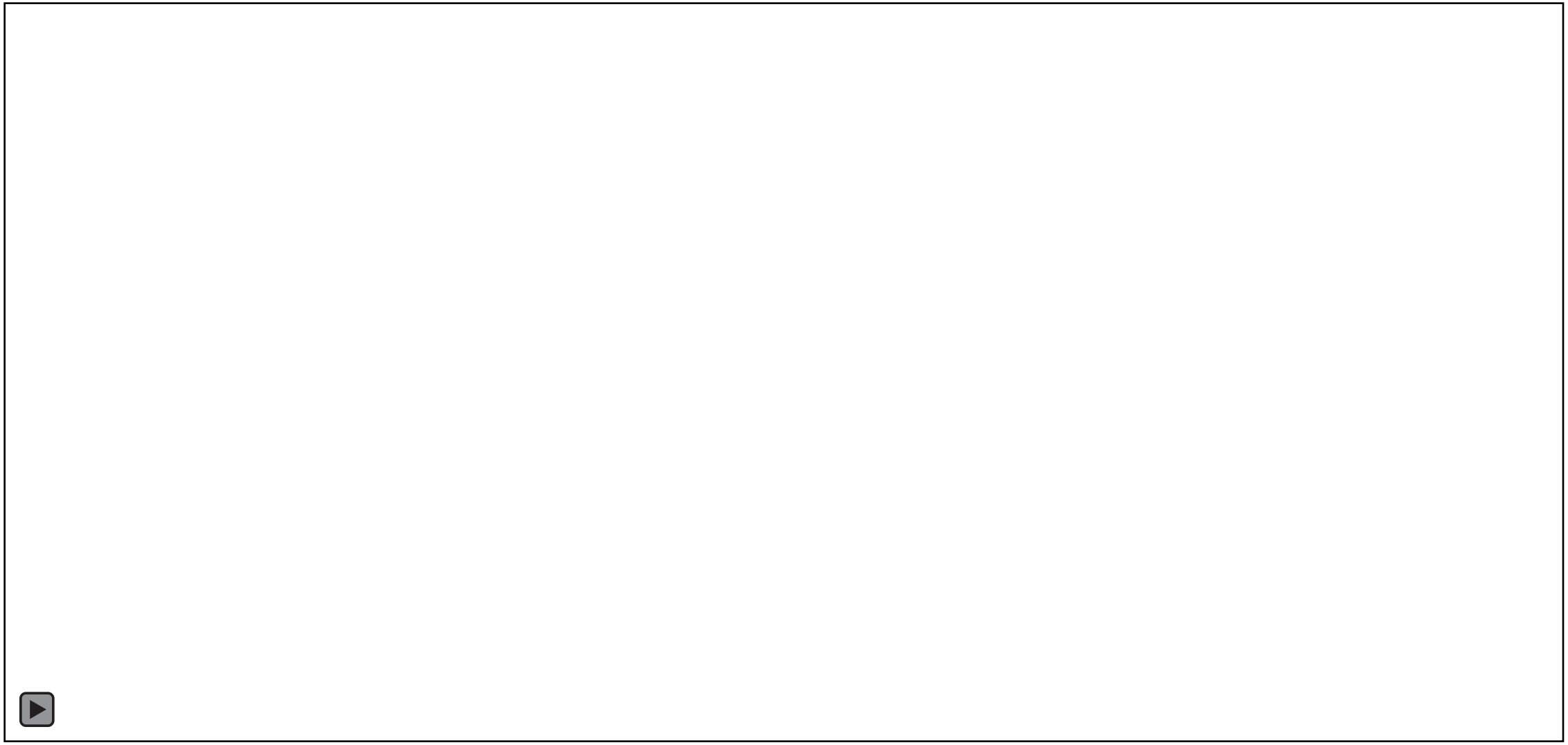


/karma.conf.js





Jasmine/Karma – Demo



Get the code

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

sitecore®
SYMPOSIUM
ORLANDO

sitecore®

SYMPOSIUM ORLANDO

Let's continue the conversation on Twitter **@AnastasiyaFlynn**

#SitecoreSYM