1. create a new project folder (mkdir PROJECT\_NAME & cd PROJECT NAME)

2. run npm init (to initialize a new node project)

3. run npm add -D typescript ts-node nodemon

- ts-node is used to run typescript without compiling

- nodemon is used to run/restart node automatically when files changed

4. run npx tsc --init

5. adjust tsconfig.json to the following

{

"compilerOptions": {

"target": "es6",

"module": "commonjs",

"declaration": true,

"sourceMap": true,

"outDir": "dist",

"rootDir": "src",

"strict": true,

"esModuleInterop": true,

}

}

6. create folder src and index.ts file

1. create a the following folder tree inside your project: src/templates/sample-project.

2. inside the sample-projects folder create a package.json file that contains the following:

{

"name": "sample-project",

"version": "1.0.0",

"license": "MIT"

}

1. #!/usr/bin/env node

This is known as "shebang" this tells node to run the typescript code.

1. run npm add -D @types/node @types/inquirer

2. run npm add inquirer

3. update src/index.ts

import \* as fs from 'fs';

import \* as path from 'path';

import \* as inquirer from 'inquirer';

import chalk from 'chalk';

const CHOICES = fs.readdirSync(path.join(\_\_dirname, 'templates'));

const QUESTIONS = [

{

name: 'template',

type: 'list',

message: 'What project template would you like to use?',

choices: CHOICES

},

{

name: 'name',

type: 'input',

message: 'New project name?'

}];

inquirer.prompt(QUESTIONS)

.then(answers => {

console.log(answers);

});

4. to test update package.json script

"scripts": {

"start": "ts-node src/index.ts"

}

and run npm start

1. update src/index.ts with the following

2. export interface CliOptions {

projectName: string

templateName: string

templatePath: string

tartgetPath: string

}

const CURR\_DIR = process.cwd();

inquirer.prompt(QUESTIONS)

.then(answers => {

const projectChoice = answers['template'];

const projectName = answers['name'];

const templatePath = path.join(\_\_dirname, 'templates', projectChoice);

const tartgetPath = path.join(CURR\_DIR, projectName);

const options: CliOptions = {

projectName,

templateName: projectChoice,

templatePath,

tartgetPath

}

console.log(options);

});

function createProject(projectPath: string) {

if (fs.existsSync(projectPath)) {

console.log(chalk.red(`Folder ${projectPath} exists. Delete or use another name.`));

return false;

}

fs.mkdirSync(projectPath);

return true;

}

inquirer.prompt(QUESTIONS)

.then(answers => {

....

if (!createProject(tartgetPath)) {

return;

}

});

1. Add to src/index.ts

2. // list of file/folder that should not be copied

const SKIP\_FILES = ['node\_modules', '.template.json'];

function createDirectoryContents(templatePath: string, projectName: string) {

// read all files/folders (1 level) from template folder

const filesToCreate = fs.readdirSync(templatePath);

// loop each file/folder

filesToCreate.forEach(file => {

const origFilePath = path.join(templatePath, file);

// get stats about the current file

const stats = fs.statSync(origFilePath);

// skip files that should not be copied

if (SKIP\_FILES.indexOf(file) > -1) return;

if (stats.isFile()) {

// read file content and transform it using template engine

let contents = fs.readFileSync(origFilePath, 'utf8');

// write file to destination folder

const writePath = path.join(CURR\_DIR, projectName, file);

fs.writeFileSync(writePath, contents, 'utf8');

} else if (stats.isDirectory()) {

// create folder in destination folder

fs.mkdirSync(path.join(CURR\_DIR, projectName, file));

// copy files/folder inside current folder recursively

createDirectoryContents(path.join(templatePath, file),

path.join(projectName, file));

}

});

}

....

if (!createProject(tartgetPath)) {

return;

}

createDirectoryContents(templatePath, projectName);

....

1. Run npm add -D shx

2. Add the following build script to package.json

"build": "tsc && shx rm -rf dist/templates && shx cp -r src/templates dist"

3. npm run build

4. Add bin to package.json

"bin": {

"template-generator": "./dist/index.js"

}

5. Register "template-generator" as a command line interface

run npm link

1. update template "src/templates/sample-project/package.json" with a placholder name

{

"name": "<%= projectName %>",

"version": "1.0.0",

....

}

2. npm add ejs

add -D @types/ejs

3. update src/utils/template.ts to render template under utils

import \* as ejs from 'ejs';

export interface TemplateData {

projectName: string

}

export function render(content: string, data: TemplateData) {

return ejs.render(content, data);

}

4. Add code to transform the content inside "src/index.ts" function "createDirectoryContents"

if (stats.isFile()) {

// read file content and transform it using template engine

let contents = fs.readFileSync(origFilePath, 'utf8');

contents = template.render(contents, { projectName });

}

5. run npm build and then generate-template to test that the new project name is inserted in the "<%= projectName %>" placeholder.

#!/usr/bin/env node

import \* as fs from 'fs';

import \* as path from 'path';

import \* as inquirer from 'inquirer';

import chalk from 'chalk';

import \* as template from './utils/template';

import \* as shell from 'shelljs';

const CHOICES = fs.readdirSync(path.join(\_\_dirname, 'templates'));

const QUESTIONS = [

{

name: 'template',

type: 'list',

message: 'What template would you like to use?',

choices: CHOICES

},

{

name: 'name',

type: 'input',

message: 'Please input a new project name:'

}];

export interface CliOptions {

projectName: string

templateName: string

templatePath: string

tartgetPath: string

}

const CURR\_DIR = process.cwd();

inquirer.prompt(QUESTIONS).then(answers => {

const projectChoice = answers['template'];

const projectName = answers['name'];

//@ts-ignore

const templatePath = path.join(\_\_dirname, 'templates', projectChoice);

//@ts-ignore

const tartgetPath = path.join(CURR\_DIR, projectName);

const options: CliOptions = {

//@ts-ignore

projectName,

//@ts-ignore

templateName: projectChoice,

templatePath,

tartgetPath

}

if (!createProject(tartgetPath)) {

return;

}

//@ts-ignore

createDirectoryContents(templatePath, projectName);

postProcess(options);

});

function createProject(projectPath: string) {

if (fs.existsSync(projectPath)) {

console.log(chalk.red(`Folder ${projectPath} exists. Delete or use another name.`));

return false;

}

fs.mkdirSync(projectPath);

return true;

}

const SKIP\_FILES = ['node\_modules', '.template.json'];

function createDirectoryContents(templatePath: string, projectName: string) {

// read all files/folders (1 level) from template folder

const filesToCreate = fs.readdirSync(templatePath);

// loop each file/folder

filesToCreate.forEach(file => {

const origFilePath = path.join(templatePath, file);

// get stats about the current file

const stats = fs.statSync(origFilePath);

// skip files that should not be copied

if (SKIP\_FILES.indexOf(file) > -1) return;

if (stats.isFile()) {

// read file content and transform it using template engine

let contents = fs.readFileSync(origFilePath, 'utf8');

contents = template.render(contents, { projectName });

// write file to destination folder

const writePath = path.join(CURR\_DIR, projectName, file);

fs.writeFileSync(writePath, contents, 'utf8');

} else if (stats.isDirectory()) {

// create folder in destination folder

fs.mkdirSync(path.join(CURR\_DIR, projectName, file));

// copy files/folder inside current folder recursively

createDirectoryContents(path.join(templatePath, file), path.join(projectName, file));

}

});

}

function postProcess(options: CliOptions) {

const isNode = fs.existsSync(path.join(options.templatePath, 'package.json'));

if (isNode) {

shell.cd(options.tartgetPath);

const result = shell.exec('npm install');

if (result.code !== 0) {

return false;

}

}

return true;