generator-makrina Documentation

Release

George Karakostas

Mar 18, 2017

Contents

Generate MEAN boilerplate. Featuring model and route CRUD endpoint sub-generators and an admin interface.

Makrina generates a starting project with the following stack and features.

CHAPTER 1

Technology Stack

- Mongo DB
- Express.js
- Angular JS
- Node.js

CHAPTER 2

Features

- EJS templates
- An admin interface based on Gentelella *Admin template*
- Latest *Angular* v1.6 scaffolding created by sub-generators
- *Mongoose* models created by sub-generators
- API route endpoints for the models
- An extended gulpfile for building and testing
- Ready for *Tests* using mocha, chai, sinon, karma, protractor.
- Generated code should have already 100% coverage or nearly.
- *CSRF* protection.

Installation

In this section we describe how to install and use Makrina.

First, install Yeoman and generator-makrina using npm. We assume you have pre-installed node.js.

```
npm install -g yo
npm install -g generator-makrina
```

Then generate your new project:

```
yo makrina
```

Main app generator

Build the MEAN project scaffolding.

Execute

The main generator is invoked using:

```
yo makrina
```

Description

The generator creates a new project featuring:

- A default `package.json` file with the appropriate description and dependencies for the following.
- Default meta documents: `README`, private `LICENCE`, `CONTRIBUTING`, `CHANGELOG`.
- Default `gitignore` and `editorconfig` files
- Standard express.js boilerplate: `app.js`, `bin/`, `public/`, `routes/`, `views/`.
- A default `gulpfile` mainly for managing front-end files.
- Standard karma and protractor (e2e) configuration.
- A `services` folder with most configuration scripts: mongoose, session, i18n, email.
- Default routes & views for index, admin interface and contact form.
- A `.yo-rc.json` local yeoman configuration file with the options used to generate the project.

It then calls the following sub-generators for an angular application with basic REST CRUD functionality for a single object. Repeat calling the sub-generators on their own to add more objects.

Sub-generators

The main generator also invokes the following sub-generators by order:

- *Angular app sub-generator*
- *Angular core service sub-generator*
- *Angular list component sub-generator*
- *Angular detail component sub-generator*
- *Mongoose model and API sub-generator*

Prompts

The generator asks the following input:

- **Project name**: the generated project name (no spaces). Used in: `package.json, README.md, newrelic.js, services/mongoose.js`.
- **Verbose name**: a verbose project name, spaces allowed. Used in: `newrelic.js, routes/index.js, routes/admin.js, routes/api/contact.js`.
- **Description**: project description. Used in: `package.json, README.md, routes/index.js, views/index.ejs`.
- **Git repository URL**. Used in `package.json`.
- **Author**. Used in `package.json, LICENSE, CONTRIBUTING.md, views/index.ejs`.

- **Organization**: Organization name for author. Used in: `views/index.ejs, views/login.ejs, views/admin.ejs`.
- **Organization URL**. Used in `views/login.ejs, views/admin.ejs`.
- **Deploy host**: optional for `npm deploy` command. Used in: `package.json`.
- **New Relic license key**. Used in `newrelic.js`.

There are also additional prompts from the sub-generators.

Angular app sub-generator

Generate an Angular application.

Execute

The sub-generator is invoked with:

```
yo makrina:angular-app
```

The sub-generator is also invoked by the main generator.

Description

It creates a standard Angular scaffolding:

- A main app module
- An angular config file
- An additional jquery file with app-related jquery functions
- An additional SASS file with app-related styling
- An animations SASS file with ng-animate styling
- An additional core sub-module for facilitating common core functions (factories, filters) in the future.

Usually this intended to allow building additional functionality with components, services etc. that can be also built using the other angular sub-generators.

Prompts

The generator asks the following input:

- **Angular app short name**: the short name for the app, eg. `admin` or `app`. Used in: destination prefix for the files.
- **Angular app name**: the full name for the app, eg `myProjectAdminApp`. Used in: `_angular-app-name_.module.js, views/admin.ejs`.
- **Angular app path**: the relative destination path for the app, eg. `public/javascripts`. Used in: destination path for the files.

Angular core service sub-generator

Generate an Angular service module using `$resource`.

Execute

The sub-generator is invoked with:

```
yo makrina:angular-core-service
```

The sub-generator is also invoked by the main generator.

Description

It creates:

- An additional core sub-module to facilitate an object factory service (REST communication with server).
- The actual service file
- A unit tests file

Prompts

The generator asks the following input:

Angular app

The angular app prompts that have been used in the respective generator.

Object

The object is usually a single entity, such as *node*, *page*, etc. It has its own mongoose model, api endpoint and angular service and component.

- Object name (recommended camelCase).
- Object title (recommended PascalCase).
- Object API URL and directory name (recommended kebab-case).

Angular list component sub-generator

Generate an Angular component for presenting list data.

Execute

The sub-generator is invoked with:

```
yo makrina:angular-component-list
```

The sub-generator is also invoked by the main generator.

Description

It creates:

- An additional sub-module to display an object list based on angular core service.
- A default angular 1.5.4 component for the list
- The angular template for the list
- A unit tests file
- An e2e scenarios file

Prompts

The generator asks the following input:

Angular app

The angular app prompts that have been used in the respective generator.

Object

The angular core service prompts that have been used in the respective generator.

Angular detail component sub-generator

Generate an Angular component for presenting detail data.

Execute

The sub-generator is invoked with:

```
yo makrina:angular-component-detail
```

The sub-generator is also invoked by the main generator.

Description

It creates:

- An additional sub-module to display an object form.
- A default angular 1.5.4 component to allow get, post, delete
- The angular template for the form

- A unit tests file
- An e2e scenarios file

Prompts

The generator asks the following input:

Angular app

The angular app prompts that have been used in the respective generator.

Object

The angular core service prompts that have been used in the respective generator.

Angular form controller sub-generator

Generate a simple module with a controller to handle a form submit.

Execute

The sub-generator is invoked with:

```
yo makrina:angular-component-detail
```

Description

A small controller to post a form such as a contact message and handle the form submit button status. Not called by main app generator.

Prompts

The generator asks the following input:

The angular app prompts that have been used in the respective generator.

Mongoose model and API sub-generator

Generate a mongoose document model with API endpoints.

Execute

The sub-generator is invoked with:

```
yo makrina:model
```

The sub-generator is also invoked by the main generator.

Description

It creates:

- A mongoose document model
- An express route with API endpoints and basic CRUD

Prompts

The generator asks the object details, which are similar to the angular core service prompts that have been used in the respective generator.

Development Process

This page briefly describes a typical MEAN app development process.

Process

- 0. Environment setup
  - Prepare OS
    - Install node.js
    - *Install Mongodb*
    - *Install global packages*
    - Yeoman and makrina *Installation*
  - Start project
    - Use makrina *Main app generator*: `yo makrina`
    - Review generated files
    - Review meta documents (`README, LICENSE, CONTRIBUTING, CHANGELOG`)
  - Configure packages
    - Review `package.json`
    - Review `gulpfile.js`
    - Start with `npm start` or `node bin/www`
- 1. *Angular* and front-end
  - 0. Bootstrapping

    * Review `views/index.ejs`
    * Review *Admin template*
    * Review and add js files in gulp
- 1. Static template
    * Add static files
    * Style up
    * Review and add sass/css files in gulp
- 2. Angular setup
    * Review `public/javascripts/app` folder
    * Review `karma.conf.js` for *Tests*
- 3. Components
    * Review list component `object-list.component.js`
    * Review list unit test `object-list.component.spec.js`
- 4. Module and file Organization
    * Review `app.module.js` and inject any additional module dependency
    * Review list template `object-list.template.html`
- 5. Filters
    * Add custom filters to list
- 6. Extend e2e tests
    * Review `protractor.conf.js`
    * Review e2e tests `e2e-tests/app.scenarios.js`
- 7. XHR
- 8. Templating list
    * Expand on the list template
- 9. Angular Routing
    * Template: review `ng-view`
    * Dependency: using `angular-route`. Review `app.module.js`
    * Configure in `app.config.js`
    * Review detail component `object-detail/object-detail.component.js`
    * Review detail module `object-detail.module.js` and inject in `app.module.js`
    * Extend e2e tests
- 10. Templating detail
    * Expand on the detail template
    * Review `node-detail.component.spec.js`
    * Extend e2e tests
- 11. Core module

    * Review core module `core/core.module.js` and inject in `app.module.js`
    * Add custom filters in `core/filter_name/filter_name.filter.js` (optional)
    * Add unit tests in `filter_name.filter.spec.js`
  - 12. Event handlers
  - 13. REST and custom services factory
    * Dependency: using `angular-resource`
    * Review module `core/object/object.module.js` with dependency to `ngResource`
    * Review service `object.service.js`
    * Inject `core.object` to `core.module.js`, `object-list.component.js`, `object-detail.component.js`
    * Review unit test `object.service.spec.js`
  - 14. Animations
    * Dependency: using `angular-animate` in `app.module.js`
    * Review `app.animations.sass`
    * Review template: jquery loaded in head, relevant classes to `ng-view`
    * Extend animations with js
- 2. Node and back-end
  - Models: *Mongoose*
    * Review `services/mongoose.js`
    * Review and create `models/`
    * Review express-session in `services/session-config.js`
    * Add *CSRF* to non-Angular forms
    * Review `newrelic.js`.
  - Routes: Express
    * Review i18n urls in `services/i18n-config.js` and `routes/index.js`
    * Connect Angular service to routes: review `routes/api/`.
  - Breadcrumb

Environment setup

Install Mongodb

- Latest Mongodb (at the time of writing) is v3.2.
- Ubuntu 16.04 comes with v2.6.
- Ubuntu 14.04 comes with v2.4 (see how to upgrade Mongo).
- Version 2.6 will be used throughout due to availability on hosts.

Install global packages

```
~/.nvm/v4.4.7/lib
- gulp-cli@1.2.1
- karma-cli@1.0.1
- npm@3.10.6
```

Install with:

```
nvm install 4
npm update -g npm
npm install -g gulp-cli karma-cli
```

Angular

The outline follows the Angular tutorial. Gulpfile is configured to concatenate the angular app.

Sanitize

`angular-sanitize` can be used to present a sanitized scope variable with `ng-bind-html`.

To further sanitize HTML stored in db, the module mongoose-html can be used. The project is abandoned and has unmet peer dependencies so don't install it directly, the relevant piece of code is very small and can be used directly.

It is based on sanitize-html which can be used at a lower level. It is not possible to easily use it though with custom validators as they do not return the value but only a boolean, and also they are yet harder to use within objects.

Tests

Unit tests

All unit tests can be invoked from `npm test`, which essentially invokes `gulp test`:

```
npm test
```

Unit tests involve front-end tests with karma and back-end tests with mocha. Both provide coverage report using istanbul. If browser is not starting, export `CHROME_BIN`. For fish see this fish configuration gist.

End-to-end tests

```
npm start
npm run protractor
```

Server needs to have started before. Protractor can only run in local. Protractor is tested only for Chrome due to #3044:

```
npm start
node_modules/protractor/bin/webdriver-manager update
node_modules/protractor/bin/protractor e2e-tests/protractor.conf.js
```

Extend tests

The generated unit tests cover 100% or nearly of the generated code. Nevertheless, the following files have been excluded out from testing and coverage:

- The object api endpoint route `routes/api/object.js`. It needs further development and custom test.
- The mongoose service `services/mongoose.js`. This will get indirectly covered by the above.
- The object angular components need further development.

Consequently, after developing the above, add the relevant files in the `gulpfile.js` `js_cover` section and `karma.conf.js` `files` and `preprocessors` section.

Admin template

The Gentelella admin theme is adapted into `views/admin.ejs`. The adaptations include:

- Most static files are loaded from minified output from gulpfile.
- Context is provided from `routes/admin.js`.
- The main view is adopted for Angular route using `ng-view`.
- Also, a `views/login.ejs` template has been added.

Lastly, the ng-gentelella package is used.

Mongoose

The files `servives/mongoose.js` and `models/object.js` need to be reviewed.

Models are used in routes by requiring:

```
var mongoose = require('mongoose');
var Object = mongoose.model('Object');
```

Mongoose quick reference

- Mongoose Quickstart
- Mongoose Guide
- Schema definition and types
- Updating nested objects
- Field update operators

Sanitize and injection

Mongodb injection is very much possible with mongo queries, especially when a where criterion is provided from a public form or url. The module mongo-sanitize can be easily used to sanitize `req.body` and `req.query` from mongo `$` operators.

CSRF

CSRF protection is ensured by the Csurf package. Angular forms respect CSRF due to the `XSRF-TOKEN` cookie that we make in `app.js`. All other forms require manual append of CSRF:

- In router context: `context.csrf = req.csrfToken();`
- In forms: `<input type="hidden" name="_csrf" value="<%= csrf %>">`

The responses are automatically checked by the module and 403 is returned appropriately. This is tested in mocha too.

Useful links

Generic MEAN REST tutorials

- https://thinkster.io/mean-stack-tutorial
- http://adrianmejia.com/blog/2014/10/01/creating-a-restful-api-tutorial-with-nodejs-and-mongodb/
- https://scotch.io/tutorials/using-mongoosejs-in-node-js-and-mongodb-applications

Testing

- http://developers.redhat.com/blog/2016/03/15/test-driven-development-for-building-apis-in-node-js-and-express/
- https://codeforgeek.com/2015/07/unit-testing-nodejs-application-using-mocha/

Updates

Most generated code gets signed by the Makrina version and date in a comment at the top of file. All changes are documented in `CHANGELOG.md`. Any change that may affect existing generated files is marked with a warning sign. Usually changes in existing files that may have been generated in existing projects exist in minor releases (version numbering: major, minor, revision).

If you feel like using a latest feature or change, you can manually update the relevant files, and append the Makrina version in the comment for future reference:

```
* Updated to yeoman generator-makrina <%= version %> on <%= date %>.
```

Or re-run the generator by carefully selecting which files to replace when asked.