test Documentation

Release 0.1.6

arunnattarayan

May 24, 2019

Contents

Boilerplate and tooling for authoring data API backends with Node.js, JWT and MongoDB. It is best suited for developing a es6 API endpoint as a standalone (micro)service (demo), backing up web front-ends and/or mobile apps. This Generator will help to us create a express api application skeleton along with these keys features

- Code with ES6 Style
- components Style coding
- Eslint Standard
- JWT authentication
- Test cases with Mocha
- Travis

CHAPTER 1

Requirements

- Node 10.15.1+
- MongoDB 3.4+
- yeoman-generator 3.2.0+

CHAPTER 2

Report

- Report any issues or feature enhancements in our tracker

CHAPTER 3

Contents

3.1 Quickstart

3.1.1 Install

Steps to create a node application skeleton:

```
$ npm install -g yo
$ npm install -g generator-node-api-boilerplate
$ yo node-api-boilerplate
```

Now the app was created and start running in your system

3.1.2 Usage

start the express server:

```
$ npm start
```

Test The unit testcases:

```
$ npm test
```

find Lint issues in source code:

```
$ npm run lint
```

find Lint issues in test cases:

```
$ npm run lint:test
```

Note: `npm run lint:test:fix` and `npm run lint:fix` used to fix the lint errors

3.2 Project Structure

Your project will look like this:

```
├── package.json
├── public
│   ├── logs
│   │   └── access.log
│   ├── docs
│   │    └── swagger.json
│   |
├── README.md
├── src
│   ├── app
│   │   └── auth
│   │       ├── AuthRoute.js
│   │       ├── UserController.js
│   │       ├── UserMiddlerware.js
│   │       ├── UserModel.js
│   │       ├── UserRoute.js
│   │       └── UserService.js
│   ├── config
│   │   ├── db
│   │   │   └── connection.js
│   │   ├── environments
│   │   │   ├── config.js
│   │   │   ├── development.js
│   │   │   ├── index.js
│   │   │   ├── production.js
│   │   │   └── test.js
│   │   ├── express-middleware.js
│   │   ├── logger.js
│   │   └── route
│   │       ├── route.index.js
│   │       └── routes.js
│   └── server.js
├── test
│   ├── index.js
│   ├── shared.spec.js
│   └── users
│       └── users.spec.js
```

3.2.1 src

src is the source code directory:

```
├── src
│   ├── app
│   │
│   ├── config
```

(continues on next page)

(continued from previous page)

```
│   └── server.js
```

- `app` contains all needed components
- `config` has all the configurations like `DB`, `ENV`
- `server.js` is the app boostrap file

3.2.2 Structure your solution by components

Instead of `MVC` pattern we recommended components based pattern:

```
├── src
│   ├── app
│   │   └── auth
│   │       ├── AuthRoute.js
│   │       ├── UserController.js
│   │       ├── UserMiddlerware.js
│   │       ├── UserModel.js
│   │       ├── UserRoute.js
│   │       └── UserService.js
```

- `auth` is the component name and that contains `route`, `Controller`, `middleware`, `model`, and the `service` files.

3.2.3 config

- Config folder structure:

```
├── src
|   ├── config
│   │   ├── db
│   │   │   └── connection.js
│   │   ├── environments
│   │   ├── express-middleware.js
│   │   ├── logger.js
│   │   └── route
```

- `DB` holds the information of MongoDB connection
- `environments` use to define *ENV* variables. We can set different values for different ENV.
- `express-middleware` deines express middleware need to run this app
- `logger.js` logger details are defined here
- `route` creating interface between our components (indeside *src/app*) and application.

```
├── src
|   ├── config
│   │   ├── environments
│   │   │   ├── config.js
│   │   │   ├── development.js
│   │   │   ├── index.js
```

(continues on next page)

(continued from previous page)

```
|   |   |   ├── production.js
|   |   |   └── test.js
```

config.js

It holds common ENV variables across all environments. `development`, `production`, and the‘‘test‘‘ are extends this file.

route

```
├── src
|   ├── config
|   |   └── route
|   |   |       ├── route.index.js
|   |   |       └── routes.js
```

Need to link our components in routes.js like this

```
import UserRoute from '../../app/auth/UserRoute';

const Routes = [
{
    url: 'users',
    route: UserRoute,
    gaurd: false
}
];

export default Routes;
```

- `url` group name of api endpoint
- `route` component route object
- `gaurd` (optional) if you want to skip JWT verification set false. By default it sets true