

# Fractal—Nodejs app structure



Deepak Gupta

Jun 30, 2018 · 6 min read ★



Photo by [Tom Grimbert](#) on [Unsplash](#)

This article about structuring the backend for REST API's which can be extend to GraphQL.

*Note: The article give an overview of structuring and [here](#) is the code base for detailed understanding.*

• • •

After working with many backend projects, i realized that structuring the app is equally important as the choice of tech stack and fractal app structure is always my choice.

Fractal pattern convey that similar patterns recur progressively and the same thought process is applied to the structuring of codebase i.e **All units repeat themselves.**

*Note: To code a fractal pattern and see it visually, check [here](#) and don't forget to follow me on [Github](#).*

Now, lets start with the crux,

## App Root Structure

The root folder always remain same in any case of API's structuring.

```
▸ app
▸ docs
▸ lib
▸ migrations
▸ node_modules
▸ tests
📄 .babelrc
🔧 .eslintrc
📁 .gitignore
📄 config.js
📄 config.sample.js
📄 migrate.js
📄 package.json
```

So, At root level we have -:

1. **app** is where our main app code goes and will discuss in depth under app structure below.
2. **docs**(optional) is for documentation purpose.
3. **lib**(optional) is where the compiled app code goes which is used for production env.
4. **migrations** is where all the table schema. I personally use knex.js and an self made ORM name tabel written on top of it.

#### └ migrations

```
JS 20171026223642_CreateUsersTable.js
JS 20171026223719_CreateTasksTable.js
JS 20171026223754_CreateTaskCommentsTable...
JS 20171026230158_CreateProjectsTable.js
JS 20171026230159_CreateUserProjectsTable.js
JS 20171026230757_CreateUserTasksTable.js
```

Some of migration files (They are automatically generate from CLI)

5. **node\_modules** you already know it 😊.
6. **tests** is where we write all our unit test.

#### └ tests

```
JS auth.js
JS projects.js
JS users.js
```

The structuring of tests folder is like the actions or routes under app folder and will be explained in app structure section.

7. **config** and **config.sample.js** are where all of our configuration related to database, authentication, external api keys etc are stored.

If you are wondering why we have two such files, then **config.sample.js** goes to your repository so that other developer can know the file shape and can use to form their local **config.js** file.

Also, we can also have a **config.production** for production environment.

*Note: config can be a separate folder too, that can have separate sub configs files for database, external Api's key etc but it feel overkill to me.*

8. **eslint** is where the lint logic goes. Read more [here](#).

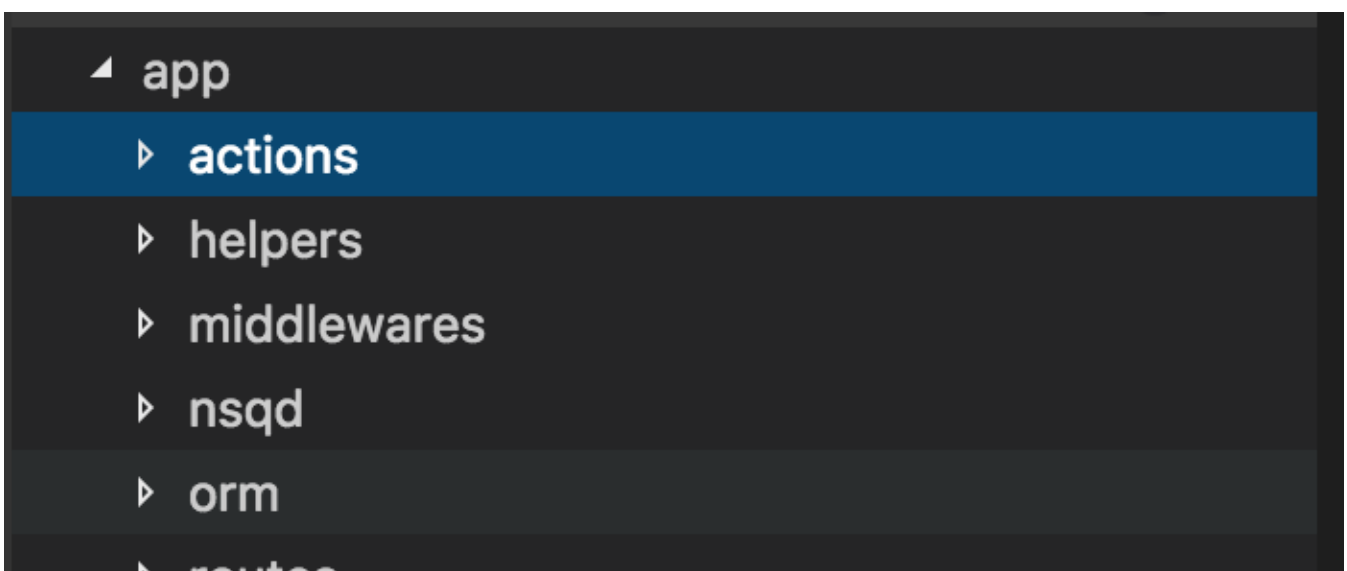
9. **migrate** is the file that pass our configuration from **config.js** to ORM and help us run migration from CLI, check the code [here](#).

10. **gitignore** is where we path to folder or filename that we don't want to push on repository.

11. **package.json** — you already know it 😊.

*Note: We will discuss only the app folder structure because other are straight forward. If you need any help or want to understand anything in code base then hit me up.*

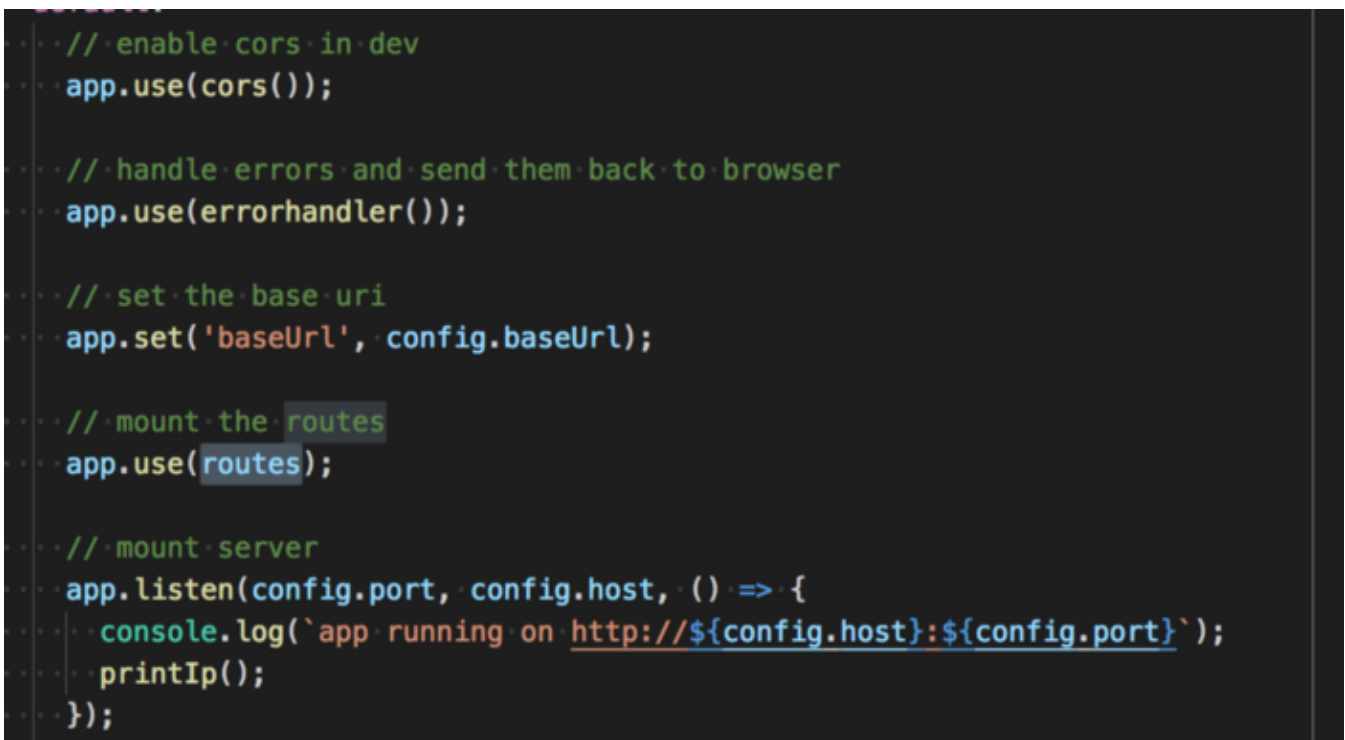
## The app Structure





app folder


1. **server.js** is the file that start your server and will have all middleware that is required for request parsing like `bodyparser`, `cors`, `multer` and `errorhandlers` etc and finally we add routes middleware.



Sample code for server.js

2. **routes** will have sub-files or sub-folders made as per the entities in project.



A dark-themed screenshot of a file explorer. It shows a folder named 'routes' containing two files: 'tasks.js' and 'users.js'. The file names are highlighted in yellow.

routes folder

The index.js under routes looks like as shown below. It have request handler middlewares and a catch all route.

```
1 const app = module.exports = require('express')();
2
3 app.get('/', (req, res) => {
4   res.send({msg: 'hello! Server is up and running'});
5 });
6
7 app.use('/auth', require('./auth'));
8 app.use('/users', require('./users'));
9 app.use('/projects', require('./projects'));
10 app.use('/tasks', require('./tasks'));
11
12 // the catch all route
13 app.all('*', (req, res) => {
14   res.status(404).send({msg: 'not found'});
15 });
16
```

index.js under routes

*Note: We can abstract a big entity like teams in a folder and can call it as teams which can then have sub subfolder like members, contacts, dashboard etc. under routes.*

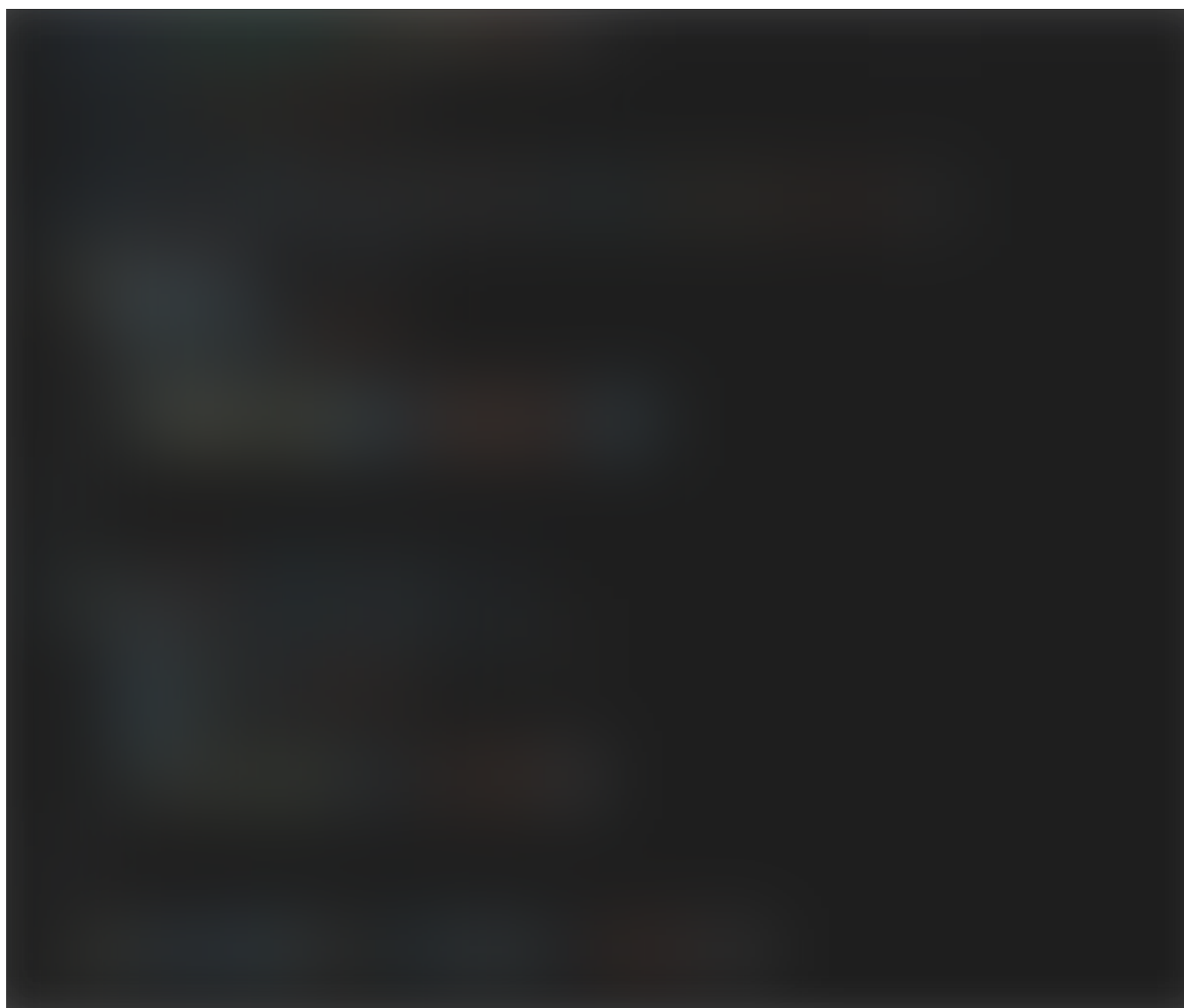






Fractal pattern structuring in routes

A file under routes is where all request handler or sub-routes are written and the only purpose of them is to either call the action that is responsible for that route or the sub-route folder.



auth.js under routes

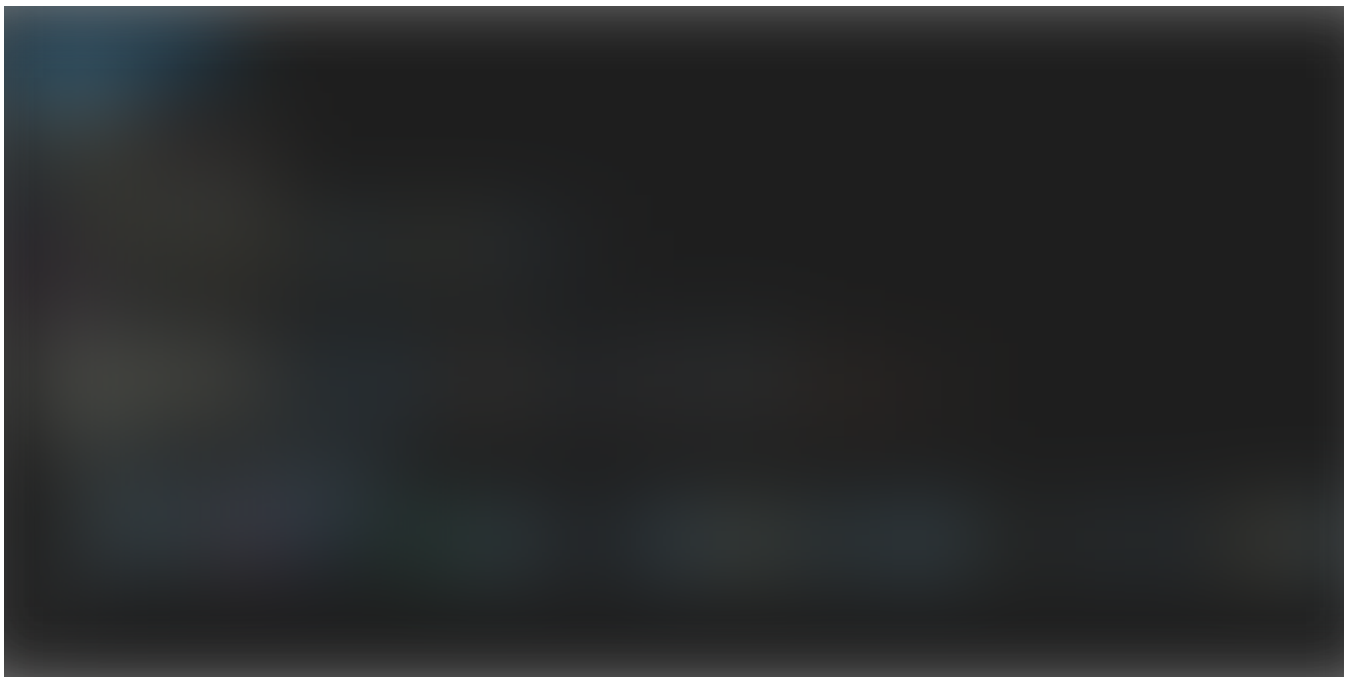
*Note: **signup, login, logout, refresh and forgetpassword** in above image all are actions that are defined under action folder.*

3. Most of the times **actions folder** will have same the same structuring as the routes folder.



actions folder

Each file in **actions** is for business logic i.e all code that store, get, change real entities either in database tables or doing queuing function like enqueue and dequeue etc all happen here.



Sample code for auth file for signup action



---

*Note: Actions can call other actions too.*

---

4. **So, middlewares, utils, helpers, tasks** will follow the same way of structuring as action or routes do i.e main entity will act as a root folder and then will can have subfolder as per sub-entities and repeat.

5. **tasks.js** is used for running tasks from CLI created under task folder like database seed, clean database etc.

6. **nsqd**(optional) is use case specific for distributed messaging.

---

*Note: NSQ is one the best i have used for distributed messaging it works great with no pain in production.*

---

**utils** and **helpers** can be merged if need, i love to keep them separate so that i can reuse utils as its more generic in all projects

## Why fractal

1. Gives us phenomenal scale.
2. Easy to debug as everything go with the domain logic flow.
3. Easy to onboard a new developer to this codebase.

. . .

In Case of Graphql we can get rid of routes folder. Few month back i tried using this structure with Graphql check [here](#) (a raw sample)

If you are totally new to node.js and want to make a backend for REST API's then use [this](#). It contains everything you need from setup to authentication already done for you, just run the shell script.

---

*Note: I personally use our own self created ORM called tabel which we have used in production for many years and i promise you will love it. This ORM and this app structure is all you need for making any kind of backend with REST API/GraphQL.*

---

. . .

It is (and will always be) a work in progress on fractal pattern, i was recently working on removing the entire routes folder and exposing action directly to UI so that the UI can directly call the function with the required parameter which is what GraphQL do but with REST API's.

## Fractal allows you to:

- Reason about location of files
- Manage and create complex domain entities with ease
- Iterate quickly and scale repeatably

. . .

Special thanks to [Kapil Verma](#) who is creator of fractal pattern.

. . .

Tips Are Appreciated! 💰 😊

My Bitcoin address: 132Ndcy1ZHs6DU4pV3q2X1GzSCdBEXX6pz

My Ethereum address: 0xc46204dfc8449Ffb0f02a9e1aD81F30D3f027010

Please consider **entering your email here** if you'd like to be added to my email list and **follow me on medium** to read more article on javascript and on **github** to see my **crazy code**. If anything is not clear or you want to point out something, please comment down below.

You may also like my other articles

1. [Javascript- Currying VS Partial Application](#)
2. [Javascript data structure with map, reduce, filter](#)
3. [Javascript ES6 — Iterables and Iterators](#)
4. [Javascript performance test — for vs for each vs \(map, reduce, filter, find\).](#)
5. [Javascript — Proxy](#)
6. [Javascript — Generator-Yield/Next & Async-Await](#)

. . .

If you liked the article, please clap your heart out. Tip — You can clap 50 times! Also, recommend and share to help others find it!

THANK YOU!

✉ Subscribe to *CodeBurst*'s once-weekly [Email Blast](#), 🐦 Follow *CodeBurst* on [Twitter](#), view 📖 [The 2018 Web Developer Roadmap](#), and 🧠 [Learn Full Stack Web Development](#).

JavaScript

Coding

Nodejs

Rest Api

Backend

**Medium**

About Help Legal