Estructuración del codigo según patrón MVC

Gabriel Rodríguez Flores

November 18, 2021

ADA ITS
INSTITUTO TECNOLÓGICO SUPERIOR

- Estructuración del codigo según patrón MVC

Contents

1 Teoría

1.1 Estructura de carpetas

```
express-api
 | src
 | | controllers
 | | | users.js
 | | loaders
 | | | express.js
 | | | index.js
 | | models
 | | | index.js
 | | routes
 | | | index.js
 | | | users.js
 | | services
 | | | index.js
 | | utils
 | | | index.js
 | | app.js
 | | config.js
 | | index.js
 | test
 | | controllers
 | | | users.test.js
 | | loaders
 | | | express.test.js
 | | routes
 | | | users.test.js
 | | services
 | | sonar.js
 | .env
 | README.md
 | env.template
 | package-lock.json
 | package.json
```

1.2 SRC (Source)

Es el contenido del código.

Existen 3 ficheros en raiz:

- `index.js` Entrada de la ejecución de la aplicación. Importa la configuración y la aplicación en sí misma para arrancar el servidor. Se usa para separar la ejecución de la creación de la aplicación y así poder recoger esta sin necesidad de arrancarla (de cara al uso de los test).

- `app.js` Lanza la carga inicial (loaders) preparando la aplicación para ser arrancada. Exporta el objeto aplicación para que pueda ser usado con los test.
- `config.js` Centraliza la configuración inicial de toda la aplicación.

1.2.1 Loaders

Módulo de carga de servicios. Aquí va todo lo que haya de ser ejecutado una vez en el arranque del servidor y no se vuelva a ejecutar.

- Ejemplo de uso: Conexión con una base de datos

1.2.2 Rutas

En esta carpeta alojaremos las rutas, dentro de ella pueden haber más carpetas según la encapsulación de los endpoints.

1.2.3 Controladores

Los controladores son el cerebro de la aplicación, donde se gestiona y maneja todo el algoritmo y secuencia de instrucciones que conlleva operar la petición y devolver una respuesta.

1.2.4 Servicios

Los servicios son las partes del códgio donde la aplicación actúa de cliente. Se trata de user servicios externos y adaptar la respuesta para el uso de la aplicación.

- Ejemplo de uso: Peticiones a una base de datos, peticiones a un servicio SMTP (correo electrónico), etc …

1.2.5 Modelos

Cuando se trabaja con una base de datos, hay que definir cuál es el modelo de datos que lleva cada una de las tablas o colecciones con las que se va a trabajar. Este es el lugar indicado para ello.

1.2.6 Utils

Es el lugar donde un código de apoyo (alguna transformación de datos, constantes, etc) que no tiene cabida en otro lugar, y se define a parte para ser global y reutilizada.

- *helpers* Todo algoritmo genérico, no atado al funcionamiento particular de la aplicación. Han de ser funciones puras y con un algoritmo genérico, no se debe particularizar la salida para una operación concreta.
- Ejemplo de uso: Un sistema de logger, funciones de manipulación de datos (objetos, array, ...), alacenamiento en memoria, constantes, etc.

1.3 Test

Como ya se ha visto, la carpeta *test* ha de ser una copia en estructura y ficheros del proyecto en sí. Esta granularidad nos permite realizar los test fichero a fichero evaluando el funcionamiento correcto en pequeñas partes y etapas.

Hay 2 tipos de test que podemos dividir en 2 carpetas distintas:

- *unit* Test unitarios como los conocidos con Jest o Ava
- *e2e* Test de integración o funcionales, usados para probar la aplicación como usuario final.

1.4 Ficheros en raíz

1.4.1 ENV

El fichero `.env` nos permite definir variables de entorno para la ejecución, a través de las cuales especificar los datos sensibles que no queremos que estén almacenados en código, y que pueden cambiar con el tiempo o dependiendo del entorno de ejecución que nos encontremos.

> Nota: El fichero `.env` ha de ser añadido al `.gitignore` para proteger los datos sensibles y no exponerlos por error.

Ya que no le hacemos un seguimiento, para tener una copia del `.env` es común crear un fichero `env.template` con todos los nombres de las variables que se quieren definir, y unos valores por defectos para su ejecución en local, siempre asegurando que los valores proporcionados no sean los reales del entorno de producción.

1.4.2 Readme.md

El fichero de documentación escrito en Markdown

1.4.3 Package.json

El fichero de configuración e información del proyecto NPM.

2 Ejemplos

2.1 Módulos

Ejemplos de ficheros en los distintos módulos.

- `index.js`

```
const { logger } = require('./utils');
const app = require('./app');
const config = require('./config');

const { port } = config.app;

app.listen(port, err => {
  if (err) {
    logger.error(err);
    return;
  }
  logger.info(`App listening on port ${port}!`);
});
```

- `app.js`

```
const express = require('express');
const loaders = require('./loaders');
const config = require('./config');

const app = express();

loaders.init(app, config);

module.exports = app;
```

- `config.js` Con el uso de `dotenv`

```
require('dotenv').config();

const app = {
  port: process.env.PORT,
};

const config = {
  app,
};

module.exports = config;
```

2.1.1 Loaders

- index.js

```
const expressLoader = require('./express');

function init(app, config) {
  expressLoader(app, config.security);
}

module.exports = {
  init,
};
```

- express.js

```
const express = require('express');
const cors = require('cors');

const routes = require('../routes');

module.exports = (expressApp, config) => {
  expressApp.use(cors());
  expressApp.use(express.json({ limit: '50mb' }));
  expressApp.use(express.urlencoded({ limit: '50mb', extended: true }))
      ;
  expressApp.use('/api/v1', routes);
  expressApp.use((req, res) => res.status(404).send({ message: 'Not
     Found' }));
};
```

2.1.2 Rutas

- index.js

```
const { Router } = require('express');

const users = require('./users');

const router = Router();

router.use('/users', users);

module.exports = router;
```

- users.js

```
const { Router } = require('express');

const { getUsers, getUserId, createUser } = require('../controllers/
    users');

const router = Router();

router.get('', getUsers);
router.post('', createUser);
router.get('/:id', getUserId);

module.exports = router;
```

2.1.3 Controladores

- users.js

```
const { mongodbService } = require('../services');

async function getUsers(req, res, next) {
  const filters = req.query;
  return res.status(200).send({ results: [], filters });
}

function getUserId(req, res, next) {
  return res.status(200).send(req.params.id);
}

function createUser(req, res, next) {
  mongodbService.createUser(req.body);
  return res.status(201).send({ message: 'Created' });
}

module.exports = {
  getUsers,
  getUserId,
```

```
20   createUser,
21 };
```

2.1.4 Servicios

> Se verá más adelante cuando se exploten servicios externos

2.1.5 Modelos

> Se verá más adelante con la integración de una base de datos

2.1.6 Utils

- index.js

```
1  const logger = require('./logger');
2
3  module.exports = {
4    logger,
5  };
```

- logger.js

```
1  const winston = require('winston');
2
3  const { format, transports, createLogger } = winston;
4
5  const {
6    combine, timestamp, printf, colorize,
7  } = format;
8
9  const logger = createLogger({
10   transports: [
11     new transports.Console({
12       format: combine(
13         timestamp({ format: 'YYYY-MM-DD HH:mm:ss' }),
14         colorize(),
15         printf(info => `[${info.timestamp}] ${info.level} ${info.
               message}`),
16       ),
17       silent: process.env.NODE_ENV === 'test',
18     }),
19   ],
20   exitOnError: false,
21 });
```

```
22
23
24 module.exports = logger;
```

3 Ejercicios

1. Crear una estructura base de ejemplo, que sea ejecutable y permita responder a la ruta `/ping`, con el mensaje 'pong'
2. Crear un proyecto bien estructurado con la ruta `/fibonacci` realizando la correcta separación del código.
    - *controllers* Contiene las llamadas a los distintos servicios si los requiere
    - *routes* Contiene la definición de rutas y selección del controlador que emite la respuesta
    - *loaders* Contiene el despliegue y configuracion del servidor
    - *utils* (opcional) Contiene elementos de utilidad, como el logger.
    - Realizar los test unitarios a todos los ficheros creados (a excepción de los `index.js`)

4 Entregables

4.1 En clase

- Ejercicio 1

4.2 Tarea

- Ejercicio 2

4.3 Trabajo

- Implementar la estructura para el proyecto de notas aplicando la división del código en los distintos módulos.