Instalación y configuración para el uso del framework para el desarrollo Backend: Node.js x Express.js

Node.js y NPM

Como paso inicial, al tener que ocupar el manejador de paquetes npm en nuestros proyectos debemos de instalarlo, para eso instalamos Node.js en nuestro sistema. Se instala la versión más actual (v17.7.2), y en el instalador se ubica en donde se requiera, con las opciones predeterminadas.

Para más información de Node.js consultar Documentation | Node.js (nodejs.org)

Para determinar si se realizó la instalación de npm de forma correcta, se ejecuta el comando

npm -v

dándonos como resultado que la versión actual de npm es la 8.5.2. Podemos modificar esta versión con el comando

npm install -g npm@[version]

Para más información de npm consultar npm Docs (npmjs.com)

Express.js

Debemos de ubicar dónde queremos crear nuestro proyecto para la parte de back-end, si estamos trabajando un proyecto donde esta tanto back-end y front-end entonces debemos de ponerlo al mismo nivel que la carpeta (recomendable 'server') para el back-end y llamar a nuestro proyecto de React (recomendable 'client').
Si no, solo nos ubicamos en un directorio donde queramos crear el proyecto. Entramos a este directorio.

Ejecutamos el comando

npm init

Esto va a crear el archivo de configuración para nuestro proyecto y podemos especificar en la terminal el nombre del proyecto, entre otras cosas. Vamos a poder agregar o modificar comandos en el archivo de configuración package.json, además de que cuando instalemos dependencias se guardarán aquí.

Para poder utilizar Express.js, debemos de instalarlo, esto se realiza con

npm install express

En este momento ya podemos realizar lo que queramos dentro del directorio.

Esqueleto de Proyecto o Boilerplate Express.js

Dentro de los boilerplates, se tiene una variedad para cada tipo de proyecto, y dependen muchos de las bibliotecas a utilizar. Existen para REST, Microservicios, y dentro de ellos para diseños complejos o simples.

Nosotros construiremos un RESTful API, donde los endpoints son creados en Express.js. Además de eso, utilizando mongoose para que se puedan manejar los datos validando un esquema de forma sencilla para MongoDB. Como lo que vamos a construir es específico del proyecto, necesitaremos tener una carpeta para las diversas conexiones a las base de datos, un archivo general en el directorio 'server' para recolectar todo, los esquemas van a estar en su propio directorio de 'models', tendremos de igual forma una carpeta para las rutas y controladores donde tendremos las promesas para conseguir realizar todas las operaciones CRUD y sus respectivas rutas, teniendo que dividir para cada uno de estos directorios entre los servicios o secciones de la aplicación a trabajar. Resultando en el siguiente árbol de directorios:

- *server*
  - *controllers*
    - *…*
  - *database*
    - *…*
  - *models*
    - *…*
  - *routes*
    - *…*

Conceptos a utilizar de Node.js, Express.js, Mongoose

Dentro de Node.js, para la fácil realización de endpoints se utiliza Express.js, en nuestro archivo principal de aplicación necesitamos instanciar la aplicación que es un objeto el cual tiene métodos que nos ayudan a recibir estas solicitudes de HTTP y escuchar a puertos.

```
const express = require('express')
const app = express()
…
app.get('/', (req, res) => {
        …
});
…
```

app.listen(port);

Dentro de estos métodos para recibir solicitudes HTTP del cliente, se encuentra la ruta que queremos seguir, además de una función callback. Node.js maneja acciones tanto síncronas y asíncronas, pero principalmente ataca las asíncronas, y una vez que se quiere realizar una operación asíncrona, se puede invocar esta función callback para completar otro proceso, el cual estará ligado a nuestras reglas de negocio. Principalmente se utilizan estos callbacks como middleware o precondiciones de una ruta.

Una vez que tenemos esto en claro, debemos de dividir estas solicitudes por sección, y al final utilizar el método use para que cada sección la cual es un router que usa express.router para realizar solicitudes, donde cada una de estas crea promesas para realizar operaciones CRUD a la base de datos.

```
….
const db = require('./db')
const router = require('./routes/router')
app.use('/api', router)
```

Las operaciones CRUD serán C de crear, R de Leer o Recibir, U de Actualizar, y D de eliminar, esto utilizando el esquema previamente creado para poder realizar queries a la base de datos. Estas promesas simplemente son objetos que pueden estar en 3 estados, pendiente, exitoso, o rechazado, donde en el estado pendiente el valor es la promesa, y en el exitoso o rechazado serán la respuesta de la solicitud. Se tienen que manejar los datos, y validar los posibles códigos de estado HTTP, cachar los errores y mandar los datos correctos.

Un esquema tiene la siguiente forma dentro de Mongoose:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const Esquema = new Schema(
        {
                …
        },
        …
);
module.exports = mongoose.model(‘esquema’, Esquema);
```

Y para manejar estas solicitudes como lo mencionado previamente, se puede utilizar el formato de Promise o async-await:

```
accionCRUD = (req, res) => {
        const body = req.body;

        if (!body) {
                …
        }
```

```
        const data = new Esquema(body);

        if(!data){
                ...
        }

        data.save().then(...).catch(...);
};

router.post('/accioncrud', Controlador.accionCRUD);
```

Esto permite una fácil manipulación de la base de datos para el cliente, para el desarrollador, además de que se puede realizar la validación del esquema de forma simple.

Para más información sobre Express.js consultar https://expressjs.com/en/5x/api.html#express

Para más información sobre Mongoose consultar https://mongoosejs.com/docs/guide.html

Instalación e implementación de bibliotecas para Express.js

Para instalar cualquier biblioteca que se quiera usar, se utiliza el comando

npm install [biblioteca]

Todas las bibliotecas o la mayoría de ellas sigue el patrón de tener una configuración en el componente principal o tener su carpeta separada dentro del directorio principal, pueden ser ayudas para alguno de los procesos por lo que es específico de la biblioteca como implementarlo.

Para más información, buscar la biblioteca a utilizar para su instalación, implementación, y documentación en general.

Repositorio con código de un ejemplo de aplicación: https://github.com/jpfragoso30/ExpressUsers