Práctica WEB Backend Node.js-MongoDB

Bootcamp Web17 2024

Imaginemos que un cliente nos pasa el siguiente briefing para que le hagamos este trabajo:

Desarrollar un website con SSR (ejs) para un servicio de venta de artículos de segunda mano llamado Nodepop.

El servicio mantiene productos a la venta y permite buscar como poner filtros por varios criterios.

Cada producto publicado tiene los siguientes datos:

- Nombre del producto, un anuncio siempre tendrá un solo artículo
- Propietario. Usuario que ha publicado el producto
- Precio
- Foto del producto. Cada producto anunciado tendrá solo una foto.
- Tags del producto. Podrá contener uno o varios de estos cuatro: work, lifestyle, motor y mobile

Operaciones que debe permitir realizar:

- Lista de productos del usuario, con posibilidad de paginación. Con filtros por tag, rango de precio (precio min. y precio max.) y nombre de producto (que empiece por el dato buscado)
    - Por ejemplo si hacemos una petición a http://localhost:3000/?skip=1&limit=3&sort=name&tag=lifestyle mostraría los productos correspondientes.
- Creación de producto
- Borrado de producto

Importante, cada usuario solo puede ver o eliminar sus propios productos.

Los sistemas donde se desplegará el website utilizan base de datos MongoDB.

Se solicita que el entregable venga acompañado de una mínima documentación y el código esté bien formateado para facilitar su mantenimiento. En esta fase, ya que se desea probar si el modelo de negocio va a funcionar, no serán necesarios ni tests unitarios ni de integración.

Notas para el desarrollador

Cómo empezar

El orden de las primeras tareas podría ser:

1. Crear app Express y probarla (npx express-generator nodepop --ejs)
2. Instalar Mongoose, modelo de productos y probarlo (con algún product.save por ejemplo)
3. Hacer un script de inicialización de la base de datos, que cargue productos y usuarios iniciales. Se puede llamar p.e. initDB.js, debería borrar las tablas y cargar posteriormente. Lo podemos poner en el package.json para poder usar npm run initDB. Referencias:
    a. Cargar vuestro módulo connectMongoose.js y vuestros modelos
    b. Usar deleteMany e insertMany para cargar las entidades.
    c. Estas operaciones deberán hacerse una detrás de la otra, o dicho de otro modo, cuando termine la primera se lanzará la segunda. Para esto podéis usar callbacks, promesas (si miráis la doc veréis que ambos devuelven una promesa) o promesas con async/await (**recomendado**).
4. Hacer un fichero README.md con las instrucciones de uso puede ser una muy buena idea, lo ponemos en la raíz del proyecto y si apuntamos ahí como arrancarlo, como inicializar la BD, etc nos vendrá bien para cuando lo olvidemos o lo coja otra persona
5. Hacer una primera versión básica, por ejemplo GET / que devuelva la lista de productos sin filtros en JSON.
6. Crear la página de inicio del site y sacar la lista de productos
7. Hacer login y logout
8. Mejorar la lista de productos del usuario poniendo filtros, paginación, etc
9. A partir de aquí ya podemos meternos con la creación, borrado y edición!

Detalles útiles

Tras analizar el briefing vemos que tenemos que guardar cosas en la base de datos, como por ejemplo los productos.

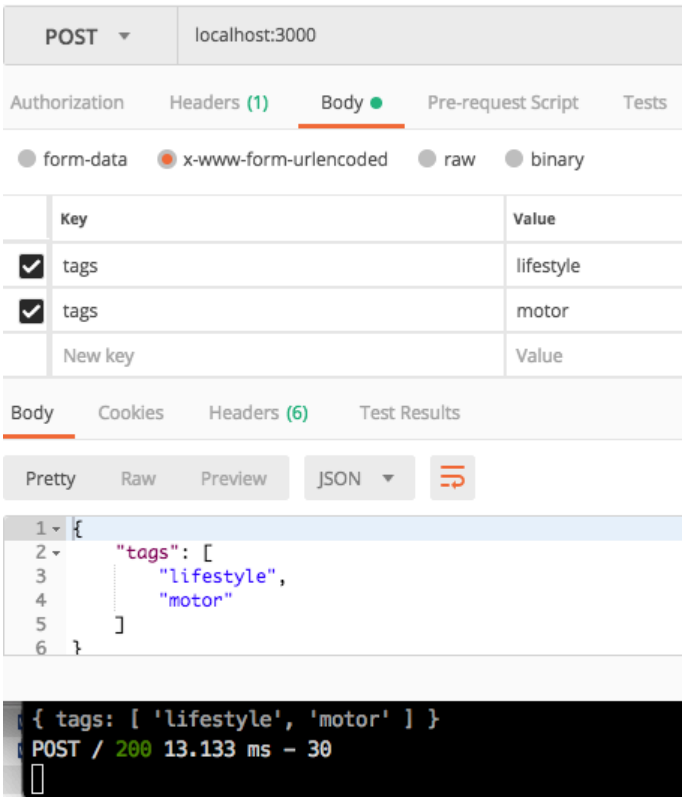
Por tanto, podemos hacer modelos de mongoose con esta definición.

```
var productSchema = mongoose.Schema({
    name: String,
    owner: { ref: 'User', type: mongoose.Schema.Types.ObjectId },
    price: Number,
```

```
    image: String,
    tags: [String]
});
```

Nos vendrá bien hacer un script de inicialización de la base de datos, que podemos llamar initDB.js. Este script debería borrar las tablas si existen y cargar los datos iniciales.

Tip: Enviar arrays con Postman



Lista de productos

Lista de productos paginada.

Filtros:

- por tag, tendremos que buscar incluyendo una condición por tag
- rango de precio (precio min. y precio max.), podemos usar un parámetro en la query string llamado precio que tenga una de estas combinaciones:
    - 10-50 buscará productos con precio incluido entre estos valores `{ price: { '$gte': '10', '$lte': '50' } }`
    - 10- buscará los que tengan precio mayor que 10 `{ price: { '$gte': '10' } }`
    - -50 buscará los que tengan precio menor de 50 `{ price: { '$lte': '50' } }`
    - 50 buscará los que tengan precio igual a 50 `{ price: '50' }`
- nombre de producto, que empiece por el dato buscado en el parámetro nombre. Una expresión regular nos puede ayudar `filters.name = new RegExp('^' + req.query.name, "i");`

Para mostrar una lista de productos, la llamada podría ser una como esta:
`GET http://localhost:3000/?tag=mobile&name=ip&price=50-&skip=0&limit=2&sort=price`

Documentación y calidad de código

Como nos piden algo de documentación podemos usar la página index de nuestro proyecto o un fichero README.md para escribir la documentación del API, y los más valientes pueden probar a hacerlo con Swagger en Express (OpenAPI).

En cuanto a la calidad de código, será un punto a nuestro favor que lo validemos con ESLint (https://eslint.org/), que unifica revisión de estilo de código y de posibles bugs.

Para saber más echa un vistazo a https://eslint.org/docs/latest/use/getting-started