Práctica WEB-API/Node.js/MongoDB

Bootcamp Web9 2020

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

Desarrollar el API que se ejecutará en el servidor de un servicio de venta de artículos de segunda mano llamado Nodepop. Hazte a la idea que esta API que vas a construir sería utilizado por otros desarrolladores de iOS o Android.

El servicio mantiene anuncios de compra o venta de artículos y permite buscar como poner filtros por varios criterios, por tanto la API a desarrollar deberá proveer los métodos necesarios para esto.

Cada anuncio tiene los siguientes datos:

- Nombre del artículo, un anuncio siempre tendrá un solo artículo
- Si el artículo se vende o se busca
- Precio. Será el precio del artículo en caso de ser una oferta de venta. En caso de que sea un anuncio de 'se busca' será el precio que el solicitante estaría dispuesto a pagar
- Foto del artículo. Cada anuncio tendrá solo una foto.
- Tags del anuncio. Podrá contener uno o varios de estos cuatro: work, lifestyle, motor y mobile

Operaciones que debe realizar el API a crear:

- Lista de anuncios con posibilidad de paginación. Con filtros por tag, tipo de anuncio (venta o búsqueda), rango de precio (precio min. y precio max.) y nombre de artículo (que empiece por el dato buscado)
- Lista de tags existentes
- Creación de anuncio

Los sistemas donde se desplegará el API utilizan bases de datos MongoDB.

Se solicita que el entregable venga acompañado de una mínima documentación y el código del API 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.

El site donde se despliegue tendrá, **además del API**, una página (frontend) que muestre una lista de anuncios con filtros en su página principal. Obtendrá información de la base de datos usando los modelos y la mostrará.
Para mostrar los datos **será suficiente con una página EJS que muestre la lista de anuncios**, en la que si ponemos filtros en la URL los aplique, algo como lo que hicimos en clase con el API, pero en una página web.

Por ejemplo si hacemos una petición a *http://localhost:3000/?start=1&limit=3&sort=name&tag=lifestyle* mostraría los anuncios correspondientes a esos filtros.

Notas para el desarrollador

Cómo empezar

El orden de las primeras tareas podría ser:

1. Crear app Express y probarla (express nodepop --ejs)
2. Instalar Mongoose, modelo de anuncios y probarlo (con algún anuncio.save por ejemplo)
3. Hacer un script de inicialización de la base de datos, que cargue el json de anuncios. Se puede llamar p.e. install_db.js, debería borrar las tablas y cargar anuncios. Lo podemos poner en el package.json para poder usar npm run installDB. Referencias:
    a. Cargar vuestro módulo connectMongoose.js y vuestros modelos, luego usad *conn.once('open', () => { ...* para empezar el proceso
    b. http://mongoosejs.com/docs/api.html#query_Query-deleteMany
    c. http://mongoosejs.com/docs/api.html#model_Model.insertMany
    d. 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 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 del API, por ejemplo GET /apiv1/anuncios que devuelva la lista de anuncios sin filtros.
6. Crear la página de inicio del site y sacar la lista de anuncios
7. Mejorar la lista de anuncios poniendo filtros, paginación, etc
8. A partir de aquí ya tendríamos mucho hecho!

Detalles útiles

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

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

```
var anuncioSchema = mongoose.Schema({
    nombre: String,
    venta: Boolean,
    precio: Number,
```

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

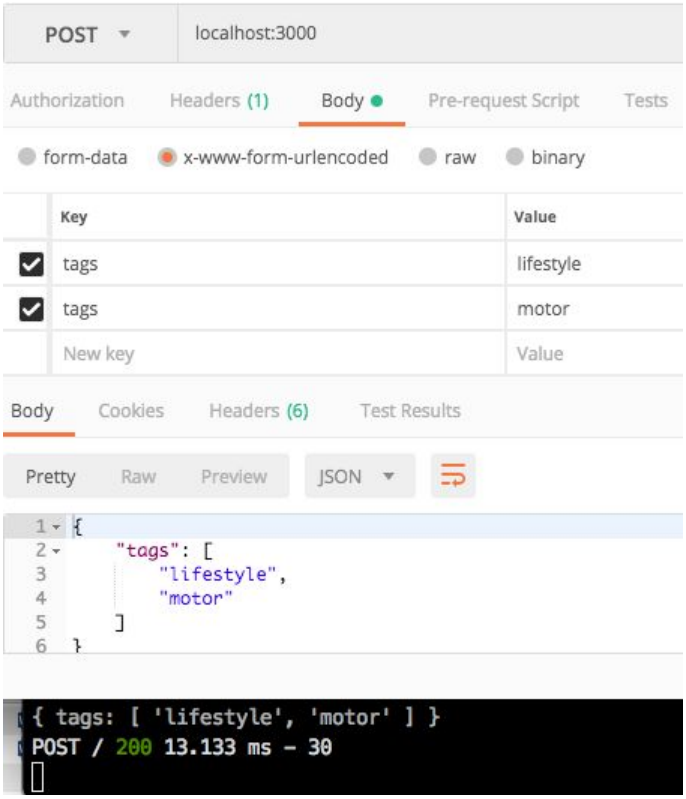
Nos vendrá bien hacer un script de inicialización de la base de datos, que podemos llamar install_bd.js. Este script debería borrar las tablas si existen y cargar un fichero llamado anuncios.json que tendrá un contenido similar a este:

```
// anuncios.json
{
  "anuncios": [
    {
      "nombre": "Bicicleta",
      "venta": true,
      "precio": 230.15,
      "foto": "bici.jpg",
      "tags": [ "lifestyle", "motor"]
    },
    {
      "nombre": "iPhone 3GS",
      "venta": false,
      "precio": 50.00,
      "foto": "iphone.png",
      "tags": [ "lifestyle", "mobile"]
    }
  ]
}
```

Podéis añadir más anuncios si queréis.

Las fotos podéis hacerlas con el móvil o sacarlas de algún banco fotográfico gratuito… el API tendrá que devolver las imágenes, por ejemplo de la carpeta /public/images/<nombreRecurso>, por tanto obtendriamos una imagen haciendo una petición en la url http://localhost:3000/images/anuncios/**iphone.png**

Enviar arrays con Postman



Lista de anuncios

Lista de anuncios paginada.

Filtros:

- por tag, tendremos que buscar incluyendo una condición por tag
- tipo de anuncio (venta o búsqueda), podemos usar un parámetro en query string llamado venta que tenga true o false
- 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 anuncios con precio incluido entre estos valores `{ precio: { '$gte': '10', '$lte': '50' } }`
    - 10- buscará los que tengan precio mayor que 10 `{ precio: { '$gte': '10' } }`

    - -50 buscará los que tengan precio menor de 50 `{ precio: { '$lte': '50' } }`
    - 50 buscará los que tengan precio igual a 50 `{ precio: '50' }`
- nombre de artículo, que empiece por el dato buscado en el parámetro nombre. Una expresión regular nos puede ayudar `filters.nombre = new RegExp('^' + req.query.nombre, "i");`

Para recibir la lista de anuncios, la llamada podría ser una como esta:
`GET http://localhost:3000/apiv1/anuncios?tag=mobile&venta=false&nombre=ip&precio=50-&start=0&limit=2&sort=precio`

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 iodocs.

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

En ESlint podemos crear un fichero para definir las reglas con:

```
$ npm install eslint
```

```
$ ./node_modules/.bin/eslint --init
```

```
{
  "node": true,
  "esnext": true,
  "globals": {},
```

```
  "globalstrict": true,
  "quotmark": "single",
  "undef": true,
  "unused": true
}
```

Esto debería haberte creado un fichero .eslintrc para que vayas añadiendo las reglas que quieras. Para saber más echa un vistazo a https://eslint.org/docs/user-guide/getting-started

Ver https://www.sitepoint.com/comparison-javascript-linting-tools/

Si las utilidades como ESLint las metemos como scripts de NPM (http://www.jayway.com/2014/03/28/running-scripts-with-npm/) nos será muy fácil pasarlas con frecuencia.