Práctica WEB-API/Node.js/MongoDB

Bootcamp Web3 2017

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

Desarrollar el software que se ejecutará en el servidor dando servicio a una app (API) de venta de artículos de segunda mano, llamada Nodepop. Con esta API que vas a construir se comunicará tanto la app versión iOS y como la versión Android.

La pantalla principal de la app muestra una lista de anuncios y permite tanto buscar como poner filtros por varios criterios, por tanto la API a desarrollar deberá proveer los métodos necesarios para esto.

Cada anuncio mostrará 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

La app estará disponible en inglés o español, por tanto el API será utilizado especificando el idioma del usuario en cada petición. Los tags se tratarán siempre en inglés por tanto no necesitan traducciones. Lo único que el API devolverá traducido al lenguaje del usuario son los mensajes de error, ya que la app mostrará estos mensajes al usuario.

Operaciones que debe realizar el API a crear:

- Lista de anuncios paginada. 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.

El API recibirá bastantes peticiones en algunos momentos del día, especialmente los fines de semana, por tanto queremos que aproveche lo mejor posible los recursos del servidor donde estará instalado.

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 una lista de anuncios con filtros en su página principal. La cual utilizará el API para obtener la información. Esta página puede hacerse con la tecnología que el desarrollador elija (solo javascript, con algún framework como Angular o Vue.js, o con JQuery, etc).

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, y algún usuario. Lo podemos poner en el package.json para poder usar npm run installDB.
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. Para tener los errores en un formato estándar podéis hacer un módulo con un objeto CustomError y usarlo en los distintos sitios donde tengáis que devolverlos. Esto además nos facilitará el trabajo cuando tengamos que hacer que salgan en distintos idiomas.
7. Crear la página de inicio del site y sacar la lista de anuncios
8. Mejorar la lista de anuncios poniendo filtros, paginación, etc
9. 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 en la url http://localhost:3000/images/anuncios/**iphone.png**

Internacionalización

Solo lo haremos con los mensajes de error. Un módulo con una función que traduzca puede ser una buena idea.

Esa función recibiría una clave, por ejemplo 'FIELD_NOT_FOUND', y la buscaría en una tabla de literales filtrando por el idioma de la petición, por ejemplo 'es'.

La tabla de literales puede ser perfectamente un JSON en el filesystem que nuestro módulo cargará la primera vez que alguien le requiera.

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&includeTotal=true&token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJfaWQiOiI1NWZkOWFiZGE4Y2QxZDlhMjQwYzgyMzEiLCJub21icmUiOiJhZG1pbiIsImVtYWlsIjoiamFtZzQ0QGdtYWlsLmNvbSIsImNsYXZlIjoiMTIzIiwiX192IjowfQ.y8wPJhNaS8Vf51ZlX9qZBlrTLGGy4JzDgN2eGSHeQfg
```

Nos piden que aproveche los recursos, por tanto pondremos cluster.

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 jshint, pudiendo añadir si tenemos más tiempo validación con JSCS.

En jshint podemos usar este fichero para definir qué reglas manejar:

```
// .jshintrc (poner en el root del proyecto)
{
  "node": true,
  "esnext": true,
  "globals": {},
  "globalstrict": true,
  "quotmark": "single",
  "undef": true,
  "unused": true
}
```

También de forma alternativa podemos probar a utilizar ESLint (http://eslint.org/) que tiene las dos cosas en uno. Ver https://www.sitepoint.com/comparison-javascript-linting-tools/

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