


Paginación

 gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/persistencia-de-datos-con-mongodb/paginacion.html

Normalmente cuando consultamos el listado de las publicaciones no es recomendable enviar todos los documentos que están guardados en una colección, puede que en desarrollo no lo veamos ya que pueden ser unos cuantos, pero en producción pueden ser miles e incluso millones y eso afectaría el rendimiento de esa respuesta de la aplicación, por lo tanto la practica recomendada es enviar los documentos con paginación, por ejemplo podemos enviar por defecto enviar grupos de 10 e indicar con la meta información cuantos documentos en total están disponibles para consultar en la colección, adicionalmente podemos crear los siguientes parámetros para permitirle al usuario que documentos consultar:

- `limit` : Determina el número de documentos en la petición, si el usuario no lo especifica por defecto lo estableceremos en 10.
- `skip` : Indica el número de documentos a saltar de la colección, si el usuario no lo especifica por defecto lo estableceremos en 0.
- `page` : Indica el número de la página de los documentos de la petición, este campo brinda al usuario una opción mas amigable para navegar entre los documentos de la colección, pero internamente lo utilizaremos para establecer el valor del `skip` , si el usuario no lo especifica por defecto lo estableceremos en 1.

En la respuesta de la petición ademas de enviar los valores mencionados anteriormente también vamos a añadir los siguientes campos:

- `total` : Indica el número total de documentos en la colección, esto le sirve al usuario para saber si existen mas documentos en la colección.
- `pages` : Indica el número de páginas, este es un campo calculado dependiendo los valores anterior y le sirve de información al usuario.

Primero que todo modificamos la configuración (`server/config/index.js`) para establecer los valores por defecto de los parámetros mencionados:

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

const config = {
  server: {
    hostname: process.env.SERVER_HOSTNAME,
    port: process.env.SERVER_PORT,
  },
  database: {
    url: process.env.DATABASE_URL,
  },
  pagination: {
    limit: 10,
    skip: 0,
    page: 1,
  },
};

module.exports = config;
```

Antes de empezar a modificar el código es muy importante mencionar que en bases de datos *NoSQL* como *MongoDB* en una sola consulta no podemos traer el numero de documentos totales y al mismo tiempo los documentos, se deben realizar dos consultas, pero afortunadamente las Promesas tienen un patrón para este tipo de casos llamado: `Promise.all` donde recibe un *Array* de promesas y una vez se cumplan todas ejecuta la función `then` antes vista, esto es mas optimo que hacer una primero y otra despues, ya que con este patrón se ejecutarán en paralelo y no de manera secuencial.

Procedemos a modificar nuestra acción de `all` en el controlador de *posts* (`server/api/v1/posts/controller.js`):

```

const logger = require('winston');

const config = require(' ../../../../config');
const Model = require('../model');

const { pagination } = config;

...

exports.all = (req, res, next) => {
  const { query: qs } = req;
  const limit = qs.limit ? parseInt(qs.limit, 10) : pagination.limit;
  const page = qs.page ? parseInt(qs.page, 10) : pagination.page;
  const skip = qs.page ? ((page - 1) * limit) : (parseInt(qs.skip, 10) ||
  pagination.skip);

  const all = Model.find().limit(limit).skip(skip);
  const count = Model.count();

  Promise.all([all.exec(), count.exec()])
    .then((data) => {
      const [docs, total] = data;
      const pages = Math.ceil(total / limit);

      res.json({
        success: true,
        items: docs,
        meta: {
          limit,
          skip,
          total,
          page,
          pages,
        },
      });
    })
    .catch((err) => {
      next(new Error(err));
    });
};

...

```

Lo más probable es que la lógica del calculo de las variables de paginación la utilizemos varias veces con otro recurso, por lo tanto vamos a reunir esta lógica en una función para ellos creamos el archivo `server/utills/index.js` y colocamos todo lo asociado a esta lógica.

```

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

const {
  pagination,
} = config;

const parsePaginationParams = ({
  limit = pagination.limit,
  page = pagination.page,
  skip = pagination.skip,
}) => ({
  limit: parseInt(limit, 10),
  page: parseInt(page, 10),
  skip: skip ? parseInt(skip, 10) : (page - 1) * limit,
});

module.exports = {
  parsePaginationParams,
};

```

En el fragmento de código anterior hemos utilizado varias funcionalidades de los Arrow Functions disponibles desde ES2015:

1. Hemos realizado un *Destructuring assignment* en los parámetros para solo tomar los que necesitamos.
2. Hemos asignado el valor por defecto si es que estos parámetros no existirían en el objeto que se recibe como argumento de esta función con los *Default Values*.
3. Utilizamos el *Concise Syntax* en la línea de retorno de la función ya que es una sola línea.

Luego volvemos a modificar nuestro controlador (`server/api/v1/posts/controller.js`) para incluir nuevamente la lógica:

```

const logger = require('winston');

const { parsePaginationParams } = require('../../../../utils');
const Model = require('../model');

...

exports.all = (req, res, next) => {
  const { query = {} } = req;
  const { limit, page, skip } = parsePaginationParams(query);

  const all = Model.find().limit(limit).skip(skip);
  const count = Model.count();

  ...
};

```