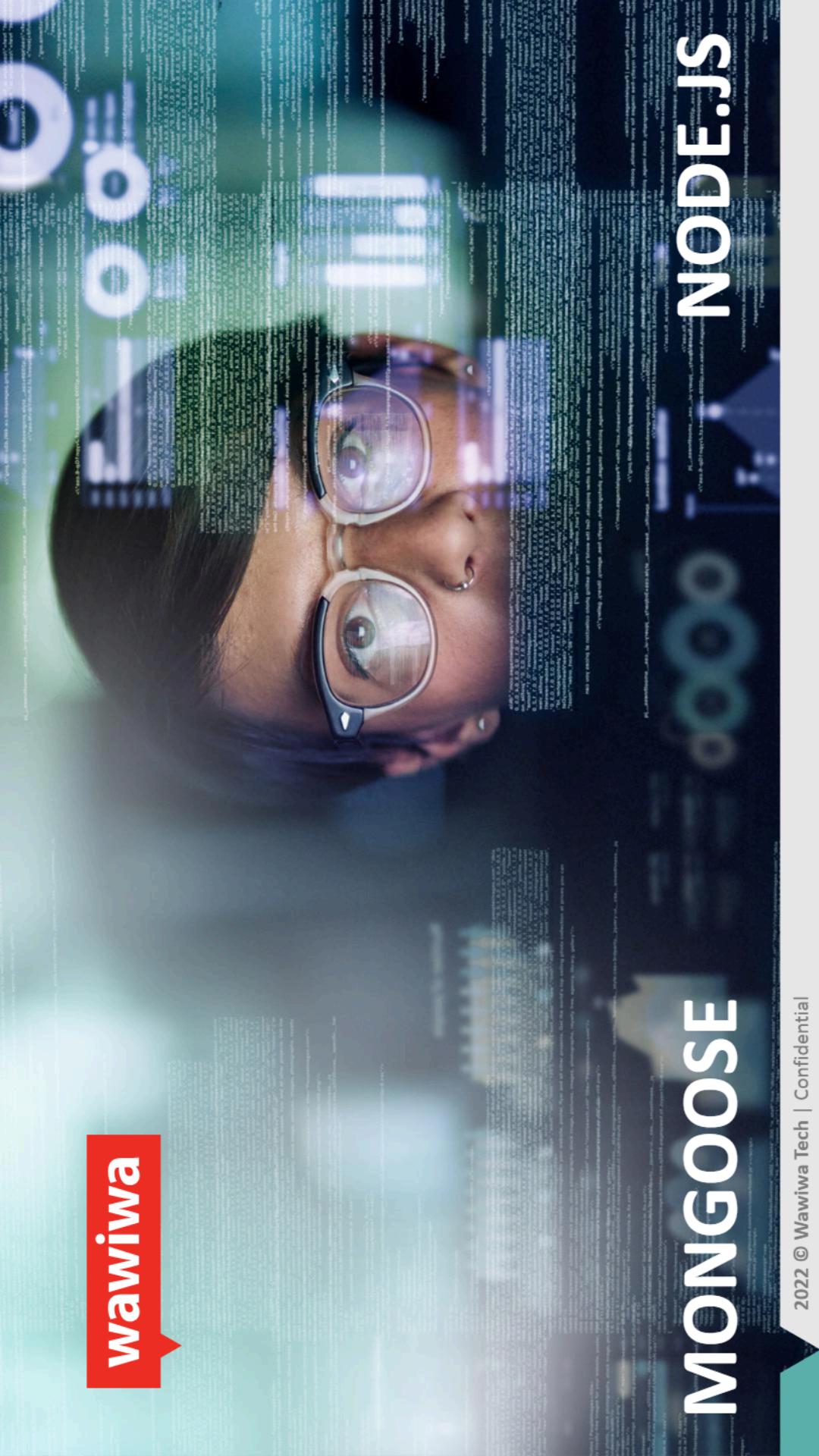


NODE.JS

MONGOOSE

wawiwa



Solicitudes de Mongoose - [mongoose](#)

Enlace a Solicitudes de Mongoose

The screenshot shows the official [mongoose](#) documentation page for version 6.1.5. The 'Queries' section is highlighted. It lists various static helper functions for CRUD operations:

- Model.deleteMany()
- Model.deleteOne()
- Model.find()
- Model.findById()
- Model.findByIdAndDelete()
- Model.findByIdAndRemove()
- Model.findOneAndDelete()
- Model.findOneAndRemove()
- Model.findOneAndReplace()
- Model.findOneAndUpdate()
- Model.replaceOne()
- Model.replaceOne()
- Model.updateMany()
- Model.updateOne()

A note states: "A mongoose query can be executed in one of two ways. First, if you pass in a `callback` function, Mongoose will execute the query asynchronously and pass the results to the `callback`. A query also has a `then()` function, and thus can be used as a promise."

At the bottom right, there's a 'Learn More' button and a small note: "An Enterprise React application built by the [Eduardo SCO](#). The only data grid with integrated charts."

Queries

The screenshot shows the official [mongoose](#) documentation page for version 6.1.5. The 'Queries' section is highlighted. It lists various static helper functions for CRUD operations:

- Model.deleteMany()
- Model.deleteOne()
- Model.find()
- Model.findById()
- Model.findByIdAndDelete()
- Model.findByIdAndRemove()
- Model.findOneAndDelete()
- Model.findOneAndRemove()
- Model.findOneAndReplace()
- Model.findOneAndUpdate()
- Model.replaceOne()
- Model.replaceOne()
- Model.updateMany()
- Model.updateOne()

A note states: "Mongoose models provide several static `mongoose Query` object."

Esquema

Todo en Mongoose comienza con un **Esquema**. Cada esquema se mapea a una colección de MongoDB y define la estructura de los documentos dentro de una colección.

Esquema

Crea una nueva carpeta (elige un nombre) y ejecuta **npm init** en la terminal. Luego ejecuta **npm install mongoose** y **npm install express**.

Crea el archivo **app.js**

Crea el archivo **ProductModel.js**

Ejemplo de Esquema MongoDB- ProductModel.js

Crea un archivo con el nombre del modelo que deseas guardar en la base de datos (en nuestro caso ProductModel):

JS ProductModel.js

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;
const ProductSchema = new Schema({
  title:String,
  Description:String,
  price:Number
  created: Date
});
module.exports = mongoose.model('product', ProductSchema);
```

El Esquema define los campos del modelo

Exporta el Esquema para que otros módulos puedan usarlos

El nombre de la colección de mongoDB será 'product'

Esquema MongoDB - Agregar validación - ProductModel.js

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;
const ProductSchema = new Schema({
  title: {
    type: String,
    require: [true, 'A product must have title'],
    validate: {
      validator: function() {
        if (!this.title) {
          throw new Error('Title is required');
        }
      },
      message: 'Puedes agregar validadores'
    }
  },
  description: {
    type: String,
    minlength: [5, 'Description is minimum 20 characters'],
    maxlength: [1000, 'Description is maximum 1000 characters']
  },
  price: {
    type: Number,
    required: [true, 'A product must have price'],
    validate: {
      validator: function() {
        if (!this.price) {
          throw new Error('Price is required');
        }
        if (this.price <= 0) {
          throw new Error('Price must be above 0');
        }
        if (this.price > 10000) {
          throw new Error('Price must be below 10000');
        }
      },
      message: 'Validadores'
    }
  },
  created: Date
});

module.exports = mongoose.model('product', ProductSchema);
```

Insertar información a MongoDB - app.js

```
const express = require('express');
const app = express();
app.use(express.json());

const CurrentProduct = require('./ProductModel');

const mongoose = require('mongoose');

const strConnect =
  "mongodb+srv://asaf:asaf@cluster0.hqfhv.mongodb.net/myFirstDatabase?retryWrites=true&w=majority";

const OPT = {
  useNewUrlParser: true
};

Usa la API de
MongoDB - enlace
```

Obtén el ProductModel
que creamos

The MongoDB Node.js driver rewrote the tool it uses to parse MongoDB connection strings. Because this is such a big change, they put the new connection string parser behind a flag. To turn on this option, pass the `useNewUrlParser` option to `mongoose.connect()` or `mongoose.createConnection()`.

Continúa en la siguiente diapositiva

Este debe ser tu propio **string de conexión** a la base de datos (ve a la primera diapositiva de este capítulo para ver cómo obtenerlo)

Insertar información a MongoDB -app.js

```
app.post('/api/v1/products', function(req, res, next) {  
  let p1= req.body;  
  console.log(req.body)  
  const newItem = new CurrentProduct(p1);  
  newItem.save().then(item=>{  
    res.json({item:item})  
  }) .catch(err=>{  
    console.log("error 🤦 : " +err)  
  }) ;  
});
```

Continúa en la siguiente diapositiva

Insertar información a MongoDB -app.js

Conectate a MongoDB
usando mongoose

```
mongoose.connect(strConnect, OPT);  
  
const port = process.env.PORT || 3000;  
  
app.listen(port, function() {  
  
    console.log("Running on port " + port);  
  
})
```

Insertar información a MongoDB

Corre tu aplicación ejecutando el comando **node app** en la terminal.
localhost:3000/api/v1/products

The screenshot shows a POST request to `localhost:3000/api/v1/products`. The Body tab contains the following JSON payload:

```
1   {
2     "title": "Red ball",
3     "description": "This is a red ball",
4     "created": "10/10/2023",
5     "price": 300
6 }
```

The Response tab shows the following JSON object:

```
1   {
2     "item": {
3       "title": "Red ball",
4       "description": "This is a red ball",
5       "price": 300,
6       "created": "2023-10-09T21:00:00.000Z",
7       "_id": "61dc1f9e8693b53f3e4f461f",
8       "__v": 0
9     }
10 }
```

Details from the top right: Status 200 OK, 189 ms, 390 B, Save Response.

Insertar información a MongoDB

En tu proyecto en el sitio de MongoDB website ve a clusters -> collections.

Y podrás ver el producto recién agregado bajo “productos”

Insertar información a MongoDB usando await y async

```
app.post('/api/v1/products', async function(req, res, next) {  
  try{  
    let p1= req.body;  
    const newItem = await CurrentProduct.create(p1);  
    res.status(201).json({  
      status:"success",  
      data:newItem  
    })  
  }  
  catch(err){  
    res.status(400).json({  
      status:"fail",  
      message:"error: " + err  
    })  
  }  
});
```

Hazlo tú mismo 1

1. Crea una carpeta que contenga un archivo app.js
2. Agrega a la carpeta un modelo llamado PersonModel.js que contenga los siguientes campos: **first name, family, city, country, salary**
3. Agrega una nueva función que inserte “person” a la base de datos
4. Corre la aplicación e inserta 3 personas a la base de datos
5. Revisa en mongodb que las filas se hayan agregado

Agrega una nueva función para consultar la información de la base de datos: Vuelve a correr la aplicación y navega a /getall

```
app.get('/api/v1/products', function(req, res, next) {
  CurrentProduct.find({}).then(function(data) {
    res.status(200).json({
      status: "success",
      data: data
    })
  }).catch(err=>{
    res.status(404).json({
      status: "fail",
      message: "error: " + err
    })
  })
})
```

La función “find” está vacía porque no queremos filtrar los resultados, sino que recibirlos todos

Agrega una nueva función para consultar la información de la base de datos: Vuelve a correr la aplicación y navega a localhost:3000/api/v1/products

The screenshot shows a REST API client interface with the following details:

- Method:** GET
- URL:** localhost:3000/api/v1/products
- Headers:** Authorization, Headers (8), Body (●), Pre-request Script, Tests, Settings
- Query Params:** (empty)
- Body:** (empty)
- Cookies:** (empty)
- Send** button

The response is displayed in a table:

KEY	VALUE	DESCRIPTION	o/o	Bulk Edit
Key	Value	Description		

The response body is shown as:

```
1 "status": "success",
2 "data": [
3   {
4     "_id": "61dc1f9e8693b53f3e4f461f",
5     "title": "Red ball",
6     "description": "This is a red ball",
7     "price": 300,
8     "created": "2023-10-09T21:00:00.000Z",
9     "y": 0
10   }
11 ]
12
13 ]
```

Details of the response:
Status: 200 OK
Time: 1013 ms
Size: 411 B
Save Response

Hazlo tú mismo 2

1. Agrega una nueva función a app.js para consultar la información de la base de datos
2. Vuelve a correr la aplicación y navega a localhost:3000/api/v1/products

Agrega una nueva función para consultar la información de la base de datos:
Vuelve a correr la aplicación y navega a localhost:3000/api/v1/products/61dc1f9e8693b5373e4f461f

```
app.get('/api/v1/products/:id', function(req, res, next) {  
  let id = req.params.id  
  CurrentProduct.find({_id:id}) then(function(data){  
    res.status(200).json({  
      status: "success",  
      data: data  
    })  
  }).catch(err=>{  
    res.status(404).json({  
      status: "fail",  
      message: "error: " + err  
    })  
  })  
})
```

Filtrar por id

Agrega una nueva función para consultar la información de la base de datos:
Vuelve a correr la aplicación y navega a localhost:3000/api/v1/products/61dc1f9e8693b53f3e4f461f

Escribe aquí el id de tu objeto

Hazlo tú mismo 3

1. Agrega una nueva función a app.js para consultar una “person” por su _id desde la base de datos
2. Vuelve a correr la aplicación y navega a localhost:3000/api/v1/products/_id (inserta el id)

Agrega una nueva función para consultar la información de la base de datos: Vuelve a correr la aplicación y navega a

```
app.patch('/api/v1/products/:id', function(req, res, next) {
  let id = req.params.id
  CurrentProduct.findByIdAndUpdate(id, req.body, { new: true, runValidators: true })
    .then(function(data) {
      res.status(200).json({
        status: "success",
        data: data
      })
    })
    .catch(err=>{
      res.status(404).json({
        status: "fail",
        message: "error: " + err
      })
    })
})
```

new:true y runValidators:true
significa que retorna un nuevo objeto luego de actualizar parámetros

Más información sobre findByIdAndUpdate:
https://mongoosejs.com/docs/api.html#model_findByUpdate

Agrega una nueva función para consultar la información de la base de datos:
Vuelve a correr la aplicación y navega a localhost:3000/api/v1/products/61dc1f9e8693b5313e4f461f

Actualiza "price" a 400

Escribe aquí el id de tu objeto

The screenshot shows a POSTMAN interface with a PATCH request. The URL is `localhost:3000/api/v1/products/61dc1f9e8693b5313e4f461f`. The Headers tab shows `Content-Type: application/json`. The Body tab contains the following JSON payload:

```
1 {  
2   ...  
3   "title": "Red ball",  
4   "description": "This is a red ball",  
5   "created": "10/30/2022",  
6   "price": 400  
7 }
```

The response shows a status of 200 OK with a response time of 196 ms. The response body is:

```
1 {  
2   "status": "success",  
3   "data": {  
4     "id": "61dc1f9e8693b5313e4f461f",  
5     "title": "Red ball",  
6     "description": "This is a red ball",  
7     "price": 400,  
8     "created": "2023-10-09T21:00:00Z",  
9     "v": 0  
10    }  
11 }
```

Hazlo tú mismo 4

1. Agrega una nueva función a app.js para actualizar “person” por su _id
2. Vuelve a correr la aplicación y navega a localhost:3000/api/v1/products/_----_(elige un id y ejecuta desde postman una solicitud PATCH y actualiza “salary” a 10000 para el id seleccionado)

Agrega una nueva función para consultar la información de la base de datos: Vuelve a correr la aplicación y navega a

```
app.delete('/api/v1/products/:id', function(req, res, next) {
  let id = req.params.id
  CurrentProduct.findByIdAndDelete(id)
    .then(function(data) {
      res.status(404).json({
        status: "success",
        data:null
      })
    })
    .catch(err=>{
      res.status(404).json({
        status: "fail",
        message:"error: "+err
      })
    })
})
```

Agrega una nueva función para consultar la información de la base de datos:
Vuelve a correr la aplicación y navega a localhost:3000/api/v1/products/61dc1f9e8693b53f3e4f461f

Solicitud DELETE: 61dc1f9e8693b53f3e4f461f

Escribe aquí el id de tu objeto

localhost:3000/api/v1/products/61dc1f9e8693b53f3e4f461f

DELETE Headers (8) Body Pre-request Script Tests Settings Cookies

Params Authorization none form-data x-www-form-urlencoded raw binary GraphQl JSON

1 "title": "Red ball",
2 "description": "This is a red ball",
3 "created": "10/10/2023",
4 "price": 400
5
6

404 Not Found 1402 ms 274 B Save Response

Pretty Raw Preview Visualize JSON

1 "status": "success",
2 "data": null
3
4

Hazlo tú mismo 5

1. Agrega una nueva función a app.js para eliminar una “person” por su _id
2. Vuelve a correr la aplicación y navega a localhost:3000/api/v1/products/_----_(elige un id y ejecuta desde postman una solicitud DELETE y elimina la “person” seleccionada)

Actualiza la función "get" `/api/v1/products` para filtrar la información de la base de datos: Vuelve a correr la aplicación y navega a

- En el futuro nos gustaría filtrar por "limit", "page", "sort" y más!
- Por ejemplo:
`localhost:3000/api/v1/products?price=500&sort=price&fields=title,price&limit=10&page=2`
- En la solicitud anterior queremos todos los productos con **price = 500, sort** (ordenar según "price"), en la **page 2** con un **limit** de 10.
- Mongoose no conoce **limit**, **sort**, **fields** ni **page**, por lo que tenemos que quitarlos del queryObj.
 - Limit - El número de resultados en cada página
 - Page - Representa la página que queremos
 - Sort - Ordenar según algún campo
 - Fields - Los campos que queremos consultar

Actualiza la función "get" `/api/v1/products` para filtrar la información de la base de datos: Vuelve a correr la aplicación y navega a

Ejemplo de solicitud: localhost:3000/api/v1/products?price=500&page=2

```
app.get('/api/v1/products', async function(req, res, next) {
  console.log("hello")
  let queryObj = { ...req.query }
  let withOutFields = ['page', 'sort', 'limit', 'fields']
  withOutFields.forEach(e1 => {
    delete queryObj[e1]
  })
  CurrentProduct.find(queryObj).then(function(data) {
    res.status(200).json({
      status: "success",
      data: data
    })
  }).catch(err=>{
    res.status(404).json({
      status: "fail",
      message: "error: " + err
    })
  })
})
```

Quitar
sort,limit,
page,fields de
la solicitud

Actualiza la función "get" `/api/v1/products` para filtrar la información de la base de datos: Vuelve a correr la aplicación y navega a

Ejemplo de solicitud: localhost:3000/api/v1/products?price=500&page=2

```
let queryObj = { ...req.query }  
console.log(queryObj) // { price: '500', page: '2' }
```

Accede al arreglo de
parametros

```
let withOutFields = [ 'page', 'sort', 'limit', 'fields' ]  
withOutFields.forEach(el => {  
  delete queryObj[el]  
}) ;
```

Quitar page,sort,limit y fields del queryObj.

```
console.log(queryObj) // { price: '500' }
```

{ price: '500', page: '2' } ← Antes de withOutFields
{ price: '500' } ← Luego usaremos el campo "page" también

Después de withOutFields

Actualiza la función "get" `/api/v1/products` para filtrar la información de la base de datos: Vuelve a correr la aplicación y navega a

Ejemplo de solicitud: localhost:3000/api/v1/products?price=500&page=2

```
app.get('/api/v1/products', async function(req, res, next) {
  let queryObj = { ...req.query } //
  let withoutFields = ['page', 'sort', 'limit', 'fields']
  withoutFields.forEach(el => {
    delete queryObj[el]
  });
  CurrentProduct.find(queryObj).then(function(data) {
    res.status(200).json({
      status: "success",
      data: data
    })
  }).catch(err=>{
    res.status(404).json({
      status: "fail",
      message: "error: " + err
    })
  })
})
```

Vuelve a correr la aplicación y navega a localhost:3000/api/v1/products?price=500

Todos los productos con "price" 500

The screenshot shows a POSTMAN interface with the following details:

- Method:** GET
- URL:** localhost:3000/api/v1/products?price=500
- Headers:** Authorization, Body (checkbox selected), Pre-request Script, Tests, Settings
- Query Params:** price (checkbox selected, value 500)
- Body:** Cookies, Headers (7), Test Results
- Preview:** JSON, Raw, Preview
- Response:** 200 OK, 199 ms, 413 B, Save Response
- Code:** (A snippet of the JSON response is shown below)

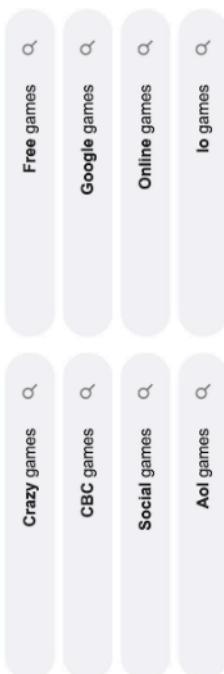
```
[{"id": "61dc55ce0ddc1f89e5d8a7e6", "title": "Blue ball", "description": "This is a blue ball.", "price": 500, "created": "2023-10-09T21:00:00.000Z", "v": 0}]
```

Hazlo tú mismo 6

1. Actualiza la función del ejercicio 2 para filtrar la información de la base de datos
2. Vuelve a correr la aplicación y navega a `localhost:3000/api/v1/persons/` y consulta todas las "person" con "salary" 1000
3. Vuelve a correr la aplicación y navega a `localhost:3000/api/v1/persons/` y consulta todas las "person" con "firstname" Mike

Paginación y límite

La paginación es una manera de separar el contenido de la web en páginas, así se presenta la información de manera limitada y legible. Los resultados de Google es un ejemplo clásico de este mecanismo.

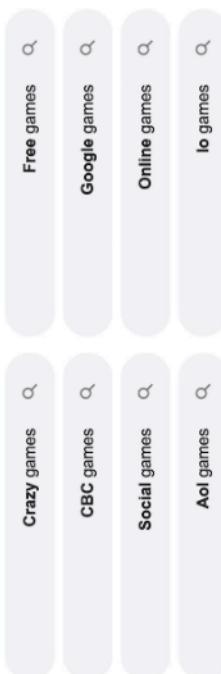


Google >
1 2 3 4 5 6 7 8 9 10

Cada página contiene resultados limitados

Mongoose - paginación y límite

Mongodb no sabe cómo manejar la paginación ni el orden. En las próximas diapositivas aprenderemos cómo manejar paginación, límite, orden y más campos.



Google >
1 2 3 4 5 6 7 8 9 10

Cada página contiene resultados limitados

Leer y filtrar información - Nivel avanzado

Actualiza la función "get" `/api/v1/products` para filtrar la información de la base de datos: Vuelve a correr la aplicación y navega a

```
app.get('/api/v1/products', async function(req, res, next) {
  /phase 1 - filtering
  let queryObj = {...req.query}
  let withoutFields = ['page', 'sort', 'limit', 'fields']
  withoutFields.forEach(el => {
    delete queryObj[el]
  });
});
```

Las solicitudes de Mongodbs no
saben cómo manejar: 'page',
'sort', 'limit' ni 'fields'

Quitemos 'page', 'sort', 'limit', 'fields'
del queryobj

Leer y filtrar información - Nivel avanzado

Actualiza la función “get” /api/v1/products para filtrar la información de la base de datos: Vuelve a correr la aplicación y navega a

```
app.get('/api/v1/products', async function(req, res, next) {  
    //phase 1 - filtering  
    let queryObj = {...req.query}  
    let withOutFields = ['page', 'sort', 'limit', '  
    withOutFields.forEach(e1 => {  
        delete queryObj[e1]  
    });  
    //phase 2 - advance filtering  
    let strQuery = JSON.stringify(queryObj)  
    strQuery = strQuery.replace(/\\b(gt|lte|lt)\\b/g, match => `$$ ${match} `)  
    queryObj = JSON.parse(strQuery)  
    console.log(queryObj)  
    CurrentProduct.find(queryObj).then(function(data) {  
        res.status(200).json({  
            status: "success",  
            data: data  
        })  
    }).catch(err=>{  
        res.status(404).json({  
            status: "fail",  
            message: "error: " + err  
        })  
    })  
})
```

Leer y filtrar información - Nivel avanzado

Actualiza la función "get" /api/v1/products para filtrar la información de la base de datos. Vuelve a correr la aplicación y navega a

```
app.get('/api/v1/products', async function(req, res, next) {  
  //phase 1 - filtering  
  let queryObj = {...req.query}  
  let withOutFields = ['page', 'sort', 'limit', 'fields']  
  withOutFields.forEach(e1 => {  
    delete queryObj[e1]  
  });  
  //phase 2 - advance filtering  
  let strQuery = JSON.stringify(queryObj)  
  strQuery = strQuery.replace(/(\b(gt|lte|lt)\b/g, match => `${match}`)  
  queryObj = JSON.parse(strQuery)  
  console.log(queryObj) // { price: { $gte: '70', $lt: '400' } }  
  
  CurrentProduct.find(queryObj).then(function(data){  
    res.status(200).json({  
      status: "success",  
      data: data  
    })  
  }).catch(err=>{  
    res.status(404).json({  
      status: "fail",  
      message: "error: 😞 : " + err  
    })  
  })  
})
```

Para agregar \$ usaremos una expresión regular

Ahora que tenemos \$ antes de 'gte' y 'lt' podemos ejecutar la función

Filtrar información - Nivel avanzado

Todos los 'products' con 'price' mayor o igual a 70 y menor que 400

The screenshot shows a REST client interface with the following details:

- URL:** localhost:3000/api/v1/products?price[gt]=70&price[lt]=400
- Method:** GET
- Headers:** (8)
- Body:** (empty)
- Params:** price[gt] (checkbox checked), price[lt] (checkbox checked)
- Query Params:** (empty)
- Cookies:** (empty)
- Headers:** (17)
- Test Results:** (empty)
- Buttons:** Save, Send, Bulk Edit, Save Response, 200 OK, 177 ms, 411 B, Save Response, Raw, Preview, Visualize, JSON, v.

The response body is displayed as JSON:

```
1  {
2      "status": "success",
3      "data": [
4          {
5              "id": "64d55b4e0c1f89e5d0a7e1",
6              "title": "Red ball",
7              "description": "This is a red ball",
8              "price": 300,
9              "created": "2023-10-09T21:00:00.000Z",
10             "v": 0
11         }
12     ]
```

Vuelve a correr la aplicación y navega a localhost:3000/api/v1/products?price[gt]=70&price[lt]=400

Hazlo tú mismo 7

1. Actualiza la función del ejercicio 5 para usar filtros avanzados de la base de datos.
2. Vuelve a correr la aplicación y navega a `localhost:3000/api/v1/persons/` y consulta todas las 'persons' con 'salary' mayor a 1000
3. Vuelve a correr la aplicación y navega a `localhost:3000/api/v1/persons/` y consulta todas las 'persons' con 'salary' menor a 1000
4. Vuelve a correr la aplicación y navega a `localhost:3000/api/v1/persons/` y agrega parámetros a la solicitud que consulta todas las 'persons' con 'salary' entre 1000 y 2000

Ordenar la información

Actualiza la función "get" `/api/v1/products` para filtrar la información de la base de datos: Vuelve a correr la aplicación y navega a

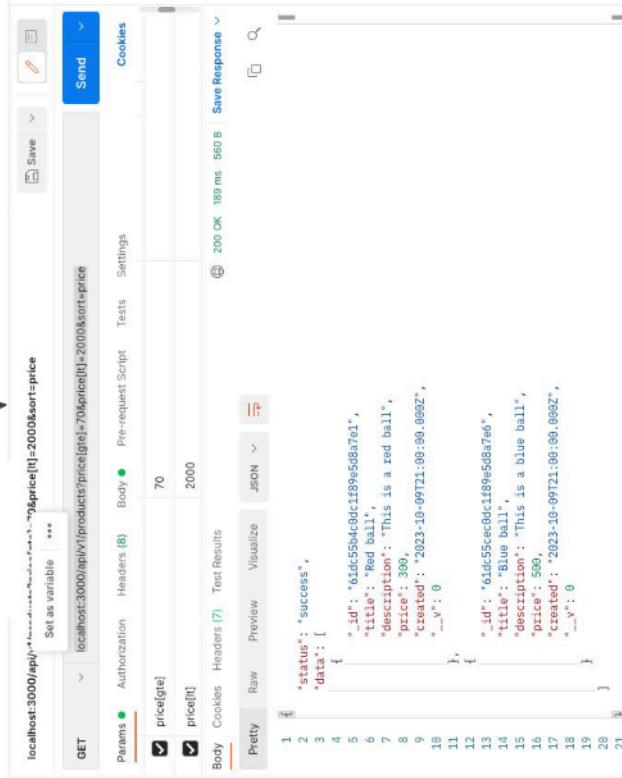
```
app.get('/api/v1/products', async function(req, res, next) {
  //phase 1 - filtering
  let queryobj = {...req.query}
  let withoutFields = ['page', 'sort', 'limit', 'fields']
  withoutFields.forEach(el => {
    delete queryobj[el]
  });
  //phase 2 - advance filtering
  let strquery = JSON.stringify(queryobj)
  strquery = strquery.replace(/\\b(gt|lte|lt)\\b/g, match => `$$ ${match} `)
  queryobj = JSON.parse(strquery)
  let sort="";
  if(req.query.sort){
    sort = req.query.sort.split(',') .join('') // add more sorts
  }
  currentProduct.find(queryobj) .sort(sort) .then(function(data) {
    res.status(200) .json({
      status:"success",
      data :data
    })
  }) .catch(err=>{
    res.status(404) .json({
      status:"fail",
      message:"error: " + err
    })
  })
})
```

Si se especifica un orden (sort), es posible que incluya más de uno

Sort data

Vuelve a correr la aplicación y navega a `localhost:3000/api/v1/products?price[gt]=70&price[lt]=2000&sort=price`

Todos los 'products' con 'price' mayor o igual a 70, menor a 500 y orden de acuerdo a 'price'



```
localhost:3000/api/v1/products?price[gt]=70&price[lt]=2000&sort=price
Set as variable
GET Authorization Headers (8) Body Pre-request Script Tests Settings Cookies
price[gt] price[lt]
Body Cookies Headers (7) Test Results
Pretty Raw Preview Visualize JSON Save Response
200 OK 189 ms 560 B
1 {
  "status": "success",
  "data": [
    {
      "_id": "61dc55b4fc0dc1f89e5d8a761",
      "title": "Red ball",
      "description": "This is a red ball",
      "price": 399,
      "created": "2023-10-09T21:00:00.099Z",
      "___v": 0
    },
    {
      "_id": "61dc55b4fc0dc1f89e5d8a766",
      "title": "Blue ball",
      "description": "This is a blue ball",
      "price": 500,
      "created": "2023-10-09T21:00:00.099Z",
      "___v": 0
    }
  ]
}
```

Hazlo tú mismo 8

1. Actualiza la función del ejercicio 6 y agrega la opción de orden.
2. Vuelve a correr la aplicación y navega a localhost:3000/api/v1/persons/ y consulta todas las 'persons' ordenadas de acuerdo a 'salary'

Quitar campos de la respuesta

Actualiza la función “get” /api/v1/products para filtrar la información de la base de datos

```
app.get('/api/v1/products', async function(req, res, next) {  
  //phase 1 - filtering  
  let queryObj = {...req.query}  
  let withOutFields = ['page', 'sort', 'limit', 'fields']  
  withOutFields.forEach(e1 => {  
    delete queryObj[e1]  
  }) ;  
  //phase 2 - advance filtering  
  let strQuery = JSON.stringify(queryObj)  
  strQuery = strQuery.replace(/\b(gt|gte|lte)\b/g, match =>`$${match}`)  
  queryObj = JSON.parse(strQuery)  
  console.log(queryObj)  
  let sort="";  
  let selected = "";  
  if(req.query.sort){  
    sort = req.query.sort.split(',').join(' ') // add more sorts  
  }  
});
```

Continúa ->

Quitar campos de la respuesta

```
if(req.query.fields){  
    selected = req.query.fields.split(',') .join(' ') // show fields  
    // -fields will remove a fields  
}  
  
CurrentProduct.find(queryObj) .select(selected) .sort(sort) .then(function(data) {  
    res.status(200) .json({  
        status:"success",  
        data:data  
    })  
}) .catch(err=>{  
    res.status(404) .json({  
        status:"fail",  
        message:"error: " + err  
    })  
})
```

Mostrar los campos

Select - qué campos mostrar

Quitar campos de la respuesta

Actualiza la función "get" para consultar la información de la base de datos Vuelve a correr la aplicación y navega a

localhost:3000/api/v1/advancefilterproducts?price[gte]=70&price[lte]=200&sort=price&fields=title,price

```
localhost:3000/api/v1/advancefilterproducts?price[gte]=70&price[lte]=200&sort=price&fields=title,price

GET
localhost:3000/api/v1/products?price[gte]=70&price[lte]=600&sort=price&fields=title,price

Params • Authorization Headers (8) Body • Pre-request Script Tests Settings
Cookies

Fields
Key Value Description
title,price
Value

Body Cookies Headers (7) Test Results
Headers (7)
200 OK 180 ms 398 B Save Response ▾
Save ▾ Send ▾
Cookies

Pretty Raw Preview Visualize JSON ▾
Raw Visualize JSON ▾

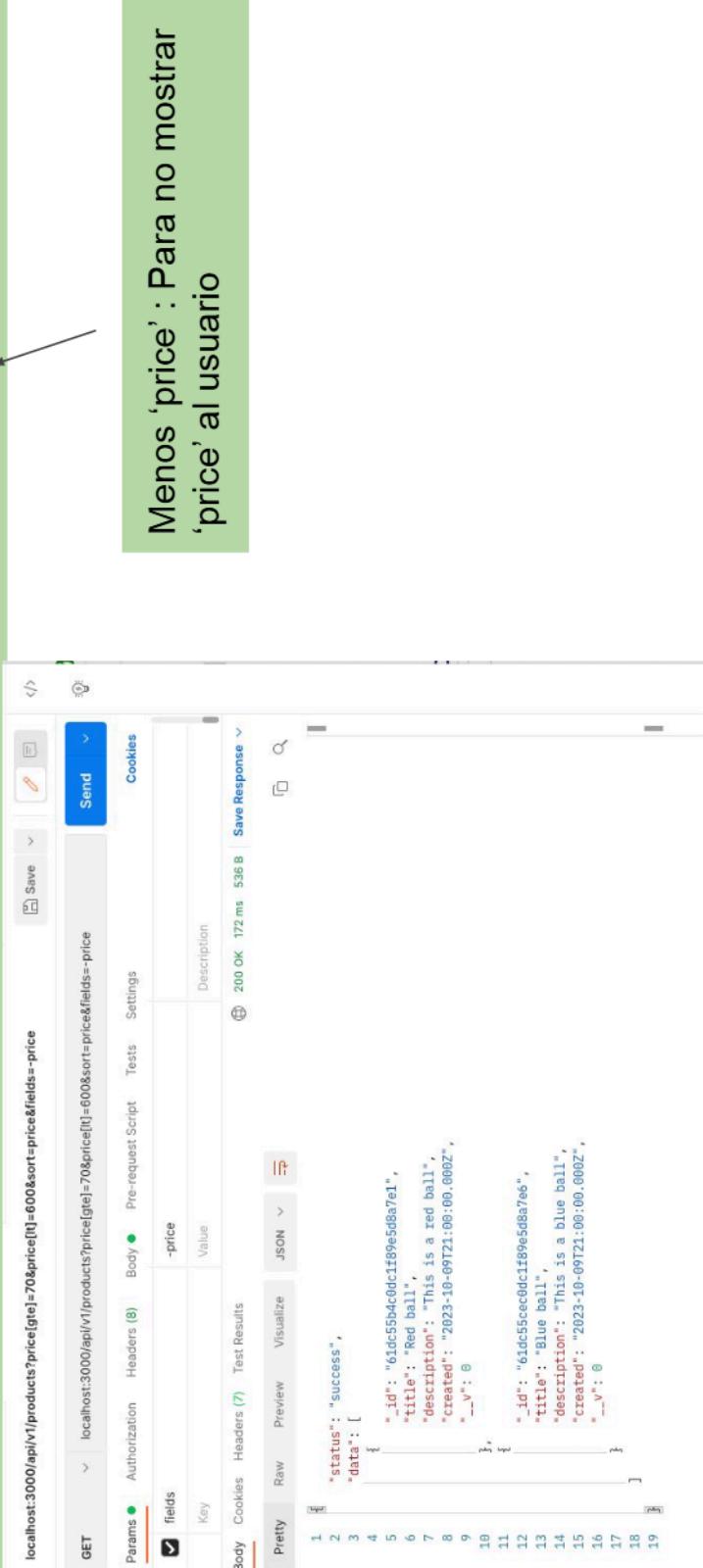
1
{
  "status": "success",
  "data": [
    {
      "_id": "61dc55b4c0dc1f89e5cd8a7e1",
      "title": "Red ball",
      "price": 390
    },
    {
      "_id": "61dd55cecd0c1f89e5cd8a7e6",
      "title": "Blue ball",
      "price": 590
    }
  ]
}

2022 © Wawiwa Tech | Confidential
```

Mostrar sólo: 'title' y 'price'

Quitar campos de la respuesta

Actualiza la función "get" para consultar la información de la base de datos Vuelve a correr la aplicación y navega a localhost:3000/api/v1/advancefilterproducts?price[gt]=70&price[lte]=200&sort=price&fields=-price



The screenshot shows a Postman request to `localhost:3000/api/v1/products?price[gt]=70&price[lte]=200&sort=price&fields=-price`. The response body is displayed in JSON format, showing a success status and a data array. The 'price' field is explicitly listed as a removed field in the response.

```
1  "status": "success",
2  "data": [
3    {
4      "_id": "61dc55b4c0dc189e5d8a7e1",
5      "title": "Red ball",
6      "description": "This is a red ball",
7      "created": "2023-10-09T21:00:00.000Z",
8      "__v": 0
9    }
10   ]
11
12   {
13     "_id": "61dc55c0dd189e5d8a7e6",
14     "title": "Blue ball",
15     "description": "This is a blue ball",
16     "created": "2023-10-09T21:00:00.000Z",
17     "__v": 0
18   }
19 ]
```

Menos 'price' : Para no mostrar 'price' al usuario

Paginación

Actualiza la función “get” `/api/v1/products` para filtrar la información de la base de datos

```
app.get('/api/v1/products',async function(req, res, next) {  
    //phase 1 - filtering  
    let queryobj = {...req.query}  
    let withoutFields = ['page', 'sort', 'limit', 'fields']  
    withoutFields.forEach(el => {  
        delete queryobj[el]  
    })  
    //phase 2 - advance filtering  
    let strquery = JSON.stringify(queryobj)  
    strquery = strquery.replace (/\\b(gte|gt|lte|lt)\\b/g,match => `${match}`)  
    queryobj = JSON.parse(strquery)  
    console.log(queryobj)  
    let sort='';  
    let selected = '';  
    if (req.query.sort){  
        sort = req.query.sort.join(' ') // add more sorts  
    }  
    if (req.query.fields){  
        selected = req.query.fields.split(',') .join(' ') // show fields  
    }  
})
```

Continúa ->

Paginación

Actualiza la función "get" /api/v1/products para filtrar la información de la base de datos

```
let limit = req.query.limit || 100
let page = req.query.page || 1
let skip = (page-1)*limit
let documents = await CurrentProduct.countDocuments()
if(skip>documents){
  res.status(404).json({
    status:"fail",
    data:"no data on this page and limit"
  })
  return
}
```

Límite por página - por defecto 100

Página seleccionada - por defecto 1

Retornar 404 si la página no existe

Paginación

Actualiza la función "get" `/api/v1/products` para filtrar la información de la base de datos

```
CurrentProduct.find(queryObj) .select(selected) .sort(sort) .then(function(data) {
  res.status(200) .json({
    status:"success",
    data:data
  })
}) .catch(err=>{
  res.status(404) .json({
    status:"fail",
    message:"error: " + err
  })
})
```

Página

Agrega una nueva función para consultar información de la base de datos: Vuelve a correr la aplicación y navega a `localhost:3000/api/v1/products?sort=price&page=1&limit=100`

```
GET      localhost:3000/api/v1/products?sort=price&page=1&limit=100 ...
Params  ● Authorization Headers (8) Body ● Pre-request Script Tests Settings
Cookies limit 100 ↴
Body Cookies Headers (7) Test Results
Key Value
Pretty Raw Preview Visualize JSON ↴
Description
Save Response ↴
Send
Cookies
Q
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
1 "status": "success",
2 [
3   {
4     "id": "61dc55bd4c06c1f89e5d8a7e1",
5     "title": "Red ball",
6     "description": "This is a red ball",
7     "price": 399,
8     "created": "2023-10-99T21:00:00.000Z",
9     "___v": 0
10   },
11   {
12     "id": "61dc55ce00dc1f89e5d8a7e6",
13     "title": "Blue ball",
14     "description": "This is a blue ball",
15     "price": 500,
16     "created": "2023-10-99T21:00:00.000Z",
17     "___v": 0
18   },
19   {
20     "id": "61dc55cc00dc1f89e5d8a7e5",
21     "title": "Yellow ball",
22     "description": "This is a yellow ball",
23     "price": 450,
24     "created": "2023-10-99T21:00:00.000Z",
25     "___v": 0
26   }
]
```

AGGREGATE (AGREGACIÓN)

Operación ‘aggregate’ en mongodb

La agregación procesa múltiples documentos y retorna resultados calculados. Puedes usar la agregación para:

- Agrupar valores de múltiples documentos.
- Realizar operaciones sobre los datos agrupados para retornar un resultado único.
- Analizar cambios en los datos a lo largo del tiempo.

Aggregación - mongodb

La fase **\$match**:

- Filtra los documentos con estado urgente
- Retorna los documentos filtrados a la fase **\$group**

La fase **\$group**:

- Agrupa los documentos por su 'productName'.
- Usa **\$sum** para calcular la cantidad total de cada 'productName', que es guardada en el campo 'sumQuantity' y retornada por la agregación.

Agrega una nueva función a 'statistic' /api/v1/products/get/statistic: Vuelve a correr la aplicación y navega a

```
app.get('/api/v1/products/get/statistic', function(req, res, next) {  
    CurrentProduct.aggregate([  
        {  
            $match : {price:{$gt:30}}  
        },  
        {  
            $group:{  
                id:null,  
                count:{$sum:1},  
                avgPrice:{$avg:'$price'},  
                minPrice:{$min:'$price'},  
                maxPrice:{$max:'$price'}  
            }  
        },  
        {  
            $sort:{avgPrice:1} // 1 its asc, 0 its decn  
        }  
    ])  
})
```

La solicitud - Por ejemplo todos los 'products' con 'price' mayor a 30

La agrupación incluye la estadística que queremos

```
id:null,  
count:{$sum:1},  
avgPrice:{$avg:'$price'},  
minPrice:{$min:'$price'},  
maxPrice:{$max:'$price'}
```

)

,

```
$sort:{avgPrice:1} // 1 its asc, 0 its decn
```

)
\$sort = ordenar de acuerdo al campo : en orden
descendente o ascendente

\$sum = la suma de la agrupación

\$avg = el promedio de la agrupación

\$min = el mínimo valor de 'price'

\$max = el máximo valor de 'price'

Agrega una nueva función a 'statistic' /api/v1/products/get/statistic: Vuelve a correr la aplicación y navega a

```
localhost:3000/api/v1/products/get/statistic
GET      localhost:3000/api/v1/products/get/statistic
Send >

.then(function(data) {
  console.log(data)
  res.status(200).json({
    status: "success",
    data: data
  })
}) .catch(err=>{
  res.status(404).json({
    status: "fail",
    message: "error: " + err
  })
})
}

  KEY          VALUE
  Key          Value
  Body          Headers (8)
  Authorization
  Headers (8)
  Body ●  pre-request Script
  Tests
  Settings
  Cookies
  Description
  Description
  200 OK 185 ms 332 B Save Response >
  200 OK 185 ms 332 B Save Response >
  Body          Headers (7)
  Cookies
  Headers (7)
  Test Results
  Pretty
  Raw
  Preview
  Visualize
  JSON >
```

```
1
2   "status": "success",
3   "data": [
4     {
5       "id": null,
6       "count": 2,
7       "avgPrice": 400,
8       "minPrice": 300,
9       "maxPrice": 500
10    }
11  ]
12 ]
```

Hazlo tú mismo 9

1. Agrega una función que muestre estadísticas sobre las ‘persons’. La función debe retornar el ‘salary’ promedio, el mínimo y el máximo
2. Vuelve a correr la aplicación y navega a `localhost:3000/api/v1/get/statistic/` y consulta los valores promedio, mínimo y máximo de ‘salary’

middleware mongoose

Middleware (también llamado pre y post *hooks*) son funciones que pasan el control durante la ejecución de funciones asíncronas. Middleware se especifica en el nivel de esquema y es útil para escribir plugins (“complementos”).

Mongoose tiene 5 tipos de middleware:

1. document middleware,
2. query middleware
3. aggregate middleware,
4. model middleware,

document middleware mongodb

Document middleware es compatible con las siguientes funciones de documentos. En las funciones de middleware de documentos, `this` se refiere al documento.

- `validate`
- `save`
- `remove`
- `updateOne`
- `deleteOne`
- `init` (nota: los hooks init son síncronos)

Query middleware mongdb

Query middleware es compatible con las siguientes funciones de Model y Query. En las funciones middleware de query, **this** se refiere a la solicitud.

- count
- countDocuments
- deleteMany
- deleteOne
- estimatedDocumentCount
- find
- findOne
- findOneAndDelete
- findOneAndRemove
- findOneAndReplace
- findOneAndUpdate
- remove
- replaceOne
- update
- updateOne
- updateMany

aggregate middleware mongdb

- Aggregate middleware es para MyModel.aggregate(). Los middleware de Aggregate se ejecutan cuando llamas a exec() en un objeto agregado. En los middleware de aggregate, **this** se refiere al objeto de agregación.

model middleware mongdb

- Model middleware es compatible con las siguientes funciones del model. En las funciones middleware the modelo **this** se refiere al model.

`insertMany`

Middleware mongoose

All middleware types support `pre` and `post` hooks. How `pre` and `post` hooks work is described in more detail below.

Note: If you specify `schema.pre('remove')`, Mongoose will register this middleware for `doc.remove()` by default. If you want to your middleware to run on `Query.remove()` use `schema.pre('remove', { query: true, document: false }, fn)`.

Note: Unlike `schema.pre('remove')`, Mongoose registers `updateOne` and `deleteOne` middleware on `Query#updateOne()` and `Query#deleteOne()` by default. This means that both `doc.updateOne()` and `Model.updateOne()` trigger `updateOne` hooks, but `this` refers to a query, not a document. To register `updateOne` or `deleteOne` middleware as document middleware, use `schema.pre('updateOne', { document: true, query: false })`.

Note: The `create()` function fires `save()` hooks.



EXAMPLES

Agrega un middleware al esquema de MongoDB

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;
const ProductSchema = new Schema({
  title: {
    type: String,
    required: [true, 'A product must have title'],
    unique: true,
    trim: true
  },
  description: {
    type: String,
    minlength: [5, 'Description is minimum 20 characters'],
    maxlength: [1000, 'Description is maximum 1000 characters']
  },
  price: {
    type: Number,
    required: [true, 'A product must have price'],
    min: [0, 'price must be above 0'],
    max: [10000, 'price must be below 10000']
  },
  created: Date
});

module.exports = mongoose.model('product', ProductSchema);
```

Document middleware - pre save

Este middleware se ejecuta antes de los comandos de "save" o "create"

```
ProductSchema.pre('save', function(data, next) {  
    console.log(data)  
    next();  
})
```

Document middleware - pre save

Este middleware se ejecuta antes de los comandos de "save" o "create"

```
ProductSchema.pre('save', function(data, next) {  
    console.log(data)  
    next();  
})
```

Query middleware - pre save

Este middleware se ejecuta antes del comando "find"

```
//query middleware pre - todas las solicitudes que empiezan con  
"find"
```

```
ProductSchema.pre('find', function(next) {  
  this.find({price : {$gt: 0}}).  
  next();  
});
```

Query middleware - pre save

Si quieres todas las solicitudes que empiecen con "find" usa "Regex"

Este middleware corre antes de los comandos "save" y "create"

//query middleware pre - todas las solicitudes que empiezan con "find"

```
ProductSchema.pre(/^find/, function(next) {  
    this.find({price : {$gt: 0}}).  
    next();  
});
```

Query middleware - post find

Este middleware se ejecuta antes del comando "find"

```
//query middleware pre - todas las solicitudes que empiezan con  
"find"
```

```
ProductSchema.post('find', function(next) {  
  this.find({price : {$gt:0}})()  
  next();  
});
```

Query middleware - post save

Si quieres todas las solicitudes que empiezan con "find", usa "Regex"

Este middleware corre después de los comandos "save" o "create"

//query middleware post - todas las solicitudes que empiezan con "find"

```
ProductSchema.post(/^find/, function(next) {  
    this.find({price : {$gt: 0}}).  
    next();  
});
```

aggregate middleware - post save

This middleware run before 'aggregate' command

```
//query middleware post - all the query start with find
ProductSchema.pre('aggregate', function(next) {
  console.log(this.pipeline())
  this.pipeline().unshift({$match:{price:{$gt:0}}}) // add to array
  next();
});
```

wawiwa

¿Preguntas?