

## CRUD - OPERATION

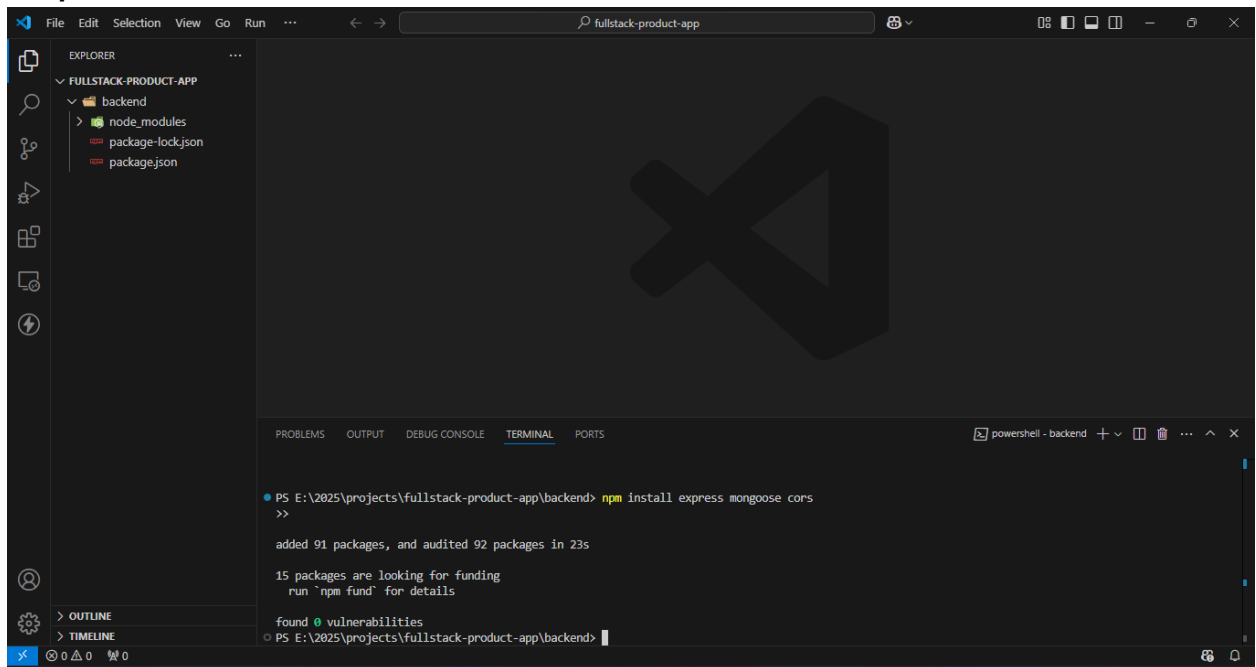
### Steps:

#### 1. Open CMD:

- `mkdir fullstack-product-app`
- `cd fullstack-product-app`

- `mkdir backend (run the commands)`
  - `cd backend`
  - `npm init -y`
  - `npm install express mongoose cors`
  - `npm install config`
  - `npm install dotenv`

### Output:



The screenshot shows the Visual Studio Code interface with the Explorer panel on the left displaying the project structure: `fullstack-product-app` > `backend` > `node_modules`, `package-lock.json`, and `package.json`. The Terminal panel at the bottom shows the execution of the `npm install express mongoose cors` command in a PowerShell session. The output indicates that 91 packages were added and 92 packages were audited in 23 seconds. It also shows that 15 packages are looking for funding and that no vulnerabilities were found.

```
PS E:\2025\projects\fullstack-product-app\backend> npm install express mongoose cors
>>

added 91 packages, and audited 92 packages in 23s

15 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS E:\2025\projects\fullstack-product-app\backend>
```

## 2. Create a Folders:

- mkdir controllers
  - touch productControllers.js

### // Code: productControllers.js:

```
const Product = require('../models/productModels');

// Create a new product
exports.createProduct = async (req, res) => {
  try {
    const product = await Product.create(req.body);
    res.status(201).json({
      success: true,
      data: product
    });
  } catch (error) {
    res.status(400).json({
      success: false,
      error: error.message
    });
  }
};

// Get all products
exports.getAllProducts = async (req, res) => {
  try {
    const products = await Product.find();
    res.status(200).json({
      success: true,
      count: products.length,
      data: products
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      error: error.message
    });
  }
};

// Get single product by ID
exports.getProductById = async (req, res) => {
  try {
```

```
const product = await Product.findById(req.params.id);
if (!product) {
  return res.status(404).json({
    success: false,
    error: 'Product not found'
  });
}
res.status(200).json({
  success: true,
  data: product
});
} catch (error) {
  res.status(500).json({
    success: false,
    error: error.message
  });
}
};
```

```
// Update product by ID
exports.updateProduct = async (req, res) => {
  try {
    const product = await Product.findByIdAndUpdate(
      req.params.id,
      req.body,
      {
        new: true,
        runValidators: true
      }
    );
    if (!product) {
      return res.status(404).json({
        success: false,
        error: 'Product not found'
      });
    }
    res.status(200).json({
      success: true,
      data: product
    });
  } catch (error) {
    res.status(400).json({
      success: false,
```

```
        error: error.message
      });
    }
  };

  // Delete product by ID
  exports.deleteProduct = async (req, res) => {
    try {
      const product = await Product.findByIdAndDelete(req.params.id);
      if (!product) {
        return res.status(404).json({
          success: false,
          error: 'Product not found'
        });
      }
      res.status(200).json({
        success: true,
        data: {}
      });
    } catch (error) {
      res.status(500).json({
        success: false,
        error: error.message
      });
    }
  };
};
```

### 3. Create a Folders:

- `mkdir models`
  - `touch productModels.js`

**// Code: productModels.js:**

```
const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({
  productName: {
    type: String,
    required: [true, 'Product name is required'],
    trim: true
  },
  description: {
    type: String,
    required: [true, 'Product description is required'],
    trim: true
  },
  category: {
    type: String,
    required: [true, 'Product category is required'],
    enum: ['Mobile', 'Laptop', 'Tablet'],
    trim: true
  },
  quantity: {
    type: Number,
    required: [true, 'Product quantity is required'],
    min: [0, 'Quantity cannot be negative']
  },
  price: {
    type: Number,
    required: [true, 'Product price is required'],
    min: [0, 'Price cannot be negative']
  }
}, {
  timestamps: true
});

module.exports = mongoose.model('Product', productSchema);
```

#### 4. Create a Folders:

- mkdir routes
  - touch productRoutes.js

##### // Code: productRoutes.js:

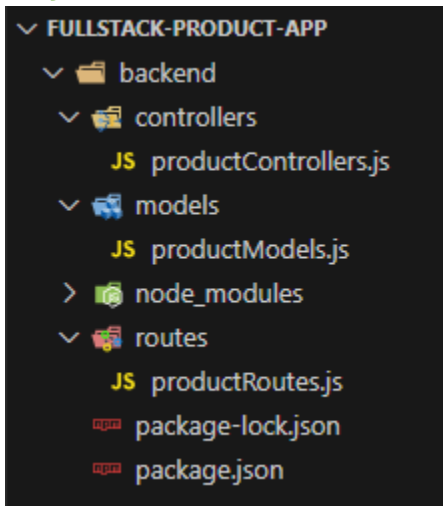
```
const express = require('express');
const router = express.Router();
const {
  createProduct,
  getAllProducts,
  getProductById,
  updateProduct,
  deleteProduct
} = require('../controllers/productControllers');

router.route('/')
  .get(getAllProducts)
  .post(createProduct);

router.route('/:id')
  .get(getProductById)
  .put(updateProduct)
  .delete(deleteProduct);
```

```
module.exports = router;
```

#### Project Structure:



### 5. Create .env file:

```
touch .env
```

```
// .env file:  
PORT=3000  
MONGODB_URI=mongodb://localhost:27017/productDB
```

### 6. Create Three Files in backend:

```
- config.js  
- index.js  
- .gitignore
```

### 7. Codes:

```
//config.js  
  
require('dotenv').config();  
  
module.exports = {  
  port: process.env.PORT || 3000,  
  mongoUri: process.env.MONGODB_URI ||  
  'mongodb://localhost:27017/productDB'  
};
```

```
//index.js  
  
// run `node index.js` in the terminal  
console.log(` Hello Node.js v${process.versions.node}!` );
```

```
//.gitignore  
  
.node_modules  
.bolt
```

## 8. Create server.js file:

```
//server.js
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const config = require('./config');
const productRoutes = require('./routes/productRoutes');

const app = express();

// Middleware
app.use(cors());
app.use(express.json());
// Connect to MongoDB
mongoose.connect('mongodb://127.0.0.1:27017/productDB', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  serverSelectionTimeoutMS: 15000
})
.then(() => console.log('MongoDB connected successfully'))
.catch(err => console.log('MongoDB connection error:', err));

// Routes
app.use('/api/products', productRoutes);

// Basic route
app.get('/', (req, res) => {
  res.json({ message: 'Welcome to Product Management API' });
});

// Error handling middleware
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({
    success: false,
    error: 'Something went wrong!'
  });
});

// Start server
app.listen(config.port, () => {
  console.log(` Server is running on port ${config.port} `);
});
```



### 9. Open MongoDB Compass:

- **Connect** the localhost 127.0.0.1:27017

### 10. Open Terminal Run Command:

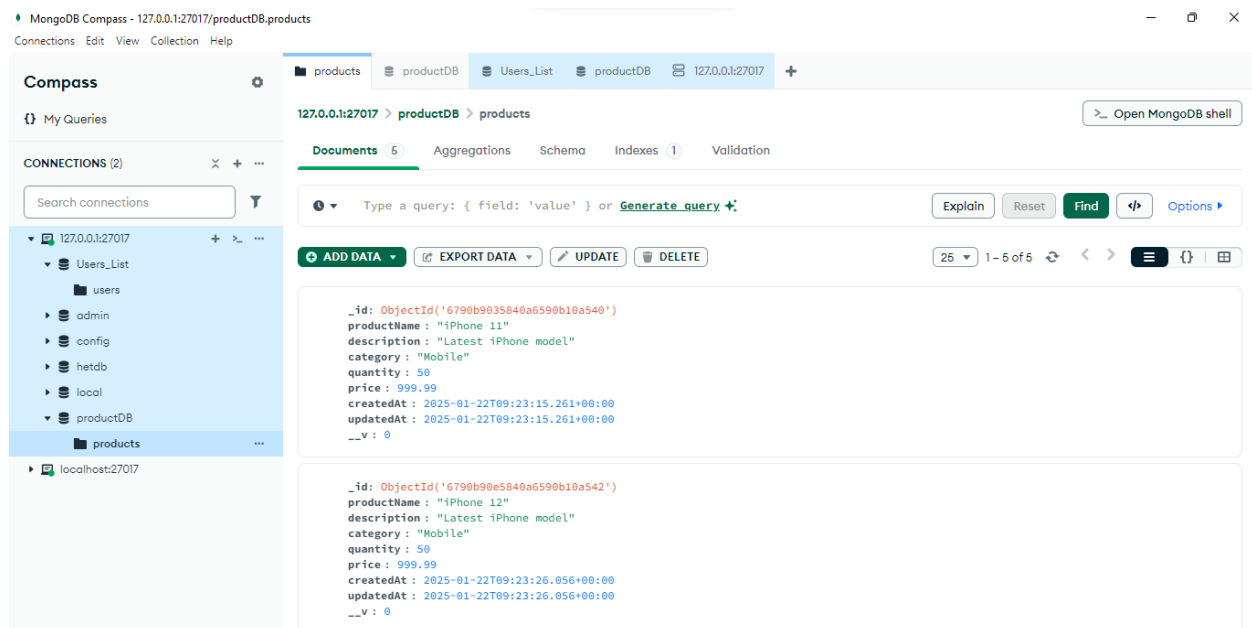
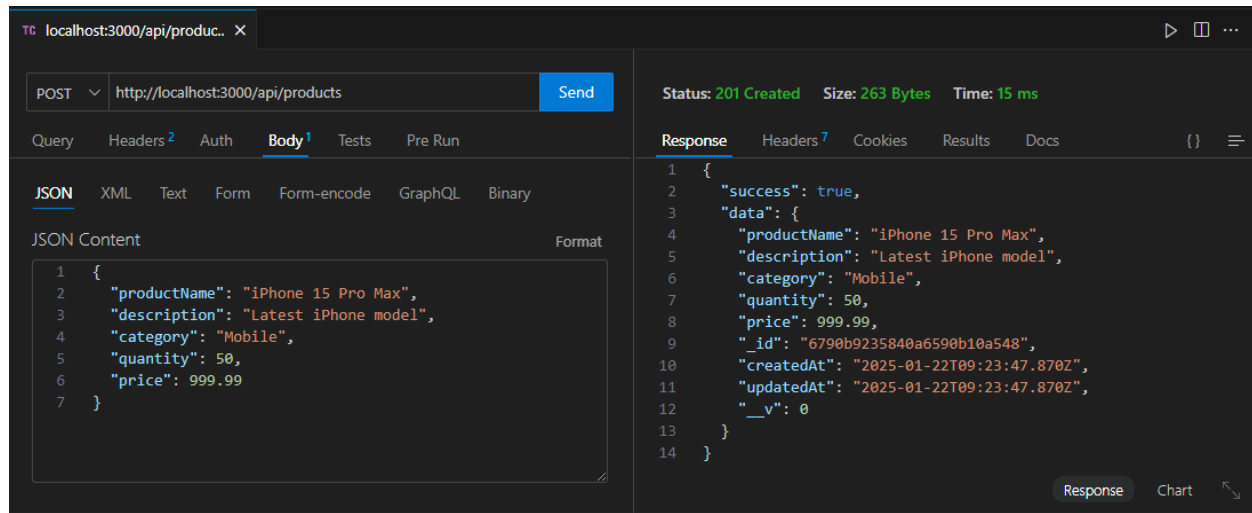
```
node server.js
```

```
PS E:\2025\projects\fullstack-product-app\backend> node server.js
(node:2828) [MONGODB DRIVER] Warning: useUrlParser is a deprecated option: use
rsion
(Use `node --trace-warnings ...` to show where the warning was created)
(node:2828) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option:
for version
Server is running on port 3000
MongoDB connected successfully
█
```

## 11. Open ThunderClient:

✓ **Create new product:**

**POST** **http://localhost:3000/api/products**



✓ **Get all products:**

**GET** **http://localhost:3000/api/products**

The screenshot shows a REST client interface with a GET request to `http://localhost:3000/api/products` sent successfully. The response is a JSON array of two product objects. A blue arrow points from the 'Body' tab to the 'Response' tab.

**Request:**

```
GET http://localhost:3000/api/products
```

**Response:**

```
{
  "success": true,
  "count": 5,
  "data": [
    {
      "_id": "6790b9035840a6590b10a540",
      "productName": "iPhone 11",
      "description": "Latest iPhone model",
      "category": "Mobile",
      "quantity": 50,
      "price": 999.99,
      "createdAt": "2025-01-22T09:23:15.261Z",
      "updatedAt": "2025-01-22T09:23:15.261Z",
      "__v": 0
    },
    {
      "_id": "6790b90e5840a6590b10a542",
      "productName": "iPhone 12",
      "description": "Latest iPhone model",
      "category": "Mobile",
      "quantity": 50,
      "price": 999.99,
      "createdAt": "2025-01-22T09:23:26.056Z",
      "updatedAt": "2025-01-22T09:23:26.056Z",
      "__v": 0
    }
  ]
}
```

The screenshot shows the MongoDB Compass interface. The left sidebar shows the database structure with 'productDB' selected. The main panel shows the 'products' collection with two documents. The 'Documents' tab is active, displaying the JSON data for each product.

**Documents:**

```
{
  "_id": "6790b9035840a6590b10a540",
  "productName": "iPhone 11",
  "description": "Latest iPhone model",
  "category": "Mobile",
  "quantity": 50,
  "price": 999.99,
  "createdAt": "2025-01-22T09:23:15.261+00:00",
  "updatedAt": "2025-01-22T09:23:15.261+00:00",
  "__v": 0
},
{
  "_id": "6790b90e5840a6590b10a542",
  "productName": "iPhone 12",
  "description": "Latest iPhone model",
  "category": "Mobile",
  "quantity": 50,
  "price": 999.99,
  "createdAt": "2025-01-22T09:23:26.056+00:00",
  "updatedAt": "2025-01-22T09:23:26.056+00:00",
  "__v": 0
}
```

✓ Get product by ID:

**GET** `http://localhost:3000/api/products/6790b9235840a6590b10a548`

localhost:3000/api/produ... X

GET `http://localhost:3000/api/products/6790b9235840a6590b10a548` Send

Status: 200 OK Size: 263 Bytes Time: 37 ms

Response Headers Cookies Results Docs

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {
2   "productName": "iPhone 15 Pro Max",
3   "description": "Latest iPhone model",
4   "category": "Mobile",
5   "quantity": 50,
6   "price": 999.99
7 }
```

Response

```
1 {
2   "success": true,
3   "data": {
4     "_id": "6790b9235840a6590b10a548",
5     "productName": "iPhone 15 Pro Max",
6     "description": "Latest iPhone model",
7     "category": "Mobile",
8     "quantity": 50,
9     "price": 999.99,
10    "createdAt": "2025-01-22T09:23:47.870Z",
11    "updatedAt": "2025-01-22T09:23:47.870Z",
12    "__v": 0
13  }
14 }
```

MongoDB Compass - 127.0.0.1:27017/productDB.products

Connections Edit View Collection Help

Compass

My Queries

CONNECTIONS (2)

127.0.0.1:27017

- Users\_List
- productDB
  - products

127.0.0.1:27017 > productDB > products

Documents 5 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or Generate query

ADD DATA EXPORT DATA UPDATE DELETE

25 1 - 5 of 5

```
{
  "_id": ObjectId("6790b9145840a6590b10a546"),
  "productName": "iPhone 14",
  "description": "Latest iPhone model",
  "category": "Mobile",
  "quantity": 50,
  "price": 999.99,
  "createdAt": "2025-01-22T09:23:32.594+00:00",
  "updatedAt": "2025-01-22T09:23:32.594+00:00",
  "__v": 0
}
```

```
{
  "_id": ObjectId("6790b9235840a6590b10a548"),
  "productName": "iPhone 15 Pro Max",
  "description": "Latest iPhone model",
  "category": "Mobile",
  "quantity": 50,
  "price": 999.99,
  "createdAt": "2025-01-22T09:23:47.870+00:00",
  "updatedAt": "2025-01-22T09:23:47.870+00:00",
  "__v": 0
}
```

### ✓ Update product:

**PUT** **http://localhost:3000/api/products/6790b9235840a6590b10a548**

The screenshot shows a REST client interface with a PUT request to `http://localhost:3000/api/products/6790b9235840a6590b10a548`. The request body is a JSON object representing an iPhone 15 Pro Max. The response is a 200 OK status with a JSON body indicating success and providing details about the updated product.

```
PUT http://localhost:3000/api/products/6790b9235840a6590b10a548
```

JSON Content:

```
1 {
2   "productName": "iPhone 15 Pro Max",
3   "description": "Latest iPhone model",
4   "category": "Mobile",
5   "quantity": 50,
6   "price": 999.99
7 }
```

Response:

```
1 {
2   "success": true,
3   "data": {
4     "_id": "6790b9235840a6590b10a548",
5     "productName": "iPhone 15 Pro Max",
6     "description": "Latest iPhone model",
7     "category": "Mobile",
8     "quantity": 50,
9     "price": 999.99,
10    "createdAt": "2025-01-22T09:23:47.870Z",
11    "updatedAt": "2025-01-22T09:41:37.463Z",
12    "__v": 0
13  }
14 }
```

### After:

The screenshot shows the MongoDB Compass interface. The left sidebar displays the database structure, including the `products` collection. The main panel shows the `products` collection with two documents. The second document, which has been updated, is highlighted with a blue arrow pointing to its `_id` field.

Documents:

```
1 {
2   "_id": ObjectId('6790b9145840a6590b10a546'),
3   "productName": "iPhone 14",
4   "description": "Latest iPhone model",
5   "category": "Mobile",
6   "quantity": 50,
7   "price": 999.99,
8   "createdAt": "2025-01-22T09:23:32.594+00:00",
9   "updatedAt": "2025-01-22T09:23:32.594+00:00",
10  "__v": 0
11 }
12
13 {
14   "_id": ObjectId('6790b9235840a6590b10a548'),
15   "productName": "iPhone 15 Pro Max",
16   "description": "Latest iPhone model",
17   "category": "Mobile",
18   "quantity": 50,
19   "price": 999.99,
20   "createdAt": "2025-01-22T09:23:47.870+00:00",
21   "updatedAt": "2025-01-22T09:41:37.463+00:00",
22   "__v": 0
23 }
```

✓ **Delete product:**

<b>DELETE</b>	<b><code>http://localhost:3000/api/products/6790b9235840a6590b10a548</code></b>
---------------	---

**Before Delete:**

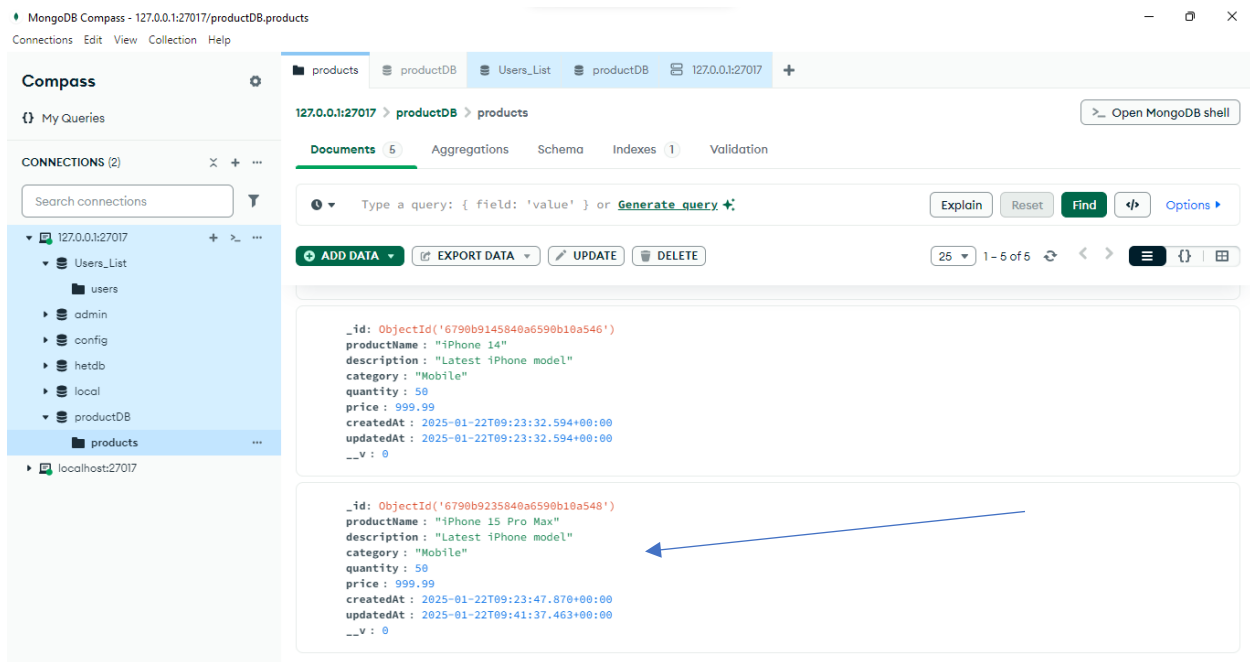
The screenshot shows a REST client interface with a DELETE request to `http://localhost:3000/api/products/6790b9235840a6590b10a548`. The response is a 200 OK status with a size of 263 Bytes and a time of 18 ms. The response body is a JSON object:

```
{
  "success": true,
  "data": {
    "_id": "6790b9235840a6590b10a548",
    "productName": "iPhone 15 Pro Max",
    "description": "Latest iPhone model",
    "category": "Mobile",
    "quantity": 50,
    "price": 999.99,
    "createdAt": "2025-01-22T09:23:47.870Z",
    "updatedAt": "2025-01-22T09:41:37.463Z",
    "__v": 0
  }
}
```

**After Delete:**

The screenshot shows the same DELETE request, but the response is now a 200 OK status with a size of 26 Bytes and a time of 35 ms. The response body is a simplified JSON object:

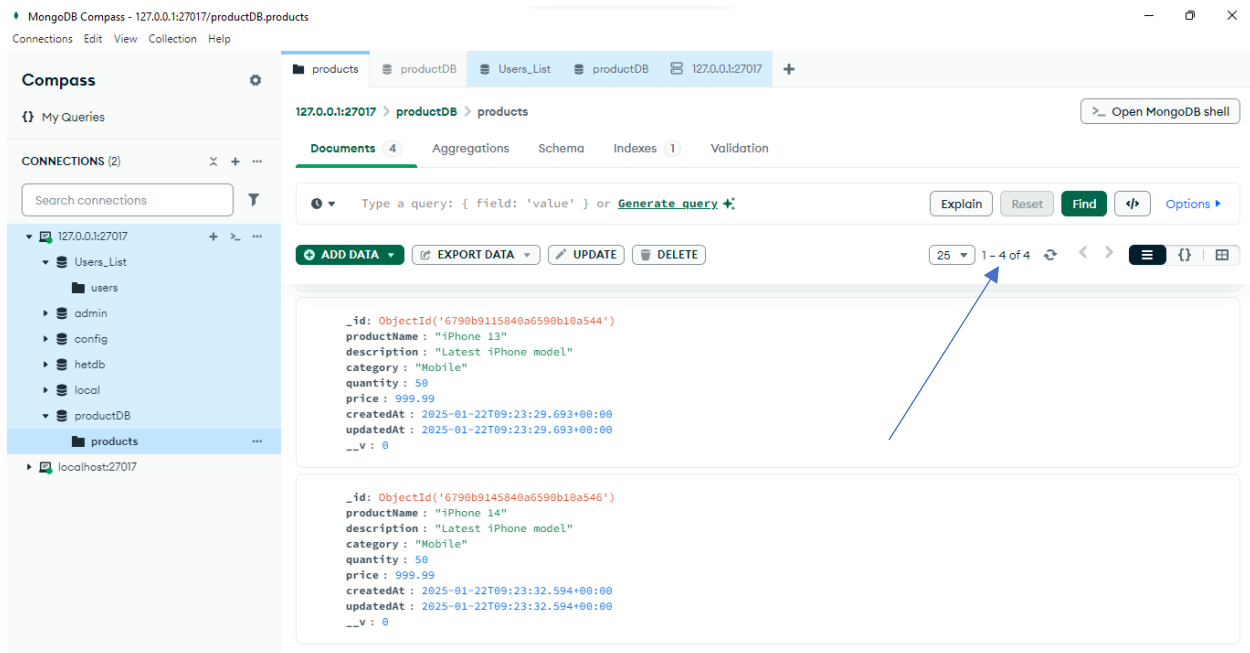
```
{
  "success": true,
  "data": {}
}
```

**Before:**

MongoDB Compass interface showing the **products** collection in the **productDB** database. The **Documents** tab is selected, displaying 5 documents. The query bar shows a filter: `{ field: 'value' }` or [Generate query](#). The document list shows:

- Document 1: `{ "_id": ObjectId('6790b9145840a6590b10a546'), "productName": "iPhone 14", "description": "Latest iPhone model", "category": "Mobile", "quantity": 50, "price": 999.99, "createdAt": "2025-01-22T09:23:32.594+00:00", "updatedAt": "2025-01-22T09:23:32.594+00:00", "__v": 0 }`
- Document 2: `{ "_id": ObjectId('6790b9235840a6590b10a548'), "productName": "iPhone 15 Pro Max", "description": "Latest iPhone model", "category": "Mobile", "quantity": 50, "price": 999.99, "createdAt": "2025-01-22T09:23:47.870+00:00", "updatedAt": "2025-01-22T09:41:37.463+00:00", "__v": 0 }`

A blue arrow points to the second document.

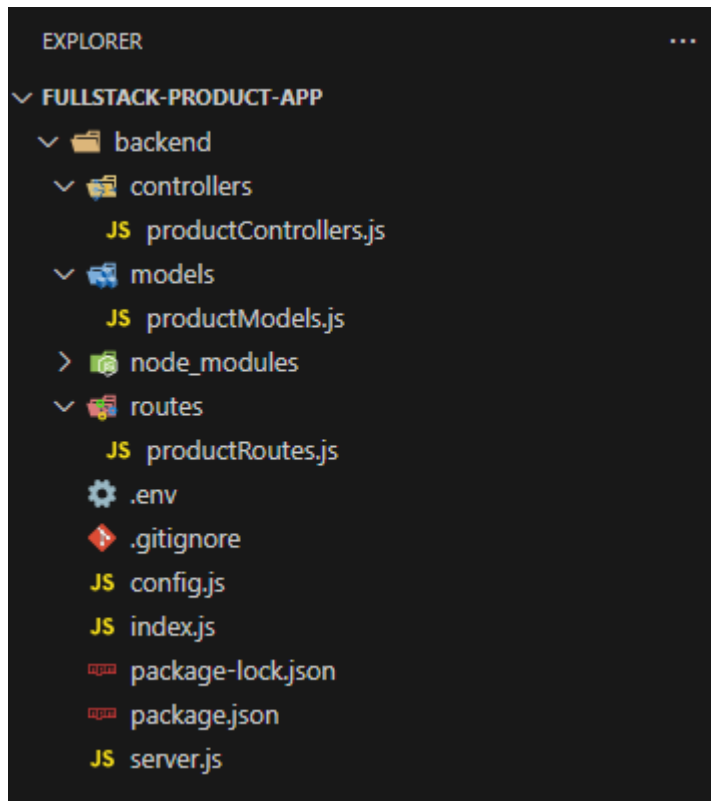
**After:**

MongoDB Compass interface showing the **products** collection in the **productDB** database. The **Documents** tab is selected, displaying 4 documents. The query bar shows a filter: `{ field: 'value' }` or [Generate query](#). The document list shows:

- Document 1: `{ "_id": ObjectId('6790b9115840a6590b10a544'), "productName": "iPhone 13", "description": "Latest iPhone model", "category": "Mobile", "quantity": 50, "price": 999.99, "createdAt": "2025-01-22T09:23:29.693+00:00", "updatedAt": "2025-01-22T09:23:29.693+00:00", "__v": 0 }`
- Document 2: `{ "_id": ObjectId('6790b9145840a6590b10a546'), "productName": "iPhone 14", "description": "Latest iPhone model", "category": "Mobile", "quantity": 50, "price": 999.99, "createdAt": "2025-01-22T09:23:32.594+00:00", "updatedAt": "2025-01-22T09:23:32.594+00:00", "__v": 0 }`

A blue arrow points to the pagination control showing **1 - 4 of 4**.

### ***Project Structure:***



**Author:**

**- HET DABHI**

***"Stay connected with me for more tech updates and projects!"***

 **GitHub: [hetdabhi](#)**

 **LinkedIn: [hetdabhi](#)**