## Page:1

# Express CRUD Operation

## Importing Required Modules

const express = require("express");

const fs = require("fs");

const app = express();

const cors = require("cors");

const express = require("express");: This line imports the Express module. Express is a web framework for Node.js that helps you build web applications and APIs.

const fs = require("fs");: This imports the fs (file system) module, which allows you to interact with the file system, including reading from and writing to files.

const app = express();: Here, you create an instance of the Express application by calling express(). This app object will be used to set up routes, middleware, and more.

const cors = require("cors");: This imports the cors module, which is a middleware that allows your application to handle cross-origin requests, making it accessible from different domains.

## Middleware Setup

app.use(express.json());

app.use(cors());

* **app.use(express.json());**: This middleware parses incoming requests with JSON payloads and makes the data available in req.body. This is useful for handling POST, PUT, and PATCH requests where data is sent in JSON format.
* **app.use(cors());**: This middleware enables CORS (Cross-Origin Resource Sharing). It allows your server to accept requests from other domains, which is often necessary when developing frontend applications that interact with your backend API.

## GET Route: Fetch All Products

app.get("/product", (req, res) => {

fs.readFile("./db.json", "utf-8", (err, data) => {

if (err) {

res.send(err);

} else {

res.send(data);

}

});

});

* **app.get("/product", (req, res) => { ... });**: This sets up a GET route at /product. When a GET request is made to this route, the provided callback function is executed.
* **fs.readFile("./db.json", "utf-8", (err, data) => { ... });**: The fs.readFile function reads the contents of the db.json file. The second argument, "utf-8", specifies the encoding format. The callback function is executed once the file reading is complete.
  + **if (err) { res.send(err); }**: If an error occurs while reading the file, the error is sent back in the response.
  + **else { res.send(data); }**: If no error occurs, the data from the file is sent as a response.

## POST Route: Add a New Product

app.post("/addproduct", (req, res) => {

fs.readFile("./db.json", "utf-8", (err, data) => {

if (err) {

res.send(err);

} else {

const newdata = JSON.parse(data);

newdata.push(req.body);

fs.writeFile("./db.json", JSON.stringify(newdata), (err) => {

if (err) {

res.send(err);

} else {

res.send("product added successfully");

}

});

}

});

});

* **app.post("/addproduct", (req, res) => { ... });**: This sets up a POST route at /addproduct. When a POST request is made to this route, the callback function is executed.
* **fs.readFile("./db.json", "utf-8", (err, data) => { ... });**: The server reads the current data from db.json.
  + **const newdata = JSON.parse(data);**: The JSON data read from the file is parsed into a JavaScript object.
  + **newdata.push(req.body);**: The new product data, which is sent in the request body (req.body), is added to the list of products.
  + **fs.writeFile("./db.json", JSON.stringify(newdata), (err) => { ... });**: The updated list of products is written back to db.json. If an error occurs during writing, it is sent in the response. If successful, a success message is sent.

## PATCH Route: Update a Product Partially

app.patch("/editproduct/:id", (req, res) => {

const { id } = req.params;

fs.readFile("./db.json", "utf-8", (err, data) => {

if (err) {

res.send(err);

} else {

const productdata = JSON.parse(data);

const index = productdata.findIndex((el) => el.id == id);

if (index != -1) {

productdata[index] = { ...productdata[index], ...req.body };

fs.writeFile("./db.json", JSON.stringify(productdata), (err) => {

if (err) {

res.send(err);

} else {

res.send("data edit successfully");

}

});

} else {

res.send("data is not found");

}

}

});

});

* **app.patch("/editproduct/:id", (req, res) => { ... });**: This sets up a PATCH route at /editproduct/:id. The :id is a route parameter that will be replaced with the actual product ID when the request is made.
* **const { id } = req.params;**: This line extracts the id from the request parameters (req.params).
* **fs.readFile("./db.json", "utf-8", (err, data) => { ... });**: The server reads the current data from db.json.
  + **const productdata = JSON.parse(data);**: The JSON data is parsed into a JavaScript object.
  + **const index = productdata.findIndex((el) => el.id == id);**: This line finds the index of the product with the matching id.
  + **if (index != -1) { ... } else { ... }**: If a matching product is found, its data is updated with the new data from req.body. If no matching product is found, an error message is sent.
  + **productdata[index] = { ...productdata[index], ...req.body };**: The existing product data is merged with the new data. The spread operator (...) is used to combine the existing data with the new data.
  + **fs.writeFile("./db.json", JSON.stringify(productdata), (err) => { ... });**: The updated product data is written back to db.json.

# DELETE Route: Delete a Product

app.delete("/deleteproduct/:id", (req, res) => {

const { id } = req.params;

fs.readFile("./db.json", "utf8", (err, data) => {

if (err) {

res.send(err);

} else {

const productdata = JSON.parse(data);

const filterdata = productdata.filter((el) => el.id != id);

fs.writeFile("./db.json", JSON.stringify(filterdata), (err) => {

if (err) {

res.send(err);

} else {

res.send("I am From Delete Route");

}

});

}

});

});

* **app.delete("/deleteproduct/:id", (req, res) => { ... });**: This sets up a DELETE route at /deleteproduct/:id.
* **const { id } = req.params;**: The id parameter is extracted from the route.
* **fs.readFile("./db.json", "utf8", (err, data) => { ... });**: The server reads the current data from db.json.
  + **const productdata = JSON.parse(data);**: The JSON data is parsed into a JavaScript object.
  + **const filterdata = productdata.filter((el) => el.id != id);**: This line filters out the product with the specified id from the list of products.
  + **fs.writeFile("./db.json", JSON.stringify(filterdata), (err) => { ... });**: The updated product list (without the deleted product) is written back to db.json.
  + **res.send("I am From Delete Route");**: A success message is sent.

# PUT Route: Replace a Product Entirely

app.put("/updateproduct/:id", (req, res) => {

const { id } = req.params;

fs.readFile("./db.json", "utf-8", (err, data) => {

if (err) {

res.send(err);

} else {

const productdata = JSON.parse(data);

const index = productdata.findIndex((el) => el.id == id);

if (index !== -1) {

productdata[index] = { ...req.body, id: productdata[index].id };

fs.writeFile("./db.json", JSON.stringify(productdata), (err) => {

if (err) {

res.send(err);

} else {

res.send("Product updated successfully");

}

});

} else {

res.send("Product not found");

}

}

});

});

* **app.put("/updateproduct/:id", (req, res) => { ... });**: This sets up a PUT route at /updateproduct/:id.
* **const { id } = req.params;**: The id parameter is extracted from the route.
* **fs.readFile("./db.json", "utf-8", (err, data) => { ... });**: The server reads the current data from db.json.
  + **const productdata = JSON.parse(data);**: The JSON data is parsed into a JavaScript object.
  + **const index = productdata.findIndex((el) => el.id == id);**: This line finds the index of the product with the specified id.
  + **if (index !== -1) { ... } else { ... }**: If a matching product is found, it is replaced entirely with the new data from req.body, except for the id, which remains unchanged. If no matching product is found, an error message is sent.
  + **productdata[index] = { ...req.body, id: productdata[index].id };**: The existing product is replaced with the new data, but the original id is retained.
  + **fs.writeFile("./db.json", JSON.stringify(productdata), (err) => { ... });**: The updated product data is written back to db.json.

# Starting the Server

app.listen(8080, () => {

console.log("server is running on port 8080");

});

* **app.listen(8080, () => { ... });**: This line starts the server and listens for incoming requests on port 8080. The callback function is executed once the server is successfully running.
* **console.log("server is running on port 8080");**: This logs a message to the console indicating that the server is up and running.

# Summary of Basic Concepts:

* **Express**: A web framework for Node.js that simplifies routing and handling HTTP requests.
* **Middleware**: Functions that process incoming requests before they reach your routes. Examples include express.json() for parsing JSON and cors() for handling cross-origin requests.
* **Routing**: Defining endpoints in your application that respond to specific HTTP methods (GET, POST, PATCH, DELETE, PUT).
* **File Handling**: Reading from and writing to files using Node.js's fs module.
* **CRUD Operations**: Create (POST), Read (GET), Update (PATCH/PUT), Delete (DELETE) operations performed on the products stored in db.json.

# Program

const express = require("express");

const fs = require("fs");

const app = express();

const cors = require("cors");

app.use(express.json());

app.use(cors()); //middle-ware

// #### Get Routes #####

app.get("/product", (req, res) => {

  fs.readFile("./db.json", "utf-8", (err, data) => {

    if (err) {

      res.send(err);

    } else {

      res.send(data);

    }

  });

});

//#### Post Routes #####

app.post("/addproduct", (req, res) => {

  fs.readFile("./db.json", "utf-8", (err, data) => {

    if (err) {

      res.send(err);

    } else {

      const newdata = JSON.parse(data);

      newdata.push(req.body);

      fs.writeFile("./db.json", JSON.stringify(newdata), (err) => {

        if (err) {

          res.send(err);

        } else {

          res.send("product added successfully");

        }

      });

    }

  });

});

//#####PATCH ROUTES

app.patch("/editproduct/:id", (req, res) => {

  //   console.log(req.body, req.params, req.params.id);

  //read the data

  //destructure of id

  const { id } = req.params;

  fs.readFile("./db.json", "utf-8", (err, data) => {

    if (err) {

      res.send(err);

    } else {

      // console.log(data);// get json data

      const productdata = JSON.parse(data);

      //   console.log(productdata); //get normal data

      //   productdata.findIndex((el) => console.log(el)); //all data

      const index = productdata.findIndex((el) => el.id == id); //now store in filterdata

      //   console.log(index);

      //   console.log(productdata[index]);

      if (index != -1) {

        productdata[index] = { ...productdata[index], ...req.body };

        // console.log(productdata);

        fs.writeFile("./db.json", JSON.stringify(productdata), (err) => {

          if (err) {

            res.send(err);

          } else {

            res.send("data edit successfully");

          }

        });

        // res.send("data edit successfully");

      } else {

        res.send("data is not found");

      }

      //   res.send("");

    }

  });

  //   res.send("I am from Edit Route");

});

// DELETE ###

app.delete("/deleteproduct/:id", (req, res) => {

  console.log(req.params);

  const { id } = req.params;

  fs.readFile("./db.json", "utf8", (err, data) => {

    if (err) {

      res.send(err);

    } else {

      const productdata = JSON.parse(data);

      // console.log(productdata);all data

      const filterdata = productdata.filter((el, index) => el.id != id);

      // console.log(filterdata);

      fs.writeFile("./db.json", JSON.stringify(filterdata), (err) => {

        if (err) {

          res.send(err);

        } else {

          res.send("I am From Delete Route");

        }

      });

    }

  });

  // res.send("I am From Delete Route");

});

// ##### PUT ROUTE #####

app.put("/updateproduct/:id", (req, res) => {

  console.log(req.params);

  const { id } = req.params;

  // Read the data from the file

  fs.readFile("./db.json", "utf-8", (err, data) => {

    if (err) {

      res.send(err);

    } else {

      const productdata = JSON.parse(data);

      const index = productdata.findIndex((el) => el.id == id);

      if (index !== -1) {

        // Replace the existing product with the new data

        productdata[index] = { ...req.body, id: productdata[index].id }; // Retain the original ID

        // Write the updated data back to the file

        fs.writeFile("./db.json", JSON.stringify(productdata), (err) => {

          if (err) {

            res.send(err);

          } else {

            res.send("Product updated successfully");

          }

        });

      } else {

        res.send("Product not found");

      }

    }

  });

});

app.listen(8080, () => {

  console.log("server is running on port 8080");

});

## db.json

[

  {

    "id": 1,

    "title": "Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops",

    "price": 900,

    "description": "Your perfect pack for everyday use and walks in the forest. Stash your laptop (up to 15 inches) in the padded sleeve, your everyday",

    "category": "men's clothing",

    "image": "https://fakestoreapi.com/img/81fPKd-2AYL.\_AC\_SL1500\_.jpg",

    "rating": { "rate": 3.9, "count": 120 }

  }]

For example only

# Package.json

{

"name": "express",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"server": "nodemon index.js",

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [],

"author": "",

"license": "ISC",

"dependencies": {

"cors": "^2.8.5",

"cross-env": "^7.0.3",

"express": "^4.19.2",

"nodemon": "^3.1.4"

}

}