Node.js

TABLE OF CONTENTS

01

Route params

Express

Route params

- Route parameters are variable parts of the URL segment.
- That can be used to pass extra information to a given route.
  - /greet/john
  - /greet/jane
  - /greet/world
  - and so on ..
- In these examples, john, jane, and world strings are route parameters.
- That the greet route can use to respond to requests.

Express

Route params – Example 1

```
const express = require("express");
const app = express();

const users = [
  {
    id: 1,
    name: "John",
  },
  {
    id: 2,
    name: "Paul",
  },
];

app.get("/users/:userId", (req, res) => {
  const { userId } = req.params; // userId: '42'
  res.status(200).send(users.find((user) => user.id == userId));
});
```

Route param

Express

Route params – Example 2

```
const products = [
  {
    id: 1,
    name: "Iphone 14",
    price: 200000
  },
  {
    id: 2,
    name: "HP Laptop",
    price: 150000
  },
];

app.get("/products", (req, res) => {
    res.status(200).send(products);
  });

app.get("/users/:productId", (req, res) => {
  const { productId } = req.params; // productId: '42'
  res.status(200).send(products.find((product) => product.id == productId));
});
```

Route param

Express

Route params – Example 2

Route param

```
app.get("/users/:productId/reviews/:reviewId", (req, res) => {
  const { productId, reviewId } = req.params; // productId: '1' | reviewId = 12
  console.log(productId, reviewId);
});
```

02

Query params

Express

Query params

- Parameters attached to the end of a URL.
- Separated from the URL by a question mark (?).
- The section before the question mark is the path parameter
- The section after the question mark is the query parameter.
- The path parameter defines the resource location.
- While the query parameter defines sort, pagination, or filter operations.
- The user's input (the query) is passed as a variable in the query parameter.

Express

Query params – Example 1

localhost:5000/clothing/shirts?color=red

```
app.get("/clothing/shirts", (req, res) => {
  const params = req.query; // { color: 'red' }
});
```

Express

Query params – Example 2

localhost:5000/clothing/shirts?color=red&category=polo

```
app.get("/clothing/shirts", (req, res) => {
  const params = req.query; // { color: 'red', category: 'polo' }
});
```

03

Add middleware layer

Express

Add middleware layer

- Suppose we want same code to run in some different middlewares.

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

  const method = req.method;
  const url = req.url;
  const year = new Date().getFullYear();
  console.log(method, url, year);

  res.status(200).send("Home page");
});

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

  const method = req.method;
  const url = req.url;
  const year = new Date().getFullYear();
  console.log(method, url, year);

  res.status(200).send(products);
});
```

Same code

Express

Add middleware layer

- Create a new middleware function.

```
const logger = (req, res, next) => {
    const method = req.method
    const url = req.url
    const year = new Date().getFullYear()
    console.log(method, url, year)
    next() // execute next middleware
}
```

Express

Add middleware layer

- Pass the custom created middleware as a layer between middleware.

```
app.get("/", logger , (req, res) => {
  res.status(200).send("Home page");
});

app.get("/products", logger , (req, res) => {
  res.status(200).send(products);
});
```

Express

Add middleware layer

- But now what if we have to apply this middleware to all the other middlewares?
- Should we add this middleware to all of other middlewares just like we did here?
- This will not be efficient also violent the rule of DRY code.

```
app.get("/", logger , (req, res) => {
  res.status(200).send("Home page");
});

app.get("/products", logger , (req, res) => {
  res.status(200).send(products);
});
```

Express

Add middleware layer – app.use( )

- But now what if we have to apply this middleware to all the other middlewares?

```
app.use(logger)

app.get("/" , (req, res) => {
  res.status(200).send("Home page");
});

app.get("/products" , (req, res) => {
  res.status(200).send(products);
});
```

Express

app.use( )

- Adds a middleware function to the processing chain.

```
app.use(logger)

app.get("/" , (req, res) => {
  res.status(200).send("Home page");
});

app.get("/products" , (req, res) => {
  res.status(200).send(products);
});
```

- Handles the incoming request that has method as “GET” and endpoint as “/clothing”

04

Multiple middleware functions

Express

Multiple middleware functions

- Now suppose we have to add more middleware functions…
- We can pass an array of middleware functions to the app.use( ) method.
- NOTE: order of the middleware functions does matter!

```
app.use([ authorize, logger ]);

app.get("/", (req, res) => {
  res.status(200).send("Home page");
});

app.get("/products", (req, res) => {
  res.status(200).send(products);
});
```

05

Express PostgreSQL Integration

Express PostgreSQL integration - node-postgres

- We'll use the node-postgres module to create a pool of connections.
- node-postgres is a collection of node.js modules for interfacing with your PostgreSQL database.
- To install enter the following command in terminal:

```
$ npm install pg
```

Express PostgreSQL integration

- Create a file called db.config.js and set up the configuration of your PostgreSQL connection:

```
const Pool = require('pg').Pool
const pool = new Pool({
  user: 'me',
  host: 'localhost',
  database: 'api',
  password: 'password',
  port: 5432,
})
```

Express PostgreSQL integration

- Now try accessing tables from your DB by creating routes and controllers:

```
app.get("/users", (req, res) => {
  pool.query("SELECT * FROM users", (error, results) => {
    if (error) {
      throw error;
    }
    res.status(200).json(results.rows);
  });
});
```

- If you don’t have any table to access, you may want to add a table just of demo…

<QnA>

Thanks!

Regards: Hassan Bilal