Taller NodeJS – Express: Rutas y controladores

Instructor: Abner Saavedra
Fecha: sept. de 2024
Email: ingenieroabnersaavedra@gmail.com
GracoSoft Centro Empresarial Plaza Madrid, piso 9, oficinas 9-7 a la 9-10

Agenda

- Rutas y controladores
- Definición de Rutas
- Parámetros de Ruta
- Controladores en Express.js
- Middleware de Rutas
- Rutas Anidadas y Modularización
- Ejemplo Completo

Rutas y controladores

En esta lección, aprenderemos cómo definir y manejar rutas en Express.js, y cómo utilizar controladores para organizar la lógica de nuestras rutas de manera modular y eficiente.

Definición de Rutas

Las rutas en Express.js son puntos de entrada a nuestra aplicación que responden a solicitudes HTTP. Puedes definir rutas para los métodos HTTP más comunes como GET, POST, PUT, DELETE, entre otros.

Ejemplo básico de rutas:

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

// Ruta GET para la página principal
app.get('/', (req, res) => {
  res.send('Página principal');
});

// Ruta GET para la página de "about"
app.get('/about', (req, res) => {
  res.send('Página About');
});

// Ruta POST para el manejo de formularios
app.post('/submit', (req, res) => {
  res.send('Formulario enviado');
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Servidor ejecutándose en http://localhost:${PORT}`);
});
```

Parámetros de Ruta

Los parámetros de ruta permiten a tus rutas capturar valores específicos desde la URL.

Ejemplo de uso de parámetros de ruta:

```
// Definir una ruta con un parámetro de ruta
app.get('/user/:id', (req, res) => {
  res.send(`Usuario ID: ${req.params.id}`);
});
```

Cuando navegas a `/user/123`, la respuesta será `"Usuario ID: 123"`.

Controladores en Express.js

Para mantener nuestro código limpio y organizado, es una buena práctica separar la lógica de nuestras rutas en controladores. Los controladores son simplemente módulos que exportan funciones manejar la lógica de las rutas.

Estructura del proyecto:

```
mi-proyecto-express
|-- app.js
|-- routes
    |-- index.js
|-- controllers
    |-- userController.js
```

1. **Crear el controlador:**

```
// controllers/userController.js
exports.getUser = (req, res) => {
  res.send(`Usuario ID: ${req.params.id}`);
};
```

2. **Definir la ruta y utilizar el controlador:**

```
// routes/index.js
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');

router.get('/user/:id', userController.getUser);

module.exports = router;
```

3. **Usar las rutas en la aplicación principal:**

```
// app.js
const express = require('express');
const app = express();
const indexRouter = require('./routes/index');

app.use('/', indexRouter);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Servidor ejecutándose en
http://localhost:${PORT}`);
});
```

Middleware de Rutas

Puedes usar middlewares específicos para determinadas rutas para realizar tareas previas a la ejecución del controlador.
Ejemplo de middleware de ruta:

```
// Middleware específico para la ruta /admin
const adminMiddleware = (req, res, next) => {
  console.log('Se ha accedido a la ruta /admin');
  next();
};

// Usar middleware en la ruta
app.get('/admin', adminMiddleware, (req, res) => {
  res.send('
```

Rutas Anidadas y Modularización

Si tienes una aplicación grande, es una buena práctica modularizar tus rutas en diferentes archivos y carpetas.

Estructura del proyecto modularizado:

```
mi-proyecto-express
|-- app.js
|-- routes
|   |-- index.js
|   |-- userRoutes.js
|-- controllers
|   |-- userController.js
```

1. **Definir rutas anidadas:**

```
// routes/userRoutes.js
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');

router.get('/:id', userController.getUser);

module.exports = router;
```

2. **Añadir rutas anidadas a la aplicación principal:**

```
// routes/index.js
const express = require('express');
const router = express.Router();
const userRoutes = require('./userRoutes');

router.use('/user', userRoutes);

module.exports = router;
```

3. **Configurar la aplicación para usar el conjunto de rutas modularizadas:**

```
// app.js
const express = require('express');
const app = express();
const indexRouter = require('./routes/index');

app.use('/', indexRouter);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Servidor ejecutándose en http://localhost:${PORT}`);
});
```

Ejemplo Completo

A continuación, un ejemplo completo que combina varios de estos conceptos:

Estructura del proyecto:

```
mi-proyecto-express
|-- app.js
|-- routes
|   |-- index.js
|   |-- userRoutes.js
|-- controllers
|   |-- userController.js
```

1. **app.js:**

```
const express = require('express');
const app = express();
const indexRouter = require('./routes/index');

app.use(express.json());
app.use('/', indexRouter);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Servidor ejecutándose en http://localhost:${PORT}`);
});
```

2. **controllers/userController.js:**

```
exports.getUser = (req, res) => {
  res.send(`Usuario ID: ${req.params.id}`);
};

exports.createUser = (req, res) => {
  const { username, email } = req.body;
  res.send(`Usuario creado: ${username}, Email: ${email}`);
};
```

3. **routes/userRoutes.js:**

```
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');

router.get('/:id', userController.getUser);
router.post('/', userController.createUser);

module.exports = router;
```

4. **routes/index.js:**

```
const express = require('express');
const router = express.Router();
const userRoutes = require('./userRoutes');

router.use('/user', userRoutes);

router.get('/', (req, res) => {
  res.send('Página principal');
});

module.exports = router;
```

Con este conocimiento sobre rutas y controladores en Express.js, puedes organizar y gestionar la lógica de tu aplicación de manera eficiente, modular y escalable.

Ejercicio

Construir una API utilizando Node.js y Express para gestionar productos. La API debe permitir realizar las operaciones básicas de creación, lectura, actualización y eliminación (CRUD) de productos

Requisitos:

1. **Creación de Producto:**
   - Implementa una ruta para agregar un nuevo producto a la base de datos.
   - El cuerpo de la solicitud (`body`) debe contener la información necesaria del producto, como nombre, precio y cantidad.
2. **Consulta de Producto:**
   - Implementa una ruta para consultar la información de un producto específico por su identificador único.
   - La información del producto debe incluir al menos el nombre, precio y cantidad.
3. **Modificación de Producto:**
   - Crea una ruta para actualizar la información de un producto existente.
   - El cuerpo de la solicitud debe contener los campos que se desean actualizar.
4. **Eliminación de Producto:**
   - Implementa una ruta para eliminar un producto según su identificador único.
5. **Filtrado de Información:**
   - Proporciona la capacidad de filtrar la información de los productos según ciertos criterios (por ejemplo, por precio o cantidad).
   - Los criterios de filtrado deben ser parámetros de la URL.

Notas Adicionales:

- Utiliza Express para crear el servidor web.
- Utiliza el módulo Express Router para crear las rutas de acceso e implementa controladores para modularizar el código.
- Puedes almacenar la información de los productos en una base de datos simple, como un archivo JSON o en memoria.

GRACOSOFT ES EXCELENCIA EDUCATIVA