Instructivo: Construcción de API con Express.js

Julian F. Latorre

1 Introducción

Este instructivo guiará a través del proceso de diseño e implementación de una API RESTful utilizando Express.js. Crearemos una API simple para gestionar una lista de tareas (todo list).

2 Diseño de la API

El diseño de una API establece las bases para el desarrollo y el éxito a largo plazo de una aplicación. A continuación se propone un proceso detallado para diseñar nuestra API de lista de tareas.

2.1 Identificación de Recursos

El primer paso es identificar los recursos principales que nuestra API va a manejar. En este caso, nuestro recurso principal es una "tarea" (task).

2.2 Definición de Operaciones CRUD

Para cada recurso, definimos las operaciones CRUD (Create, Read, Update, Delete) básicas:

- Crear una nueva tarea
- Leer una tarea existente
- Actualizar una tarea existente
- Eliminar una tarea existente

Además, consideramos una operación adicional para leer todas las tareas.

2.3 Mapeo de Operaciones a Métodos HTTP

Ahora, mapeamos estas operaciones a métodos HTTP específicos:

- POST /tasks - Crear una nueva tarea

- GET /tasks/:id - Obtener una tarea específica
- GET /tasks - Obtener todas las tareas
- PUT /tasks/:id - Actualizar una tarea existente
- DELETE /tasks/:id - Eliminar una tarea

2.4 Diseño de la Estructura de Datos

Definimos la estructura de datos para una tarea:

```
{
  "id": "string (UUID)",
  "title": "string",
  "description": "string",
  "completed": "boolean",
  "createdAt": "date"
}
```

2.5 Definición de Parámetros de Solicitud

Para cada endpoint, especificamos los parámetros de solicitud necesarios:

- POST /tasks
  - Cuerpo: { title: string, description: string (opcional) }
- GET /tasks/:id
  - Parámetro de ruta: id (UUID)
- GET /tasks
  - Parámetros de consulta opcionales: completed (boolean)
- PUT /tasks/:id
  - Parámetro de ruta: id (UUID)
  - Cuerpo: { title: string (opcional), description: string (opcional), completed: boolean (opcional) }
- DELETE /tasks/:id
  - Parámetro de ruta: id (UUID)

2.6 Especificación de Respuestas

Definimos las respuestas esperadas para cada endpoint, incluyendo los códigos de estado HTTP:

- POST /tasks
  - Éxito: 201 Created, cuerpo: objeto tarea creado
  - Error: 400 Bad Request si los datos son inválidos
- GET /tasks/:id
  - Éxito: 200 OK, cuerpo: objeto tarea
  - Error: 404 Not Found si la tarea no existe
- GET /tasks
  - Éxito: 200 OK, cuerpo: array de objetos tarea
- PUT /tasks/:id
  - Éxito: 200 OK, cuerpo: objeto tarea actualizado
  - Error: 404 Not Found si la tarea no existe
  - Error: 400 Bad Request si los datos son inválidos
- DELETE /tasks/:id
  - Éxito: 204 No Content
  - Error: 404 Not Found si la tarea no existe

2.7 Consideraciones de Seguridad

Aunque no implementaremos autenticación en este ejemplo básico, en una API de producción consideraríamos:

- Autenticación: Usar JWT o OAuth para autenticar solicitudes
- Autorización: Asegurar que los usuarios solo puedan acceder a sus propias tareas
- Rate Limiting: Limitar el número de solicitudes por usuario para prevenir abusos
- HTTPS: Asegurar todas las comunicaciones mediante HTTPS

2.8 Versionado de la API

Para futuras actualizaciones, consideramos el versionado de la API. Podríamos prefijar todas las rutas con /v1, por ejemplo:

- POST /v1/tasks
- GET /v1/tasks/:id
- etc.

2.9 Documentación

Planificamos documentar nuestra API utilizando herramientas como Swagger/OpenAPI. Esto proporcionará una interfaz interactiva para que los desarrolladores prueben y entiendan nuestra API.

2.10 Pruebas

Diseñamos un plan de pruebas que incluye:

- Pruebas unitarias para cada función del controlador
- Pruebas de integración para cada endpoint
- Pruebas de carga para asegurar que la API puede manejar múltiples solicitudes simultáneas

Este proceso de diseño establece una base sólida para nuestra API, considerando no solo la funcionalidad básica, sino también aspectos importantes como la seguridad, la escalabilidad y la facilidad de uso para los desarrolladores que consumirán la API.

3 Configuración del Proyecto

3.1 Crear el directorio del proyecto

Abra una terminal y ejecute los siguientes comandos:

```
mkdir express-todo-api
cd express-todo-api
```

3.2 Inicializar el proyecto Node.js

Ejecute el siguiente comando:

```
npm init -y
```

3.3 Instalar dependencias

Instale las dependencias necesarias con el siguiente comando:

```
npm install express body-parser uuid
```

4 Estructura del Proyecto

La estructura del proyecto será la siguiente:

```
express-todo-api/
├── src/
│   ├── routes/
│   │   └── tasks.js
│   ├── controllers/
│   │   └── taskController.js
│   ├── models/
│   │   └── task.js
│   └── app.js
└── package.json
```

Esta estructura organiza nuestro código en módulos lógicos:

- `routes/`: Contiene las definiciones de rutas de la API.
- `controllers/`: Contiene la lógica de manejo de las solicitudes.
- `models/`: Contiene las definiciones de los modelos de datos.
- `app.js`: El punto de entrada principal de nuestra aplicación.

5 Implementación

5.1 Modelo de Tarea

Cree el archivo `src/models/task.js`:

```
const { v4: uuidv4 } = require('uuid');

class Task {
  constructor(title, description = '') {
    this.id = uuidv4();
    this.title = title;
    this.description = description;
    this.completed = false;
    this.createdAt = new Date();
  }
}

module.exports = Task;
```

5.2 Controlador de Tareas

Cree el archivo `src/controllers/taskController.js`:

```
const Task = require('../models/task');

let tasks = [];

exports.getAllTasks = (req, res) => {
  res.json(tasks);
};

exports.createTask = (req, res) => {
  const { title, description } = req.body;
  const newTask = new Task(title, description);
  tasks.push(newTask);
  res.status(201).json(newTask);
};

exports.getTask = (req, res) => {
  const task = tasks.find(t => t.id === req.params.id);
  if (task) {
    res.json(task);
  } else {
    res.status(404).json({ message: 'Task not found' });
  }
};

exports.updateTask = (req, res) => {
  const index = tasks.findIndex(t => t.id === req.params.id);
  if (index !== -1) {
    tasks[index] = { ...tasks[index], ...req.body };
    res.json(tasks[index]);
  } else {
    res.status(404).json({ message: 'Task not found' });
  }
};

exports.deleteTask = (req, res) => {
  const index = tasks.findIndex(t => t.id === req.params.id);
  if (index !== -1) {
    tasks.splice(index, 1);
    res.status(204).send();
  } else {
    res.status(404).json({ message: 'Task not found' });
  }
};
```

5.3 Rutas de Tareas

Cree el archivo `src/routes/tasks.js`:

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

router.get('/', taskController.getAllTasks);
```

```
router.post('/', taskController.createTask);
router.get('/:id', taskController.getTask);
router.put('/:id', taskController.updateTask);
router.delete('/:id', taskController.deleteTask);

module.exports = router;
```

5.4 Aplicación Principal

Cree el archivo `src/app.js`:

```
const express = require('express');
const bodyParser = require('body-parser');
const taskRoutes = require('./routes/tasks');

const app = express();
const PORT = process.env.PORT || 3000;

app.use(bodyParser.json());

app.use('/tasks', taskRoutes);

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

6 Ejecución de la API

Para ejecutar la API, use el siguiente comando en la terminal:

```
node src/app.js
```

La API ahora estará disponible en `http://localhost:3000`.

7 Prueba de la API

Pruebe la API utilizando herramientas como Postman o curl. Aquí hay algunos ejemplos de comandos curl:

```
# Crear una tarea
curl -X POST -H "Content-Type: application/json" -d '{"title":"
    Comprar leche","description":"2 litros de leche"}' http://
    localhost:3000/tasks

# Obtener todas las tareas
curl http://localhost:3000/tasks

# Obtener una tarea específica (reemplace <id> con un ID real)
curl http://localhost:3000/tasks/<id>

# Actualizar una tarea
curl -X PUT -H "Content-Type: application/json" -d '{"completed":
    true}' http://localhost:3000/tasks/<id>
```

```
# Eliminar una tarea
curl -X DELETE http://localhost:3000/tasks/<id>
```