express.js

Índice

- Índice
- Instalación
- Creación de un servidor
- Rutas
- Requests
  - Métodos HTTP
    - GET
      - Query parameters
      - Path parameters
    - POST
      - Body parameters
    - PUT
    - DELETE
- Responses
  - Métodos de respuesta

Instalación

Para instalar Express deberemos ejecutar el siguiente comando (dentro de un proyecto de Node ya inicializado):

```
npm install express
```

Creación de un servidor

Para crear un servidor con Express deberemos crear un archivo `index.js` y añadir el siguiente código (recordar que para utilizar `import` deben tener la configuración `"type": "module"` en el archivo `package.json`):

```
// Importamos la librería express
import express from "express";
// Creamos el servidor de Express con la configuración estándar básica
const app = express();

// Asociamos la ruta "/" a la función pasada como segundo parámetro
app.get("/", (req, res) => {
    // Esto envía el texto "Hello World!" como respuesta a la HTTP request
    res.send("Hello World!");
});

// Iniciamos el servidor en el puerto 3000
app.listen(3000, () => {
    console.log("Example app listening on port 3000!");
});
```

Luego se puede ejecutar con el siguiente comando:

```
node index.js
```

Rutas

Las rutas son lo que permiten al cliente utilizar diferentes URLs para hacer uso de diferentes funcionalidades de nuestro programa. Para añadir una ruta a nuestro servidor, simplemente debemos añadir una nueva llamada a `app.get` (o `app.post`, `app.put`, `app.delete`, etc.) con la URL deseada y la función que se ejecutará cuando se realice la HTTP request con el método indicado en esa URL.

```
app.get("/ruta", (req, res) => {
    // Esto envía el texto "Hello World!" como respuesta a la HTTP request
    res.send("Hello World!");
});
```

Con la siguiente HTTP request se ejecutará la función asociada a la ruta `/ruta`:

```
GET http://host:port/ruta HTTP/1.1
```

Requests

Métodos HTTP

Los métodos HTTP más comunes son:

- `GET`: Se utiliza para obtener información del servidor. Toda la información enviada en una petición GET es visible en la URL.
- `POST`: Se utiliza para enviar información al servidor. La información enviada en una petición POST no es visible en la URL, se envía en el cuerpo (*body*) de la petición.
- `PUT`: Se utiliza para actualizar información en el servidor. Se suele enviar información tanto en la URL como en el cuerpo de la petición.
- `DELETE`: Se utiliza para eliminar información del servidor. Al igual que el método `GET`, la información a eliminar se suele enviar en la URL.

GET

La información enviada en una petición GET es visible en la URL.

Query parameters

Una de las formas de envíar datos dentro de la URL, es utilzando los *query parameters*.

Por ejemplo, si queremos enviar un parámetro `nombre` con el valor `Juan`, la URL sería `http://host:port/ruta?nombre=Juan`.

Si quereos enviar más de un parámetro, simplemente debemos separarlos con el símbolo `&`. Por ejemplo, `http://host:port/ruta?nombre=Juan&apellido=Perez`.

```
GET http://host:port/ruta?nombre=Juan&apellido=Perez HTTP/1.1
```

Para obtener los valores de los *query parameters* en el servidor, los encontramos en la propiedad `req.query`.

```
app.get("/ruta", (req, res) => {
    // Obtenemos el valor del parámetro "nombre"
    const nombre = req.query.nombre;
    // Obtenemos el valor del parámetro "apellido"
    const apellido = req.query.apellido;
    // Esto envía el texto "Hello <nombre> <apellido>!" como respuesta
    res.send(`Hello ${nombre} ${apellido}!`);
});
```

Path parameters

Otra forma de enviar datos dentro de la URL, es utilizando *path parameters*.

Por ejemplo, si queremos enviar un parámetro `id` con el valor `123`, la URL sería `http://host:port/ruta/123`.

Si quereos enviar más de un parámetro, simplemente debemos separarlos con el símbolo `/`. Por ejemplo, `http://host:port/ruta/123/Juan`.

```
GET http://host:port/ruta/123/Juan HTTP/1.1
```

Para obtener los valores de los *path parameters* en el servidor. Debemos definir la ruta de una manera especial, indicando el nombre del parámetro precedido de `:`. El nombre del parámetro lo elegimos nosotros, y no afecta a la ruta en la URL.

Cuando queremos recibir más de un parámetro, simplemente debemos añadir más `/:` seguidos del nombre del parámetro. Y los encontramos en la propiedad `req.params`.

Es importante tener en cuenta que irán en el mismo orden en el que los hemos definido en la URL.

```
app.get("/ruta/:id/:nombre", (req, res) => {
    // Obtenemos el valor del parámetro "id"
    const id = req.params.id;
    // Obtenemos el valor del parámetro "nombre"
    const nombre = req.params.nombre;
    // Esto envía el texto "Hello <nombre> with id <id>!" como respuesta
    res.send(`Hello ${nombre} with id ${id}!`);
});
```

POST

La información enviada en una petición POST no es visible en la URL, se envía en el cuerpo (*body*) de la petición.

Cuando se crea un servidor en Express, debemos decidir cómo vamos a recibir la información enviada en una petición POST. En nuestro caso (para la materia Base de Datos), vamos a recibir la información en formato JSON, por lo que debemos añadir el siguiente *middleware* al servidor:

```
app.use(express.json());
```

Esto permite que Express pueda interpretar el cuerpo de la petición como un objeto JSON.

Importante: Si no añadimos este *middleware*, el cuerpo de la petición estaría vacío, y no podremos acceder a los datos enviados en la petición POST.

Body parameters

Para enviar información en una petición POST, debemos añadir un objeto JSON en el cuerpo de la petición.

```
POST http://host:port/ruta HTTP/1.1
Content-Type: application/json

{
    "nombre": "Juan",
    "apellido": "Perez"
}
```

Para obtener los valores de los *body parameters* en el servidor, los encontramos en la propiedad `req.body`.

```
app.post("/ruta", (req, res) => {
    // Obtenemos el valor del parámetro "nombre"
    const nombre = req.body.nombre;
    // Obtenemos el valor del parámetro "apellido"
    const apellido = req.body.apellido;
    // Esto envía el texto "Hello <nombre> <apellido>!" como respuesta
    res.send(`Hello ${nombre} ${apellido}!`);
});
```

PUT

Este método es muy similar al método POST, pero se suele utilizar para actualizar información en el servidor.

```
PUT http://host:port/ruta/:id HTTP/1.1
Content-Type: application/json

{
    "nombre": "Juan"
}
```

Para obtener los valores de los *body parameters* en el servidor, los encontramos en la propiedad `req.body`.

```
app.put("/ruta/:id", (req, res) => {
    // Obtenemos el valor del parámetro "id"
    const id = req.params.id;
    // Obtenemos el valor del parámetro "nombre"
    const nombre = req.body.nombre;

    // Aquí iría la lógica para actualizar al recurso con el id indicado

    // Esto envía el texto "Hello <nombre> with id <id>!" como respuesta
    res.send(`Hello ${nombre} with id ${id}!`);
});
```

DELETE

Este método es muy similar al método GET, pero se suele utilizar para eliminar información en lugar de obtenerla.

```
DELETE http://host:port/ruta/:id HTTP/1.1
```

Para obtener los valores de los *path parameters* en el servidor. Los encontramos en la propiedad `req.params`.

```
app.delete("/ruta/:id", (req, res) => {
    // Obtenemos el valor del parámetro "id"
    const id = req.params.id;

    // Aquí iría la lógica para eliminar al recurso con el id indicado

    // Esto envía el texto "User with id <id> deleted!" como respuesta
    res.send(`User with id ${id} deleted!`);
});
```

Responses

A la hora de manejar las respuestas en nuestro servidor, Express utiliza una lógica que puede resultar confusa al principio, pero que es muy potente y flexible.

La respuesta es primero enviada como parámetro a la función que se ejecuta cuando se realiza la petición. A partir de ahí, podemos utilizar diferentes métodos para modificar la respuesta antes de enviarla al cliente.

Métodos de respuesta

- `res.send(text)`: Envía una respuesta al cliente. El parámetro puede ser un objeto, un array, un string, un número, o `null`.
- `res.json(data)`: Envía una respuesta JSON al cliente. El parámetro puede ser un objeto, un array, un string, un número, o `null`.
- `res.status(status)`: Establece el código de estado de la respuesta..
- `res.sendStatus(status)`: Establece el código de estado de la respuesta y envía el texto asociado a ese código de estado.

Por ejemplo, si queremos enviar un objeto JSON como respuesta a una petición GET, podemos hacerlo de la siguiente manera:

```
app.get("/ruta", (req, res) => {
    // Creamos un objeto JSON
    const data = {
        nombre: "Juan",
        apellido: "Perez"
    };
    // Enviamos el objeto JSON como respuesta
    res.json(data);
});
```