Routing (direccionamiento)

- El direccionamiento hace referencia a cómo debe responder una aplicación a una solicitud de cliente en un determinado endpoint, que es un URI y un método específico (**GET, POST, PUT, DELETE**, etc.)
- Cada ruta puede tener una o varias funciones de manejador, que se excluyen cuando se correlaciona la ruta.

▸ La definición de ruta tiene la siguiente estructura:

```
app.METHOD(PATH, HANDLER)
```

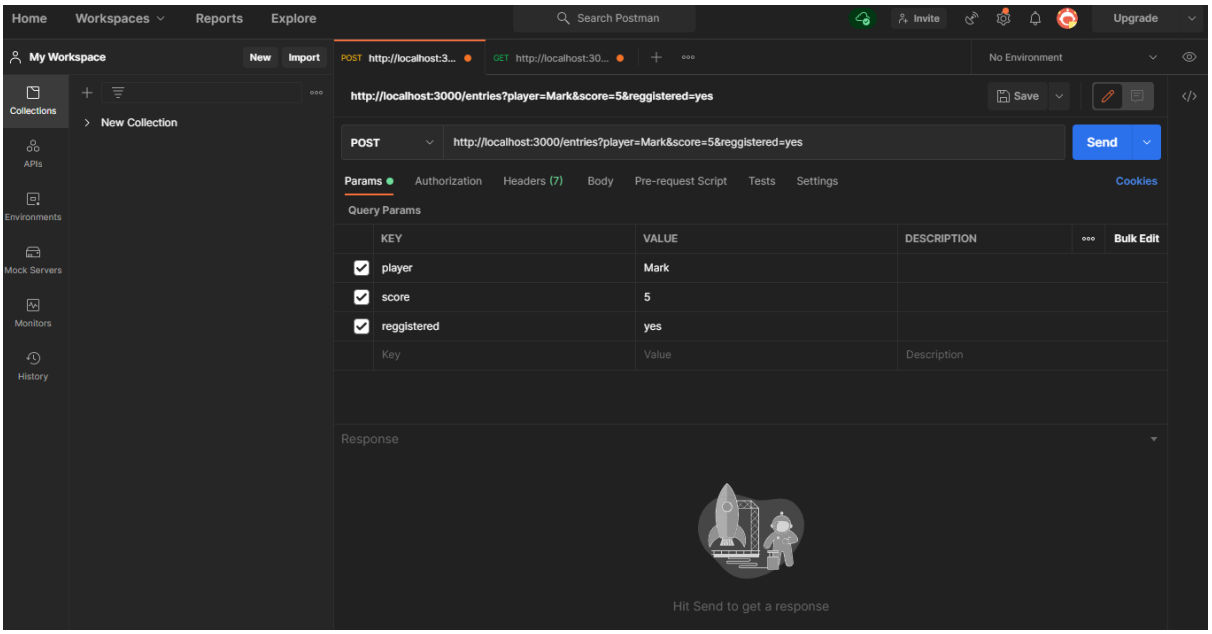
app	: Es una instancia de express.
METHOD	: Es un método de solicitud HTTP.
PATH	: Es una vía de acceso en el servidor.
HANDLER	: Es la función que se ejecuta cuando se correlaciona la ruta.

Por ejemplo:

```
app.get('/', function (req, res) {
  res.send('Hello World!');
});
app.post('/', function (req, res) {
  res.send('Hello World!');
});
app.put('/usuario', function (req, res) {
  res.send('Un request PUT en /usuario');
});
...
```

Request y Response Objects en Express

El objeto **req (Request)** en Express representa el llamado **HTTP** y tiene diferentes propiedades del llamado, como la cadena de texto query (Query params), los parámetros de la URL (URL params), el cuerpo (Body), los encabezados (HTTP headers), etc.



Como por ejemplo así lo hemos visto siempre:

```
app.get("/user/:id", function(req, res) {
  res.send("user " + req.params.id);
});
```

Pero también funcionaria sin problemas:

```
app.get("/user/:id", function(request, response) {
  response.send("user " + request.params.id);
});
```

Exploremos las propiedades más importantes

req.body

Contiene los pares de llave-valor de los datos enviados en el cuerpo (body) del llamado (request). Por defecto es undefined pero es establecido cuando se usa algún “body-parser” middleware como body-parser y multer.
En Postman cuando hacemos un request y enviamos datos en la pestaña Body, estos middlewares son los que nos ayudan a entender el tipo de datos que vamos a recibir en el req.body.
Aquí podemos ver como se pueden usar estos middlwares para establecer el valor del req.body:

```
const app = require("express")();
const bodyParser = require("body-parser");
const multer = require("multer");
const upload = multer(); // Para datos tipo multipart/form-data

app.use(bodyParser.json()); // Para datos tipo application/json
app.use(bodyParser.urlencoded({ extended: true })); // Para datos tipo
application/x-www-form-urlencoded

app.post("/profile", upload.array(), function(req, res, next) {
  console.log(req.body);
  res.json(req.body);
});
```

req.params

Esta propiedad contiene un objeto con las propiedades equivalentes a los parámetros nombrados en la ruta. Por ejemplo, si tenemos una ruta de la forma /user/:name entonces la propiedad name está disponible como req.params.name y allí podremos ver su valor. Supongamos que llamaramos a la ruta con /user/glrodasz, entonces el valor de req.params.name sería glrodasz. Este objeto por defecto tiene el valor de un objeto vacío {}.

```
// GET /user/glrodasz
req.params.name;
// => "glrodasz"
```

Response Object

El objeto **res** representa la respuesta HTTP que envía una aplicación en Express.
Para acceder al res basta con acceder al segundo parámetro de nuestros router handlers (router middleware) o middleware.

Como por ejemplo así lo hemos visto siempre:

```
app.get("/user/:id", function(req, res) {
  res.send("user " + req.params.id);
});
```

Exploremos los métodos más comunes

res.end()

Finaliza el proceso de respuesta. Este método viene realmente del core de Node.js, específicamente del método response.end() de http.ServerResponse.
Se usa para finalizar el request rápidamente sin ningún dato. Si necesitas enviar datos se debe usar res.send() y res.json().

```
res.end();
res.status(404).end();
```

res.json()

Envía una respuesta JSON. Este método envía una respuesta (con el content-type correcto) y convierte el parámetro enviado a una cadena de texto JSON haciendo uso de JSON.stringify().
El parámetro puede ser cualquier tipo de JSON, incluido un objeto, un arreglo, una cadena de texto, un boolean, número, null y también puede ser usado para convertir otros valores a JSON.

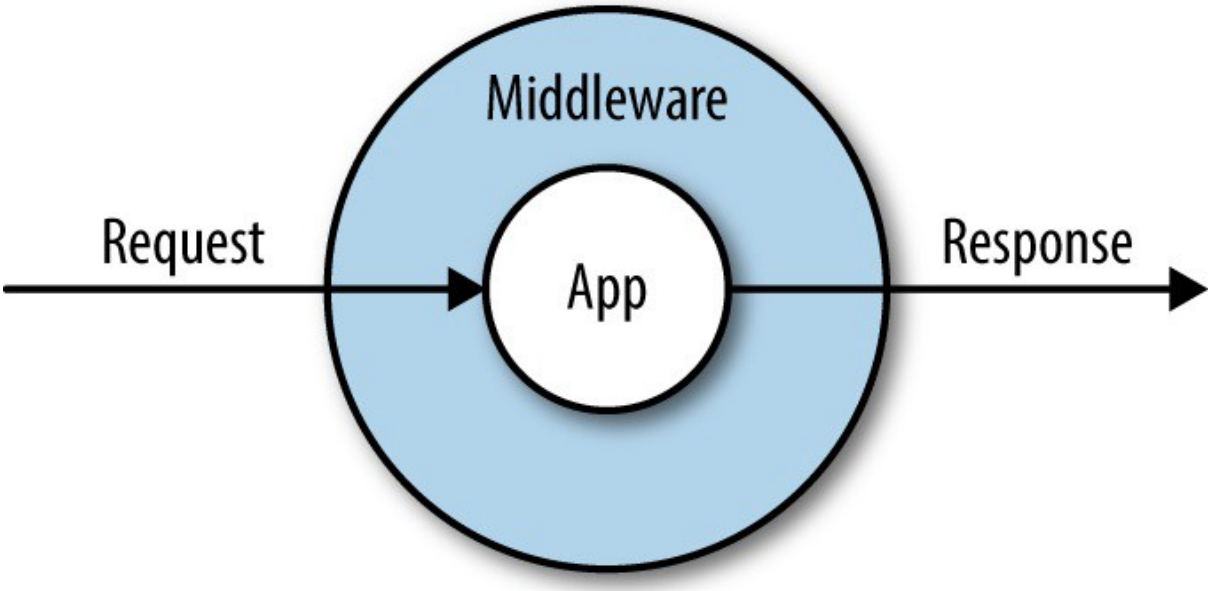
```
res.json(null);
res.json({ user: "tobi" });
res.status(500).json({ error: "message" });
```

res.send()

Envía una respuesta HTTP. El parámetro body puede ser un objeto tipo Buffer, una cadena de texto, un objeto, o un arreglo. Por ejemplo:

```
res.send(Buffer.from("whoop"));
res.send({ some: "json" });
res.send("<p>some html</p>");
res.status(404).send("Sorry, we cannot find that!");
res.status(500).send({ error: "something blew up" });
```

Middleware en Express JS



Un middleware es una función que se puede ejecutar antes o después del manejo de una ruta. Esta función tiene acceso al objeto Request, Response y la función next().

Las funciones middleware suelen ser utilizadas como mecanismo para verificar niveles de acceso antes de entrar en una ruta, manejo de errores, validación de datos, etc.

Veamos el siguiente ejemplo:

Tenemos definida una ruta a la cual solo usuarios administradores pueden ingresar, por lo tanto, se necesita comprobar antes de entrar a esa ruta, si el usuario es o no, un administrador. Para ello se enviará un parámetro llamado isAdmin, que puede tener dos valores: true o false. El cual indicará si el usuario es administrador (valor true).
Primero se creará el servidor y se definirá la ruta para administradores, a continuación puedes ver cómo debería quedar.

```
const app = express();

const port = 3000;

// Permite recibir parámetros en formato JSON.

app.use(express.json());

// Ruta a la cual solo deben ingresar usuarios administradores.

app.get('/dashboard', (req, res) => {

 res.send('You are an admin');

});

app.listen(port, () => {

 console.log(`Server listeting on port ${port}`)

});
```

Una vez creado el servidor, se necesita definir la función middleware, recordemos que un middleware tiene acceso al objeto **Request**, **Response** y la función **next()**, por lo tanto la función que se definirá contará con tres parámetros: **req, res y next.** Los cuales hacen referencia a los objetos mencionados anteriormente.

```
function isAdmin(req, res, next) {

}
```

Después, se agregará la validación para comprobar si el usuario es administrador, utilizando el parámetro **isAdmin** enviado por el cliente, quedando de la siguiente forma:

```
// Middlewa que verifica si el usuario es un administrador.
function isAdmin(req, res, next) {
  if (req.body.isAdmin) {
    next();
  } else {
    res.status(403).send(`Sorry but you are not an admin and you do not have access to route ${req.url}`);
  }
}
```

Como puedes ver si el valor de **req.body.isAdmin** es igual a true, entonces se pasa el control de la petición a la declaración de la ruta, utilizando el parámetro **next**. En caso contrario, se envía un mensaje al usuario advirtiendo que no es un administrador, por lo tanto no tiene permitido ingresar a la ruta **/dashboard.**

Por último se debe agregar la función middleware a la aplicación:

app.use(isAdmin);

Nota: "Se debe colocar antes de declarar la ruta"

Codigo Completo

```
const express = require('express');
conconst express = require('express');
const app = express();
const port = 3000;

// Middleware que verifica si el usuario es un administrador.
function isAdmin(req, res, next) {
  if (req.body.isAdmin) {
    next();
  } else {
    res.status(403).send(`Sorry but you are not an admin and you do not have access to route ${req.url}`);
  }
}

// Permite recibir parámetros en formato JSON.
```

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

// Se agrega el middleware en la aplicación.
app.use(isAdmin);

// Ruta a la cual solo deben ingresar usuarios administradores.
app.get('/dashboard', (req, res) => {
  res.send('You are an admin');
});


app.listen(port, () => {
  console.log(`Server listeting on port ${port}`)
});
st app = express();
const port = 3000;

// Middlewa que verifica si el usuario es un administrador.
function isAdmin(req, res, next) {
  if (req.body.isAdmin) {
    next();
  } else {
    res.status(403).send(`Sorry but you are not an admin and you do not have access to route ${req.url}`);
  }
}

// Permite recibir parámetros en formato JSON.
app.use(express.json());

// Se agrega el middleware en la aplicación.
app.use(isAdmin);

// Ruta a la cual solo deben ingresar usuarios administradores.
app.get('/dashboard', (req, res) => {
  res.send('You are an admin');
});


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

Pruebas

A continuación se muestran las respuestas obtenidas para los casos definidos, el primero cuando el usuario sea un administrador y el segundo cuando no lo es. Se utilizará Postman para realizar ambas peticiones.