UTN FACULTAD REGIONAL CÓRDOBA
INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Desarrollo de Software
Notas de Clase

Apunte 14 - Express - Router

Ahora vamos a ocuparnos de las rutas en una aplicación web. Las rutas son esencialmente las URL que definen la estructura y navegación de la aplicación. Cada ruta corresponde a una página o recurso específico.

En el contexto de un servidor web o una aplicación de backend, una ruta es una definición de cómo el servidor debe responder a una solicitud HTTP específica. Esto incluye qué código se debe ejecutar y qué datos se deben devolver.

Por ejemplo, en una aplicación de blog, podrías tener rutas como:

/ para la página de inicio /posts para la lista de todas las publicaciones del blog /posts/:id para ver una publicación individual del blog (donde :id es un parámetro que representa el ID de la publicación) /about para la página "Acerca de"

Vamos a trabajar con la clase **express.router** que tienen como finalidad crear manejadores de rutas montables y modulares. Una instancia router es un sistema de middleware y direccionamiento completo; por este motivo, funciona como una miniaplicación que se encarga de gestionar las rutas.

En el siguiente ejemplo creamos un *"router"* o un enrutador, definimos dos rutas que tienen relación en el archivo **articulos.js** y posteriormente se monta el módulo de router en la aplicación principal en el archivo **index.js**.

- Archivo: **articulos.js**

```
const router = express.Router();

router.get("/api/articulos", async function (req, res, next) {
  //Código del método
});

router.get("/api/articulos/:id", async function (req, res, next) {
  //Código del método
});
```

- archivo: **index.js**

```
const articulosRouter = require("./routes/articulos");
app.use(articulosRouter);
```

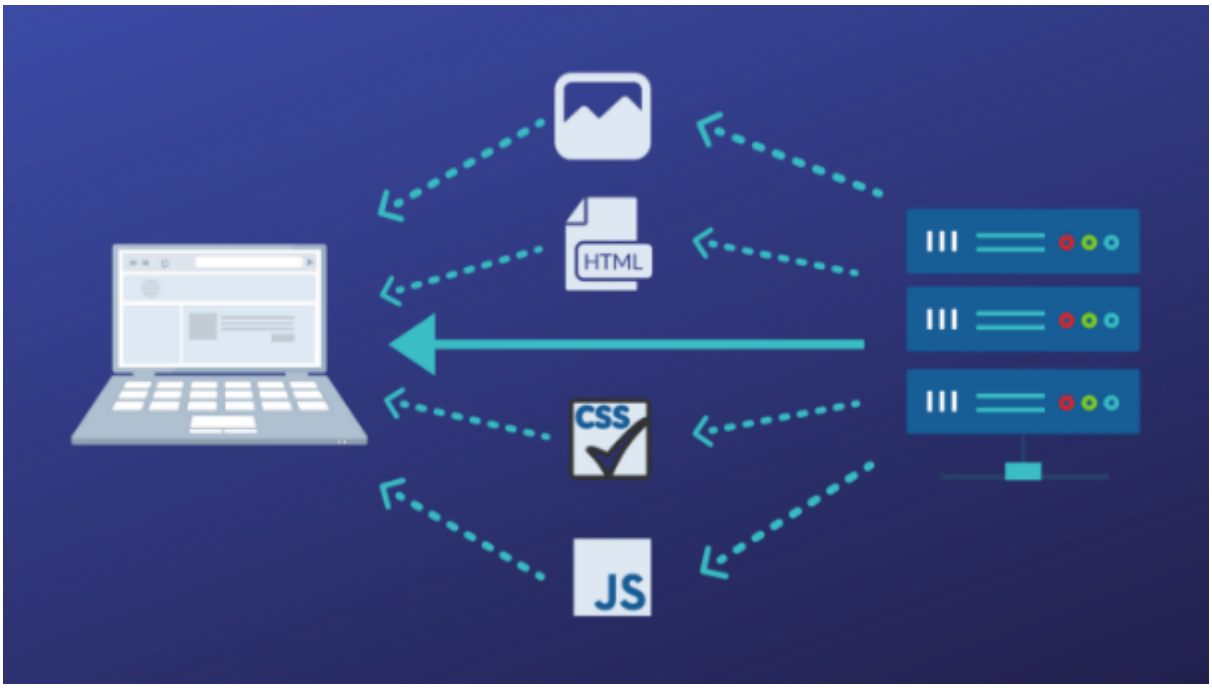
Este enrutador define dos rutas:

"/api/articulos": Esta ruta maneja las solicitudes GET a la URL "/api/articulos". Cuando se recibe una solicitud a esta URL, se ejecuta la función de callback asincrónica proporcionada.

"/api/articulos/:id": Esta ruta maneja las solicitudes GET a la URL "/api/articulos/:id", donde ":id" es un parámetro que puede ser cualquier valor. Al igual que la ruta anterior, cuando se recibe una solicitud a esta URL, se ejecuta la función de callback asincrónica proporcionada.

Finalmente, en el archivo index.js, el enrutador de artículos se importa y se utiliza en la aplicación Express con app.use(articulosRouter). Esto significa que todas las rutas definidas en el enrutador de artículos se añadirán a la aplicación Express.

Sirviendo archivos estáticos



Recordemos que son los archivos estáticos, son recursos que se pueden entregar al cliente tal como son, sin necesidad de ser generados, manipulados o procesados por el servidor de ninguna manera. Estos incluyen archivos como imágenes, CSS y JavaScript.

Cuando usas express.static('public'), estás diciendo a Express que cualquier solicitud de un archivo estático debe buscar en el directorio 'public'. Por ejemplo, si tienes un archivo llamado 'style.css' en el directorio 'public', y el cliente solicita `http://tu-sitio-web/style.css`, Express servirá el archivo 'public/style.css'.

El código app.use(express.static('public')); en Express.js se utiliza para servir archivos estáticos desde un directorio llamado 'public' en el directorio raíz de tu aplicación.

```
app.use(express.static('public'));
```

Podríamos tener algunas imágenes en el subdirectorio public/images de la siguiente manera:

```
index.js
public/
public/images
public/images/logo.png
```

Al ejecutar la aplicación y abrir el explorardor en en la URL: http://127.0.0.1:3000/images/logo.png y podes ver la imagen correspondiente a su archivo *logo.png*.

Usando el módulo path y la propiedad _dirname

El módulo "path" de Node.js proporciona utilidades para trabajar con rutas de archivos y directorios. Podes usarlo para manipular rutas de archivos, como unir diferentes segmentos de ruta, resolver una ruta completa, obtener la extensión de un archivo, entre otras cosas. Esto es útil en Express cuando necesitas referenciar archivos que queres exponer, como archivos estáticos.

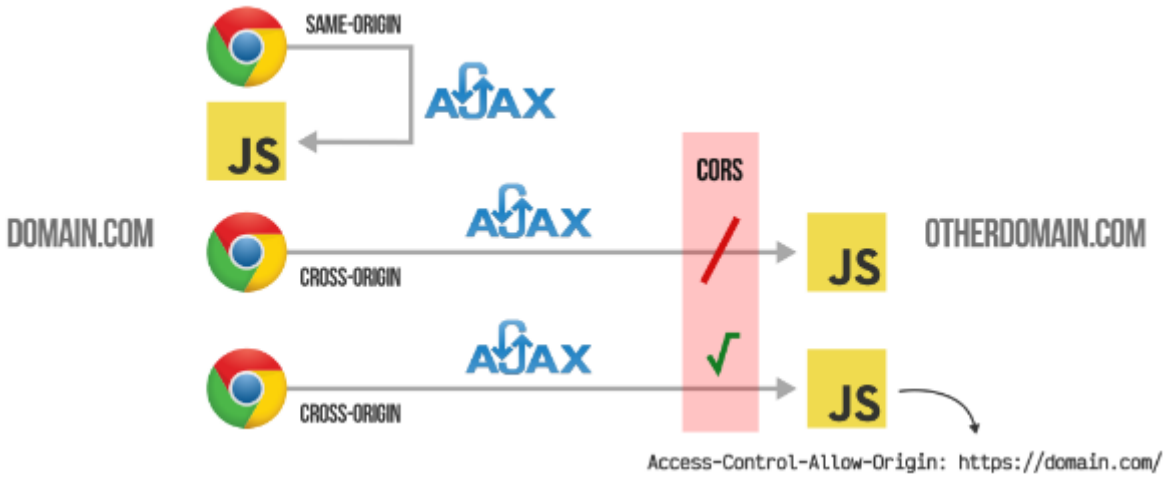
En este ejemplo, path.join(__dirname, 'public') está creando una ruta completa al directorio 'public' que está en el mismo directorio que el archivo de script de Node.js que se está ejecutando.

La propiedad "__dirname" de Node.js es una variable global que contiene la ruta del directorio del archivo de script de Node.js que se está ejecutando actualmente. Es útil cuando necesitas referenciar archivos en relación con el archivo de script actual, en lugar de en relación con el directorio de trabajo actual.

En el ejemplo anterior, __dirname se usa para crear una ruta al directorio 'public' que está en el mismo directorio que el archivo de script de Node.js que se está ejecutando. Esto asegura que la ruta al directorio 'public' sea correcta, sin importar desde dónde se ejecute el script.

CORS

CORS significa **Cross-Origin Resource Sharing**, es una política a nivel de navegador web que se aplica para prevenir que el dominio determinado evite acceder a recursos de otro dominio usando solicitudes del tipo AJAX como cuando usamos **fetch()** o **XMLHttpRequest** o cualquier librería como por ej Axios



Referencia: https://lenguajejs.com/javascript/peticiones-http/cors/

Lo primero que necesitamos saber es que si tenemos dos dominios, por ejemplo dds-frontend.com y dds-backend.com en principio no pueden comunicarse. Si nosotros queremos que por ejemplo dds-backend.com pueda permitir a otros dominios acceder a sus recursos, podemos hacerlo a través del módulo de cors.

Entonces tenemos que instalar y configurar dicha librería...

- Comando para instalar:

```
npm i cors
```

- configurar en el sistema

```
app.use(cors());
var express = require('express');
var cors = require('cors');
var app = express();
app.use(cors());
app.get('/products/:id', function (req, res, next) {
  res.json({msg: 'This is CORS-enabled for all origins!'})
});

app.listen(80, function () {
  console.log('CORS-enabled web server listening on port 80')
});
```

Con esto ya estamos permitiendo a nuestro dominio recibir solicitudes de otros dominios. Pero si queremos limitar solo a ciertos dominios acceder a nuestros recursos podemos igual hacerlo a través de una lista blanca, en donde definimos los dominios y validamos que cada vez que haya una solicitud a una ruta específica se ejecute ese procedimiento de confirmación para aprobar o descargar el dominio.

En nuestro código paso a paso podemos ver su configuración en el archivo **index.js**

Referencias

Paso a Paso backend backend online backend

Paso a Paso front frontend online frontend