<codoacodo/>

Agencia de
Aprendizaje
a lo largo
de la vida

Desarrollo Fullstack

BA

Les damos la bienvenida

Vamos a comenzar a grabar la clase

<codoacodo/>

Clase 27

Node JS

- Request y Response
- GET
- Rutas Parte I
- Path Params
- Query Params

Clase 28

Node JS

- Express Router
- Body Parser
- Method Override
- Middlewares

Clase 29

Node JS

- Error 404
- Controladores
- ENV

NODE JS

Rutas II

Rutas

Como vimos hasta el momento, la comunicación entre clientes y servidores o entre programas que interactúan a través de la web, se hace **mediante rutas**.

A estas **rutas** se las conoce comúnmente como ENDPOINTS y necesitaremos uno por cada flujo que posea nuestro servidor.

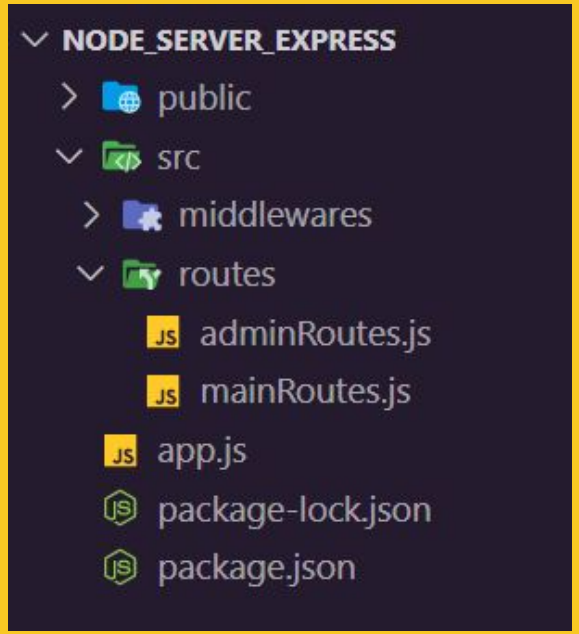
Ahora si, nuestras rutas se van complejizando.

Express Router

Permite separar rutas en diversos archivos que van a devolver a cada parte de nuestro programa.

Primero, **ordenemos nuestro código** creando una **carpeta src** para guardar toda la lógica de nuestro programa.

Dentro creamos una carpeta nueva llamada routes.



Express Router

En el archivo mainRoutes.js llamamos al método Router() de express.

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

Express Router

Luego, creamos o migramos desde el archivo app todas nuestras rutas:

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

/* MAIN ROUTES */

router.get('/home', (req, res) => res.send("Página de Home"));
router.get('/contact', (req, res) => res.send("Página de Contacto"));
router.get('/about', (req, res) => res.send("Página Sobre Nosotros"));
```

En lugar de app.get(...) utilizamos **router.get(...)**

Express Router

Una vez definidas todas las rutas o **endpoints** debemos exportar el módulo router para ser utilizado desde app.js:

```
module.exports = router;
```

Express Router

Finalmente llamamos nuestro archivo de rutas desde app.js y a través del **middleware** app.use() indicamos que peticiones deben ser respondidas con esas rutas.

```
const express = require('express');
const app = express();
const mainRoutes = require('./src/routes/mainRoutes.js')

const PORT = 3000;

app.use(express.static('public'));
app.use('/', mainRoutes);

app.listen(PORT, () => console.log(`Servidor corriendo en http://localhost:${PORT}`));
```

Probemos nuestros endpoints.

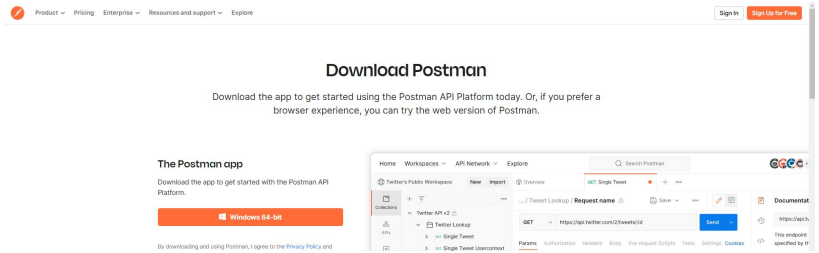
POSTMAN/INSOMNIA

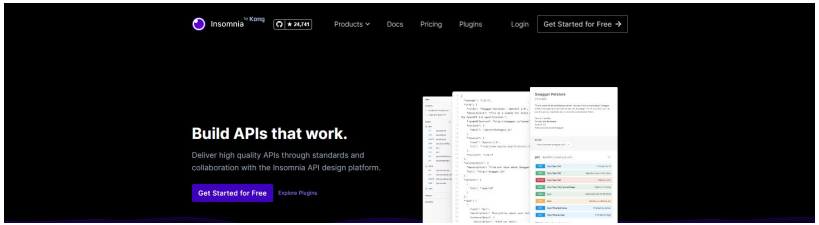
Para lograr esto podemos usar programas que nos permiten **“simular”** estas peticiones a través de los distintos métodos HTTP.

Estas herramientas facilitan enviar consultas a nuestro servidor sin **depender de un Frontend armado**.

https://www.postman.com/

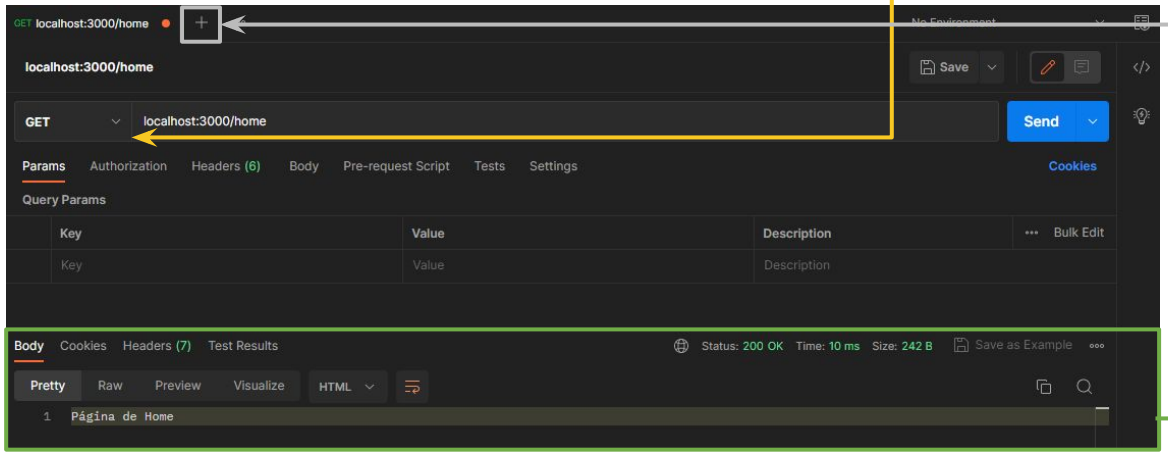
https://insomnia.rest/download





POSTMAN

Dentro de **POSTMAN** podemos crear una nueva petición con el botón de **+**, luego seleccionamos el método (en este caso **GET**) y la **url** a consultar.



Finalmente presionamos **SEND** y esperamos la respuesta del servidor.

<codoacodo/>

De esta manera probamos nuestro backend sin un frontend.

Agencia de Aprendizaje a lo largo de la vida

Buenos Aires Ciudad
Educación

Ahora con POST.

Parseando datos recibidos

Hasta el momento venimos trabajando con **GET**, pero para que nuestro servidor pueda recibir peticiones por POST necesitamos *convertir* los datos recibidos en el **BODY** a un formato que **entienda el servidor**.

app.js

```
/*
* Convertimos los datos entrantes en formato
* application/x-www-form-urlencoded y application/json
* a un formato entendible por el servidor
*/

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

Usando los **middlewares** nativos .urlencoded() y .json() podemos convertir la data de estos formatos a uno que el **servidor pueda manejar**.

***nota:** en versiones previas a **express 4.16.0** se utilizaba una librería llamada body-parser para este propósito.*

Finalmente con PUT y DELETE

Sobre escribiendo métodos

Los navegadores web, por ejemplo desde un formulario únicamente soportan los métodos **GET** y **POST**, por lo que cuando deseamos utilizar un método diferente, es preciso **sobreescribirlo**.

```
npm install method-override
```

```
const methodOverride = require('method-override');

// Override para habilitar los métodos PUT y DELETE

app.use(methodOverride('_method'));
```

Para esto utilizaremos una dependencia llamada **method-override**, otro middleware que captura la petición y si posee una mención a **algún método no soportado**, **lo sobreescribe**.

***nota:** en versiones previas a **express 4.16.0** se utilizaba una librería llamada body-parser para este propósito.*

Sobre escribiendo métodos

Ahora desde nuestros **formularios HTML**, solo debemos anteponer `?_method=PUT` o `?_method=DELETE` en la URL del atributo action para indicar el **método real** y utilizar **POST** en el atributo method del formulario.

```
<form
    action="/admin/edit/20?_method=PUT"
    method="POST"
    enctype="multipart/form-data"
>
  ...
</form>
```

Para terminar, hay una palabra que se usó mucho pero no vimos...

¿Cuál era? 🤔

Ahh, si.

Middleware

Middleware

Casi de forma instintiva, ya utilizamos varios middlewares para configurar nuestro programa.

¿Qué es un middleware entonces?

Son simplemente funciones que se ejecutan antes o después de otras y los hay de distintos tipos:

- **Middlewares de nivel de aplicación**: Son middlewares que se aplican a toda la aplicación y se configuran utilizando `app.use()`
- **Middlewares de nivel de ruta**: Son middlewares que se aplican a una ruta específica.
- **Middlewares de manejo de errores:** Son middlewares especiales que se utilizan para manejar errores en la aplicación.

A lo largo de las clases iremos viendo distintos middlewares y también aprenderemos a crear los propios.

No te olvides de dar el presente

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.

Todo en el Aula Virtual.

Gracias