

A graphic on the left side of the slide. It features a stack of four horizontal bars in purple, orange, yellow, and blue. The text 'Agencia de Aprendizaje a lo largo de la vida' is written across these bars in white. An orange arrow points to the right from the end of the orange bar.

Agencia de
Aprendizaje
a lo largo
de la vida

FULL STACK

Clase 27

Node 3

Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 26

Node

- Módulos
- Gestores de Paquetes
- Creación de un servidor
- Implementaciones de Node

Clase 27

Node

- Express
- Servidor estático
Nodemon
Rutas

Clase 28

Node



Un **framework**
a la medida.

Frameworks Node

Montar un server de forma nativa con Node para **proyectos robustos** resulta **algo tedioso y difícil de escalar**.

Por eso **existen diversos Frameworks** de Node como HapiJS, Koa, NestJS o **Express**, entre otros.

Este último es el más popular y actualmente se encuentra bajo el soporte de la **OpenJS Foundation**.

<https://openjsf.org/about/>

Express JS

Es un framework de aplicaciones web mínimo y flexible en el entorno de NodeJS.

Posee métodos y middlewares para **programas HTTP** que **facilitan la creación de una API sólida de forma rápida y sencilla.**

Una de las ventajas de Express es que nos **permite levantar un servidor web muy fácilmente.**

Pero antes de eso, debemos preparar nuestro proyecto para trabajar con librerías.



Express JS

En la terminal corremos el comando:

```
npm init -y
```

Instalamos Express mediante npm:

```
npm install express --save
```

El flag `--save` indica que debe registrar la **dependencia** y sus subdependencias **actualizadas**.

A continuación, vemos en el package.json que contamos con una lista de dependencias con express en la versión que acaba de instalar.

```
•>> node_server_express 02:54 npm init -y
Wrote to C:\Users\pol_m\Desktop\node_server_express\package.json:

{
  "name": "node_server_express",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```



```
•>> node_server_express 03:00 npm install express --save

added 57 packages, and audited 58 packages in 60s

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Inspeccionando package.json

```
{  
  "name": "noder_server_express",  
  "version": "1.0.0",  
  "description": "",  
  "main": "app.js",  
  ▶ Debug  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "express": "^4.18.2" ←  
  }  
}
```


Ya tenemos **Express** instalado.

¡Es momento de crear un
nuevo server!

Servidor estático con Express

Para el código del servidor:
Creamos el archivo **app.js** en él
importamos el módulo de express.

Luego ejecutamos la función
`express()` y la guardamos en una
variable llamada **app**.

```
//express es una función que contiene un objeto
const express = require('express');

//Instanciación o inicializacion de la funcion express()
dentro de la variable app
const app = express();
```

Servidor estático con Express

Una vez importado el módulo y ejecutada una instancia de express tenemos que definir el puerto que va a estar escuchando nuestro servidor y configurar nuestra primera ruta con la respuesta a su petición:

El método **.get()** de app escuchará las peticiones a la ruta / a través del método HTTP GET y responderá el texto citado.

El método **.listen()** recibe un parámetro port con el puerto donde correrá el server y un callback que en este caso lo usamos para enviar un mensaje por consola.

```
const port = 3000;

app.get("/miRuta", (req, res) => {
  res.send('Hola Mundo!');
});

app.listen(port, () => {
  console.log(`Example app listening`+
    `at http://localhost:\${port}`);
});
```

Servidor estático con Express

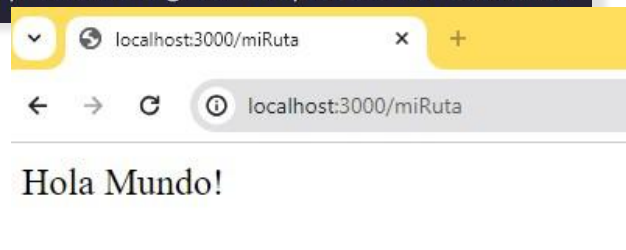
Para ejecutar nuestro servidor podemos hacerlo igual que antes mediante la terminal con:

```
node app.js
```

o definiendo un script para ello en nuestro archivo package.json:

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "start": "node app.js" },
```

```
o >> noder_server_express 03:18 npm start  
  
> noder_server_express@1.0.0 start  
> node app.js  
  
Example app listening at http://localhost:3000
```



De esta manera **podemos devolver**
un **texto** o un **archivo estático** que
responda a una ruta específica
mediante el **método .get()**

Desafío I: tabajando con express

1. Iniciar un proyecto.
2. Instalar npm install express --save.
3. Crear el punto de acceso app.js
4. Desarrollar el servidor estático que, a la petición '/saludo', responda con un saludo en el navegador.
5. Modificar el Script en el package para ejecutarlo con npm start.
6. Ejecutarlo y enviar la solicitud utilizando el navegador y puerto 3000.
7. Levantalo a tu github como parte de tu repo.



https://github.com/Ferlucena/Clase26_Node_Express.git

Archivos Estáticos

Veamos cómo podemos, con Express, **definir una carpeta** que **sirva archivos** tal como lo haría un **servidor estático**.

Pero **antes un paso es necesario ver nodemon** 🐉 ...

¿Qué es Nodemon?

Es una herramienta que se utiliza en el desarrollo de aplicaciones Node.js para automatizar el proceso de reinicio del servidor cada vez que se detectan cambios en los archivos del proyecto. Esta librería nos ayuda recargando el servidor frente a cada cambio, sin tener que hacerlo manualmente.

La instalamos como dependencia de desarrollo:

```
npm install -D nodemon
```

* Las dependencias de desarrollo son aquellas que solo se utilizarán mientras creamos el proyecto.

Una vez lista, **modificamos ligeramente el script** de nuestro package.json por:

```
"scripts": {  
  "start": "nodemon app.js"  
}
```



Node v18+

Desde la versión 18 de **NODE** no se necesita instalar Nodemon ya que el mismo programa cuenta con una funcionalidad propia para recargar nuestro servidor, el flag `--watch`.

Al ser una característica (feature) tan reciente y dado que no todas las PCs son compatibles con la última versión, por el momento se puede trabajar con Nodemon.



```
"scripts" : {  
  "start": "node --watch app.js"  
}
```

Ahora crearemos un proyecto desde cero que sirva archivos estáticos.

Carpeta public

Lo primero es **crear** una carpeta **public**.

Allí irán **todos los archivos** que deberán ser enviados tal como fueron alojados.

Luego en nuestro archivo **app.js** agregaremos **la ruta a esta carpeta**, indicando a Express de que se trata.

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

El método **.use()** es un **middleware**, concepto que veremos más adelante.

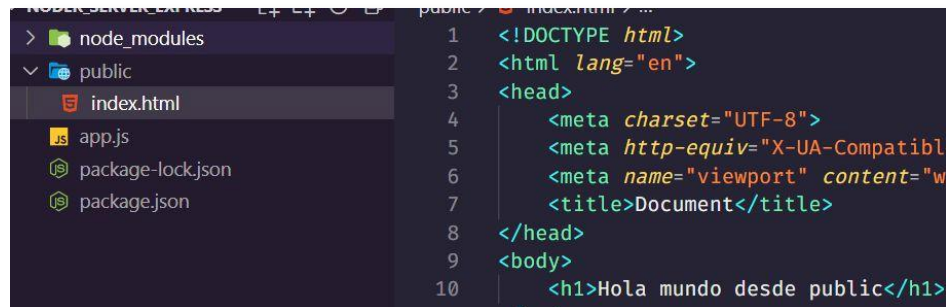
Por el momento debemos saber que nos permite interceptar lo que se ejecute dentro antes que la derivación de nuestras rutas.



Archivos estáticos

Ahora creamos un archivo **index.html** y accedemos a la ruta **http://localhost:3000**

¡Magia! Nos devuelve nuestro archivo **HTML**.



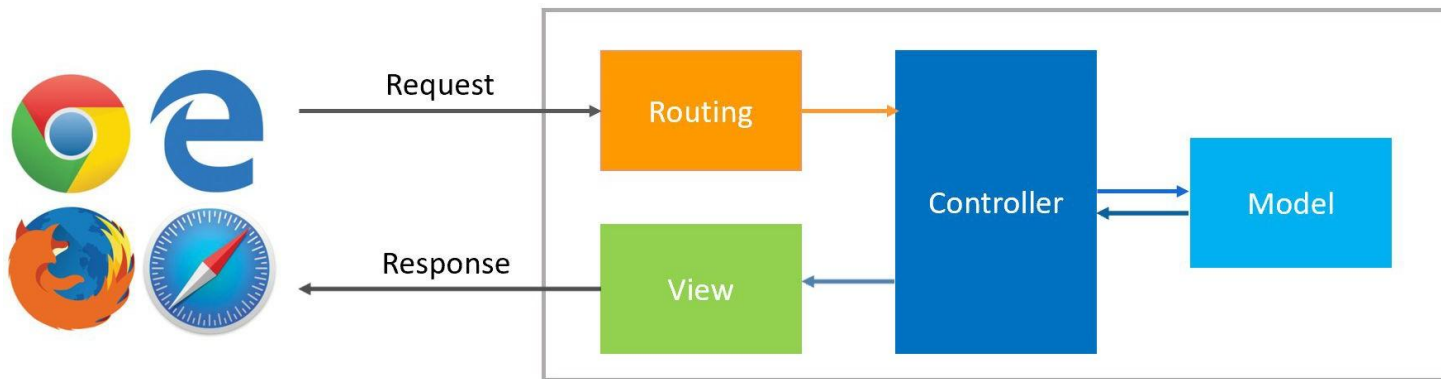
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="width=device-width, initial-scale=1">
6   <meta name="viewport" content="width=device-width, initial-scale=1">
7   <title>Document</title>
8 </head>
9 <body>
10  <h1>Hola mundo desde public</h1>
```



Así como **index.html**,
podemos devolver cualquier
recurso a través de su ruta.

Rutas

El **cliente buscará acceder** al contenido **NO ESTÁTICO** de nuestro Backend a través de **peticiones HTTP** a diferentes **rutas o endpoints** configurados en nuestra aplicación.



Rutas

Nuestras rutas serán **definidas en Express** de la siguiente manera:

```
app.get('/nosotros', (req, res) => {  
  res.sendFile(__dirname + './nosotros.html');  
})
```

El método **get** de app escuchará las peticiones a la ruta **/nosotros** a través del método HTTP GET y responderá el archivo solicitado.

La variable **app** de express puede escuchar a todos los métodos HTTP, entre ellos **GET**, **POST**, **PATCH**, **PUT** y **DELETE**, entre otros.

__dirname nos permite tomar como referencia el lugar actual de nuestro archivo dentro del servidor y llegar a un recurso desde esa ruta.

Las rutas son la manera que tiene nuestro servidor de exponer contenido *“no estático”* a través de la web.

Rutas

Recordemos que al usar el **protocolo HTTP**, las peticiones o **requests** se hacen mediante el uso de los HTTP methods.

GET

POST

PATCH

PUT

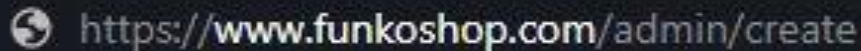
DELETE

Rutas

Por ende, nuestro servidor no solo escuchará **“paths”** o rutas si no que también tendrá en cuenta el método utilizado en la request.

Esto nos permite usar la misma ruta para diferentes cosas.

No es lo mismo solicitar la ruta “/admin/create” mediante GET en la URL:



Que utilizar un formulario para enviar datos a través de POST:

CREAR NUEVO ITEM

Categoría: Licencia:

Nombre del producto: Kakashi Hatake Shippuden Saga

Descripción del producto

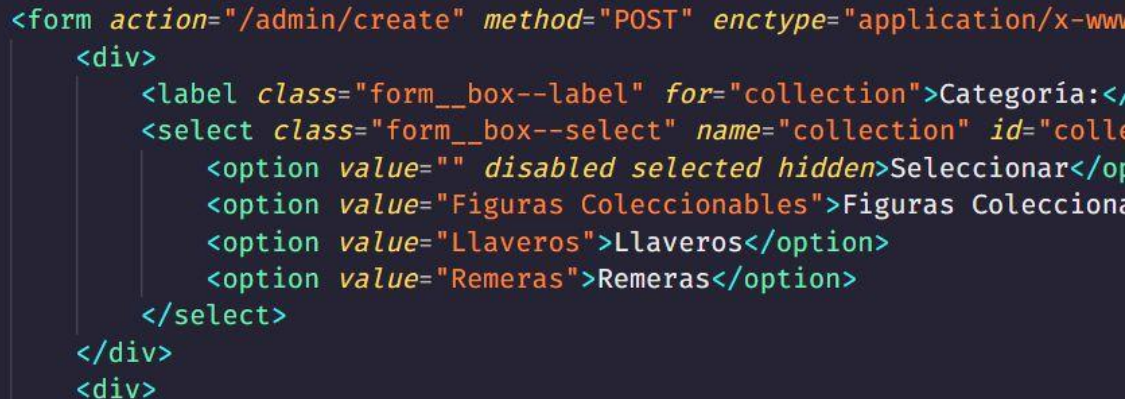
SKU: SSK111AB001 Precio: \$ 0.000,00 Stock: 0

Descuento: 0% Cuotas: 3 Cuotas sin interés

Imágenes: No se ha seleccionado ningún archivo

Agregar Producto


Limpiar



```
<form action="/admin/create" method="POST" enctype="application/x-www
<div>
  <label class="form__box--label" for="collection">Categoría:</
  <select class="form__box--select" name="collection" id="colle
    <option value="" disabled selected hidden>Seleccionar</op
    <option value="Figuras Coleccionables">Figuras Colecciona
    <option value="Llaveros">Llaveros</option>
    <option value="Remeras">Remeras</option>
  </select>
</div>
</div>
```

Rutas

Nuestras rutas serán definidas en Express de la siguiente manera:



El método **get** de **app** escuchará las peticiones a la ruta **/admin/create** a través del método **HTTP GET** y responderá el archivo solicitado.

```
app.get('/admin/create', (req, res) => {  
  res.send(__dirname + './create.html');  
})
```

la variable **app de express puede escuchar a todos los métodos HTTP, entre ellos **GET**, **POST**, **PATCH**, **PUT** y **DELETE**, entre otros.*

__dirname nos permite tomar como referencia **el lugar actual de nuestro archivo** dentro del servidor y **llegar a un recurso** desde esa ruta.

Respondiendo a rutas

Tenemos un archivo `items.json`, el cual **queremos leer** y **devolver** cuando un cliente pida la ruta `"/items"`.

```
app.get("/items", (req, res) => {  
  const getItems = fs.readFileSync(_dirname + '/data/items.json');  
  res.send(JSON.parse(getItems));  
})
```

En este caso si **entramos** a `localhost:3000` seguiremos obteniendo nuestro archivo `index.html` (estático) pero si solicitamos esta ruta nos devolverá un **array con los productos** en formato JSON.

*`readFileSync` nos devuelve un string o cadena de texto con la información y nosotros la convertimos a JSON a través del método `JSON.parse()`.

Desafío II: tabajando con express

1. Iniciar un proyecto.
2. Instalar express.
3. Crear el punto de acceso app.js
4. Desarrollar el servidor estático que, a la petición '/', sirva el archivo index.html de la carpeta public.
5. Modificar el Script en el package para ejecutarlo con npm start y programarlo para que quede escuchando a los cambios con nodemon o node --watch.
6. Ejecutarlo y enviar la solicitud utilizando el navegador y puerto 3000.
7. Crear por último 3 rutas distintas.
8. Levantalo a tu github como parte de tu repo.



https://github.com/Ferlucena/clase26_Express_ServerEstaticoDesde0.git

Rutas Parametrizadas

Rutas parametrizadas - params

Ahora supongamos que queremos **traer solo un ítem de la lista**

¿Cómo podríamos resolverlo?

Las respuestas son las “***rutas parametrizadas***”, gracias a ellas podemos leer una parte de la URL y utilizarla para devolver una respuesta diferente según el caso.

```
app.get("/items/:id", (req, res) => {  
  const id = req.params.id;  
  
  // lógica  
})
```


Rutas parametrizadas - params

En este caso **:id** será un valor que pasaremos en la URL y será leído al momento de recibir la petición.

A través de **params**, capturamos el valor de la URL.

```
app.get("/items/:id", (req, res) => {  
  const id = req.params.id;  
  // lógica  
})
```

Rutas parametrizadas - query

Otra manera de **pedir datos a través de la URL** es mediante los **querys**.

En las rutas parametrizadas en lugar de incluir los parámetros directamente en la ruta de la URL, como en las rutas parametrizadas convencionales, **los parámetros de consulta se incluyen como pares clave-valor** en la parte de la URL que **sigue al signo de interrogación ?**

<http://localhost:3000/busqueda?palabra=perro&tipo=animal>

Rutas parametrizadas - query

<http://localhost:3000/busqueda?palabra=perro&tipo=animal>

En esta URL, **busqueda es la ruta**, y **palabra y tipo son los parámetros de consulta**, palabra=perro y tipo=animal son los valores asociados a esos parámetros.

En este caso el cliente está buscando resultados relacionados con la palabra "perro" y el tipo de resultado es "animal".

En el servidor, estos parámetros de consulta pueden ser recuperados y utilizados para realizar acciones específicas, como realizar consultas en una base de datos o generar contenido dinámico para el cliente en función de los valores de los parámetros de consulta.

Rutas parametrizadas - query

En otro ejemplo de **pedir datos a través de la URL** es mediante los **querys**.

`http://localhost:3000/items?licence=pokemon`

```
app.get("/items", (req, res) => {  
  const licence = req.query.licence; //pokemon  
  // lógica que filtra los ítems de la licencia pokémon  
})
```

Rutas parametrizadas - query

Usamos **la misma ruta que para todos los items**, solo que si recibimos un **query param**, capturamos el valor de la URL e incluimos una lógica para devolver solo los valores coincidentes.

Cuando se recibe una solicitud en esta ruta, Express.js analiza automáticamente los parámetros de consulta de la URL y los hace accesibles a través del objeto **req.query**. En este caso, estamos **extrayendo el valor del parámetro licence** con **req.query.licence**.

```
app.get("/items", (req, res) => {  
  const licence = req.query.licence; //pokemon  
  // lógica que filtra los ítems de la  
  // licencia pokémon  
})
```

No te olvides de dar el presente

Recordá:

- **Revisar la Cartelera de Novedades.**
- **Hacer tus consultas en el Foro.**
- **Realizar los Ejercicios obligatorios.**

Todo en el Aula Virtual.

Muchas gracias por tu atención.

Nos vemos pronto