

Agencia de
Aprendizaje
a lo largo
de la vida

Desarrollo Fullstack



Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 25

Node JS

- ▶ Módulos
- ▶ Node Package Manager
- ▶ Servidor Web Node Nativo
- ▶ Enviar Texto
- ▶ Enviar Archivos

Clase 26

Express JS

- ▶ Express JS
- ▶ Express Generator
- ▶ Servidor Estático con Node

Clase 27

Node JS

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

NODE JS

Express





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**.

Express JS

Es un framework de aplicaciones web NODE mínimo y flexible.

Posee miles 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.

En la terminal corremos el comando:

```
npm init -y
```

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

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

```
✓ NODER_SERVER_EXPRESS  [F] [F] [U] [C]
  JS app.js
  JS package.json
```

Express JS

Ahora nos toca **instalar Express** mediante **npm**:

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

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

```
● >> noder_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
```

```
{
  "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"
  }
}
```

Vemos en el package.json que contamos con una lista de dependencias con express en la versión que acaba de instalar.

Ya tenemos **Express** instalado.

¡Es momento de crear un nuevo server!

Servidor Estático con Express

Borramos el contenido de nuestro archivo `app.js` e importamos el módulo de `express`.

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

```
const express = require('express');  
  
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:

```
const port = 3000;

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

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

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

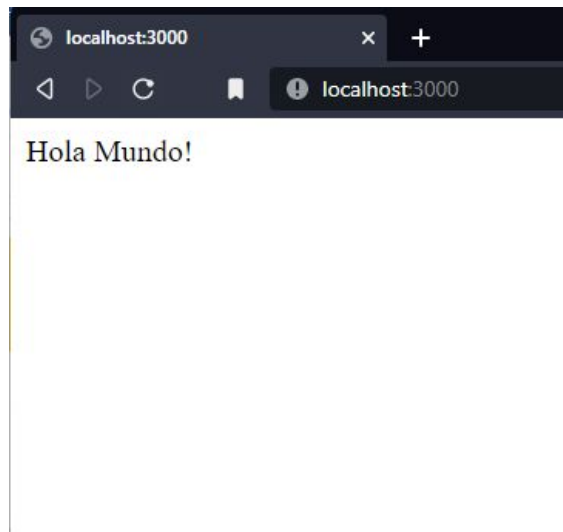
El método **listen** recibe un parámetro **port** con el puerto donde correrá el server y un callback que en este caso lo usamos para enviar un mensaje por consola.

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

Express Generator

Es otra forma de instalar **Express** con un *boilerplate* ya configurado.

Para poder usar este **comando**, hay que instalar express generator en nuestra **PC** de forma **global**:

```
npm install express-generator -g
```

Express Generator

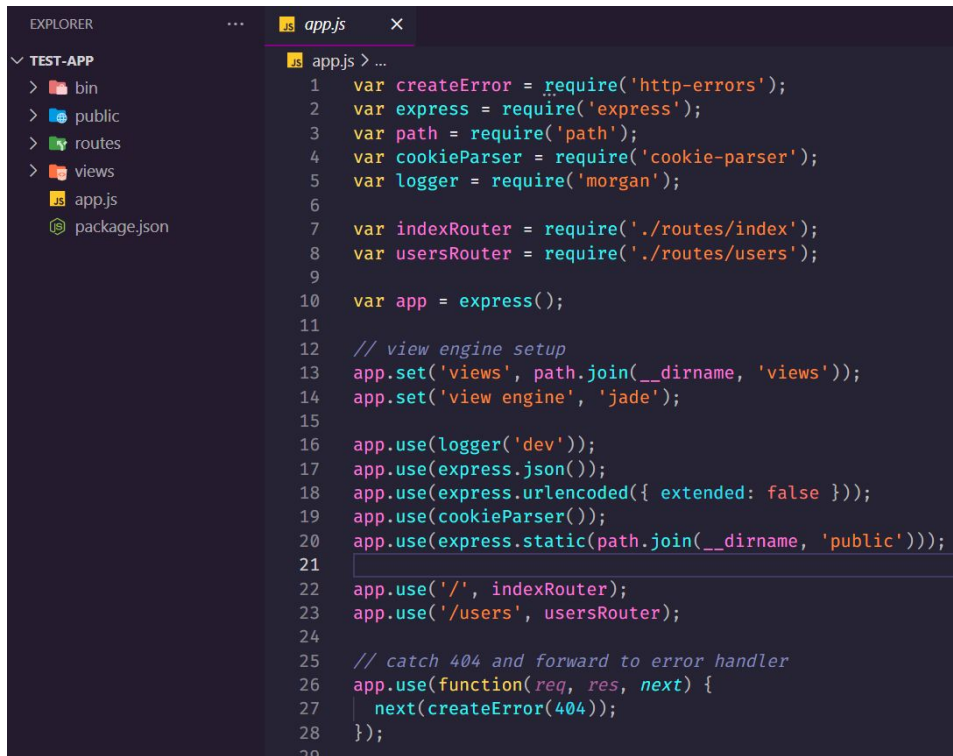
Una vez instalado podemos correr el comando:

```
express test-app
```

Esto nos creará un proyecto nuevo en una carpeta llamada **“test-app”** con una serie de archivos y carpetas ya preparados para comenzar a desarrollar nuestra app.

De esta manera tenemos nos ahorramos la configuración básica de express

****No olvidar correr el comando npm install para instalar todas las dependencias necesarias.***



```
EXPLORER
...
app.js
TEST-APP
├── bin
├── public
├── routes
├── views
├── app.js
└── package.json

app.js
1  var createError = require('http-errors');
2  var express = require('express');
3  var path = require('path');
4  var cookieParser = require('cookie-parser');
5  var logger = require('morgan');
6
7  var indexRouter = require('./routes/index');
8  var usersRouter = require('./routes/users');
9
10 var app = express();
11
12 // view engine setup
13 app.set('views', path.join(__dirname, 'views'));
14 app.set('view engine', 'jade');
15
16 app.use(logger('dev'));
17 app.use(express.json());
18 app.use(express.urlencoded({ extended: false }));
19 app.use(cookieParser());
20 app.use(express.static(path.join(__dirname, 'public')));
21
22 app.use('/', indexRouter);
23 app.use('/users', usersRouter);
24
25 // catch 404 and forward to error handler
26 app.use(function(req, res, next) {
27   next(createError(404));
28 });
29
```

**Ya sabemos crear un servidor con
Express, es momento de profundizar un
poco más.**

Archivos Estáticos

La clase pasada hablamos sobre los **servidores estáticos** y **dinámicos**.

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

Pero antes un paso súper necesario, **nodemon**...

Nodemon

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 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 sigamos con los archivos estáticos.

Carpeta public

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

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



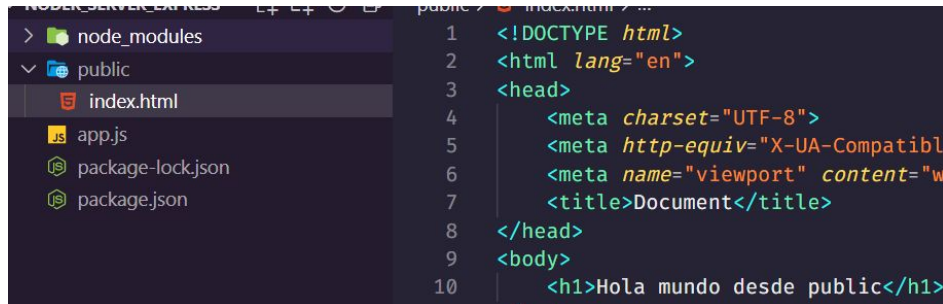
Luego en nuestro archivo **app.js** agregaremos la ruta a esta carpeta, indicando a Express de que se trata.

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

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

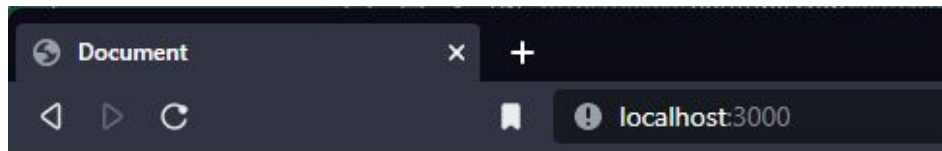
Por el momento debemos saber que nos permite interceptar lo que se ejecute dentro antes que la derivación de nuestras rutas.

Archivos estáticos



```
node_modules
public
  index.html
  app.js
  package-lock.json
  package.json

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
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>
```



Hola mundo desde public

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

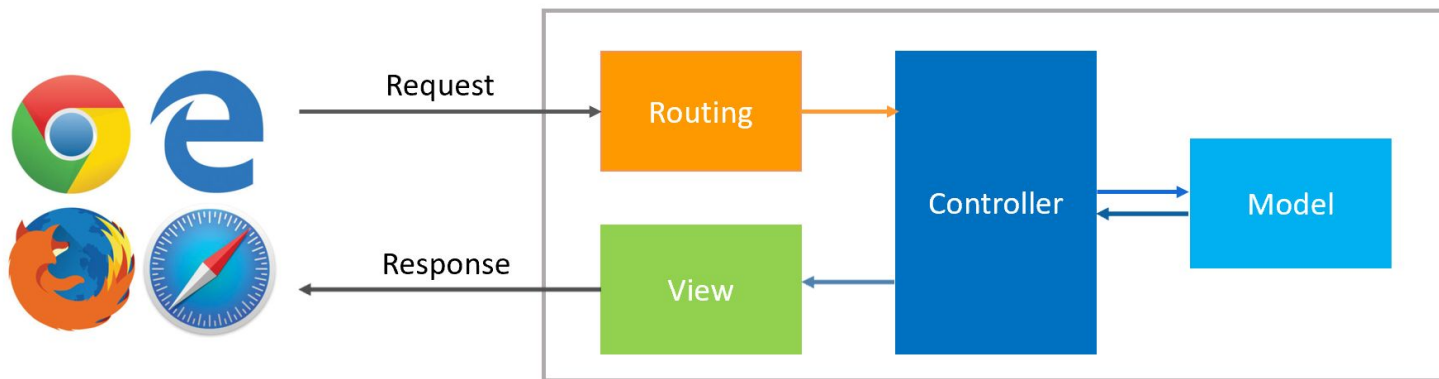
¡Magia! Nos devuelve nuestro archivo **HTML**.

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

Hablando de rutas...


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:



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

```
app.get('/nosotros', (req, res) => {  
  res.send(__dirname + './nosotros.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.

Pero dejemos eso para la próxima clase.

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