

Desarrollo Backend

Bienvenid@s

Servidor Web con Express I
(*NPM - el framework Express*)

Clase 11

UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO



Pon a grabar la clase



Temario

- **Administración de paquetes y dependencias**
 - Qué es administrar paquetes/dependencias
- **NPM**
 - Gestión de dependencias
 - Comandos más utilizados
 - La carpeta node_modules
- **El framework Express**
 - Qué es un framework
 - Ventajas de Express sobre el módulo HTTP
 - Instalación y creación de nuestro servidor web
 - Servir archivos/sitios web con FileSystem
- El archivo .env
- El archivo .gitignore



Administración de paquetes y dependencias

Administración de paquetes y dependencias

Esto se refiere al proceso de **gestionar e instalar paquetes de software** y sus dependencias en una aplicación Node.js.

En éste, los paquetes de software se distribuyen a través de **npm** (*Node Package Manager*), que es una herramienta de línea de comandos que permite a los desarrolladores descargar, instalar y actualizar paquetes de software desde un repositorio centralizado de paquetes.



Administración de paquetes y dependencias

La administración de paquetes en Node.js implica la creación y mantenimiento de un archivo llamado **package.json**, que describe la aplicación y sus dependencias.

Este archivo incluye información sobre los paquetes necesarios para que la aplicación funcione correctamente, así como versiones específicas de cada paquete.



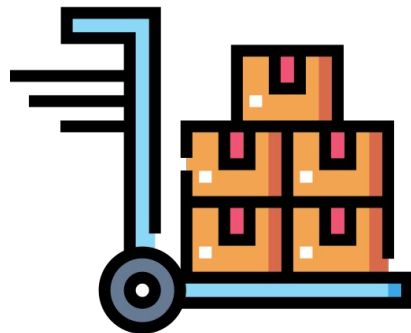
UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Administración de paquetes y dependencias

Esta administración de paquetes y dependencias, permitirá que sumemos a nuestros proyectos Node.js, distintas librerías, frameworks, y otros recursos adicionales, que harán que el desarrollo de una aplicación de backend, sea mucho más fácil.

El término “*dependencias*” viene justamente de las librerías y/o framework de los cuales Node.js depende para agregar nuevas características a la tarea de programar.

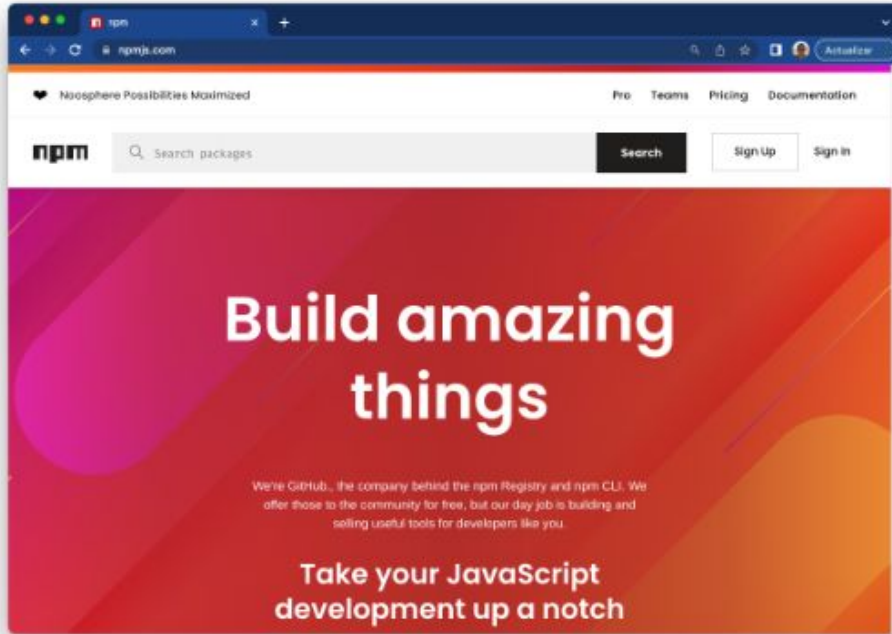


UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

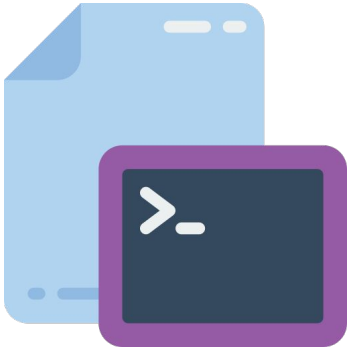
NPM

NPM



En nuestras primeras clases, hablamos de NPM, un gestor de paquetes y dependencias que viene incluido dentro de Node. Desde hoy, comenzaremos a utilizarlo de forma frecuente.

NPM



NPM se rige por la línea de comandos para administrar los paquetes y dependencias en Node.js. Si bien existen otros tantos gestores como NPM, este es el más utilizado en la comunidad de Node.js, y tuvo estos últimos años, importantes mejoras de performance y manejo de errores.

NPM nos permite descargar, instalar y actualizar paquetes de software desde su repositorio centralizado: <https://www.npmjs.com/>

Gestión de dependencias

NPM gestiona toda la actividad en aplicaciones Node.js de la mano de diferentes comandos. Estos se utilizan a través de la ventana Terminal del S.O. o la ventana integrada a VS Code.

Comando	Descripción
<code>npm init</code>	Crea un archivo "package.json" para describir la aplicación y sus dependencias. Es el comando utilizado para inicializar una aplicación Node.js.
<code>npm install</code>	Descarga e instala las dependencias necesarias en la carpeta "node_modules".
<code>npm update</code>	Actualiza los paquetes de software a las últimas versiones disponibles.
<code>npm uninstall</code>	Desinstala un paquete de software específico y lo elimina de "package.json".
<code>npm run</code>	Ejecuta un script personalizado especificado en "package.json".
<code>npm search</code>	Busca paquetes de software disponibles en el repositorio de NPM.
<code>npm publish</code>	Publica un paquete de software creado por el desarrollador en el repositorio de NPM para que otros puedan descargarlo e instalarlo.

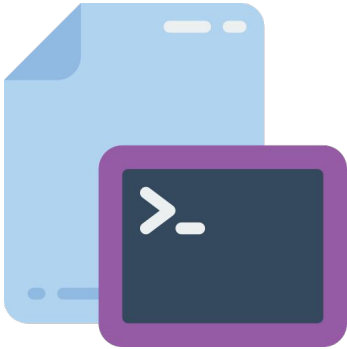
Gestión de dependencias

Estos son solo algunos de los comandos que dispone NPM y que pueden denominarse como los más utilizados.

Veamos a continuación algunos aspectos técnicos de NPM, package.json y la carpeta node_modules.

NPM

En resumen, NPM cuenta con los siguientes componentes:



Repositorio: mantiene un repositorio web y descarga desde allí los paquetes de software que utilizaremos.

Package.json: utiliza este archivo para describir las características técnicas de la aplicación, y de sus dependencias.

Línea de comandos: utiliza la Terminal para ejecutar comandos que permiten instalar, actualizar y desinstalar paquetes de software.

La carpeta node_modules

La carpeta node_modules

Cada dependencia que instalamos con NPM, descarga una serie de archivos y carpetas a nuestro proyecto. Estas se almacenan en la subcarpeta **node_modules**.

Cuando instalemos algo, veremos que dicha carpeta se crea automáticamente, al utilizar el comando NPM.



La carpeta node_modules

Su contenido crece significativamente, y habitualmente no necesitamos buscar nada de allí.

Con referenciar la librería o framework desde nuestras aplicaciones JS, Node.js sabrá cómo y dónde ubicar a éste, dentro de la carpeta node_modules.



La carpeta node_modules

Debemos tener la precaución, cuando subimos nuestros proyectos a un repositorio, de no subir también la carpeta **node_modules**. Esta crece mucho y ocupa demasiado espacio de almacenamiento.

Si descargamos el proyecto Node.js en otra computadora, gracias a NPM y el archivo package.json, podremos reinstalar rápidamente sus dependencias, generando nuevamente la carpeta node_modules.

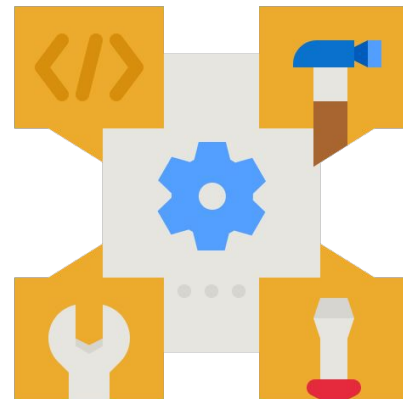


El framework Express

Qué es un framework

Un **framework**, tanto en JS como en otros lenguajes de programación, es un **conjunto de herramientas, librerías y estructuras de código predefinidas que se utilizan para desarrollar aplicaciones** web o móviles utilizando un lenguaje de programación; en este caso: JavaScript.

Los frameworks **proporcionan una base sólida para el desarrollo de aplicaciones al ofrecer una arquitectura predefinida y una serie de herramientas y componentes para construir rápidamente aplicaciones web.**



Qué es un framework

En particular, para el desarrollo de aplicaciones en el backend utilizando

Node.js, existen varios frameworks populares como ser:

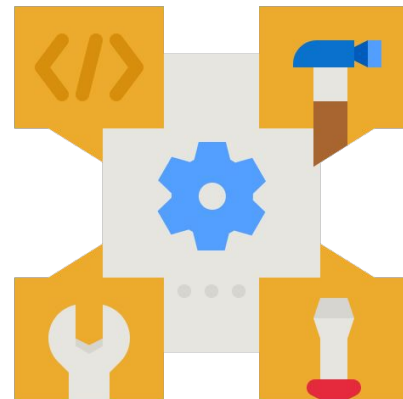
- Express.js
- Nest.js
- Koa.js
- Hapi.js

entre otros.

Proporcionan una amplia gama de funcionalidades como el manejo de rutas, la

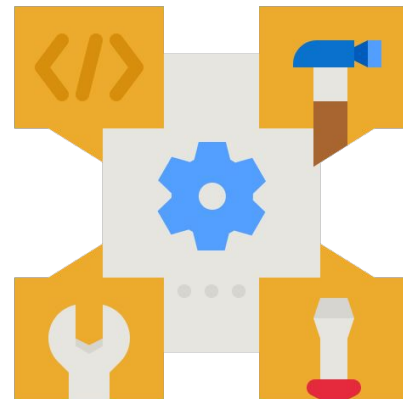
gestión de peticiones y respuestas HTTP, la integración con bases de datos y

middleware para la validación de datos, autenticación, y seguridad.



Qué es un framework

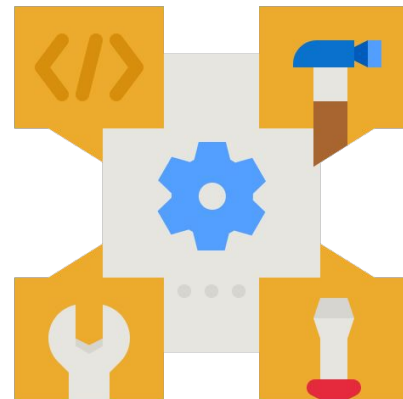
En general, **los frameworks JS para Node.js simplifican el proceso de desarrollo de aplicaciones en el backend**, permitiendo a los desarrolladores enfocarse en la lógica de negocio en lugar de preocuparse por la implementación de la infraestructura técnica básica.



Qué es un framework

Express.js es uno de los frameworks más populares para el **desarrollo de aplicaciones web en Node.js**, proporcionando una estructura flexible y minimalista para construir aplicaciones web y APIs.

Será el elegido por nosotr@s para poder construir aplicaciones de backend basadas en servidores web, de una forma mucho más cómoda que la que propone el **módulo HTTP**.



Ventajas de Express sobre el módulo HTTP

Ventajas de Express sobre el módulo HTTP

Existen varias ventajas al utilizar Express.js en lugar del módulo HTTP nativo de Node.js para el desarrollo de aplicaciones web:

Ventajas	Descripción
Estructura de código	Proporciona una estructura de código clara y organizada para el desarrollo de aplicaciones web, con funciones y métodos predefinidos que facilitan la creación de rutas, middlewares y controladores.
Middleware	Tiene un sistema de middleware flexible que permite agregar funcionalidades adicionales a la aplicación web, como autenticación, manejo de errores, compresión de archivos, entre otros.
Escalabilidad	Es altamente escalable, por lo cual puede manejar gran cantidad de solicitudes sin afectar el rendimiento de la aplicación. Además, su estructura modular permite agregar nuevas funcionalidades extendiendo la aplicación según sea necesario.
Comunidad y documentación	Es uno de los frameworks JS más populares y utilizados en Node.js. Cuenta con una gran comunidad de desarrolladores y documentación exhaustiva y clara, que facilita el aprendizaje y la implementación de la tecnología.
Integración	Se integra fácilmente con otros módulos y librerías de Node.js, permitiendo agregar funcionalidades adicionales a la aplicación web, como bases de datos, servicios de almacenamiento en la nube, servicios de autenticación, entre otros.

Ventajas de Express sobre el módulo HTTP

En resumen, Express.js es una excelente opción para el desarrollo de aplicaciones web en Node.js, ya que proporciona una estructura de código organizada, un sistema de middleware flexible, escalabilidad y una gran comunidad y documentación.

Todo esto hace que el desarrollo de aplicaciones web sea más rápido, fácil y eficiente.

Instalación de Express JS

Instalación de Express JS

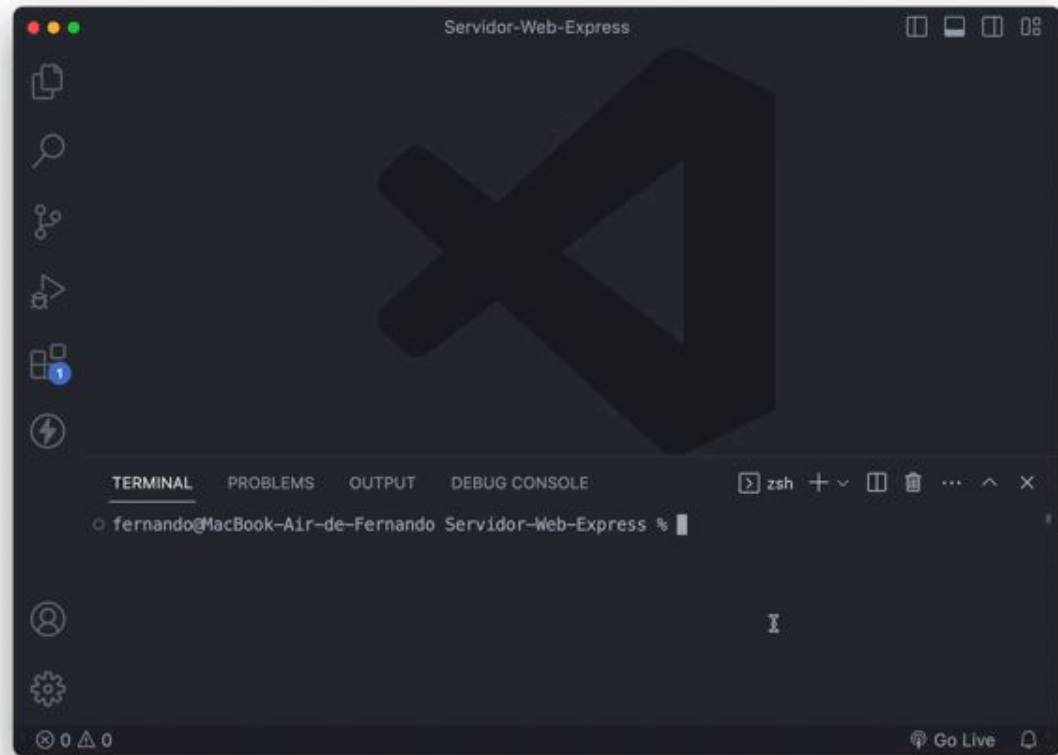
Crearemos una nueva aplicación web backend. Definamos para ello, una carpeta en nuestra computadora, la cual llamaremos 📁

Servidor-Web-Express.

Recuerda crearla junto al resto de los proyectos, en un lugar donde puedas crear una copia de seguridad a tu cuenta de Github.

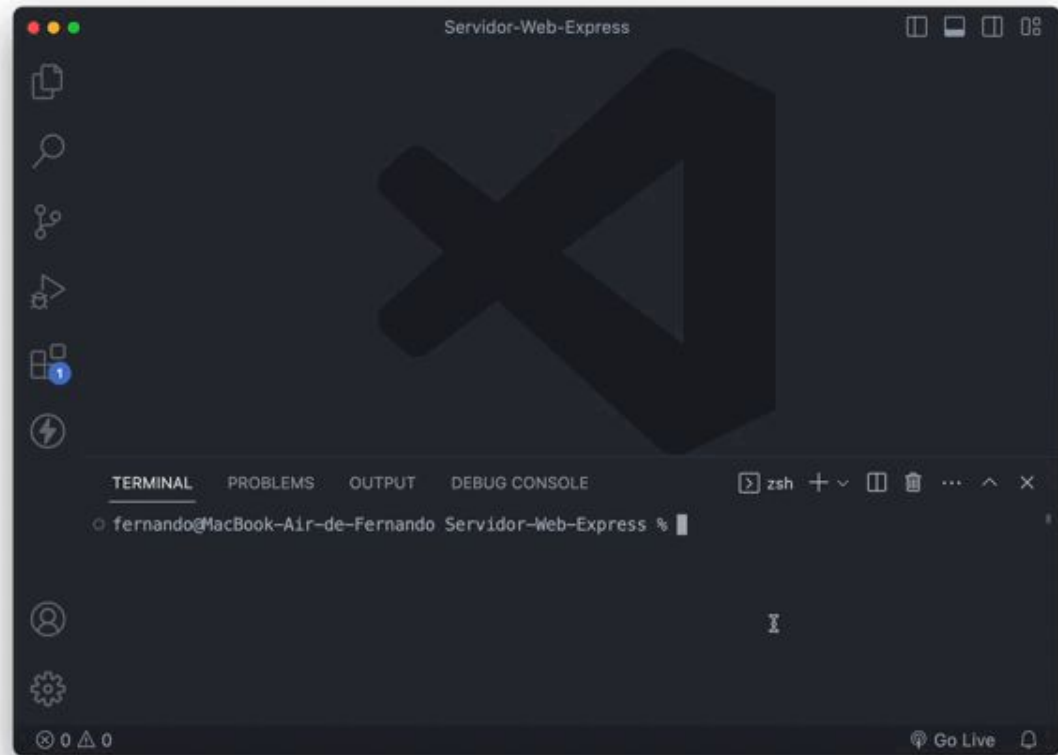


Instalación de Express JS



Abrimos la carpeta con Visual Studio Code. Luego, inicializamos el proyecto Node.js utilizando la **Terminal** y definiendo el comando **`npm init -y`**. Se creará el archivo **`package.json`** en el directorio raíz de este proyecto.

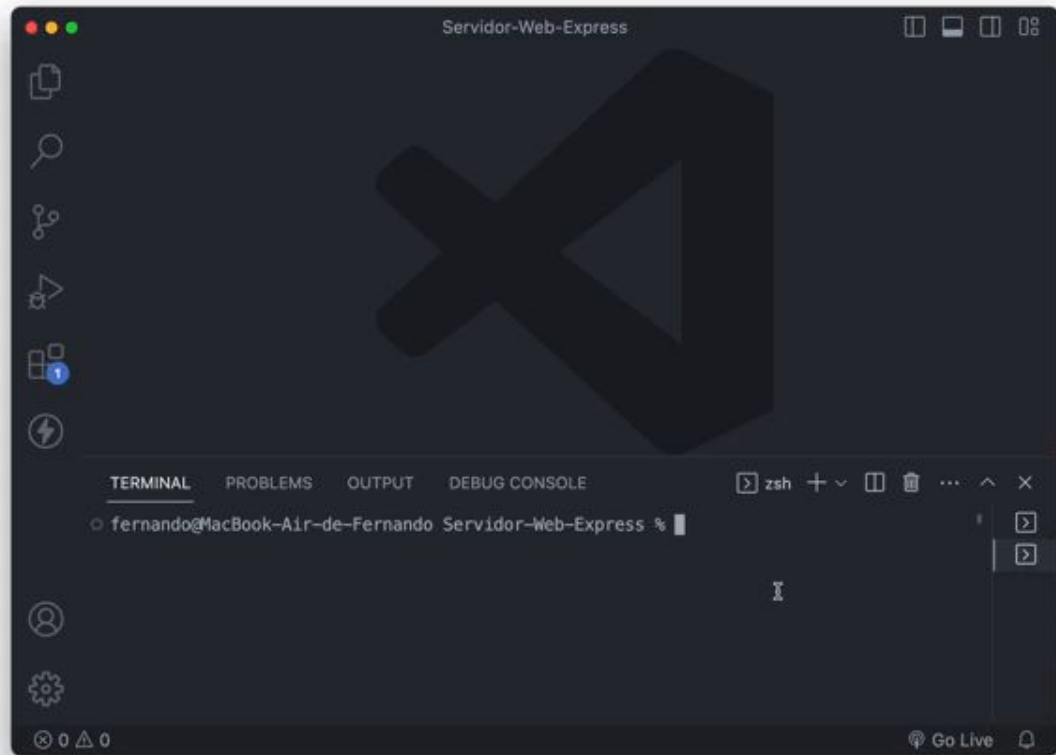
Instalación de Express JS



Generado el archivo **package.json**, lo editamos y completamos algunos datos extra, así nos queda el proyecto con información real.

Puedes obviar el cambio en el nombre de archivo JavaScript, dejando **index.js** por defecto.

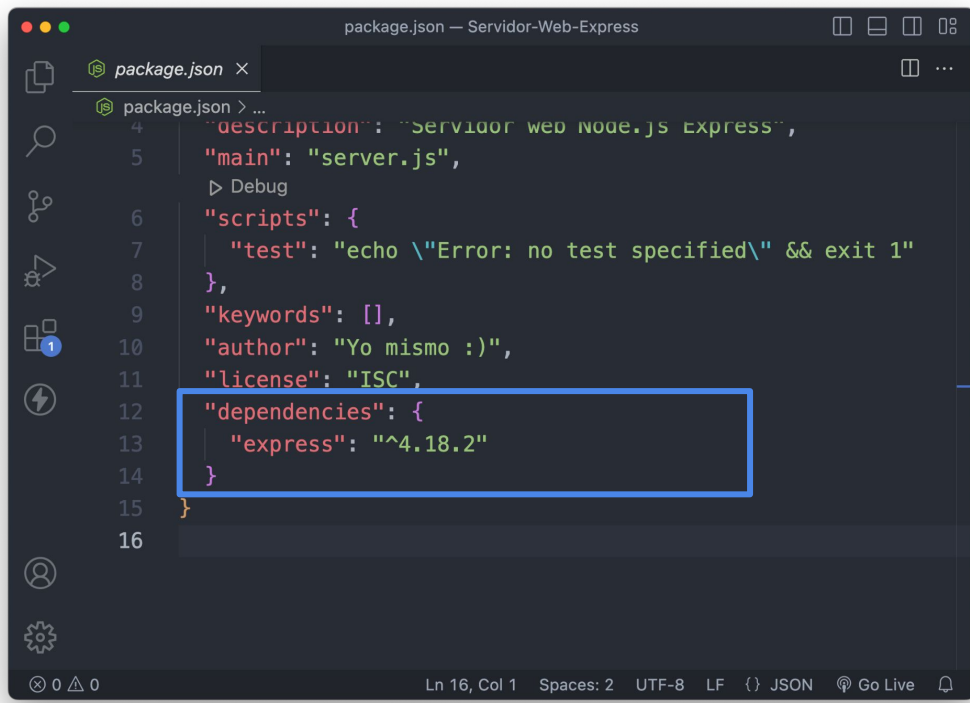
Instalación de Express JS



Luego, necesitas instalar Express JS usando el comando `npm install express`.

Esto descargará e instalará Express JS y todas sus dependencias en el directorio `node_modules` de tu proyecto.

Instalación de Express JS



```
package.json — Servidor-Web-Express
package.json > ...
 4  "description": "Servidor web Node.js Express",
 5  "main": "server.js",
 6  "scripts": {
 7    "test": "echo \\\"Error: no test specified\\\" && exit 1"
 8  },
 9  "keywords": [],
10  "author": "Yo mismo :)",
11  "license": "ISC",
12  "dependencies": {
13    "express": "^4.18.2"
14  }
15 }
16
```

Toda futura dependencia que agreguemos en este proyecto, se reflejará en el apartado **dependencies** del archivo **package.json**, separando una de las otras, por una coma.

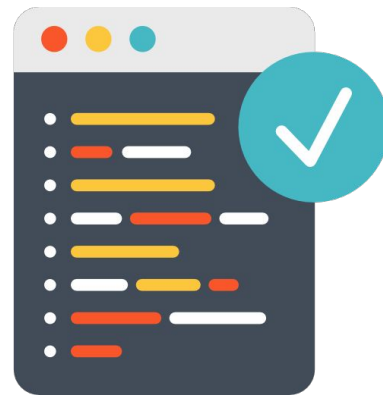
Veremos el nombre de la dependencia y qué versión tenemos instalada.

Creación de un servidor web con Express JS

Creación de un servidor web con Express JS

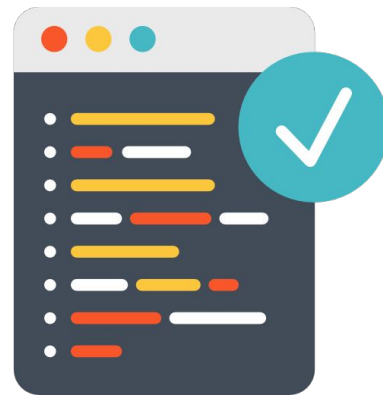
Ya instalado Express JS, podemos crear un servidor en nuestra aplicación de backend. Para ello, creamos en el raíz del proyecto el archivo JavaScript correspondiente.

Lo editamos, y lo primero que haremos será instanciar la librería Express para utilizarla en nuestro proyecto.



Creación de un servidor web con Express JS

Express es una Clase JS basada en objetos. Como está pensada para trabajar de forma modular en Node.js, debemos referenciar la misma en nuestra aplicación Node utilizando la función **require()**. En el contexto de la creación de JS backend, **require()** se utiliza para incluir módulos en nuestra aplicación.



UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Creación de un servidor web con Express JS

Node.js carga el módulo en nuestro archivo

JavaScript, resolviendo automáticamente la ruta original de éste.

En el snippet contiguo, `require('express')` carga el módulo **express** y lo asigna a la constante homónima. Luego, instanciamos el framework dentro de la constante **app**.



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

A partir de aquí, ya podemos utilizar sus propiedades y métodos.

Creación de un servidor web con Express JS

El paso siguiente, utilizando desde aquí la constante **app**, podemos definir la ruta principal de nuestro servidor web Express. La misma, como en todo sitio web, está referenciada por la ruta `'/'` raíz.

El método **get()** se ocupa de estar atento a las peticiones entrantes, recibiendo dos parámetros: el primero, la ruta peticionada, el segundo, la función de respuesta a la petición, con **request** y **response** como parámetros.



```
// Define una ruta básica
app.get('/', (req, res) => {
  res.send('¡Hola, mundo!');
});
```

Creación de un servidor web con Express JS

Finalmente, el método **send()**, asociado a la respuesta (**response**), se utiliza para enviar la respuesta a dicha petición.

Como vemos, hasta aquí todo es bastante similar a lo que aprendimos trabajando con el módulo HTTP.

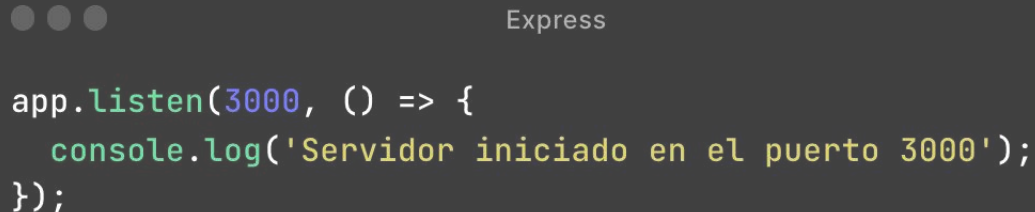


```
// Define una ruta básica
app.get('/', (req, res) => {
  res.send('¡Hola, mundo!');
});
```

Creación de un servidor web con Express JS

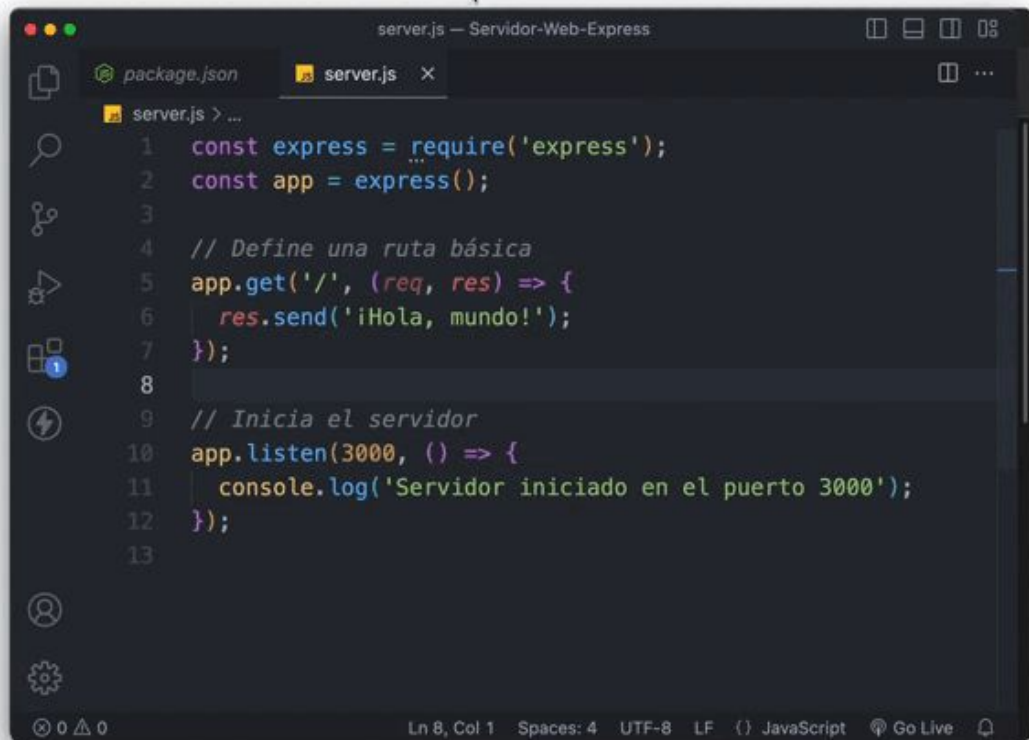
Nos queda definir el puerto de escucha para nuestro servidor. El mecanismo también es similar al utilizado con el módulo HTTP.

Con todo esto, ya podemos ejecutar nuestro servidor web Express.

A dark-themed terminal window with the title 'Express'. It contains three window control buttons (red, yellow, green) in the top-left corner. The code inside is:

```
app.listen(3000, () => {  
  console.log('Servidor iniciado en el puerto 3000');  
});
```

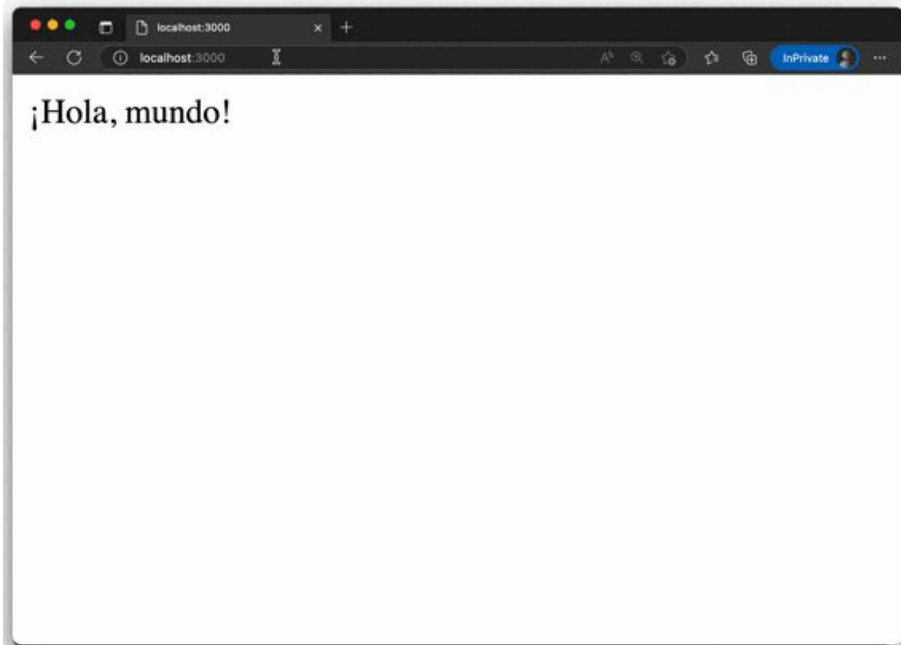
Creación de un servidor web con Express JS



```
server.js — Servidor-Web-Express  
package.json server.js  
server.js > ...  
1  const express = require('express');  
2  const app = express();  
3  
4  // Define una ruta básica  
5  app.get('/', (req, res) => {  
6    res.send('¡Hola, mundo!');  
7  });  
8  
9  // Inicia el servidor  
10 app.listen(3000, () => {  
11   console.log('Servidor iniciado en el puerto 3000');  
12 });  
13
```

Ya podemos abrir la **Terminal**, y ejecutar el comando **node server.js**, para poner a funcionar nuestro servidor web Express. Luego lo probamos desde un navegador web.

Creación de un servidor web con Express JS



Debemos tener presente que Express JS, a diferencia del módulo HTTP, responde solo a las rutas creadas. Si petitionamos una ruta inexistente o no controlada por Express, este devolverá un error no controlado.

Debemos definir cada una de las rutas que utilizaremos, y un mensaje de error personalizado para todas aquellas rutas inexistentes en este servidor. 🙄

UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

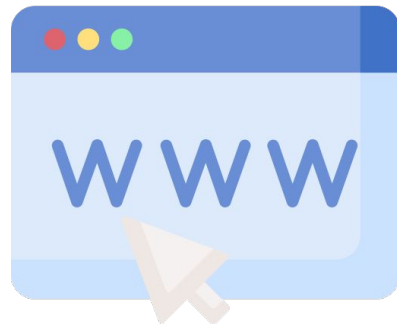
Servir un sitio web con Express JS

Servir un sitio web con Express JS

Express JS tiene un montón de ventajas respecto al módulo HTTP.

Una de ellas es cuando debemos servir un sitio web frontend.

Cuando realizamos esta tarea con el módulo HTTP, vimos que debíamos vincular el módulo **filepath**, **fileSystem API**, la variable global **__dirname** y a su vez controlar el tipo de recurso asociado al documento HTML para servirlo correctamente (*css, js, imágenes, etcétera*)



UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Servir un sitio web con Express JS

Efectivamente, el módulo HTTP requiere mucho trabajo para servir un sitio web sobre todo, si este tiene diferentes tipos de archivos como recursos. Los **MIME/Types** serán interminables de controlar.



Creación de un servidor de sitio web

```
let filePath = path.join(__dirname, 'public', 'controller-hogar');
let extname = path.extname(filePath);
let contentType = 'text/html'; req.url;

switch (extname) {
  case '.js':
    contentType = 'text/javascript';
    break;
  case '.css':
    contentType = 'text/css';
    break;
  case '.json':
    contentType = 'application/json';
    break;
  case '.png':
    contentType = 'image/png';
    break;
  case '.jpg':
    contentType = 'image/jpg';
    break;
  case '.wav':
    contentType = 'audio/wav';
    break;
}
```

Dentro del método `.join()` agregamos una regla del tipo **Operador Ternario**, donde evaluamos el nombre del archivo o recurso a enviar.

Luego, sumamos una estructura **switch** la cual evalúa como parámetro al archivo definido en la variable `filePath`.

UNTREF
UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Servir un sitio web con Express JS

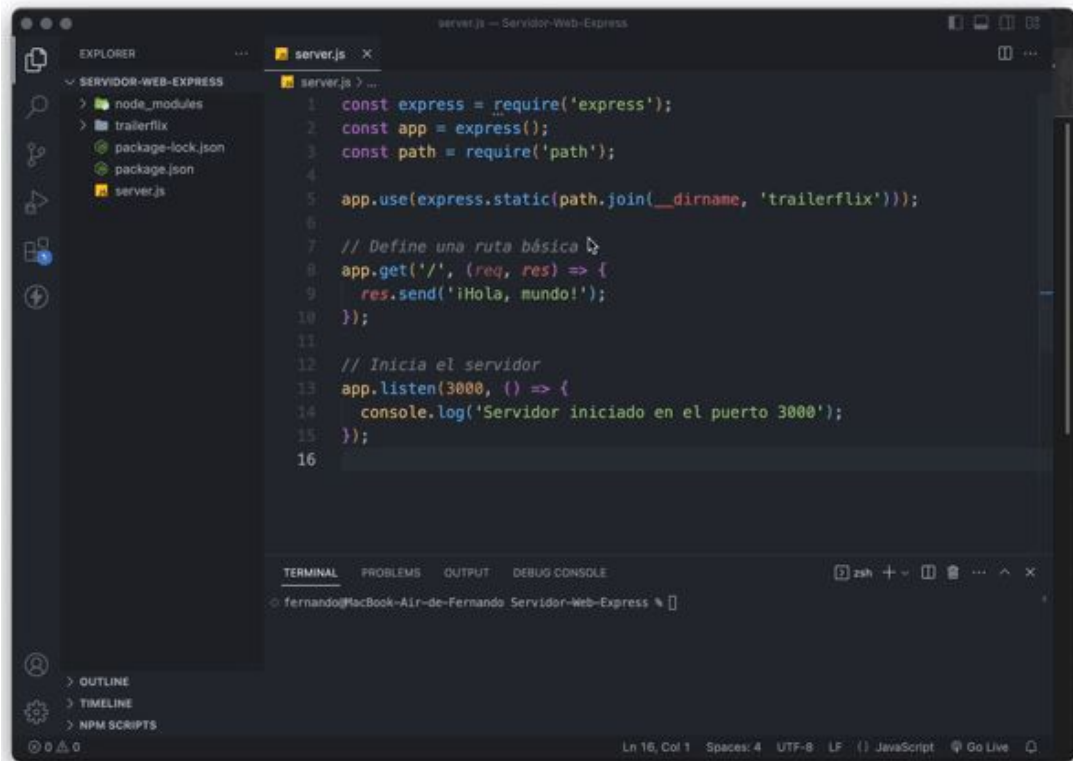
Para resolver esta tarea, debemos referenciar al módulo **path**, luego usamos el método **static()** de Express JS. Por último, invocamos al método **path.join()** concatenando **__dirname** junto a la subcarpeta que contiene el sitio web.

```
Express

const path = require('path');

app.use(express.static(path.join(__dirname, 'trailerflix')));
```

Servir un sitio web con Express JS



The screenshot shows a VS Code editor window with a file named `server.js` open. The file contains the following JavaScript code:

```
1  const express = require('express');
2  const app = express();
3  const path = require('path');
4
5  app.use(express.static(path.join(__dirname, 'trailerflix')));
6
7  // Define una ruta básica
8  app.get('/', (req, res) => {
9    res.send('¡Hola, mundo!');
10 });
11
12 // Inicia el servidor
13 app.listen(3000, () => {
14   console.log('Servidor iniciado en el puerto 3000');
15 });
16
```

The left sidebar shows the Explorer view with the project structure:

- SERVER-WEB-EXPRESS
 - node_modules
 - trailerflix
 - package-lock.json
 - package.json
 - server.js

The bottom panel shows the TERMINAL view with the command prompt:

```
fernando@MacBook-Air-de-Fernando: Servidor-Web-Express %
```

Efectivamente, Express JS, es el Framework para crear servidores web que nos facilita la vida y acorta notoriamente los tiempos de desarrollo web.

👉 Aquí, un claro ejemplo de ello.

Resumen de Express JS

Express JS es una herramienta más potente y completa para construir aplicaciones web y APIs que el módulo HTTP.

Si bien el módulo HTTP es útil para situaciones simples, como servir archivos estáticos, Express JS proporciona una funcionalidad más avanzada y un conjunto de herramientas más completo para manejar aplicaciones web y APIs de manera más sofisticada.

Variables de entorno y el archivo **.env**

Variables de entorno y el archivo .env

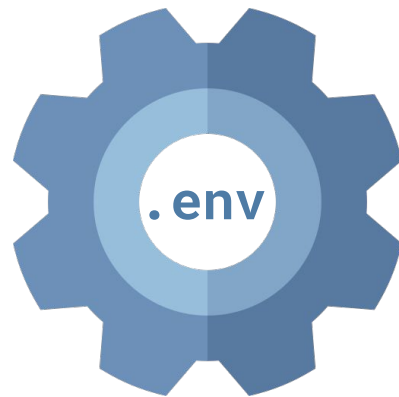
En las aplicaciones backend existen muchas variables denominadas variables de entorno. Éstas se definen en el sistema operativo y se utilizan para configurar diferentes aspectos del entorno de desarrollo, como por ejemplo la **URL de la base de datos**, la **clave secreta para autenticación de usuarios**, la **dirección IP del servidor**, etc.



Variables de entorno y el archivo .env

Todo lo referente a direcciones de sitios web, claves secretas, puertos de servidor, rutas de aplicaciones frontend, y demás cuestiones, son manejadas en diferentes rutas, de acuerdo al Rol de cada persona técnica en un equipo de IT.

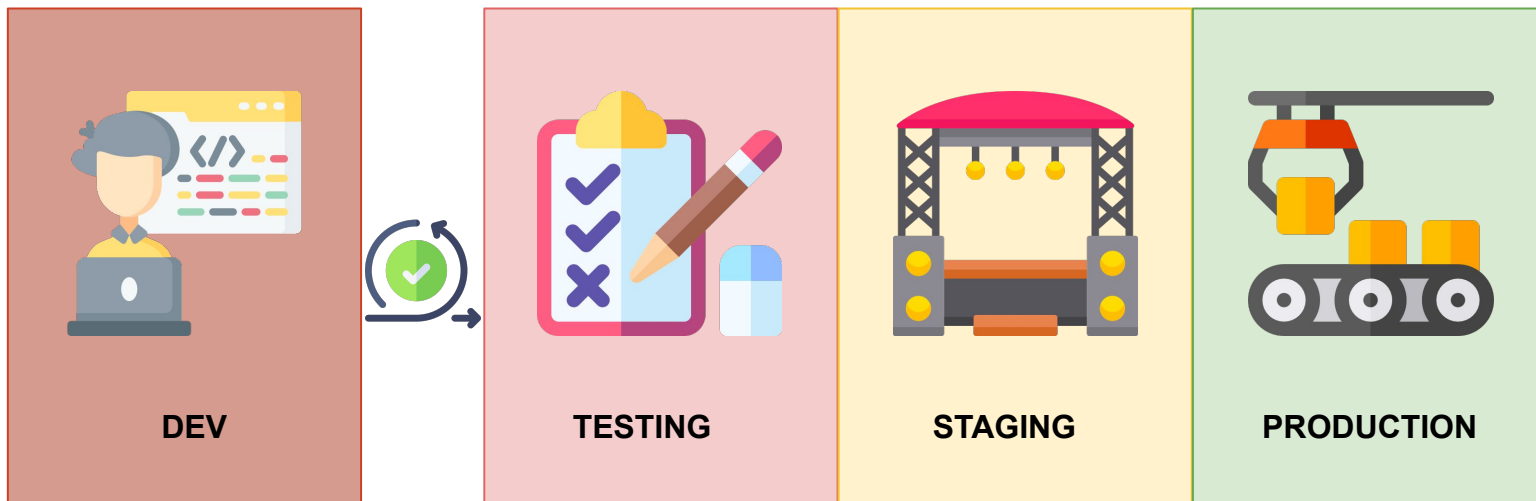
Veamos algunos ejemplos, a continuación:



UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

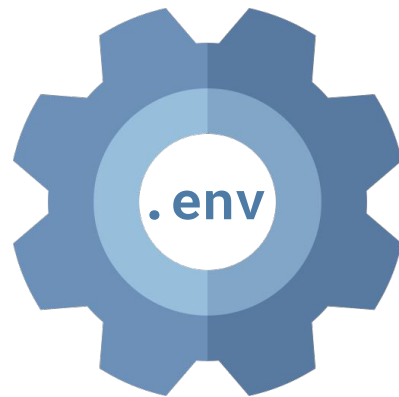
Variables de entorno y el archivo .env



***Los diferentes entornos de trabajo con aplicaciones software que posee
una empresa mediana / grande.***

Variables de entorno y el archivo .env

Por ello, en todo proyecto web, debemos contemplar el uso de variables de entorno, e incorporar esta práctica en el desarrollo de nuestras aplicaciones de software. Y, en Node.js, el archivo **.env** es quien cumple perfectamente el rol de poder definir variables de este tipo.



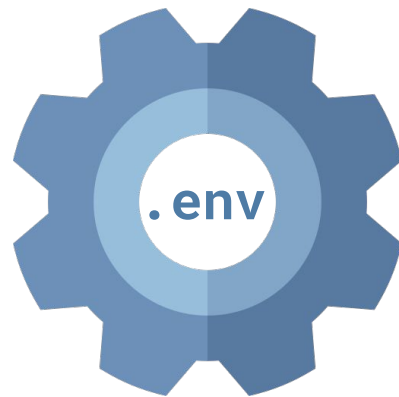
.env es un archivo de configuración que se utiliza para almacenar estas variables de entorno.

Variables de entorno y el archivo .env

Este archivo debemos definirlo en la raíz de nuestros proyectos Node.js.

Su nombre es literalmente '**.env**'.

La notación punto (.) antes de la palabra **env**, lo convierte en un archivo invisible. Podremos verlo a través de un editor de código como VS Code, pero no será visible en una carpeta o subcarpeta de servidor, sobre todo si el mismo es un servidor **Linux** o derivado de sistemas **Unix**.



UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

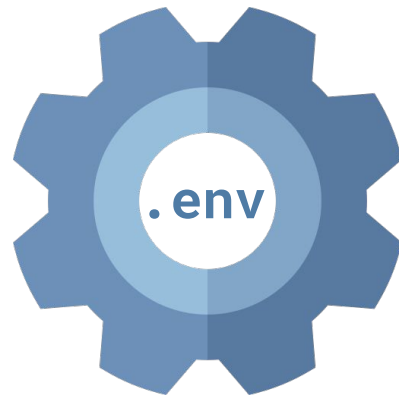
Variables de entorno y el archivo .env

Este archivo debemos definirlo en la raíz de nuestros proyectos Node.js.

Su nombre es, literalmente, '**.env**'.

Su estructura es, básicamente, un archivo de texto plano que se encuentra en la raíz de tu proyecto **Node.js** y contiene las variables de entorno que necesita tu aplicación.

Cada variable de entorno se define como una línea en el archivo en el formato **NOMBRE=VALOR**, donde **NOMBRE** hace referencia a la variable en sí y, **VALOR**, al valor almacenado en ésta.



UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Variables de entorno y el archivo .env

Este archivo debemos definirlo en la raíz de nuestros proyectos Node.js.

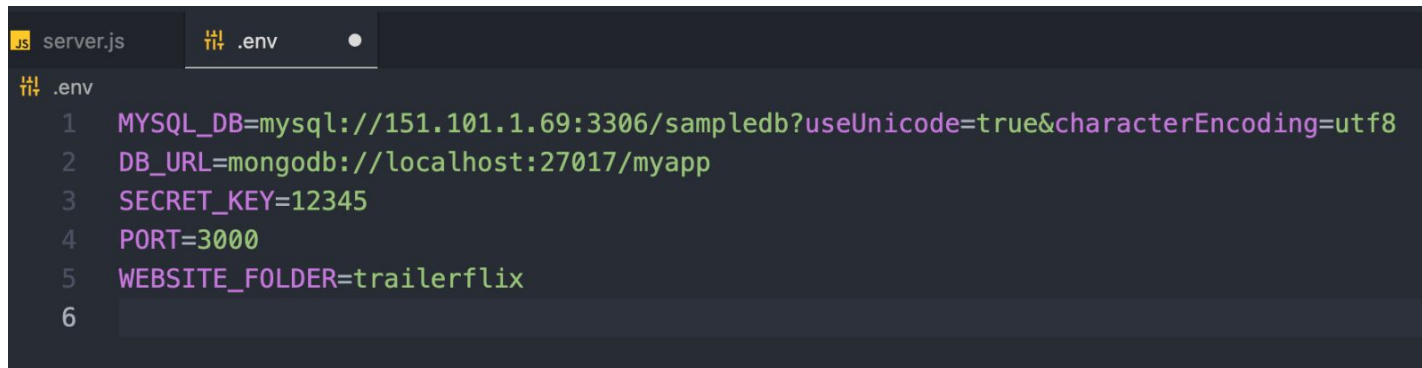
Su nombre es, literalmente, '**.env**'.

Su estructura es, básicamente, un archivo de texto plano que se encuentra en la raíz de tu proyecto **Node.js** y contiene las variables de entorno que necesita tu aplicación.

Cada variable de entorno se define como una línea en el archivo en el formato **NOMBRE=VALOR**, donde **NOMBRE** hace referencia a la variable en sí y, **VALOR**, al valor almacenado en ésta.



Variables de entorno y el archivo .env

A screenshot of a code editor with a dark theme. At the top, there are two tabs: 'server.js' with a JavaScript icon and '.env' with a file icon. The '.env' tab is active, showing a list of environment variables. The variables are: 1. MYSQL_DB=mysql://151.101.1.69:3306/sampled?useUnicode=true&characterEncoding=utf8, 2. DB_URL=mongodb://localhost:27017/myapp, 3. SECRET_KEY=12345, 4. PORT=3000, 5. WEBSITE_FOLDER=trailerflix, and 6. An empty line. The lines are numbered 1 through 6 on the left side of the editor.


```
.env
1  MYSQL_DB=mysql://151.101.1.69:3306/sampled?useUnicode=true&characterEncoding=utf8
2  DB_URL=mongodb://localhost:27017/myapp
3  SECRET_KEY=12345
4  PORT=3000
5  WEBSITE_FOLDER=trailerflix
6
```

Aquí tenemos un ejemplo de cómo referir cada una de las variables creadas dentro del archivo `.env`, y los valores asociados a éstas. Así, variables claves como **PORT**, **URL** de bases de datos, y **subcarpetas** con aplicaciones frontend, no estarán atadas a una URL estática dentro del código de nuestras aplicaciones backend.

Implementar las variables de entorno

Implementar las variables de entorno

Creado el archivo `.env` y declaradas las variables pertinentes, nos queda instalar la dependencia **dotenv** con **npm**, y luego referenciarla en nuestra aplicación. Finalmente, el método `.config()` agrega las variables de entorno definidas en este archivo, dentro del objeto JS **process.env**.



```
dotenv

//En la Terminal
\> npm install dotenv (enter)

//En el encabezado de nuestro archivo .JS
const dotenv = require('dotenv');
dotenv.config();
```

Implementar las variables de entorno

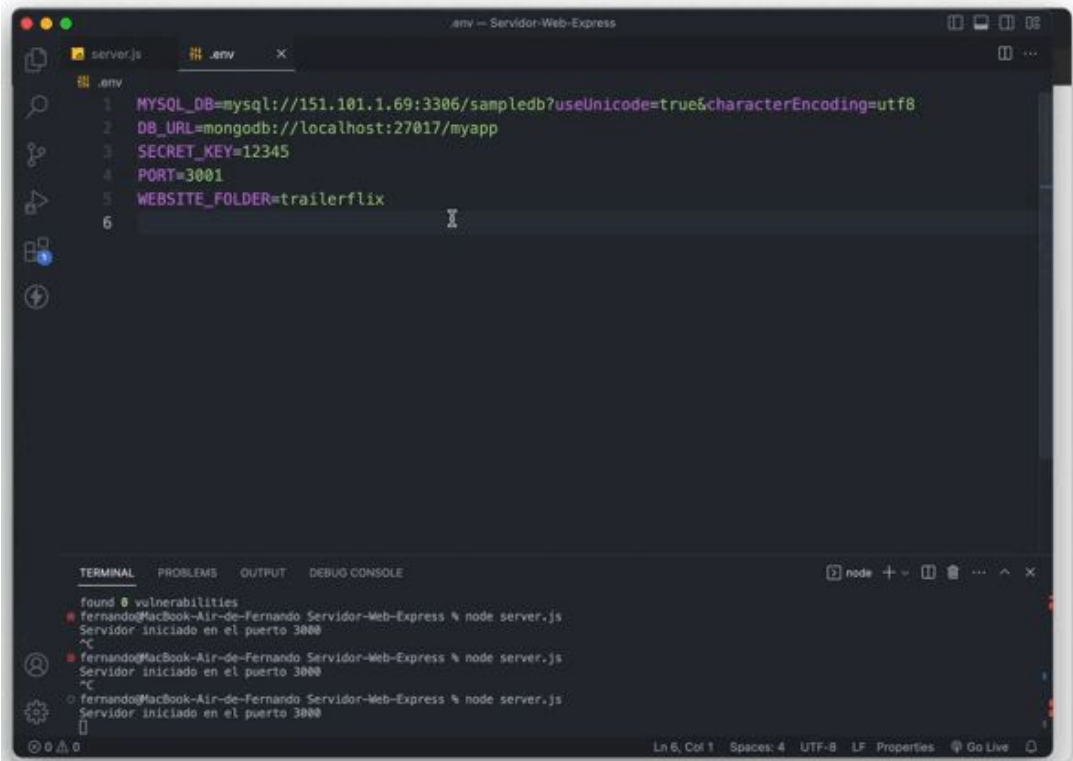
A través del objeto `process.env`, accederemos al valor definido en esta variable. Ahora, el servidor web, escuchará en el puerto definido **PORT** (*variable de entorno*). Si no existe dicha variable, entonces asumirá el puerto **3000** como predeterminado.

```
dotenv

// reemplazamos el nombre de la carpeta 'trailerflix'
// definido de manera estática.
app.use(express.static(path.join(__dirname, process.env.WEBSITE_FOLDER)));

//Corregimos la referencia al puerto de escucha del servidor web
app.listen(process.env.PORT || 3000, () => {
  console.log('Servidor iniciado en el puerto 3000');
});
```

Servir un sitio web con Express JS



The image shows a VS Code editor window with a file named `.env` open. The file contains the following environment variables:

```
1 MYSQL_DB=mysql://151.101.1.69:3306/sampledb?useUnicode=true&characterEncoding=utf8
2 DB_URL=mongodb://localhost:27017/myapp
3 SECRET_KEY=12345
4 PORT=3001
5 WEBSITE_FOLDER=trailerflix
6
```

Below the editor, a terminal window is open, showing the output of running `node server.js` three times. Each time, it reports "found 0 vulnerabilities" and "Servidor iniciado en el puerto 3000".

Con las variables de entorno **PORT** y **WEBSITE_FOLDER**, ya evitamos tener que forzar a la aplicación web a trabajar en un puerto específico o ubicar al sitio web en una subcarpeta determinada.

Cada entorno involucrado en las etapas de desarrollo y pruebas, solo deberá cambiar los valores de las variables de entorno, por las rutas específicas de su entorno de trabajo.

Variables de entorno y el archivo .env

La notación punto (.) antes de la palabra env, lo convierte en un archivo invisible.

Podremos verlo a través de un editor de código como VS Code, pero no será visible en una carpeta o subcarpeta de servidor, sobre todo si el mismo es un servidor Linux o derivado de sistemas Unix.

El archivo `.gitignore`

El archivo .gitignore

Cuando trabajamos en un proyecto de software, es común que se generen archivos que no son necesarios para el funcionamiento de la aplicación y/o que no deban incluirse en el control de versiones.

Algunos ejemplos son:

- archivos temporales
- archivos generados por el sistema
- archivos de log
- archivos de configuración con información sensible

entre otros...



El archivo **.gitignore**

El archivo **.gitignore** permite especificar qué archivos y directorios se deben ignorar en el control de versiones, evitando así que se añadan accidentalmente al repositorio de Git.

Para hacer esto, simplemente debemoss incluir en el archivo **.gitignore** una lista de patrones que describan los archivos y directorios que se deben ignorar.



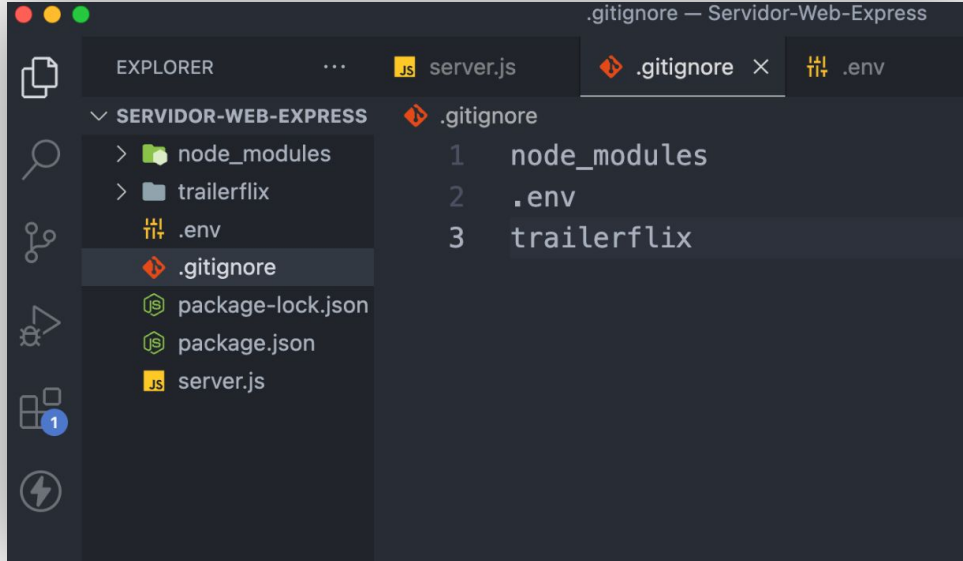
El archivo `.gitignore`

El archivo **`.gitignore`** permite especificar qué archivos y directorios se deben ignorar en el control de versiones, evitando así que se añadan accidentalmente al repositorio de Git.

Para hacer esto, simplemente debemos incluir en el archivo `.gitignore` una lista de patrones que describa los archivos y/o directorios que se deban ignorar.



El archivo .gitignore



Creado el archivo **.gitignore**, podemos agregar en nuestro proyecto, las referencias a la carpeta **trailerflix** o nuestro proyecto frontend, el archivo **.env** con la configuración más sensible del entorno y, por supuesto, la carpeta **node_modules**.

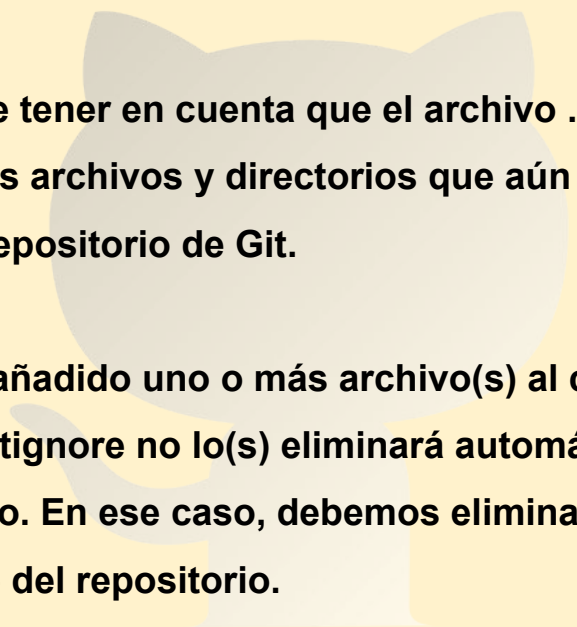
El archivo .gitignore

Con esta simple configuración, ya nos aseguramos de poder publicar en nuestro repositorio Git o Github, el proyecto que venimos trabajando, sin poner a disposición de terceros la información clave o personal de ciertos recursos asociados al proyecto.

También puedes referenciar archivos como **.TXT** o **.PDF**, **.DOCX**, o **.XLSX**, para evitar que documentación o apuntes del proyecto se guarden en el repositorio en cuestión.



El archivo .gitignore



Es importante tener en cuenta que el archivo .gitignore solo afecta los archivos y directorios que aún no han sido añadidos al repositorio de Git.

Si ya hemos añadido uno o más archivo(s) al control de versiones, .gitignore no lo(s) eliminará automáticamente del repositorio. En ese caso, debemos eliminarlo(s) manualmente del repositorio.

```
const questions = [ 'dudas', 'consultas', '🧐' ]
```



```
> node gracias.js
```

UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO