

Activación de saberes previos

- Conocimientos básicos en informática, lo que abarca uso del equipo de cómputo (prender, apagar, iniciar sesión, ejecutar programas, instalar y desinstalar programas, capturas de pantalla).
- Conocimientos básicos en inglés, puesto que algunas palabras, páginas web, herramientas o instaladores pueden estar en este idioma.
- Conocimientos en la navegación por internet, abrir páginas web, realizar una búsqueda, descargar un archivo o programa.
- Conocimientos en la distribución del teclado como la ubicación y nombre de símbolos, caracteres especiales, números y letras.
- Uso del correo electrónico, enviar y reenviar correos, adjuntar archivos.
- Uso básico de herramientas de reunión virtual como Zoom o Google Meet, conexión a una reunión, compartir pantalla, activar y desactivar audio y video.
- Herramientas de programación instaladas, VS Code y extensiones.
- Versionamiento básico, Git y Github
- Creación de sitios web con HTML, CSS y JS
- Manejo de funciones y eventos con JS

¿Qué voy a aprender?

- Qué es MVC, NodeJS y Express
- Estructura de un servidor de aplicativo web
- Rutas y vistas

¿Qué necesito?

- Equipo de Cómputo
- Conexión a Internet
- Completar el módulo 1

Lección 1: Qué es MVC, NodeJS y Express

Planteamiento de la sesión

Tiempo de ejecución: 2 horas



Materiales:

- <https://developer.mozilla.org/es/docs/Glossary/MVC>
- <https://n9.cl/awftl>
- <https://nodejs.org/en/download>
- [express - npm](#)
- [Instalación de Express Express 4.x - Referencia de API](#)
- [Ejemplo "Hello World" de Express](#)
- [Express JS HTTP Methods - GeeksforGeeks](#)

MVC

En programación, no siempre existe un único camino para resolver un problema. Generalmente hay varios puntos de vista y, dependiendo de nuestros enfoques, podemos llegar al mismo resultado a través de diversos caminos.

Si bien esto no está del todo mal, cuando nos vemos en la necesidad de trabajar en equipo y sobre un proyecto que escale quizás lo mejor sea estar alineado en conjunto con todo nuestro equipo de trabajo.

Es por esta razón que existen los ****patrones de diseño****. Estos resuelven los problemas más comunes con los que nos encontramos al momento de programar, ya que proponen un camino bastante definido para llegar a destino.

Justamente de eso trata ****MVC**** : un patrón de diseño que busca resolver de una manera estándar el problema habitual de ****comunicación entre el frontend y el backend**** de nuestra aplicación.

MVC hace referencia a las siglas: Modelo, Vista y Controlador. Y propone que cada uno de estos componentes resuelva y se encargue de una particularidad en específico, sin tomar atribuciones y responsabilidades de otros componentes.

Aunque el patrón o modelo solo incluya las siglas para MVC, existen varios puntos intermedios que hacen parte de lo que se va a construir, es decir que un proyecto bajo este patrón, contendrá más carpetas y archivos, adicionales a las correspondientes a los Modelos, Vistas y Controladores.

La Vista se encargará de la parte visual de la aplicación. El Modelo, de administrar la información de la base de datos. Y el Controlador, de ser un puente de comunicación entre los otros dos, en donde también estará un componente grueso de la lógica de la aplicación y el qué hacer con los datos que obtengo para almacenarlos o presentarlos.

Permite crear proyectos escalables y modulares, dividiendo los mismos en tres componentes lógicos:

- Modelos
- Vistas
- Controladores

Cada uno de ellos cumple un rol específico, actúa de manera independiente, y está aislado de los otros dos.

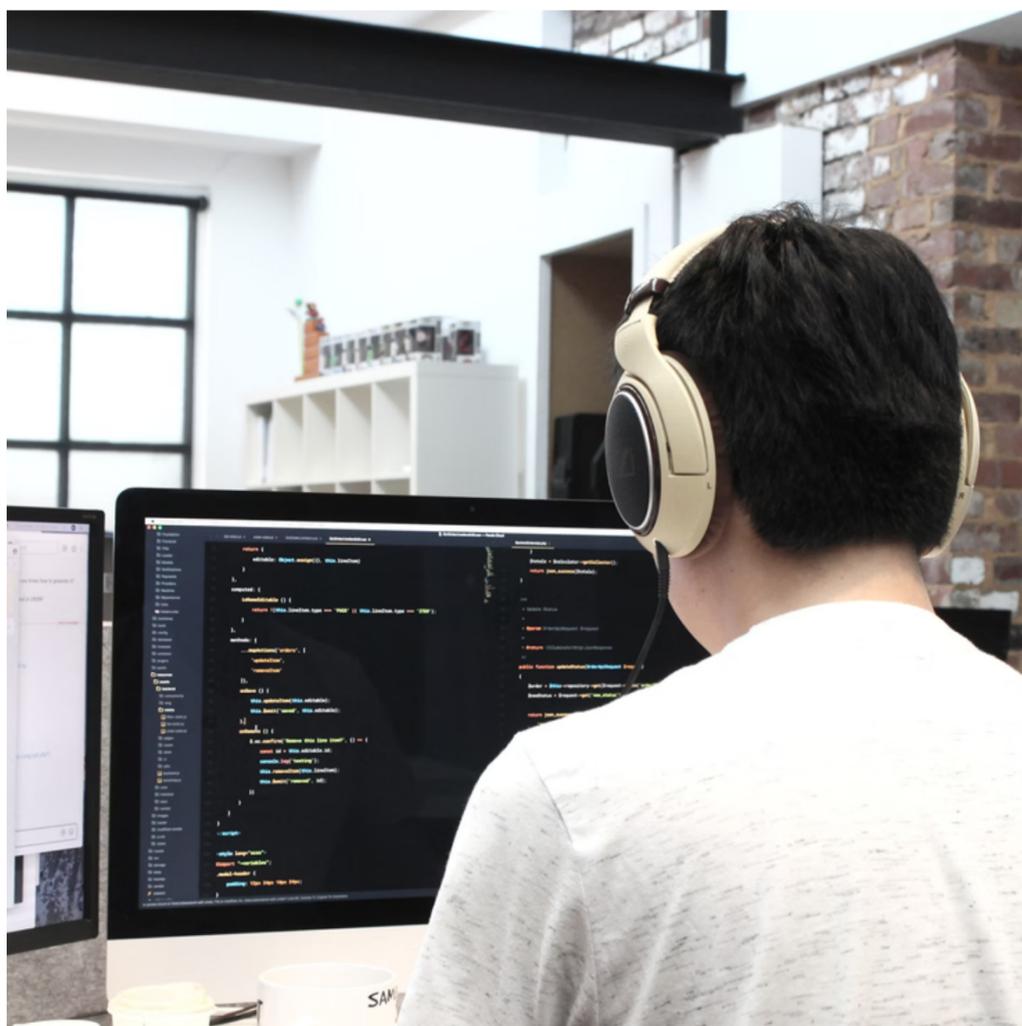
Vista

Es la capa con la que interactúa el cliente, es la interfaz gráfica en sí, lo que ve el usuario. Muestra y obtiene información, y funciona a través de EVENTOS que son las interacciones que tiene el usuario con la interfaz.

Construidas normalmente con HTML, CSS y JS.

la Vista es una representación del estado del Modelo en un momento concreto y en el contexto de una acción determinada.

Por ejemplo, si un usuario está consultando una factura a través de una aplicación web, la Vista se encargará de representar visualmente el estado actual de la misma en forma de página visualizable en su navegador. Si en un contexto B2B el cliente de nuestro sistema es a su vez otro sistema, la vista podría ser un archivo XML con la información solicitada. En ambos casos se trataría de la misma factura, es decir, la misma entidad del Modelo, pero su representación es distinta en función de los requisitos.



Modelo

El modelo define qué datos debe contener la aplicación. Si el estado de estos datos cambia, el modelo generalmente notificará a la vista (para que la pantalla pueda cambiar según sea necesario) y, a veces, el controlador (si se necesita una lógica diferente para controlar la vista actualizada).

El Modelo será también el encargado de gestionar el almacenamiento y recuperación de datos, es decir, incluirá mecanismos de persistencia o será capaz de interactuar con ellos. Dado que habitualmente la persistencia se delega a un motor de bases de datos, es muy frecuente encontrar en el Modelo la implementación de componentes tipo DAL (Data Access Layer, o Capa de Acceso a Datos) y ORMs.

Controlador

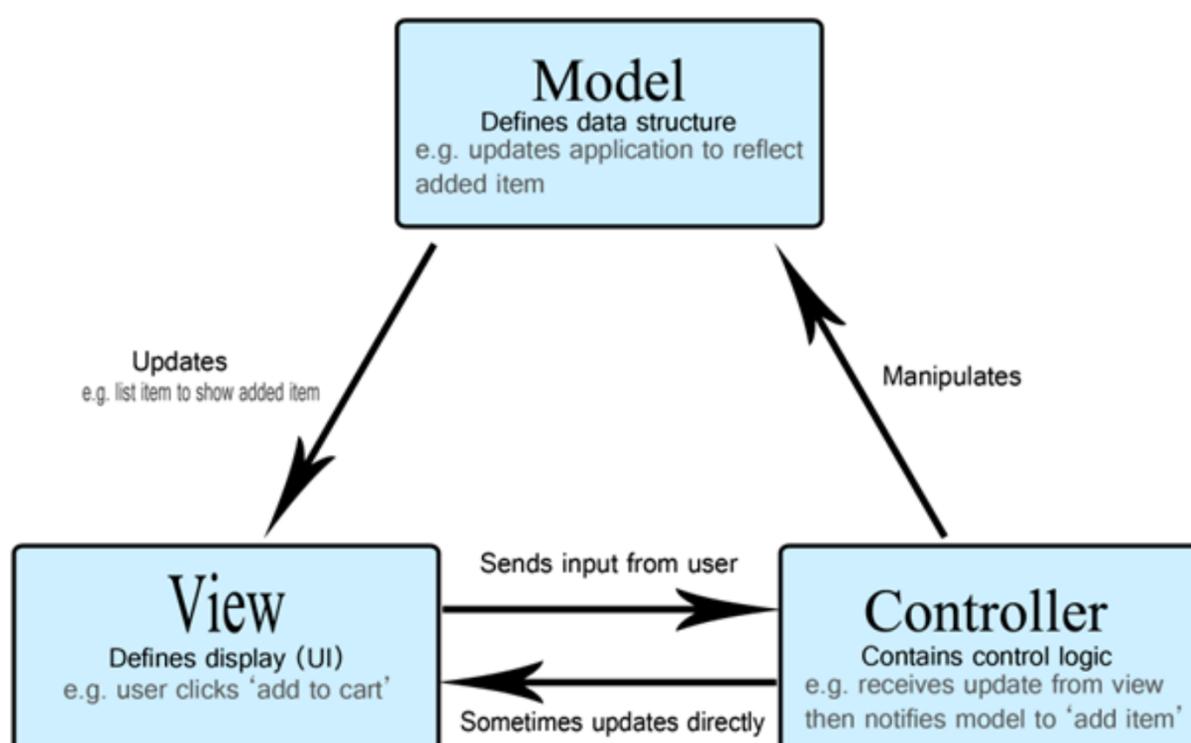
Responde a las acciones o eventos que realice el cliente a través de las vistas. Se encarga de elegir las vistas a mostrar y los datos a procesar. Es el enlace que interactúa con vistas y modelos.

La vista no tiene comunicación directa con el modelo ni viceversa, por eso el controlador cumple un rol fundamental en ese diálogo.

Este se encargará de atender las peticiones hechas por el cliente a través de la Vista, de entablar una comunicación directa con el Modelo para solicitar información almacenada en la base de datos y de volver con dicha información y traspasarla directamente a la Vista, para que esta última pueda mostrarla fácilmente a la persona que visita nuestra aplicación web.

Si bien dentro del patrón MVC estos tres componentes son fundamentales, el Controlador es quien toma un protagonismo importante.

Fuente: MVC, Glosario mozilla developer, consultado en febrero de 2024, disponible en <https://developer.mozilla.org/es/docs/Glossary/MVC/model-view-controller-light-blue.png>



MVC en la WEB

Como desarrollador web, este patrón probablemente será bastante familiar, incluso si nunca lo has usado conscientemente antes. Su modelo de datos probablemente esté contenido en algún tipo de base de datos. El código de control de su aplicación probablemente esté escrito en HTML / JavaScript , y su interfaz de usuario probablemente esté escrita usando HTML / CSS / o lo que sea. Esto se parece mucho a MVC, pero MVC hace que estos componentes sigan un patrón más rígido.

En los primeros días de la Web, la arquitectura MVC se implementó principalmente en el lado del servidor, con el cliente solicitando actualizaciones a través de formularios o enlaces, y recibiendo vistas actualizadas para mostrar en el navegador. Sin embargo, en estos días, mucha de la lógica se enviaba al cliente con la llegada de los almacenes de datos del lado del cliente, y XMLHttpRequest permitía actualizaciones parciales de la página según era necesario.

Una de las ventajas de trabajar con este patrón, como ya se ha dicho, es la escalabilidad y modularidad, lo que permite agregar nuevas funcionalidades o modificar algunas específicas sin afectar el funcionamiento general de la aplicación.

NodeJS

Node es una plataforma que permite la ejecución de rutinas de JS por fuera del navegador, La versión actual de Node utiliza el motor V8 de JS, que es el núcleo de como se ejecuta en Google Chrome, lo que se traduce en un excelente desempeño.

Para contar con Node en nuestras máquinas o servidores, solo es necesario realizar la instalación, ya sea utilizando el instalador disponible en la página web <https://nodejs.org/en/download>, o a través de un manejador de paquetes como YUM en el servidor.

Las aplicaciones escritas en Node se ejecutan bajo un único proceso, sin crear nuevos hilos para cada petición que reciben. Para evitar bloqueos, Node provee una serie de recursos y librerías asíncronas.

Por ejemplo, para la lectura y salida de información que requiera interacción con base de datos, recursos en línea o el sistema de archivos, Node retoma las operaciones en el momento en el que una respuesta es devuelta, lo que permite que en ese tiempo intermedio, se ejecuten otra serie de procesos que requieran de los recursos.

Esto también se traduce en que Node puede manejar una gran cantidad de peticiones o conexiones en un único servidor, sin la necesidad del manejo o administración de hilos concurrentes.

Otro de los elementos que se destacan de utilizar Node es que el JS que se escribe para la implementación del frontend, es prácticamente el mismo que se necesita para la creación de servidores, lo que permite que conociendo un solo lenguaje de programación se pueda trabajar en el lado del navegador o en el lado del servidor.

JS trabaja por versiones llamadas ECMAScript, cada nueva versión requiere actualización de los navegadores para soportarla, lo que hace que dependa de cada usuario si ya ha actualizado su navegador para soportar con una nueva funcionalidad o no, mientras que con Node, usted como desarrollador puede escoger la versión sobre la que va a trabajar y no necesita de actualizaciones por parte del usuario.



Instalación de NodeJS

El proceso para la instalación de NodeJS es muy sencillo, suponiendo que trabaja en Windows o Mac, basta con descargar el instalador y al ejecutarlo hacer clic en siguiente hasta finalizar, solo debe estar atento a que en la opción que indica agregar Node al PATH del sistema, esté marcada.

La instalación de Node incluye su manejador de paquetes npm, para verificar que todo se encuentre instalado correctamente puede probar ejecutar los comandos:

Para verificar la versión de Node instalada

```
> node -v
```

Para verificar la versión de npm instalada

```
> npm -v
```

```

Microsoft Windows [Versión 10.0.22631.3007]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\jolgo>npm --version
8.3.1

C:\Users\jolgo>node --version
v16.14.0

C:\Users\jolgo>

```

Con la instalación finalizada es hora de crear el primer proyecto con node.

Hola Mundo Node

El servidor que se levantará a continuación, es solo para una muestra, ya que los proyectos futuros se construirán utilizando un framework para Node llamado Express. Para iniciar con el proyecto debe:

- definir el directorio de trabajo o crear uno nuevo, puede hacerlo desde la terminal o desde el explorador de archivos
- Inicializar un nuevo proyecto desde la terminal con el comando `npm init` y esto creará un archivo llamado `package.json` con un contenido similar al siguiente

```
{
  "name": "m2",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

La etiqueta “name” es asignada automáticamente dependiendo del nombre de la carpeta donde se ha inicializado el proyecto, pero puede ser modificado.

Otro de los puntos de interés en el package es el “main”, allí se indica el nombre del archivo que el proyecto espera para comenzar a ejecutarse, es conocido como el **entry point**. El nombre por defecto es “index.js” pero puede ser modificado, para el ejemplo y el resto de proyecto se renombrará a “app.js”

- Ahora debe crear el archivo “app.js”, abrir la carpeta del proyecto en VS Code y agregar en el **entry point** el código a continuación para crear el primer servidor web con node.

```
const http = require('node:http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

El proyecto se debe ver como el siguiente:

```

1  const http = require('node:http');
2
3  const hostname = '127.0.0.1';
4  const port = 3000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/plain');
9    res.end('Hello World\n');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${hostname}:${port}/`);
14 });

```

```

jolge@danielAsus MINGW64 /d/bootcamp/codigos/innovador/m2
$ node app.js
Server running at http://127.0.0.1:3000/

```

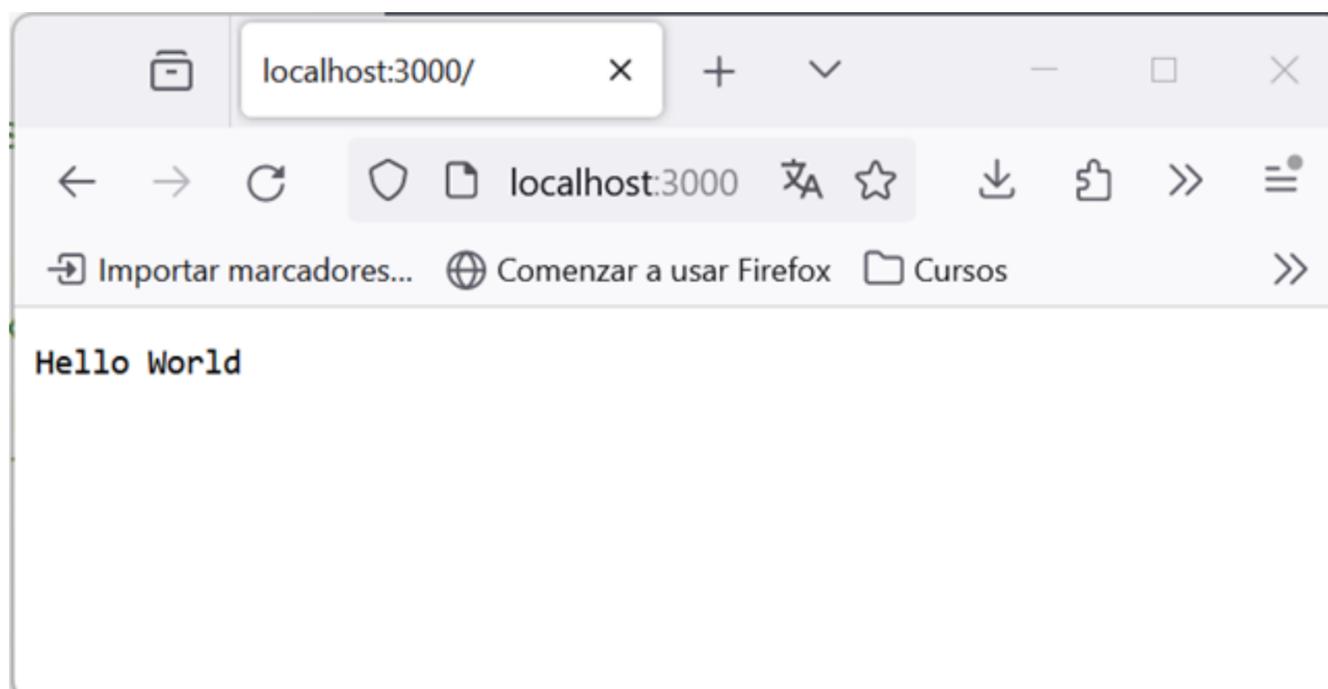
Para ejecutar el proyecto basta con abrir una terminal, puede utilizar el comando “CTRL + ñ” en windows o utilizar la git bash y navegar hasta la carpeta o en el explorador, navegar a la carpeta y con clic derecho, seleccionar la opción “Abrir en terminal” u “Open git bash here”. Luego de abierta la terminal ejecutar el comando “node app.js”

```

$ node app.js
Server running at http://127.0.0.1:3000/

```

Debería obtener una respuesta como la de la imagen, ahora debe copiar la dirección que se definió en el código, se explicará en un momento, al pegarla en un navegador debe aparecer en pantalla el mensaje “Hello World”.



Si todo ha salido bien, el resultado debe ser el mismo que el presentado aquí.

Para detener el servidor, desde la terminal presionar CTRL + C

Ahora se revisarán las líneas de código agregadas al archivo app.js, no se entrarán en detalles, ya que como se mencionó anteriormente, el servidor web se implementará utilizando Express.

Se utiliza como referencia el número de línea que entrega VS Code al pegar el código.

La línea 1 hace el llamado de el módulo http de node, este paquete es asignado a una variable y cuenta con los métodos para levantar el servidor.

las líneas 3 y 4, definen el host y el puerto en donde se ejecuta el proyecto.

De las líneas 6 a la 10 se crea el servidor y se establece el código de respuesta, el tipo de respuesta y la respuesta que se enviará a la petición.

las líneas 12 a 14, levanta el servidor dejando el escuchar de peticiones activo, indicando el puerto y el host sobre el que quedará funcionando, finalmente se logea en consola un mensaje indicando que el servidor se encuentra corriendo en el host y puerto establecidos.

NPM como manejador de paquetes

NPM es el registro de software más grande del mundo en donde es posible, a través de la CLI (Command Line Interface) descargar paquetes y agregarlos al proyecto. npm también permite compartir nuestros códigos con otros usuarios o incluso que se utilicen para el manejo de desarrollos privados.

El primer paquete con el que se va a trabajar es Express, aunque puede ser buscando desde la biblioteca y buscador de npm, también cuenta con sitio web oficial.

[express - npm](#)

[Instalación de Express](#)

Como la documentación lo indica, lo único necesario para instalar el paquete es ejecutar el comando `npm i express --save` nuevamente en una terminal dentro del proyecto, es decir, en el mismo punto donde se levantó el servidor, al finalizar el proceso de `npm`, ya se podrá hacer uso de Express.

Primeros pasos con Express

Express es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles.

Con miles de métodos de programa de utilidad HTTP y middleware a su disposición, la creación de una API sólida es rápida y sencilla.

Ahora que se tiene instalado Express, es hora de modificar el ejercicio anterior para levantar un primer servidor ahora usando Express.

Lo primero es notar que ahora el archivo "package.json" tiene una nueva sección bajo la llave "dependencies" con express y la versión instalada.

```
{
  "name": "m2",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.3"
  }
}
```

La sección de dependencias se actualizará cada que se agregue un nuevo paquete, algunas dependencias pueden ser utilizadas o configuradas para funcionar el modo desarrollo, por lo que tendrán su sección adicional también.

Ahora, siguiendo la documentación oficial, Ejemplo "Hello World" de Express, se debe actualizar el contenido del archivo "app.js" para crear un nuevo servidor que escucha las conexiones del puerto 3000 y responde "Hello World!" a las peticiones de la ruta raíz ("/"), ahora utilizando la configuración de Express. Cualquier petición a una ruta diferente provoca que la aplicación conteste con un error 404 Not Found.

```
const express = require('express')
const app = express()
const port = 3000

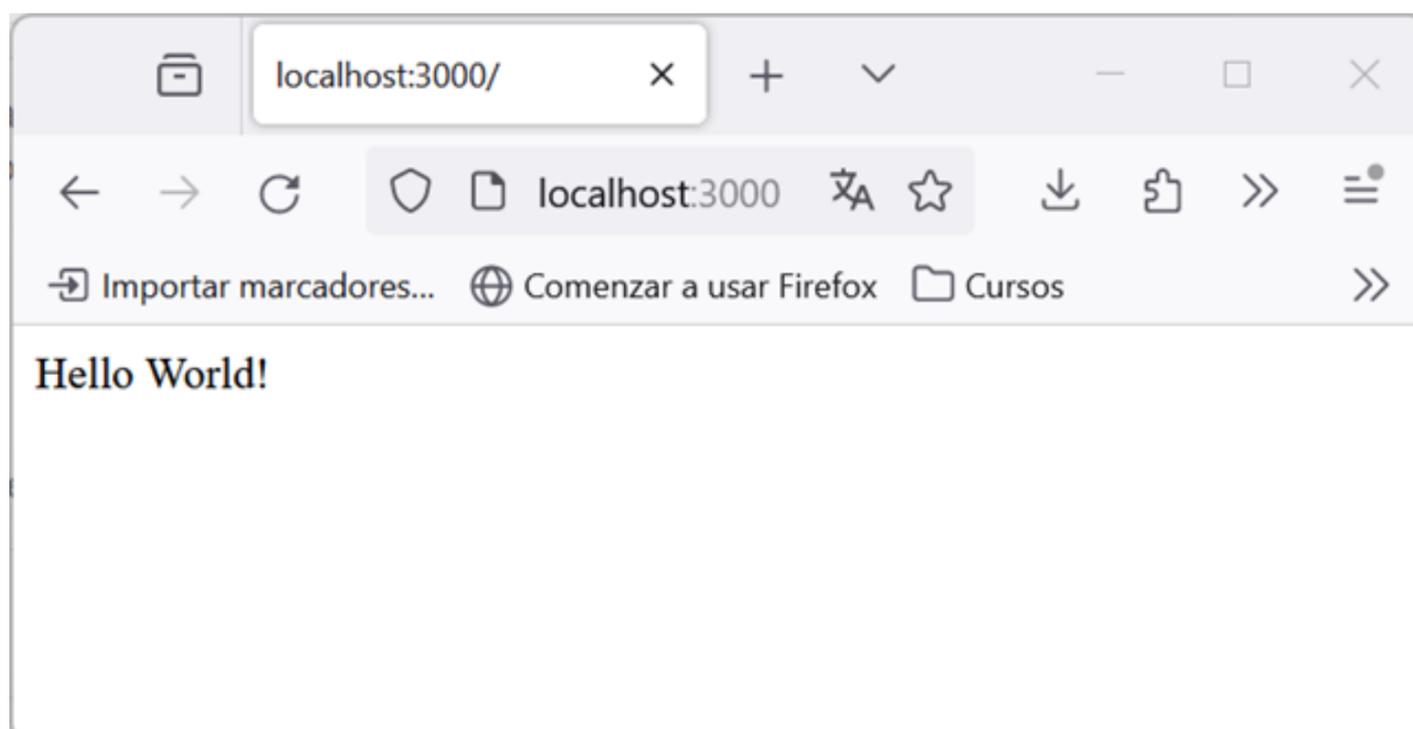
app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Ahora es necesario ejecutar el servidor, para ello se utiliza nuevamente el comando del ejemplo anterior "node app.js".

```
$ node app.js
Example app listening on port 3000
```

Al ingresar en el navegador a <http://localhost:3000/>, deberá ver como resultado el mensaje.



La explicación del nuevo servidor es similar a la anterior, sin embargo, ya que se usará Express para el resto de aplicaciones, se revisará de mejor manera el bloque central “app.get”, ya que será uno de los métodos más comúnmente usado.

La línea 1 y 2 hacen el llamado o importación del paquete, en este caso express y se asigna a una variable, a continuación se crea la aplicación ejecutando el método “express()”, normalmente esta aplicación se asigna a una variable llamada “app”, que a partir de este punto contará con los métodos para el enrutamiento, configuraciones de middleware, uso de vistas, entre otros.

La línea 3 define el puerto sobre el cual se ejecutará el servidor, por defecto se utiliza el 3000, pero puede ser cualquier otro puerto que no se encuentre ocupado o donde corran aplicaciones por defecto.

Uno de los métodos dentro de “app” será el “listen()” de la línea 9, el cual levanta el servidor que queda atento a las peticiones que lleguen a las rutas establecidas o contestando con un mensaje de error para las rutas no definidas. El mensaje del console.log() es para que nosotros como desarrolladores identifiquemos que el servidor ha levantado.

Las líneas 5 y 6 tiene definido el manejo de la ruta raíz a través del método “app.get()”, lo cual hace referencia a un método HTTP de los disponibles en Express. [Express JS HTTP Methods - GeeksforGeeks](#).

El primer parámetro define la ruta que la aplicación está atendiendo, siendo “/” para la raíz y, por ejemplo, “/about-us” para una ruta que entregue información acerca de la empresa o proyecto.

Dentro de este método get aparecen dos parámetros llamados req y res, estos hacen referencia a la petición “request” y a la respuesta “response”.

Cuando una nueva petición es recibida, el evento request se dispara, dejando a disponibilidad dos objetos, req con la petición entrante, del cual se pueden consultar query strings, parámetros, cierto y cabecera y, res para la respuesta entregada desde el servidor, ya sea un texto, un json o una vista.

Más adelante se implementarán más métodos HTTP para las peticiones y se revisarán las opciones con los objetos req y res, estos dos objetos funcionan como parámetros o entradas de una función anónima que se ejecuta como respuesta a la petición hecha a la ruta definida y contiene la respuesta que se le envíe al usuario u otro sistema.

Esta función anónima, puede tener tanta lógica como haga falta siempre que se ubique antes de un res o que se maneje flujo de la respuesta dependiendo de la lógica que se valida, sin embargo, bajo un res en un nivel determinado, no se deben incluir más instrucciones.

Express Generator

Hasta el momento se ha creado una aplicación sencilla línea por línea, sin embargo, Express cuenta con un generador de aplicaciones que permite crear una parte del esqueleto del proyecto, se conoce como express generator y debe ser instalado por el manejador de paquetes antes de poder hacer uso de él.

Para instalarlo basta con ejecutar, dentro de la carpeta que designamos para el proyecto, el comando:

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

Ahora es posible crear un proyecto con solo una instrucción, como por ejemplo un proyecto con vistas utilizando el motor EJS:

```
express --view=ejs nombre_app
```

Puede consultar las opciones a la hora de crear un proyecto en la documentación oficial [Generador de aplicaciones Express](#) y con el comando “express -h”.

The screenshot shows a Visual Studio Code editor window with a project named 'm2'. The Explorer sidebar on the left shows the following structure:

- OPEN EDITORS: package.json
- M2:
 - bin
 - public
 - routes
 - views
 - app.js
 - package.json

The main editor displays the content of package.json:

```

1 {
2   "name": "m2",
3   "version": "0.0.0",
4   "private": true,
5   "scripts": {
6     "start": "node ./bin/www"
7   },
8   "dependencies": {
9     "cookie-parser": "~1.4.4",
10    "debug": "~2.6.9",
11    "ejs": "~2.6.1",
12    "express": "~4.16.1",
13    "http-errors": "~1.6.3",
14    "morgan": "~1.9.1"
15  }
16 }
17

```

The terminal at the bottom shows the command to run the application:

```

run the app:
> SET DEBUG=m2:* & npm start

```

The terminal prompt is: `jolgo@danielAsus MINGW64 /d/bootcamp/codigos/innovador/m2`

El proyecto generado se debe ver como la imagen anterior, con una serie de carpetas nuevas creadas, concretamente ha creado todo lo de la siguiente lista.

- create : public\
- create : public\javascripts\
- create : public\images\
- create : public\stylesheets\
- create : public\stylesheets\style.css
- create : routes\
- create : routes\index.js
- create : routes\users.js
- create : views\
- create : views\error.ejs
- create : views\index.ejs
- create : app.js
- create : package.json
- create : bin\
- create : bin\www

install dependencies:
> npm install

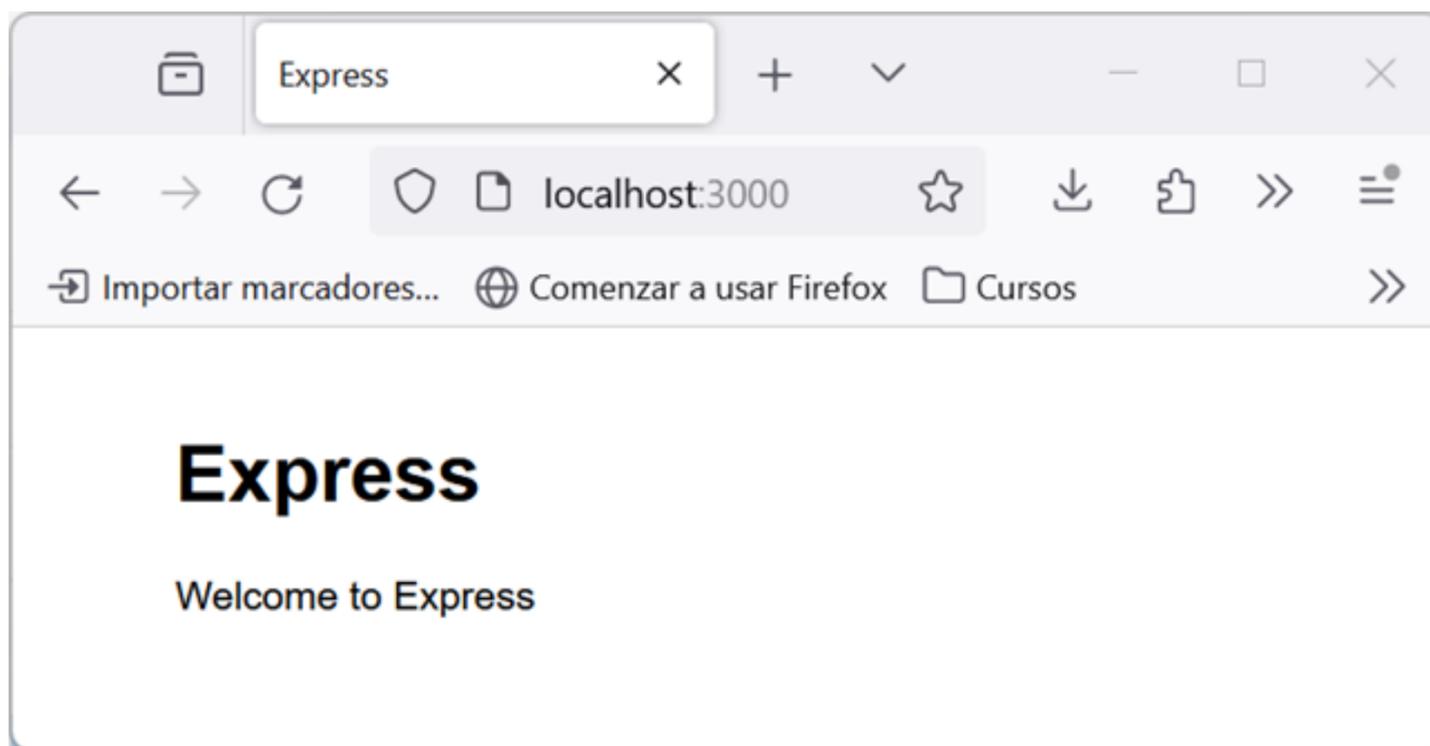
Muchos elementos que aún desconocemos pero que tendrán mucho sentido en la medida que se avance un poco con la herramienta.

Antes de ejecutar el servidor es necesario realizar la instalación de las dependencias como lo indica el final del mensaje “npm install”, dicha instalación permitirá ejecutar el proyecto con normalidad y agrega una carpeta adicional llamada “node_modules”. Con las dependencias instaladas ya es posible ejecutar el servidor.

Este tipo de proyectos generados tiene una particularidad dentro del “package.json” y es que agrega una nueva sección llamada “scripts”, la cual contiene diferentes formas para ejecutar el servidor; para este caso, y una configuración casi que por defecto, el servidor se levanta con el comando “npm start”, obteniendo una respuesta como la siguiente:

```
$ npm start
> m2@0.0.0 start
> node ./bin/www
```

Ahora con el servidor arriba, se puede visitar nuevamente el localhost en un navegador web, indicando el puerto seleccionado y debería tener una respuesta como la siguiente:



Con esto, hemos creado nuestra primera aplicación utilizando el generador de Express, el siguiente paso es revisar qué son todas las carpetas o la estructura del servidor que el generador ha definido.