UTN.BA

Nuestro Primer Servidor

Repasemos... *Qué es Node.js*

- Node.js es un entorno de tiempo de ejecución de JavaScript.
- Node.js fue creado por los desarrolladores originales de JavaScript e incluye todo lo que se necesita para **ejecutar un programa escrito en JavaScript por fuera del navegador**.
- Se basa en el motor de tiempo de ejecución JavaScript V8, el mismo que usa Chrome para convertir el Javascript en código máquina.
- Node.js está escrito en C++ y dispone de módulos nativos.

Funcionamiento

- Usa el **Patrón Reactor:** un **modelo de entrada y salida sin bloqueo** y **controlado por eventos.** Así logra seguir siendo liviano y eficiente frente a las aplicaciones en tiempo real de uso de datos que se ejecutan en los dispositivos. Esto permite soportar decenas de miles de conexiones al mismo tiempo mantenidas en el bucle de eventos.

Funcionamiento

- Brilla en la **creación de aplicaciones de red rápidas**, ya que es capaz de manejar una gran cantidad de conexiones simultáneas con un alto nivel de rendimiento, lo que equivale a una **alta escalabilidad**.

Funcionamiento

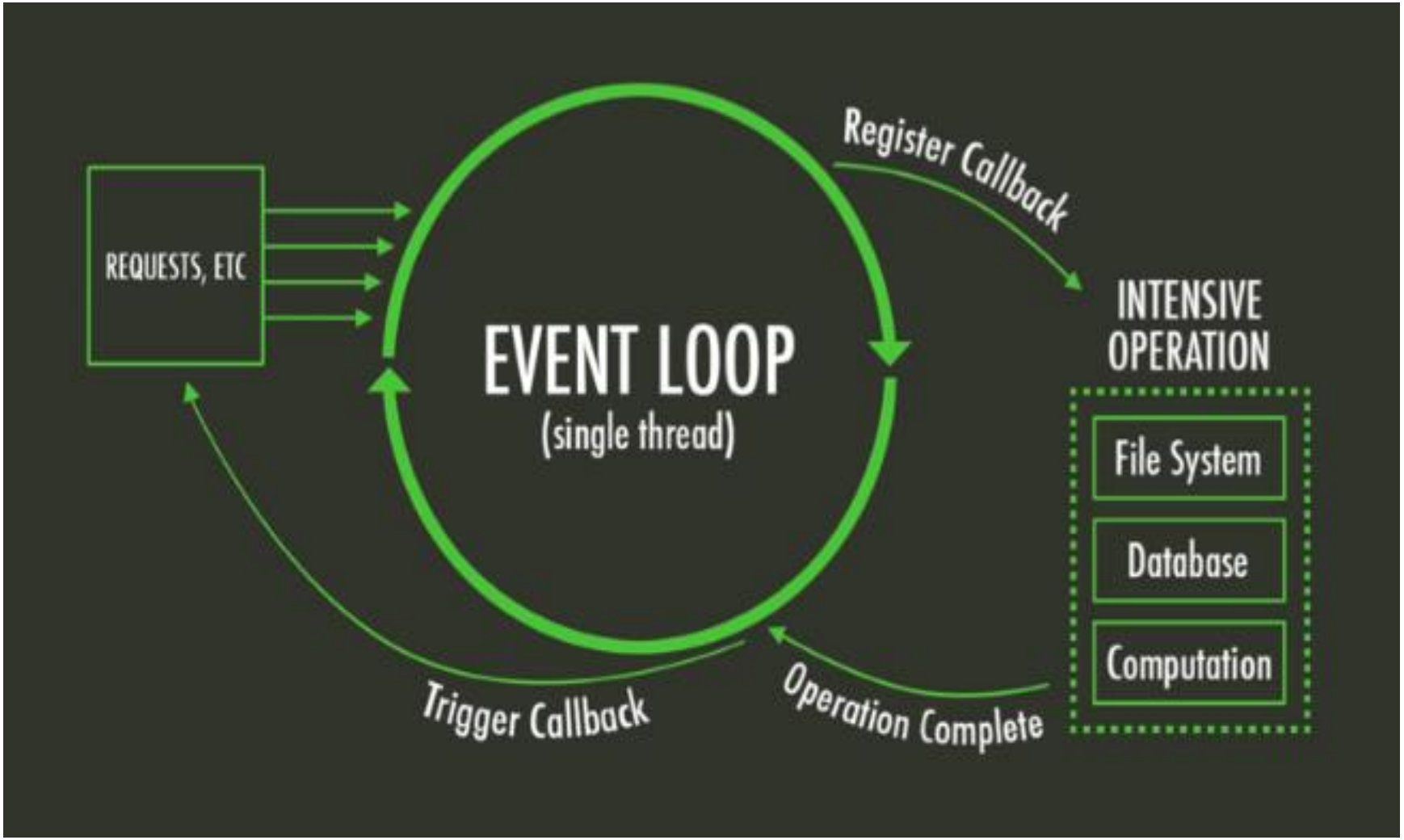
- Opera en **un solo subproceso**, a diferencia de las técnicas tradicionales de servicios web donde cada conexión (que crea una solicitud) genera un nuevo subproceso ocupando la mucha más memoria RAM del sistema.

UTN.BA

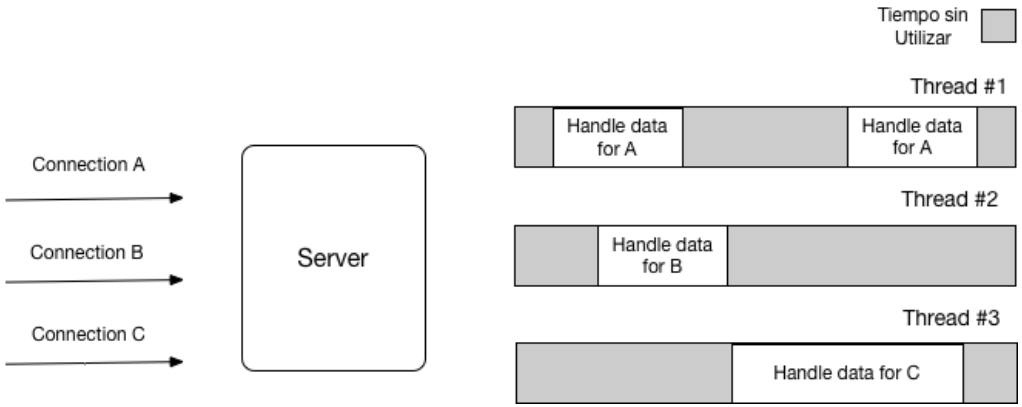
Funcionamiento

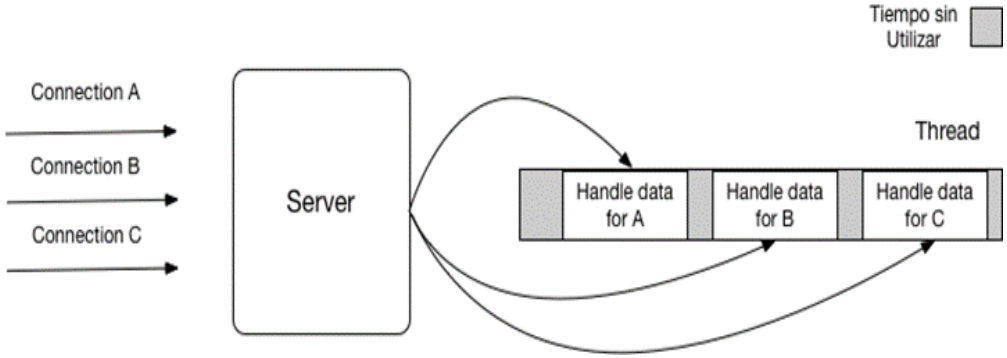
- Node está completamente **controlado por eventos**: Vamos a tener un subproceso que procesa un evento tras otro.

Funcionamiento



Funcionamiento





UTN.BA

Primer Servidor en Node.js

- Tener instalada la última versión de **Node.js** “últimas características” https://nodejs.org/es/
- **Consola**
- **NVM** https://github.com/coreybutler/nvm-windows/releases
- **Visual Studio code** https://code.visualstudio.com/
- **Git** https://git-scm.com/

Node.js

1. Crear una carpeta de proyecto
2. Crear un archivo llamado ***index.js*** dentro de esa carpeta
3. Inicializar un proyecto Node.js con el comando ***npm init***
4. Escribir código en ***index.js***
5. Guardar y ejecutar con ***node index.js o nodemon index.js*** desde la **consola externa** o desde la **consola interna de Visual Studio Code**
6. Si ejecuto con **node** el programa finalizará mostrando resultados en consola. Con **nodemon** la consola quedará bloqueada atenta a los cambios en el código, que en ese caso se relanzará la ejecución nuevamente.

UTN.BA

Módulos

- Un módulo es un **conjunto** de **funciones** y **objetos** de **JavaScript** que las **aplicaciones externas pueden usar**.
- Node.js posee varios módulos incorporados (nativos) compilados en binario. Estos módulos básicos están **definidos** en el código fuente de Node en la **carpeta lib/**.
- Los **módulos básicos** tienen la preferencia de **cargarse primero** si su identificador es pasado **desde require()**.
- Por ejemplo, require('http') siempre devolverá lo construido en el módulo HTTP, incluso si hay un fichero con ese nombre.

UTN.BA

Módulo HTTP

- HTTP es un **módulo nativo** de Node.js
- Trabaja con el **protocolo HTTP**, que es el que se utiliza en Internet para transferir datos en la Web.
- Nos va a servir para **crear un servidor HTTP** que acepte solicitudes desde un cliente web.
- Para poder utilizarlo en nuestro código, tenemos que requerirlo mediante la instrucción "require()".
- Por ejemplo, require('http') siempre devolverá lo construido en el módulo HTTP.

UTN.BA

Servidor HTTP

```
var http = require("http");
```

- A partir de este momento tenemos una **variable http** (que en realidad es un objeto) sobre la que **podemos invocar métodos** que estaban en el módulo requerido.
- Por ejemplo, una de las tareas implementadas en el módulo HTTP es la de **crear un servidor**, que se hace con el módulo "**createServer()**".
- Este método recibirá un callback que se ejecutará cada vez que el servidor reciba una petición.

UTN.BA

Servidor HTTP

```
var server = http.createServer(function (peticion, respuesta){
   respuesta.end("Hola");
});
```

- La función **callback** que enviamos a createServer() **recibe** dos parámetros que son la **petición** y la **respuesta**.
- La petición por ahora no la usamos, pero contiene datos de la petición realizada.
- La respuesta la usaremos para enviarle datos al cliente que hizo la petición.
- De modo que "**respuesta.end()**" sirve para **terminar** la **petición** y **enviar** los **datos** al **cliente**.

UTN.BA

Servidor HTTP

```
server.listen(3000, function(){
   console.log("tu servidor está listo en " + this.address().port);
});
```

- Con esto le decimos al **servidor** que **escuche** en el **puerto 3000**, aunque podríamos haber puesto cualquier otro puerto que nos hubiera gustado.
- "**listen()**" **recibe** también una **función callback** que realmente no sería necesaria, pero que nos sirve para hacer cosas cuando el servidor se haya iniciado y esté listo.
- Simplemente, en esa función callback **indico** que estoy **listo** y **escuchando** en el **puerto configurado**.

UTN.BA

Servidor HTTP

```
var http = require("http");
var server = http.createServer(function (peticion, respuesta){
   respuesta.end("Hola");
});
server.listen(3000, function(){
   console.log("tu servidor está listo en " + this.address().port);
});
```

- Este es el código completo. En muy pocas líneas de código **generamos un servidor web que está escuchando en un puerto dado**. Ahora podemos guardar ese archivo con extensión .js, por ejemplo “servidor.js”.

Ejercicio2.js

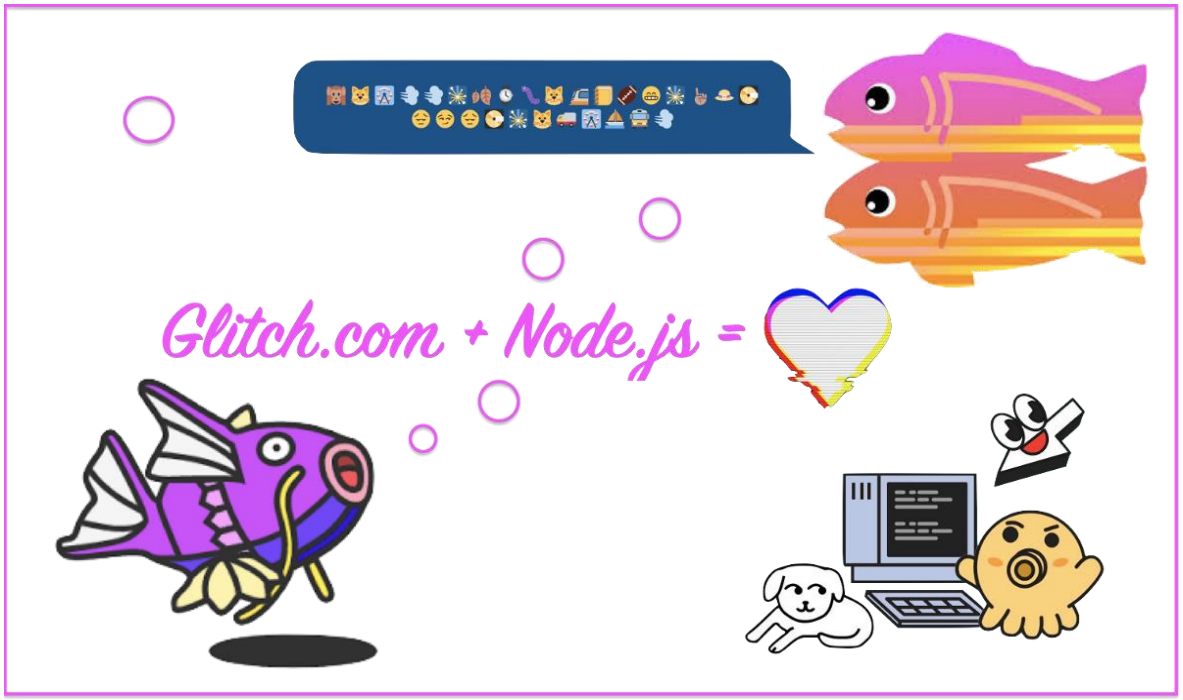
Desarrollar un servidor en node.js que escuche peticiones en el puerto 8080 y responda un mensaje de acuerdo a la hora actual:

- Si la hora actual se encuentra entre las 6 y las 12 hs será 'Buenos días!'.
- Entre las 13 y las 19 hs será 'Buenas tardes!'.
- De 20 a 5 hs será 'Buenas noches!’.

Se mostrará por consola cuando el servidor esté listo para operar y en que puerto lo está haciendo. Pasen su Servidor por el chat.

Prácticas

https://glitch.com/



Ejercicio 3

- Desarrollar un servidor en node.js que con cada requerimiento devuelva como resultado un objeto con las siguientes características:

```
{    id: (número aleatorio entre 1 y 10),
   title: "Producto " + (número aleatorio entre 1 y 10),
   price: (número aleatorio entre 0.00 y 9999.99),
thumbnail: "Foto " + (número aleatorio entre 1 y 10)}
```

UTN.BA

Manejo de Archivos

- En todo sistema, es posible que nos topemos con la necesidad de que algunos **datos persistan más allá de la ejecución del programa**.
- Una de las opciones con las que contamos es el uso de archivos.
- Según el caso, existen ventajas y desventajas en utilizar el sistema de archivos como medio de almacenamiento de información.

UTN.BA

Manejo de Archivos

- Son fáciles de usar.
- No requieren el uso de programas externos para su creación, lectura o edición.
- En ocasiones, pueden ser abiertos y editados desde programas de edición de texto simples como un bloc de notas (¡siempre que se trate de texto!).
- Son fáciles de compartir o enviar a otros usuarios/programas.

UTN.BA

Manejo de Archivos

- Consultas sobre algún dato puntual entre todos los datos almacenados (y no podamos guardar todo el lote de datos en memoria).
- Ediciones de datos puntuales (que no requieren sobreescribir el archivo por completo).
- Lecturas que combinen datos obtenidos de varios archivos (nuevamente, suponiendo que no podemos guardar todos los datos en memoria).
- Probablemente sea mejor considerar el uso de un motor de base de datos.

Módulo nativo file system: fs

- **fs** es la abreviatura en inglés para file system o sistema de archivos y es, además, uno de los módulos más básicos y útiles de Node.js.
- En Node.js es posible manipular archivos a través de fs (crear, leer, modificar, etc.).
- La mayoría de las funciones que contiene este módulo pueden usarse tanto de manera sincrónica como asincrónica.

Aclaración: Hay que tener en cuenta que esto sólo aplica a Node.js, desde el navegador no es posible manipular archivos dado que sería muy inseguro.

Uso de fs en nuestro código

Para poder usar este módulo solo debemos **importarlo** al comienzo de nuestro archivo fuente, utilizando cualquiera de estas declaraciones:

- con import
  - ***import fs from 'fs'***
- con la función require
  - ***const fs = require(‘fs’)***

Leer un archivo

- *fs.readFileSync(path, encoding)*

```
const data = fs.readFileSync('./test-input-sync.txt', 'utf-8')
console.log(data)
```

- El **primer parámetro** es un **string** con la **ruta** del **archivo** que queremos **leer**
- El **segundo parámetro** indica el **formato** de **codificación** de **caracteres** con que fue escrito el dato que estamos leyendo
- El formato que utilizaremos con más frecuencia será **'utf-8'** (inglés: 8-bit Unicode Transformation Format, español: Formato de Codificación de caracteres Unicode).

Sobreescribir un archivo

- *fs.writeFileSync(ruta, datos) //sobreescribe archivo*

```
fs.writeFileSync('./test-output-sync.txt', 'ESTO ES UNA PRUEBA\n')
```

- El **primer parámetro** es un **string** con la **ruta** del **archivo** en el que queremos **escribir**
- El **segundo parámetro** indica **lo que queremos escribir**.
- La función admite un **tercer parámetro opcional** para **indicar** el **formato** de codificación de caracteres con que queremos escribir el texto: por defecto *'utf-8'.*
- Si la **ruta** provista fuera **válida**, pero el nombre de **archivo no existiera**, la función creará un **nuevo archivo** con el nombre provisto.