- **¿Qué es Node js?**

Node es un programa de servidor. Node.js es una forma de ejecutar JavaScript en el servidor. Node.js es un entorno Javascript del lado del servidor, basado en eventos. Node ejecuta javascript utilizando el motor V8, desarrollado por Google para uso de su navegador Chrome. Aprovechando el motor V8 permite a Node proporcionar un entorno de ejecución del lado del servidor que compila y ejecuta javascript a velocidades increíbles. El aumento de velocidad es importante debido a que V8 compila Javascript en código de máquina nativo, en lugar de interpretarlo o ejecutarlo como bytecode. Node es de código abierto, y se ejecuta en Mac OS X, Windows y Linux.

- **¿Qué usos reales se le da a nodejs?**

Aplicaciones web, aplicaciones en línea de comandos, scripts para administración de sistemas, todo tipo de aplicaciones de red, etc.

- **¿Porqué es importante node js?**

El desarrollo es más rápido, la ejecución de tests de unidad se puede hacer más rápido, las aplicaciones son más rápidas y por tanto la experiencia de usuario es mejor y menor coste de infraestructura.

- **Paypal y NodeJS**

Paypal tenía un problema en su plataforma por la existencia de entornos dispersos dentro de su semántica que realizaban tareas de forma independiente entre las aplicaciones y el servidor. Pero después de implementar NodeJS resolvieron el inconveniente, y lograron trabajar el doble de rápido en las tareas de desarrollo web y se había reducido un tercio la cantidad de código.

- **Netflix y Node js**

Netflix pudo reducir el tiempo de carga de su web en torno a un 70%. Además gracias a NodeJS la plataforma logró perfeccionar las interfaces de los usuarios y optimizar la compilación de sus directorios.

- **Linkedin y NodeJS**

La aplicación para móviles de Linkedin está desarrollada íntegramente con NodeJS. Linkedin ha conseguido una notable reducción de recursos, Mejora indiscutible en la experiencia de usuario, Optimización de la velocidad de carga

- **La NASA y NodeJS**

La NASA se enfrentaba a un problema de seguridad y la herramienta utilizada fue NodeJS y el resultado fue un sistema mucho más optimizado que se basa una única base de datos para dar servicio a todas las necesidades de la empresa. Sus mejoras derivaron en una reducción (300 veces inferior) en los tiempos de espera.

- **Formas en las que Node permite la creación de Buffers:**

1. Método 1: var buf = new Buffer(10);
2. Método 2: var buf = new Buffer([10, 20, 30, 40, 50]);

3. Método 3: var buf = new Buffer("Simply Easy Learning", "utf-8");

- **Sintaxis de la función write y sus componentes:**
  buf.write(string[, offset][, length][, encoding])
  - String: es el dato que se quiere escribir en el buffer
  - Offset: es el indice del buffer donde iniciara a escribir, por default 0.
  - Length: Es el numero de bytes a escribir.
  - Encoding.-Es la codificación a utilizar, por default es “utf8”

- **Método para convertir un buffer a su representación JSON:**
  buf.toJSON()
- **Método para Concatenar buffers**
  Buffer.concat(list[, totalLength])
- **Método para comparar Buffers**
  buf.compare(otherBuffer);
- **Método para copiar Buffer**
  buf.copy(targetBuffer[,targetStart][,sourceStart][,sourceEnd])
  - targetBuffer: Buffer donde se copiará.
  - targetStar: numero opcional, por default 0.
  - sourceStart: numero, opcional, por default 0.
  - sourceEnd: numero opcional, por default la longitud del buffer.
- **Mencione 5 métodos de la tabla de métodos de buffer**
  - new Buffer(size) Allocates a new buffer of size octets
  - buf.length Returns the size of the buffer in bytes.
  - buf[index] Get and set the octet at index.
  - buf.equals(otherBuffer) Returns a boolean if this buffer and otherBuffer have the same bytes.
  - buf.slice([start][, end]) Returns a new buffer which references the same memory as the old, but offset and cropped by the start (defaults to 0) and end (defaults to buffer.length) indexes.
- ¿**Qué son los streams? Mencione los 4 tipos de streams**

son objetos que nos permiten leer datos de una fuente o escribir datos de manera continua. Cada tipo de stream es una instancia de unEventEmittery lanza diferentes eventosen distintas instancias de tiempo.

  - Readable, Writable, Duplex, Transform
- **Piping**

es un mecanismo usado para pasar información donde nosotros proveemos la salida de un stream como la entrada de otro stream.

- **Chaining**

un mecanismo usando con las operaciones de piping para conectar la salida de un flujo a otro y crear una cadena de operaciones de múltiples flujos.

- **Sintaxis para abrir un archivo en nodede forma asíncrona y sus parámetros:**
  `fs.open(path, flags[, mode], callback)`
  - path: nombre del archivey ruta.
  - Flags:Banderas de apertura.
  - Mode:Modo de archivo, permisos, etc.
  - Callback:Funcion con dos argumentos (err, fd)
- **Algunas banderas:**

Flag	Description
r	Open file for reading. An exception occurs if the file does not exist.
r+	Open file for reading and writing. An exception occurs if the file does not exist.
rs	Open file for reading in synchronous mode.
rs+	Open file for reading and writing, asking the OS to open it synchronously. See notes for 'rs' about using this with caution.
w	Open file for writing. The file is created (if it does not exist) or truncated (if it exists).
wx	Like 'w' but fails if the path exists.
w+	Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists).
wx+	Like 'w+' but fails if the path exists.
a	Open file for appending. The file is created if it does not exist.
ax	Like 'a' but fails if the path exists.

- **Método para obtener información del archivo y sus parámetros**:
  `fs.stat(path, callback)`
  - Path: Nombre de archivoy
  - Ruta: Callback.
  - Funcion que regresa dos argumentos (erry stats) donde stats es un objeto de el sistema de archivos.
- **Algunos métodos de Stat**

Method	Description
stats.isFile()	Returns true if file type of a simple file.
stats.isDirectory()	Returns true if file type of a directory.
stats.isBlockDevice()	Returns true if file type of a block device.
stats.isCharacterDevice()	Returns true if file type of a character device.
stats.isSymbolicLink()	Returns true if file type of a symbolic link.
stats.isFIFO()	Returns true if file type of a FIFO.
stats.isSocket()	Returns true if file type of a socket.

- **Método para escritura de un archivo y parámetros**
  `fs.writeFile(filename, data[, options], callback)`
  - Path: Nombre de archivoy ruta.
  - Data: String o buffer a escribir enel archivo.
  - Options: Opciones de archivo{encoding, mode, flag} por default{utf8, 0666, w}
  - Callback: Funcion callback con un parametro (err)en caso de error.
- **Método para lectura de un archivo y parámetros**
  `fs.read(fd, buffer, offset, length, position, callback)`
  - Fd: Descriptor del archivo, retornado por fs.open();
  - Buffer: Buffer donde se almacenarán los datos leídos
  - Offset: Offset del buffer.
  - Length: Numero de bytes a escribir.
  - Position: Posicion de inicio de lectura en el archivo.
  - Callback: Funcion callback con tresparametros(err, bytesRead, buffer).
- **Método para cerrar un archivo y parámetros:**
  `fs.close(fd, callback)`
  - fd: Descriptor del archivo, retornado por fs.open();
  - Callback: Funcion callback sinparametros.
- **Módulos en node**
  usados mientras se desarrolla cualquier tipo de aplicación basada en Node.js. Algunos son

S. No.	Module Name & Description
1	**OS Module** Provides basic operating-system related utility functions.
2	**Path Module** Provides utilities for handling and transforming file paths.
3	**Net Module** Provides both servers and clients as streams. Acts as a network wrapper.
4	**DNS Module** Provides functions to do actual DNS lookup as well as to use underlying operating system name resolution functionalities.
5	**Domain Module** Provides ways to handle multiple different I/O operations as a single group.

- **Modulo os:**
  Provee algunas funciones basicas relacionadas con el sistema operativo
  `var os = require("os")`

S. No.	Method & Description
1	**os.tmpdir()** Returns the operating system's default directory for temp files.
2	**os.endianness()** Returns the endianness of the CPU. Possible values are "BE" or "LE".
3	**os.hostname()** Returns the hostname of the operating system.
4	**os.type()** Returns the operating system name.
5	**os.platform()** Returns the operating system platform.

Properties

S. No.	Property & Description
1	**os.EOL** A constant defining the appropriate End-of-line marker for the operating system.

- **Modulo path**
  Es usado para manejar y transformar rutas de archivos. `var path = require("path")`

Methods

S.No.	Method & Description
1	**path.normalize(p)** Normalize a string path, taking care of '..' and '.' parts.
2	**path.join([path1][, path2][, ...])** Join all the arguments together and normalize the resulting path.
3	**path.resolve([from ...], to)** Resolves to an absolute path.
4	**path.isAbsolute(path)** Determines whether the path is an absolute path. An absolute path will always resolve to the same location, regardless of the working directory.

9	**path.parse(pathString)** Returns an object from a path string.
10	**path.format(pathObject)** Returns a path string from an object, the opposite of **path.parse** above.

Properties

S. No.	Property & Description
1	**path.sep** The platform-specific file separator. '\\' or '/'.
2	**path.delimiter** The platform-specific path delimiter, ; or ':'.
3	**path.posix** Provide access to aforementioned path methods but always interact in a posix compatible way.

- **Modulo Net**
  Este modulo es usado para crear tanto clientes como servidores y provee un wrapper de red
  ```
  var net = require("net")
  ```

Methods

S. No.	Method & Description
1	**net.createServer([options][, connectionListener])** Creates a new TCP server. The connectionListener argument is automatically set as a listener for the 'connection' event.
2	**net.connect(options[, connectionListener])** A factory method, which returns a new 'net.Socket' and connects to the supplied address and port.
3	**net.createConnection(options[, connectionListener])** A factory method, which returns a new 'net.Socket' and connects to the supplied address and port.
4	**net.connect(port[, host][, connectListener])** Creates a TCP connection to port on host. If host is omitted, 'localhost' will be assumed. The connectListener parameter will be added as a listener for the 'connect' event. It is a factory method which returns a new 'net.Socket'.
5	**net.createConnection(port[, host][, connectListener])** Creates a TCP connection to port on host. If host is omitted, 'localhost' will be assumed. The connectListener parameter will be added as a listener for the 'connect' event. It is a factory method which returns a new 'net.Socket'.

- **Clase net.Server**
  Esta es usada para crear un servidor local o TCP

Methods

S.No.	Method & Description
1	**server.listen(port[, host][, backlog][, callback])** Begin accepting connections on the specified port and host. If the host is omitted, the server will accept connections directed to any IPv4 address (INADDR_ANY). A port value of zero will assign a random port.
2	**server.listen(path[, callback])** Start a local socket server listening for connections on the given path.
3	**server.listen(handle[, callback])** The handle object can be set to either a server or socket (anything with an underlying _handle member), or a {fd: <n>} object. It will cause the server to accept connections on the specified handle, but it is presumed that the file descriptor or handle has already been bound to a port or domain socket. Listening on a file descriptor is not supported on Windows.
4	**server.listen(options[, callback])** The port, host, and backlog properties of options, as well as the optional callback function, behave as they do on a call to server.listen(port,
	[host], [backlog], [callback]) . Alternatively, the path option can be used to specify a UNIX socket.
5	**server.close([callback])** Finally closed when all connections are ended and the server emits a 'close' event.

Events

S. No.	Events & Description
1	**listening** Emitted when the server has been bound after calling server.listen.
2	**connection** Emitted when a new connection is made. Socket object, the connection object is available to event handler. Socket is an instance of net.Socket.
3	**close** Emitted when the server closes. Note that if connections exist, this event is not emitted until all the connections are ended.
4	**error** Emitted when an error occurs. The 'close' event will be called directly following this event.

- **Clase net.Socket**
  Este es una abstracción de un socket local o TCP. El usuario puede crearlos y utilizarlos como cliente (con connect ()) o Node puede crearlos y transmitirlos al usuario a través del evento de "conexión" de un servidor.

S. No.	Events & Description
1	**lookup** Emitted after resolving the hostname but before connecting. Not applicable to UNIX sockets.
2	**connect** Emitted when a socket connection is successfully established.
3	**data** Emitted when data is received. The argument data will be a Buffer or String. Encoding of data is set by socket.setEncoding().
4	**end** Emitted when the other end of the socket sends a FIN packet.
5	**timeout** Emitted if the socket times out from inactivity. This is only to notify that the socket has been idle. The user must manually close the connection.

S. No.	Property & Description
1	**socket.bufferSize** This property shows the number of characters currently buffered to be written.
2	**socket.remoteAddress** The string representation of the remote IP address. For example, '74.125.127.100' or '2001:4860:a005::68'.
3	**socket.remoteFamily** The string representation of the remote IP family. 'IPv4' or 'IPv6'.
4	**socket.remotePort** The numeric representation of the remote port. For example, 80 or 21.
5	**socket.localAddress** The string representation of the local IP address the remote client is connecting on. For example, if you are listening on '0.0.0.0' and the client connects on '192.168.1.1', the value would be '192.168.1.1'.

Metodos:

S. No.	Method & Description
1	**new net.Socket([options])** Construct a new socket object.
2	**socket.connect(port[, host][, connectListener])** Opens the connection for a given socket. If port and host are given, then the socket will be opened as a TCP socket, if host is omitted, localhost will be assumed. If a path is given, the socket will be opened as a Unix socket to that path.
3	**socket.connect(path[, connectListener])** Opens the connection for a given socket. If port and host are given, then the socket will be opened as a TCP socket, if host is omitted, localhost will be assumed. If a path is given, the socket will be opened as a Unix socket to that path.
4	**socket.setEncoding([encoding])** Set the encoding for the socket as a Readable Stream.
5	**socket.write(data[, encoding][, callback])** Sends data on the socket. The second parameter specifies the

- **¿Qué es express?**
  Express.js es un framework de desarrollo de aplicaciones web minimalista y flexible. Está inspirado en Sinatra. Es robusto, rápido, flexible y muy simple, entre otras características, ofrece Router de URL (Get, Post, Put …), facilidades para motores de plantillas (Jade, EJS, JinJS …), Middeleware via Connect, etc.

- **¿Qué parámetros tiene la función que usa express en sus aplicaciones?**
  Request y Response

app.get('/', function (req, res) { // -- })

- **Objeto Request**
  El objeto req representa una petición HTTP y tiene propiedades, parámetros, cuerpo, cabecera HTTP, etc. Algunas de sus propiedades son:

S. No.	Properties & Description
1	**req.app** This property holds a reference to the instance of the express application that is using the middleware.
2	**req.baseUrl** The URL path on which a router instance was mounted.
3	**req.body** Contains key-value pairs of data submitted in the request body. By default, it is undefined, and is populated when you use body-parsing middleware such as **body-parser**
4	**req.cookies** When using cookie-parser middleware, this property is an object that contains cookies sent by the request.
5	**req.fresh** Indicates whether the request is "fresh." It is the opposite of req.stale.

Sus métodos:

req.accepts(types)

```
req.accepts(types)
```

req.get(field)

```
req.get(field)
```

req.is(type)

```
req.is(type)
```

req.param(name [, defaultValue])

```
req.param(name [, defaultValue])
```

- **Objeto Response**
  El objeto res representa una respuesta HTTP que una aplicación Express envía cuando recibe una petición HTTP.

S. No.	Properties & Description
1	**res.app** This property holds a reference to the instance of the express application that is using the middleware.
2	**res.headersSent** Boolean property that indicates if the app sent HTTP headers for the response.
3	**res.locals** An object that contains response local variables scoped to the request.

Sus metodos:

res.append(field [, value])

```
res.append(field [, value])
```

res.attachment([filename])

```
res.attachment([filename])
```

res.cookie(name, value [, options])

```
res.cookie(name, value [, options])
```

res.clearCookie(name [, options])

```
res.clearCookie(name [, options])
```

res.download(path [, filename] [, fn])

```
res.download(path [, filename] [, fn])
```

- **¿Qué es el enrutamiento?**

  se refiere a determinar cómo responde una aplicación a una solicitud de cliente a un punto final en particular, que es un URI (o ruta) y un método de solicitud HTTP específico (GET, POST, etc.).

- **¿Qué proporciona express?**

un middleware express.static incorporado para servir archivos estáticos, como imágenes, CSS, JavaScript, etc. Para ello, se indica la ruta al middleware express.static para comenzar a servir los archivos directamente.

- **Rest**

  es una arquitectura basada en el estándar web que utiliza el protocolo HTTP. Gira en torno a los recursos donde cada componente es un recurso y se accede a un recurso mediante una interfaz común utilizando los métodos estándar de HTTP

- **¿Qué proporciona un servidor Rest?**

  proporciona acceso a los recursos y un cliente REST accede y modifica los recursos mediante el protocolo HTTP. Aquí cada recurso es identificado por URIs / global IDs. REST usa varias representaciones para representar un recurso, por ejemplo, texto, JSON, XML, pero JSON es el más popular

- **¿Cuáles son los cuatro métodos HTTP se usan comúnmente en la arquitectura basada en REST?**
  - GET: se usa para proporcionar un acceso de solo lectura a un recurso.
  - PUT: esto se usa para crear un nuevo recurso.
  - DELETE: se usa para eliminar un recurso. •
  - POST: se usa para actualizar un recurso existente o crear un nuevo recurso.
- **RESTful Web services**

  Un servicio web es una colección de protocolos y estándares abiertos utilizados para intercambiar datos entre aplicaciones o sistemas. Un servicio web RESTful generalmente define un URI, Identificador uniforme de recursos, que proporciona representación de recursos, como JSON y un conjunto de métodos HTTP.

- **Node.js se ejecuta en un modo de un solo hilo, pero además qué utiliza**

  un paradigma basado en eventos para manejar la concurrencia. También facilita la creación de procesos secundarios para aprovechar el procesamiento paralelo en sistemas basados en CPU de múltiples núcleos.

- **Secuencias de procesos secundarios:**

  child.stdin, child.stdout y child.stderr que pueden compartirse con las secuencias de stdio del proceso principal.

- **Node proporciona el módulo child_process que tiene las siguientes tres formas principales de crear un proceso secundario:**

- exec - child_process.exec método ejecuta un comando en un shell / consola y manda por un buffer la salida.
- spawn - child_process.spawn inicia un nuevo proceso con un comando dado.
- fork: el método child_process.fork es un caso especial de spawn () para crear procesos secundarios.

El método exec()

child_process.exec(command[, options], callback)

Parameters

Here is the description of the parameters used:

- **command** (String) The command to run, with space-separated arguments
- **options** (Object) may comprise one or more of the following options:
  - **cwd** (String) Current working directory of the child process
  - **env** (Object) Environment key-value pairs
  - **encoding** (String) Default: 'utf8'
  - **shell** (String) Shell to execute the command with. Default: '/bin/sh' on UNIX, 'cmd.exe' on Windows. The shell should understand the -c switch on UNIX or /s /c on Windows. On Windows, command line parsing should be compatible with cmd.exe.
  - **timeout** (Number) Default: 0
  - **maxBuffer** (Number) Default: 200*1024
  - **killSignal** (String) Default: 'SIGTERM'
  - **uid** (Number) Sets the user identity of the process.
  - **gid** (Number) Sets the group identity of the process.
- **callback** The function gets three arguments **error**, **stdout**, and **stderr** which are called with the output when the process terminates.

The exec() method returns a buffer with a max size and waits for the process to end and tries to return all the buffered data at once.

El método spawn

child_process.spawn(command[, args][, options])

Parameters

Here is the description of the parameters used:

- **command** (String) The command to run.
- **args** (Array) List of string arguments.
- **options** (Object) It may comprise one or more of the following options:
  - **cwd** (String) Current working directory of the child process.
  - **env** (Object) Environment key-value pairs.
  - **stdio** (Array|String) Child's stdio configuration.
  - **customFds** (Array) Deprecated File descriptors for the child to use for stdio.
  - **detached** (Boolean) The child will be a process group leader.
  - **uid** (Number) Sets the user identity of the process.
  - **gid** (Number) Sets the group identity of the process.

The spawn() method returns streams (stdout & stderr) and it should be used when the process returns a large volume of data. spawn() starts receiving the response as soon as the process starts executing.

El método fork()

child_process.fork(modulePath[, args][, options])

Parameters

Here is the description of the parameters used:

- **modulePath** (String) The module to run in the child.
- **args** (Array) List of string arguments.
- **options** (Object) It may comprise one or more of the following options:
  - **cwd** (String) Current working directory of the child process.
  - **env** (Object) Environment key-value pairs.
  - **execPath** (String) Executable used to create the child process.
  - **execArgv** (Array) List of string arguments passed to the executable (Default: process.execArgv)
  - **silent** (Boolean) If true, stdin, stdout, and stderr of the child will be piped to the parent, otherwise they will be inherited from the parent. See the "pipe" and "inherit" options for spawn()'s stdio for more detail on this. Default is false.
  - **uid** (Number) Sets the user identity of the process.
  - **gid** (Number) Sets the group identity of the process.

The fork method returns an object with a built-in communication channel in addition to having all the methods in a normal ChildProcess instance.