José Montoya Guzmán

- Tech Lead Front End en BBVA.
- Git Evangelist.
- Experiencia en Java y SQL.
- Sensei por hobby.
- Hp / pokemón Lover.

https://github.com/montoyaguzman

montoyaguzman ó montoyaguzman7

DEV.F::
DESARROLLAMOS(PERSONAS);

Daniel Gloria Florencio

- Software Developer en Softtek
- CIO en TaquitoSoftware S.A.S. de C.V.
- Scrum Evangelist.
- Consultor de software.
- Sensei para pagar mis deudas.
- Me encanta el exceso.

https://github.com/dannielgloria

https://www.instagram.com/danniel.gloria

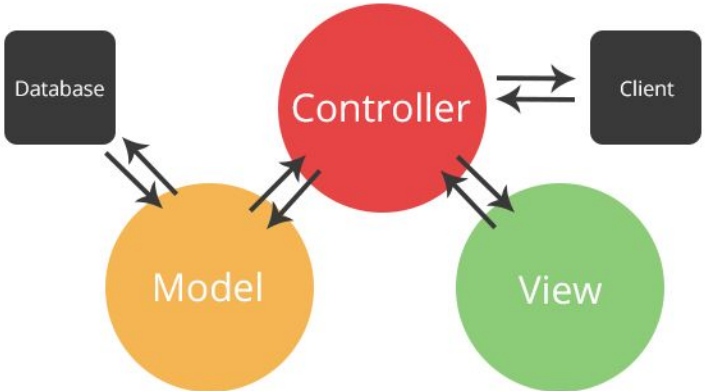
DEV.F::
DESARROLLAMOS(PERSONAS);

KATA
JavaScript Avanzado

DEV.F::
DESARROLLAMOS(PERSONAS);

¿Qué veremos en está kata?

- Node.
- Npm y paquetes.
- Node para front y back end.
- Arquitectura de software.
- Stacks de desarrollo web.
- Asincronía y Event Loop.
- Comparativa entre Node vs JavaScript.
- Conceptos de una API.
- Qué es REST y sus principios.
- API Rest.
- ExpressJS.
- Deploys.



Semana 1

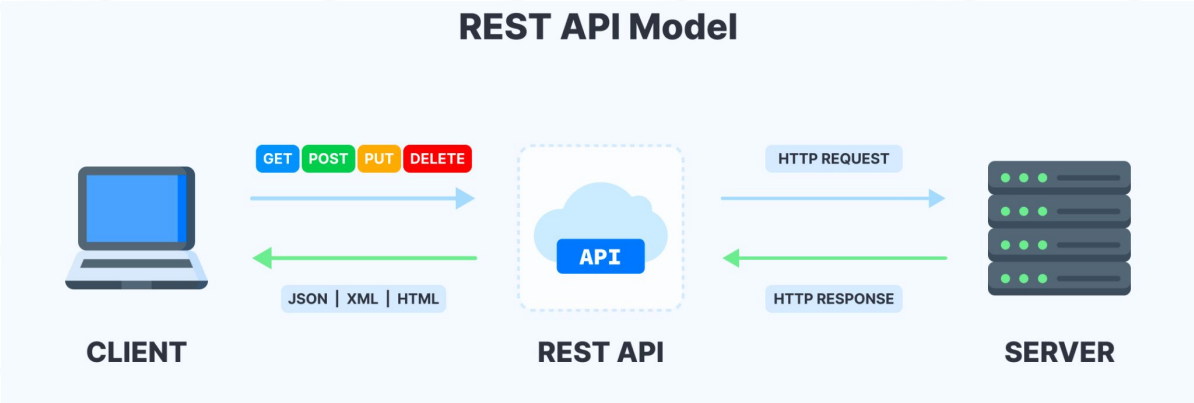
- Node.
- Npm y paquetes.
- Node para front y back end.
- Frontend.
- Backend.
- Fullstack.

Semana 2

- Arquitectura de software.
- Stacks de desarrollo web
- Asincronía y Event Loop.
- Comparativa entre Node vs JavaScript.

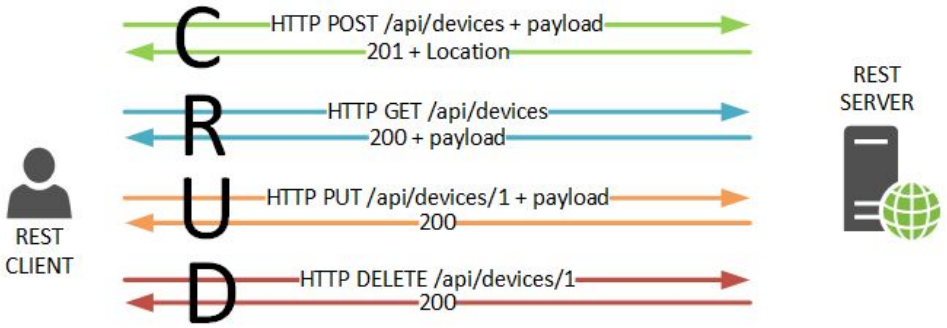
Semana 3

- HTTP/HTTPS
- Conceptos de una API.
- Qué es REST y sus principios.
- API Rest.
- API Restful.
- ExpressJS.
- Deploys.



Semana 4

- Construir una API Rest que se conecte con un fake back end, y realice las 4 operaciones básicas de un CRUD con los 5 verbos http.



REPASO

DEV.F::
DESARROLLAMOS(PERSONAS);

Repaso

- Terminal.
- Git.
- Sentencias condicionales y ciclos.
- Funciones, parámetros.
- Prototype.
- POO.
- ES6.

Node

DEV.F::
DESARROLLAMOS(PERSONAS);

Node

- Mayo 2009 (Ryan Lenhiart).
- Entorno en tiempo de ejecución **multiplataforma** para la **capa del servidor (no se limita a ello).**
- Basado en el motor V8 de google.
- Escrito en C++.
- Basado en módulos.
- Es **asíncrono** y trabaja con base en un **bucle de eventos**.

¿Qué puedo hacer con Node?

- **Realizar API Rest**.
- Acceder a bases de datos relacionales y no relacionales.
- **Generar páginas dinámicas en un servidor web. => server side render**
- Crear, leer y escribir archivos.
- Procesar y almacenar archivos enviados desde una página web.
- Recuperar datos de formularios HTML.
- Acceder a funciones del sistema operativo y/o hardware.

JS vs Node

- Lenguaje de scripting vs Entorno de ejecución.
- Acceso al browser vs Acceso al servidor
- Motor del navegador vs V8.
- Js interactúa directamente con el DOM.
- Navegadores usan Libevent.
- Node usa Libuv.
- Ambos se basan en 0 retraso.
- No hay solicitudes Ajax (consumo de API’s).
- No hay temporizador (setTimeout y setInterval).

Práctica 1

- Instalación de node y npm.
- Validación en la CLI de Node.
  - node -v
  - npm -v
- Uso de la documentación oficial.

Práctica 2

- Objeto Global (this).
- Uso de las funciones base de node:
  - global.
  - os.
  - file.
  - path
  - http.

Práctica 3

- Crear un archivo de texto con node.

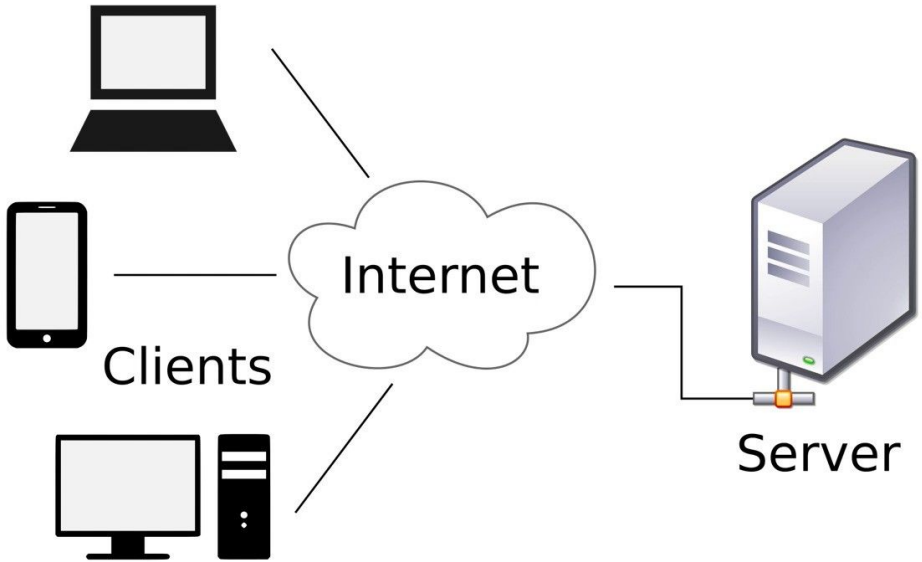
Práctica 4

- Primer servidor en node.

Práctica 5

- Crear un servidor que responda páginas web.

Intro al modelo cliente servidor



Intro al concepto de URL

Uniform resource identifier, sirve para acceder a un recurso físico o abstracto por Internet. A continuación se muestran los elementos mínimos de una URL: **protocolo://dominio/path?queryParamsEjemplo:**

https://www.tutorialesprogramacionya.com/javascriptya/nodejsya/detalleconcepto.php?punto=1&codigo=1&inicio=0

Mime Types

Los mime types indican la naturaleza y formato de los archivos que son transmitidos en la respuesta de una solicitud web.

- Listado.

Localhost

Es la dirección de mi propio computador, también se le conoce como dirección loopback.

Localhost: http://localhost:8080

Loopback: http://127.0.0.1:8080

Módulos

DEV.F:
DESARROLLAMOS(PERSONAS);

Módulos

Permiten aislar parte de nuestro código en diferentes archivos y mandarlos llamar sólo cuándo los necesitamos. Existen dos formas de utilizar módulos en node:

- Common JS.
- ES6 Imports (.mjs, --experimental-modules y “type”: “module” en package.json).

Práctica 6

- Crear una calculadora en node, utilizando los import common y los es6 import.

Npm

DEV.F::
DESARROLLAMOS(PERSONAS);

Npm

Node Package Manager o manejador de paquetes de node, es la herramienta más popular de JavaScript para compartir e instalar paquetes. Se compone de 2 partes:

- **Un repositorio online para publicar paquetes** de software libre para ser utilizados en proyectos Node.js
- **Una herramienta para la terminal (CLI)** para interactuar con dicho repositorio que ayuda a la instalación de utilidades, manejo de dependencias y la publicación de paquetes.

NOTA: Se puede considerar un gestor de dependencias de proyectos de tipo npm.

Comandos de Npm

Inicialización y arranque

- **npm init:** Inicializa una carpeta como un proyecto de npm.
- **npm start:** Es el único comando por defecto para iniciar un proyecto de node.

Comandos de Npm

Instalar/desinstalar dependencias

Dependencia: Son los recursos o librerías externas que utilizan los proyectos para funcionar.

- npm i
- npm install -g <pakage-name>
- npm install <package-name>
- npm install –save <pakage-name>
- npm install -D <package-name>
- npm uninstall <package-name>
- npm uninstall -g <package-name>

NOTA: install = i

Comandos de Npm

Gestión de dependencias

- npm search <package-name>
- npm ls
- npm update -save
- npm list
- npm list -g –depth 0
- npm outdated

Paquetes

Son módulos distribuidos en forma de librerías que resuelven alguna necesidad de desarrollo. A continuación se listan los más populares al 2022:

- npm.
- create-react-app.
- vue-cli.
- grunt-cli.
- mocha.
- react-native-cli.
- gatsby-cli.
- forever.

Scripts

Son comandos propios que se pueden agregar al package.json para poderlos ejecutar con **npm run <my-comand>.**

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "node index.js",
  "dev": "nodemon index.js"
},
```

Estructura de proyecto npm

- **node_modules:** Carpeta donde se instalan las dependencias de un proyecto npm, normalmente está carpeta se agrega al .gitignore.
- **package.json:** Guardan las dependencias y los comandos de node.
- **package-lock.json:** Guarda un snapshot de las dependencias que se instalaron en un determinado momento.

Detalle del package.json

Este archivo guarda las dependencias y los comandos de node.

- name.
- version.
- description.
- license.
- scripts.
- **devDependencies:** Son dependencias que sólo se instalan en el entorno local.
- **dependencies:** Son dependencias que se instalan en cualquier entorno (local, test, qa, prepro y pro).

Detalle del package-lock.json

- Este archivo tiene las versiones exactas de las dependencias utilizadas por un proyecto npm.
- No está pensado para ser leído línea por línea por los desarrolladores.
- Es usualmente generado por el comando **npm install.**

Práctica 7

- Servidor de archivos multimedia con node.

Práctica 8

- Definir los siguientes conceptos:
  - Entorno de ejecución
  - Manejador de paquetes.
  - Diferencia entre node y npm.
  - CLI, comando, dependencia, gestor de dependencia, dependencia de desarrollo y script.
- Explicar en una oración o diagrama como funciona la comunicación en internet.

Práctica 9

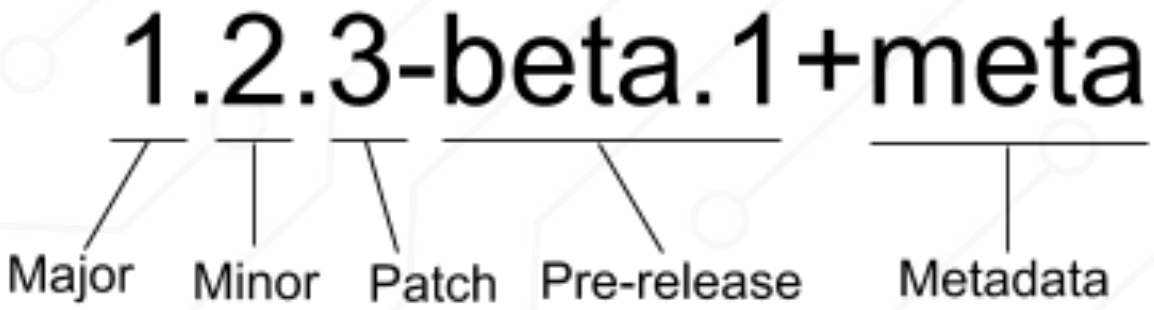
- Desarrollar un proyecto node que regrese plantillas dinámicas (server side render).

Semantic Versioning

DEV.F::
DESARROLLAMOS(PERSONAS);

Semantic Versión

Es un conjunto simple de reglas y requerimientos que dictan cómo asignar e incrementar los números de la versión de un software. Evitan la pérdida de versiones y mejoran la gestión de dependencias.



Funcionamiento de semantic versión

Dado un número de versión **MAYOR.MENOR.PARCHE**, se incrementa:

- La versión **MAYOR** cuando realizas un cambio incompatible en el proyecto.
- La versión **MENOR** cuando añades funcionalidad que compatible con versiones anteriores.
- La versión **PARCHE** cuando reparas errores compatibles con versiones anteriores.

MAYOR.MENOR.PARCHE = MAJOR.MINOR.PATCH.

Ejemplo: 1.2.1