UNIVERSIDAD DE
COSTA RICA

IF 3000 PROGRAMACIÓN II

Clase16: Introducción al modelo Cliente-Servidor

- Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación JavaScript, asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google.

- npm (node package manager) es el gestor de paquetes o módulos sobre el entorno de ejecución node JS

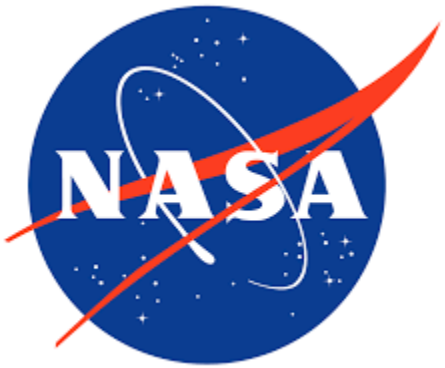
Live-server

- para que los cambios al salvar se muestren automáticamente (para los archivos .html) en el navegador.

Nodemon

- permitirá como live-server actualizar automáticamente los cambios, pero en este caso es para los archivos .js

Sitios web que usan en Node JS



The best IDEs for JavaScript Development in 2022

- Visual Studio Code
- Atom
- Webstrom
- IntelliJ IDEA
- Sublime Text
- Brackets
- ActiveState Komodo IDE
- Apache NetBeans
- Eclipse

REST API

- **RE**presentational **S**tate **T**ransfer, es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP.
- Fue definida en el 2000 por Roy Fielding, uno de los padres de la especificación HTTP y un referente en la arquitectura de redes.
- Hoy por hoy, la mayoría de las aplicaciones que se desarrollan disponen de una API REST para el intercambio de información entre el front y el back.
- Lo que la hace tan potente es precisamente el aislamiento que proporciona entre la lógica del back-end y cualquier cliente consumidor de éste. Esto le permite ser usada por cualquier tipo de cliente: web, móvil, etc.

REST API

- Cuando el cliente envía una solicitud a través de una API de RESTful, esta transfiere una representación del estado del recurso requerido a quien lo haya solicitado o al extremo.
- La información se entrega por medio de HTTP en uno de estos formatos: JSON (JavaScript Object Notation), HTML, XLT, Python, PHP o texto sin formato.
- JSON es el más popular, ya que tanto las máquinas como las personas lo pueden comprender y no depende de ningún lenguaje, a pesar de que su nombre indique lo contrario.

Uso de la especificación HTTP

- **POST**: crear un recurso nuevo.
- **PUT**: modificar un recurso existente.
- **GET**: consultar información de un recurso.
- **DELETE**: eliminar un recurso determinado.
- **PATCH**: modificar solamente un atributo de un recurso.

Posibles respuestas de una API REST.

- **200 OK.** Respuesta estándar para peticiones correctas.
- **201 Created.** La petición ha sido completada y ha resultado en la creación de un nuevo recurso.
- **202 Accepted.** La petición ha sido aceptada para procesamiento, pero este no ha sido completado.
- **400 Bad Request.** La solicitud contiene sintaxis errónea.
- **403 Forbidden.** La solicitud fue legal, pero el servidor rehúsa responder dado que el cliente no tiene los privilegios para hacerla.
- **404 Not Found.** Recurso no encontrado. Se utiliza cuando el servidor web no encuentra la página o recurso solicitado.
- **500 Internal Server Error.** Es un código comúnmente emitido por aplicaciones empotradas en servidores web, cuando se encuentran con situaciones de error ajenas a la naturaleza del servidor web.

Módulos de Node Js: express

- Express.js, o simplemente Express, es un marco de aplicación web de back-end para Node.js, lanzado como software gratuito y de código abierto bajo la licencia MIT.
- Está diseñado para crear aplicaciones web y API. Se le ha llamado el marco de servidor estándar de facto para Node.js.

Instalación de express

1. Crear una carpeta de trabajo:

```
md myApp
cd myApp
```

2. Crear de un archivo **package.json** para nuestra aplicación.

```
npm init -y
```

3. Instalar Express en el directorio myapp y guardarlo en la lista de dependencias

```
npm install express --save
```

Express-generator

- Express-generator es una herramienta generadora de aplicaciones para crear rápidamente un esqueleto de aplicación. A partir de Node.js 8.2.0 es posible ejecutar el generador de aplicaciones con el comando npx:

```
npx express-generator
```

Para redireccionar el archivo de inicio en el archivo de configuración:

```
$env:DEBUG='myapp:*'; npm start
```

```
.
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.pug
    ├── index.pug
    └── layout.pug
```

"Hola mundo" con express

```
//'require' es una instrucción que usaremos para determinar la necesidad de importar algo.
//Debemos crear este tipo de instancias en la parte superior del archivo.
const express = require('express')
//Estamos creando la aplicación rápida configurándola como variable de aplicación.
const app = express()
//Parámetros de conexión
const hostname = '127.0.0.1'
const port = 3000
//'.get' está diciendo que cuando obtiene esa ruta debe dar la respuesta que se especifica en
la función.
//Toma 2 argumentos: (1) la url (2) la función que le dice a express qué enviar a la persona
que hace la solicitud.
app.get('/', (req, res) => {
    res.send('Hello World!')
})
//'.listen' vinculará la aplicación al puerto de escucha de nuestra máquina.
app.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`)
})
```

- MongoDB (del inglés humongous, "enorme") es un sistema de base de datos NoSQL, orientado a documentos y de código abierto.
- En lugar de guardar los datos en tablas, tal y como se hace en las bases de datos relacionales, MongoDB guarda estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

```
npm install mongodb
```

```
https://www.mongodb.com/try/download/community
```

mongoDB.

COMPASS

- Compass nos permite explorar la estructura de los documentos de las distintas colecciones que componen una base de datos de una manera fácil e intuitiva. Además, permite realizar algunas queries de manera visual a base de clics de ratón sin necesidad de saber la sintaxis del CRUD de mongoDB.
- Es el entorno predeterminado para mongoDB por lo que se instala automáticamente al instalar el Servidor de Bases de datos de mongoDB.

mongoose

elegant mongodb object modeling for node.js

- Mongoose es una biblioteca de programación orientada a objetos de JavaScript que crea una conexión entre MongoDB y el marco de la aplicación web Express.

- `npm install mongoose`

Caso de estudio

- Vamos a crear una REST API en node JS por medio de un servidor express y una gestor de bases de datos en MongoDb accesado por medio de mongoose. (Back end)
- Se trata de un servidor de base de datos sobre estudiantes donde vamos a almacenar el carné, nombre y edad, para luego hacer un CRUD.
- Esquema JSON `{carne: String, nombre: String, edad: Number}`
- Vamos a crear una Interfaz cliente implementando Bootstrap para manipular la base de datos del lado del cliente. (Front end)

Detallamos el Node js Rest API

Route	Method	Type	Posted JSON	Description
/est	GET	JSON	–	Obtener todos los estudiantes
/est/{carne}	GET	JSON	–	Obtener un estudiante por medio de su carné
/est	POST	JSON	{"carne": "C10101", "nombre": "Juan", "edad" : 19}	Agregar un estudiante a la base de datos
/est	PUT	JSON	{"carne": "C10101", "nombre": "Juan", "edad" : 19}	Modificar un estudiante en la base de datos
/est	DELETE	JSON	{"carne": "C10101"}	Eliminar un estudiante de la base de datos
/est	PATCH	JSON	{"edad" : 19}	Modificar un atributo de un estudiante

Estructura del proyecto

Elemento	Funcionalidad
package.json	Este archivo tendrá todas las dependencias de nodejs.
config/database.js	Este archivo será usado para la conexión de la base de datos.
model/estudiante.js	Este archivo será usado para crear el esquema y modelo de un estudiante.
routes/users.js	Este archivo será usado para crear las rutas del API REST.
main.js	Este archivo será usado para crear la aplicación servidor.
node_modules folder	Este archivo contendrá todos los paquetes de node js.

Paso1: Crear el archivo database.js (conexión a mongoDB)

```
// config/database.js
// revisar el puerto de conexión en Compass
// se exporta para que pueda ser usado en otro archivo
module.exports = {
url:'mongodb://127.0.0.1:27017/UniversidadDB'}
```

Paso2: Crear el archivo estudiante.js (esquema de la BD)

```
// model/estudiante.js
const mongoose = require('mongoose')
EstSchema = new mongoose.Schema({
   _id:Number,
   carne : String,
   nombre: String,
   edad: Number
}, {
   versionKey: false
})
module.exports = mongoose.model(‘Estudiante', EstSchema, ‘Estudiante')
.
```

Paso 3: Agregar las dependencias en el archivo principal

```
// app.js


const mongoose = require('mongoose');

const database = require('./config/database');
```

Paso 4: Crear la conexión con la base de datos

```
// app.js

mongoose.set('strictQuery', true);
mongoose.connect(database.url)
.then(()=>console.log('MongoDb connected'))
.catch(()=>console.log("Connection Error"))
```

Paso 5: Agregar un par de funciones middleware

```
let cors = require('cors')
app.use(cors({ origin: true, credentials: true  }))


app.use('/est',indexRoutes)//para agregar la ruta para el API REST
app.use(express.json())//para poder reconocer los formatos de intercambio en JSON
```

En resumen: archivo app.js

```
//solicitar dependencias
const express = require('express')
const mongoose = require('mongoose')
const database = require('./config/database')
const estRoutes = require('./routes/users')
//conexión con la base de datos
mongoose.connect(database.url)
   .then(() => console.log('MongoDb connected'))
   .catch(() => console.log('Connection Error'))
//servidor web
const app = express()
const hostname = '127.0.0.1'
const port = 3000
//funciones middleware
app.use(express.json())//para poder reconocer los formatos de intercambio en JSON
app.use('/est', estRoutes)//para agregar la ruta para el API REST
//iniciar el servidor
app.listen(port, hostname, () => {
   console.log(`Server running at http://${hostname}:${port}/`)
})
```

users.js REST (GET) todos los elementos

```
//requiere el esquema de la base de datos para el objeto estudiante
const Estudiante = require('../model/estudiante');

// GET todos los elementos de la base de datos.
router.get('/', (req, res) => {
  Estudiante
    .find()
    .then((data) => res.json(data))
    .catch((error) => res.json({ message: error }));
});
```

funciones.js

```
function obtenerTodos() {
   const xhr = new XMLHttpRequest();
   const divRespuesta = document.getElementById('response');
   xhr.addEventListener('load', () => {
      if (xhr.status >= 200 && xhr.status <= 299) {
         const respuesta = xhr.response;
         respuesta.forEach(e => {
            let user = document.createElement("p");
            user.innerHTML = `<h3>${e._id}   ${e.carne}   ${e.nombre}   ${e.edad}</h3>`;
            divRespuesta.appendChild(user);
         });
      } else {
         document.getElementById('responseText').innerHTML =
            `Error: ${xhr.status}, el recurso no se ha encontrado.`;
      }
   });
   xhr.open('GET', 'http://127.0.0.1:3000/est');
   xhr.responseType = 'json';
   xhr.send();
}
```

Index.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <script src="javascripts/funciones.js">
   </script>
</head>
<body>
   <h1>Ajax - Objeto XMLHttpRequest</h1>
   <input type="button" onclick="obtenerTodos()" value="Consultar">
   <h2>Lista de estudiantes</h2>
    <h3>Id   Carné   Nombre   Edad</h3>
   <hr>
   <div id='response'></div>
</body>
</html>
```