Node.js

Universidad Autónoma de Coahuila

Ing. Urbano de J. Flores Zaragoza

Que es node

- Es un ambiente para ejecutar código JS
- En escencia es un programa en C++, se puede encontrar dentro del engine de Chome v8
- Es uno de los más rápidos
- Se puede utilizar para construir aplicaciones de redes rápidas y escalables
- Lo utilizaremos para construir servicios REST

Node

- Las aplicaciones node son de un solo hilo (single-threaded), significa que se usa un solo hilo para atender a todos los clientes
- Las aplicaciones son asíncronas y sin bloqueo por default, cuando en la aplicación se involucran operaciones de entrada y salida (por ejemplo acceso a archivos o redes) el thread no espera (o bloquea) por el resultado de la operación. Se libera para servir a otros clientes.

Instalar node

- https://nodejs.org/es/

app.js

```
function saludar(nombre) {
    console.log('Hola ' + nombre);
}

saludar('Pumas');
```

Sistema de módulos en node

- os
- fs
- events
- http

El objeto global

- Algunas funciones de global
  - console
  - setTimeout()
  - clearTimeout()
  - setInterval()
  - clearInterval()

Creación de módulos

```
var url = 'http://mylogger.io/log';
function log(mensaje){
    console.log(mensaje);
}

module.exports.log = log;
module.exports.endPoint = url;
```

Uso de un módulo

```
const logger = require('./logger');

function diHola(nombre) {
    logger.log('Hola '+ nombre);
}

diHola('Pumas');
```

Módulo http

- Es uno de los módulos mas útiles
- Permite crear servidores de http
- Permite realizar llamadas a servicios por medio de http

Ejemplo de servidor http

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

const server = http.createServer((req, res) => {
    if(req.url === '/'){
        res.write('Hola');
        res.end();
    }
    if(req.url === '/api/lista'){
        res.write(JSON.stringify([1,2,3]));
        res.end();
    }
});
server.listen(3000);    // puerto 3000
console.log('Escuchando en el puerto 3000...');
```

Node Package Manager (npm)

- Permite la administración de modulos (https://www.npmjs.com/)
  - publicación
  - instalación
    - npm init –yes
    - npm i underscore --save

Uso de un paquete

```
var _ = require('underscore');
var resultado =_.contains([1,2,3], 4);
console.log(resultado);
```

mongoose

- Paquete para manipular el acceso a mongoDB
  - npm i mongoose --save

Servicios REST

- Es una interfaz entre sistemas que usa HTTP para obtener datos o aplicar operaciones sobre estos datos
- Puede utilizarse XML y JSON
- Es una alternativa a SOAP
- Clientes/Servidor sin estado, cada petición contiene toda la información necesaria para ejecutarse

Servicios REST

- Métodos
  - POST - crear
  - GET - consultar
  - PUT - editar
  - DELETE - eliminar

GET

Request

```
GET /api/customers
```

Response

```
[
    { id: 1, name: '' },
    { id: 2, name: '' },
    ...
]
```

PUT

Request

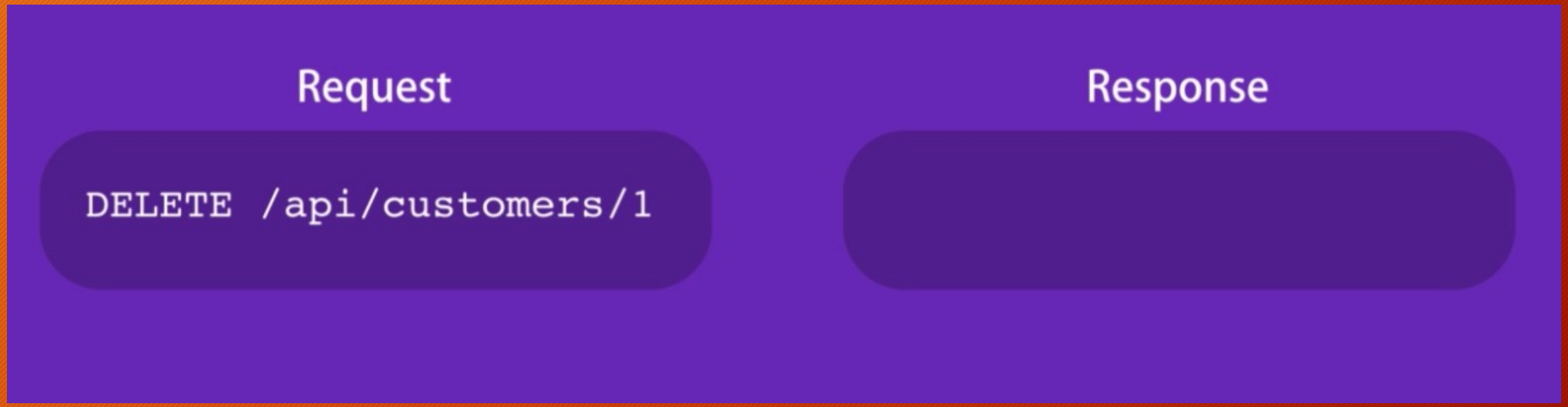
```
PUT /api/customers/1

{ name: '' }
```

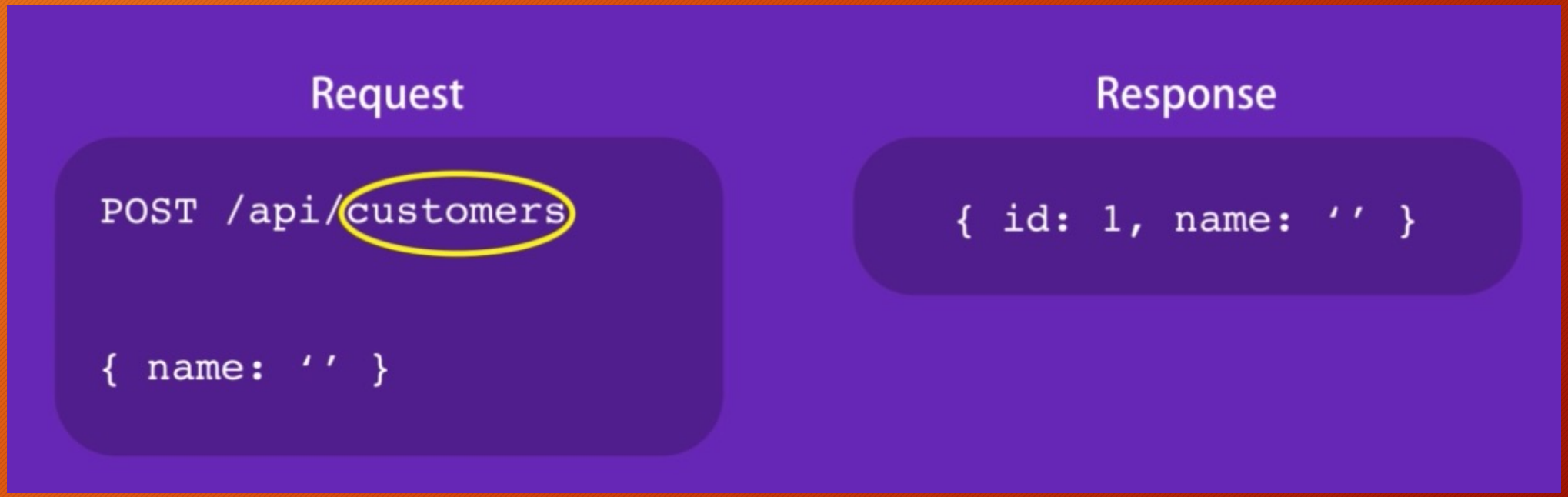
Response

```
{ id: 1, name: '' }
```

DELETE



POST



Express

- Es un paquete que nos permite crear servicios REST de una manera sencilla
- instalación
  - npm i express --save

Ejemplo de GET

```
const express = require('express');
const app = express();

app.get('/', (req,res) =>{
    res.send('Hola');
});

app.get('/api/lista', (req,res) =>{
    res.send([1,2,3]);
});

app.listen(3000, ()=> console.log('Escuchando en 3000'));
```

nodemon

- Permite ejecutar un script de forma continua
- npm i –g nodemon

Parámetros

```
const express = require('express');
const app = express();
app.get('/api/articulos/:anio/:mes', (req,res) =>{

    res.send(req.params.anio + '/' + req.params.mes);
});
app.listen(3000, ()=> console.log('Escuchando en 3000'));
```

Query parameters

```
const express = require('express');
const app = express();

app.get('/api/articulos/:anio/:mes', (req,res) =>{
    res.send(req.query);
});
app.listen(3000, ()=> console.log('Escuchando en 3000'));
```

Manejo de request (GET)

JS index.js > ...

```
1   const express = require('express');
2   const app = express();
3   const materias = [
4       {id: 1, nombre: 'PROGRAMACION I'}, {id: 2, nombre: 'PROGRAMACION II'},
5       {id: 3, nombre: 'MATEMATICAS I'},  {id: 4, nombre: 'LOGICA'}
6   ];
7
8   app.get('/api/materias', (req,res) =>{
9       res.send(materias);
10  });
11  app.get('/api/materias/:clave', (req,res) =>{
12      var materia = materias.find(c=>c.id === parseInt(req.params.clave));
13      if(!materia) res.status(404).send('Materia no encontrada');
14      else res.send(materia);
15  });
16
17  app.listen(3000, ()=> console.log('Escuchando en 3000'));
```

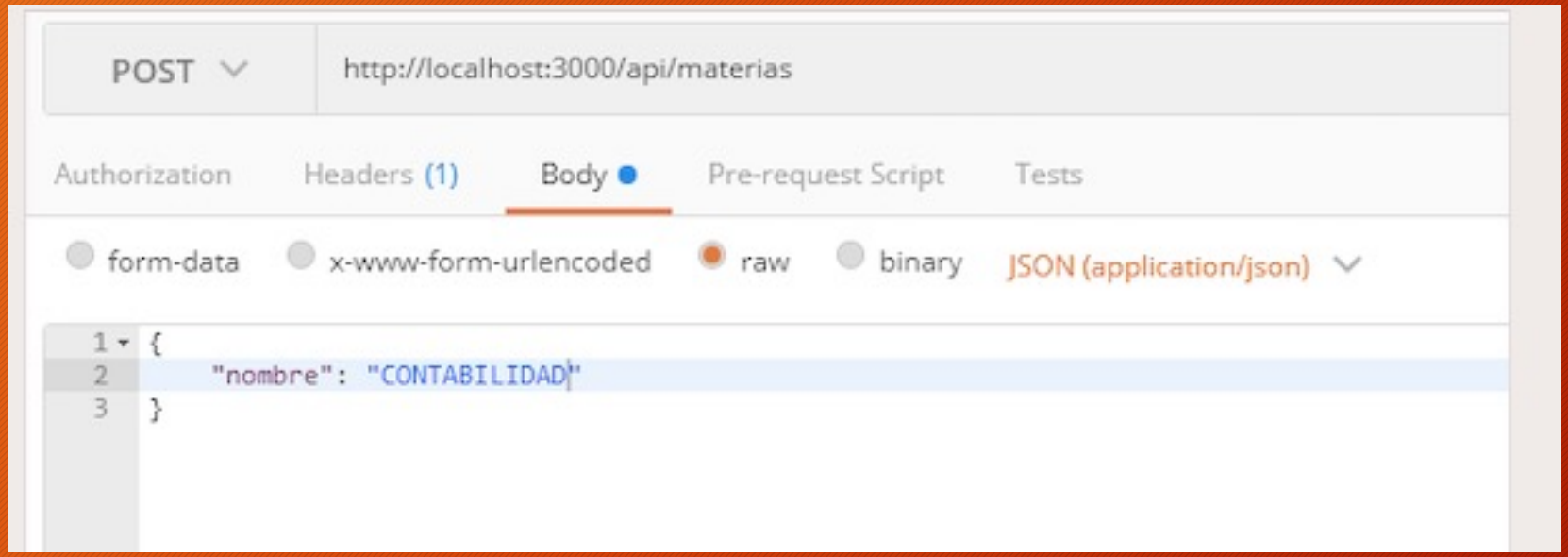
Manejo de request (POST)

```
idex.js > app.post('/api/materias/') callback
const express = require('express');
const app = express();
app.use(express.json());
const materias = [
    {id: 1, nombre: 'PROGRAMACION I'}, {id: 2, nombre: 'PROGRAMACION II'},
    {id: 3, nombre: 'MATEMATICAS I'},  {id: 4, nombre: 'LOGICA'}
];
app.post('/api/materias/', (req, res)=>{
    const materia = {
        id: materias.length +1,
        nombre: req.body.nombre
    };
    materias.push(materia);
    res.send(materia);

});
```

Probar POST

- Instalar postman en chrome



Validación

```
app.post('/api/materias/', (req, res)=>{
    if(!req.body.nombre || req.body.nombre.length<3){
        res.status(400).send('El nombre es requerido y debe ser de minimo 3 caracteres');
        return;
    }

    const materia = {
        id: materias.length +1,
        nombre: req.body.nombre
    };
    materias.push(materia);
    res.send(materia);

}
);
```

Manejo de request (PUT)

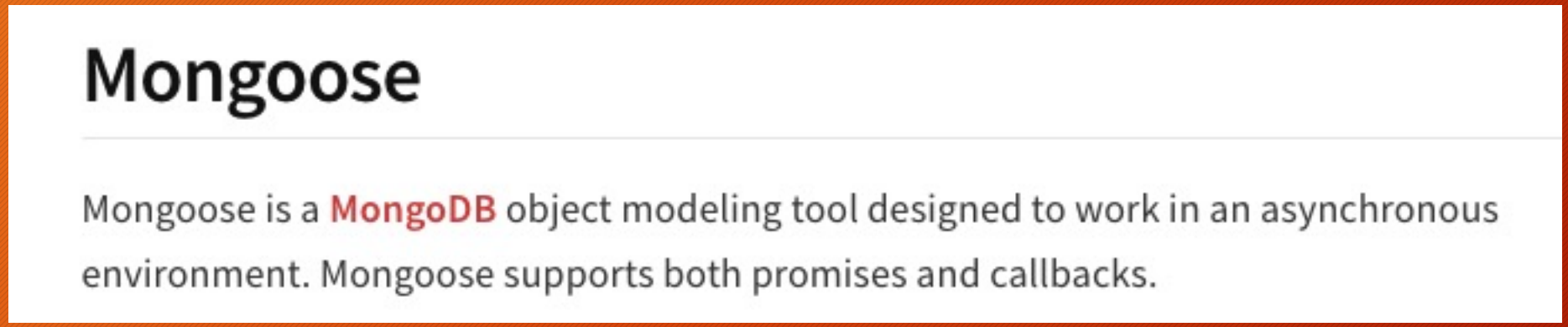
```
app.put('/api/materias/:id', (req, res)=>{
    var materia = materias.find(c=>c.id === parseInt(req.params.id));
    if(!materia) res.status(404).send('Materia no encontrada');
    if(!req.body.nombre || req.body.nombre.length<3)
        return res.status(400).send('El nombre es requerido y debe ser de minimo 3 caracteres');
    materia.nombre = req.body.nombre;
    res.send(materia);
});
```

Manejo de request (DELETE)

```
app.delete('/api/materias/:id', (req, res)=>{
    var materia = materias.find(c=>c.id === parseInt(req.params.id));
    if(!materia) res.status(404).send('Materia no encontrada');
    const index = materias.indexOf(materia);
    materias.splice(index,1);
    res.send(materia);
});
```

Conectar a mongoDB

- Mogoose



- npm i mongoose --save

Conectar a mongoDB

JS index.js > catch() callback

```
1   const mongoose = require('mongoose');
2
3   mongoose.connect('mongodb://127.0.0.1:27017/Personas')
4       .then(()=> console.log('Conectado a mongoDB'))
5       .catch(err => console.error('No se puede conectar', err));
6
```

- then() siginifica que connect regresa una promesa, una promesa representa una operación que no ha sido completada aún, se dá en llamadas asíncronas.

Esquemas

```
6
7   const personaSchema = new mongoose.Schema({
8       nombre: String,
9       apellido: String,
10      edad: Number
11  });
```

Model

- Un modelo es una esquema compilado en una clase a partir de la cual podemos crear instancias

```
const Persona = mongoose.model('Persona', personaSchema);

const persona = new Persona({
    nombre: "Ernesto",
    apellido: "Rodriguez",
    edad: 25
});
```

Guardado de un documento

- El método save() regresa una promesa, por lo cual es asincrona, lo encapsulamos en una función y hacemos el llamado con await

```
const Persona = mongoose.model('Persona', personaSchema);

async function crearPersona() {

    const persona = new Persona({
        nombre: "Ernesto",
        apellido: "Rodriguez",
        edad: 25
    });

    const resultado = await persona.save();
    console.log(resultado);
}

crearPersona();
```

Consulta de documentos

```
async function getPersonas(){
    const personas = await Persona.find();
    console.log(personas);
}

getPersonas();
```

Consulta de documentos

```
async function getPersonas(){
    const personas = await Persona.find({
        apellido: 'Flores', edad: 25
    })
    .limit(5)
    .sort({nombre:1})
    .select({nombre:1})
    ;
    console.log(personas);
}
```

Actualizar documentos

- Encontrando primero el documento

```
async function updatePersona(id) {
    const persona = await Persona.findById(id);
    if(!persona) return;
    persona.nombre = 'JUAN';
    const resultado =await persona.save();
    console.log(resultado);
}

updatePersona('60c3a18e16f00c0b60c3fa8b');
```

Actualizar documentos

- Actualización directa

```
async function updatePersona(id) {
    const resultado = await Persona.updateOne({_id: id},{
        $set: {
            nombre: 'Armando'
        }
    });
   console.log(resultado);
}
```

Actualizar documentos

- Alternativa

```
async function updatePersona(id) {
    const persona = await Persona.findByIdAndUpdate(id,{
        $set: {
            nombre: 'Rodrigo'
        }
    });
   console.log(persona);
}
```

- observar que persona tiene los valores anteriores al update, tendrìamos que pasar {new: true} como tercer argumento

Eliminar documentos

- se puede usar también deleteMany o findByIdAndRemove

```
async function deletePersona(id) {
    const result = await Persona.deleteOne({_id: id});
    console.log(result);
}

deletePersona('60c3a18e16f00c0b60c3fa8b');
```

Referenciar documentos

- Ejemplo: ciudad y estado

```
const estadoSchema = new mongoose.Schema({
    nombre: String
});
const Estado = mongoose.model('Estado', personaSchema);

async function crearEstado() {
    const estado = new Estado({
        nombre: "NUEVO LEON"
    });

    const resultado = await estado.save();
    console.log(resultado);
}
```

Referenciar documentos

```
const ciudadSchema = new mongoose.Schema({
    nombre: String,
    estado: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Estado'
    }
});
const Ciudad = mongoose.model('Ciudad', ciudadSchema);
async function crearCiudad() {
    const ciudad = new Ciudad({
        nombre: "SALTILLO3",
        estado: '60c3b257bbd43a33e427b778'
    });
    const resultado = await ciudad.save();
    console.log(resultado);
}
```

Populate

```
{
  _id: 60c3b41178f22d14e8cbb510,
  nombre: 'SALTILLO3',
  estado: { _id: 60c3b159f2e6bb0910efde3d, nombre: 'COAHUILA', __v: 0 },
  __v: 0
},
{
  _id: 60c3b421a3d5c12784f66f52,
  nombre: 'SALTILLO3',
  estado: null,
  __v: 0
}
```

```
async function getCiudades(){
    const ciudades = await Ciudad
        .find()
        .populate('estado');
    console.log(ciudades);
}
getCiudades();
```

Documentos embebidos

```
const ciudadSchema = new mongoose.Schema({
    nombre: String,
    estado: estadoSchema
});

const Ciudad = mongoose.model('Ciudad', ciudadSchema);
async function crearCiudad() {
    const ciudad = new Ciudad({
        nombre: "MONTERREY",
        estado: new Estado({nombre: 'NUEVO LEON'})
    });
    const resultado = await ciudad.save();
    console.log(resultado);
}
crearCiudad();
return;
```