UNIVERSIDAD DE
COSTA RICA

IF 3000 PROGRAMACIÓN II

Clase16: Introducción al modelo Cliente-Servidor API REST

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.
app.js	Este archivo será usado para crear la aplicación servidor.
node_modules folder	Este archivo contendrá todos los paquetes de node js.

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 }));
});
```

Probar nuestro API REST desde Visual Studio Code

- Instalar la extensión REST Client en VSCode
- Crear un archivo en la carpeta principal llamado **request.http** y digitar lo siguiente:

```
###
GET http://127.0.0.1:3000/est HTTP/1.1
```

- Luego, hacer clic en Send Request

users.js REST (GET) un elemento por su id

```
// GET un elemento de la base de datos indicando su id
router.get('/:id', (req, res) => {
  const { id } = req.params;
  Estudiante
    .findById(id)
    .then((data) => res.json(data))
    .catch((error) => res.json({ message: error }));
});
```

Prueba en REST Client VSCode

```
### donde el 2 es el parámetro
GET http://127.0.0.1:3000/est/2 HTTP/1.1
```

- Luego, hacer clic en Send Request

users.js REST (POST) para agregar un elemento

```
// POST un elemento nuevo en la base de datos
router.post('/', (req, res)=> {
  const est = new Estudiante(req.body)
    est
    .save()
    .then((data) => res.json(est))
    .catch((error) => res.json({ message: error }));
});
```

Prueba en REST Client VSCode

```
###
POST http://127.0.0.1:3000/est HTTP/1.1
Content-Type: application/json

{"_id":5,
"carne":"C02256",
"nombre":"Andrea",
"edad":20}
```

- Luego, hacer clic en Send Request

users.js REST (PUT) modificar un elemento

```
// PUT un elemento nuevo en la base de datos
router.put('/:id', (req, res) => {
  const { id } = req.params;
  const {_id,carne,nombre,edad}=req.body
  Estudiante
    .updateOne({_id:id},{$set:{carne,nombre,edad}})
    .then((data) => res.json(data))
    .catch((error) => res.json({ message: error }));
});
```

Prueba en REST Client VSCode

```
###
PUT http://127.0.0.1:3000/est/5 HTTP/1.1
Content-Type: application/json

{"_id":5,
"carne":"B53454",
"nombre":"Juan",
"edad":22}
```

- Luego, hacer clic en Send Request

users.js REST (PATCH) modificar un atributo del elemento

```
// PATCH un elemento de la base de datos indicando su id
router.patch('/:id', (req, res) => {
  const { id } = req.params;
  const nombre=req.body.nombre
  Estudiante
    .updateOne({_id:id},{$set:{nombre}})
    .then((data) => res.json(data))
    .catch((error) => res.json({ message: error }));
});
```

Prueba en REST Client VSCode

```
###
PATCH http://127.0.0.1:3000/est/5 HTTP/1.1
Content-Type: application/json

{"nombre":"Maria"}
```

- Luego, hacer clic en Send Request

users.js REST (DELETE) elimina un elemento de la base

```
// DELETE un elemento nuevo en la base de datos
router.delete('/:id', (req, res) => {
  const { id } = req.params;
  Estudiante
    .deleteOne({_id:id})
    .then((data) => res.json(data))
    .catch((error) => res.json({ message: error }));
});
```

Prueba en REST Client VSCode

```
###
DELETE http://127.0.0.1:3000/est/5 HTTP/1.1
```

- Luego, hacer clic en Send Request

Crear un cliente html (front-end)

```
<!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>
```

Crear el archivo 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();
}
```

- El ejercicio completo funcional se subió a MediaciónVirtual