Transacciones y API REST

Registros (Parte I)

{desafío} latam_

Implementar operaciones transaccionales en una base de datos para mantener la consistencia de los datos utilizando el entorno Node.js.

Implementar la capa de acceso a datos utilizando un ORM con entidades no relacionadas para realizar operaciones CRUD.

Utilizar asociaciones uno a uno, uno a muchos y muchos a muchos en la definición de las relaciones entre las entidades de un modelo que resuelven un problema.

{desafío} latam_

- Unidad 1:
  Implementación y gestión de una base de datos
- Unidad 2:
  Transacciones y API REST
- Unidad 3:
  Trabajo práctico

Te encuentras aquí

¿Qué aprenderás en esta sesión?

- *Manejo de bases de datos PostgreSQL y la interacción con ellas a través de operaciones de inserción y consulta.*

{desafío} latam_

¿Qué factores creen que son más críticos al elegir un método de autenticación para asegurar la mejor experiencia para los usuarios y la seguridad de la aplicación?"

/* Insertando datos a PostgreSQL desde el servidor */

{desafío}
latam_

Insertando datos a PostgreSQL desde el servidor

Con lo aprendido en el capítulo anterior, podemos iniciar con la creación de una API REST para disponibilizar las funcionalidades de un sistema CRUD a nuestras aplicaciones clientes, y eso es justo lo que haremos ahora con el siguiente ejercicio.

Ejercicio guiado:
POST & CREATE (CREATE)

{desafío}
latam_

POST & CREATE

Ejercicio guiado

El gimnasio Heart Strong, necesita desarrollar un servidor en Node que devuelva varios formularios HTML, para una gestión de tipo CRUD de los registros almacenados en una tabla llamada **ejercicios**, de su base de datos llamada **gym**.

Para iniciar con este ejercicio necesitarás hacer lo siguiente:

- Abre tu terminal de PostgreSQL.
- Crea la base de datos con la siguiente instrucción:

```
CREATE DATABASE gym;
```

POST & CREATE

Ejercicio guiado

- Conéctate a la base de datos gym con la siguiente instrucción.

```
\c gym;
```

- Crea la tabla ejercicios con los campos: nombre, series, repeticiones y descanso. Para esto utiliza la siguiente instrucción:

```
CREATE TABLE ejercicios (nombre varchar(30), series varchar(30),
repeticiones varchar(30), descanso varchar(30));
```

POST & CREATE

Ejercicio guiado

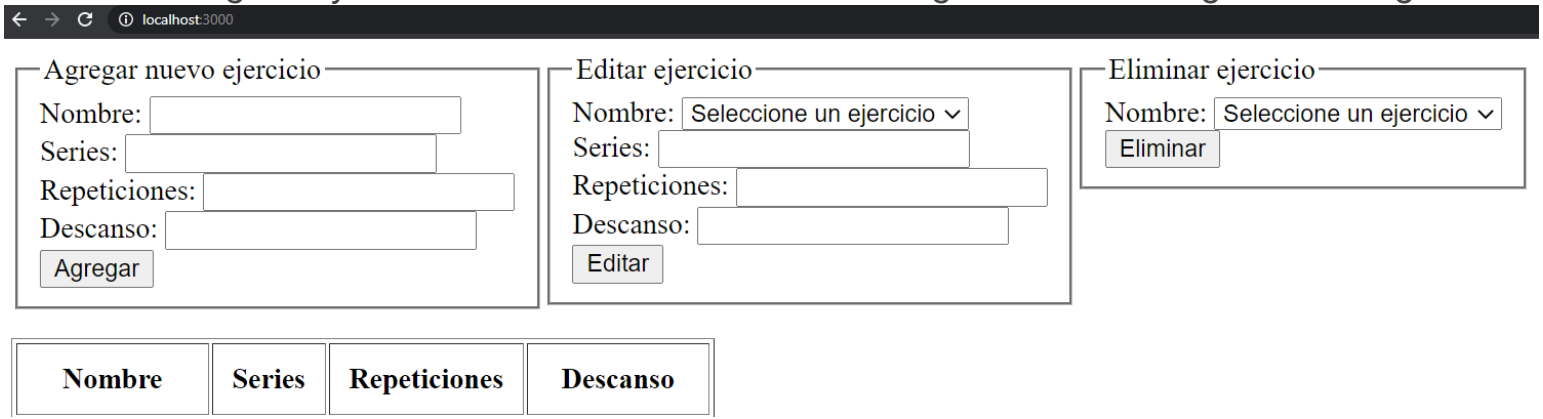
Con la tabla ejercicios creada, ahora si podemos iniciar a construir la API REST, partiendo con la creación de un servidor con una ruta raíz que devuelva un documento HTML (index.html), que encontrarás como material de apoyo llamado Apoyo - Gimnasio Heart Strong en el que está desarrollada una interfaz básica pero funcional, que consultará rutas a nuestro servidor.

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

app.listen(3000, console.log("Server ON"))


app.get("/", (req, res) => {
    res.sendFile(__dirname + "/index.html");
})
```

POST & CREATE

Ejercicio guiado

Ahora abre tu navegador y entra al servidor. Deberás recibir algo similar a la siguiente imagen:



Perfecto, ahora que tenemos los formularios a nuestra disposición en el lado del cliente, debemos crear la función asíncrona en el archivo “consultas.js”, encargada de recibir un payload enviado desde el cliente en el formulario con leyenda “Agregar nuevo ejercicio”, te recomiendo que te tomes unos 5 minutos para revisar con detención el código escrito en el HTML y luego continúes.

{desafío} latam_

/* Función Insertar */

Función Insertar

Prosigue con los siguientes pasos para el desarrollo de esta etapa:

- **Paso 1:** Agregar la base de datos “gym” al objeto de configuración de la clase Pool.
- **Paso 2:** Crear una función asíncrona llamada “insertar” que reciba un parámetro “datos”. El cual será un arreglo enviado desde el servidor.
- **Paso 3:** Realizar una consulta parametrizada con un JSON como argumento definiendo como “values” el parámetro “datos” de la función y retornando el objeto “result”.

Función Insertar

- **Paso 4:** Exportar un objeto con la función “insertar”

```
const { Pool } = require("pg");

const pool = new Pool({
  user: "postgres",
  host: "localhost",
  password: "postgres",
  port: 5432,
  // Paso 1
  database: "gym",
});

// Paso 2
const insertar = async (datos) => {
  // Paso 3
  const consulta = {
    text: "INSERT INTO ejercicios values($1, $2, $3, $4)",
    values: datos,
  };
  const result = await pool.query(consulta);
  return result;
};
// Paso 4
module.exports = { insertar };
```

/* Ruta POST */

Ruta POST

Con lo anterior terminado, ahora necesitamos crear en nuestro servidor una ruta **POST /ejercicios**, que obtenga los datos enviados desde el cliente y se la pase como argumento, en formato de arreglo a la función “insertar” para finalmente devolver el resultado.

- **Paso 1:** Importar la función ingresar en el script del servidor
- **Paso 2:** Declarar el middleware que permita recibir payloads en las rutas del servidor.
- **Paso 3:** Crear una ruta **POST /ejercicios**.
- **Paso 4:** Utiliza el Object.values() para almacenar en un arreglo el objeto recibido en el cuerpo de la consulta

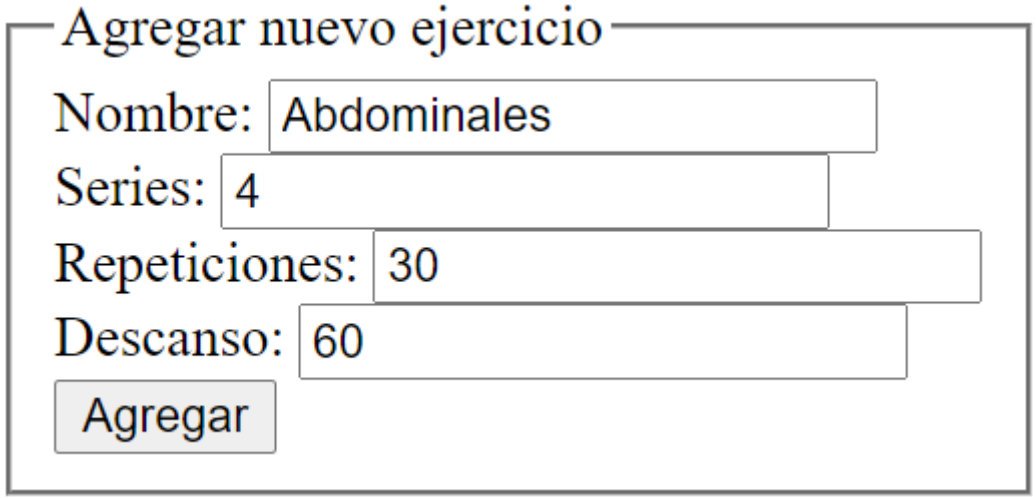
Ruta POST

- **Paso 5:** Utiliza la función ingresar y devuelve la respuesta a la aplicación cliente.

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

// Paso 1
const { insertar } = require('./consultas')

app.listen(3000, console.log("Server ON"))
// Paso 2
app.use(express.json())

app.get("/", async (req, res) => {
    res.sendFile(__dirname + "/index.html");
})

// Paso 3
app.post("/", async (req, res) => {
    try {
        // Paso 4
        const datos = Object.values(req.body)
        // Paso 5
        const respuesta = await insertar(datos)
        res.json(respuesta)
    } catch (error) {
        res.status(500).send("Algo salió mal :/ ...")
    }
})
```

Ruta POST

Ahora que tenemos la ruta creada, preparada en el servidor y la función insertar desarrollada en el archivo “consultas.js”, es momento de probar esto. Intenta insertar un nuevo ejercicio a través del formulario HTML, como te muestro en la siguiente imagen:



Ruta POST

Ahora, con los datos escritos en los inputs, presiona el botón “Agregar” y consulta en la terminal de PostgreSQL los datos de la tabla ejercicios con la siguiente instrucción:

```
SELECT * FROM ejercicios;
```

```
gym=# select * from ejercicios;
                    clientes
   nombre    | series | repeticiones | descanso
-------------+--------+--------------+----------
 Abdominales | 4      | 30           | 60
(1 fila)
```

¡Felicitaciones!! Ahí está, como puedes observar en la imagen anterior, se ha agregado un registro a la tabla ejercicios con los datos escritos en el formulario HTML.

{desafío} latam_

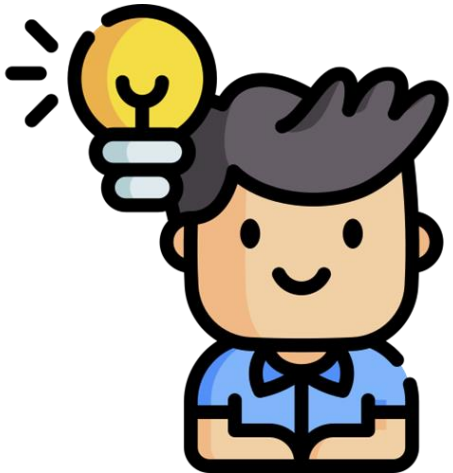
Antes de continuar con nuestro próximo tema:

Si tuvieran que resumir en una sola palabra o frase lo más importante que han aprendido hasta ahora en la sesión, ¿cuál sería?

/* Consultando registros de una tabla */

Consultando registros de una tabla

Con lo aprendido en el capítulo anterior, podemos continuar con el desarrollo de nuestra API REST, en esta ocasión nos enfocaremos en la consulta de registros de nuestra tabla **ejercicios**.



{desafío} latam_

Ejercicio guiado:
GET & SELECT (READ)

{desafío}
latam_

GET & SELECT (READ)

Ejercicio guiado

Al igual que la función “insertar” necesitaremos una función “consultar” en el archivo “consultas.js” que contenga la lógica para realizar la consulta SQL y obtener todos los registros de la tabla. Prosigue con los siguientes pasos para agregar la función “insertar” e iniciar con el desarrollo de esta etapa.

- **Paso 1:** Crear una función asíncrona llamada “consultar”.
- **Paso 2:** Generar una consulta SQL que solicite todos los registros de la tabla **ejercicios**.
- **Paso 3:** Incluir entre los módulos para exportar la función creada.

GET & SELECT (READ)

Ejercicio guiado

```
// Paso 1
const consultar = async () => {

 //   Paso 2
 const result = await pool.query("SELECT * FROM ejercicios");
 return result;

};

// Paso 3
module.exports = { insertar, consultar };
```

/* Ruta GET */

Ruta GET

Con la función “consultar” creada, ahora procedemos con la inclusión en el servidor de la ruta **GET /ejercicios** que utilizará esta función para devolver los registros de la tabla **ejercicios**.

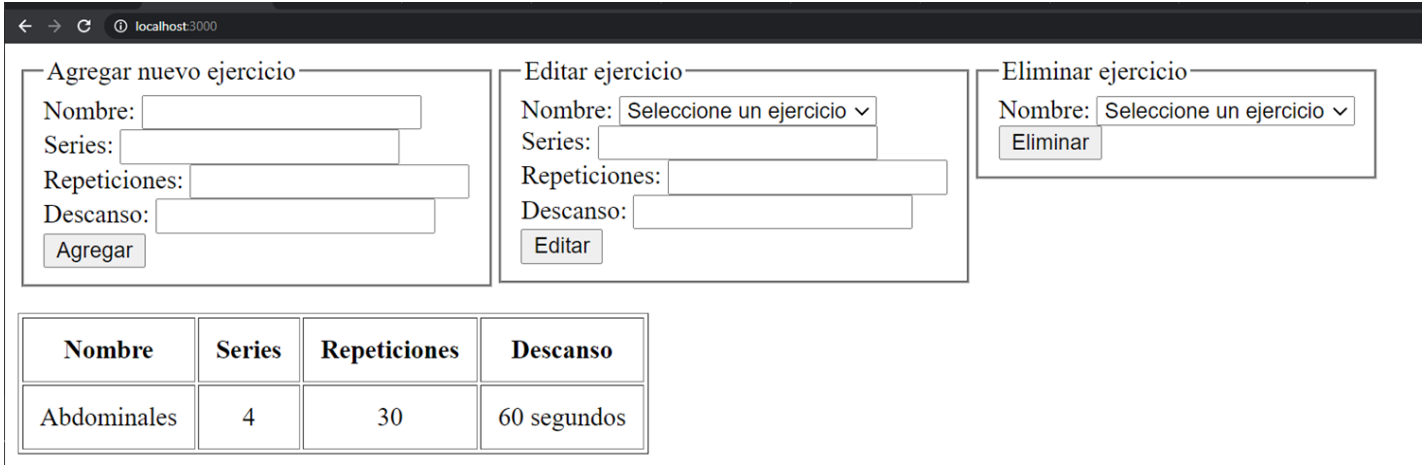
- **Paso 1:** Importar la función consultar.
- **Paso 2:** Crear la ruta **GET /ejercicios**.
- **Paso 3:** Ejecuta la función consultar y devuelve los registros a la aplicación cliente.

Ruta GET

```
// Paso 1
const { insertar, consultar } = require('./consultas')

// Paso 2
app.get("/ejercicios", async (req, res) => {
    try {
        // Paso 3
        const registros = await consultar();
        res.json(registros);
    } catch (error) {
        res.status(500).send("Algo salió mal :/ ...")
    }
})
```

Ruta GET

Ahora veamos si hicimos todo bien, para esto simplemente vuelve a consultar el servidor, porque en el documento HTML tiene escrita una función que consultará esta ruta para cargar la tabla automáticamente al cargar el sitio web. Si entras al servidor en la ruta raíz deberás ver lo que te muestro en la siguiente imagen:

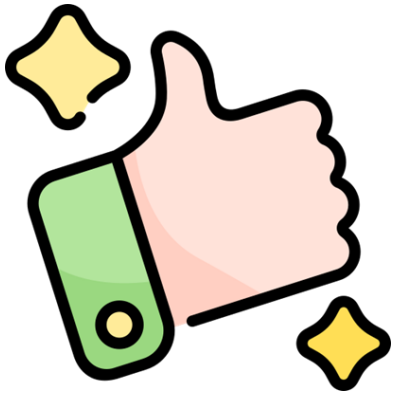


Ruta GET

Como puedes observar la tabla HTML se ha cargado con los registros de la tabla ejercicios. Si quieres comprobar de otra manera que todo salió bien, abre pestaña nueva en el navegador y consultando la dirección http://localhost:3000/ejercicios deberás ver lo que te muestro en la siguiente imagen:

Ruta GET

Como puedes observar en la imagen anterior, estamos recibiendo el objeto “result” que nos devuelve PostgreSQL cuando realizamos una consulta SQL con el comando SELECT, este JSON es el que estamos devolviendo desde el servidor y el que contiene en la propiedad “rows” todos los registros.



¿Qué temas encontraron más desafiantes o interesantes durante la sesión?

Próxima sesión...

- *Aspectos cruciales del desarrollo web, desde la implementación de operaciones CRUD y gestión de sesiones hasta la seguridad y el despliegue efectivo de aplicaciones web.*

{desafío} latam_

{desafío}
latam_

Academia de
talentos digitales