



Acceso a Base de Datos con Node (Parte I)



**Activen las cámaras los que puedan y
pasemos asistencia**





Inicio

{desafío}
latam_



Activación de conceptos

Contesta la pregunta correctamente y gana un punto

Instrucciones:

- Se realizará una pregunta, el primero en escribir “YO” por el chat, dará su respuesta al resto de la clase.
- El docente validará la respuesta.
- En caso de que no sea correcta, dará la oportunidad a la segunda persona que dijo “Yo”.
- Cada estudiante podrá participar un máximo de 2 veces.
- Al final, el/la docente indicará el 1º, 2º y 3º lugar.
- Esta actividad no es calificada, es solo una dinámica para recordar los conceptos clave para abordar esta sesión.





Activación de conceptos



¿Es Express un framework de Frontend o Backend?



Activación de conceptos



¿Qué es un middleware en Express JS?



Activación de conceptos



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

app.listen(3000, console.log("¡Servidor encendido!"))

app.post("/productos", (req, res) => {
  const producto = req.body
  res.send(`id producto ${producto.id}`)
})
```

¿Qué le hace falta al código para funcionar?



Activación de conceptos



¿Qué significa CORS?



Activación de conceptos



¿Cómo activamos CORS en una aplicación en
Express JS ?



Activación de conceptos



¿Cómo se deben exportar las siguientes funciones para poder importarlas de esta forma?

En esta unidad aprenderemos a crear un servidor Rest con ExpressJS que lea y guarde los datos de un pedido en una base de datos.

Objetivos

/ Insertar datos en una tabla alojada en PostgreSQL usando el paquete pg*/*

/ Mostrar por consola datos alojados en PostgreSQL usando el paquete pg */*

/ Crear una ruta GET con Express para devolver los registros de una tabla alojada en PostgreSQL */*

/ Crear una ruta POST con Express que reciba y almacene en PostgreSQL un nuevo registro */*

Objetivos



Desarrollo

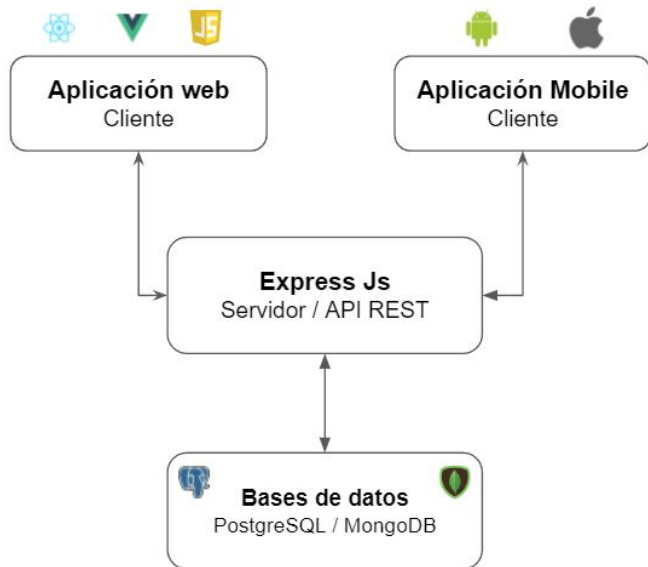
{desafío}
latam_



Express y bases de datos

Introducción a bases de datos en Node

Arquitectura de una aplicación con bases de datos

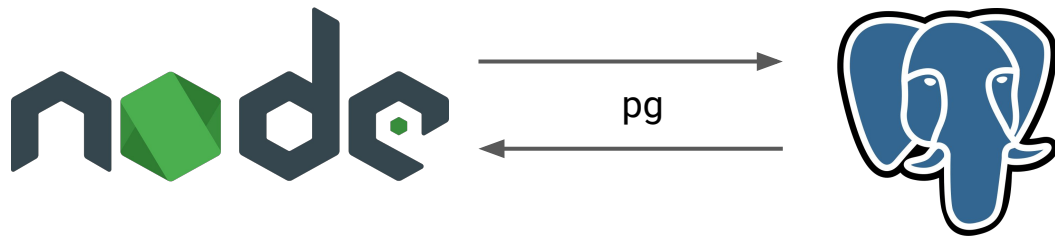


Con Node y Express podemos crear aplicaciones que pidan o guarden datos en distintos motores de bases de datos.

En esta unidad aprenderemos a hacerlo con postgresSQL

Introducción a bases de datos en Node

El paquete pg



El paquete pg nos permite conectarnos e interactuar con una base de datos PostgreSQL y está disponible en NPM.

Introducción a bases de datos en Node

Instalación del paquete pg

Para empezar a utilizar el paquete **pg** primero hay que instalarlo por npm, para esto, crea una nueva carpeta, dentro de la carpeta ejecuta la siguiente línea de comando para iniciar un proyecto NPM:

```
npm init -y
```



Asegúrate que el nombre de la carpeta no incluya mayúsculas

Ahora instalemos paquete **pg** con la siguiente línea de comando:

```
npm install pg
```

Luego de instalarse deberás ver "pg" en el archivo *package.json*



```
"dependencies": {  
  "pg": "^8.7.3"  
}
```

Introducción a bases de datos en Node

Primera consulta SQL con Node

Para empezar a hacer consultas SQL desde Node, crearemos una base de datos que tenga como objetivo almacenar los destinos y el presupuesto de un plan de viajes vacacionales.

Abre la terminal **psql** y escribe las siguientes instrucciones para crear una base de datos **plan_de_viajes** y una tabla **viajes**:

```
CREATE DATABASE plan_de_viajes;  
  
\c plan_de_viajes;  
  
CREATE TABLE viajes (id SERIAL, destino VARCHAR(50) NOT NULL, presupuesto INT NOT NULL);
```

Acceso a base datos con Node

Primera consulta SQL con Node

Ahora que tenemos la base de datos, realicemos nuestra primera consulta SQL desde Node haciendo lo siguiente:

1. Crear un archivo **consultas.js**
2. Importar la clase Pool del paquete pg
3. Crear una instancia de la clase Pool, usando un objeto de configuración con las credenciales.
4. Crear una función **getDate**
5. Usar el método **query()** para emitir una consulta que devuelva la fecha actual con la función NOW()
6. Mostrar el resultado de la consulta por consola.

```
const { Pool } = require('pg')

const pool = new Pool({
  host: 'localhost',
  user: 'postgres',
  password: 'postgres',
  database: 'plan_de_viajes',
  allowExitOnIdle: true
})

const getDate = async () => {
  const result = await pool.query("SELECT NOW()")
  console.log(result)
}

getDate()
```

Acceso a base datos con Node

Explicando el código

La clase **Pool** nos permite soportar multiconexiones y un mejor rendimiento en las consultas.

Esta propiedad le indicará a PostgreSQL que cierre la conexión luego de cada consulta.

Cada consulta devuelve un objeto **result** con el detalle obtenido en su ejecución.

```
const { Pool } = require('pg')

const pool = new Pool({
  host: 'localhost',
  user: 'postgres',
  password: 'postgres',
  database: 'plan_de_viajes',
  allowExitOnIdle: true
})

const getDate = async () => {
  const result = await pool.query("SELECT NOW()")

  console.log(result)
}

getDate()
```

Acceso a base datos con Node

El objeto result

El objeto result contiene entre varias propiedades:

- **command:** El comando SQL que se utilizó en la consulta.
- **rowCount:** Cantidad de filas que fueron procesadas en la consulta.
- **rows:** Un arreglo de objetos con todos los resultados o filas obtenido en la consulta.
- **fields:** La estructura de cada uno de los campos o columnas en las filas obtenidas.

```
Result {
  command: 'SELECT',
  rowCount: 1,
  oid: null,
  rows: [ { now: 2022-05-02T20:00:56.197Z } ],
  fields: [
    Field {
      name: 'now',
      tableID: 0,
      columnID: 0,
      dataTypeID: 1184,
      dataTypeSize: 8,
      dataTypeModifier: -1,
      format: 'text'
    }
  ],
  _parsers: [ [Function: parseDate] ],
  _types: TypeOverrides {
    _types: {
      getTypeParser: [Function: getTypeParser],
      setTypeParser: [Function: setTypeParser],
      arrayParser: [Object],
      builtins: [Object]
    },
    text: {},
    binary: {}
  },
  RowCtor: null,
  rowAsArray: false
}
```

Acceso a base datos con Node

Primer registro en PostgreSQL desde Node

Registremos un primer viaje agregando al final del script **consulta.js** la siguiente función:

```
const agregarViaje = async (destino, presupuesto) => {  
  const consulta = "INSERT INTO viajes values (DEFAULT, $1, $2)"  
  const values = [destino, presupuesto]  
  const result = await pool.query(consulta, values)  
  console.log("Viaje agregado")  
}
```

En esta ocasión estamos usando el método **query()** para hacer una consulta parametrizada que nos ayuda a evitar un problema llamado SQL Injection, el cual se discute en la guía de esta unidad.

En donde cada parámetro se representa por el símbolo del dólar(\$) seguido del orden en el que se declaran sus valores en un segundo argumento(values).



```
pool.query(<consulta parametrizada>, <arreglo de valores>)
```

Acceso a base datos con Node

Primer registro en PostgreSQL desde Node

Ejecutemos la función **agregarViaje** pasando como argumentos el destino “Valdivia” y como presupuesto 150000

```
agregarViaje("Valdivia", 150000)
```

Y luego revisamos manualmente en la terminal **psql** si el registro se realizó con éxito usando la siguiente consulta:

```
plan_de_viajes=# SELECT * FROM viajes;
 id | destino | presupuesto
-----+-----+-----
  1 | Valdivia |      150000
(1 fila)
```


Ejercicio

Ejecuta la siguiente consulta SQL en la base de datos **plan_de_viajes** para crear una tabla que registre un inventario del equipamiento que se piensa llevar en las vacaciones:


```
CREATE TABLE equipamiento (id SERIAL, nombre VARCHAR(50));
```

Luego:

1. Crea un nuevo script de nombre **equipamiento.js**
2. Importa la clase **Pool** del paquete pg
3. Crea una instancia de la clase **Pool** con las credenciales de la base de datos.
4. Crea una función llamada **agregarEquipamiento(nombre)** que realice una consulta parametrizada para hacer un nuevo registro en la tabla correspondiente.
5. Ejecuta la función y revisa manualmente si el registro se logró con éxito.

Ejercicio ¡Manos al teclado!



/* Registrar nuevas filas en una tabla alojada en PostgreSQL usando el paquete pg*/ 

/* Mostrar por consola datos alojados en PostgreSQL usando el paquete pg*/

/* Crear una ruta GET con Express para devolver los registros de una tabla alojada en PostgreSQL */

/* Crear una ruta POST con Express que reciba y almacene en PostgreSQL un nuevo registro */

Objetivos

Acceso a base datos con Node

Obteniendo registros de PostgreSQL desde Node

Para obtener los registros almacenados en PostgreSQL crea una nueva función en el archivo **consultas.js** que muestre por consola y retorne las filas de la tabla **viajes**:

```
const obtenerViajes = async () => {  
  const { rows } = await pool.query("SELECT * FROM viajes")  
  console.log(rows)  
  return rows  
}  
  
obtenerViajes()
```

Ejecutando esta función veremos por consola el viaje a Valdivia registrado anteriormente.

```
$ node consultas.js  
[ { id: 1, destino: 'Valdivia', presupuesto: 150000 } ]
```

Ejercicio

En el archivo **equipamiento.js** crea una función que muestre por consola y retorne los equipamientos registrados en la tabla equipamiento.

Luego ejecuta la función y observa por consola si se muestra el equipamiento que registraste en el ejercicio anterior.

```
$ node equipamiento.js  
[ { id: 1, nombre: 'Selfie Stick' } ]
```

Ejercicio ¡Manos al teclado!



*/** Registrar nuevas filas en una tabla alojada en PostgreSQL usando el paquete pg**/* ✓

*/** Mostrar por consola datos alojados en PostgreSQL usando el paquete pg**/* ✓

*/** Crear una ruta GET con Express para devolver los registros de una tabla alojada en PostgreSQL **/*

*/** Crear una ruta POST con Express que reciba y almacene en PostgreSQL un nuevo registro **/*

Objetivos

Acceso a base datos con Node

API REST con PostgreSQL(GET)

Ahora que podemos interactuar con una base de datos PostgreSQL para obtener y hacer registros, unamos los conocimientos previos para crear una API REST con Express que ofrezca una ruta **GET /viajes** que devuelva todos los viajes registrados en la tabla.

Para esto será necesario que el archivo **consultas.js** exporte las funciones que creamos recientemente. Agrega la siguiente línea de código al final del archivo

```
module.exports = { agregarViaje, obtenerViajes }
```

Acceso a base datos con Node

API REST con PostgreSQL(GET)

Ahora que podemos acceder a las funciones como módulos, procedamos con la creación de la API REST haciendo lo siguiente:

1. Crea un archivo ***index.js***
2. Instala express
3. Importa el paquete express y las funciones de ***consultas.js***
4. Crea un servidor en el puerto 3000
5. Crea una ruta **GET /viajes** que utilice la función **obtenerViajes** para devolver los registros a una aplicación cliente.

```
const { agregarViaje, obtenerViajes } = require('./consultas')
const express = require('express');
const app = express();

app.listen(3000, console.log("SERVIDOR ENCENDIDO"))

app.get("/viajes", async (req, res) => {
  const viajes = await obtenerViajes()
  res.json(viajes)
})
```

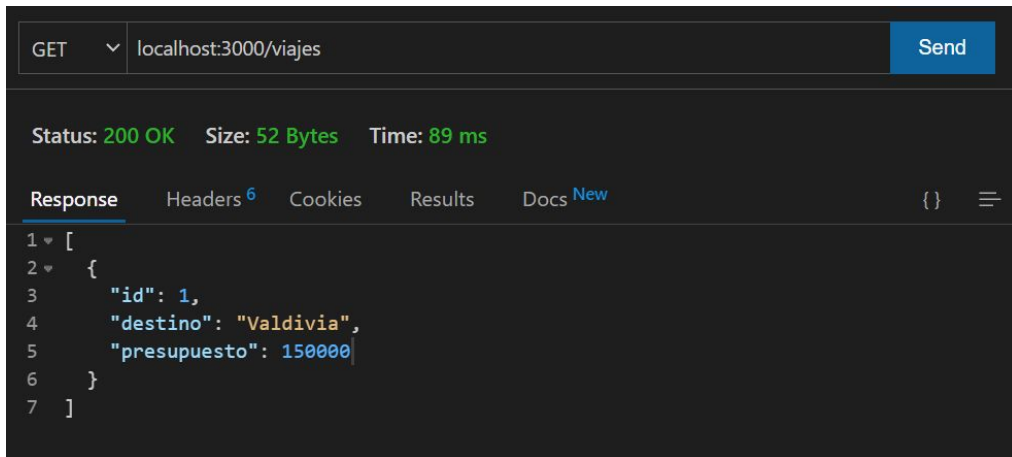
Acceso a base datos con Node

API REST con PostgreSQL(GET)

Levanta el servidor ejecutando el script **index.js** por la terminal y utiliza la extensión Thunder Client para consultar la ruta:

localhost:3000/viajes

Y deberás recibir un arreglo de objetos con el viaje de Valdivia registrado anteriormente.



Ejercicio


En el archivo **equipamiento.js** exporta las funciones creadas para agregar y obtener los registros de la tabla equipamiento.


Luego crea un archivo **server.js** en donde deberás:


1. Importar las funciones del archivo **equipamiento.js**
2. Importar el paquete express
3. Crear un servidor en el puerto 3001
4. Crear una ruta **GET /equipamientos** que devuelva los equipamientos registrados en la base de datos.

Ejercicio ¡Manos al teclado!



/* Registrar nuevas filas en una tabla alojada en PostgreSQL usando el paquete pg*/ 

/* Mostrar por consola datos alojados en PostgreSQL usando el paquete pg*/ 

/* Crear una ruta GET con Express para devolver los registros de una tabla alojada en PostgreSQL */ 

/* Crear una ruta POST con Express que reciba y almacene en PostgreSQL un nuevo registro */

Objetivos

Acceso a base datos con Node

API REST con PostgreSQL(POST)

Para permitir desde nuestro servidor la posibilidad de registrar nuevos equipamientos procedamos a crear una ruta **POST /viajes** que utilice la función **agregarViaje()** y un payload recibido en la consulta.

Al inicio del código agrega el middleware que nos permite parsear el cuerpo de la consulta:

```
app.use(express.json())
```

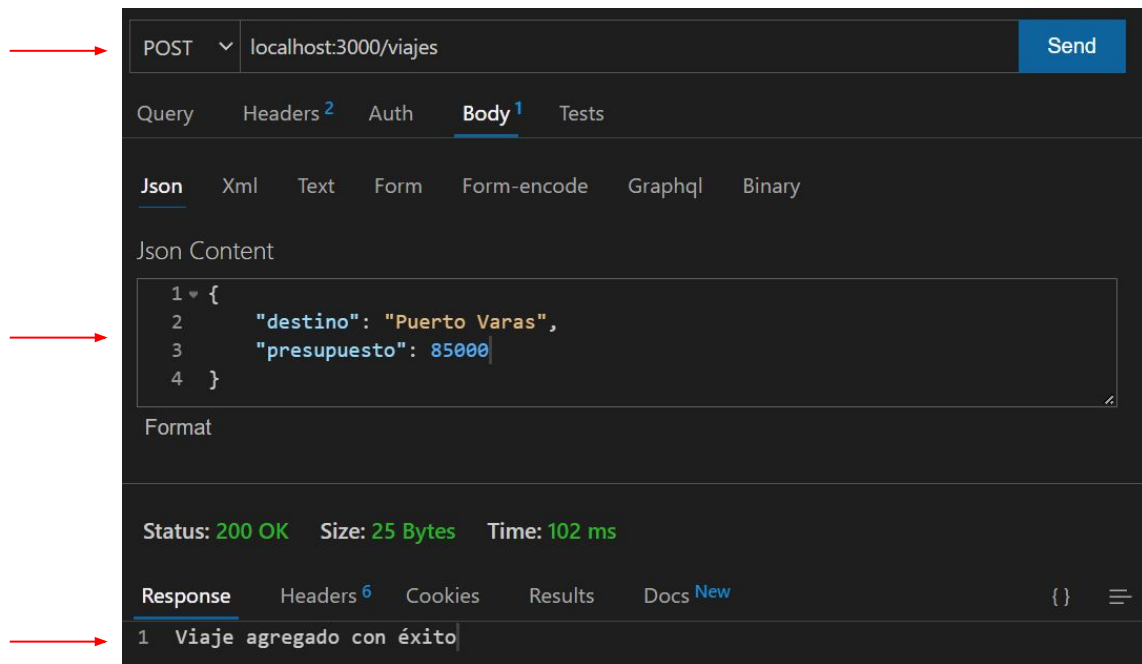
Y al final del script creamos la ruta **POST /viajes**:

```
app.post("/viajes", async (req, res) => {  
  const { destino, presupuesto } = req.body  
  await agregarViaje(destino, presupuesto)  
  res.send("Viaje agregado con éxito")  
})
```

Acceso a base datos con Node

API REST con PostgreSQL(POST)

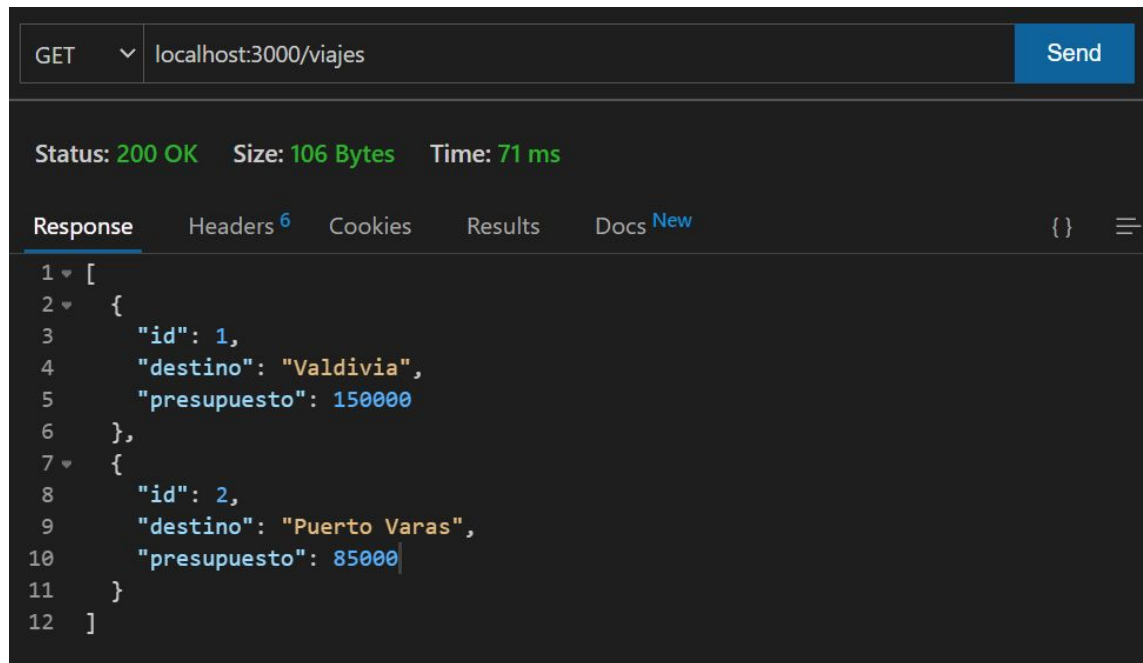
Para probar la ruta creada realicemos otra consulta con Thunder Client usando el método **POST** y agreguemos como cuerpo un objeto con un destino y un presupuesto:



Acceso a base datos con Node

API REST con PostgreSQL(POST)

Podemos confirmar que el viaje fue agregado con éxito revisando manualmente en la terminal **psql** o realizando otra consulta **GET /viajes**



```
GET localhost:3000/viajes Send

Status: 200 OK Size: 106 Bytes Time: 71 ms

Response Headers 6 Cookies Results Docs New {} ≡

1 [
2   {
3     "id": 1,
4     "destino": "Valdivia",
5     "presupuesto": 150000
6   },
7   {
8     "id": 2,
9     "destino": "Puerto Varas",
10    "presupuesto": 85000
11  }
12 ]
```

Ejercicio

En el archivo **server.js**:

1. Agrega el middleware para parsear el payload de la consulta.
2. Crea una ruta **POST /equipamientos**
3. Usa la función creada para agregar un nuevo equipamiento en base de datos.
4. Consulta la ruta **GET /equipamientos** para verificar que el registro fue realizado con éxito.

Ejercicio ¡Manos al teclado!



/* Registrar nuevas filas en una tabla alojada en PostgreSQL usando el paquete pg*/ ✓

/* Mostrar por consola datos alojados en PostgreSQL usando el paquete pg*/ ✓

/* Crear una ruta GET con Express para devolver los registros de una tabla alojada en PostgreSQL */ ✓

/* Crear una ruta POST con Express que reciba y almacene en PostgreSQL un nuevo registro */ ✓

Objetivos



Cierre

{desafío}
latam_



¿Existe algún concepto que no
hayas comprendido?

Reflexionemos

- En la guía encontrarás el contenido de esta clase, en esta ocasión no se profundiza en otros temas.
- Revisar en conjunto el desafío.

¿Qué sigue?



*Academia de
talentos digitales*

www.desafiolatam.com



/DesafioLatam



/DesafioLatam



/DesafioLatam



/DesafioLatam