

Actividad 4

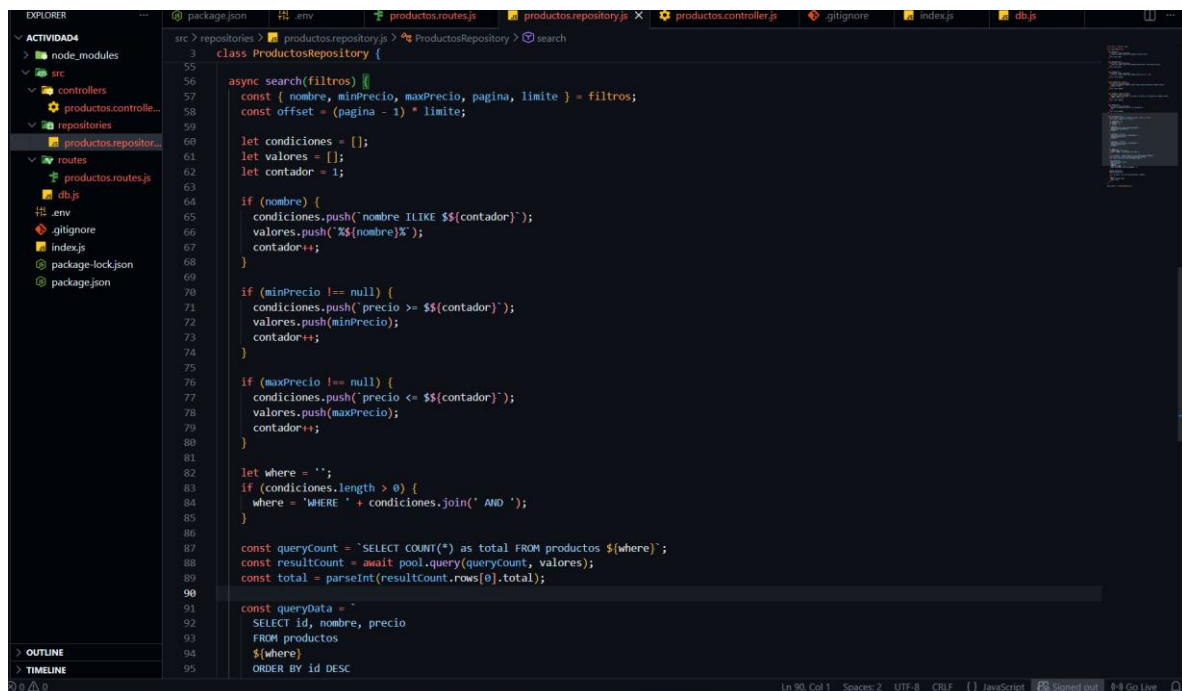
Abish Abril Santana García
Desarrollo full stack
Universidad Tecmilenio

Cómo construiste la query dinámica

En mi archivo productos.repository.js, dentro de la función search:

- Primero, creé un array que se llama condiciones. Ahí voy guardando cada filtro que el usuario pide en la url.
 - Segundo, creé otro array que se llama valores. Aquí guardo los datos reales que el usuario escribió.
 - Tercero, usé un contador que empieza en 1 y va aumentando cada vez que agrego un filtro.
 - Cuarto, al final reviso si hay algo en condiciones. Si tiene, las junto con AND para formar el WHERE. Si no hay filtros, no pongo WHERE.
1. Para la paginación, recibo page y limit, calculo OFFSET y los uso en la consulta.

Así mi consulta funciona con cualquier combinación de filtros que el usuario quiera usar.



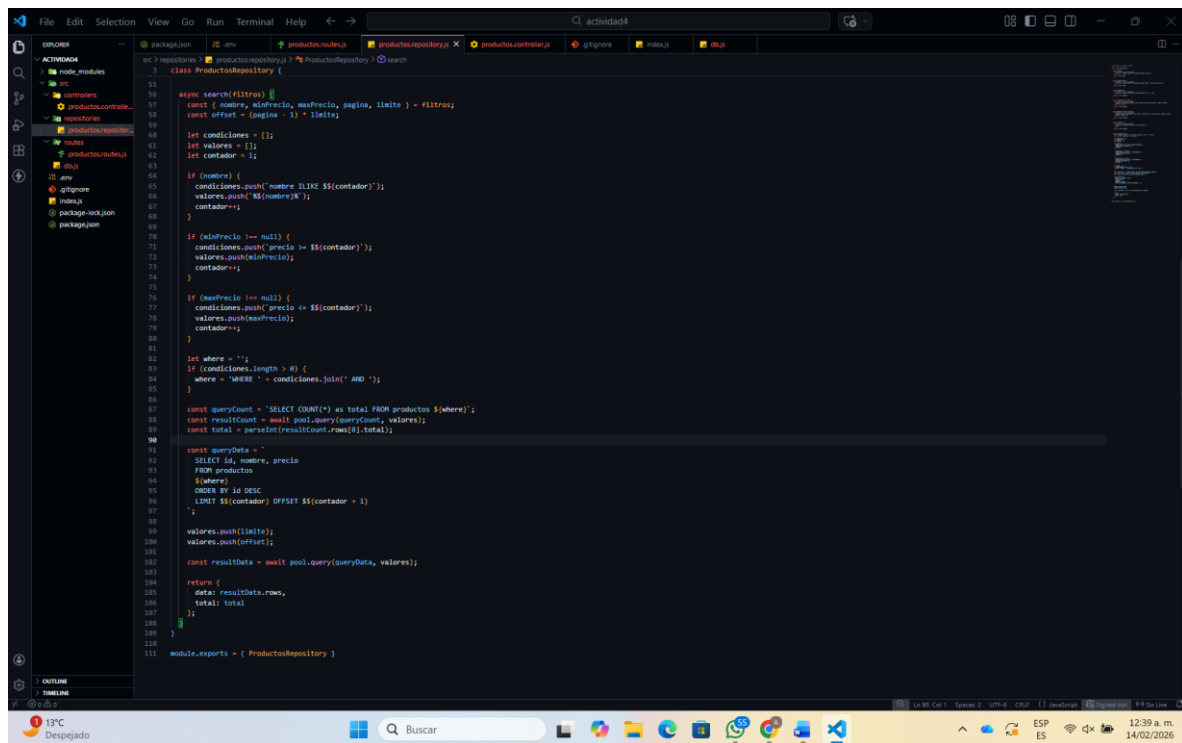
```
src > repositories > productos.repository.js > search
class ProductosRepository {
  55
  56 async search(filtros) {
  57   const { nombre, minPrecio, maxPrecio, pagina, limite } = filtros;
  58   const offset = (pagina - 1) * limite;
  59
  60   let condiciones = [];
  61   let valores = [];
  62   let contador = 1;
  63
  64   if (nombre) {
  65     condiciones.push(`nombre ILIKE ${`${contador}`}`);
  66     valores.push(`${nombre}%`);
  67     contador++;
  68   }
  69
  70   if (minPrecio !== null) {
  71     condiciones.push(`precio >= ${`${contador}`}`);
  72     valores.push(minPrecio);
  73     contador++;
  74   }
  75
  76   if (maxPrecio !== null) {
  77     condiciones.push(`precio <= ${`${contador}`}`);
  78     valores.push(maxPrecio);
  79     contador++;
  80   }
  81
  82   let where = '';
  83   if (condiciones.length > 0) {
  84     where = `WHERE ` + condiciones.join(' AND ');
  85   }
  86
  87   const queryCount = `SELECT COUNT(*) as total FROM productos ${where}`;
  88   const resultCount = await pool.query(queryCount, valores);
  89   const total = parseInt(resultCount.rows[0].total);
  90
  91   const queryData = `
  92     SELECT id, nombre, precio
  93     FROM productos
  94     ${where}
  95     ORDER BY id DESC
```

Cómo evitaste SQL Injection

En mi archivo productos.repository.js, dentro de la función search:

- Primero, nunca concatené los datos directamente en la consulta.
- Segundo, utilicé los parámetros \$1 para recibir valores de forma segura. Por ejemplo, en lugar de pegar el texto, escribo "WHERE nombre ILIKE \$1".
- Tercero, todos los datos que el usuario escribe los guardé en el array valores. Así la consulta queda limpia y los datos van aparte.
- Cuarto, la librería pg que usamos para conectar con la base de datos se encarga automáticamente de escapar cualquier carácter especial.
- Quinto, en mi controlador también validé que page y limit sean números antes de usarlos.

De esta forma, aunque alguien intente meter código malicioso en la búsqueda, la base de datos lo interpreta solo como texto normal y no como parte de la consulta.



```
17  async search(filters) {
18    const { nombre, minPrecio, maxPrecio, pagina, limite } = filters;
19    const offset = (pagina - 1) * limite;
20
21    let condiciones = [];
22    let valores = [];
23    let contador = 1;
24
25    if (nombre) {
26      condiciones.push(`nombre ILIKE $${contador}`);
27      valores.push(`%${nombre}%`);
28      contador++;
29    }
30
31    if (minPrecio !== null) {
32      condiciones.push(`precio >= $${contador}`);
33      valores.push(minPrecio);
34      contador++;
35    }
36
37    if (maxPrecio !== null) {
38      condiciones.push(`precio <= $${contador}`);
39      valores.push(maxPrecio);
40      contador++;
41    }
42
43    let where = '';
44    if (condiciones.length > 0) {
45      where = `WHERE ` + condiciones.join(' AND ');
46    }
47
48    const queryCount = `SELECT COUNT(*) as total FROM productos $${where}`;
49    const resultCount = await pool.query(queryCount, valores);
50    const total = parseInt(resultCount.rows[0].total);
51
52    const queryData = `
53    SELECT id, nombre, precio
54    FROM productos
55    $${where}
56    ORDER BY id DESC
57    LIMIT $${contador} OFFSET $${contador - 1}
58    `;
59
60    valores.push(limite);
61    valores.push(offset);
62
63    const resultData = await pool.query(queryData, valores);
64
65    return {
66      data: resultData.rows,
67      total: total
68    };
69  }
70
71  module.exports = { ProductosRepository }
```

Link de postman <https://documenter.getpostman.com/view/51907016/2sBXcBnheh>