

**EVALUACIÓN PERMANENTE 4
(INTEGRACIÓN DE APLICACIONES)
2024-10**

APELLIDOS Y NOMBRES DEL ESTUDIANTE:	CORREO ELECTRÓNICO:
Luque Chuquillanqui, Jose Antonio	Joseluque195@gmail.com

Deberás leer detenidamente cada una de las indicaciones de la evaluación con la finalidad de cumplir con todos los puntos solicitados.

INSTRUCCIONES GENERALES:

- Esta es una actividad individual.
- Si tuvieras consultas con respecto a lo solicitado en uno o varios puntos, deberás comunicarte oportunamente con tu docente para que la inquietud sea aclarada en un plazo prudente y puedas cumplir con los plazos de entrega de la actividad.
- Culminada la evaluación, deberás subir el archivo guardándolo con tu NRC, apellido y nombre.
- Es responsabilidad exclusiva del estudiante subir adecuadamente el documento solicitado corroborando que sea el correcto y que se haya cargado sin errores a la plataforma ISIL+.
- NO SE REVISARÁN LAS EVALUACIONES ENTREGADAS FUERA DEL PLAZO ESTABLECIDO.

CONSIDERACIONES DEL ENTREGABLE

- La actividad debe estar ordenada en cuanto a forma y fondo.
- Si se van a incluir imágenes de referencia en la actividad, debes revisar que estén colocadas de manera ordenada y alineada al texto. No colocar imágenes de mucho peso o gran tamaño.
- La actividad debe mostrar los puntos solicitados en el mismo orden en el que se han solicitado.
- Las fuentes de información utilizadas deben ser citadas utilizando las normas APA.

CONTENIDO DE LA EVALUACIÓN:

I. INSTRUCCIONES

Estimado alumno, esta es tu evaluación permanente N°03 del curso de Integración de Aplicaciones, consta de 2 preguntas. Recuerde que esta evaluación es individual y deberás subirla a la plataforma hasta la fecha programada. Considere entregar todas las evidencias que considere para demostrar la óptima realización de su práctica (GitHub, GoogleDrive, OneDrive, etc.). Buena suerte.....!!!

II. CASO DE DESARROLLO

Una empresa comercial, "**TechStore**", se especializa en la venta de productos electrónicos, incluyendo computadoras, teléfonos móviles, tabletas, y accesorios. La empresa desea modernizar su infraestructura tecnológica y ha decidido implementar un API RestFull que permita gestionar sus operaciones diarias de manera más eficiente.

El API debe permitir realizar las siguientes operaciones:

1. Gestión de Productos (5ptos)

- Crear un producto nuevo: La empresa debe poder añadir nuevos productos con detalles como nombre, descripción, precio, cantidad en stock, y categoría.
- Obtener detalles de un producto: Permitir la consulta de información detallada de un producto específico por su ID.
- Actualizar un producto existente: Modificar la información de un producto, incluyendo ajustes en el precio y la cantidad en stock.
- Eliminar un producto: Permitir la eliminación de productos que ya no estén disponibles o que hayan sido descontinuados.

2. Gestión de Clientes (5ptos)

- Registrar un cliente nuevo: Añadir nuevos clientes con detalles como nombre, dirección de correo electrónico, número de teléfono, y dirección de envío.
- Obtener información de un cliente: Consultar los detalles de un cliente específico por su ID.
- Actualizar información del cliente: Permitir la actualización de la información del cliente, como la dirección de envío y el número de teléfono.
- Eliminar un cliente: Eliminar los registros de clientes que ya no compran en la tienda.

3. Gestión de Órdenes (5ptos)

- Crear una nueva orden: Registrar una nueva orden de compra, incluyendo el ID del cliente, los productos comprados, las cantidades y el total de la orden.
- Obtener detalles de una orden: Consultar la información de una orden específica por su ID.
- Actualizar el estado de una orden: Modificar el estado de la orden (por ejemplo, de "pendiente" a "enviado" o "entregado").
- Eliminar una orden: Eliminar registros de órdenes canceladas o erróneas.

4. Gestión de Categorías (5ptos)

- Crear una nueva categoría: Añadir nuevas categorías de productos.
- Obtener todas las categorías: Consultar todas las categorías disponibles.
- Actualizar una categoría: Modificar el nombre o la descripción de una categoría existente.
- Eliminar una categoría: Eliminar categorías que ya no se utilizan.

III. ESPECIFICACIONES TÉCNICAS

- El API debe ser desarrollado utilizando Node.js con Express.
- Utilizar una base de datos relacional como MySQL o PostgreSQL.
- Asegurarse de que el API siga los principios RESTful, incluyendo el uso correcto de los métodos HTTP (GET, POST, PUT, DELETE).

Package.json

```
package.json > ...
1  {
2    "name": "apiREST_techstore",
3    "version": "1.0.0",
4    "description": "Evaluacion permanente 4 - Integracion de aplicaciones",
5    "main": "index.js",
6    "scripts": {
7      "babel-node": "babel-node --presets=@babel/preset-env",
8      "dev": "nodemon --exec npm run babel-node src/index.js"
9    },
10   "author": "Jose Luque",
11   "license": "ISC",
12   "dependencies": {
13     "dotenv": "^16.4.5",
14     "express": "^4.19.2",
15     "promise-mysql": "^5.2.0"
16   },
17   "devDependencies": {
18     "@babel/cli": "^7.24.7",
19     "@babel/core": "^7.24.7",
20     "@babel/node": "^7.24.7",
21     "@babel/preset-env": "^7.24.7",
22     "morgan": "^1.10.0",
23     "nodemon": "^3.1.4"
24   }
25 }
```

README.md

EVALUACION PERMANENTE 4

Para ejecutar:

Realizar instalación de módulos de Node

```
npm install
```

Ejecutar e iniciar servidor

```
npm run dev
```

Index.js

```
src > JS index.js > ...  
1  import app from "./app"  
2  
3  const main = () => {  
4    app.listen(app.get("port"));  
5    console.log(`Server on port ${app.get("port")}`);  
6  };  
7  
8  main();|
```

Config.js

```
src > JS config.js > ...  
1  import { config } from 'dotenv';  
2  
3  config();  
4  
5  export default {  
6    host: process.env.HOST || '',  
7    database: process.env.DATABASE || '',  
8    user: process.env.USER || '',  
9    password: process.env.PASSWORD || '',  
10 };  
11
```


App.js

```
src > js app.js > ...
1  import express from 'express';
2  import morgan from 'morgan';
3  // Routes
4  import categoriaRoutes from './routes/categoria.routes';
5  import clienteRoutes from './routes/cliente.routes';
6  import productoRoutes from './routes/producto.routes';
7  import ordenRoutes from './routes/orden.routes';
8
9  const app = express();
10
11 // Configuraciones
12 app.set('port', 4000);
13
14 // Middlewares
15 app.use(morgan('dev'));
16 app.use(express.json());
17
18 // Rutas
19 app.use('/apiTechStore/categoria', categoriaRoutes);
20 app.use('/apiTechStore/cliente', clienteRoutes);
21 app.use('/apiTechStore/producto', productoRoutes);
22 app.use('/apiTechStore/orden', ordenRoutes);
23
24 export default app;
25
```

Database.js

```
src > database > js database.js > ...
1  import mysql from 'promise-mysql';
2  import config from '../config';
3
4  const conexion = mysql.createConnection({
5    host: config.host,
6    database: config.database,
7    user: config.user,
8    password: config.password,
9  });
10
11 const getConexion = () => {
12   return conexion;
13 };
14
15 module.exports = {
16   getConexion,
17 };
18
```

Categoría.routes.js

```
src > routes >  categoria.routes.js > [🔍] default
1  import { Router } from "express";
2  import { methods as categoriaController } from "../controllers/categoria.controller";
3
4  const router = Router();
5
6  router.get("/", categoriaController.listarCategorias);
7  router.get("/:id", categoriaController.listarCategoriaPorId);
8  router.post("/", categoriaController.anadirCategoria);
9  router.put("/:id", categoriaController.actualizarCategoria);
10 router.delete("/:id", categoriaController.eliminarCategoria);
11
12 export default router;
```

Categoría.controller.js

```
src > controllers >  categoria.controller.js > [🔍] actualizarCategoria > [🔍] resultado
1  import { getConnection } from '../database/database';
2
3  const listarCategorias = async (req, res) => {
4    try {
5      const conexion = await getConnection();
6      const resultado = await conexion.query(
7        'SELECT idCategoria, nombre, descripcion FROM categoria'
8      );
9      res.json(resultado);
10   } catch (error) {
11     res.status(500);
12     res.send(error.message);
13   }
14 };
15
16 const listarCategoriaPorId = async (req, res) => {
17   try {
18     const { id } = req.params;
19     const conexion = await getConnection();
20     const resultado = await conexion.query(
21       'SELECT idCategoria, nombre, descripcion FROM categoria WHERE idCategoria = ?',
22       id
23     );
24     res.json(resultado);
25   } catch (error) {
26     res.status(500);
27     res.send(error.message);
28   }
29 };
30
```

```
31 const anadirCategoria = async (req, res) => {
32   try {
33     const { nombre, descripcion } = req.body;
34
35     if (nombre === undefined || descripcion === undefined) {
36       return res.status(400).json({
37         message: 'Por favor ingrese todos los campos',
38       });
39     }
40
41     const categoria = { nombre, descripcion };
42     const conexion = await getConexion();
43     await conexion.query('INSERT INTO categoria SET ?', categoria);
44     res.json({
45       message: 'Categoria añadida con exito',
46     });
47   } catch (error) {
48     res.status(500);
49     res.send(error.message);
50   }
51 };
52
```

```
53 const actualizarCategoria = async (req, res) => {
54   try {
55     const { id } = req.params;
56     const { nombre, descripcion } = req.body;
57
58     if (
59       id === undefined ||
60       nombre === undefined ||
61       descripcion === undefined
62     ) {
63       return res.status(400).json({
64         message: 'Por favor ingrese todos los campos',
65       });
66     }
67
68     const categoria = { idCategoria: id, nombre, descripcion };
69     const conexion = await getConexion();
70     const resultado = await conexion.query(
71       'UPDATE categoria SET ? WHERE idCategoria = ?',
72       [categoria, id]
73     );
74     res.json(resultado);
75   } catch (error) {
76     res.status(500);
77     res.send(error.message);
78   }
79 };
```

```
80
81   const eliminarCategoria = async (req, res) => {
82     try {
83       const { id } = req.params;
84       const conexion = await getConexion();
85       const resultado = await conexion.query(
86         'DELETE FROM categoria WHERE idCategoria = ?',
87         id
88       );
89       res.json(resultado);
90     } catch (error) {
91       res.status(500);
92       res.send(error.message);
93     }
94   };
95
96   export const methods = {
97     listarCategorias,
98     listarCategoriaPorId,
99     anadirCategoria,
100    actualizarCategoria,
101    eliminarCategoria,
102  };
103
```

Cliente.router.js

```
src > routes > cliente.routes.js > [0] default
1   import { Router } from "express";
2   import { methods as clienteController } from "../controllers/cliente.controller";
3
4   const router = Router();
5
6   router.get("/", clienteController.listarClientes);
7   router.get("/:id", clienteController.listarClientePorId);
8   router.post("/", clienteController.anadirCliente);
9   router.put("/:id", clienteController.actualizarCliente);
10  router.delete("/:id", clienteController.eliminarCliente);
11
12  export default router;
```





```
src > controllers > cliente.controller.js > [x] actualizarCliente
1  import { getConexion } from '../database/database';
2
3  const listarClientes = async (req, res) => {
4    try {
5      const conexion = await getConexion();
6      const resultado = await conexion.query(
7        'SELECT idCliente, nombre, correo, telefono, direccion FROM cliente'
8      );
9      res.json(resultado);
10   } catch (error) {
11     res.status(500);
12     res.send(error.message);
13   }
14 };
15
16 const listarClientePorId = async (req, res) => {
17   try {
18     const { id } = req.params;
19     const conexion = await getConexion();
20     const resultado = await conexion.query(
21       'SELECT idCliente, nombre, correo, telefono, direccion FROM cliente WHERE idCliente = ?',
22       id
23     );
24     res.json(resultado);
25   } catch (error) {
26     res.status(500);
27     res.send(error.message);
28   }
29 };
30
```

```
30
31 const anadirCliente = async (req, res) => {
32   try {
33     const { nombre, correo, telefono, direccion } = req.body;
34
35     if (
36       nombre === undefined ||
37       correo === undefined ||
38       telefono === undefined ||
39       direccion === undefined
40     ) {
41       return res.status(400).json({
42         message: 'Por favor ingrese todos los campos',
43       });
44     }
45
46     const cliente = { nombre, correo, telefono, direccion };
47     const conexion = await getConexion();
48     await conexion.query('INSERT INTO cliente SET ?', cliente);
49     res.json({
50       message: 'Cliente añadido con éxito',
51     });
52   } catch (error) {
53     res.status(500);
54     res.send(error.message);
55   }
56 };
57
```

```
57
58 const actualizarCliente = async (req, res) => {
59   try {
60     const { id } = req.params;
61     const { nombre, correo, telefono, direccion } = req.body;
62
63     if (
64       id === undefined ||
65       nombre === undefined ||
66       correo === undefined ||
67       telefono === undefined ||
68       direccion === undefined
69     ) {
70       return res.status(400).json({
71         message: 'Por favor ingrese todos los campos',
72       });
73     }
74
75     const cliente = { idCliente: id, nombre, correo, telefono, direccion };
76     const conexion = await getConexion();
77     const resultado = await conexion.query(
78       'UPDATE cliente SET ? WHERE idCliente = ?',
79       [cliente, id]
80     );
81     res.json(resultado);
82   } catch (error) {
83     res.status(500);
84     res.send(error.message);
85   }
86 };
```

```
87
88 const eliminarCliente = async (req, res) => {
89   try {
90     const { id } = req.params;
91     const conexion = await getConexion();
92     const resultado = await conexion.query(
93       'DELETE FROM cliente WHERE idCliente = ?',
94       id
95     );
96     res.json(resultado);
97   } catch (error) {
98     res.status(500);
99     res.send(error.message);
100   }
101 };
102
103 export const methods = {
104   listarClientes,
105   listarClientePorId,
106   anadirCliente,
107   actualizarCliente,
108   eliminarCliente,
109 };
110
```

Producto.routes.js

```
src > routes >  producto.routes.js >  default
1  import { Router } from "express";
2  import { methods as productoController } from "../controllers/producto.controller"
3
4  const router = Router();
5
6  router.get("/", productoController.listarProductos);
7  router.get("/:id", productoController.listarProductoPorId);
8  router.post("/", productoController.anadirProducto);
9  router.put("/:id", productoController.actualizarProducto);
10 router.delete("/:id", productoController.eliminarProducto);
11
12 export default router;
```

Producto.controllers.js

```
src > controllers >  producto.controller.js >  actualizarProducto
1  import { getConexion } from '../database/database';
2
3  const listarProductos = async (req, res) => {
4    try {
5      const conexion = await getConexion();
6      const resultado = await conexion.query(
7        'SELECT idProducto, nombre, descripcion, precio, cantidad, idCategoria FROM producto'
8      );
9      res.json(resultado);
10   } catch (error) {
11     res.status(500);
12     res.send(error.message);
13   }
14 };
15
16 const listarProductoPorId = async (req, res) => {
17   try {
18     const { id } = req.params;
19     const conexion = await getConexion();
20     const resultado = await conexion.query(
21       'SELECT idProducto, nombre, descripcion, precio, cantidad, idCategoria FROM producto WHERE
22         idProducto = ?',
23       id
24     );
25     res.json(resultado);
26   } catch (error) {
27     res.status(500);
28     res.send(error.message);
29   }
30 };
```


```
30
31 const anadirProducto = async (req, res) => {
32   try {
33     const { nombre, descripcion, precio, cantidad, idCategoria } = req.body;
34
35     if (
36       nombre === undefined ||
37       descripcion === undefined ||
38       precio === undefined ||
39       cantidad === undefined ||
40       idCategoria === undefined
41     ) {
42       return res.status(400).json({
43         message: 'Por favor ingrese todos los campos',
44       });
45     }
46
47     // Validar categoria existente
48     const conexion = await getConexion();
49     const categoriaExiste = await conexion.query(
50       'SELECT * FROM categoria WHERE idCategoria = ?',
51       idCategoria
52     );
53
54     if (categoriaExiste.length === 0) {
55       return res.status(400).json({
56         message: 'La categoria indicada no existe',
57       });
58     }
59
60     const producto = { nombre, descripcion, precio, cantidad, idCategoria };
61
62     await conexion.query(
63       'INSERT INTO producto SET ?',
64       producto
65     );
66     res.json({
67       message: 'Producto añadido con éxito',
68     });
69   } catch (error) {
70     res.status(500);
71     res.send(error.message);
72   }
73 };
```

```
74
75 const actualizarProducto = async (req, res) => {
76   try {
77     const { id } = req.params;
78     const { nombre, descripcion, precio, cantidad, idCategoria } = req.body;
79
80     if (
81       id === undefined ||
82       nombre === undefined ||
83       descripcion === undefined ||
84       precio === undefined ||
85       cantidad === undefined ||
86       idCategoria === undefined
87     ) {
88       return res.status(400).json({
89         message: 'Por favor ingrese todos los campos',
90       });
91     }
92   }
```



```
92
93 // Validar categoria existente
94 const conexion = await getConexion();
95 const categoriaExiste = await conexion.query(
96   'SELECT * FROM categoria WHERE idCategoria = ?',
97   idCategoria
98 );
99
100 if (categoriaExiste.length === 0) {
101   return res.status(400).json({
102     message: 'La categoria indicada no existe',
103   });
104 }
105
106 const producto = {
107   idProducto: id,
108   nombre,
109   descripcion,
110   precio,
111   cantidad,
112   idCategoria,
113 };
114
115 const resultado = await conexion.query(
116   'UPDATE producto SET ? WHERE idProducto = ?',
117   [producto, id]
118 );
119 res.json(resultado);
120 } catch (error) {
121   res.status(500);
122   res.send(error.message);
123 }
124 };
125
```

```
123 const eliminarProducto = async (req, res) => {
124   try {
125     const { id } = req.params;
126     const conexion = await getConexion();
127     const resultado = await conexion.query(
128       'DELETE FROM producto WHERE idProducto = ?',
129       id
130     );
131     res.json(resultado);
132   } catch (error) {
133     res.status(500);
134     res.send(error.message);
135   }
136 };
137
138 export const methods = {
139   listarProductos,
140   listarProductoPorId,
141   anadirProducto,
142   actualizarProducto,
143   eliminarProducto,
144 };
145
```

Orden.routes.js

```
src > routes >  orden.routes.js > ...  
7   router.get('/:id', ordenController.listarOrdenPorId);  
8   router.post('/', ordenController.anadirOrden);  
9   router.patch('/:id', ordenController.actualizarEstadoOrden);  
10  router.delete('/:id', ordenController.eliminarOrden);  
11  
12  export default router;  
13
```

Orden.controller.js

```
src > controllers >  orden.controller.js >  actualizarEstadoOrden  
1  import { getConnection } from '../database/database';  
2  
3  const listarOrdenes = async (req, res) => {  
4    try {  
5      const conexion = await getConnection();  
6      const resultado = await conexion.query(  
7        'SELECT idOrden, cantidad, total, estado, idProducto, idCliente FROM orden'  
8      );  
9      res.json(resultado);  
10   } catch (error) {  
11     res.status(500);  
12     res.send(error.message);  
13   }  
14 };  
15  
16 const listarOrdenPorId = async (req, res) => {  
17   try {  
18     const { id } = req.params;  
19     const conexion = await getConnection();  
20     const resultado = await conexion.query(  
21       'SELECT idOrden, cantidad, total, estado, idProducto, idCliente FROM orden WHERE idOrden = ?',  
22       id  
23     );  
24     res.json(resultado);  
25   } catch (error) {  
26     res.status(500);  
27     res.send(error.message);  
28   }  
29 };  
30
```

```
31 ∨ const anadirOrden = async (req, res) => {
32 ∨   try {
33     const { cantidad, total, estado, idProducto, idCliente } = req.body;
34
35     if (
36       cantidad === undefined ||
37       total === undefined ||
38       estado === undefined ||
39       idProducto === undefined ||
40       idCliente === undefined
41     ) {
42       return res.status(400).json({
43         message: 'Por favor ingrese todos los campos',
44       });
45     }
46
47     // Validar producto y cliente existentes
48     const conexion = await getConexion();
49     const productoExiste = await conexion.query(
50       'SELECT * FROM producto WHERE idProducto = ?',
51       idProducto
52     );
53     const clienteExiste = await conexion.query(
54       'SELECT * FROM cliente WHERE idCliente = ?',
55       idCliente
56     );
57
58     if (productoExiste.length === 0 || clienteExiste.length === 0) {
59       return res.status(400).json({
60         message: 'El producto y/o cliente indicado no existe',
61       });
62     }
63 }
```

```
63
64 // Validar estado existente
65 if (
66   estado.toLowerCase() !== 'pendiente' &&
67   estado.toLowerCase() !== 'enviado' &&
68   estado.toLowerCase() !== 'entregado'
69 ) {
70   return res.status(400).json({
71     message: 'Por favor ingrese un estado valido',
72   });
73 }
74
75 const orden = { cantidad, total, estado, idProducto, idCliente };
76
77 await conexion.query('INSERT INTO orden SET ?', orden);
78 res.json({
79   message: 'Producto añadido con éxito',
80 });
81 } catch (error) {
82   res.status(500);
83   res.send(error.message);
84 }
85 };
```

```
86
87 const actualizarEstadoOrden = async (req, res) => {
88   try {
89     const { id } = req.params;
90     const { estado } = req.body;
91
92     if (id === undefined || estado === undefined) {
93       return res.status(400).json({
94         message: 'Por favor ingrese un estado',
95       });
96     }
97
98     // Validar estado existente
99     if (
100       estado.toLowerCase() !== 'pendiente' &&
101       estado.toLowerCase() !== 'enviado' &&
102       estado.toLowerCase() !== 'entregado'
103     ) {
104       return res.status(400).json({
105         message: 'Por favor ingrese un estado valido',
106       });
107     }
108
109     const conexion = await getConexion();
110     const orden = { estado };
111
112     await conexion.query('UPDATE orden SET ? WHERE idOrden = ?', [orden, id]);
113     res.json({
114       message: 'Estado modificado con exito',
115     });
116   } catch (error) {
117     res.status(500);
118     res.send(error.message);
119   }
120 };
121
```

```
122 const eliminarOrden = async (req, res) => {
123   try {
124     const { id } = req.params;
125     const conexion = await getConexion();
126     const resultado = await conexion.query(
127       'DELETE FROM orden WHERE idOrden = ?',
128       id
129     );
130     res.json(resultado);
131   } catch (error) {
132     res.status(500);
133     res.send(error.message);
134   }
135 };
136
137 export const methods = {
138   listarOrdenes,
139   listarOrdenPorId,
140   anadirOrden,
141   actualizarEstadoOrden,
142   eliminarOrden,
143 };
144
```