



PRACTICA 2



ESTUDIANTE: Alexander Condori Martinez

CÉDULA DE IDENTIDAD: 8343695

CARRERA: Informatica

MATERIA: Programación Web 3 (INF 133)

DOCENTE: Lic. Jhony Felipez Andrade

FECHA: 12/11/2025

UNIVERSIDAD MAYOR DE SAN ANDRÉS (UMSA)

La Paz - Bolivia

Noviembre, 2025

1. Crea un endpoint POST /categorias que permita registrar una nueva categoría enviando nombre y descripción en el body de la petición.

```
// ejercicio1
```

```
export const createCategoria = async (req, res) => {  
  const { nombre, descripción } = req.body;  
  if (!nombre) return res.status(400).json({ message: "El campo 'nombre' es obligatorio." });  
  
  try {  
    const id = await categoria.insertaCategoria(nombre, descripción);  
    res.status(201).json({ id, nombre, message: "Categoría creada exitosamente." });  
  } catch (error) {  
    res.status(500).json({ error: error.message });  
  }  
};
```

2. Crea un endpoint GET /categorias que devuelva todas las categorías registradas en la base de datos.

```
// ejercicio2
```

```
export const getCategories = async (req, res) => {  
  try {  
    const categorias = await categoria.obtenerTodasCategorias();  
    res.status(200).json(categorias);  
  } catch (error) {  
    res.status(500).json({ error: error.message });  
  }  
};
```

3. Crea un endpoint GET /categorias/:id que devuelva la categoría con el ID especificado y además, incluya todos los productos que pertenecen a esa categoría.

```
// ejercicio3

export const getCategoriaById = async (req, res) => {

  try {
    const categoria = await categoria.obtenerCategoriaConProductos(req.params.id);

    if (!categoria) return res.status(404).json({ message: "Categoría no encontrada." });

    res.status(200).json(categoria);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};
```

4. Crea un endpoint PUT /categorias/:id que permita actualizar todos los datos de la categoría con el ID especificado.

```
// ejercicio4

export const updateCategoria = async (req, res) => {

  const { nombre, descripcion } = req.body;

  if (!nombre) return res.status(400).json({ message: "El campo 'nombre' es obligatorio." });

  try {
    const affectedRows = await categoria.actualizaCategoria(req.params.id, nombre, descripcion);

    if (affectedRows === 0) return res.status(404).json({ message: "Categoría no encontrada para actualizar." });

    res.status(200).json({ message: "Categoría actualizada exitosamente." });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};
```

5. Crea un endpoint DELETE /categorias/:id que elimine la categoría indicada y, al mismo tiempo, elimine automáticamente todos los productos que pertenecen a esa categoría

```
// ejercicio5

export const deleteCategoria = async (req, res) => {

  try {
    const affectedRows = await categoria.eliminaCategoria(req.params.id);

    if (affectedRows === 0) return res.status(404).json({ message: "Categoría no encontrada para eliminar." });

    // Gracias a ON DELETE CASCADE, los productos asociados también fueron eliminados.

    res.status(200).json({ message: "Categoría y sus productos eliminados exitosamente." });

  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

};
```

6. Crea un endpoint POST /productos que permita registrar un nuevo producto enviando nombre, precio, stock y categoria_id para asociarlo a una categoría existente.

```
// ejercicio6. POST /productos

export const createProducto = async (req, res) => {
  const { nombre, precio, stock, categoria_id } = req.body;
  if (!nombre || !precio || !stock || !categoria_id) {
    return res.status(400).json({ message: "Faltan datos obligatorios (nombre, precio, stock, categoria_id)." });
  }
  try {
    const id = await producto.insertaProducto(nombre, precio, stock, categoria_id);
    res.status(201).json({ id, nombre, message: "Producto creado exitosamente." });
  } catch (error) {
    // Podría ser un error 500 o 400 si categoria_id no existe (Foreign Key Constraint)
    res.status(500).json({ error: error.message });
  }
};
```

7. Crea un endpoint GET /productos que devuelva todos los productos y muestre el nombre de la categoría a la que pertenece cada uno.

```
// ejercicio7. GET /productos
export const getProductos = async (req, res) => {
  try {
    const productos = await producto.obtenerTodosProductos();
    res.status(200).json(productos);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};
```

8. Crea un endpoint GET /productos/:id que devuelva la información de un producto por su ID, incluyendo el nombre de la categoría asociada.

```
// ejercicio8. GET /productos/:id
export const getProductoById = async (req, res) => {
  try {
    const producto = await producto.obtenerProductoPorId(req.params.id);
    if (!producto) return res.status(404).json({ message: "Producto no encontrado." });
    res.status(200).json(producto);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};
```

9. Crea un endpoint PUT /productos/:id que permita actualizar todos los datos de un producto, incluyendo la opción de cambiar la categoría a la que pertenece mediante categoria_id.

```
// ejercicio9. PUT /productos/:id
export const updateProducto = async (req, res) => {
  const { nombre, precio, stock, categoria_id } = req.body;
  if (!nombre || !precio || !stock || !categoria_id) {
    return res.status(400).json({ message: "Faltan datos obligatorios para la actualización." });
  }
  try {
    const affectedRows = await producto.actualizaProducto(req.params.id, nombre, precio, stock, categoria_id);
```

```

if (affectedRows === 0) return res.status(404).json({ message: "Producto no encontrado para actualizar." });
res.status(200).json({ message: "Producto actualizado exitosamente." });
} catch (error) {
  res.status(500).json({ error: error.message });
}
};


```

10. Crea un endpoint PATCH /productos/:id/stock que permita incrementar o decrementar el stock de un producto enviando al body la cantidad que se desea sumar o restar.

```

// ejercicio10. PATCH /productos/:id/stock

export const patchStock = async (req, res) => {
  const { cantidad } = req.body;
  // Valida que 'cantidad' sea un número válido y no cero
  if (typeof cantidad !== 'number' || cantidad === 0) {
    return res.status(400).json({ message: "Se requiere un valor numérico 'cantidad' (positivo o negativo) válido." });
  }
  try {
    const affectedRows = await producto.actualizaStock(req.params.id, cantidad);
    if (affectedRows === 0) return res.status(404).json({ message: "Producto no encontrado o stock no actualizado." });
    res.status(200).json({ message: `Stock actualizado en ${cantidad} unidades.` });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};


```