**Integrantes:**Jonathan Oviedo  
Andrés Lagos

La complejidad ciclomática se puede calcular usando la fórmula:

Donde:

* M es la complejidad ciclomática.
* E es el número de aristas en el grafo de control de flujo.
* N es el número de nodos en el grafo.
* P es el número de componentes conectados (normalmente 1 para una función individual).

Veamos las funciones en phonesController.js:

**getAllPhonesRPC**

javascript

Copiar código

async function getAllPhonesRPC(req, res) {

try {

const phones = await Phone.findAll();

res.json(phones);

} catch (error) {

res.status(500).json({ error: 'Internal Server Error' });

}

}

* Aristas (E): 4 (inicio, llamada a findAll, éxito, error)
* Nodos (N): 4 (inicio, findAll, éxito, error)
* Componentes conectados (P): 1

**getPhoneByIMEIRPC**

javascript

Copiar código

async function getPhoneByIMEIRPC(req, res) {

const { imei } = req.body;

try {

const phone = await Phone.findByPk(imei);

if (!phone) {

return res.status(404).json({ error: 'Phone not found' });

}

res.json(phone);

} catch (error) {

res.status(500).json({ error: 'Internal Server Error' });

}

}

* Aristas (E): 7
* Nodos (N): 6
* Componentes conectados (P): 1

**addPhoneRPC**

javascript

Copiar código

async function addPhoneRPC(req, res) {

const { model, brand, price } = req.body;

try {

const imei = 'IMEI-' + Math.floor(Math.random() \* 1000000000000);

await Phone.create({ imei, model, brand, price });

res.status(201).json({ message: 'Phone added successfully', imei });

} catch (error) {

res.status(500).json({ error: 'Internal Server Error' });

}

}

* Aristas (E): 5
* Nodos (N): 5
* Componentes conectados (P): 1

**updatePhoneRPC**

javascript

Copiar código

async function updatePhoneRPC(req, res) {

const { imei, model, brand, price } = req.body;

try {

const phone = await Phone.findByPk(imei);

if (!phone) throw new Error('Phone not found');

await phone.update({ model, brand, price });

res.json({ message: `Phone with IMEI ${imei} updated successfully` });

} catch (error) {

res.status(500).json({ error: 'Internal Server Error' });

}

}

* Aristas (E): 7
* Nodos (N): 6
* Componentes conectados (P): 1

**deletePhoneRPC**

javascript

Copiar código

async function deletePhoneRPC(req, res) {

const { imei } = req.body;

try {

const phone = await Phone.findByPk(imei);

if (!phone) throw new Error('Phone not found');

await phone.destroy();

res.json({ message: `Phone with IMEI ${imei} deleted successfully` });

} catch (error) {

res.status(500).json({ error: 'Internal Server Error' });

}

}

* Aristas (E): 7
* Nodos (N): 6
* Componentes conectados (P): 1

Para cada ruta independiente (calculada como M, necesitamos una prueba. A continuación, se muestra cómo se pueden mapear las pruebas:

**getAllPhonesRPC**

* Complejidad ciclomática: 2 (2 pruebas)
  + Ruta 1: Éxito en obtener todos los teléfonos.
  + Ruta 2: Error al obtener los teléfonos.

**getPhoneByIMEIRPC**

* Complejidad ciclomática: 3 (3 pruebas)
  + Ruta 1: Éxito en obtener el teléfono por IMEI.
  + Ruta 2: Teléfono no encontrado.
  + Ruta 3: Error en la base de datos.

**addPhoneRPC**

* Complejidad ciclomática: 2 (2 pruebas)
  + Ruta 1: Éxito en agregar un teléfono.
  + Ruta 2: Error al agregar un teléfono.

**updatePhoneRPC**

* Complejidad ciclomática: 3 (3 pruebas)
  + Ruta 1: Éxito en actualizar un teléfono.
  + Ruta 2: Teléfono no encontrado.
  + Ruta 3: Error en la base de datos.

**deletePhoneRPC**

* Complejidad ciclomática: 3 (3 pruebas)
  + Ruta 1: Éxito en eliminar un teléfono.
  + Ruta 2: Teléfono no encontrado.
  + Ruta 3: Error en la base de datos.