Soluciones:

Estados de una operación:

(1) pendiente: No cuenta para límites hasta ser resuelta.

(2) completado: Aplica para límites diarios.

(3)rechazado: No debe contar para los limites diarios.

(4) expirado: Estado automático después de 48h en pendiente. No cuenta para límites.

Límites Diarios para las categorías:

standard: 2 operaciones "completado" por día.

premium: 4 operaciones "completado".

vip: Sin límite, pero transacciones >5,500 USD requieren una revisión manual(Esto aplica para todas las categorias).

Colecciones:

operation

client

Caso 1:

Consulta realizada:

```
kambista> db.operation.find({ clientId: ObjectId("65a3f2007a1b4f9d3c8e2b70") })
[
 {
  _id: ObjectId('65a3f1f87a1b4f9d3c8e2b60'),
  isTransferred: true,
  clientId: ObjectId('65a3f2007a1b4f9d3c8e2b70'),
  amountDestination: 300,
  currency: 'USD',
  status: 2,
  date: ISODate('2024-03-28T09:00:00.000Z'),
  category: 'standard'
 },
 {
  _id: ObjectId('65a3f1f87a1b4f9d3c8e2b61'),
```

```
  isTransferred: true,
  clientId: ObjectId('65a3f2007a1b4f9d3c8e2b70'),
  amountDestination: 200,
  currency: 'USD',
  status: 2,
  date: ISODate('2024-03-28T10:30:00.000Z'),
  category: 'standard'
 }
]
```

65a3f2007a1b4f9d3c8e2b70 es categoría standard, por ello solo le permite 2 operaciones completadas por día. Actualmente ya tiene 2 operaciones completadas el 28 de marzo. Por eso ya no puede crear más el mismo día.

*No amerita crear ticket, ya que se encuentra dentro de las reglas del negocio.

Caso 2:

Consulta realizada:

```
db.operation.find({ clientId: ObjectId("65a3f2007a1b4f9d3c8e2b71") })
```

Quinta operación que no corresponde:

```
 _id: ObjectId('65a3f1f87a1b4f9d3c8e2b69'),
  isTransferred: false,
  clientId: ObjectId('65a3f2007a1b4f9d3c8e2b71'),
  amountDestination: 1000,
  currency: 'USD',
  status: 2,
  date: ISODate('2024-03-28T18:00:00.000Z'),
  category: 'premium'
```

Se detecta una quinta operación la cual no debería haberse creado, ya que el cliente es premium, solo puede hacer 4 operaciones diarias, las cuales ya hizo.

Se puede decir que debe tratarse de un error de los permisos de ese cliente, lo cual sí se requiere generar ticket para la revisión correspondiente.

Respuesta al equipo:

Chicos, vamos a revisar el caso para poder brindar una solución al cliente. De igual manera, consultar al cliente si desea completar su perfil para pasar al Vip y pueda hacer operaciones sin límite diario.

Caso 3:

Consulta realizada:

```
db.operation.aggregate([
 {
  $match: {
   status: { $in: [1, 2] }
  }
 },
 {
  $lookup: {
   from: "client",
   localField: "clientId",
   foreignField: "_id",
   as: "client"
  }
 },
{ $unwind: "$client" },
 {
  $addFields: {
   convertedAmountSoles: {
    $cond: [
     { $eq: ["$currency", "USD"] },
     { $multiply: ["$amountDestination", 3.78] },
     "$amountDestination"
```

```
      ]
    },
    manualReview: {
     $gt: ["$amountDestination", 5500]
    }
   }
  },
  {
   $group: {
    _id: "$clientId",
    name: { $first: "$client.name" },
    category: { $first: "$client.category" },
    totalOperaciones: { $sum: 1 },
    totalSoles: { $sum: "$convertedAmountSoles" },
    operaciones: {
     $push: {
      amount: "$amountDestination",
      moneda: "$currency",
      fecha: "$date",
      requiereRevision: "$manualReview"
     }
    }
   }
  }
]).pretty()
```

Caso 4:

Consulta realizada:

```
db.operation.updateMany(
 {
  status: 1,
```

```
  date: { $lt: new Date(Date.now() - 48 * 60 * 60 * 1000) }
 },
 {
  $set: { status: 4 } // 4 = expirado
 }
)
```

Actualizar categoría

```
db.client.updateOne(
 { _id: ObjectId("65a3f2007a1b4f9d3c8e2b91") },
 { $set: { category: "standard" } }
)
```

*Se busca actualizar los clientes que tienen estatus 1 para cambiarlos a estatus 4.

*Se procede a cambiar la categoría de los clientes que tienen 2 o más operaciones expiradas.

*Se procede a crear script para simplificar lo realizado.

Caso 5:

Titulo

Transacciones en PEN no disparan revisión manual al superar equivalente a 5,500 USD

Descripción del Problema

El sistema marca para revisión manual aquellas operaciones cuya equivalencia en USD supera los 5,500 dólares. Sin embargo, en las transacciones en PEN (currency: "PEN"), esta validación se ejecuta antes de calcular convertedAmount, por lo tanto, el chequeo falla, aunque el monto convertido supere el umbral.

Query para identificar el error

```
db.operation.find({
 currency: "PEN",
 status: 2,
 convertedAmount: { $gt: 5500 },
 $or: [
```

```
  { manualReview: { $exists: false } },
  { manualReview: false }
 ]
});
```

Solución temporal

```
db.operation.updateMany(
 {
  currency: "PEN",
  status: 2,
  convertedAmount: { $gt: 5500 },
  $or: [
   { manualReview: { $exists: false } },
   { manualReview: false }
  ]
 },
 {
  $set: { manualReview: true }
 }
);
```

Solución propuesta:

1. Ajustar el orden de validaciones en el backend

Mover la validación de manualReview después de calcular el convertedAmount, asegurando que se revise el monto ya convertido, no el original.

2. Bases de datos afectadas:

kambista (base MongoDB principal usada por operaciones y clientes)

3. Servicios afectados:

Servicio de procesamiento de operaciones

Servicio de validación de límites (back-end)

4. Plataforma afectada:

Plataforma de cambio web (front) y móvil (si usa la misma lógica del backend)

Caso 6:

Orden de prioridad:

1. Error 500 en MongoDB, bloquea el acceso para todos los usuarios.

2. Transacciones completadas no reflejadas, ya que los clientes no pueden visualizar su cambio.

3. Script fallido que degradó 1,200 usuarios, manejable ya que no tienen operaciones en curso.

1. Preguntar:

¿Está MongoDB corriendo correctamente?

¿Hubo reinicio del servidor?

2. Preguntar:

- ¿Cuáles son los DNI de los afectados?

- ¿Pedir el estado de la operación en Mongodb?

- ¿Verificar si hay salida en el banco o hay problemas con el banco?

- ¿Si el cliente pudo entrar nuevamente a revisar sus balances?

3. Preguntar:

-¿Cuál fue el último cambio que se hizo en la db?

-¿Si hay forma de regresar a una versión anterior?

-¿Hay clientes críticos que desean operar?

Script del caso 4:

```
const { MongoClient, ObjectId } = require("mongodb");

const XLSX = require("xlsx");



async function main() {
```

```
const uri = "mongodb://admin:secret@localhost:27017";
const client = new MongoClient(uri);

try {
 await client.connect();
 const db = client.db("kambista");
 const operation = db.collection("operation");
 const clientColl = db.collection("client");

 // 1. Actualizar operaciones pendientes > 48h a expiradas
 const expiradas = await operation.find({
  status: 1,
  date: { $lt: new Date(Date.now() - 48 * 60 * 60 * 1000) }
 }).toArray();

 const expiredIds = expiradas.map(op => op._id);
 if (expiredIds.length > 0) {
  await operation.updateMany(
   { _id: { $in: expiredIds } },
   { $set: { status: 4 } }
  );
 }

 console.log(` 🔄 Operaciones actualizadas a expiradas: ${expiredIds.length}\n`);

 // 2. Obtener reporte de operaciones expiradas
 const operacionesExpiradas = expiradas.map(op => ({
  idOperacion: op._id.toString(),
  montoDestino: op.amountDestination,
  moneda: op.currency,
  estadoExpirada: true
```

```
}));

console.log("📄 Reporte de Operaciones Expiradas:");
console.table(operacionesExpiradas);

// 3. Verificar clientes con 2 o más expiradas y degradarlos
const expiradasPorCliente = await operation.aggregate([
 { $match: { status: 4 } },
 { $group: { _id: "$clientId", total: { $sum: 1 } } },
 { $match: { total: { $gte: 2 } } }
]).toArray();

const clientesDegradados = [];

for (const cliente of expiradasPorCliente) {
 const datos = await clientColl.findOne({ _id: cliente._id });
 if (datos && datos.category !== "standard") {
  await clientColl.updateOne(
   { _id: cliente._id },
   { $set: { category: "standard" } }
  );
  clientesDegradados.push({
   idCliente: cliente._id.toString(),
   nombre: datos.name,
   nivelAnterior: datos.category,
   nivelActual: "standard"
  });
 }
}

console.log("\n📄 Reporte de Clientes Degradados:");
```

```
  console.table(clientesDegradados);

  // 4. Exportar ambos reportes a Excel
  const wb = XLSX.utils.book_new();
  const hojaOperaciones = XLSX.utils.json_to_sheet(operacionesExpiradas);
  const hojaClientes = XLSX.utils.json_to_sheet(clientesDegradados);

  XLSX.utils.book_append_sheet(wb, hojaOperaciones, "Operaciones Expiradas");
  XLSX.utils.book_append_sheet(wb, hojaClientes, "Clientes Degradados");

  XLSX.writeFile(wb, "Reporte_Caso4.xlsx");
  console.log("\n✅ Archivo Excel generado: Reporte_Caso4.xlsx");

 } catch (err) {
  console.error("❌ Error ejecutando el caso 4:", err);
 } finally {
  await client.close();
 }
}

main();
```

```
🔍 Reporte de Operaciones Expiradas:
┌─────────┬──────────────────────────┬──────────────┬──────────┬─────────────────┐
│ (index) │ idOperacion              │ montoDestino │ moneda   │ estadoExpirada  │
├─────────┼──────────────────────────┼──────────────┼──────────┼─────────────────┤
│ 0       │ '65a3f1f87a1b4f9d3c8e2b03' │ 4000         │ 'USD'    │ true            │
│ 1       │ '65a3f1f87a1b4f9d3c8e2b04' │ 8000         │ 'PEN'    │ true            │
│ 2       │ '65a3f1f87a1b4f9d3c8e2b05' │ 9000         │ 'PEN'    │ true            │
└─────────┴──────────────────────────┴──────────────┴──────────┴─────────────────┘

🔍 Reporte de Clientes Degradados:
┌─────────┬──────────────────────────┬──────────────┬─────────────────┬──────────────┐
│ (index) │ idCliente                │ nombre       │ nivelAnterior   │ nivelActual  │
├─────────┼──────────────────────────┼──────────────┼─────────────────┼──────────────┤
│ 0       │ '65a3f2007a1b4f9d3c8e2b91' │ 'Albert Hammond' │ 'premium'       │ 'standard'   │
└─────────┴──────────────────────────┴──────────────┴─────────────────┴──────────────┘

✅ Archivo Excel generado: Reporte_Caso4.xlsx
PS C:\Users\KAMBISTA\Desktop\tec>
```