Entrega 3 Proyecto Final

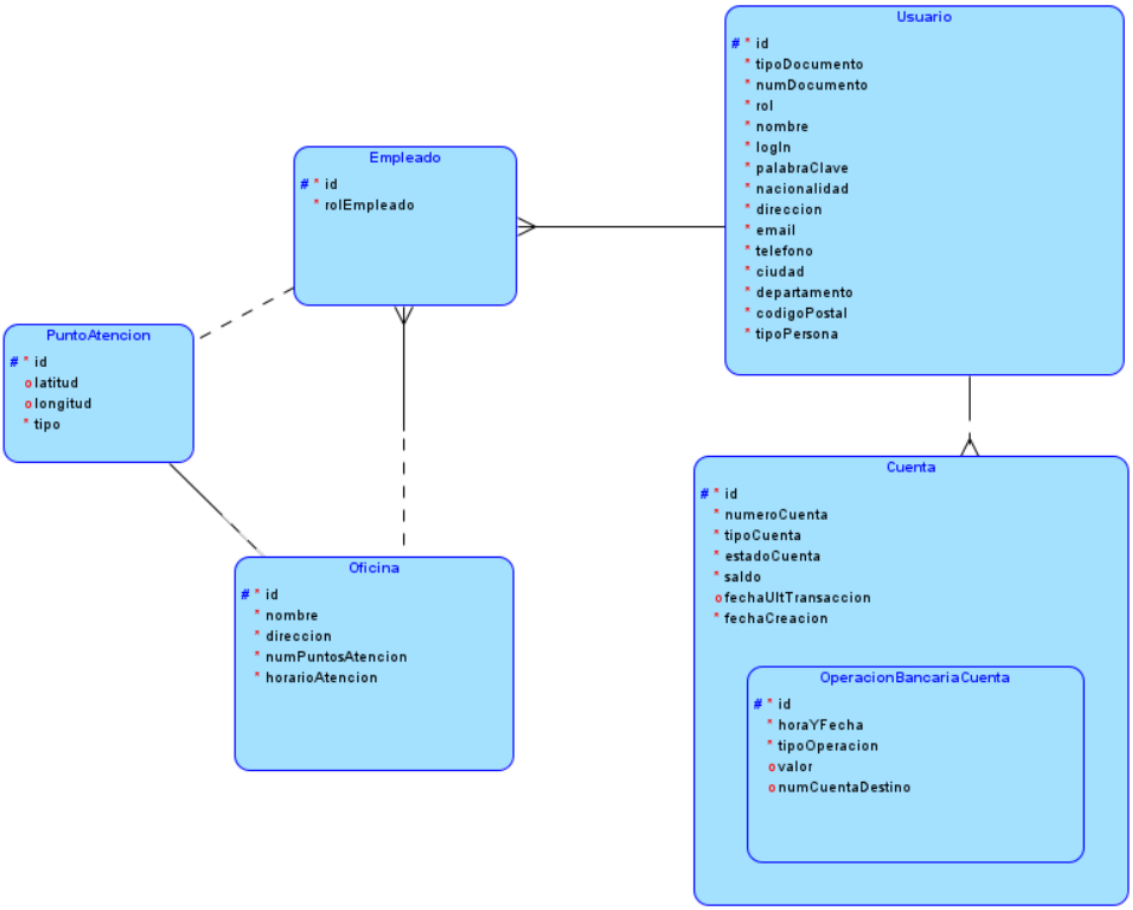
Grupo 8

Alejandro Pardo - 202223709

Sebastián Rodríguez - 202221424

Daniel Neira – 201816386

Modelo E/R



Entidades y atributos

1. PuntoAtencion

- id
- altitud
- longitud
- tipo

2. Empleado

- id
- rolEmpleado

3. Oficina

- id
- nombre
- direccion
- numPuntosAtencion
- horarioAtencion

4. Usuario

- id
- tipoDocumento
- numDocumento
- nombre
- apellido
- palabraClave
- nacionalidad
- direccion
- telefono
- email
- codigoPostal
- tipoPersona

5. Cuenta

- id
- numeroCuenta
- tipoCuenta
- estadoCuenta
- saldoCuenta
- fechaUltTransaccion
- fechaCreacion

6. OperacionBancariaCuenta (enmebida en cuenta)

- id
- horaYFecha

- tipoOperacion
- valor
- NumCuentaDestino

Análisis de Operaciones por Entidad

Entidades	Operaciones	Info Requerida	Tipo
Usuario	Crear usuario	Datos del usuario (id, tipoDocumento, numDocumento, nombre, apellido, palabraClave, nacionalidad, direccion, telefono, email, codigoPostal, tipoPersona)	Write
	Consultar detalles del usuario	Datos del usuario	Read
Oficina	Crear oficina	Nombre, direccion, numPuntosAtencion, horarioAtencion	Write
	Consultar detalles de la oficina	Detalles de la oficina	Read
PuntoAtencion	Crear punto de atención	Tipo, localización geográfica, id de la oficina (si aplica)	Write
	Borrar punto de atención	Id del punto de atención	Write
	Consultar detalles del punto de atención	Detalles del punto de atención	Read
Cuenta	Crear cuenta	Tipo, cliente, número de cuenta, saldo, fecha de última transacción, estado	Write
	Cambiar estado de cuenta	Estado, id de la cuenta	Write
	Consultar detalles de la cuenta	Detalles de la cuenta	Read
	Consultar cuentas con filtros	Tipo de cuenta, rango de saldos, fecha de creación, fecha del	Read

		último movimiento, cliente	
OperacionBancariaCuenta	Registrar operación sobre cuenta	Fecha de la operación, número de la cuenta afectada, monto de la operación, tipo de operación, saldo actualizado	Write
	Consultar extracto bancario	Número de cuenta, mes	Read
Empleado	Crear empleado	Datos del empleado (id, rolEmpleado)	Write

Cuantificación de Operaciones por Entidad

Entidades	Operaciones	Info requerida	Tipo	Rate
Usuario	Crear usuario	Datos del usuario (id, tipoDocumento, numDocumento, nombre, apellido, palabraClave, nacionalidad, dirección, teléfono, email, códigoPostal, tipoPersona)	Write	200/día
Usuario	Consultar detalles del usuario	Datos del usuario	Read	500/día
Oficina	Crear oficina	Nombre, dirección, numPuntosAtencion, horarioAtencion	Write	1/mes
Oficina	Consultar detalles de la oficina	Detalles de la oficina	Read	1/semana
PuntoAtencion	Crear punto de atención	Tipo, localización geográfica, id de la oficina (si aplica)	Write	1/mes
PuntoAtencion	Consultar detalles del punto de atención	Detalles del punto de atención	Read	1/semana
Cuenta	Crear cuenta	Tipo, cliente, número de cuenta, saldo, fecha de	Write	500/día

		última transacción, estado		
Cuenta	Cambiar estado de cuenta	Estado, id de la cuenta	Write	500/ día
Cuenta	Consultar detalles de la cuenta	Detalles de la cuenta	Read	5000/ día
OperacionBancariaCuenta	Registrar operación sobre cuenta	Fecha de la operación, número de la cuenta afectada, monto de la operación, tipo de operación, saldo actualizado	Write	20000/ día
OperacionBancariaCuenta	Consultar operaciones de la cuenta	Número de cuenta, mes	Read	5000/ día
Empleado	Crear empleado	Datos del empleado (id, rolEmpleado)	Write	1/mes
Empleado	Consultar detalles del empleado	Datos del empleado (id, rolEmpleado)	Read	1/semana

Descripción de Entidades y Relaciones con su Cardinalidad

1. Usuario - Cuenta
    - Relación: Un usuario puede tener una o varias cuentas.
    - Cardinalidad: Uno a muchos
2. Oficina - Empleado
    - Relación: Una oficina puede tener uno o varios empleados.
    - Cardinalidad: Uno a muchos

3. Oficina - PuntoAtencion
    - Relación: Una oficina puede tener uno o varios puntos de atención.
    - Cardinalidad: Uno a muchos

4. PuntoAtencion - Empleado
    - Relación: Un punto de atención puede está atendido por un empleado
    - Cardinalidad: Uno a uno

5. Cuenta - OperacionBancariaCuenta
    - Relación: Una cuenta puede tener una o varias operaciones bancarias.
    - Cardinalidad: Uno a muchos

6. Empleado- Usuario
    - Relación: Un usuario puede ser asociado con un único empleado y viceversa.
    - Cardinalidad: Uno a uno

Análisis de Selección de Esquema de Asociación

Utilizando los criterios y pautas de la siguiente tabla:

ANEXO C – TABLA DE ANÁLISIS DE ESQUEMA DE RELACIÓN ENTRE ENTIDDADES

Guideline Name	Question	Embed	Reference
Simplicity	Would keeping the pieces of information together lead to a simpler data model and code?	Yes	No
Go Together	Do the pieces of information have a "has-a," "contains," or similar relationship?	Yes	No
Query Atomicity	Does the application query the pieces of information together?	Yes	No
Update Complexity	Are the pieces of information updated together?	Yes	No
Archival	Should the pieces of information be archived at the same time?	Yes	No
Cardinality	Is there a high cardinality (current or growing) in the child side of the relationship?	No	Yes
Data Duplication	Would data duplication be too complicated to manage and undesired?	No	Yes
Document Size	Would the combined size of the pieces of information take too much memory or transfer bandwidth for the application?	No	Yes
Document Growth	Would the embedded piece grow without bound?	No	Yes
Workload	Are the pieces of information written at different times in a write-heavy workload?	No	Yes
Individuality	For the children side of the relationship, can the pieces exist by themselves without a parent?	No	Yes

1. Usuario-Cuenta:
    - Referenciado
    - Justificación:

        - Simplicity: Mantener los datos del usuario y de la cuenta por separado lleva a un modelo de datos y código más claro y manejable.
        - Go Together: Aunque tienen una relación de "tiene-un", pueden mantenerse separados sin perder consistencia.
        - Query Atomicity: La aplicación puede consultar los datos del usuario y de la cuenta de manera independiente según sea necesario.
        - Update Complexity: Los datos del usuario y de la cuenta no siempre se actualizan juntos, permitiendo actualizaciones independientes.
        - Archival: Los datos del usuario y de la cuenta pueden archivarse por separado.
        - Cardinality: Existe una alta cardinalidad en el lado hijo (cuenta) de la relación.
        - Data Duplication: La duplicación de datos sería complicada y no deseada.
        - Document Size: El tamaño combinado de los datos podría ser grande si se embeben.
        - Document Growth: El crecimiento de los documentos podría ser ilimitado.
        - Workload: Los datos del usuario y de la cuenta pueden ser escritos en diferentes momentos, especialmente en escenarios con alta carga de trabajo.
        - Individuality: Las cuentas pueden existir independientemente del usuario en la base de datos.

2. Oficina-PuntoAtencion:
    - Referenciado
    - Justificación:
        - Simplicity: Mantener los datos de la oficina y del punto de atención por separado lleva a un modelo de datos y código más claro y manejable.
        - Go Together: Aunque tienen una relación de "tiene-un", pueden mantenerse separados sin perder consistencia.
        - Query Atomicity: La aplicación puede consultar los datos de la oficina y del punto de atención de manera independiente según sea necesario.

        - Update Complexity: Los datos de la oficina y del punto de atención no siempre se actualizan juntos, permitiendo actualizaciones independientes.
        - Archival: Los datos de la oficina y del punto de atención pueden archivarse por separado.
        - Cardinality: Existe una alta cardinalidad en el lado hijo (punto de atención) de la relación.
        - Data Duplication: La duplicación de datos sería complicada y no deseada.
        - Document Size: El tamaño combinado de los datos podría ser grande si se embeben.
        - Document Growth: El crecimiento de los documentos podría ser ilimitado.
        - Workload: Los datos de la oficina y del punto de atención pueden ser escritos en diferentes momentos, especialmente en escenarios con alta carga de trabajo.
        - Individuality: Los puntos de atención pueden existir independientemente de la oficina en la base de datos.

3. Oficina-Empleado:
    - Referenciado
    - Justificación:
        - Simplicity: Mantener los datos de la oficina y del empleado por separado lleva a un modelo de datos y código más claro y manejable.
        - Go Together: Aunque tienen una relación de "tiene-un", pueden mantenerse separados sin perder consistencia.
        - Query Atomicity: La aplicación puede consultar los datos de la oficina y del empleado de manera independiente según sea necesario.
        - Update Complexity: Los datos de la oficina y del empleado no siempre se actualizan juntos, permitiendo actualizaciones independientes.
        - Archival: Los datos de la oficina y del empleado pueden archivarse por separado.
        - Cardinality: Existe una alta cardinalidad en el lado hijo (empleado) de la relación.
        - Data Duplication: La duplicación de datos sería complicada y no deseada.

        - Document Size: El tamaño combinado de los datos podría ser grande si se embeben.
        - Document Growth: El crecimiento de los documentos podría ser ilimitado.
        - Workload: Los datos de la oficina y del empleado pueden ser escritos en diferentes momentos, especialmente en escenarios con alta carga de trabajo.
        - Individuality: Los empleados pueden existir independientemente de la oficina en la base de datos.
        -

4. PuntoAtencion-Empleado:
    - Referenciado
    - Justificación:
        - Simplicity: Mantener separados los datos del punto de atención y del empleado puede llevar a un modelo de datos y código más claro.
        - Go Together: Aunque tienen una relación de "tiene-un", pueden mantenerse separados sin perder consistencia.
        - Query Atomicity: La aplicación puede consultar los datos del punto de atención y del empleado por separado según sea necesario.
        - Update Complexity: Los datos del punto de atención y del empleado no siempre se actualizan juntos, permitiendo actualizaciones independientes.
        - Archival: Los datos del punto de atención y del empleado pueden archivarse por separado.
        - Cardinality: No hay alta cardinalidad en el lado hijo (empleado) de la relación.
        - Data Duplication: La duplicación de datos se evita manteniéndolos referenciados.
        - Document Size: El tamaño combinado de los datos no afecta significativamente la memoria ni el ancho de banda.
        - Document Growth: No hay preocupación por el crecimiento ilimitado de los documentos embebidos.
        - Workload: Los datos del punto de atención y del empleado pueden escribirse en diferentes momentos según la carga de trabajo.
        - Individuality: Las piezas de información pueden existir por sí solas sin necesidad de la otra.

5. Cuenta-OperacionBancariaCuenta:
    - Embebido
    - Justificación:
        - Simplicity: Mantener juntas las operaciones bancarias y la cuenta lleva a un modelo de datos y código más simple.
        - Go Together: Las operaciones bancarias tienen una relación de "tiene-un" con la cuenta.
        - Query Atomicity: Lo que se actualiza junto se consulta junto.
        - Update Complexity: Las operaciones bancarias se actualizan junto con la cuenta.
        - Archival: Las operaciones bancarias se deben archivar junto con la cuenta.
        - Cardinality: No hay alta cardinalidad en el lado hijo (operacion) de la relación.
        - Data Duplication: La duplicación de datos no es un problema en este caso.
        - Document Size: El tamaño combinado de las piezas de información no toma demasiado memoria o ancho de banda.
        - Document Growth: Las piezas embebidas no crecerán sin límite.
        - Workload: Las piezas de información se escriben juntas en la misma carga de trabajo.
        - Individuality: Las operaciones bancarias no pueden existir por sí mismas sin la cuenta.
6. Empleado-Usuario:
    - Referenciado
    - Justificación:
        - Simplicity: Mantener los datos del empleado y del usuario por separado lleva a un modelo de datos y código más claro y manejable.
        - Go Together: Aunque tienen una relación de "tiene-un", pueden mantenerse separados sin perder consistencia.
        - Query Atomicity: La aplicación puede consultar los datos del empleado y del usuario de manera independiente según sea necesario.
        - Update Complexity: Los datos del empleado y del usuario no siempre se actualizan juntos, permitiendo actualizaciones independientes.
        - Archival: Los datos del empleado y del usuario pueden archivarse por separado.
        - Cardinality: La relación es uno a uno, por lo que la cardinalidad no presenta un problema significativo.

- Data Duplication: La duplicación de datos sería complicada y no deseada.
- Document Size: El tamaño combinado de los datos podría ser grande si se embeben.
- Document Growth: El crecimiento de los documentos podría ser sin límite si se embeben.
- Workload: Los datos del empleado y del usuario pueden ser escritos en diferentes momentos, especialmente en escenarios con alta carga de trabajo.
- Individuality: Los empleados y los usuarios pueden existir independientemente uno del otro en la base de datos.

Descripción Gráfica en JSON de las Relaciones entre Entidades

- Usuario-Cuenta:

```
// Documento de usuario
{
   "_id": "<ObjectId1>",
   "tipoDocumento": "CC",
   "numDocumento": "123456789",
   "nombre": "Juan",
.....
 }

// Documento de cuenta
{
   "_id": "<ObjectId2>",
   "usuario_id": "<ObjectId1>",
   "numeroCuenta": "987654321",
   "tipoCuenta": "Ahorros",
.....
  }
```

- Oficina-Empleado:

```
// Documento de oficina
{
   "_id": "<ObjectId1>",
   "nombre": "Oficina Centro",
....
 }
```

// Documento de empleado
{
"_id": "<ObjectId2>",
"oficina_id": "<**ObjectId1**>",
"rolEmpleado": "Gerente",
....
....
}

- Oficina-PuntoAtencion:

// Documento de oficina
{
"_id": "<**ObjectId1**>",
"nombre": "Oficina Norte",
....
}

// Documento de punto de atención
{
"_id": "<ObjectId2>",
"oficina_id": "<**ObjectId1**>",
"altitud": "4.60971",
....
}

- Empleado-Usuario:

// Documento de usuario
{
"_id": "<**ObjectId1**>",
"tipoDocumento": "CC",
....
}

// Documento de empleado
{
"_id": "<ObjectId2>",
"usuario_id": "<**ObjectId1**>",
"rolEmpleado": "Cajero",
...

}

- Cuenta-OperacionBancariaCuenta:

{
  "_id": "cuenta001",
  "numeroCuenta": "123456789",
  "tipoCuenta": "Ahorros",
  "estadoCuenta": "Activa",
  "saldoCuenta": 1000000,
  "fechaUltTransaccion": "2024-05-25",
  "fechaCreacion": "2022-01-01",
  "operacionesBancarias": [
    {
      "id": "operacion001",
      "horaYFecha": "2024-05-24T14:00:00",
      "tipoOperacion": "Depósito",
      "valor": 500000,
      "numCuentaDestino": null
    },
    {
      "id": "operacion002",
      "horaYFecha": "2024-05-25T10:00:00",
      "tipoOperacion": "Retiro",
      "valor": 200000,
      "numCuentaDestino": null
    },
    {
      "id": "operacion003",
      "horaYFecha": "2024-05-25T12:00:00",
      "tipoOperacion": "Transferencia",
      "valor": 100000,
      "numCuentaDestino": "987654321"
    }
  ]
}

Esquema de validación

USUARIOS

```
db.createCollection("usuarios", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["numeroDocumento", "tipoDocumento", "rol", "nombre", "login", "palabraClave", "nacionalidad", "direccion", "correoElectronico", "telefono", "ciudad", "departamento", "codigoPostal", "tipoPersona"],
      properties: {
        numeroDocumento: {
          bsonType: "string",
          description: "must be a string and is required",
          maxLength: 10
        },
        tipoDocumento: {
          bsonType: "string",
          enum: ["Cedula", "Tarjeta de Identidad", "Pasaporte", "Cedula extranjera"],
          description: "can only be one of the enum values and is required"
        },
        rol: {
          bsonType: "string",
          enum: ["Empleado", "Cliente", "Administrador"],
          description: "can only be one of the enum values and is required"
        },
        nombre: {
          bsonType: "string",
          description: "must be a string and is required"
```

```
},
login: {
  bsonType: "string",
  description: "must be a string and is required"
},
palabraClave: {
  bsonType: "string",
  description: "must be a string and is required"
},
nacionalidad: {
  bsonType: "string",
  description: "must be a string and is required"
},
direccion: {
  bsonType: "string",
  description: "must be a string and is required"
},
correoElectronico: {
  bsonType: "string",
  description: "must be a string and is required"
},
telefono: {
  bsonType: "string",
  description: "must be a string and is required",
  maxLength: 10
},
ciudad: {
  bsonType: "string",
```

```
          description: "must be a string and is required"
        },
        departamento: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        codigoPostal: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        tipoPersona: {
          bsonType: "string",
          enum: ["Natural", "Juridica"],
          description: "can only be one of the enum values and is required"
        }
      }
    }
  }
});
```

OFICINAS

```
db.createCollection("oficinas", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nombre", "direccion", "numeroPuntosAtencion", "horarioDeAtencion"],
      properties: {
        nombre: {
```

```
          bsonType: "string",
          description: "must be a string and is required"
        },
        direccion: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        numeroPuntosAtencion: {
          bsonType: "int",
          minimum: 1,
          description: "must be an integer greater than 0 and is required"
        },
        horarioDeAtencion: {
          bsonType: "string",
          description: "must be a string and is required"
        }
      }
    }
  }
});
```

PUNTOS DE ATENCION

```
db.createCollection("puntosDeAtencion", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
```

```
      required: ["tipo"],
      properties: {
        latitud: {
          bsonType: ["double", "null"],
          description: "must be a double or null"
        },
        longitud: {
          bsonType: ["double", "null"],
          description: "must be a double or null"
        },
        tipo: {
          bsonType: "string",
          enum: ["Personalizado", "Cajero Automatico", "Portal Web"],
          description: "can only be one of the enum values and is required"
        },
        idOficina: {
          bsonType: ["objectId", "null"],
          description: "must be an objectId or null"
        }
      }
    }
  }
});
```

EMPLEADOS

```
db.createCollection("empleados", {
  validator: {
```

```
    $jsonSchema: {
      bsonType: "object",
      required: ["id_usuario", "rolEmpleado"],
      properties: {
        id_usuario: {
          bsonType: "objectId",
          description: "must be an objectId and is required"
        },
        rolEmpleado: {
          bsonType: "string",
          enum: ["Gerente general", "Gerente oficina", "Cajero", "Asesores"],
          description: "can only be one of the enum values and is required"
        },
        id_oficina: {
          bsonType: ["objectId", "null"],
          description: "must be an objectId or null"
        },
        id_punto_de_atencion: {
          bsonType: ["objectId", "null"],
          description: "must be an objectId or null"
        }
      }
    }
  }
});
```

CUENTAS

```
db.createCollection("cuentas", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["numeroCuenta", "id_usuario", "tipoCuenta", "estadoCuenta", "saldo", "fechaCreacion", "operacionesBancarias"],
      properties: {
        numeroCuenta: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        id_usuario: {
          bsonType: "objectId",
          description: "must be an objectId and is required"
        },
        tipoCuenta: {
          bsonType: "string",
          enum: ["Ahorros", "Corriente", "AFC"],
          description: "can only be one of the enum values and is required"
        },
        estadoCuenta: {
          bsonType: "string",
          enum: ["Activa", "Cerrada", "Desactivada"],
          description: "can only be one of the enum values and is required"
        },
        saldo: {
          bsonType: "number",
          description: "must be a number and is required"
```

```
},
fechaUltimaTransaccion: {
  bsonType: "date",
  description: "must be a date"
},
fechaCreacion: {
  bsonType: "date",
  description: "must be a date and is required"
},
operacionesBancarias: {
  bsonType: "array",
  description: "must be an array of objects",
  items: {
    bsonType: "object",
    required: ["fecha", "hora", "tipoOperacion", "saldoResultante"],
    properties: {
      fecha: {
        bsonType: "date",
        description: "must be a date and is required"
      },
      hora: {
        bsonType: "string",
        description: "must be a string and is required"
      },
      tipoOperacion: {
        bsonType: "string",
        enum: ["Apertura", "Cierre", "Consignar", "Transferir", "Retirar"],
        description: "can only be one of the enum values and is required"
```

```
            },
            valor: {
              bsonType: ["number", "null"],
              description: "must be a number or null"
            },
            numeroCuentaDestino: {
              bsonType: ["string", "null"],
              description: "must be a string or null"
            },
            saldoResultante: {
              bsonType: "number",
              description: "must be a number and is required"
            }
          }
        }
      }
    }
  }
});
```

Escenarios de prueba

Scirpt que incumple esquemas de validación:

- Usuarios:

```
db.usuarios.insertOne({
  numeroDocumento: "123456789012", // excede el maxLength de 10
  tipoDocumento: "Licencia", // no está en el enum permitido
  rol: "Cliente",
```

```
  nombre: "Juan Perez",
  login: "jperez",
  palabraClave: "1234",
  nacionalidad: "Colombiana",
  direccion: "Calle 123",
  correoElectronico: "jperez@example.com",
  telefono: "1234567890",
  ciudad: "Bogotá",
  departamento: "Cundinamarca",
  codigoPostal: "110111",
  tipoPersona: "Natural"
});
```

- Oficina:

```
db.oficinas.insertOne({
  nombre: "Oficina Central",
  direccion: "Avenida Principal 123",
  numeroPuntosAtencion: 0, // menor que el mínimo de 1
  horarioDeAtencion: "8am - 5pm"
});
```

- Punto de Atencion:

```
db.puntosDeAtencion.insertOne({
  latitud: 4.676,
  longitud: -74.045,
  tipo: "Atencion Remota", // no está en el enum permitido
  idOficina: null
});
```

- Empleado:

```
db.empleados.insertOne({
```

```
  id_usuario: "string_instead_of_objectId", // no es un ObjectId
  rolEmpleado: "Recepcionista", // no está en el enum permitido
  id_oficina: null,
  id_punto_de_atencion: null
});
```

- Cuenta:

```
db.cuentas.insertOne({
  numeroCuenta: "1234567890",
  id_usuario: ObjectId("60b8d6af0c073724b48864f1"),
  tipoCuenta: "Corriente",
  estadoCuenta: "Activa",
  saldo: "mil pesos", // no es un número
  fechaUltimaTransaccion: "2022-06-01T00:00:00Z", // incorrecta fecha ISO-8601 para date
  fechaCreacion: new Date("2022-06-01"),
  operacionesBancarias: [
    {
      fecha: new Date("2022-06-01"),
      hora: "14:00",
      tipoOperacion: "Depositar", // no está en el enum permitido
      valor: 100000,
      numeroCuentaDestino: "9876543210",
      saldoResultante: 100000
    }
  ]
});
```