Snigdha Ghosh(23ICS10004)

Practice 3 - Backend

Account Transfer System with Balance Validation in Node.js

Objective: Learn how to implement a secure money transfer API in Node.js and MongoDB without using database transactions. This helps you understand dependent multi-document updates, balance validation, and proper error handling.

Concept Overview:

In financial systems, fund transfers require consistent updates to multiple user accounts. This exercise teaches logical validation and sequential updates to ensure accuracy without database transactions.

Steps / Procedure:

Step 1: Initialize Project

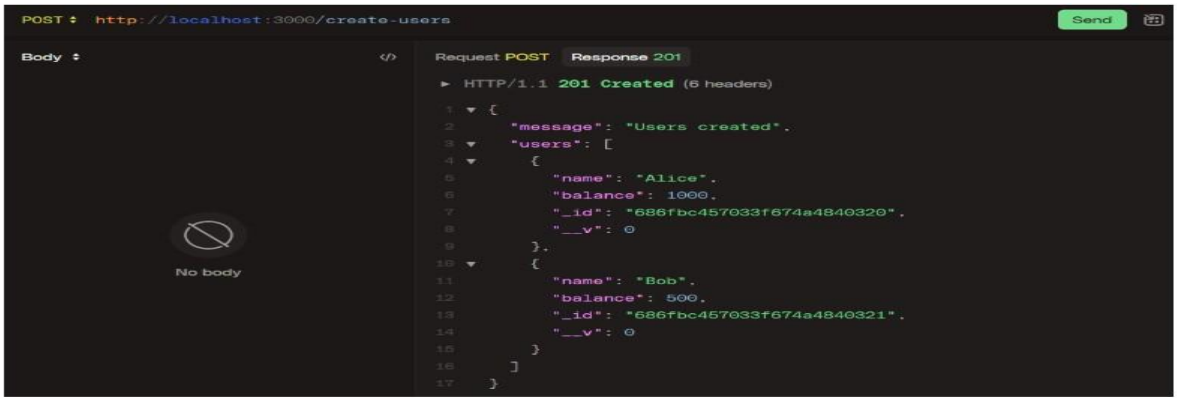
```
mkdir account-transfer-system
cd account-transfer-system
npm init -y
npm install express mongoose body-parser
```

Step 2: Create server.js

```
const express = require('express'); const
mongoose = require('mongoose'); const
bodyParser = require('body-parser'); const
app = express(); const PORT = 3000;

app.use(bodyParser.json());

mongoose.connect('mongodb://localhost:27017/bankDB', {
useNewUrlParser: true,   useUnifiedTopology: true });

const userSchema = new mongoose.Schema({
name: String,   balance: Number });

const User = mongoose.model('User', userSchema);

app.post('/create-users', async (req, res) => {
try {     await User.deleteMany({});     const
users = await User.insertMany([       { name:
'Alice', balance: 1000 },
      { name: 'Bob', balance: 500 }     ]);
res.status(201).json({ message: 'Users created', users });   }
catch (err) {     res.status(500).json({ message: 'Error
creating users' });   } });

app.post('/transfer', async (req, res) => {   try {
const { fromUserId, toUserId, amount } = req.body;
const sender = await User.findById(fromUserId);
const receiver = await User.findById(toUserId);
```

```
    if (!sender || !receiver) {        return
res.status(404).json({ message: 'User not found' });      }

    if (sender.balance < amount) {        return res.status(400).json({
message: 'Insufficient balance' });      }

    sender.balance -= amount;
receiver.balance += amount;

    await sender.save();
await receiver.save();

    res.status(200).json({        message: `Transferred $${amount} from
${sender.name} to ${receiver.name}`,        senderBalance: sender.balance,
receiverBalance: receiver.balance      });    } catch (error) {
res.status(500).json({ message: 'Transfer failed', error: error.message });    }
});

app.listen(PORT, () => {   console.log(`Server running on
http://localhost:${PORT}`); });
```

Expected Output:



Result:

The system successfully transfers funds between users with proper validation and error handling. Logical checks ensure consistent data even without transactions.