Experiment 7.3

Account Transfer System with Balance Validation in Node.js

Aim: *To implement a secure money transfer API in Node.js and MongoDB without using database transactions, ensuring logical correctness through balance validation and sequential updates.*

Tools Used: - **Node.js** - **Express.js** - **MongoDB** - **Mongoose** - **Postman** *(for testing API)* - **npm (Node Package Manager)**

Procedure: 1. **Initialize a Node.js project** and install dependencies: `express`, `mongoose`, `body-parser`. 2. **Connect to MongoDB** using Mongoose. 3. **Create a user schema** with fields: `username`, `balance`. 4. **Seed the database** with sample user accounts. 5. **Implement an API endpoint** `/transfer` that accepts sender, receiver, and amount. 6. **Validate the sender's balance** before performing the transfer. 7. **Update sender and receiver balances sequentially** if the validation passes. 8. **Handle errors** for insufficient balance or non-existent accounts. 9. **Test the API using Postman** to demonstrate both successful and failed transfers.

Code:

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

const app = express();
app.use(bodyParser.json());

// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/bank', { useNewUrlParser: true,
useUnifiedTopology: true });

// User Schema
const userSchema = new mongoose.Schema({
  username: String,
  balance: Number
});
const User = mongoose.model('User', userSchema);

// Transfer endpoint
app.post('/transfer', async (req, res) => {
  const { sender, receiver, amount } = req.body;

  const senderUser = await User.findOne({ username: sender });
```

```
  const receiverUser = await User.findOne({ username: receiver });

  if (!senderUser || !receiverUser) {
    return res.status(404).json({ message: 'Sender or receiver account not
found' });
  }

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

  // Sequential update
  senderUser.balance -= amount;
  receiverUser.balance += amount;

  await senderUser.save();
  await receiverUser.save();

  res.json({ message: `Transferred ${amount} from ${sender} to ${receiver}` });
});

app.listen(3000, () => console.log('Server running on port 3000'));
```

Expected Output: - Successful transfer updates balances correctly. - Insufficient balance returns an error message. - Non-existent sender or receiver returns an error. - Logical correctness maintained without database transactions.

Learning Outcomes: - Learned to implement **dependent multi-document updates** without transactions. - Ensured **balance validation** before updates. - Understood how to handle **errors** and prevent inconsistent states. - Practiced **testing APIs** for success and failure scenarios.