Experiment-7.3

Account Transfer System with Balance Validation in Node.js

Code-

Bank Transfer Server (server.js)

This file contains the complete code for the Node.js banking application with MongoDB integration for secure account transfers with balance validation.

```
const express = require('express');
const mongoose = require('mongoose');
const app = express();

// Middleware to parse JSON
app.use(express.json());

// MongoDB Connection
const MONGO_URI = 'mongodb://localhost:27017/bankingDB';

mongoose.connect(MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
.then(() => console.log('Connected to MongoDB'))
.catch(err => console.error('MongoDB connection error:', err));

// --- 1. Account Schema and Model ---
/**
 * Defines the structure for bank account documents in MongoDB.
 * Each account has a unique account number, holder name, balance, and email.
 */
const accountSchema = new mongoose.Schema({
  accountNumber: {
    type: String,
    required: true,
    unique: true,
    trim: true
  },
  accountHolder: {
    type: String,
    required: true,
    trim: true
  },
  balance: {
```

```
    type: Number,
    required: true,
    min: 0,
    default: 0
  },
  email: {
    type: String,
    required: true,
    trim: true
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
});

const Account = mongoose.model('Account', accountSchema);

// --- 2. Transfer History Schema ---
/**
 * Stores all transfer attempts for auditing purposes.
 * Tracks both successful and failed transfers with reasons.
 */
const transferSchema = new mongoose.Schema({
  fromAccount: {
    type: String,
    required: true
  },
  toAccount: {
    type: String,
    required: true
  },
  amount: {
    type: Number,
    required: true
  },
  status: {
    type: String,
    enum: ['success', 'failed'],
    required: true
  },
  reason: {
    type: String
  },
  timestamp: {
    type: Date,
    default: Date.now
  }
});

const Transfer = mongoose.model('Transfer', transferSchema);
```

```
87
88 // --- 3. Routes ---
89
90 /**
91  * POST /api/accounts/create
92  * Creates a new bank account with validation.
93  */
94 app.post('/api/accounts/create', async (req, res) => {
95   try {
96     const { accountNumber, accountHolder, balance, email } =
     req.body;
97
98     // Validate input
99     if (!accountNumber || !accountHolder || !email) {
100       return res.status(400).json({
101         error: 'Account number, account holder name, and email
      are required.'
102       });
103     }
104
105     // Check if account already exists
106     const existingAccount = await Account.findOne({
     accountNumber });
107     if (existingAccount) {
108       return res.status(409).json({
109         error: 'Account number already exists.'
110       });
111     }
112
113     // Create new account
114     const newAccount = new Account({
115       accountNumber,
116       accountHolder,
117       balance: balance || 0,
118       email
119     });
120
121     await newAccount.save();
122
123     res.status(201).json({
124       message: 'Account created successfully',
125       account: {
126         accountNumber: newAccount.accountNumber,
127         accountHolder: newAccount.accountHolder,
128         balance: newAccount.balance,
129         email: newAccount.email
130       }
131     });
132
133   } catch (error) {
134     console.error('Error creating account:', error);
```

```
    res.status(500).json({
      error: 'Internal server error while creating account.'
    });
  }
});

/**
 * GET /api/accounts/:accountNumber
 * Retrieves account details by account number.
 */
app.get('/api/accounts/:accountNumber', async (req, res) => {
  try {
    const { accountNumber } = req.params;

    const account = await Account.findOne({ accountNumber });

    if (!account) {
      return res.status(404).json({
        error: 'Account not found.'
      });
    }

    res.json({
      accountNumber: account.accountNumber,
      accountHolder: account.accountHolder,
      balance: account.balance,
      email: account.email,
      createdAt: account.createdAt
    });

  } catch (error) {
    console.error('Error fetching account:', error);
    res.status(500).json({
      error: 'Internal server error while fetching account.'
    });
  }
});

/**
 * GET /api/accounts
 * Retrieves all accounts (for testing and verification).
 */
app.get('/api/accounts', async (req, res) => {
  try {
    const accounts = await Account.find().select('-__v');

    res.json({
      count: accounts.length,
      accounts: accounts
    });

```

```
186   } catch (error) {
187     console.error('Error fetching accounts:', error);
188     res.status(500).json({
189       error: 'Internal server error while fetching accounts.'
190     });
191   }
192 });
193
194 /**
195  * POST /api/transfer
196  * Transfers money from one account to another with validation.
197  * This implementation ensures logical correctness without database transactions
198  * by validating balances before performing sequential updates.
199  *
200  * Process:
201  * 1. Validate input (required fields, positive amount)
202  * 2. Fetch both sender and receiver accounts
203  * 3. Verify sender has sufficient balance
204  * 4. Deduct amount from sender
205  * 5. Add amount to receiver
206  * 6. Log the transfer
207  */
208 app.post('/api/transfer', async (req, res) => {
209   try {
210     const { fromAccount, toAccount, amount } = req.body;
211
212     // --- Step 1: Input Validation ---
213     if (!fromAccount || !toAccount || !amount) {
214       return res.status(400).json({
215         error: 'From account, to account, and amount are required.'
216       });
217     }
218
219     if (amount <= 0) {
220       return res.status(400).json({
221         error: 'Transfer amount must be greater than zero.'
222       });
223     }
224
225     if (fromAccount === toAccount) {
226       return res.status(400).json({
227         error: 'Cannot transfer to the same account.'
228       });
229     }
230
231     // --- Step 2: Fetch Both Accounts ---
232     const senderAccount = await Account.findOne({
```

```
233         accountNumber: fromAccount
234       });
235       const receiverAccount = await Account.findOne({
236         accountNumber: toAccount
237       });
238
239       // Check if sender account exists
240       if (!senderAccount) {
241         await logTransfer(fromAccount, toAccount, amount, '
      failed',
242           'Sender account not found');
243         return res.status(404).json({
244           error: 'Sender account not found.',
245           accountNumber: fromAccount
246         });
247       }
248
249       // Check if receiver account exists
250       if (!receiverAccount) {
251         await logTransfer(fromAccount, toAccount, amount, '
      failed',
252           'Receiver account not found');
253         return res.status(404).json({
254           error: 'Receiver account not found.',
255           accountNumber: toAccount
256         });
257       }
258
259       // --- Step 3: Balance Validation ---
260       // This is the critical check that prevents invalid
      transfers
261       if (senderAccount.balance < amount) {
262         await logTransfer(fromAccount, toAccount, amount, '
      failed',
263           'Insufficient balance');
264         return res.status(400).json({
265           error: 'Insufficient balance.',
266           currentBalance: senderAccount.balance,
267           requestedAmount: amount,
268           shortfall: amount - senderAccount.balance
269         });
270       }
271
272       // --- Step 4: Perform Sequential Updates ---
273       // First, deduct from sender
274       senderAccount.balance -= amount;
275       await senderAccount.save();
276
277       // Then, add to receiver
278       receiverAccount.balance += amount;
279       await receiverAccount.save();
```

```
280
281     // --- Step 5: Log successful transfer ---
282     await logTransfer(fromAccount, toAccount, amount, 'success
        ',
283       'Transfer completed successfully');
284
285     // --- Step 6: Return success response ---
286     res.json({
287       message: 'Transfer successful',
288       transfer: {
289         from: {
290           accountNumber: senderAccount.accountNumber,
291           accountHolder: senderAccount.accountHolder,
292           newBalance: senderAccount.balance
293         },
294         to: {
295           accountNumber: receiverAccount.accountNumber,
296           accountHolder: receiverAccount.accountHolder,
297           newBalance: receiverAccount.balance
298         },
299         amount: amount,
300         timestamp: new Date()
301       }
302     });
303
304   } catch (error) {
305     console.error('Error during transfer:', error);
306
307     // Log failed transfer due to system error
308     if (req.body.fromAccount && req.body.toAccount && req.body
        .amount) {
309       await logTransfer(
310         req.body.fromAccount,
311         req.body.toAccount,
312         req.body.amount,
313         'failed',
314         'System error: ' + error.message
315       );
316     }
317
318     res.status(500).json({
319       error: 'Internal server error during transfer.',
320       details: error.message
321     });
322   }
323 });
324
325 /**
326  * GET /api/transfers
327  * Retrieves transfer history for auditing purposes.
328  * Returns the 50 most recent transfers sorted by timestamp.
```

```
 */
app.get('/api/transfers', async (req, res) => {
  try {
    const transfers = await Transfer.find()
      .sort({ timestamp: -1 })
      .limit(50);

    res.json({
      count: transfers.length,
      transfers: transfers
    });

  } catch (error) {
    console.error('Error fetching transfers:', error);
    res.status(500).json({
      error: 'Internal server error while fetching transfers.'
    });
  }
});

/**
 * DELETE /api/accounts/:accountNumber
 * Deletes an account (useful for testing and cleanup).
 */
app.delete('/api/accounts/:accountNumber', async (req, res) =>
    {
  try {
    const { accountNumber } = req.params;

    const result = await Account.deleteOne({ accountNumber });

    if (result.deletedCount === 0) {
      return res.status(404).json({
        error: 'Account not found.'
      });
    }

    res.json({
      message: 'Account deleted successfully',
      accountNumber: accountNumber
    });

  } catch (error) {
    console.error('Error deleting account:', error);
    res.status(500).json({
      error: 'Internal server error while deleting account.'
    });
  }
});

// --- 4. Helper Functions ---
```

```
379
380 /**
381  * Helper function to log transfer attempts to the database.
382  * This provides an audit trail for both successful and failed
       transfers.
383  */
384 async function logTransfer(fromAccount, toAccount, amount,
       status,
385   reason = '') {
386   try {
387     const transfer = new Transfer({
388       fromAccount,
389       toAccount,
390       amount,
391       status,
392       reason
393     });
394     await transfer.save();
395   } catch (error) {
396     console.error('Error logging transfer:', error);
397   }
398 }
399
400 /**
401  * POST /api/seed-data
402  * Seeds the database with initial test accounts.
403  * Useful for demonstration and testing purposes.
404  */
405 app.post('/api/seed-data', async (req, res) => {
406   try {
407     // Clear existing data
408     await Account.deleteMany({});
409     await Transfer.deleteMany({});
410
411     // Create sample accounts with different balances
412     const accounts = [
413       {
414         accountNumber: 'ACC001',
415         accountHolder: 'John Doe',
416         balance: 10000,
417         email: 'john@example.com'
418       },
419       {
420         accountNumber: 'ACC002',
421         accountHolder: 'Jane Smith',
422         balance: 5000,
423         email: 'jane@example.com'
424       },
425       {
426         accountNumber: 'ACC003',
427         accountHolder: 'Bob Johnson',
```

```
428         balance: 2000,
429         email: 'bob@example.com'
430       }
431     ];
432 
433     await Account.insertMany(accounts);
434 
435     res.json({
436       message: 'Sample data seeded successfully',
437       accounts: accounts.map(acc => ({
438         accountNumber: acc.accountNumber,
439         accountHolder: acc.accountHolder,
440         balance: acc.balance
441       }))
442     });
443 
444   } catch (error) {
445     console.error('Error seeding data:', error);
446     res.status(500).json({
447       error: 'Error seeding data.'
448     });
449   }
450 });
451 
452 // --- 5. Start Server ---
453 const PORT = 3000;
454 app.listen(PORT, () => {
455   console.log(`\n${'='.repeat(60)}`);
456   console.log(`  Bank Transfer System Server`);
457   console.log(`${'='.repeat(60)}\n`);
458   console.log(`Server running on http://localhost:${PORT}\n`);
459   console.log('Available Endpoints:');
460   console.log('-'.repeat(60));
461   console.log('POST   /api/seed-data             - Seed test
      accounts');
462   console.log('POST   /api/accounts/create       - Create new
       account');
463   console.log('GET    /api/accounts              - Get all
      accounts');
464   console.log('GET    /api/accounts/:accountNumber - Get
      account details');
465   console.log('POST   /api/transfer              - Transfer
      money');
466   console.log('GET    /api/transfers             - View
      transfer history');
467   console.log('DELETE /api/accounts/:accountNumber - Delete
      account');
468   console.log('-'.repeat(60));
469   console.log('\nQuick Start:');
470   console.log('1. POST http://localhost:3000/api/seed-data');
471   console.log('2. POST http://localhost:3000/api/transfer');
```

```
472   console.log('   Body: {');
473   console.log('     "fromAccount": "ACC001",');
474   console.log('     "toAccount": "ACC002",');
475   console.log('     "amount": 500');
476   console.log('   }\n');
477   console.log(`${'='.repeat(60)}\n`);
478 });
```

Listing 1: server.js - Account Transfer System Server