Exercise 2: MongoDB CRUD Operations

Invoice Management System - Database Integration

Objective: Implement MongoDB integration for invoice management system with complete CRUD operations.

Step 1: Setup the Node.js Project

```
mkdir invoice-management-api
cd invoice-management-api
npm init -y
```

This creates a `package.json` file to manage dependencies.

Install required dependencies:

```
npm install express
npm install mongoose
npm install nodemon --save-dev
```

`express`: Framework for building the REST API.
`mongoose`: For connecting to mongodb.
`nodemon`: For automatically restarting the server during development.

Setup the project structure:

```
invoice-management-api/
├── index.js
├── db.js
├── routes/
│   └── invoiceRoutes.js
├── middlewares/
│   └── validationMiddleware.js
└── models/
    └── invoice.js
```

Step 2: Define the Invoice Structure

Create a basic invoice model in `models/invoice.js`

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


const itemSchema = new mongoose.Schema({
  itemName: { type: String, required: true },
  quantity: { type: Number, required: true, min: 1 },
```

```
  price: { type: Number, required: true, min: 0 },
  amount: { type: Number, required: true }
});

const invoiceSchema = new mongoose.Schema({
  invoiceNumber: { type: Number, unique: true, required: true },
  customerName: { type: String, required: true },
  billingAddress: { type: String, required: true },
  date: { type: Date, default: Date.now },
  items: [itemSchema],
  totalAmount: { type: Number, required: true }
});

const Invoice = mongoose.model('Invoice', invoiceSchema);

module.exports = Invoice;
```

Step 3: Implement API Endpoints

Define the routes in routes/invoiceRoutes.js:

```
const express = require('express');
const Invoice = require('../models/invoice');
const router = express.Router();

router.get('/', async (req, res) => {
    try {
      const invoices = await Invoice.find();
      res.status(200).json({ success: true, data: invoices });
    } catch (err) {
      res.status(500).json({ success: false, message: err.message
});
    }
  });
```

```
router.post('/', async (req, res) => {
    try {
      const { customerName, billingAddress, items } = req.body;

      // Calculate items' amounts and total amount
      const calculatedItems = items.map(item => ({
        ...item,
        amount: item.quantity * item.price,
      }));
      const totalAmount = calculatedItems.reduce((sum, item) => sum
+ item.amount, 0);

      // Determine next invoice number
      const lastInvoice = await Invoice.findOne().sort({
invoiceNumber: -1 });
      const invoiceNumber = lastInvoice ? lastInvoice.invoiceNumber
+ 1 : 1;

      const invoice = new Invoice({
        invoiceNumber,
        customerName,
        billingAddress,
        items: calculatedItems,
        totalAmount,
      });

      const savedInvoice = await invoice.save();
      res.status(201).json({ success: true, data: savedInvoice,
message: 'Invoice created' });
    } catch (err) {
      res.status(500).json({ success: false, message: err.message
});
    }
```

```
  });


router.get('/:id', async (req, res) => {
    try {
      const invoice = await Invoice.findOne({ invoiceNumber:
req.params.id });
      if (!invoice) {
        return res.status(404).json({ success: false, message:
'Invoice not found' });
      }
      res.status(200).json({ success: true, data: invoice });
    } catch (err) {
      res.status(500).json({ success: false, message: err.message
});
    }
  });


  router.put('/:id', async (req, res) => {
    try {
      const { customerName, billingAddress, items } = req.body;


      // Recalculate totals
      const calculatedItems = items.map(item => ({
        ...item,
        amount: item.quantity * item.price,
      }));
      const totalAmount = calculatedItems.reduce((sum, item) => sum
+ item.amount, 0);


      const updatedInvoice = await Invoice.findOneAndUpdate(
        { invoiceNumber: req.params.id },
        { customerName, billingAddress, items: calculatedItems,
totalAmount },
        { new: true, runValidators: true }
```

```
      );


      if (!updatedInvoice) {
        return res.status(404).json({ success: false, message:
'Invoice not found' });
      }


      res.status(200).json({ success: true, data: updatedInvoice,
message: 'Invoice updated' });
    } catch (err) {
      res.status(500).json({ success: false, message: err.message
});
    }
  });


  router.delete('/:id', async (req, res) => {
    try {
      const deletedInvoice = await Invoice.findOneAndDelete({
invoiceNumber: req.params.id });
      if (!deletedInvoice) {
        return res.status(404).json({ success: false, message:
'Invoice not found' });
      }
      res.status(200).json({ success: true, data: deletedInvoice,
message: 'Invoice deleted' });
    } catch (err) {
      res.status(500).json({ success: false, message: err.message
});
    }
  });


  module.exports = router;
```

Step 4: Setup Express Server

Define the main server in `index.js`:

```
const express = require('express');
const invoiceRoutes = require('./routes/invoiceRoutes');
const connectDB = require('./db');

const app = express();
const PORT = 5000;

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

connectDB();

// Routes
app.use('/api/invoices', invoiceRoutes);

// Error handling middleware
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ error: 'Something went wrong' });
});

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

Step 5: Testing

Use **Postman** or **ThunderClient** to test:

- **GET /api/invoices:** Fetch all invoices.
- **POST /api/invoices:** Create a new invoice.
- **GET /api/invoices/id:** Fetch particular invoices.
- **PUT /api/invoices/id:** Update particular invoice.
- **DELETE /api/invoices/id:** Delete particular invoice.