Lecture #12: Introduction to Node.js, Express and MongoDB

1. Install Express & mongoDB
   npm install express
   npm install mongodb

2. Create Node Server
   Server.js

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

const app = express();
const PORT = 3000;

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

// Basic route
app.get('/', (req, res) => {
    res.send('Server is running!');
});

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

3. Run server with following command:
   Node server.js

   Connecting with MongoDB:

   Mongo.js

```
const express = require('express');
const { MongoClient, ObjectId } = require('mongodb');
const bodyParser = require('body-parser');

// Create Express app
const app = express();
const port = 3000;

// Middleware
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

// MongoDB Connection String
const mongoURI = 'mongodb://localhost:27017';
const dbName = 'expressMongoApp';
```

```
// MongoDB Client
const client = new MongoClient(mongoURI);

// Connect to MongoDB
let db;
async function connectToMongo() {
  try {
    await client.connect();
    console.log('Connected to MongoDB');
    db = client.db(dbName);

    // Optional: Create indexes for better performance
    await db.collection('tasks').createIndex({ "title": 1 });

    return true;
  } catch (err) {
    console.error('Failed to connect to MongoDB', err);
    process.exit(1);
  }
}

// Basic route
app.get('/', (req, res) => {
  res.send('Express + MongoDB API is running');
});


// Start the server after connecting to MongoDB
async function startServer() {
  const connected = await connectToMongo();
  if (connected) {
    app.listen(port, () => {
      console.log(`Server running at http://localhost:${port}`);
    });
  }
}

startServer();

// Graceful shutdown
process.on('SIGINT', async () => {
  try {
    await client.close();
    console.log('MongoDB connection closed');
    process.exit(0);
  } catch (err) {
    console.error(err);
    process.exit(1);
  }
});
```

Get All Tasks API:

```
// GET all tasks
app.get('/api/tasks', async (req, res) => {
  try {
    const tasks = await db.collection('tasks').find({}).toArray();
    res.json(tasks);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

Get Single Task

```
// GET a single task
app.get('/api/tasks/:id', async (req, res) => {
  try {
    const taskId = req.params.id;

    // Validate ObjectId
    if (!ObjectId.isValid(taskId)) {
      return res.status(400).json({ message: 'Invalid task ID format' });
    }

    const task = await db.collection('tasks').findOne({ _id: new ObjectId(taskId) });

    if (!task) {
      return res.status(404).json({ message: 'Task not found' });
    }

    res.json(task);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

Insert New Task

```
// POST a new task
app.post('/api/tasks', async (req, res) => {
  try {
    const { title, description, status } = req.body;

    // Validate required fields
    if (!title) {
      return res.status(400).json({ message: 'Title is required' });
    }

    const newTask = {
      title,
      description: description || '',
```

```
    status: status || 'pending',
    createdAt: new Date()
   };

   const result = await db.collection('tasks').insertOne(newTask);

   res.status(201).json({
    message: 'Task created successfully',
    taskId: result.insertedId,
    task: { _id: result.insertedId, ...newTask }
   });
  } catch (err) {
   res.status(500).json({ error: err.message });
  }
});
```

Update a Task

```
// PUT (update) a task
app.put('/api/tasks/:id', async (req, res) => {
  try {
   const taskId = req.params.id;
   const { title, description, status } = req.body;

   // Validate ObjectId
   if (!ObjectId.isValid(taskId)) {
    return res.status(400).json({ message: 'Invalid task ID format' });
   }

   // Create update object with only provided fields
   const updateData = {};
   if (title !== undefined) updateData.title = title;
   if (description !== undefined) updateData.description = description;
   if (status !== undefined) updateData.status = status;
   updateData.updatedAt = new Date();

   const result = await db.collection('tasks').updateOne(
    { _id: new ObjectId(taskId) },
    { $set: updateData }
   );

   if (result.matchedCount === 0) {
    return res.status(404).json({ message: 'Task not found' });
   }
   // Get the updated task
   const updatedTask = await db.collection('tasks').findOne({ _id: new ObjectId(taskId) });

   res.json({
    message: 'Task updated successfully',
    task: updatedTask
   });
  } catch (err) {
```

```
    res.status(500).json({ error: err.message });
  }
});
```

Delete a Task:

```
// DELETE a task
app.delete('/api/tasks/:id', async (req, res) => {
  try {
    const taskId = req.params.id;

    // Validate ObjectId
    if (!ObjectId.isValid(taskId)) {
      return res.status(400).json({ message: 'Invalid task ID format' });
    }

    const result = await db.collection('tasks').deleteOne({ _id: new ObjectId(taskId) });

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

    res.json({ message: 'Task deleted successfully' });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```