Complete Mongoose Learning Roadmap: Beginner to Expert

Prerequisites

- Basic JavaScript knowledge

- Understanding of Node.js

- Basic MongoDB concepts

- Familiarity with async/await and Promises

Phase 1: Foundation (Beginner Level)

1. Setup and Installation

```bash
bash

# Initialize a new Node.js project
npm init -y

# Install Mongoose
npm install mongoose

# Install additional dependencies for examples
npm install express dotenv
```

2. Basic Connection

```javascript
javascript
```

```javascript
// connection.js
const mongoose = require('mongoose');

// Basic connection
async function connectDB() {
  try {
    await mongoose.connect('mongodb://localhost:27017/myapp', {
      useNewUrlParser: true,
      useUnifiedTopology: true
    });
    console.log('MongoDB Connected Successfully');
  } catch (error) {
    console.error('Connection failed:', error.message);
    process.exit(1);
  }
}

module.exports = connectDB;
```

3. Your First Schema and Model

```javascript
```

```javascript
// models/User.js
const mongoose = require('mongoose');

// Define Schema
const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
        trim: true
    },
    email: {
        type: String,
        required: true,
        unique: true,
        lowercase: true
    },
    age: {
        type: Number,
        min: 0,
        max: 120
    },
    createdAt: {
        type: Date,
        default: Date.now
    }
});

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

module.exports = User;
```

Phase 2: CRUD Operations (Intermediate Beginner)

1. Creating Documents

```javascript
javascript
```

```javascript
// create-operations.js
const User = require('./models/User');

// Method 1: Using constructor and save()
async function createUserMethod1() {
    try {
        const user = new User({
            name: 'John Doe',
            email: 'john@example.com',
            age: 25
        });

        const savedUser = await user.save();
        console.log('User created:', savedUser);
        return savedUser;
    } catch (error) {
        console.error('Error creating user:', error.message);
    }
}

// Method 2: Using create()
async function createUserMethod2() {
    try {
        const user = await User.create({
            name: 'Jane Smith',
            email: 'jane@example.com',
            age: 30
        });
        console.log('User created:', user);
        return user;
    } catch (error) {
        console.error('Error creating user:', error.message);
    }
}

// Method 3: Creating multiple documents
async function createMultipleUsers() {
    try {
        const users = await User.insertMany([
            { name: 'Alice', email: 'alice@example.com', age: 28 },
            { name: 'Bob', email: 'bob@example.com', age: 32 },
            { name: 'Charlie', email: 'charlie@example.com', age: 24 }
        ]);
```

```javascript
        console.log('Multiple users created:', users);
        return users;
    } catch (error) {
        console.error('Error creating multiple users:', error.message);
    }
}
```

2. Reading Documents

```javascript
javascript
```

```javascript
// read-operations.js
const User = require('./models/User');

// Find all documents
async function findAllUsers() {
    try {
        const users = await User.find();
        console.log('All users:', users);
        return users;
    } catch (error) {
        console.error('Error finding users:', error.message);
    }
}

// Find by ID
async function findUserById(userId) {
    try {
        const user = await User.findById(userId);
        if (!user) {
            console.log('User not found');
            return null;
        }
        console.log('User found:', user);
        return user;
    } catch (error) {
        console.error('Error finding user by ID:', error.message);
    }
}

// Find with conditions
async function findUsersWithConditions() {
    try {
        // Find users older than 25
        const olderUsers = await User.find({ age: { $gt: 25 } });

        // Find user by email
        const userByEmail = await User.findOne({ email: 'john@example.com' });

        // Find with multiple conditions
        const specificUsers = await User.find({
            age: { $gte: 20, $lte: 35 },
            name: { $regex: /^J/, $options: 'i' } // Names starting with 'J'
        });
```

```javascript
        console.log('Older users:', olderUsers);
        console.log('User by email:', userByEmail);
        console.log('Specific users:', specificUsers);

        return { olderUsers, userByEmail, specificUsers };
    } catch (error) {
        console.error('Error in conditional queries:', error.message);
    }
}

// Advanced querying
async function advancedQueries() {
    try {
        // Limit and skip (pagination)
        const paginatedUsers = await User.find()
            .limit(5)
            .skip(0)
            .sort({ createdAt: -1 }); // Sort by creation date, newest first

        // Select specific fields
        const usersWithSelectedFields = await User.find()
            .select('name email -_id'); // Include name and email, exclude _id

        // Count documents
        const userCount = await User.countDocuments({ age: { $gte: 18 } });

        console.log('Paginated users:', paginatedUsers);
        console.log('Selected fields:', usersWithSelectedFields);
        console.log('Adult user count:', userCount);

        return { paginatedUsers, usersWithSelectedFields, userCount };
    } catch (error) {
        console.error('Error in advanced queries:', error.message);
    }
}
```

3. Updating Documents

```
javascript
```

```javascript
// update-operations.js
const User = require('./models/User');

// Update one document
async function updateUser(userId, updateData) {
    try {
        // Method 1: findByIdAndUpdate
        const updatedUser = await User.findByIdAndUpdate(
            userId,
            updateData,
            {
                new: true, // Return updated document
                runValidators: true // Run schema validators
            }
        );

        if (!updatedUser) {
            console.log('User not found');
            return null;
        }

        console.log('Updated user:', updatedUser);
        return updatedUser;
    } catch (error) {
        console.error('Error updating user:', error.message);
    }
}

// Update with conditions
async function updateUsersWithConditions() {
    try {
        // Update one document matching condition
        const result1 = await User.updateOne(
            { email: 'john@example.com' },
            { $set: { age: 26 } }
        );

        // Update multiple documents
        const result2 = await User.updateMany(
            { age: { $lt: 25 } },
            { $inc: { age: 1 } } // Increment age by 1
        );
```

```javascript
        console.log('Single update result:', result1);
        console.log('Multiple update result:', result2);

        return { result1, result2 };
    } catch (error) {
        console.error('Error in conditional updates:', error.message);
    }
}

// Find and update with custom logic
async function findAndUpdate(email) {
    try {
        const user = await User.findOne({ email });

        if (user) {
            user.age += 1;
            user.name = user.name.toUpperCase();
            const savedUser = await user.save();
            console.log('Updated user with custom logic:', savedUser);
            return savedUser;
        } else {
            console.log('User not found');
            return null;
        }
    } catch (error) {
        console.error('Error in find and update:', error.message);
    }
}
```

4. Deleting Documents

```
javascript
```

```javascript
// delete-operations.js
const User = require('./models/User');

// Delete by ID
async function deleteUserById(userId) {
    try {
        const deletedUser = await User.findByIdAndDelete(userId);

        if (!deletedUser) {
            console.log('User not found');
            return null;
        }

        console.log('Deleted user:', deletedUser);
        return deletedUser;
    } catch (error) {
        console.error('Error deleting user:', error.message);
    }
}

// Delete with conditions
async function deleteUsersWithConditions() {
    try {
        // Delete one document
        const result1 = await User.deleteOne({ email: 'test@example.com' });

        // Delete multiple documents
        const result2 = await User.deleteMany({ age: { $lt: 18 } });

        console.log('Single delete result:', result1);
        console.log('Multiple delete result:', result2);

        return { result1, result2 };
    } catch (error) {
        console.error('Error in conditional deletes:', error.message);
    }
}

// Find and delete
async function findAndDelete(email) {
    try {
        const user = await User.findOneAndDelete({ email });
```

```javascript
      if (user) {
        console.log('Found and deleted user:', user);
        return user;
      } else {
        console.log('User not found');
        return null;
      }
    } catch (error) {
      console.error('Error in find and delete:', error.message);
    }
  }
}
```

Phase 3: Intermediate Level

1. Advanced Schema Features

```javascript
```

```javascript
// models/AdvancedUser.js
const mongoose = require('mongoose');

const advancedUserSchema = new mongoose.Schema({
  // String validations
  username: {
    type: String,
    required: [true, 'Username is required'],
    unique: true,
    minlength: [3, 'Username must be at least 3 characters'],
    maxlength: [20, 'Username cannot exceed 20 characters'],
    match: [/^[a-zA-Z0-9_]+$/, 'Username can only contain letters, numbers, and underscores']
  },

  // Email with custom validator
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true,
    validate: {
      validator: function(email) {
        return /^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/.test(email);
      },
      message: 'Please enter a valid email address'
    }
  },

  // Number validations
  age: {
    type: Number,
    min: [0, 'Age cannot be negative'],
    max: [120, 'Age cannot exceed 120'],
    validate: {
      validator: Number.isInteger,
      message: 'Age must be an integer'
    }
  },

  // Enum field
  role: {
    type: String,
    enum: {
```

```
      values: ['user', 'admin', 'moderator'],
      message: 'Role must be either user, admin, or moderator'
    },
    default: 'user'
  },

  // Array of strings
  interests: [{
    type: String,
    trim: true
  }],

  // Nested object
  profile: {
    firstName: {
      type: String,
      required: true,
      trim: true
    },
    lastName: {
      type: String,
      required: true,
      trim: true
    },
    bio: String,
    avatar: String
  },

  // Reference to another model
  posts: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Post'
  }],

  // Mixed type
  metadata: mongoose.Schema.Types.Mixed,

  // Timestamps
  isActive: {
    type: Boolean,
    default: true
  }
}, {
  timestamps: true, // Adds createdAt and updatedAt
```

```javascript
    versionKey: false // Removes __v field
});

// Virtual field (not stored in database)
advancedUserSchema.virtual('fullName').get(function() {
    return `${this.profile.firstName} ${this.profile.lastName}`;
});

// Virtual populate
advancedUserSchema.virtual('postCount', {
    ref: 'Post',
    localField: '_id',
    foreignField: 'author',
    count: true
});

// Pre-save middleware
advancedUserSchema.pre('save', function(next) {
    if (this.isModified('email')) {
        this.email = this.email.toLowerCase();
    }
    next();
});

// Post-save middleware
advancedUserSchema.post('save', function(doc) {
    console.log(`User ${doc.username} has been saved`);
});

// Instance method
advancedUserSchema.methods.getPublicProfile = function() {
    return {
        username: this.username,
        fullName: this.fullName,
        interests: this.interests,
        bio: this.profile.bio
    };
};

// Static method
advancedUserSchema.statics.findByRole = function(role) {
    return this.find({ role });
};
```

```javascript
const AdvancedUser = mongoose.model('AdvancedUser', advancedUserSchema);
module.exports = AdvancedUser;
```

2. Relationships and Population

```
javascript
```

```javascript
// models/Post.js
const mongoose = require('mongoose');

const postSchema = new mongoose.Schema({
    title: {
        type: String,
        required: true,
        trim: true
    },
    content: {
        type: String,
        required: true
    },
    author: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'AdvancedUser',
        required: true
    },
    tags: [{
        type: String,
        trim: true
    }],
    likes: [{
        user: {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'AdvancedUser'
        },
        createdAt: {
            type: Date,
            default: Date.now
        }
    }],
    comments: [{
        user: {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'AdvancedUser',
            required: true
        },
        text: {
            type: String,
            required: true
        },
        createdAt: {
```

```javascript
        type: Date,
        default: Date.now
      }
    }]
}, {
    timestamps: true
});


const Post = mongoose.model('Post', postSchema);
module.exports = Post;

// Using relationships
const AdvancedUser = require('./AdvancedUser');

async function createPostWithUser() {
    try {
        // Create a user first
        const user = await AdvancedUser.create({
            username: 'johndoe',
            email: 'john@example.com',
            age: 25,
            profile: {
                firstName: 'John',
                lastName: 'Doe',
                bio: 'Software developer'
            },
            interests: ['programming', 'reading']
        });

        // Create a post
        const post = await Post.create({
            title: 'My First Post',
            content: 'This is my first blog post!',
            author: user._id,
            tags: ['intro', 'blog']
        });

        console.log('Created post:', post);
        return { user, post };
    } catch (error) {
        console.error('Error:', error.message);
    }
}
```

```javascript
// Population examples
async function populationExamples() {
  try {
    // Basic population
    const posts = await Post.find().populate('author');

    // Population with field selection
    const postsWithSelectedFields = await Post.find()
      .populate('author', 'username profile.firstName profile.lastName');

    // Nested population
    const postsWithComments = await Post.find()
      .populate({
        path: 'comments.user',
        select: 'username profile.firstName'
      })
      .populate('author', 'username');

    // Multiple populations
    const fullPosts = await Post.find()
      .populate('author', 'username profile')
      .populate('likes.user', 'username')
      .populate('comments.user', 'username');

    console.log('Posts with basic population:', posts);
    console.log('Posts with selected fields:', postsWithSelectedFields);
    console.log('Posts with nested population:', postsWithComments);

    return { posts, postsWithSelectedFields, postsWithComments, fullPosts };
  } catch (error) {
    console.error('Error in population:', error.message);
  }
}
```

Phase 4: Advanced Level

1. Aggregation Pipeline

```
javascript
```

```javascript
// aggregation-examples.js
const Post = require('./models/Post');
const AdvancedUser = require('./models/AdvancedUser');

async function aggregationExamples() {
  try {
    // Basic aggregation - count posts by author
    const postCountsByAuthor = await Post.aggregate([
      {
        $group: {
          _id: '$author',
          postCount: { $sum: 1 },
          totalLikes: { $sum: { $size: '$likes' } }
        }
      },
      {
        $lookup: {
          from: 'advancedusers',
          localField: '_id',
          foreignField: '_id',
          as: 'author'
        }
      },
      {
        $unwind: '$author'
      },
      {
        $project: {
          _id: 0,
          username: '$author.username',
          fullName: {
            $concat: ['$author.profile.firstName', ' ', '$author.profile.lastName']
          },
          postCount: 1,
          totalLikes: 1,
          avgLikesPerPost: { $divide: ['$totalLikes', '$postCount'] }
        }
      },
      {
        $sort: { postCount: -1 }
      }
    ]);
```

```javascript
// Advanced aggregation - posts with engagement metrics
const postsWithMetrics = await Post.aggregate([
  {
    $addFields: {
      likeCount: { $size: '$likes' },
      commentCount: { $size: '$comments' },
      engagementScore: {
        $add: [
          { $size: '$likes' },
          { $multiply: [{ $size: '$comments' }, 2] }
        ]
      }
    }
  },
  {
    $match: {
      engagementScore: { $gte: 5 }
    }
  },
  {
    $lookup: {
      from: 'advancedusers',
      localField: 'author',
      foreignField: '_id',
      as: 'authorInfo'
    }
  },
  {
    $unwind: '$authorInfo'
  },
  {
    $project: {
      title: 1,
      content: { $substr: ['$content', 0, 100] },
      author: '$authorInfo.username',
      likeCount: 1,
      commentCount: 1,
      engagementScore: 1,
      createdAt: 1
    }
  },
  {
    $sort: { engagementScore: -1 }
  },
```

```javascript
      {
        $limit: 10
      }
    ]);

    console.log('Post counts by author:', postCountsByAuthor);
    console.log('High engagement posts:', postsWithMetrics);

    return { postCountsByAuthor, postsWithMetrics };
  } catch (error) {
    console.error('Error in aggregation:', error.message);
  }
}

// Time-based aggregation
async function timeBasedAggregation() {
  try {
    const monthlyStats = await Post.aggregate([
      {
        $group: {
          _id: {
            year: { $year: '$createdAt' },
            month: { $month: '$createdAt' }
          },
          postCount: { $sum: 1 },
          totalLikes: { $sum: { $size: '$likes' } },
          avgLikes: { $avg: { $size: '$likes' } },
          posts: { $push: { title: '$title', likes: { $size: '$likes' } } }
        }
      },
      {
        $sort: { '_id.year': -1, '_id.month': -1 }
      },
      {
        $project: {
          _id: 0,
          period: {
            $dateFromParts: {
              year: '$_id.year',
              month: '$_id.month'
            }
          },
          postCount: 1,
          totalLikes: 1,
```

```javascript
                avgLikes: { $round: ['$avgLikes', 2] },
                topPost: {
                    $arrayElemAt: [
                        {
                            $sortArray: {
                                input: '$posts',
                                sortBy: { likes: -1 }
                            }
                        },
                        0
                    ]
                }
            }
        }
    ]);

    console.log('Monthly statistics:', monthlyStats);
    return monthlyStats;
} catch (error) {
    console.error('Error in time-based aggregation:', error.message);
}
}
```

2. Transactions

```javascript
javascript
```

```javascript
// transactions.js
const mongoose = require('mongoose');
const AdvancedUser = require('./models/AdvancedUser');
const Post = require('./models/Post');

async function createUserAndPostTransaction() {
    const session = await mongoose.startSession();

    try {
        await session.withTransaction(async () => {
            // Create user
            const user = new AdvancedUser({
                username: 'transactionuser',
                email: 'transaction@example.com',
                age: 28,
                profile: {
                    firstName: 'Transaction',
                    lastName: 'User'
                }
            });

            await user.save({ session });

            // Create post
            const post = new Post({
                title: 'Transaction Post',
                content: 'This post was created in a transaction',
                author: user._id,
                tags: ['transaction', 'example']
            });

            await post.save({ session });

            // Update user with post reference
            user.posts.push(post._id);
            await user.save({ session });

            console.log('Transaction completed successfully');
            return { user, post };
        });
    } catch (error) {
        console.error('Transaction failed:', error.message);
        throw error;
```

```javascript
    } finally {
        await session.endSession();
    }
}


// Complex transaction with error handling
async function transferLikesTransaction(fromPostId, toPostId, userId) {
    const session = await mongoose.startSession();

    try {
        const result = await session.withTransaction(async () => {
            // Find posts
            const fromPost = await Post.findById(fromPostId).session(session);
            const toPost = await Post.findById(toPostId).session(session);

            if (!fromPost || !toPost) {
                throw new Error('One or both posts not found');
            }

            // Check if user liked the from post
            const likeIndex = fromPost.likes.findIndex(
                like => like.user.toString() === userId.toString()
            );

            if (likeIndex === -1) {
                throw new Error('User has not liked the source post');
            }

            // Remove like from source post
            fromPost.likes.splice(likeIndex, 1);
            await fromPost.save({ session });

            // Add like to destination post
            const existingLike = toPost.likes.find(
                like => like.user.toString() === userId.toString()
            );

            if (!existingLike) {
                toPost.likes.push({ user: userId });
                await toPost.save({ session });
            }

            return { fromPost, toPost };
        });
```

```javascript
      console.log('Like transfer completed');
      return result;
    } catch (error) {
      console.error('Like transfer failed:', error.message);
      throw error;
    } finally {
      await session.endSession();
    }
}
```

3. Performance Optimization

```javascript
```

```javascript
// optimization.js
const AdvancedUser = require('./models/AdvancedUser');
const Post = require('./models/Post');

// Indexing examples
async function createIndexes() {
  try {
    // Single field index
    await AdvancedUser.collection.createIndex({ email: 1 });

    // Compound index
    await Post.collection.createIndex({ author: 1, createdAt: -1 });

    // Text index for search
    await Post.collection.createIndex({
      title: 'text',
      content: 'text',
      tags: 'text'
    });

    // Sparse index (only for documents that have the field)
    await AdvancedUser.collection.createIndex(
      { 'profile.bio': 1 },
      { sparse: true }
    );

    console.log('Indexes created successfully');
  } catch (error) {
    console.error('Error creating indexes:', error.message);
  }
}

// Efficient queries
async function efficientQueries() {
  try {
    // Use projection to limit fields
    const users = await AdvancedUser.find({ role: 'user' })
      .select('username email profile.firstName profile.lastName')
      .lean(); // Returns plain JavaScript objects (faster)

    // Use indexes effectively
    const recentPosts = await Post.find({
      createdAt: { $gte: new Date(Date.now() - 7 * 24 * 60 * 60 * 1000) }
```

```javascript
    })
      .sort({ createdAt: -1 })
      .limit(20)
      .populate('author', 'username')
      .lean();

    // Text search
    const searchResults = await Post.find(
      { $text: { $search: 'javascript programming' } },
      { score: { $meta: 'textScore' } }
    )
      .sort({ score: { $meta: 'textScore' } })
      .limit(10);

    console.log('Efficient queries completed');
    return { users, recentPosts, searchResults };
  } catch (error) {
    console.error('Error in efficient queries:', error.message);
  }
}

// Batch operations
async function batchOperations() {
  try {
    // Bulk write operations
    const bulkOps = [
      {
        updateOne: {
          filter: { username: 'user1' },
          update: { $inc: { age: 1 } }
        }
      },
      {
        updateMany: {
          filter: { role: 'user' },
          update: { $set: { isActive: true } }
        }
      },
      {
        deleteOne: {
          filter: { email: 'tobedeleted@example.com' }
        }
      }
    ];
```

```javascript
    const result = await AdvancedUser.bulkWrite(bulkOps);
    console.log('Bulk operations result:', result);

    return result;
  } catch (error) {
    console.error('Error in batch operations:', error.message);
  }
}
```

Phase 5: Expert Level

1. Custom Plugins

```javascript
```

```javascript
// plugins/timestampPlugin.js
function timestampPlugin(schema, options) {
    const { paths = ['createdAt', 'updatedAt'], index = false } = options || {};

    // Add timestamp fields
    if (paths.includes('createdAt')) {
        schema.add({
            createdAt: {
                type: Date,
                default: Date.now,
                immutable: true
            }
        });
    }

    if (paths.includes('updatedAt')) {
        schema.add({
            updatedAt: {
                type: Date,
                default: Date.now
            }
        });
    }

    // Pre-save middleware to update 'updatedAt'
    schema.pre('save', function(next) {
        if (paths.includes('updatedAt') && this.isModified() && !this.isNew) {
            this.updatedAt = new Date();
        }
        next();
    });

    // Pre-update middleware
    schema.pre(['updateOne', 'updateMany', 'findOneAndUpdate'], function() {
        if (paths.includes('updatedAt')) {
            this.set({ updatedAt: new Date() });
        }
    });

    // Create indexes if requested
    if (index) {
        if (paths.includes('createdAt')) {
            schema.index({ createdAt: 1 });
```

```javascript
    }
    if (paths.includes('updatedAt')) {
      schema.index({ updatedAt: 1 });
    }
  }
}

module.exports = timestampPlugin;
```

2. Advanced Middleware

```javascript
```

```javascript
// middleware/auditPlugin.js
function auditPlugin(schema, options) {
    const auditSchema = {
        auditLog: [{
            action: {
                type: String,
                enum: ['create', 'update', 'delete'],
                required: true
            },
            changes: mongoose.Schema.Types.Mixed,
            user: {
                type: mongoose.Schema.Types.ObjectId,
                ref: 'AdvancedUser'
            },
            timestamp: {
                type: Date,
                default: Date.now
            },
            ip: String,
            userAgent: String
        }]
    };

    schema.add(auditSchema);

    // Pre-save middleware for auditing
    schema.pre('save', function(next) {
        const doc = this;
        const isNew = doc.isNew;
        const modifiedPaths = doc.modifiedPaths();

        if (isNew) {
            doc.auditLog.push({
                action: 'create',
                changes: doc.toObject(),
                user: doc.$locals.currentUser,
                ip: doc.$locals.ip,
                userAgent: doc.$locals.userAgent
            });
        } else if (modifiedPaths.length > 0) {
            const changes = {};
            modifiedPaths.forEach(path => {
                if (path !== 'auditLog') {
```

```javascript
            changes[path] = {
                old: doc.$locals.original?.[path],
                new: doc[path]
            };
        }
    });

    doc.auditLog.push({
        action: 'update',
        changes,
        user: doc.$locals.currentUser,
        ip: doc.$locals.ip,
        userAgent: doc.$locals.userAgent
    });
    }

    next();
    });
}

module.exports = auditPlugin;
```

3. Complete Application Example

```javascript
```

```javascript
// app.js - Complete Express + Mongoose Application
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
require('dotenv').config();

// Import models
const AdvancedUser = require('./models/AdvancedUser');
const Post = require('./models/Post');

// Import plugins
const timestampPlugin = require('./plugins/timestampPlugin');
const auditPlugin = require('./middleware/auditPlugin');

// Apply plugins globally
mongoose.plugin(timestampPlugin, { index: true });
mongoose.plugin(auditPlugin);

const app = express();

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

// Database connection with advanced options
async function connectDB() {
  try {
    await mongoose.connect(process.env.MONGODB_URI || 'mongodb://localhost:27017/advanced_blog', {
      useNewUrlParser: true,
      useUnifiedTopology: true,
      maxPoolSize: 10,
      serverSelectionTimeoutMS: 5000,
      socketTimeoutMS: 45000,
      bufferCommands: false,
      bufferMaxEntries: 0
    });
    console.log('✅ MongoDB Connected');
  } catch (error) {
    console.error('❌ MongoDB connection failed:', error.message);
    process.exit(1);
  }
}
```

```javascript
// Error handling middleware
app.use((error, req, res, next) => {
  if (error.name === 'ValidationError') {
    const errors = Object.values(error.errors).map(err => err.message);
    return res.status(400).json({ error: 'Validation Error', details: errors });
  }

  if (error.code === 11000) {
    const field = Object.keys(error.keyValue)[0];
    return res.status(400).json({ error: `${field} already exists` });
  }

  console.error(error);
  res.status(500).json({ error: 'Internal Server Error' });
});

// Routes
app.use('/api/users', require('./routes/users'));
app.use('/api/posts', require('./routes/posts'));

// Start server
const PORT = process.env.PORT || 3000;
app.listen(PORT, async () => {
  await connectDB();
  console.log(`🚀 Server running on port ${PORT}`);
});

module.exports = app;

// routes/users.js - Complete User Routes
const express = require('express');
const router = express.Router();
const AdvancedUser = require('../models/AdvancedUser');

// GET /api/users - Get all users with filtering and pagination
router.get('/', async (req, res, next) => {
  try {
    const {
      page = 1,
      limit = 10,
      role,
      search,
      sortBy = 'createdAt',
```

```
      sortOrder = 'desc'
    } = req.query;

    // Build query
    const query = {};
    if (role) query.role = role;
    if (search) {
      query.$or = [
        { username: { $regex: search, $options: 'i' } },
        { 'profile.firstName': { $regex: search, $options: 'i' } },
        { 'profile.lastName': { $regex: search, $options: 'i' } }
      ];
    }

    // Execute query with pagination
    const skip = (parseInt(page) - 1) * parseInt(limit);
    const sortOptions = { [sortBy]: sortOrder === 'desc' ? -1 : 1 };

    const [users, total] = await Promise.all([
      AdvancedUser.find(query)
        .select('-auditLog')
        .sort(sortOptions)
        .skip(skip)
        .limit(parseInt(limit))
        .populate('posts', 'title createdAt')
        .lean(),
      AdvancedUser.countDocuments(query)
    ]);

    res.json({
      users,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total,
        pages: Math.ceil(total / parseInt(limit))
      }
    });
  } catch (error) {
    next(error);
  }
});

// GET /api/users/:id - Get user by ID
```

```javascript
router.get('/:id', async (req, res, next) => {
  try {
    const user = await AdvancedUser.findById(req.params.id)
      .populate('posts')
      .populate('postCount');

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

    res.json(user);
  } catch (error) {
    next(error);
  }
});

// POST /api/users - Create new user
router.post('/', async (req, res, next) => {
  try {
    const user = new AdvancedUser(req.body);

    // Add audit context
    user.$locals = {
      currentUser: req.user?.id,
      ip: req.ip,
      userAgent: req.get('User-Agent')
    };

    await user.save();

    res.status(201).json(user.getPublicProfile());
  } catch (error) {
    next(error);
  }
});

// PUT /api/users/:id - Update user
router.put('/:id', async (req, res, next) => {
  try {
    const user = await AdvancedUser.findById(req.params.id);

    if (!user) {
      return res.status(404).json({ error: 'User not found' });
    }
```

```javascript
      // Store original values for audit
      user.$locals = {
        original: user.toObject(),
        currentUser: req.user?.id,
        ip: req.ip,
        userAgent: req.get('User-Agent')
      };

      // Update fields
      Object.keys(req.body).forEach(key => {
        if (key !== 'auditLog') {
          user[key] = req.body[key];
        }
      });

      await user.save();
      res.json(user);
    } catch (error) {
      next(error);
    }
  }
});

// DELETE /api/users/:id - Delete user
router.delete('/:id', async (req, res, next) => {
  try {
    const user = await AdvancedUser.findByIdAndDelete(req.params.id);

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

    res.json({ message: 'User deleted successfully' });
  } catch (error) {
    next(error);
  }
});

// GET /api/users/:id/stats - Get user statistics
router.get('/:id/stats', async (req, res, next) => {
  try {
    const stats = await AdvancedUser.aggregate([
      { $match: { _id: new mongoose.Types.ObjectId(req.params.id) } },
      {
```

```
    $lookup: {
        from: 'posts',
        localField: '_id',
        foreignField: 'author',
        as: 'userPosts'
    }
},
{
    $project: {
        username: 1,
        totalPosts: { $size: '$userPosts' },
        totalLikes: {
            $sum: {
                $map: {
                    input: '$userPosts',
                    as: 'post',
                    in: { $size: '$post.likes' }
                }
            }
        },
        totalComments: {
            $sum: {
                $map: {
                    input: '$userPosts',
                    as: 'post',
                    in: { $size: '$post.comments' }
                }
            }
        },
        avgLikesPerPost: {
            $cond: {
                if: { $gt: [{ $size: '$userPosts' }, 0] },
                then: {
                    $divide: [
                        {
                            $sum: {
                                $map: {
                                    input: '$userPosts',
                                    as: 'post',
                                    in: { $size: '$post.likes' }
                                }
                            }
                        },
                        { $size: '$userPosts' }
```

```javascript
                ]
              },
                else: 0
            }
          }
        }
      ]);

      if (stats.length === 0) {
        return res.status(404).json({ error: 'User not found' });
      }

      res.json(stats[0]);
    } catch (error) {
      next(error);
    }
  }
});

module.exports = router;

// routes/posts.js - Complete Post Routes
const express = require('express');
const router = express.Router();
const Post = require('../models/Post');
const AdvancedUser = require('../models/AdvancedUser');

// GET /api/posts - Get all posts with advanced filtering
router.get('/', async (req, res, next) => {
  try {
    const {
      page = 1,
      limit = 10,
      author,
      tags,
      search,
      sortBy = 'createdAt',
      sortOrder = 'desc',
      minLikes,
      dateFrom,
      dateTo
    } = req.query;

    // Build aggregation pipeline
```

```javascript
const pipeline = [];

// Match stage
const matchConditions = {};
if (author) matchConditions.author = new mongoose.Types.ObjectId(author);
if (tags) matchConditions.tags = { $in: tags.split(',') };
if (minLikes) matchConditions.$expr = { $gte: [{ $size: '$likes' }, parseInt(minLikes)] };
if (dateFrom || dateTo) {
    matchConditions.createdAt = {};
    if (dateFrom) matchConditions.createdAt.$gte = new Date(dateFrom);
    if (dateTo) matchConditions.createdAt.$lte = new Date(dateTo);
}
if (search) {
    matchConditions.$or = [
        { title: { $regex: search, $options: 'i' } },
        { content: { $regex: search, $options: 'i' } },
        { tags: { $elemMatch: { $regex: search, $options: 'i' } } }
    ];
}

if (Object.keys(matchConditions).length > 0) {
    pipeline.push({ $match: matchConditions });
}

// Add computed fields
pipeline.push({
    $addFields: {
        likeCount: { $size: '$likes' },
        commentCount: { $size: '$comments' },
        engagementScore: {
            $add: [
                { $size: '$likes' },
                { $multiply: [{ $size: '$comments' }, 2] }
            ]
        }
    }
});

// Lookup author
pipeline.push({
    $lookup: {
        from: 'advancedusers',
        localField: 'author',
        foreignField: '_id',
```

```javascript
      as: 'authorInfo'
    }
  });

  pipeline.push({ $unwind: '$authorInfo' });

  // Sort
  const sortStage = {};
  sortStage[sortBy] = sortOrder === 'desc' ? -1 : 1;
  pipeline.push({ $sort: sortStage });

  // Facet for pagination
  pipeline.push({
    $facet: {
      posts: [
        { $skip: (parseInt(page) - 1) * parseInt(limit) },
        { $limit: parseInt(limit) },
        {
          $project: {
            title: 1,
            content: { $substr: ['$content', 0, 200] },
            tags: 1,
            likeCount: 1,
            commentCount: 1,
            engagementScore: 1,
            createdAt: 1,
            updatedAt: 1,
            author: {
              _id: '$authorInfo._id',
              username: '$authorInfo.username',
              fullName: {
                $concat: [
                  '$authorInfo.profile.firstName',
                  ' ',
                  '$authorInfo.profile.lastName'
                ]
              }
            }
          }
        }
      ],
      totalCount: [{ $count: 'count' }]
    }
  });
```

```javascript
    const result = await Post.aggregate(pipeline);
    const posts = result[0].posts;
    const total = result[0].totalCount[0]?.count || 0;

    res.json({
        posts,
        pagination: {
            page: parseInt(page),
            limit: parseInt(limit),
            total,
            pages: Math.ceil(total / parseInt(limit))
        }
    });
  } catch (error) {
    next(error);
  }
});

// GET /api/posts/:id - Get post by ID with full details
router.get('/:id', async (req, res, next) => {
  try {
    const post = await Post.findById(req.params.id)
        .populate('author', 'username profile')
        .populate('comments.user', 'username profile.firstName profile.lastName')
        .populate('likes.user', 'username');

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

    res.json(post);
  } catch (error) {
    next(error);
  }
});

// POST /api/posts - Create new post
router.post('/', async (req, res, next) => {
  try {
    const { title, content, tags, authorId } = req.body;

    // Verify author exists
    const author = await AdvancedUser.findById(authorId);
```

```javascript
    if (!author) {
      return res.status(400).json({ error: 'Author not found' });
    }

    const post = new Post({
      title,
      content,
      author: authorId,
      tags: tags || []
    });

    await post.save();

    // Add post reference to user
    author.posts.push(post._id);
    await author.save();

    await post.populate('author', 'username profile');
    res.status(201).json(post);
  } catch (error) {
    next(error);
  }
});

// PUT /api/posts/:id - Update post
router.put('/:id', async (req, res, next) => {
  try {
    const { title, content, tags } = req.body;

    const post = await Post.findByIdAndUpdate(
      req.params.id,
      { title, content, tags },
      { new: true, runValidators: true }
    ).populate('author', 'username profile');

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

    res.json(post);
  } catch (error) {
    next(error);
  }
});
```

```javascript
// DELETE /api/posts/:id - Delete post
router.delete('/:id', async (req, res, next) => {
  const session = await mongoose.startSession();

  try {
    await session.withTransaction(async () => {
      const post = await Post.findByIdAndDelete(req.params.id).session(session);

      if (!post) {
        throw new Error('Post not found');
      }

      // Remove post reference from user
      await AdvancedUser.updateOne(
        { _id: post.author },
        { $pull: { posts: post._id } }
      ).session(session);
    });

    res.json({ message: 'Post deleted successfully' });
  } catch (error) {
    if (error.message === 'Post not found') {
      return res.status(404).json({ error: 'Post not found' });
    }
    next(error);
  } finally {
    await session.endSession();
  }
});

// POST /api/posts/:id/like - Like/unlike a post
router.post('/:id/like', async (req, res, next) => {
  try {
    const { userId } = req.body;
    const post = await Post.findById(req.params.id);

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

    const existingLikeIndex = post.likes.findIndex(
      like => like.user.toString() === userId
    );
```

```javascript
      if (existingLikeIndex > -1) {
        // Unlike
        post.likes.splice(existingLikeIndex, 1);
      } else {
        // Like
        post.likes.push({ user: userId });
      }

      await post.save();
      await post.populate('likes.user', 'username');

      res.json({
        liked: existingLikeIndex === -1,
        likeCount: post.likes.length,
        likes: post.likes
      });
    } catch (error) {
      next(error);
    }
});

// POST /api/posts/:id/comments - Add comment to post
router.post('/:id/comments', async (req, res, next) => {
  try {
    const { userId, text } = req.body;
    const post = await Post.findById(req.params.id);

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

    post.comments.push({
      user: userId,
      text,
      createdAt: new Date()
    });

    await post.save();
    await post.populate('comments.user', 'username profile.firstName profile.lastName');

    res.status(201).json(post.comments[post.comments.length - 1]);
  } catch (error) {
    next(error);
```

```javascript
    }
});

// GET /api/posts/analytics/engagement - Get engagement analytics
router.get('/analytics/engagement', async (req, res, next) => {
    try {
        const analytics = await Post.aggregate([
            {
                $addFields: {
                    likeCount: { $size: '$likes' },
                    commentCount: { $size: '$comments' },
                    engagementScore: {
                        $add: [
                            { $size: '$likes' },
                            { $multiply: [{ $size: '$comments' }, 2] }
                        ]
                    }
                }
            },
            {
                $group: {
                    _id: null,
                    totalPosts: { $sum: 1 },
                    avgLikes: { $avg: '$likeCount' },
                    avgComments: { $avg: '$commentCount' },
                    avgEngagement: { $avg: '$engagementScore' },
                    maxEngagement: { $max: '$engagementScore' },
                    topPosts: {
                        $push: {
                            $cond: {
                                if: { $gte: ['$engagementScore', 10] },
                                then: {
                                    _id: '$_id',
                                    title: '$title',
                                    engagementScore: '$engagementScore'
                                },
                                else: '$REMOVE'
                            }
                        }
                    }
                }
            },
            {
                $project: {
```

```
                    _id: 0,
                    totalPosts: 1,
                    avgLikes: { $round: ['$avgLikes', 2] },
                    avgComments: { $round: ['$avgComments', 2] },
                    avgEngagement: { $round: ['$avgEngagement', 2] },
                    maxEngagement: 1,
                    topPosts: { $slice: ['$topPosts', 10] }
                }
            }
        ]);

        res.json(analytics[0] || {});
    } catch (error) {
        next(error);
    }
});

module.exports = router;
```