Cloudinary+MongoDB

MongoDb with cloudinary:

File Name: mongodb-chatbot-cloudinary.js

```
// MongoDB Chatbot Implementation with Cloudinary Integration for MongoDB Compass
// Filename: mongodb-chatbot-cloudinary.js

// Required packages
// npm install mongodb cloudinary dotenv

const cloudinary = require('cloudinary').v2;
const { MongoClient, ObjectId } = require('mongodb');
require('dotenv').config();

// Configure Cloudinary
cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET
});

// Connect to MongoDB - Using local MongoDB Compass connection string
const uri = process.env.MONGODB_URI || "mongodb://localhost:27017/chatbot";
const client = new MongoClient(uri);

async function initDatabase() {
  try {
        await client.connect();
        console.log("Connected to MongoDB via Compass");
        const db = client.db();

        // Create collections and indexes
        await setupCollections(db);
        return db;
  } catch (error) {
        console.error("Database initialization error:", error);
        throw error;
  }
}
```

```
async function setupCollections(db) {
  // User Collection
  try {
        await db.createCollection("users", {
        validator: {
        $jsonSchema: {
        bsonType: "object",
        required: ["username", "preferences", "createdAt"],
        properties: {
        username: {
        bsonType: "string",
        description: "Username must be a string and is required"
        },
        preferences: {
        bsonType: "object",
        properties: {
                theme: { bsonType: "string", enum: ["light", "dark", "system"] },
                notifications: { bsonType: "bool" },
                languagePreference: { bsonType: "string" }
        }
        },
        createdAt: { bsonType: "date" },
        lastLogin: { bsonType: "date" }
        }
        }
        }
        });
  } catch (error) {
        // Collection may already exist in Compass
        console.log("Users collection setup:", error.message);
  }

  // Sessions Collection
  try {
        await db.createCollection("sessions", {
        validator: {
        $jsonSchema: {
        bsonType: "object",
        required: ["userId", "startTime", "status"],
        properties: {
        userId: { bsonType: "objectId" },
        startTime: { bsonType: "date" },
        endTime: { bsonType: "date" },
```

```
        status: { bsonType: "string", enum: ["active", "closed"] },
        deviceInfo: { bsonType: "object" }
        }
        }
        }
        });
} catch (error) {
        console.log("Sessions collection setup:", error.message);
}

// Chat Collection
try {
        await db.createCollection("chats", {
        validator: {
        $jsonSchema: {
        bsonType: "object",
        required: ["sessionId", "timestamp", "type"],
        properties: {
        sessionId: { bsonType: "objectId" },
        timestamp: { bsonType: "date" },
        type: { bsonType: "string", enum: ["query", "response", "feedback", "context"] },
        content: { bsonType: "string" },
        metadata: {
        bsonType: "object",
        properties: {
                contextDocs: { bsonType: "array", items: { bsonType: "objectId" } },
                confidence: { bsonType: "double" },
                processingTime: { bsonType: "double" },
                rating: { bsonType: "int" },
                query: { bsonType: "string" }
        }
        }
        }
        }
        }
        });
} catch (error) {
        console.log("Chats collection setup:", error.message);
}

// Documents Collection
try {
        await db.createCollection("documents", {
        validator: {
```

```
        $jsonSchema: {
        bsonType: "object",
        required: ["title", "createdAt"],
        properties: {
        title: { bsonType: "string" },
        description: { bsonType: "string" },
        content: { bsonType: "string" },
        createdAt: { bsonType: "date" },
        updatedAt: { bsonType: "date" },
        category: { bsonType: "string" },
        tags: { bsonType: "array", items: { bsonType: "string" } },
        cloudinaryId: { bsonType: "string" },
        cloudinaryUrl: { bsonType: "string" },
        fileType: { bsonType: "string" },
        fileSize: { bsonType: "int" },
        textContent: { bsonType: "string" }
        // No vector field for local MongoDB Compass installation
        }
        }
        }
        });
 } catch (error) {
        console.log("Documents collection setup:", error.message);
 }

 // Create indexes - compatible with MongoDB Compass
 try {
        await db.collection('users').createIndex({ username: 1 }, { unique: true });
        await db.collection('sessions').createIndex({ userId: 1 });
        await db.collection('sessions').createIndex({ startTime: 1 });
        await db.collection('chats').createIndex({ sessionId: 1 });
        await db.collection('chats').createIndex({ timestamp: 1 });
        await db.collection('documents').createIndex({ title: "text", textContent: "text",
description: "text" });
        await db.collection('documents').createIndex({ tags: 1 });
        await db.collection('documents').createIndex({ category: 1 });
        await db.collection('documents').createIndex({ createdAt: 1 });
        await db.collection('documents').createIndex({ cloudinaryId: 1 });

        console.log("Collections and indexes created successfully");
 } catch (error) {
        console.error("Error creating indexes:", error);
 }
}
```

```
// === USER FUNCTIONS ===

// Create a new user
async function createUser(db, userData) {
  try {
        const user = {
        ...userData,
        preferences: userData.preferences || {
        theme: "system",
        notifications: true,
        languagePreference: "en"
        },
        createdAt: new Date(),
        lastLogin: new Date()
        };

        const result = await db.collection('users').insertOne(user);
        return { userId: result.insertedId, ...user };
  } catch (error) {
        console.error("Error creating user:", error);
        throw error;
  }
}

// Update user preferences
async function updateUserPreferences(db, userId, preferences) {
  try {
        const result = await db.collection('users').updateOne(
        { _id: new ObjectId(userId) },
        {
        $set: {
        preferences,
        lastLogin: new Date()
        }
        }
        );

        return {
        success: result.modifiedCount > 0,
        message: result.modifiedCount > 0 ? 'Preferences updated' : 'No changes made'
        };
  } catch (error) {
        console.error("Error updating user preferences:", error);
```

```
        throw error;
  }
}

// === SESSION FUNCTIONS ===

// Create a new chat session
async function createChatSession(db, userId, deviceInfo = {}) {
  try {
        const session = {
        userId: new ObjectId(userId),
        startTime: new Date(),
        status: "active",
        deviceInfo
        };

        const result = await db.collection('sessions').insertOne(session);
        return { sessionId: result.insertedId, ...session };
  } catch (error) {
        console.error("Error creating chat session:", error);
        throw error;
  }
}

// Close a chat session
async function closeSession(db, sessionId) {
  try {
        const result = await db.collection('sessions').updateOne(
        { _id: new ObjectId(sessionId) },
        {
        $set: {
        status: "closed",
        endTime: new Date()
        }
        }
        );

        return {
        success: result.modifiedCount > 0,
        message: result.modifiedCount > 0 ? 'Session closed' : 'No changes made'
        };
  } catch (error) {
        console.error("Error closing session:", error);
        throw error;
```

```
  }
}

// Get active sessions for a user
async function getUserActiveSessions(db, userId) {
  try {
        return await db.collection('sessions')
        .find({
        userId: new ObjectId(userId),
        status: "active"
        })
        .sort({ startTime: -1 })
        .toArray();
  } catch (error) {
        console.error("Error getting active sessions:", error);
        throw error;
  }
}

// === CHAT FUNCTIONS ===

// Add a query to chat
async function addChatQuery(db, sessionId, queryText) {
  try {
        const chat = {
        sessionId: new ObjectId(sessionId),
        timestamp: new Date(),
        type: "query",
        content: queryText
        };

        const result = await db.collection('chats').insertOne(chat);
        return { chatId: result.insertedId, ...chat };
  } catch (error) {
        console.error("Error adding chat query:", error);
        throw error;
  }
}

// Add a chatbot response with context documents
async function addChatResponse(db, sessionId, responseText, contextDocIds,
confidence) {
  try {
        const chat = {
```

```
        sessionId: new ObjectId(sessionId),
        timestamp: new Date(),
        type: "response",
        content: responseText,
        metadata: {
        contextDocs: contextDocIds.map(id => new ObjectId(id)),
        confidence: confidence || 0.0,
        processingTime: 235.5 // Placeholder for actual processing time
        }
        };

        const result = await db.collection('chats').insertOne(chat);
        return { chatId: result.insertedId, ...chat };
  } catch (error) {
        console.error("Error adding chat response:", error);
        throw error;
  }
}

// Add user feedback for a chat interaction
async function addChatFeedback(db, sessionId, feedbackText, rating) {
  try {
        const chat = {
        sessionId: new ObjectId(sessionId),
        timestamp: new Date(),
        type: "feedback",
        content: feedbackText || "",
        metadata: {
        rating: rating // e.g., 1-5 stars
        }
        };

        const result = await db.collection('chats').insertOne(chat);
        return { chatId: result.insertedId, ...chat };
  } catch (error) {
        console.error("Error adding chat feedback:", error);
        throw error;
  }
}

// Get chat history for a session
async function getSessionChatHistory(db, sessionId, limit = 50) {
  try {
        return await db.collection('chats')
```

```
        .find({ sessionId: new ObjectId(sessionId) })
        .sort({ timestamp: 1 })
        .limit(limit)
        .toArray();
  } catch (error) {
        console.error("Error getting session chat history:", error);
        throw error;
  }
}

// Get recent chat history for a user across all sessions
async function getUserChatHistory(db, userId, limit = 20) {
  try {
        // Get the user's recent sessions
        const sessions = await db.collection('sessions').find({
        userId: new ObjectId(userId)
        }).sort({ startTime: -1 }).limit(5).toArray();

        if (sessions.length === 0) {
        return [];
        }

        // Get chats for these sessions
        const sessionIds = sessions.map(session => session._id);
        return await db.collection('chats').find({
        sessionId: { $in: sessionIds }
        }).sort({ timestamp: -1 }).limit(limit).toArray();
  } catch (error) {
        console.error("Error getting user chat history:", error);
        throw error;
  }
}

// === DOCUMENT FUNCTIONS WITH CLOUDINARY INTEGRATION ===

// Upload document to Cloudinary and save reference in MongoDB
async function uploadDocument(db, filePath, documentMetadata) {
  try {
        // Upload file to Cloudinary
        const cloudinaryResult = await cloudinary.uploader.upload(filePath, {
        resource_type: 'auto',
        folder: 'chatbot_documents',
        use_filename: true,
        unique_filename: true
```

```
        });

        // Save document reference to MongoDB
        const documentRecord = {
        title: documentMetadata.title,
        description: documentMetadata.description || '',
        content: documentMetadata.content || '',
        createdAt: new Date(),
        updatedAt: new Date(),
        category: documentMetadata.category || 'uncategorized',
        tags: documentMetadata.tags || [],
        cloudinaryId: cloudinaryResult.public_id,
        cloudinaryUrl: cloudinaryResult.secure_url,
        fileType: cloudinaryResult.format,
        fileSize: cloudinaryResult.bytes,
        textContent: documentMetadata.textContent || '' // Should be extracted from the
document
        };

        const result = await db.collection('documents').insertOne(documentRecord);
        return {
        documentId: result.insertedId,
        cloudinaryId: cloudinaryResult.public_id,
        url: cloudinaryResult.secure_url
        };
  } catch (error) {
        console.error('Error uploading document:', error);
        throw error;
  }
}

// Delete document from both Cloudinary and MongoDB
async function deleteDocument(db, documentId) {
  try {
        // Find document in MongoDB to get Cloudinary ID
        const document = await db.collection('documents').findOne({ _id: new
ObjectId(documentId) });

        if (!document) {
        throw new Error('Document not found');
        }

        // Delete from Cloudinary if cloudinaryId exists
        if (document.cloudinaryId) {
```

```
        await cloudinary.uploader.destroy(document.cloudinaryId);
        }

        // Delete from MongoDB
        await db.collection('documents').deleteOne({ _id: new ObjectId(documentId) });

        return { success: true, message: 'Document deleted successfully' };
  } catch (error) {
        console.error('Error deleting document:', error);
        throw error;
  }
}

// Update document metadata in MongoDB
async function updateDocumentMetadata(db, documentId, updates) {
  try {
        const updateData = {
        ...updates,
        updatedAt: new Date()
        };

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

        return {
        success: result.modifiedCount > 0,
        message: result.modifiedCount > 0 ? 'Document updated successfully' : 'No
changes made'
        };
  } catch (error) {
        console.error('Error updating document metadata:', error);
        throw error;
  }
}

// Get document with its metadata
async function getDocument(db, documentId) {
  try {
        return await db.collection('documents').findOne({ _id: new ObjectId(documentId)
});
  } catch (error) {
        console.error('Error fetching document:', error);
```

```
        throw error;
  }
}

// Search documents by text content - compatible with MongoDB Compass
async function searchDocuments(db, searchText, options = {}) {
  try {
        const query = searchText ?
        { $text: { $search: searchText } } :
        {};

        const limit = options.limit || 10;
        const skip = options.skip || 0;

        let cursor;
        if (searchText) {
        cursor = db.collection('documents')
        .find(query)
        .project({ score: { $meta: "textScore" } })
        .sort({ score: { $meta: "textScore" } });
        } else {
        cursor = db.collection('documents')
        .find(query)
        .sort({ createdAt: -1 });
        }

        return await cursor.skip(skip).limit(limit).toArray();
  } catch (error) {
        console.error('Error searching documents:', error);
        throw error;
  }
}

// Filter documents by tags and categories
async function filterDocuments(db, filters = {}, options = {}) {
  try {
        const query = {};

        if (filters.tags && filters.tags.length > 0) {
        query.tags = { $all: filters.tags };
        }

        if (filters.category) {
        query.category = filters.category;
```

```
        }

        if (filters.dateRange) {
        query.createdAt = {
        $gte: new Date(filters.dateRange.start),
        $lte: new Date(filters.dateRange.end)
        };
        }

        const limit = options.limit || 20;
        const skip = options.skip || 0;
        const sort = options.sort || { createdAt: -1 };

        return await db.collection('documents')
        .find(query)
        .sort(sort)
        .skip(skip)
        .limit(limit)
        .toArray();
  } catch (error) {
        console.error('Error filtering documents:', error);
        throw error;
  }
}

// Get contextual documents for a query and attach to chat session
async function addDocumentsToChat(db, sessionId, query) {
  try {
        // Find relevant documents based on the query
        const relevantDocs = await searchDocuments(db, query, { limit: 3 });
        const docIds = relevantDocs.map(doc => doc._id);

        // Add documents to chat context
        if (docIds.length > 0) {
        await db.collection('chats').insertOne({
        sessionId: new ObjectId(sessionId),
        timestamp: new Date(),
        type: "context",
        content: "",
        metadata: {
        contextDocs: docIds,
        query: query
        }
        });
```

```
        }

        return relevantDocs;
  } catch (error) {
        console.error('Error adding documents to chat:', error);
        throw error;
  }
}

// === EXAMPLE FLOW FOR CHAT WITH DOCUMENT INTEGRATION ===

// Sample function showing the complete flow from user query to response
async function processUserQuery(db, sessionId, queryText) {
  try {
        // 1. Log the user query
        await addChatQuery(db, sessionId, queryText);

        // 2. Find relevant documents for the query
        const relevantDocs = await addDocumentsToChat(db, sessionId, queryText);

        // 3. Generate a response (in a real system, this would involve an LLM or similar)
        const responseText = `Here is information related to your query: "${queryText}"`;
        const confidence = 0.85; // Sample confidence score

        // 4. Store the response with references to the relevant documents
        const response = await addChatResponse(
        db,
        sessionId,
        responseText,
        relevantDocs.map(doc => doc._id),
        confidence
        );

        // 5. Return the complete response with context
        return {
        response,
        contextDocuments: relevantDocs
        };
  } catch (error) {
        console.error('Error processing user query:', error);
        throw error;
  }
}
```

```
// === UTILITY FUNCTIONS FOR MONGODB COMPASS ===

// List all collections in the database
async function listCollections(db) {
  try {
        const collections = await db.listCollections().toArray();
        return collections.map(col => col.name);
  } catch (error) {
        console.error('Error listing collections:', error);
        throw error;
  }
}

// Get collection stats
async function getCollectionStats(db, collectionName) {
  try {
        return await db.command({ collStats: collectionName });
  } catch (error) {
        console.error(`Error getting stats for collection ${collectionName}:`, error);
        throw error;
  }
}

// === MAIN EXECUTION EXAMPLE ===

// Example of how to use the system
async function main() {
  let db;
  try {
        // Initialize the database
        db = await initDatabase();

        console.log("Available collections:", await listCollections(db));

        // Create a user
        const user = await createUser(db, {
        username: "testuser@example.com",
        preferences: {
        theme: "dark",
        notifications: true,
        languagePreference: "en"
        }
        });
```

```
        // Create a session
        const session = await createChatSession(db, user.userId, {
        deviceType: "web",
        browser: "Chrome",
        os: "Windows"
        });

        // Example document upload (commented out since it requires a real file)
        /*
        const document = await uploadDocument(db, "./sample_doc.pdf", {
        title: "FAQ Knowledge Base",
        description: "Common questions and answers",
        category: "support",
        tags: ["faq", "help", "support"]
        });
        */

        // Manually create a document record for testing
        const docResult = await db.collection('documents').insertOne({
        title: "FAQ Knowledge Base",
        description: "Common questions and answers about password reset",
        content: "To reset your password, go to the login page and click 'Forgot
Password'",
        textContent: "password reset login forgot password reset link email",
        createdAt: new Date(),
        updatedAt: new Date(),
        category: "support",
        tags: ["faq", "help", "support", "password"],
        cloudinaryId: "test_id",
        cloudinaryUrl: "https://example.com/test.pdf",
        fileType: "pdf",
        fileSize: 1024
        });

        // Process a user query
        const result = await processUserQuery(db, session.sessionId, "How do I reset my
password?");

        console.log("Query processed successfully:", result);

        // Add user feedback
        await addChatFeedback(db, session.sessionId, "This was helpful", 5);

        // Close the session
```

```
        await closeSession(db, session.sessionId);

  } catch (error) {
        console.error("Error in main function:", error);
  } finally {
        // Close the MongoDB connection
        if (client) {
        await client.close();
        console.log("MongoDB connection closed");
        }
  }
}

// Export all functions for use in other modules
module.exports = {
  initDatabase,
  createUser,
  updateUserPreferences,
  createChatSession,
  closeSession,
  getUserActiveSessions,
  addChatQuery,
  addChatResponse,
  addChatFeedback,
  getSessionChatHistory,
  getUserChatHistory,
  uploadDocument,
  deleteDocument,
  updateDocumentMetadata,
  getDocument,
  searchDocuments,
  filterDocuments,
  addDocumentsToChat,
  processUserQuery,
  listCollections,
  getCollectionStats
};

// Uncomment to run the example
// main().catch(console.error);
```