Task Manager App with MongoD

This guide will walk you through setting up a **Task Manager App** using **Go (Gin framework)** with **MongoDB** and **React**. The guide assumes MongoDB uses \_id for the document ID, so we will adjust the backend and frontend accordingly.

# 1. MongoDB Installation and Setup

### 1.1 Install MongoDB

Follow the installation instructions based on your operating system:

**Windows**: Download and install MongoDB from [MongoDB Download Center](https://www.mongodb.com/try/download/community).

### 1.2 Start MongoDB

Start the MongoDB server:

# 2. Go Backend with MongoDB Integration

### 2.1 Install Dependencies

Install the necessary Go packages:

|  |
| --- |
| go get go.mongodb.org/mongo-driver/mongo  go get go.mongodb.org/mongo-driver/mongo/options  go get github.com/gin-gonic/gin  go get github.com/gin-contrib/cors |

### 2.2 Create main.go

Create a file named main.go with the following code:

|  |
| --- |
| package main  import (  "context"  "log"  "net/http"  "time"  "github.com/gin-gonic/gin"  "github.com/gin-contrib/cors"  "go.mongodb.org/mongo-driver/bson"  "go.mongodb.org/mongo-driver/bson/primitive"  "go.mongodb.org/mongo-driver/mongo"  "go.mongodb.org/mongo-driver/mongo/options"  )  // MongoDB configuration  var mongoUri string = "mongodb://localhost:27017"  var mongoDbName string = "taskmanager\_db"  var mongoCollectionTask string = "tasks"  var taskCollection \*mongo.Collection  // Task model  type Task struct {  ID primitive.ObjectID `bson:"\_id,omitempty" json:"id"`  Title string `bson:"title" json:"title"`  Status string `bson:"status" json:"status"`  }  // Connect to MongoDB  func connectDB() {  client, err := mongo.Connect(context.TODO(), options.Client().ApplyURI(mongoUri))  if err != nil {  log.Fatal("MongoDB Connection Error:", err)  }  // Test the connection  err = client.Ping(context.TODO(), nil)  if err != nil {  log.Fatal("Failed to ping MongoDB:", err)  }  taskCollection = client.Database(mongoDbName).Collection(mongoCollectionTask)  log.Println("Connected to MongoDB!")  }  // CRUD Handlers  func createTask(c \*gin.Context) {  var task Task  if err := c.BindJSON(&task); err != nil {  c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid task data"})  return  }  task.ID = primitive.NewObjectID()  \_, err := taskCollection.InsertOne(context.TODO(), task)  if err != nil {  c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to create task"})  return  }  c.JSON(http.StatusCreated, task)  }  func getTasks(c \*gin.Context) {  cursor, err := taskCollection.Find(context.TODO(), bson.M{})  if err != nil {  c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to fetch tasks"})  return  }  defer cursor.Close(context.TODO())  var tasks []Task  if err = cursor.All(context.TODO(), &tasks); err != nil {  c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to parse tasks"})  return  }  c.JSON(http.StatusOK, tasks)  }  func getTaskByID(c \*gin.Context) {  id := c.Param("id")  objectID, err := primitive.ObjectIDFromHex(id)  if err != nil {  c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid task ID"})  return  }  var task Task  err = taskCollection.FindOne(context.TODO(), bson.M{"\_id": objectID}).Decode(&task)  if err != nil {  c.JSON(http.StatusNotFound, gin.H{"error": "Task not found"})  return  }  c.JSON(http.StatusOK, task)  }  func updateTask(c \*gin.Context) {  id := c.Param("id")  objectID, err := primitive.ObjectIDFromHex(id)  if err != nil {  c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid task ID"})  return  }  var updatedTask Task  if err := c.BindJSON(&updatedTask); err != nil {  c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid task data"})  return  }  filter := bson.M{"\_id": objectID}  update := bson.M{  "$set": bson.M{  "title": updatedTask.Title,  "status": updatedTask.Status,  },  }  \_, err = taskCollection.UpdateOne(context.TODO(), filter, update)  if err != nil {  c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to update task"})  return  }  c.JSON(http.StatusOK, gin.H{"message": "Task updated successfully"})  }  func deleteTask(c \*gin.Context) {  id := c.Param("id")  objectID, err := primitive.ObjectIDFromHex(id)  if err != nil {  c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid task ID"})  return  }  \_, err = taskCollection.DeleteOne(context.TODO(), bson.M{"\_id": objectID})  if err != nil {  c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to delete task"})  return  }  c.JSON(http.StatusOK, gin.H{"message": "Task deleted successfully"})  }  func main() {  connectDB()  r := gin.Default()  r.Use(cors.Default())  r.POST("/tasks", createTask)  r.GET("/tasks", getTasks)  r.GET("/tasks/:id", getTaskByID)  r.PUT("/tasks/:id", updateTask)  r.DELETE("/tasks/:id", deleteTask)  r.Run(":8080")  } |

## Explanation of Task Manager API Code

This code sets up a **Task Manager API** using the **Go Gin framework** and **MongoDB**. It includes endpoints for creating, reading, updating, and deleting tasks (CRUD operations).

### 1. Configuration

|  |
| --- |
| var mongoUri string = "mongodb://localhost:27017"  var mongoDbName string = "taskmanager\_db"  var mongoCollectionTask string = "tasks"  var taskCollection \*mongo.Collection |

* mongoUri: The MongoDB connection URI.
* mongoDbName: The name of the MongoDB database (taskmanager\_db).
* mongoCollectionTask: The collection name (tasks) where tasks are stored.
* taskCollection: A pointer to the MongoDB collection, used for database operations.

### 2. Task Model

|  |
| --- |
| type Task struct {  ID primitive.ObjectID `bson:"\_id,omitempty" json:"id"`  Title string `bson:"title" json:"title"`  Status string `bson:"status" json:"status"`  } |

* ID: The MongoDB ObjectID (\_id). It is automatically generated if not provided.
* Title: The title of the task.
* Status: The status of the task (e.g., "Pending" or "Completed").
* The bson tag specifies how the field is stored in MongoDB, and the json tag specifies how it is serialized to JSON for API responses.

The backend automatically **maps MongoDB's \_id field to id in the JSON response** sent to the React frontend. This means:

1. **MongoDB Storage**: The field will still be stored as \_id in MongoDB.
2. **API Responses**: The field will be serialized as id in the JSON response sent to the React frontend.

### What This Means for Your Frontend

Because the backend is now returning id instead of \_id:

* You can continue using task.id in the frontend (e.g., in TaskList, TaskView, and TaskUpdate).
* No updates are needed in the React components to reference task.\_id.

### 3. Database Connection

|  |
| --- |
| func connectDB() {  client, err := mongo.Connect(context.TODO(), options.Client().ApplyURI(mongoUri))  if err != nil {  log.Fatal("MongoDB Connection Error:", err)  }  // Test the connection  err = client.Ping(context.TODO(), nil)  if err != nil {  log.Fatal("Failed to ping MongoDB:", err)  }  taskCollection = client.Database(mongoDbName).Collection(mongoCollectionTask)  log.Println("Connected to MongoDB!")  } |

* mongo.Connect: Connects to the MongoDB server using the specified URI.
* client.Ping: Pings the database to ensure the connection is working.
* taskCollection: Stores the reference to the tasks collection for CRUD operations.

### 4. CRUD Handlers

**4.1 Create a Task (POST /tasks)**

|  |
| --- |
| func createTask(c \*gin.Context) {  var task Task  if err := c.BindJSON(&task); err != nil {  c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid task data"})  return  }  task.ID = primitive.NewObjectID() // Generate a new ObjectID  \_, err := taskCollection.InsertOne(context.TODO(), task)  if err != nil {  c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to create task"})  return  }  c.JSON(http.StatusCreated, task)  } |

* **Bind JSON**: Parses the incoming JSON request into a Task struct.
* **Insert Task**: Inserts the task into the MongoDB collection.
* **Response**: Returns the created task as a JSON response with status 201 Created.

**4.2 Get All Tasks (GET /tasks)**

|  |
| --- |
| func getTasks(c \*gin.Context) {  cursor, err := taskCollection.Find(context.TODO(), bson.M{})  if err != nil {  c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to fetch tasks"})  return  }  defer cursor.Close(context.TODO())  var tasks []Task  if err = cursor.All(context.TODO(), &tasks); err != nil {  c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to parse tasks"})  return  }  c.JSON(http.StatusOK, tasks)  } |

* **Find All**: taskCollection.Find fetches all tasks from the MongoDB collection (bson.M{} means no filter).
* **Cursor Iteration**: The cursor is used to iterate through the documents.
* **Response**: Returns all tasks as a JSON array with status 200 OK.

**4.3 Get Task by ID (GET /tasks/:id)**

|  |
| --- |
| func getTaskByID(c \*gin.Context) {  id := c.Param("id")  objectID, err := primitive.ObjectIDFromHex(id)  if err != nil {  c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid task ID"})  return  }  var task Task  err = taskCollection.FindOne(context.TODO(), bson.M{"\_id": objectID}).Decode(&task)  if err != nil {  c.JSON(http.StatusNotFound, gin.H{"error": "Task not found"})  return  }  c.JSON(http.StatusOK, task)  } |

* **Get ID from URL**: c.Param("id") retrieves the task ID from the URL.
* **Convert to ObjectID**: Converts the string ID to a MongoDB ObjectID.
* **Find by ID**: Searches for a document with the matching \_id field.
* **Response**: Returns the found task or an error if the task is not found.

**4.4 Update a Task (PUT /tasks/:id)**

|  |
| --- |
| func updateTask(c \*gin.Context) {  id := c.Param("id")  objectID, err := primitive.ObjectIDFromHex(id)  if err != nil {  c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid task ID"})  return  }  var updatedTask Task  if err := c.BindJSON(&updatedTask); err != nil {  c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid task data"})  return  }  filter := bson.M{"\_id": objectID}  update := bson.M{  "$set": bson.M{  "title": updatedTask.Title,  "status": updatedTask.Status,  },  }  \_, err = taskCollection.UpdateOne(context.TODO(), filter, update)  if err != nil {  c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to update task"})  return  }  c.JSON(http.StatusOK, gin.H{"message": "Task updated successfully"})  } |

* **Convert ID**: Converts the task ID to ObjectID.
* **Bind JSON**: Parses the updated task data from the request.
* **Update**: Updates the task with the new title and status.
* **Response**: Returns a success message if the update is successful.

**4.5 Delete a Task (DELETE /tasks/:id)**

|  |
| --- |
| func deleteTask(c \*gin.Context) {  id := c.Param("id")  objectID, err := primitive.ObjectIDFromHex(id)  if err != nil {  c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid task ID"})  return  }  \_, err = taskCollection.DeleteOne(context.TODO(), bson.M{"\_id": objectID})  if err != nil {  c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to delete task"})  return  }  c.JSON(http.StatusOK, gin.H{"message": "Task deleted successfully"})  } |

* **Convert ID**: Converts the task ID to ObjectID.
* **Delete**: Deletes the task with the matching \_id.
* **Response**: Returns a success message if the task is deleted.

### 5. Main Function

|  |
| --- |
| func main() {  connectDB()  r := gin.Default()  r.Use(cors.Default())  r.POST("/tasks", createTask)  r.GET("/tasks", getTasks)  r.GET("/tasks/:id", getTaskByID)  r.PUT("/tasks/:id", updateTask)  r.DELETE("/tasks/:id", deleteTask)  r.Run(":8080")  } |

* **connectDB**: Connects to MongoDB.
* **Gin Router**: Sets up the Gin router and enables CORS.
* **Routes**: Defines routes for CRUD operations.
* **Start Server**: Starts the server on localhost:8080.

### Summary

This code implements a fully functional **Task Manager API** using **Go, Gin, and MongoDB**. It supports CRUD operations for managing tasks and returns appropriate JSON responses. Let me know if you need further clarifications or modifications!

# 3. Start the Application

### 4.1 Start MongoDB

|  |
| --- |
| mongod |

### 4.2 Run the Go API

|  |
| --- |
| go run main.go |

### 4.3 Run the React Frontend

|  |
| --- |
| npm run dev |