# **Todo List API Documentation**

## **Overview**

The Todo List API is a backend service designed to manage a list of tasks. It provides endpoints for creating, retrieving, updating, and deleting tasks. The API uses **Express.js** for routing and **MongoDB** for data storage. This documentation outlines its setup, functionality, and endpoints.

## **Features**

* **CRUD Operations**:
  + Create new todos.
  + Retrieve all existing todos.
  + Update the completion status of a todo.
  + Delete a todo.
* **RESTful Architecture**.
* **JSON-based communication**.
* **MongoDB for database storage**.
* **Error handling for robust operation**.

## **Setup Instructions**

### **Prerequisites**

1. **Node.js** and **npm** installed.
2. **MongoDB** instance is running locally or is accessible via a cloud service.
3. Defined your MongoDB URL. line number 7 const url = 'mongodb://localhost:27017/todo-app';

### **Steps**

**Clone the Repository**:  
git clone https://github.com/akashmondaldev/bwu-dcs-22-017-todo-app.git

1. **Install Dependencies**:  
   npm install
2. **Start the Server**:  
   npm start  
   The server will start on the specified port (default: 3000).

## **API Endpoints**

### [**Postman Link**](https://documenter.getpostman.com/view/20935088/2sAYJ9AJEQ)

### **1. Get All Todos**

#### **Request**

* **Method**: GET
* **Endpoint**: /todos

#### **Response**

* **Status**: 200 OK

**Body**:  
[

{

"\_id": "63f2f3e56abc1234",

"title": "Sample Todo",

"completed": false

}

]

#### **Error Handling**

**Status**: 500 Internal Server Error  
{

"message": "Error message"

}

### **2. Update Todo Completion Status**

#### **Request**

* **Method**: PUT
* **Endpoint**: /todos/:id

**Body**:  
{

"completed": true

}

#### **Response**

* **Status**: 200 OK

**Body**:  
{

"\_id": "63f2f3e56abc1234",

"title": "Sample Todo",

"completed": true

}

#### **Error Handling**

**Status**: 404 Not Found  
{

"message": "Todo not found"

}

**Status**: 500 Internal Server Error  
{

"message": "Error message"

}

### **3. Delete a Todo**

#### **Request**

* **Method**: DELETE
* **Endpoint**: /todos/:id

#### **Response**

**Status**: 200 OK

**Body**:  
{

"message": "Todo deleted"

}

#### **Error Handling**

**Status**: 404 Not Found  
{

"message": "Todo not found"

}

**Status**: 500 Internal Server Error  
{

"message": "Error message"

}

## **Code Walkthrough**

### **Database Connection**

The API connects to MongoDB using the Mongoose library. Connection strings are configurable via the .env file.

mongoose.connect(process.env.MONGODB\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true,

});

### **Todo Schema**

A simple schema is used to define the structure of todos:

const todoSchema = new mongoose.Schema({

title: { type: String, required: true },

completed: { type: Boolean, default: false },

});

### **Error Handling**

All endpoints use try-catch blocks to handle errors, ensuring proper HTTP status codes and messages.

try {

// Logic here

} catch (error) {

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

}

## **Future Enhancements**

1. **Add User Authentication**:
   * Implement JWT-based authentication for secure access.
2. **Pagination**:
   * Enable pagination for the /todos endpoint to handle large datasets.
3. **Sorting and Filtering**:
   * Allow sorting by title or status and filtering by completion status.
4. **Unit Tests**:
   * Add testing for each endpoint using tools like Jest or Mocha.
5. **Frontend Integration**:
   * Create an HTML, CSS, and JS frontend for a complete user interface.

This documentation covers the current functionality of the Todo List API and provides insights for developers to understand, extend, and maintain the system.