Line-by-Line Explanation of Express.js TODO API

1. Import Modules

const express = require('express');

→ Imports the Express.js framework to create the web server.

const mongoose = require('mongoose');

→ Imports Mongoose for MongoDB operations (not used in this example).

2. Create App and Middleware

const app = express();

→ Initializes an Express application instance.

app.use(express.json());

→ Middleware to parse incoming JSON requests. Enables `req.body`.

3. In-Memory Data Store

const taskList = [];

→ An array acting as a temporary, in-memory database to store tasks.

4. POST /todo - Add a New Task

• Extracts `title` and `description` from request body.

• Assigns an `id` based on `taskList.length`.

• Pushes the new task to `taskList`.

• Returns status 201 with the updated list.

5. GET /todo - Get All Tasks

• Simply returns the entire `taskList` array.

• Status code: 200 (OK).

6. PUT /todo/:id - Update Task

• Extracts `id` from request parameters.

• Finds the task using `.find()` or `.findIndex()`.

• If not found, returns status 404.

• Updates the task and replaces it in the array.

• Returns status 200 with updated list.

7. DELETE /todo/:id - Delete Task

• Extracts `id` from request parameters.

• Finds the index using `.findIndex()`.

• If not found, returns 404.

• Deletes the task using `.splice()`.

• Returns 200 with updated task list.

8. Start the Server

• `app.listen(3000, ...)` starts the server.

• Logs the message to indicate the server is running.

9. Why MongoDB is Preferred in Real Projects

• In-memory arrays are lost when the server restarts.

• MongoDB provides data persistence and reliability.

• Handles concurrent connections and scales well.

• Mongoose allows defining schemas, validation, and query abstraction.

• Suitable for production; taskList[] is only for practice/demo purposes.