Step 1: Todo App

Simplistic Todo App

Features

- This is a simple Todo application using 3-tier architecture.
- It is built with Node.js, Express, and EJS templating engine.
- It allows users to create, read, update, and delete todo items using REST API.
- The information is stored in MongoDB database.

Data Model

The `util/util.js` has the data model for the todo items.

```
const newPost = {
  _id: newId,
  title: req.body.title,
  date: req.body.date
};
```

- Data model should be more complex in real-world applications.
- We should support better functions for communication and storage for maintainability and extensibility.

Counter for Auto-Increment IDs

This is for keeping track of auto-incrementing IDs.

- Instead of manually managing IDs, we use a separate collection to store the current count.
- This way, we don't need complex logic to find the max ID each time we add a new post.

```
const result = await counter.findOneAndUpdate(
  { name: 'count' },
  { $inc: { count: 1 } },
  { returnDocument: 'after', upsert: true }
);
```

Running Application

- Install packages and nodemon (globally), and run the application.
- With nodemon, the server will restart automatically when code changes.

```
npm install
npm install -g nodemon
nodemon ./index.js
```

Directory Structure

```
.
├── index.js
├── public
│   └── todo.png
├── tests
│   ├── db.simple.test.js
│   └── db.test.js
├── util
│   ├── db.js
│   ├── uri.js
│   └── util.js
└── views
    ├── detail.ejs
    ├── edit.ejs
    ├── list.ejs
    ├── nav.ejs
    └── write.ejs
```

index.js

The main entry point of the application. It sets up the Express server, middleware, and routes.

Views

The views directory contains EJS templates for rendering HTML pages.

Util

The util directory contains utility modules for database interactions and other helper functions.

public

The public directory contains static assets like images, CSS, and JavaScript files.

This is how to serve static files in Express:

```
app.use('/public', express.static('public'));
```

- There is no functions to use the public directory in this prototype.
- This is just to show how to serve static files in Express.

REST API Endpoints (index.js)

Create

- `POST /add` - Create a new post (runs `runAddPost`) then redirect to `/`.

Read

- `GET /` - Render the post-writing page (`write.ejs`).
- `GET /list` - Render the posts list (runs `runListGet`).
- `GET /detail/:id` - Retrieve a specific post by ID, render `detail.ejs`.
- `GET /listjson` - Return all posts as JSON (`posts.find().toArray()`).

Update

- `GET /edit/:id` - Retrieve a specific post by ID for editing, render `edit.ejs`.
- `PUT /edit` - Update a post by `id` from the request body, then redirect to `/list`.

Delete

- `DELETE /delete` - Delete a post using `_id` from the request body.

Testing Module

tests

The tests directory contains test files for the application.

- db.simple.test.js: Basic tests for unit tests operation.
- db.test.js: Basic tests for database operations.

Much More Testing Needed

This is way too few tests for a real-world application.

- No unit tests for individual modules.
- No integration tests for multiple modules working together.
- No end-to-end tests (E2E) for the entire application flow.
- No acceptance tests to ensure the application meets requirements.
- No regression tests to catch bugs after changes.

No Documentation

There is no good documentation for the application.

1. Just one simple README file is not enough.
2. No API documentation for REST endpoints.
3. No Data Model documentation.
4. No architecture documentation for explaing how the Data Model flows.
5. No design documentation for explaining design decisions on each module.
6. No inline code comments for complex logic.
7. No user documentation (manual) for how to use the app.

Why this is Low-Quality Software Product?

1. Data Model is too simple and stored in a random directory.
2. Error handling and validation are minimal.
3. Too little testing and documentation.
4. No authentication or authorization.
5. No logging or monitoring.

No easy or effective way to maintain or extend the application!