Assignment: Node.js & Mongoose Micro-Apps

Deadline: 6 Days from today

Objective

The goal of this assignment is to break the **fear of backend development** and build strong **muscle memory** using Node.js and Mongoose. You will build **5 small practical tasks** that simulate real backend scenarios.

Note: Feeling confused at the beginning is **completely normal** repetition is the key to confidence.

Prerequisites (Required for Every Task)

For each task, make sure that you:

1. Connect to MongoDB using `mongoose.connect`.
2. Create the required **Schema & Model**.
3. Create the required **Routes** (POST & GET).
4. Test your APIs using **Postman** .

Task 1: To-Do List

Concept: Using **default values** in Mongoose schemas.

- **Step 1: Create Model (Task)**
  - `title`: String, **Required**
  - `isCompleted`: Boolean, **Default: false** (Do NOT require this field from the user).
- **Step 2: Create Routes**
  - `POST /api/tasks` → Create a new task.
  - `GET /api/tasks` → Get all tasks.

Test Data (Postman):

- **Body (JSON):** `{"title": "Study Node.js"}`
- **Expected Result:** The response should automatically include `"isCompleted": false`.

Task 2: Contact Manager (Nested Data)

Concept: Working with **arrays** and **nested objects**.

- **Step 1: Create Model (Contact)**
  - `fullName`: String, **Required**
  - `phones`: Array of Strings (`[String]`).
  - `socialMedia`: Object containing `Facebook` (String) and `linkedin` (String).
- **Step 2: Create Routes**
  - `POST /api/contacts` → Create a contact.
  - `GET /api/contacts` → Retrieve contacts.

Test Data (Postman):

```
JSON
{
 "fullName": "John Doe",
 "phones": ["01012345678", "01298765432"],
 "socialMedia": {
   "twitter": "@johndoe",
   "linkedin": "linkedin.com/in/johndoe"
 }
}
```

Task 3: Library System (Relationships) (*Important)*

Concept: Creating relationships using `ObjectId` and `ref`.

- **Step 1: Create Author Model** (`name`: String).
- **Step 2: Create Book Model** (`title`: String, `author`: Type: `ObjectId`, Ref: `'Author'`).
- **Step 3: Create Routes**
  - `POST /api/authors`
  - `POST /api/books`

- `GET /api/books` → **Use `.populate('author')`** to show author details.

Test Flow:

1. **Create Author:** `{ "name": "J.K. Rowling" }` ➞ Copy the returned `_id`.
2. **Create Book:** Use the copied ID: `{"title": "Harry Potter", "author": "PASTE_ID_HERE"}`.
3. **Get Books:** Ensure the **author name** appears instead of just the ID.

Task 4: Product Inventory (Filtering)

Concept: Filtering results using `req.query`.

- **Step 1: Create Model (Product)**
  - `name`: String, `category`: String, `price`: Number.
- **Step 2: Create Routes**
  - `POST /api/products`
  - `GET /api/products` → If URL contains `?category=Electronics`, return only electronics.
  - **Hint:** Use `Product.find(req.query)`.

Test Data:

- **URL:** http://localhost:3000/api/products?category=Electronics
- **Expected Result:** Only products matching the category should appear.

Task 5: Classroom (One-to-Many)

Concept: One-to-Many relationship using an **array of ObjectIds**.

- **Step 1: Create Student Model** (`name`: String).
- **Step 2: Create Classroom Model** (`name`: String, `students`: Array of `ObjectIds` referencing `'Student'`).
- **Step 3: Create Routes** (`POST /api/students`, `POST /api/classrooms`, `GET /api/classrooms` with `populate`).

Test Data:

- **POST /api/classrooms Body:**

```
JSON
{
 "name": "Node.js Class",
 "students": ["PASTE_STUDENT_ID_1", "PASTE_STUDENT_ID_2"]
}
```

Bonus Tasks (Optional)

- Ensure email fields are **unique** and return a clear error message.
- Add a **DELETE** route for any model.
- Improve API responses to follow this format: `{"success": true, "count": X, "data": [...]}`.
- Use `populate` with **selected fields only** (e.g., name only).

Submission Instructions

1. Push your code to **GitHub**.
2. Submit the repository link before the deadline.
3. **Code clarity and structure matter more than perfection.**

Don't be afraid to fail, try a lot!

Programming rewards those with patience. Enjoy the journey! 👀😄