Node Developer Machine Test

Duration: 3 hours

Design and implement a robust RESTful API for a **Task Management System** using Node.js and Express.js. This test assesses your proficiency in core Node.js concepts, Express.js framework, API design, and best practices.-----

RequirementsTech Stack (Mandatory)

- **Backend:** Node.js (v16 or higher), Express.js (v4.x)
- **Database (Choose One):**
    - MongoDB with Mongoose
    - PostgreSQL/MySQL with an ORM (Sequelize, TypeORM, etc.) - TypeORM recommended
    - SQLite or MySQL
- **Package Manager:** npm or yarn

Optional (Bonus Points)

- TypeScript
- JWT for authentication
- Input validation library (Joi, express-validator, etc.)

-----**Expected Project Structure**

```
task-management-api/
├── src/
│   ├── controllers/
│   ├── models/
│   ├── routes/
│   ├── middlewares/
│   ├── utils/
│   └── app.js
├── config/
├── tests/ (optional)
├── .env.example
├── .gitignore
├── package.json
└── README.md
```

Features to Implement

1. User Management

Endpoint	Method	Route	Description
Registration	**POST**	`/api/auth/register`	Create a new user.
Login	**POST**	`/api/auth/login`	Authenticate and return **JWT token/session**.

Feature	Details
Registration Fields	`name` (Required), `email` (Unique, Valid format), `password` (Min 6 characters)
Password Storage	**Must be hashed** before storing.
Login Fields	`email`, `password`
Response	Success response should **exclude the password**.

2. Task Management (50 points)

Endpoint	Method	Route	Authentication
Create Task	**POST**	`/api/tasks`	**Required**
Get All Tasks	**GET**	`/api/tasks`	**Required**
Get Single Task	**GET**	`/api/tasks/:…`	**Required**
Update Task	**PUT**	`/api/tasks/:…`	**Required**
Delete Task	**DELETE**	`/api/tasks/:…`	**Required**

Feature	Details
Task Fields	`title` (Required, Max 100), `description`, `status`, `priority`, `dueDate`
`status` Values	`pending` (Default), `in-progress`,

	`completed`
`priority` Values	`low`, `medium` (Default), `high`
`dueDate` Validation	Must be a **valid future date**.
Ownership	Tasks must be associated with the logged-in user. Update/Delete must validate **ownership** (403 Forbidden if not owned).
Get All Tasks	Must support **Filtering** (`status`, `priority`), **Pagination** (`page`, `limit`), and **Sorting** (`sortBy`, `order`).
Get Single Task	Return **404 Not Found** if ID is invalid/non-existent.

3. Task Statistics

Endpoint	Method	Route	Description
Summary	**GET**	`/api/tasks/stats/summary`	Return task counts by status and priority for the logged-in user.

Example Response:

```
{
  "total": 25,
  "pending": 10,
  "in-progress": 8,
  "completed": 7,
  "overdue": 3,
  "byPriority": {
    "low": 5,
    "medium": 12,
    "high": 8
  }
}
```

4. Middleware & Error Handling

- **Authentication Middleware:** Protect all required routes. Return **401 Unauthorized** for invalid/missing tokens.

- **Error Handling Middleware:** Implement a **global error handler** with proper HTTP status codes.
- **Input Validation:** Validate all request body/params. Return clear validation error messages (e.g., **400 Bad Request**).
- **Request Logging:** Log all incoming requests (method, path, timestamp). Use Morgan or custom middleware.

Database SchemaUser Model

```
{
  id: String/Number (Primary Key),
  name: String (required),
  email: String (required, unique),
  password: String (required, hashed),
  createdAt: Date,
  updatedAt: Date
}
```

Task Model

```
{
  id: String/Number (Primary Key),
  title: String (required),
  description: String,
  status: Enum ['pending', 'in-progress', 'completed'],
  priority: Enum ['low', 'medium', 'high'],
  dueDate: Date,
  userId: Reference to User, // Association
  createdAt: Date,
  updatedAt: Date
}
```

API Response Formats

Type	HTTP Status Codes	Format
Success	200, 201, etc.	`{"success": true, "data": { /* response data */ }, "message": "Optional success message"}`

Error	400, 401, 403, 404, 500	`{"success": false, "error": "Error message", "statusCode": 400}`

Important HTTP Status Codes:

- **200:** Success
- **201:** Created
- **400:** Bad Request (Validation Errors)
- **401:** Unauthorized
- **403:** Forbidden (Ownership Issues)
- **404:** Not Found
- **500:** Server Error

-----**Environment Variables**

Create a `.env.example` file with the following:
NODE_ENV=development
PORT=3000
DATABASE_URL=your_database_url
JWT_SECRET=your_jwt_secret
JWT_EXPIRE=7d

Evaluation Criteria

Category	Points	Details
Mandatory	**70 pts**	All API endpoints functional (40), Proper authentication (10), Input validation (10), Robust error handling (10).
Code Quality	**20 pts**	Clean structure (5), Proper async/await usage (5), RESTful principles (5), Code readability and comments (5).
Bonus	**10 pts**	Unit tests (4), TypeScript (3), Docker setup (2), API documentation (1).

Submission Checklist

- **Code:** Pushed to a GitHub repository (public/private with access). Clear commit history. `.gitignore` set up.
- **`README.md` (Mandatory):** Must include project description, prerequisites, installation steps, how to run, API endpoints, environment variables setup, and any assumptions made.
- **Database:** Include schema/migrations. Seed data is recommended.
- **Testing:** Provide a Postman collection or cURL commands with sample request/response examples.

Time Management Suggestions

Component	Suggested Time
Project Setup/Structure	30 mins
User Auth & Model Setup	45 mins
Task CRUD Operations	60 mins
Filtering, Pagination, Stats	20 mins
Error Handling & Validation	15 mins
Testing & Documentation	10 mins
Total	**3 hours**

Tips for Candidates

1. **Prioritize Functionality** before optimization.
2. **Use `async/await`** over callbacks.
3. **Validate All Inputs** on the server side.
4. **DO NOT commit sensitive data** (like `.env` files).
5. **Write Clean, Readable Code** with proper naming conventions.
6. **Handle Edge Cases** (empty results, invalid IDs, etc.).

–

NOTE

(Remember: Quality over quantity. A well-implemented subset of features is better than a poorly implemented everything.)

Good Luck! 🚀