📂 Revamped Internship Project — TaskSphere API (Project ID: REVA2025/86)

Role: Full Stack Developer (Backend)
Duration: 25 Days (~1.5 hours/day)
Goal: Production-ready Task Management REST API with authentication, task CRUD, file uploads, comments, analytics, and deployment.

Note: No need to do frontend, we need only backend as per the given requirements!

Day-by-Day Plan:

Core Entities

1. User Schema

```
{
name: String,
email: String (unique),
password: String (hashed),
createdAt: Date
}
```

2. Task Schema

```
{
  title: String,
  description: String,
  status: 'pending' | 'in-progress' | 'completed',
  dueDate: Date,
  assignedTo: ObjectId (User),
  createdAt: Date,
  attachments: [String], // file paths
  comments: [
    {
      text: String,
      postedBy: ObjectId (User),
      createdAt: Date
    }
  ]
}
```

Day 1 — Project Kick-off

- Setup Node.js, Express, MongoDB connection.
- Project folder structure + .env configuration.
- Create /api/health endpoint with uptime + timestamp.
- Implement centralized logger middleware (log every request to console).

Day 2 — User Module Setup

- Create User schema with validations (name, email unique, strong password).
- Implement **User Registration**:
    - Hash password (bcrypt) before saving.
    - Validate required fields.
    - Return new user (excluding password).
- Implement **Get All Users** (GET /api/users) for admin only.

Day 3 — Authentication Module

- Implement **Login API**:
    - Verify email/password.
    - Return JWT token + user details.
- Middleware auth.js:
    - Validate token, attach req.user.
- Protected route: GET /api/users/me → return logged-in user’s data.

Day 4 — Task Module (Creation + Fetching)

- Create Task schema:
    - title, description, status, dueDate, assignedTo.
    - Default status: 'pending'.
- Implement **Create Task** (POST /api/tasks):
    - Only logged-in users.
- Implement **Get All Tasks** (GET /api/tasks):
    - Populate assignedTo.
    - Filter by status and dueDate.
    - Search by title (regex).
    - Pagination (page, limit).

Day 5 — Task Details + Modification

- **Get Task by ID** (GET /api/tasks/:id) → return populated user info.
- **Update Task** (PUT /api/tasks/:id) → only creator or assigned user can update.
- **Delete Task** (DELETE /api/tasks/:id) → only creator can delete.
- Add error handling for invalid IDs.

Day 6 — Task Ownership & Security

- Middleware isOwnerOrAssigned:
    - Restrict update/delete to assigned user or creator.
- Auto-assign createdBy field when task is created.
- Create GET /api/my-tasks → tasks assigned to logged-in user.

Day 7 — File Uploads

- Install multer.
- Add attachments array in Task schema.
- API: POST /api/tasks/:id/upload → store file path in task.

- Restrict max file size to 2MB.
- Accept only .jpg, .png, .pdf.

Day 8 — Comments Feature

- Add comments array in Task schema:
- comments: [
- {
-    text: String,
-    postedBy: ObjectId(User),
-    createdAt: Date
- }
-]
- API: Add comment → POST /api/tasks/:id/comments.
- API: Delete comment → DELETE /api/tasks/:taskId/comments/:commentId.

Day 9 — Status & Workflow

- API: PATCH /api/tasks/:id/complete → change status to completed.
- API: PATCH /api/tasks/:id/in-progress → change status.
- Restrict transitions:
  - pending → in-progress → completed
  - completed cannot go back.

Day 10 — User Roles & Permissions

- Add role field in User schema → 'user' or 'admin'.
- Middleware isAdmin.
- Admin APIs:
  - Delete any task.
  - Get all users with their task counts.

Day 11 — Dashboard Analytics

- API: GET /api/dashboard/stats:
  - Total tasks.
  - Tasks by status.
  - Tasks due today.
  - Users with most assigned tasks.

Day 12 — Notifications Simulation

- On new task creation → store a “notification” in notifications collection.
- API: GET /api/notifications → fetch user notifications.
- Mark notification as read.

Day 13 — Activity Logs

- Middleware: log every task creation/update/delete into activityLogs collection.
- API: GET /api/logs → admin only.

Day 14 — Advanced Query Optimization

- Add MongoDB indexes:
  - status index.
  - dueDate index.
- Optimize GET /api/tasks query for performance.

Day 15 — Bulk Task Operations

- API: POST /api/tasks/bulk-create → create multiple tasks at once.
- API: DELETE /api/tasks/bulk-delete → delete multiple by IDs.

Day 16 — User Profile API

- API: PUT /api/users/update-profile:
  - Update name/email/password.
  - Password change requires old password verification.
- API: POST /api/users/upload-avatar (multer).

Day 17 — Password Reset (Email Simulation)

- API: POST /api/users/forgot-password → generate reset token.
- API: POST /api/users/reset-password → validate token & update password.
- Store reset token in DB (expires in 15 min).

Day 18 — Rate Limiting & Security

- Install express-rate-limit.
- Limit login attempts to 5 per 15 min.
- Install helmet for HTTP headers security.

Day 19 — API Documentation

- Install swagger-ui-express.
- Document all routes.
- Publish /api/docs.

Day 20 — Unit & Integration Testing (You can also use Postman for testing)

- Install jest + supertest.
- Write tests for:
  - Registration
  - Login
  - Task creation
  - Task update

Day 21 — Error Handling & Response Standards

- Create global error handler middleware.
- All errors return:

```
{
 "success": false,
 "message": "Error message here",
 "code": 400
}
```

Day 22 — Environment Configs

- Setup .env for:
    - MONGO_URI
    - JWT_SECRET
    - PORT
- Create config.js to read env variables.

Day 23 — Deployment Preparation

- Optimize MongoDB indexes.
- Setup CORS for frontend.
- Prepare production build with pm2.

Day 24 — Cloud Deployment

- Deploy on Render/Railway.
- Test APIs in production environment.

Day 25 — Final Handover

- Push code to GitHub.
- Deliver:
    - Swagger docs link
    - Deployed API URL

Tools and Technologies:

- Backend: Node.js (Express)/Python (Flask/Django)/Java (Spring Boot)

- Database: PostgreSQL/MySQL/MongoDB