MEDIVUE BACKEND ASSESSMENT: TASK MANAGEMENT API

Context

Build a robust **Task Management API** that supports advanced filtering, tagging, and deadlines. This assessment evaluates your ability to design clean schemas, implement efficient filtering, and provide clear documentation.

Timeline

- **Expectation:** 4-6 hours from getting this document.

Submission

1. Push your solution to a public GitHub repository
2. Send the link of the repository by replying to this email

Technical Requirements

- **Language:** Python 3.10+
- **Framework:** FastAPI
- **Database:** PostgreSQL (preferred) or SQLite (if using SQLite, explain why in README).
- **Containerization:** Must include a Dockerfile and docker-compose.yml to run the app and the DB with a single command.

Expected API Endpoints

1. POST /tasks

Create a new task.

- **Input (JSON):**
  - title: String (Required, non-empty, max 200 chars).
  - description: String (Optional).
  - priority: Integer (1–5, where 5 is highest).
  - due_date: String (ISO format YYYY-MM-DD).
  - tags: List of strings (Optional).
- **Validation:** Ensure due_date is not in the past and priority is within range.

2. GET /tasks

Retrieve a list of tasks with support for pagination and filtering.

- **Query Parameters:**
  - completed: Boolean toggle.
  - priority: Filter by specific level.
  - tags: CSV format (e.g., ?tags=work,urgent) to match tasks containing **any** of these tags.
  - limit / offset: For pagination.
- **Response:** A paginated object including total count and the list of tasks.

3. GET/PATCH/DELETE /tasks/{id}

- **GET:** Return the task object or 404 Not Found.
- **PATCH:** Support **partial updates**. Only fields provided in the request body should be modified.
- **DELETE:** Implement either a **Soft Delete** (flagging) or **Hard Delete**.
    - *Note: Justify your choice in the README.*

Data Modelling & Performance

- **Schema Design:** Design an efficient way to handle the many-to-many relationship between **Tasks** and **Tags**.
- **Indexing:** Apply database indexes to fields frequently used in filtering (e.g., priority, completed).
- **Explanation:** In your README, briefly explain the trade-offs of your chosen tagging implementation (e.g., Join Table vs. PostgreSQL JSONB or ARRAY types).

Non-Functional Requirements

- **Validation:** Use **Pydantic** models for request/response serialisation and validation.

- **Error Handling:** Return consistent, structured error messages.

  Example Error Shape:

  ```
  {
  "error": "Validation Failed",
  "details": { "priority": "Must be between 1 and 5" }
  }
  ```

- **Testing:** Include a test suite (using pytest and httpx) covering:
    - Successful task creation vs. validation failures.
    - Filter logic for tags and priority.
    - Edge cases for partial updates (PATCH).

- **Documentation:** * The API should be fully documented via FastAPI's built-in /docs (Swagger).
    - A README.md covering setup instructions, design decisions, and "Production Readiness" improvements.