Assignment 1: Image Upload and Description Generation API

Objective:
Develop an API that allows users to upload an image, process it using a GPT or any LLM (Large Language Model), and return various descriptions of the image in different tones and styles.

Requirements:

1. **Image Upload**:
    - The API should accept an image file via a POST request.
    - Perform basic validations on the image (e.g., file size, type, resolution).
2. **Integration with GPT/LLM**:
    - Once the image is uploaded, process the image using an external service or library (e.g., a pre-trained model or external API) to extract information or analyze the content.
    - Use this information to construct prompts for a GPT/LLM API (e.g., OpenAI API or equivalent).
3. **Response Generation**:
    - The GPT/LLM should generate multiple descriptions of the image:
        - A decent and formal description.
        - A funny and humorous description.
        - A critical or satirical description.
    - Each description should be part of the API's JSON response.
4. **API Response**:
    - Return a JSON response containing:
        - The uploaded image URL (or path).
        - An array of descriptions categorized by type (e.g., "formal", "humorous", "critical").

```
{
    "image_url": "http://example.com/uploads/image.jpg",
    "descriptions": {
        "formal": "This is a serene landscape with lush greenery and a vibr
        "humorous": "Looks like Mother Nature decided to flex on Instagram!
        "critical": "The framing of the image could use some work; the hori
    }
}
```

Assignment 2: Advanced API Development with Authentication

Objective:
Create an API that includes robust authentication and demonstrates advanced Django development skills.

Requirements:

1. **API Functionality:**
    - Build an API endpoint for managing user-created "Tasks" (or another domain object of your choice).
    - The tasks should include fields such as `title`, `description`, `due_date`, and `status`.
2. **Features**:
    - CRUD Operations:
        - Implement endpoints for creating, retrieving, updating, and deleting tasks.
    - Filtering and Pagination:
        - Allow users to filter tasks based on their status (e.g., "completed", "pending").
        - Implement pagination for task lists.
3. **Authentication and Authorization:**
    - Use Django REST Framework's Token or JWT authentication.
    - Ensure that only authenticated users can access the API.
    - Ensure that users can only access or modify their own tasks.
4. **Security:**
    - Use Django's built-in CSRF protection.
    - Sanitize all inputs to prevent SQL injection or other attacks.
5. **Additional Challenge:**
    - Implement a background job (using Celery or Django Q) that sends a reminder email to the user 24 hours before the due date of a task.
    - Ensure this job is scheduled automatically when a task is created with a valid `due_date`.
6. **API Response:**
    - Return structured JSON responses with appropriate HTTP status codes for each operation.
    - Example response for a GET request:

```
{
    "tasks": [
        {
            "id": 1,
            "title": "Complete Django assignment",
            "description": "Finish the API development task by tomorrow.",
            "due_date": "2024-11-26T15:30:00Z",
            "status": "pending"
        },
        {
            "id": 2,
            "title": "Prepare for code review",
            "description": "Get the codebase ready for submission.",
            "due_date": "2024-11-27T12:00:00Z",
            "status": "completed"
        }
    ],
    "count": 2,
    "next": null,
    "previous": null
}
```

Evaluation Criteria:

- **Code Quality**: Adherence to Django best practices, modularity, and clean code.
- **API Design**: Proper structuring of endpoints, appropriate use of HTTP methods, and meaningful responses.
- **Database Integration**: Efficient and seamless integration with MongoDB, demonstrating a clear understanding of NoSQL principles.
- **Error Handling**: Ability to handle edge cases and return appropriate error messages.
- **Authentication**: Secure implementation of authentication and authorization.
- **Documentation**: Clear and concise README with precise setup and testing instructions.