NestJS Assignments

Assignment 1: DTO Class-Validator Validations (Custom Validators)

Objective: Implement a custom validation decorator using class-validator in NestJS.

Task:

1. Create a DTO class with fields like name, email, age, and phoneNumber.
2. Implement a custom validation decorator to validate that:
    - The phoneNumber follows a specific pattern (e.g., a 10-digit number for a specific country).
    - The age must be between 18 and 60.
3. Use class-validator and class-transformer for validation.
4. Create a simple controller to test the DTO validations.

Deliverables:

- DTO file with validation rules and custom decorator.
- A sample NestJS controller handling the validation.
- A Postman collection or example API requests for testing validation.

Assignment 2: API Versioning (With Swagger)

Objective: Implement API versioning in a NestJS project and document it using Swagger.

Task:

1. Create two versions (v1 and v2) of a simple UserController API.
2. Implement versioning using one of the available strategies (URI, Header, or Accept-Version).
3. Use Swagger to document both API versions separately. Set up Swagger by installing @nestjs/swagger, configuring the SwaggerModule, and adding API tags for version differentiation.
4. Each version should have different implementations:
    - v1 should return a basic user response.
    - v2 should return an extended response with additional user details.

Deliverables:

- Working API with versioning implemented.
- Swagger documentation that shows both versions.
- Example API calls for both versions.

Assignment 3: File Upload Using Multer (With Validation)

Objective: Implement file upload functionality in NestJS using multer with validation.

Task:

1. Set up a file upload endpoint in a NestJS controller.
2. Restrict uploads to only images (JPEG, PNG) with a maximum size of 2MB.
3. Save uploaded files to a designated folder (uploads/).
4. Provide appropriate error messages for invalid file types or exceeding file size limits.

Deliverables:

- NestJS service and controller handling file uploads.
- Multer configuration with file type and size validation.
- Sample API requests demonstrating successful and failed uploads.

Assignment 4: Multi-Language Messages Using i18n

Objective: Implement internationalization (i18n) in a NestJS application to support multiple languages.

Task:

1. Set up nestjs-i18n and configure translations for at least two languages (e.g., English and Spanish).
2. Store translation files in src/i18n/ as JSON files (e.g., en.json, es.json).
3. Create an endpoint that returns a localized greeting message based on the request's language.
4. Implement a way to switch languages dynamically (using headers or query parameters).

Deliverables:

- NestJS setup with nestjs-i18n configured.i
- Translation files with sample messages.
- An API endpoint that returns messages in the requested language.

Notes:

- Ensure each assignment is well-structured and follows NestJS best practices.
- Provide test cases or sample API requests to validate the implementations.
- Use appropriate error handling mechanisms in all assignments.