### **Revised Practical Task (60 Marks)**

You are asked to create a **Web API** for managing books in a library. The API should allow for CRUD operations on **Book, Author**, and **Genre** entities, implementing a many-to-many relationship between them.

**Clone:**  
[ClickHere](https://github.com/daliaMoHamed21/Quiz-2) <https://github.com/daliaMoHamed21/Quiz-2>

to clone the project that you work on.

### **Models:**

1. **Book**:

* + Id: Integer (Auto-incremented)
  + Title: String (Required)
  + PublishedYear:Date
  + Authors: Collection of Author (Many-to-Many)
  + Genres: Collection of Genre (Many-to-Many)

2. **Author**:

* + Id: Integer (Auto-incremented)
  + Name: String (Required)
  + PhoneNumber: (Validate :Phone number)
  + Email-Address : (Validate :Email Address)
  + Books: Collection of Book (Many-to-Many)

3. **Genre**:

* + Id: Integer (Auto-incremented)
  + Name: String (Required)
  + Books: Collection of Book (Many-to-Many)

### **Requirements:**

* + **Create the Books/Author/Genre Controller**:
  + The controller must manage CRUD operations for the entities.
  + Use **Dependency Injection** to interact with a **Repository** that manages data.
  + Authorize **Author** when using **GET** method using **JWT**

1. **Implement the following actions** in BooksController **Using** RepositoryPattern:

o **GET** /api/books:

* + - Retrieves all books with **Author** & **Genre**.

o **GET** /api/books/{id}:

* + - Retrieves a book by ID.
    - If the book is not found, return a **404 Not Found** status.
    - Include related Authors and Genres in the response.

o **POST** /api/books:

* + - Adds a new book to the collection.
    - Validate that Title and PublishedYear meet the constraints.
    - Allow for adding existing Authors and Genres by their IDs.
    - Return a **400 Bad Request** if validation fails.
    - Return **200 Created** with the created book details in the response body.

o **PUT** /api/books/{id}:

* + - Updates an existing book by ID.
    - Validate that the book exists before updating.
    - Allow for updating Authors and Genres.
    - Return **400 Bad Requests** for validation errors.
    - Return **200** on successful update.

o **DELETE** /api/books/{id}:

* + - Deletes a book by ID.
    - If the book does not exist, return a **404 Not Found**.

2. **Create AuthorsController** Using **RepositoryPattern**:

* + Implement the same CRUD operations for the Author entity.
  + Implement the following actions:
    - **GET** /api/authors: Retrieves all authors.
    - **GET** /api/authors/{id}: Retrieves an author by ID.
    - **POST** /api/authors: Adds a new author.
    - **PUT** /api/authors/{id}: Updates an existing author by ID.
    - **DELETE** /api/authors/{id}: Deletes an author by ID.

**3. Create GenresController**:

* + Implement the same CRUD operations for the Genre entity.
  + Implement the following actions:
    - **GET** /api/genres: Retrieves all genres.
    - **GET** /api/genres/{id}: Retrieves a genre by ID.
    - **POST** /api/genres: Adds a new genre.
    - **PUT** /api/genres/{id}: Updates an existing genre by ID.
    - **DELETE** /api/genres/{id}: Deletes a genre by ID.

4. **Model Validation**:

* + Ensure that:
    - Title and PublishedYear are required fields for the **POST** and **PUT** methods in BooksController.
    - Name is required for Author and Genre.
    - PhoneNumber for Author.
    - Email-Address for Author.
    - Return proper validation error messages in case of failure.

5. **HTTP Status Codes**:

* + Return appropriate status codes based on the result of each operation:
    - **200 OK** for successful retrieval.
    - **404 Not Found** if an entity is not found.
    - **400 Bad Requests** for validation errors.

**Note**:

* Implement **Dependency Injection** to handle the repository for books, so the controller does not directly interact with the data source.
* Use **Postman** to document and test your API.

**Marking Scheme:**

* **Multiple Choice**: 15 questions × 1 mark = 15 marks
* **Short Answer**: 5 questions × 5 marks = 25 marks
* **Practical Task**:
  + Correct controller and method implementation: 10 marks
  + Correct model validation and error handling: 15 marks
  + Use of HTTP status codes: 10 marks
  + Proper routing: 10 marks
  + (Dependency Injection and Swagger): 15 marks

**Total Marks: 100 Marks**