Create a Web Application where you specify and implement:

- Three classes :

```
public class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime DateOfBirth { get; set; }
}
public class User
{
    public string Username { get; set; }
    public string Email { get; set; }
    public int Age { get; set; }
}
public class Product
{
    public string Title { get; set; }
    public string Description { get; set; }
    public decimal Price { get; set; }
}
```

- Three collections of each class

```
// Collection of Person objects
List<Person> people = new List<Person>
{
    new Person { Id = 1, Name = "Alice", DateOfBirth = new DateTime(1990, 5, 15) },
    new Person { Id = 2, Name = "Bob", DateOfBirth = new DateTime(1985, 10, 25) },
    new Person { Id = 3, Name = "Charlie", DateOfBirth = new DateTime(1988, 3, 8) }
};

// Collection of User objects
List<User> users = new List<User>
{
    new User { Username = "user1", Email = "user1@gmail.com", Age = 30 },
    new User { Username = "user2", Email = "user2@ul.edu.lb", Age = 25 },
    new User { Username = "user3", Email = "user3@gmail.com", Age = 35 }
};

// Collection of Product objects
List<Product> products = new List<Product>
{
    new Product { Title = "Product 1", Description = "Description of Product 1",
Price = 99.99m },
    new Product { Title = "Product 2", Description = "Description of Product 2",
Price = 149.99m },
    new Product { Title = "Product 3", Description = "Description of Product 3",
Price = 199.99m }
};
```

1. GET Endpoint for Simple Type
    a. Implement a GET endpoint **/api/random** that returns a random integer between 1 and 100 as a simple type.
    b. Modify the endpoint to accept an optional query parameter count indicating the number of random integers to return in an array.

2. **GET Endpoint for Simple Type**
    a. Implement a GET endpoint **/api/square** that accepts an integer as input and returns its square as a simple type.
    b. Enhance the endpoint to accept a list of integers and return the squares of all the numbers in the list.
3. **POST Endpoint for Simple Type**
    a. Implement a POST endpoint **/api/square** that accepts an integer as input from body and returns its square as a simple type.
    b. Enhance the endpoint to accept a list of integers from body and return the squares of all the numbers in the list.
4. **POST Endpoint for Person**
    a. Implement a POST endpoint **/api/person** that accepts data of type Person and logs it to the console and save it in the collection.
    b. Enhance the endpoint to validate the input data and return appropriate error responses for invalid requests (Id should be between 1 and 1000, Name should start by a letter).
5. **POST Endpoint for Persons**
    a. Implement a POST endpoint **/api/persons** that accepts many objects of type Person and save them in the collection.
    b. Enhance the endpoint to validate the input data and return appropriate error responses for invalid requests (Id should be between 1 and 1000, Name should start by a letter).
6. **GET Endpoint Person by Id**
    a. Implement a GET endpoint **/api/person/{id}** to retrieve a person by its Id from the collection. Use route parameter for the Id.
    b. Enhance the endpoint in order to handle cases where the Id doesn't exist or the Id is not between 1 and 1000.
7. **GET Endpoint for User**
    a. Implement a GET endpoint **/api/user** that returns dummy data of type User with hardcoded values.
    b. Enhance the endpoint to return filtered data from the collection based on query parameters such as age and/or username. If we do not specify any parameter, then we should return all users.
8. **POST Endpoint a new User**
    a. Implement a POST endpoint **/api/user** to add a new user to the collection. Use request body binding to bind the user details.

b. Validate the input data using DataAnnotations, ensuring all required fields are provided and age is between 1 and 100, and the username length does not exceed 20 characters.

9. **GET Endpoint Users older than a certain age**
    a. Create a GET endpoint **/api/users** to fetch users from the collection older than a specified age. Utilize query parameters to specify the age threshold.
    b. Ensure that the age is between 1 and 100.
10. **Get Endpoint Users by Username**
    a. Implement a GET endpoint **/api/users/{username}** to retrieve a user by their username. Utilize route parameter for the username.
    b. Ensure to handle cases where the username doesn't exist and its length more than 20 characters (not valid).
11. **Post Endpoint Update User Email Address**
    a. Develop a POST endpoint **/api/user/{username}** to update a user's email address. Utilize route parameter for the username and request body binding for the new email address.
    b. Validate the email address format using DataAnnotations.
12. **Post Endpoint add new Product**
    a. Implement a POST endpoint **/api/product** to add a new product to the collection. Use request body binding to bind the product details.
    b. Validate the input data using DataAnnotations, ensuring all required fields are provided.
13. **Get Endpoint Products within a price range**
    a. Develop a GET endpoint **/api/products** to retrieve products within a specified price range. Use query parameters to specify the minimum and maximum prices.
    b. Ensure the input data is correctly parsed and validated (min < max, both min and max should be positive).
14. **Get Endpoint Products with a specific keyword in the title or description:**
    a. Implement a GET endpoint **/api/products/{title}** to fetch products from the collection containing a specific keyword in either the title or description. Use route and query parameters for the keywords.
    b. Ensure that the title and the description are not empty strings.
15. **Post Endpoint Add new Product**
    a. Design a POST endpoint **/api/product/{title}/{description}** to add a new product to the collection. Get the price of the product from query parameter. Implement a custom parsing mechanism using TryParse to convert the received parameter price into the appropriate data type. The handler should save the object in the collection after setting the title and description.

b. If no query parameter is passed, the price is set to zero.

16. GET Endpoint Retrieve Person Information

a. Design a GET endpoint /api/person/{id}/{name} to retrieve person information, utilizing **AsParameters** for the handler parameter of type Person. Return the retrieved person information as a JSON object in the response.

b. Enhance the endpoint in order to validate the id (between 1 and 1000).

17. User Information with IValidatableObject

a. Perform business validation using the IValidatableObject interface on the User class. Business validations are: Age should be multiple of 2, Email should be only from the following domains: gmail.com, ul.edu.lb.

18. GET Endpoint Retrieve Person Information with Async Binding

a. Develop a GET endpoint **/api/person/** to retrieve user information using BindAsync. The endpoint should utilize both route parameters and query string parameters to identify the user and fetch their information. Conclude!!