Сучасні інтернет технології. Лабораторне заняття №6
МОДЕЛІ ВІДОБРАЖЕННЯ. ВАЛІДАЦІЯ ДАНИХ У ЗАСТОСУНКУ ASP.NET CORE

Виконала: студентка МІТ-41, Панченко Владислава

Завдання для виконання (max - 100 балів)
1. Ознайомитися з теоретичними основами валідації (сайт Microsoft). (max - 10 балів)
Валідація — це процес перевірки того, чи відповідають вхідні дані (наприклад, введені користувачем у форму, отримані з API або файлу) певним правилам та обмеженням.

Головна мета валідації — це забезпечення:

1. **Цілісності даних:** Гарантія того, що в систему потрапляють лише дані, які мають сенс і відповідають очікуваному формату та діапазону.
2. **Безпеки:** Запобігання вразливостям, таким як **SQL-ін'єкції** або **міжсайтовий скриптинг (XSS)**, шляхом очищення та перевірки вхідних даних.
3. **Користувацького досвіду (UX):** Надання користувачам миттєвого та зрозумілого зворотного зв'язку про помилки, щоб вони могли їх виправити, перш ніж дані будуть відправлені на сервер.

Валідація зазвичай відбувається на кількох рівнях:

Валідація на Клієнті (Client-Side Validation)

- **Де відбувається:** У браузері користувача (за допомогою **HTML5-атрибутів** або **JavaScript**).
- **Переваги: Миттєвий зворотний зв'язок**, швидкий користувацький досвід, зниження навантаження на сервер.
- **Недоліки: Ненадійна** для безпеки. Зловмисник може обійти клієнтську валідацію, вимкнувши JavaScript або використовуючи інструменти.
- **Висновок:** Використовується для UX, але **ніколи не повинна бути єдиною перевіркою**.

Валідація на Сервері (Server-Side Validation)

- **Де відбувається:** На бекенді (наприклад, у контролерах або бізнес-логіці програми на .NET, Java, Python тощо).
- **Переваги: Надійна** та **критично важлива для безпеки**. Забезпечує цілісність даних незалежно від клієнта.
- **Недоліки:** Зворотний зв'язок повільніший, вимагає додаткових мережевих запитів.
- **Висновок: Обов'язкова** для всіх вхідних даних, що зберігаються або обробляються.

Типи Правил Валідації

Валідація може перевіряти різні аспекти даних:

- **Обов'язковість (Required):** Перевірка того, що поле **не є порожнім** (наприклад, `[Required]` в C#/.NET).
- **Тип даних (Data Type):** Перевірка, чи є введене значення числом, датою, булевим значенням тощо.
- **Довжина/Діапазон (Length/Range):** Перевірка, чи відповідає довжина рядка або значення числа встановленим мінімальним/максимальним обмеженням (наприклад, `[StringLength]`, `[Range]`).
- **Формат (Format/Pattern):** Перевірка за допомогою **регулярних виразів (Regex)**, чи відповідає значення певному шаблону (наприклад, для електронної пошти, номера телефону, поштового індексу). Приклад: `[EmailAddress]`, `[RegularExpression]`.
- **Унікальність (Uniqueness):** Перевірка, чи не існує таке значення вже в базі даних (наприклад, унікальний логін).
- **Консистентність (Cross-Field Consistency):** Порівняння значень кількох полів (наприклад, перевірка, що "Пароль" і "Підтвердження пароля" збігаються).

Принципи Валідації (Контекст Microsoft/ASP.NET Core)

У фреймворках, таких як **ASP.NET Core**, валідація часто реалізується через:

1. **Атрибути даних (Data Annotations):** Використання спеціальних атрибутів (наприклад, `[Required]`, `[EmailAddress]`) для декорування властивостей у моделях даних.
2. **Model Binding:** Система автоматично зчитує дані, застосовує атрибути валідації та заповнює об'єкт моделі.
3. **Model State:** Після валідації, контролер перевіряє об'єкт `ModelState`. Якщо `ModelState.IsValid` дорівнює `false`, це означає, що є помилки, і запит має бути відхилений або користувачу має бути повернуто форму з повідомленнями про помилки.

Обробка Помилок Валідації

При виявленні помилки важливо:

1. **Повідомити про помилку:** Надати користувачу чітке та зрозуміле повідомлення про те, **що** не так і **як** це виправити.
2. **Зберегти введені дані:** При поверненні форми на клієнт, зберегти коректно введені дані, щоб користувачу не доводилося вводити все заново.
3. **Логувати:** На сервері помилки валідації можуть бути залоговані для моніторингу потенційних проблем або атак.

2. Створити модель з щонайменше 5 властивостями, використовуючи стандартні атрибути валідації ([Required], [StringLength], [EmailAddress], [Range], [Compare]). (max - 10 балів)

```
using Microsoft.AspNetCore.Mvc;
using System.ComponentModel.DataAnnotations;

namespace WebApplication1.Models
{
    3 references
    public class BookViewModel
    {

        [Required(ErrorMessage = "Назва книги є обов'язковою.")]
        [StringLength(100, MinimumLength = 3, ErrorMessage = "Довжина назви має бути від 3 до 100 символів.")]
        [Remote(action: "VerifyTitle", controller: "Books", ErrorMessage = "Така назва книги вже існує.")]
        [Display(Name = "Назва книги")]
        5 references
        public string Title { get; set; }

        // [EmailAddress] - Перевірка формату email
        [Required(ErrorMessage = "Email автора обов'язковий.")]
        [EmailAddress(ErrorMessage = "Некоректний формат Email.")]
        [Display(Name = "Email автора")]
        4 references
        public string AuthorEmail { get; set; }

        // [Compare] - Порівняння з іншим полем
        [Required(ErrorMessage = "Підтвердження Email є обов'язковим.")]
        [Compare("AuthorEmail", ErrorMessage = "Email адреси не співпадають.")]
        [Display(Name = "Підтвердження Email")]
        3 references
        public string ConfirmAuthorEmail { get; set; }

        // [Range] - Діапазон значень
        [Required]
        [Range(1900, 2025, ErrorMessage = "Рік має бути між 1900 та 2025.")]
        [Display(Name = "Рік видання")]
        4 references
        public int Year { get; set; }

        [Required]
        [Range(0, 10000, ErrorMessage = "Ціна має бути більше 0.")]
        [Display(Name = "Ціна")]
        4 references
        public decimal Price { get; set; }

        // Залежна Remote валідація (перевірка залежить від ціни)
        [Remote(action: "VerifyDiscount", controller: "Books", AdditionalFields = "Price", ErrorMessage = "Знижка не може бути більшою за ціну.")]
        [Display(Name = "Знижка")]
        4 references
        public decimal Discount { get; set; }
    }
}
```

3. Реалізувати серверну валідацію у контролері через ModelState.IsValid. Забезпечити повернення помилок у View. (max - 10 балів)

```
// 1Відображення форми
[HttpGet]
0 references
public IActionResult Create()
{
    return View();
}

// Обробка форми (Серверна валідація та збереження)
[HttpPost]
0 references
public async Task<IActionResult> Create(BookViewModel model)
{
    // Перевірка валідації на сервері
    if (ModelState.IsValid)
    {
        // перетворюємо  ViewModel у реальну сутність БД
        var bookEntity = new Book
        {
            Title = model.Title,
            AuthorEmail = model.AuthorEmail,
            Year = model.Year,
            Price = model.Price,
            Discount = model.Discount
        };

        // Зберігаємо в базу даних через репозиторій
        await _repository.AddAsync(bookEntity);

        // Повертаємо повідомлення про успіх або перенаправляємо
        return Content($"Книга '{model.Title}' успішно створена та збережена в БД! ID: {bookEntity.Id}");
    }

    // Якщо є помилки, повертаємо ту саму форму з помилками
    return View(model);
}
```

```
// Remote Validation: Перевірка унікальності назви
[AcceptVerbs("GET", "POST")]
0 references
public async Task<IActionResult> VerifyTitle(string title)
{
    // Перевіряємо в реальній БД, чи є така книга
    // Використовуємо ExistsAsync, який ми додали в лабораторній №2
    var exists = await _repository.ExistsAsync<Book>(b => b.Title == title);

    if (exists)
    {
        return Json($"Книга з назвою '{title}' вже існує.");
    }

    return Json(true);
}

// Dependent Remote Validation: Перевірка знижки залежно від ціни
[AcceptVerbs("GET", "POST")]

public IActionResult VerifyDiscount(decimal discount, decimal price)
{
    if (discount > price)
    {
        return Json("Знижка не може бути більшою за ціну товару.");
    }

    return Json(true);
}
```

4. Налаштувати клієнтську валідацію у Razor View: asp-validation-for, asp validation-summary, підключення _ValidationScriptsPartial. (max - 10 балів)

```
@section Scripts {
    @{
        await Html.RenderPartialAsync("_ValidationScriptsPartial");
    }
}
```

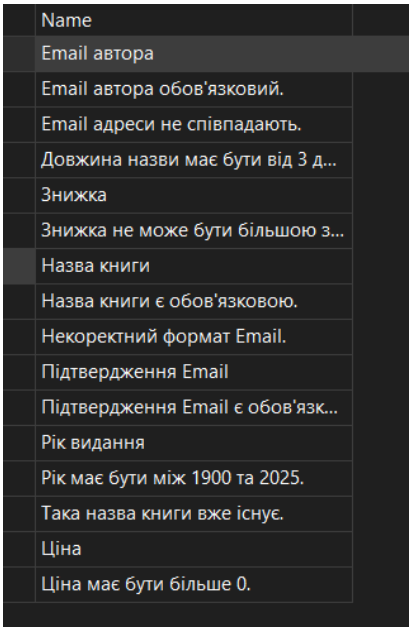
5. Реалізувати Remote Validation для перевірки унікальності email або логіна. Створити відповідний метод у контролері. (max - 20 балів)

```
[AcceptVerbs("GET", "POST")]
0 references
public async Task<IActionResult> VerifyTitle(string title)
{
    // Перевіряємо в реальній БД, чи є така книга
    // Використовуємо ExistsAsync, який ми додали в лабораторній №2
    var exists = await _repository.ExistsAsync<Book>(b => b.Title == title);

    if (exists)
    {
        return Json($"Книга з назвою '{title}' вже існує.");
    }

    return Json(true);
}
```

6. Реалізувати залежну Remote Validation з використанням AdditionalFields. Наприклад, перевірка промокоду залежно від суми замовлення (адаптувати завдання для своєї предметної області). (max - 15 балів)

```
// Dependent Remote Validation: Перевірка знижки залежно від ціни
[AcceptVerbs("GET", "POST")]
0 references
public IActionResult VerifyDiscount(decimal discount, decimal price)
{
    if (discount > price)
    {
        return Json("Знижка не може бути більшою за ціну товару.");
    }

    return Json(true);
}
```

7. Продемонструвати локалізацію повідомлень про помилки валідації через ресурсні файли. (max - 10 балів)



8. Зафіксувати зміни у проєкті на GitHub. (max - 10 балів)

9. Оформити звіт. (max - 5 балів)