Лабораторна робота №6

Звіт

З дисципліни “Сучасні інтернет технології”

на тему: “ Моделі відображення. Валідація даних у застосунку ASP.NET Core ”.

Студента 4 курсу: Групи МІТ-41
Демиденко Андрій

Київ - 2025р.

Хід виконання роботи

На тему: «Моделі відображення. Валідація даних у застосунку ASP.NET Core»
Мета роботи: Ознайомитися з теоретичними основами валідації даних в ASP.NET Core, навчитися створювати моделі відображення (ViewModels) та застосовувати до них атрибути валідації . Метою також є реалізація серверної та клієнтської валідації, впровадження механізму віддаленої (Remote) перевірки даних без перезавантаження сторінки, а також локалізація повідомлень про помилки.

Створення моделі відображення та застосування атрибутів

На початку роботи було розроблено спеціальну модель відображення OrderViewModel для сценарію оформлення замовлення. Цей клас не прив'язаний до бази даних, а слугує контейнером для передачі даних від користувача до контролера. До властивостей цієї моделі було застосовано стандартні атрибути валідації з простору імен System.ComponentModel.DataAnnotations. Зокрема, для забезпечення обов'язкового заповнення полів використано атрибут [Required], для обмеження довжини текстових полів — [StringLength], для перевірки числових значень (кількості та ціни) — [Range]. Також було використано спеціалізовані атрибути: [EmailAddress] для перевірки формату електронної пошти та [Compare] для гарантії співпадіння введеного email з полем підтвердження (рис. 1).

```
using System.ComponentModel.DataAnnotations;

namespace WebApplication.ViewModels
{
    public class OrderViewModel
    {

        [Required(ErrorMessage = "Назва товару є обов'язковою.")]
        [StringLength(50, MinimumLength = 3, ErrorMessage = "Назва товару має бути від 3 до 50 символів.")]
        [Display(Name = "Назва товару")]
        public string ProductName { get; set; }

        [Required]
        [Range(1, 100, ErrorMessage = "Кількість має бути від 1 до 100.")]
        [Display(Name = "Кількість")]
        public int Quantity { get; set; }

        [Required]
        [Range(0.01, 10000.00, ErrorMessage = "Ціна повинна бути більше 0.")]
        [Display(Name = "Ціна за одиницю")]
        public decimal Price { get; set; }

        [Required]
        [EmailAddress(ErrorMessage = "Некоректний формат Email.")]
        [Display(Name = "Email замовника")]
        public string CustomerEmail { get; set; }

        [Required]
        [Compare("CustomerEmail", ErrorMessage = "Email адреси не співпадають.")]
        [Display(Name = "Підтвердіть Email")]
        public string ConfirmCustomerEmail { get; set; }
    }
}
```

Рис. 1 – Модель OrderViewModel з атрибутами DataAnnotations.

Реалізація серверної валідації

Наступним етапом стала реалізація логіки перевірки на боці сервера. Було створено контролер OrderController із методами Create для GET та POST запитів. У POST-методі реалізовано перевірку властивості ModelState.IsValid. Ця властивість автоматично стає false, якщо отримані дані не відповідають правилам атрибутів моделі. У разі успішної валідації контролер повертає повідомлення про успіх, а у разі помилок — повертає користувачу те саме представлення з введеними даними та повідомленнями про некоректність. Для інтерфейсу було створено View Create.cshtml, де використано тег-хелпери asp-validation-for для відображення помилок біля кожного поля та asp-validation-summary для виведення загального списку помилок форми (рис. 2).

```
@model WebApplication.ViewModels.OrderViewModel

@{
    ViewData["Title"] = "Створити замовлення";
}

<h2>Оформлення замовлення</h2>

<div class="row">
    <div class="col-md-6">
        <form asp-action="Create" method="post">

            <div asp-validation-summary="ModelOnly" class="text-danger"></div>

            <div class="form-group mb-3">
                <label asp-for="ProductName" class="control-label"></label>
                <input asp-for="ProductName" class="form-control" />
                <span asp-validation-for="ProductName" class="text-danger"></span>
            </div>

            <div class="form-group mb-3">
                <label asp-for="Quantity" class="control-label"></label>
                <input asp-for="Quantity" class="form-control" />
                <span asp-validation-for="Quantity" class="text-danger"></span>
            </div>

            <div class="form-group mb-3">
                <label asp-for="Price" class="control-label"></label>
                <input asp-for="Price" class="form-control" />
                <span asp-validation-for="Price" class="text-danger"></span>
            </div>

            <div class="form-group mb-3">
                <label asp-for="CustomerEmail" class="control-label"></label>
                <input asp-for="CustomerEmail" class="form-control" />
                <span asp-validation-for="CustomerEmail" class="text-danger"></span>
            </div>

            <div class="form-group mb-3">
                <label asp-for="ConfirmCustomerEmail" class="control-label"></label>
                <input asp-for="ConfirmCustomerEmail" class="form-control" />
                <span asp-validation-for="ConfirmCustomerEmail" class="text-danger"></span>
            </div>

            <div class="form-group">
                <input type="submit" value="Замовити" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>
```

Рис. 2 – Результат роботи серверної валідації.

Налаштування клієнтської валідації

Для покращення користувацького досвіду (User Experience) було налаштовано клієнтську валідацію. У представлення Create.cshtml було підключено часткове представлення _ValidationScriptsPartial. Це дозволило активувати бібліотеки jQuery Validation та jQuery Validation Unobtrusive. Завдяки цьому перевірка даних відбувається безпосередньо у браузері миттєво після введення або втрати фокусу полем, що запобігає зайвим HTTP-запитам до сервера при явно некоректному заповненні полів (рис. 3).

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

Рис. 3 – Підключення скриптів клієнтської валідації.

Реалізація Remote Validation та залежної перевірки

Окрему увагу було приділено реалізації механізму **Remote Validation** для перевірок, що потребують серверної логіки (наприклад, перевірка унікальності даних у БД) без перезавантаження сторінки. У контролері OrderController створено метод VerifyEmail, який приймає введений email та повертає JSON-відповідь (true або повідомлення про помилку). До моделі OrderViewModel додано атрибут [Remote], що зв'язує поле CustomerEmail з цим методом.

Також було реалізовано складнішу, **залежну валідацію** для поля PromoCode. Метод VerifyPromoCode приймає не лише сам код, але й значення полів Price та Quantity завдяки властивості AdditionalFields атрибута [Remote]. Це дозволило реалізувати логіку, де валідність промокоду залежить від загальної суми замовлення (рис. 4).

```
+        [AcceptVerbs("GET", "POST")]
+        public IActionResult VerifyPromoCode(string promoCode, decimal price, int quantity)
+        {
+            if (string.Equals(promoCode, "LNU2025", StringComparison.OrdinalIgnoreCase))
+            {
+                var totalAmount = price * quantity;
+
+                if (totalAmount < 100)
+                {
+                    return Json($"Промокод '{promoCode}' діє лише для замовлень від 100 грн. Ваша
  сума: {totalAmount} грн.");
+                }
+            }
+            else
+            {
+
+                return Json($"Промокод '{promoCode}' не знайдено.");
+            }
+
+            return Json(true);
+        }
```

```
+         [Display(Name = "Промокод")]
+         [Remote(action: "VerifyPromoCode", controller: "Order", AdditionalFields =
  "Price,Quantity")]
+         public string? PromoCode { get; set; }
```

Рис. 4 – Реалізація залежної Remote-валідації.

Локалізація повідомлень валідації

На завершальному етапі було реалізовано локалізацію повідомлень про помилки. Всі "жорстко" прописані повідомлення у атрибутах валідації було замінено на ключі ресурсів (наприклад, RequiredError). У структурі проєкту створено папку Resources/ViewModels та додано файли .resx для англійської (базової), української та французької культур. Контролер OrderController було оновлено для використання сервісу IStringLocalizer, що дозволило повертати локалізовані повідомлення також і для методів Remote-валідації. Тестування підтвердило, що при зміні мови інтерфейсу всі повідомлення про помилки валідації автоматично відображаються обраною мовою (рис. 5).

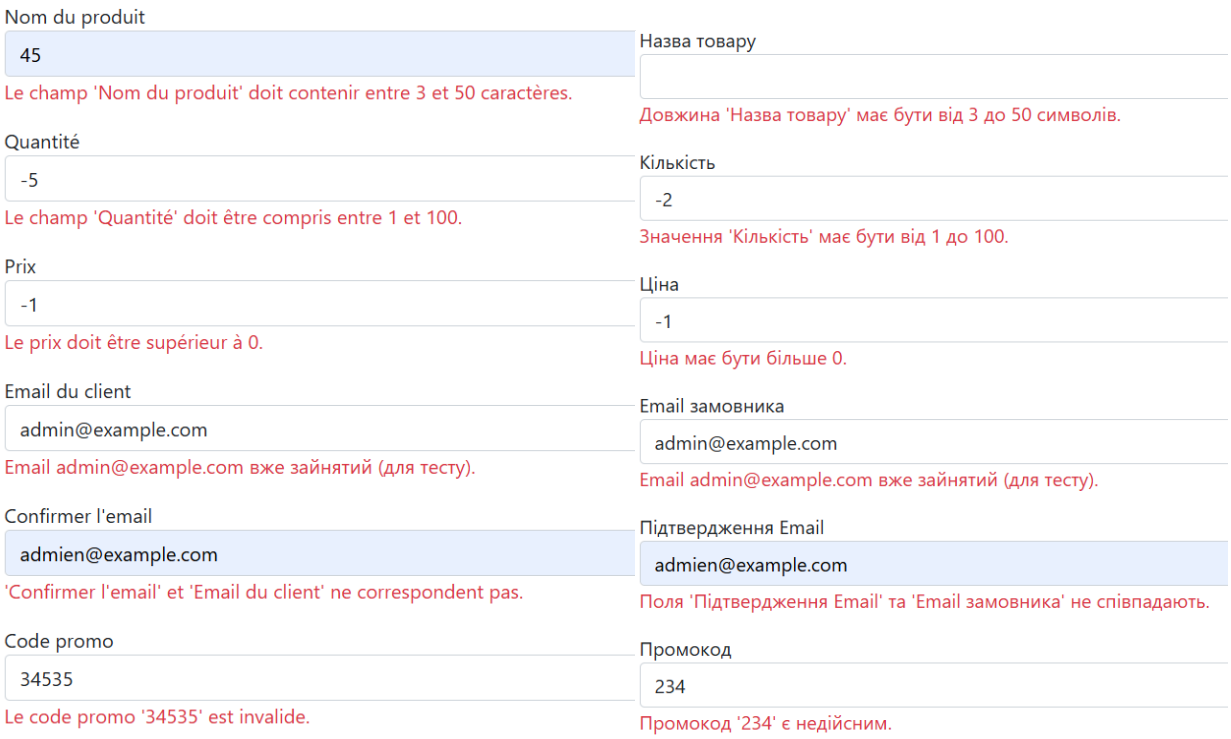

Рис. 5 – Локалізовані повідомлення про помилки.

Висновок

У ході виконання лабораторної роботи було успішно реалізовано комплексну систему перевірки даних у веб-застосунку ASP.NET Core. Ми опанували використання ViewModels для відділення логіки відображення від моделей даних, налаштували надійну серверну валідацію та зручну для користувача клієнтську валідацію. Було впроваджено технологію Remote Validation для асинхронних перевірок на сервері та забезпечено підтримку багатомовності для всіх повідомлень про помилки. Отримані навички дозволяють створювати надійні веб-інтерфейси, що запобігають введенню некоректних даних та забезпечують високий рівень інтерактивності.