ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет програмування та комп’ютерних

і телекомунікаційних систем

Кафедра інженерії програмного забезпечення

**Лабораторна робота №**5

з дисципліни «Програмування в інтернет»

на тему:

«Валідація моделі. Анотації даних для відображення властивостей. Атрибути валідації. Валідація моделі в контролері. Відображення помилок валідації. Розробка власної логіки валідації»

**Виконав:**

студент 1 курсу, групи ІПЗс-20-1, варіант 8 Н.В.Савич

**Перевірив:**\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.М.Яшина

(підпис)

Хмельницький – 2020 р.

Мета. Отримати навики розробки власної логіки валідації моделі.

Завдання

Реалізувати валідацію даних Web - додатка розробленого в завданні до лабораторної роботи №4. Реалізувати самовалідацію даних. Розробити власний провайдер валідації.

Код Програми:

Моделі:

Type.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

namespace lab1.Models

{

public class Type

{

public int Id { get; set; }

[UIHint("Назва")]

[StringLength(50, MinimumLength = 3, ErrorMessage = "Довжинарядка повинна бути від 3 до 50 символів")]

[Display(Name = "Введіть назву")]

public string Name { get; set; }

[UIHint("Введіть ціну")]

[Display(Name = "Цына")]

[DataType(DataType.Currency)]

public int Price { get; set; }

}

}

Order.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

namespace lab1.Models

{

public class Order

{

[Required]

public int OrderId { get; set; }

[UIHint("ПІБ")]

[StringLength(50, MinimumLength = 3, ErrorMessage = "Довжинарядка повинна бути від 3 до 50 символів")]

[Display(Name = "ПІБ")]

public string Person { get; set; }

[UIHint("Адреса")]

[Display(Name = "Адреса")]

public string Address { get; set; }

public int TypeId { get; set; }

[UIHint("Дата")]

[Display(Name = "Дата")]

public DateTime Date { get; set; }

}

}

AccauntViewModel.cs

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace BeekeepersBlog\_V2.Models

{

public class ExternalLoginConfirmationViewModel

{

[Required]

[Display(Name = "Адрес электронной почты")]

public string Email { get; set; }

}

public class ExternalLoginListViewModel

{

public string ReturnUrl { get; set; }

}

public class SendCodeViewModel

{

public string SelectedProvider { get; set; }

public ICollection<System.Web.Mvc.SelectListItem> Providers { get; set; }

public string ReturnUrl { get; set; }

public bool RememberMe { get; set; }

}

public class VerifyCodeViewModel

{

[Required]

public string Provider { get; set; }

[Required]

[Display(Name = "Код")]

public string Code { get; set; }

public string ReturnUrl { get; set; }

[Display(Name = "Запомнить браузер?")]

public bool RememberBrowser { get; set; }

public bool RememberMe { get; set; }

}

public class ForgotViewModel

{

[Required]

[Display(Name = "Адрес электронной почты")]

public string Email { get; set; }

}

public class LoginViewModel

{

[Required]

[Display(Name = "Адрес электронной почты")]

[EmailAddress]

public string Email { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

[Display(Name = "Запомнить меня")]

public bool RememberMe { get; set; }

}

public class RegisterViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Адрес электронной почты")]

public string Email { get; set; }

[Required]

[StringLength(100, ErrorMessage = "Значение {0} должно содержать не менее {2} символов.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Подтверждение пароля")]

[Compare("Password", ErrorMessage = "Пароль и его подтверждение не совпадают.")]

public string ConfirmPassword { get; set; }

}

public class ResetPasswordViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Адрес электронной почты")]

public string Email { get; set; }

[Required]

[StringLength(100, ErrorMessage = "Значение {0} должно содержать не менее {2} символов.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Подтверждение пароля")]

[Compare("Password", ErrorMessage = "Пароль и его подтверждение не совпадают.")]

public string ConfirmPassword { get; set; }

public string Code { get; set; }

}

public class ForgotPasswordViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Почта")]

public string Email { get; set; }

}

}

Global.asax:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.Web.Optimization;

using System.Web.Routing;

namespace BeekeepersBlog\_V2

{

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

}

}

Контролери:

AccauntController.cs

[Authorize]

public class AccountController : Controller

{

private ApplicationSignInManager \_signInManager;

private ApplicationUserManager \_userManager;

public AccountController()

{

}

public AccountController(ApplicationUserManager userManager, ApplicationSignInManager signInManager )

{

UserManager = userManager;

SignInManager = signInManager;

}

public ApplicationSignInManager SignInManager

{

get

{

return \_signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();

}

private set

{

\_signInManager = value;

}

}

public ApplicationUserManager UserManager

{

get

{

return \_userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

}

private set

{

\_userManager = value;

}

}

//

// GET: /Account/Login

[AllowAnonymous]

public ActionResult Login(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return View();

}

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)

{

if (!ModelState.IsValid)

{

return View(model);

}

var result = await SignInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, shouldLockout: false);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(returnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.RequiresVerification:

return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe = model.RememberMe });

case SignInStatus.Failure:

default:

ModelState.AddModelError("", "Неудачная попытка входа.");

return View(model);

}

}

}

HomeController.cs

using lab1.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace lab1.Controllers

{

public class HomeController : Controller

{

TypeContext db = new TypeContext();

public ActionResult Index()

{

IEnumerable<Models.Type> types = db.Types;

ViewBag.Types = types;

return View();

}

[HttpGet]

public ActionResult Buy(int id)

{

ViewBag.BookId = id;

return View();

}

[HttpPost]

public string Buy(Order order)

{

order.Date = DateTime.Now;

db.Orders.Add(order);

db.SaveChanges();

return "Спасибі," + order.Person + ", за покупку!";

}

}

}

TypeController.cs

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Web;

using System.Web.Mvc;

using lab1.Models;

namespace lab1.Controllers

{

public class TypesController : Controller

{

private TypeContext db = new TypeContext();

public ActionResult Index()

{

return View(db.Types.ToList());

}

public ActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "Id,Name,Price")] Models.Type type)

{

if (ModelState.IsValid)

{

db.Types.Add(type);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(type);

}

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Models.Type type = db.Types.Find(id);

if (type == null)

{

return HttpNotFound();

}

return View(type);

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "Id,Name,Price")] Models.Type type)

{

if (ModelState.IsValid)

{

db.Entry(type).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

return View(type);

}

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Models.Type type = db.Types.Find(id);

if (type == null)

{

return HttpNotFound();

}

return View(type);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

Models.Type type = db.Types.Find(id);

db.Types.Remove(type);

db.SaveChanges();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

Представлення:

Login.cshtml

@using BeekeepersBlog\_V2.Models

@model LoginViewModel

@{

ViewBag.Title = "Выполнить вход";

}

<h2>@ViewBag.Title.</h2>

<div class="row">

<div class="col-md-8">

<section id="loginForm">

@using (Html.BeginForm("Login", "Account", new { ReturnUrl = ViewBag.ReturnUrl }, FormMethod.Post, new { @class = "form-horizontal", role = "form" }))

{

@Html.AntiForgeryToken()

<h4>Используйте локальную учетную запись для входа.</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<div class="form-group">

@Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })

<div class="col-md-10">

@Html.TextBoxFor(m => m.Email, new { @class = "form-control" })

@Html.ValidationMessageFor(m => m.Email, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })

<div class="col-md-10">

@Html.PasswordFor(m => m.Password, new { @class = "form-control" })

@Html.ValidationMessageFor(m => m.Password, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<div class="checkbox">

@Html.CheckBoxFor(m => m.RememberMe)

@Html.LabelFor(m => m.RememberMe)

</div>

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Выполнить вход" class="btn btn-default" />

</div>

</div>

<p>

@Html.ActionLink("Регистрация нового пользователя", "Register")

</p>

}

</section>

</div>

<div class="col-md-4">

<section id="socialLoginForm">

@Html.Partial("\_ExternalLoginsListPartial", new ExternalLoginListViewModel { ReturnUrl = ViewBag.ReturnUrl })

</section>

</div>

</div>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

Register.cshtml

@model BeekeepersBlog\_V2.Models.RegisterViewModel

@{

ViewBag.Title = "Регистрация";

}

<h2>@ViewBag.Title.</h2>

@using (Html.BeginForm("Register", "Account", FormMethod.Post, new { @class = "form-horizontal", role = "form" }))

{

@Html.AntiForgeryToken()

<h4>Создайте новую учетную запись.</h4>

<hr />

@Html.ValidationSummary("", new { @class = "text-danger" })

<div class="form-group">

@Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })

<div class="col-md-10">

@Html.TextBoxFor(m => m.Email, new { @class = "form-control" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })

<div class="col-md-10">

@Html.PasswordFor(m => m.Password, new { @class = "form-control" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.ConfirmPassword, new { @class = "col-md-2 control-label" })

<div class="col-md-10">

@Html.PasswordFor(m => m.ConfirmPassword, new { @class = "form-control" })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" class="btn btn-default" value="Регистрация" />

</div>

</div>

}

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

Вигляд програми:

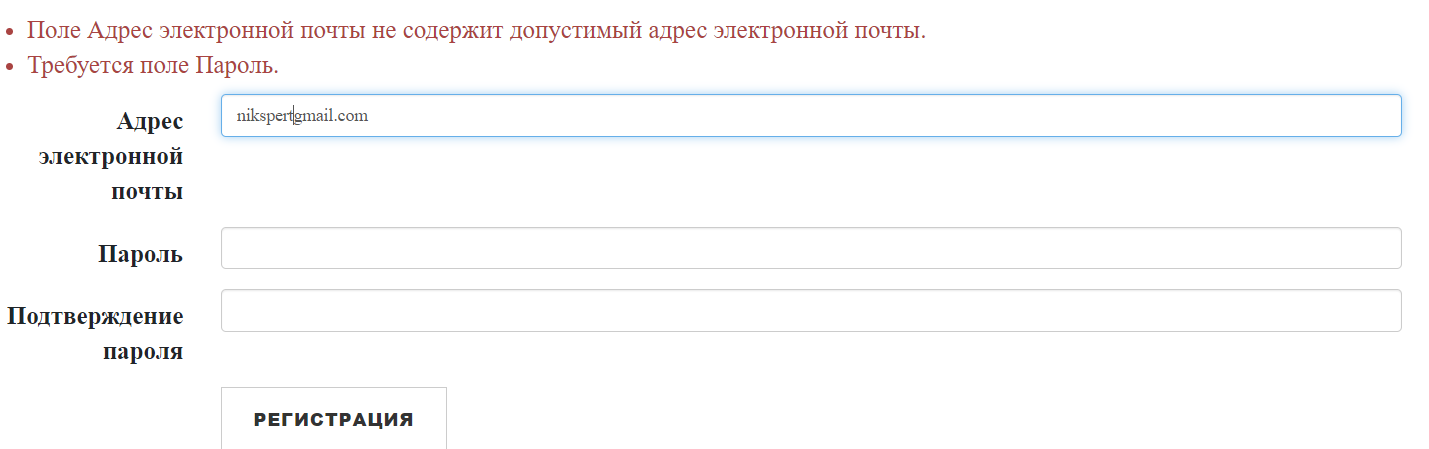


Рис 1.- Валідація на етапі реєстрації

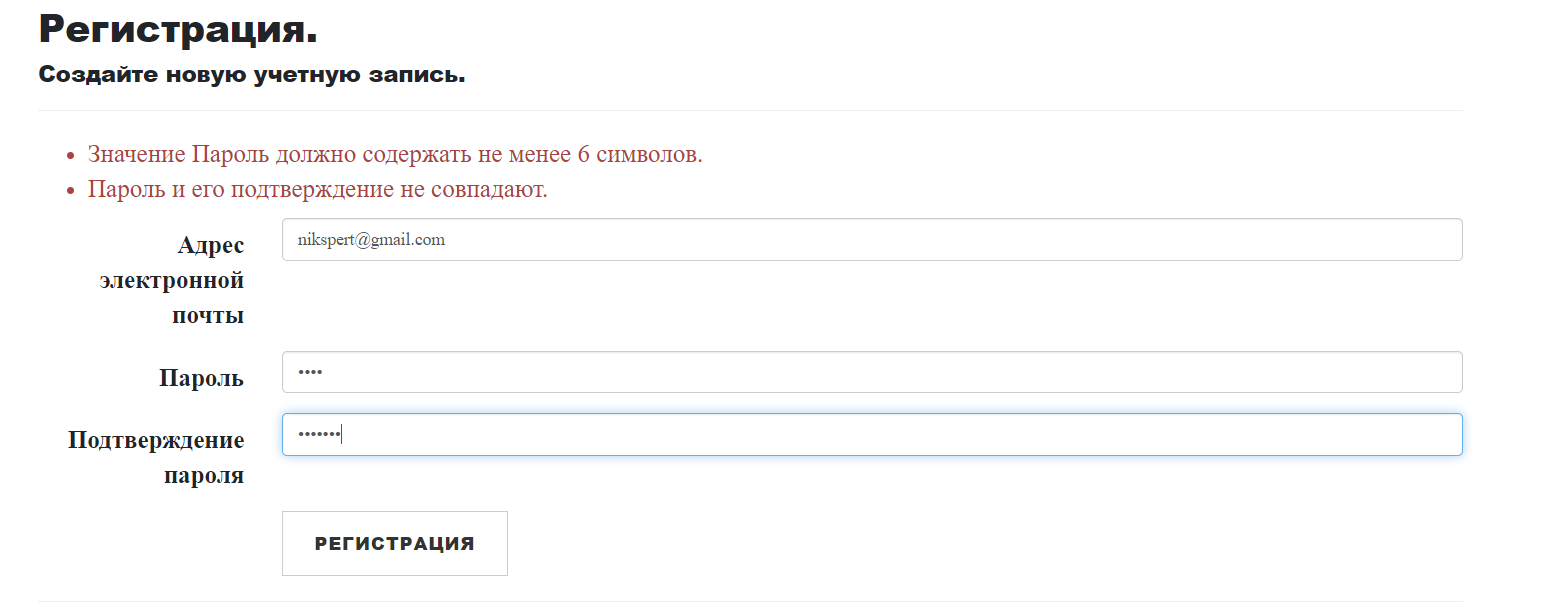


Рис 2.- Валідація на етапі реєстрації

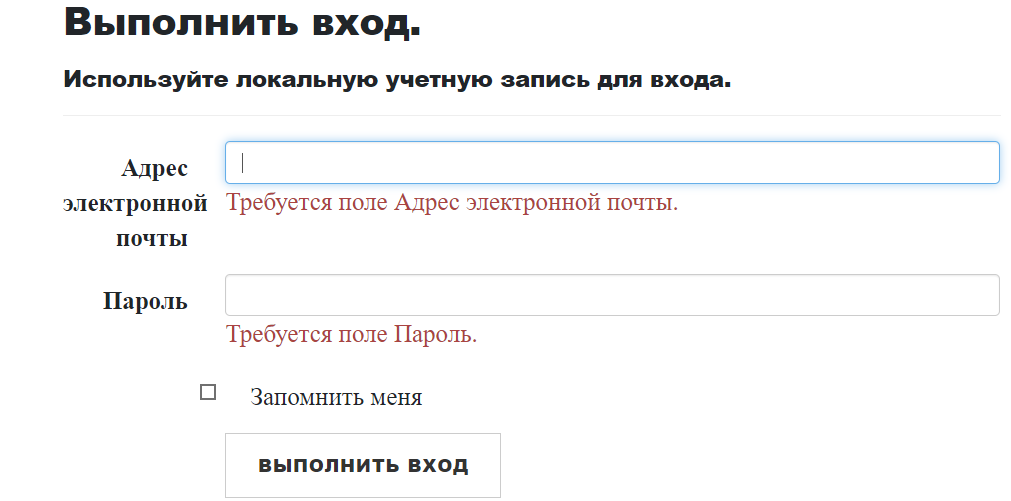


Рис 3.- Валідація логіну

Контрольні питання:

1. Метадані в MVC

При роботі з моделлю в MVC велике значення відіграють метадані. Вони дозволяють вказати деяку додаткову інформацію про об'єкт, наприклад, про те, як відображати його властивості в представленні, або про те, як здійснювати валідацію введення.

1. Анотації даних

Анотації даних для відображення властивостей. Анотації даних представляють собою атрибути, які знаходятся в просторі імен.

1. Атрибут Display

Як видно, заголовки у властивостей точно такі ж, як і назви властивостей, що не дуже зручно. Адже ми могли назвати властивості як завгодно. До того ж ми не хочемо, щоб назви на українськомовному сайті були англійською мовою. Звичайно, в цьому випадку ми могли б використовувати @Html.LabelFor для налаштування назви для кожної окремого властивості, але в цьому випадку у нас би збільшився в кілька разів обсяг коду, і ми не змогли б одним рядком вивести всю інформацію про модель. До того ж може знадобитися в декількох представленнях виводити інформацію про модель. Тому даний підхід не оптимальний. І тут нам на допомогу приходять анотації даних, а саме атрибут Display. Нам достатньо поправити модель, вказавши у неї параметри відображення:

1. Атрибут HiddenInput

Атрибут HiddenInput. У попередньому прикладі у нас залишилася одна проблемка - це поле Id. Іноді, звичайно, може знадобитися виведення поля Id. Але, наприклад, якщо б ми виводили модель в режимі редагування за допомогою хелпера @Html.EditorForModel(), то дане поле було б доступно для редагування, що не дуже добре, особливо коли ідентифікатори не повинні змінюватися. Щоб приховати це поле ми можемо застосувати атрибут HiddenInput:

1. Атрибут ScaffoldColumn

Атрибут ScaffoldColumn. При редагуванні моделі атрибут HiddenInput повністю не приховує поля, так як ми можемо подивитися вихідний код сторінки і знайти відповідні поля. Щоб повністю приховати властивість від хелперів, використовується атрибут ScaffoldColumn:

[ScaffoldColumn(false)]

public virtual int Id { get; set; }

Тепер хелпери редагування не побачать дану властивість і не створять для нього навіть приховане поле на сторінці.

1. Атрибут DataType

Атрибут DataType. Атрибут DataType дозволяє надавати середовищі виконання інформацію про використання властивості. Наприклад, припустимо, у нас є властивість Password:

[DataType (DataType.Password)]

public string Password {get; set; }

Для властивості з атрибутом DataType.Password HTML-хелпери створюють елемент введення, у якого атрибут type має значення "password". Тоді в браузері при введенні даних не побачите символи, що вводяться. Перерахування DataType може приймати кілька різних значень

1. Атрибут UIHint

Атрибут UIHint. Даний атрибут вказує, який буде використовуватися шаблон відображення при створенні розмітки html для даної властивості. Шаблон управляє, як властивість буде рендерится на сторінці. Є наступні вбудовані шаблони:

- Boolean - хелпери редагування створюють прапорець (checkbox) для булевих значень. Для значень типу bool? (nullable) створюється елемент select з параметрами True, False і Not Set. Хелпери відображення генерують ті ж елементи html, що і хелпери редагування, тільки з атрибутом disabled;

- Collection - Використовується відповідний шаблон для рендеринга кожного елемента колекції. Причому елементи можуть

бути різних типів;

- Decimal - Хелпери редагування створюють однорядкове текстове поле - елемент input;

- EmailAddress - Хелпери редагування створюють однорядкове текстове поле. Хелпери відображення генерують елемент посилання, де атрибут href має значення mailto: url;

- HiddenInput - Створюється приховане поле - елемент hidden input;

- Html - Хелпери редагування створюють однорядкове текстове поле. Хелпери відображення просто показують текст;

- MultilineText -Хелпери редагування створюють багаторядкове текстове поле (елемент textarea);

- Object - Хелпери вивчають властивості об'єкта і вибирають найбільш відповідні для нього шаблони;

- Password - Хелпери редагування створюють текстове поле для введення символів з використанням маски. Хелпери відображення показують пароль, без використання маски;

- String - Хелпери редагування створюють однорядкове текстове поле;

- Url - Хелпери редагування створюють текстове поле. Хелпери відображення створюють елемент посилання для даного Url;

1. Атрибут Required

Атрибут Required. Застосування цього атрибуту до властивості моделі означає, що дана властивість має бути обов'язково встановлена.

Щоб при валідації ми не отримували безглуздих повідомлень про помилку, цей атрибут дозволяє налаштувати текст повідомлення

1. Типізований контролер

Контролер одержує вхідні дані й перетворює їх на команди для моделі чи вигляду.

1. Використання атрибутів валідації

За допомогою атрибутів валідації моделі ми можемо управляти валідацією і укладати нескладну логіку перевірки значень властивостей вже в атрибути цих властивостей, не вдаючись до коду. Розглянемо атрибути валідації, які ми можемо застосувати в додатку на ASP.NET Core.

1. Використання хелперів валідації

Для валідації на стороні клієнта застосовується клас ValidationMessageTagHelper. Даний tag-хелперів використовується за допомогою застосування до елементу <span> атрибута asp-validation-for:

<Label asp-for = "Name"> </ label>

<Input type = "text" asp-for = "Name" />

<Span asp-validation-for = "Name" />

Атрибут asp-validation-for в якості значення приймає назву властивості моделі, для якого буде виводитися повідомлення про помилку валідації.

Даний елемент span буде генерувати наступну розмітку:

<Span class = "field-validation-valid" data-valmsg-for = "Name" data-valmsg-replace = "true" />

1. Валідація на стороні сервера

В фреймворку передбачена також валідація на стороні сервера:

[HttpPost]

public ActionResult Create(Book book)

{

if (ModelState.IsValid)

{

db.Books.Add(book);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(book);

}

return View (book);

}

1. Властивість ModelState.IsValid

За допомогою властивості ModelState.IsValid ми дізнаємося, проходить модель валідацію чи ні, і залежно від результату здійснюємо ті чи інші дії. Для демонстрації вибрано додаток за шаблоном Basic, однак якщо ви працюєте з проектом програми по шаблону Empty, то вам доведеться додавати всі ці файли jquery, код css та інше, щоб створити механізм валідації. Такий у загальному механізм валідації програми за замовчуванням. Тепер більш детально розглянемо основні моменти валідації.

1. Атрибут StringLength

Атрибут StringLength. Щоб користувач не міг ввести дуже довгий текст, використовується атрибут StringLength. Особливо це актуально, якщо в базі даних встановлено обмеження на розмір рядків. Першим параметром йде максимальна допустима довжина рядка. Іменовані параметри, зокрема MinimumLength і ErrorMessage, дозволяють поставити додаткові опції відображення

1. Атрибут RegularExpression

Атрибут RegularExpression. Застосування даного атрибута допускає, що значення яке вводиться має відповідати зазначеному в цьому атрибуті регулярному виразу. Найбільш поширений приклад - це перевірка коректності адреси електронної пошти.

1. Атрибут Range

Атрибут Range. Атрибут Range визначає мінімальні та максимальні обмеження для числових даних.

[Display(Name = "Рік")]

[Range (1700,2000, ErrorMessage = "Недопустимий рік")]

Атрибут Range може працювати як з цілочисельними значеннями, так і з числами з плаваючою крапкою:

[Range(typeof(decimal), "0.00", "49.99")]

public decimal Price { get; set; }

1. Атрибут Remote

бутів знаходиться в просторі імен System.Web.Mvc. Він дозволяє 105 виконувати валідацію на стороні клієнта із зворотними викликами на сервер. Наприклад, два користувача не можуть одночасно мати одне і теж значення UserName. Але за допомогою валідації на стороні клієнта важко гарантувати, що введене значення буде унікальним. А за допомогою атрибуту Remote ми можемо послати значення властивості UserName на сервер, а там воно вже порівнюється зі значеннями, що знаходяться в базі даних.

1. Валідація моделі в контролері

Крім валідації на стороні клієнта, ми можемо здійснювати валідацію і всередині контролера. Робиться це 106 за допомогою перевірки значення властивості ModelState.IsValid. Об'єкт ModelState зберігає всі значення, які користувач ввів для властивостей моделі, а також всі помилки, пов'язані з кожною властивістю і з моделлю в цілому. Якщо в об'єкті ModelState є якінебудь помилки, то властивість ModelState.IsValid поверне False

1. Перевірка значень окремих властивостей моделі

Ми також можемо перевіряти коректність значень окремих властивостей моделі:

[HttpPost]

public ActionResult Create(Book book) {

if (string.IsNullOrEmpty(book.Name)) {

ModelState.AddModelError("Name", "Некоректна назва книги");

}

else if (book.Name.Length > 5) {

ModelState.AddModelError("Name", "Недопустима довжина рядка");

}

if (ModelState.IsValid) {

107 ViewBag.Message = "Валідація пройдена";

db.Books.Add(book);

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.Message = "Запит не пройшов валідацію";

return View(book);

}

1. Метод ModelState.AddModelError

Метод ModelState.AddModelError додає для властивості, зазначеної в якості першого параметра

1. Хелпер Html.ValidationSummary

Для відображення повідомлень про помилки при валідації моделі ми можемо використовувати хелпер Html.ValidationSummary:

@using (Html.BeginForm()) {

108 @Html.ValidationSummary(false)

1. Хелпер Html.ValidationMessageFor

Ми можемо використовувати хелпер Html.ValidationMessageFor для виведення повідомлення про помилку для окремої властивості, причому поряд з полем для введення властивості.

1. Базовий клас ValidationAttribute

Усі атрибути валідації утворені від базового класу ValidationAttribute, який знаходиться в просторі імен System.ComponentModel. DataAnnotations. Тому саме від цього класу ми будемо утворювати свій атрибут. Припустимо, нам треба, щоб яка-небудь книга була написана обмеженим колом авторів. Створимо відповідний атрибут, який буде це перевіряти:

public class MyAuthorsAttribute : ValidationAttribute {

//масив для зберігання допустимих авторів

private static string[] myAuthors;

public MyAuthorsAttribute(string[] Authors) {

myAuthors = Authors;

}

public override bool IsValid(object value) {

if (value != null) {

string strval = value.ToString();

for (int i = 0; i < myAuthors.Length; i++) {

if (strval == myAuthors[i]) return true;

}

}

return false;

}

}

1. Атрибути валідації на рівні моделі

Атрибути валідації на рівні моделі застосовуються до перевірки комбінації властивостей.

1. Самовалідація і IValidatableObject

Самовалідація являє собою процес, при якому модель запускає механізм валідації із себе самої. І сама інкапсулює всю логіку валідації. Для цього клас моделі повинен реалізувати інтерфейс IValidatableObject.

1. Власний провайдер валідації

Для створення провайдера валідації нам потрібно створити клас, наслідуваний від класу ModelValidatorProvider і реалізувати його метод GetValidators.