Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Навчально-науковий
інститут комп'ютерних систем
КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни «Програмування та підтримка веб-застосувань»
для здобувачів першого (бакалаврського) рівня вищої освіти
за спеціальністю
124 Системний аналіз
(частина 2)

Одеса – 2025

Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Навчально-науковий
інститут комп'ютерних систем
КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни «Програмування та підтримка веб-застосувань»
для здобувачів першого (бакалаврського) рівня вищої освіти
за спеціальністю
124 Системний аналіз
(частина 2)

Затверджено
на засіданні кафедри прикладної математики та інформаційних технологій
Протокол № _ від _____

Одеса – 2025

Конспект лекцій з дисципліни «Програмування та підтримка веб-застосувань» для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 124 Системний аналіз (частина 2) / Укл.: К. О. Трифонова. − Одеса, 2025. − 34 с.

Укладач: Трифонова К. О., ст. викл.

ЗМІСТ

Вступ

Навчальна дисципліна «Програмування та підтримка веб-застосувань» є необхідною дисципліною для підвищення рівня теоретичних і прикладних знань, що формують фахівця в галузі інформаційних технологій, які сприяють утворенню у здобувачів поглиблених вмінь та навичок веб-розробки програмного забезпечення із застосуванням сучасних парадигм програмування, тобто підготовці спеціалістів для створення, впровадження та підтримки професійно-орієнтованих комп'ютерних технологій у професійній діяльності.

Дисципліна «Програмування та підтримка веб-застосувань» відповідає освітньо-професійній програмі, навчальному плану підготовки фахівців першого (бакалаврського) рівня вищої освіти спеціальності 124 Системний аналіз і є складовою циклу дисциплін професійної підготовки вибіркової частини навчального плану.

Дисципліну викладають впродовж третього семестру першого (бакалаврського) рівня. У процесі навчання передбачено лекції, лабораторні заняття.

Згідно навчального плану передбачено підсумковий контроль у вигляді заліку.

Робочу програму навчальної дисципліни укладено згідно з вимогами кредитно-модульної системи організації навчального процесу. Програма визначає обсяги компетентностей, які повинен опанувати здобувач відповідно до освітньо-професійної програми, алгоритму вивчення навчального матеріалу дисципліни «Програмування та підтримка веб-застосувань», необхідне методичне забезпечення, складові та технологію оцінювання навчальних досягнень.

Предмет навчальної дисципліни «Програмування та підтримка веб-застосувань» – сучасні технології веб-розробки програмного забезпечення.

Мета вивчення навчальної дисципліни «Програмування та підтримка веб-застосувань» – забезпечити формування поглибленої системи теоретичних і практичних знань у галузі веб-розробки програмного забезпечення для реалізації здатності використання інформаційно-комунікаційних технологій, сучасних методів і моделей інформаційної безпеки та/або кібербезпеки.

Завдання вивчення дисципліни:

– формувати теоретичні основи та практичні навички застосування методів вилучення, аналізу, обробки та синтезу інформації для формування технічного завдання при розв'язанні задач в галузі інформаційних технологій;

– формувати теоретичні основи та практичні навички сучасних парадигм веб-програмування та способів їх вибору з позицій зручності та якості застосування для реалізації методів та алгоритмів розв'язання задач в галузі інформаційних технологій;

– поглибити практичні навички кодування веб-програмного забезпечення для розв'язання задач в галузі інформаційних технологій;

– поглибити практичні навички налагодження веб-програмного забезпечення для розв'язання задач в галузі інформаційних технологій.

Стратегічні цілі дисципліни – націлити майбутніх фахівців на творче застосування отриманих знань у подальшій професійній підготовці та їх наступній практичній діяльності.

Лекція №9. Представлення

1. Створення спеціального механізму візуалізації
   1.1. Створення спеціальної реалізації інтерфейсу IView
   1.2. Створення реалізації інтерфейсу IViewEngine
   1.3. Реєстрація спеціального механізму візуалізації
   1.4. Тестування механізму візуалізації
2. Робота з механізмом Razor
   2.1. Прояснення представлень Razor
3. Додавання динамічного вмісту до представлення Razor
   3.1. Використання розділів компоновки
   3.2. Використання часткових представлень
   3.3. Додавання вмісту JSON до представлення
4. Конфігурування механізму Razor
   4.1. Розширювачі розташувань представлень

У розділі «Контролери та дії» було показано, що методи дій можуть повертати об'єкти ViewResult, які повідомляють інфраструктурі MVC про необхідність візуалізації представлення та повернення HTML-відповіді клієнту. У попередніх розділах було представлено використання представлень, в даному розділі будуть представлені деталі. Буде продемонстровано, як інфраструктура MVC обробляє об'єкти ViewResult із застосуванням механізмів візуалізації, та продемонстровано створення спеціального механізму візуалізації. Крім того, будуть описані прийоми ефективної роботи із вбудованим механізмом візуалізації Razor, у тому числі використання часткових представлень та розділів компоновки, які є важливими темами для побудови ефективних застосувань MVC.

1. Створення спеціального механізму візуалізації

Цінність створення спеціального механізму візуалізації в тому, що можна побачити те, що відбувається "за лаштунками" і розширити знання роботи MVC. На доданок можна зрозуміти, наскільки велику свободу мають механізми візуалізації при трансляції об'єктів ViewResult у відповіді, призначені клієнтам.

Механізми візуалізації – це класи, що реалізують інтерфейс IViewEngine, який визначений у просторі імен AspNetCore.Mvc.ViewEngines.

Роль механізму візуалізації полягає у трансляції запитів до представлень в об'єкти ViewEngineResult. Коли інфраструктура MVC потребує представлення, вона починає з виклику методу GetView(), який дає механізм візуалізації можливість надати представлення, просто використовуючи його ім'я.

Якщо методу GetView() не вдалося надати представлення, тоді викликається метод FindView(), так що механізм візуалізації отримує шанс пошукати представлення із застосуванням об'єкта ActionContext, який містить інформацію про метод дії, що створив об'єкт ViewResult.

Робота механізму візуалізації пов'язана з постачанням інфраструктури MVC об'єктами ViewEngineResult, які можуть використовуватися для генерації відповідей. Створювати екземпляри класу ViewEngineResult безпосередньо не можна, але для створення екземплярів він пропонує статичні методи.

Ще одним будівельним блоком системи механізму візуалізації є інтерфейс IView, який застосовується для опису функціональності, що забезпечується представленням, незалежно від того, який механізм візуалізації їх створив.

Найпростіший спосіб з'ясувати, як працюють типи IViewEngine, ViewEngineResult та IView передбачає створення механізму візуалізації. Можна побудувати простий механізм візуалізації, який повертає один вид представлення. Представлення візуалізуватиме результат, що містить

інформацію про запит і дані представлення, які згенеровані методом дії. Такий підхід дозволяє продемонструвати спосіб оперування механізмом візуалізації, не ув'язуючи у розборі шаблонів представлень і не займаючись відтворенням інших засобів, які підтримує Razor.

1.1. Створення спеціальної реалізації інтерфейсу IView

Створення спеціальної реалізації механізму візуалізації потребує реалізації інтерфейсу IView.

Коли представлення візуалізується, воно записує деталі даних маршрутизації та даних представлення, отримані з використанням аргументу ViewContext методу RenderAsync(). Відповіддю є простий текст, тому за допомогою об'єкта контексту заголовок Content-Type запиту встановлюється в text/plain. Без такого присвоєння ASP.NET за замовчуванням застосує text/html, що змусить браузер відображати дані у вигляді єдиної нерозбитої строчки символів.

1.2. Створення реалізації інтерфейсу IViewEngine

Мета механізму візуалізації – випуск об'єкта ViewEngineResult, який містить або екземпляр реалізації IView, або список розташувань, де здійснювався пошук відповідного представлення. Наступним кроком є реалізація інтерфейсу IViewEngine.

1.3. Реєстрація спеціального механізму візуалізації

Механізми візуалізації реєструються у файлі Program.cs шляхом конфігурування об'єкта MvcViewOptions.

1.4. Тестування механізму візуалізації

Коли необхідно гарантувати, що застосовуватиметься лише спеціальний механізм візуалізації, знадобиться викликати метод Clear() на колекції механізмів візуалізації, щоб видалити з неї Razor.

2. Робота з механізмом Razor

Складність в механізм візуалізації привноситься системою шаблонів представлень, яка включає фрагменти коду, підтримує компоновки і забезпечує оптимізацію продуктивності. У простому механізмі візуалізації нічого подібного не виконується, тому що вбудований механізм Razor надає всі ці засоби та багато іншого. Насправді в Razor доступна функціональність, яка потрібна практично у всіх застосуваннях MVC. Лише в незначній кількості проектів розробники беруть на себе працю створювати спеціальний механізм візуалізації.

2.1. Прояснення представлень Razor

Навіть початкове розуміння роботи механізму Razor може допомогти помістити великий обсяг функціональності в контекст і відкрити таємницю обробки файлів CSHTML.

Так яким же чином Razor бере суміш елементів HTML з операторами C# і випускає вміст для HTTP-відповіді. Механізм Razor перетворює файли CSHTML на класи C#, компілює їх і потім створює нові екземпляри щоразу, коли представленню потрібно згенерувати результат.

Ім'я класу

Механізму Razor потрібен який-небудь спосіб для трансляції імені та шляху до файлу CSHTML в клас, який він створює, коли робить розбір файлу, і Razor робить це за рахунок кодування інформації в імені класу. Механізм Razor забезпечує ім'я класу префіксом ASPV, за

яким слідує ім'я проекту, ім'я контролера і, нарешті, ім'я файлу представлення: така комбінація полегшує перевірку доступності класу при запитуванні інфраструктурою MVC представлення через IViewEngine.

Базовий клас

Класи представлень успадковуються від класу RazorPage або класу RazorPage<T>, якщо за допомогою директиви @model був вказаний тип моделі. Клас RazorPage пропонує методи та властивості, які можуть використовуватися у файлах CSHTML для доступу до засобів MVC.

Візуалізація представлень

На додаток до властивостей та методів, що надають функціональні засоби для розробників, до обов'язків класу RazorPage також входить генерування вмісту відповідей через його метод ExecuteAsync().

3. Додавання динамічного вмісту до представлення Razor

Основна мета представлень – уможливити візуалізацію частин моделі предметної області користувачам. Для цього необхідно вміти додавати до представлень динамічний вміст. Динамічний вміст генерується під час виконання та може бути різним для кожного запиту. Він є протилежністю статичного вмісту, такого як HTML-розмітка, що створюється при написанні коду застосування та однаково для всіх запитів.

3.1. Використання розділів компоновки

Механізм візуалізації Razor підтримує концепцію розділів, які дозволяють організовувати області вмісту всередині компоновки. Розділи Razor забезпечують більший контроль над тим, які частини представлення вставляються у компоновку і куди вони розташовуються.

Розділи визначаються з використанням Razor-виразу @section, за яким слідує ім'я розділу.

Розділи визначаються у представленні, але застосовуються у компоновці за допомогою виразу @RenderSection.

3.2. Використання часткових представлень

Часто виникає необхідність у застосуванні одних і тих же фрагментів дескрипторів Razor та розмітки HTML у кількох розташуваннях застосування. Замість дублювання вмісту можна використовувати часткові представлення – окремі файли представлень, що містять фрагменти дескрипторів та розмітку, які можуть бути включені до інших представлень.

3.3. Додавання вмісту JSON до представлення

Вміст JSON часто включається до представлення з метою постачання коду JavaScript клієнтської сторони даними, які можуть застосовуватися при динамічній генерації вмісту. Для підготовки до такого прикладу необхідно додати пакет jQuery.

Вираз @Json.Serialize приймає об'єкт і серіалізує його у форматі JSON.

4. Конфігурування механізму Razor

Механізм Razor можна конфігурувати із застосуванням класу RazorViewEngineOptions, який знаходиться у просторі імен Microsoft.AspNetCore.Mvc.Razor. Клас визначає дві конфігураційні властивості.

Властивість FileProvider не відноситься до тих властивостей, які будуть змінювати багато застосувань, оскільки читання файлів представлень з диска є точно тим, що потрібно в

більшості проектів. Механізм Razor використовує даний постачальник тільки для завантаження представлень, щоб їх можна було скомпілювати, коли застосування запускається вперше. Однак властивість ViewLocationExpanders корисніша, оскільки вона дозволяє застосовувати спеціальну логіку щодо способу, яким Razor шукає представлення.

4.1. Розширювачі розташувань представлень

Розширювачі розташувань представлень використовуються механізмом Razor для побудови списку розташувань, в яких повинен вестися пошук представлень. Розширювачі розташувань представлень реалізують інтерфейс IViewLocationExpander.

Контрольні питання

1. Дайте визначення поняттю представлення для шаблону програмного забезпечення MVC.
2. Дайте визначення поняттю механізм візуалізації для застосування ASP.NET Core MVC.
3. Поясніть необхідність розробки спеціального механізму візуалізації для застосування ASP.NET Core MVC.
4. Поясніть призначення класу, який реалізує інтерфейс IView, для розробки спеціального механізму візуалізації для застосування ASP.NET Core MVC.
5. Поясніть призначення класу, який реалізує інтерфейс IViewEngine, для розробки спеціального механізму візуалізації для застосування ASP.NET Core MVC.
6. Поясніть необхідність реєстрації спеціального механізму візуалізації для застосування ASP.NET Core MVC.
7. Поясніть механізм функціонування спеціального механізму візуалізації для застосування ASP.NET Core MVC.
8. Назвіть вбудований механізм візуалізації для застосування ASP.NET Core MVC.
9. Поясніть механізм функціонування вбудованого механізму візуалізації для застосування ASP.NET Core MVC.
10. Назвіть базовий клас для класу представлення, який формується вбудованим механізмом візуалізації для застосування ASP.NET Core MVC.
11. Поясніть призначення статичного вмісту для представлення для застосування ASP.NET Core MVC.
12. Поясніть призначення динамічного вмісту для представлення для застосування ASP.NET Core MVC.
13. Поясніть призначення компоновки для представлення для застосування ASP.NET Core MVC.
14. Поясніть призначення розділу представлення для компоновки для застосування ASP.NET Core MVC.
15. Поясніть необхідність виконання перевірити існування розділу представлення у компоновці для застосування ASP.NET Core MVC.
16. Поясніть необхідність використання необов'язкового розділу представлення у компоновці для застосування ASP.NET Core MVC.
17. Поясніть призначення часткових представлень для застосування ASP.NET Core MVC.
18. Поясніть необхідність використання у представленні включення даних у форматі JSON у відповідь, що надсилається браузеру для застосування ASP.NET Core MVC.
19. Поясніть стандартний механізм функціонування для вбудованого механізму візуалізації пошуку розташування представлення для застосування ASP.NET Core MVC.
20. Поясніть необхідність розширення розташування представлень для вбудованого механізму візуалізації для застосування ASP.NET Core MVC.

Лекція №10 Компоненти представлень

1. Поняття компонентів представлень
2. Створення компонента представлення
   2.1. Створення компонентів представлень POCO
   2.2. Спадкування від базового класу ViewComponent
   2.3. Поняття результатів компонентів представлень
   2.4. Отримання даних контексту
   2.5. Створення асинхронних компонентів представлень
3. Створення гібридних класів контролерів та компонентів представлень
   3.1. Створення гібридних представлень
   3.2. Застосування гібридного класу

В даному розділі розглядаються компоненти представлень, які є новим додаванням до інфраструктури ASP.NET Core MVC та замінюють собою засіб дочірніх дій із попередніх версій. Компоненти представлень – це класи, які для підтримки часткових представлень пропонують логіку в стилі дій, що дозволяє можливість вбудовувати в представлення складний вміст і одночасно зберігати код C#, що підтримує його, простим у плані супроводу і модульного тестування.

1. Поняття компонентів представлень

Застосуванням зазвичай необхідно вбудовувати у представлення вміст, який не пов'язаний з головним призначенням застосування. Найпоширеніші приклади включають інструменти навігації по сайту та панелі автентифікації, які дозволяють користувачеві входити, не відвідуючи окрему сторінку.

Загальна ідея всіх згаданих прикладів полягає в тому, що дані, необхідні для відображення вбудованого вмісту, не є частиною даних моделі, що передаються з дії до представлення. Саме з цієї причини в прикладі застосування були створені два сховища: буде необхідно відображати вміст, що генерується з використанням сховища об'єктів City, що нелегко робити в представленні, яке отримує від своїх дій дані зі сховища об'єктів Product.

Часткові представлення використовуються для створення багаторазово використовуваної розмітки, яка потрібна в застосуваннях, уникаючи дублювання одного і того ж вмісту в багатьох місцях застосування. Часткові представлення – зручний засіб, але вони просто містять фрагменти HTML-розмітки та директиви Razor, а дані, з якими вони мають справу, виходять від батьківського представлення. Якщо потрібно відображати інші дані, тоді виникає проблема. Можна було б отримувати доступ до необхідних даних прямо з часткового представлення, але такий підхід порушить принцип поділу обов'язків, що є фундаментом патерна MVC, і його результатом виявиться знаходження логіки вилучення та обробки даних у файлі представлення, де вона не може бути піддана модульному тестуванню. Альтернативно можна було б розширити застосовувані в застосуванні моделі представлень, щоб включити необхідні дані, але тоді довелося б змінити кожний метод дії і тим самим ускладнити ізоляцію функціональності методів дій з метою ефективного тестування.

Тут на допомогу приходять компоненти представлень. Компонент представлення – це клас C#, який пропонує часткове представлення з необхідними йому даними, незалежне від батьківського представлення та від візуалізуючої дії. У такому відношенні компонент представлення можна трактувати як спеціалізовану дію, яка використовується тільки для доставки часткового представлення з даними; воно не може отримувати HTTP-запити, а вміст, що їм видається, завжди включатиметься до батьківського представлення.

2. Створення компонента представлення

Компоненти представлень можна створювати трьома способами: визначаючи компонент представлення POCO, наслідуючи від базового класу ViewComponent і застосовуючи атрибут ViewComponent. Прийоми з POCO та базовим класом описані в наступних розділах, а використання атрибуту ViewComponent пояснюється в розділі "Створення гібридних класів контролерів і компонентів представлень".

2.1. Створення компонентів представлень POCO

Компонент представлення POCO є класом, який підтримує функціональність компонента представлення, не покладаючись на якісь API-інтерфейси MVC. Як і у випадку контролерів POCO, працювати з таким видом компонента представлення незручно, але корисно знати особливості його функціонування. Компонентом представлення POCO буде будь-який клас з ім'ям, що закінчується на ViewComponent, в якому визначено метод Invoke(). Класи компонентів представлень можуть визначатися будь-де в застосуванні, але за згодою вони збираються разом у папці на ім'я Components, розташованій на кореневому рівні проекту.

Для отримання необхідних служб компоненти представлень можуть використовувати засіб впровадження залежностей.

Для використання компонента представлення потрібен Razor-вираз @await Component.Invoke(). Компонент представлення вибирається за рахунок вказівки в аргументі імені класу без закінчення ViewComponent.

2.2. Спадкування від базового класу ViewComponent

Компоненти представлень POCO обмежені у функціональності, якщо тільки не задіяний API-інтерфейс MVC, що цілком можливо, але вимагає набагато більших зусиль, ніж інший поширений підхід, що передбачає успадкування від класу ViewComponent. Клас ViewComponent, визначений у просторі імен Microsoft.AspNetCore.Mvc, забезпечує зручний доступ до даних контексту та полегшує генерацію результатів.

2.3. Поняття результатів компонентів представлень

Можливість вставки в батьківське представлення простих строкових значень не дуже корисна, але на щастя компоненти представлень здатні набагато більше. Більше складних ефектів можна досягти, змусивши метод Invoke() повертати об'єкт, який реалізує інтерфейс IViewComponentResult.

Повернення часткового представлення

Найкориснішою відповіддю є незграбно іменований об'єкт ViewViewComponentResult, який повідомляє механізму Razor про необхідність візуалізації часткового представлення та включення результату до батьківського представлення. Для створення об'єктів ViewViewComponentResult базовий клас ViewComponent пропонує метод View(), який має чотири версії.

Вибір часткового представлення в компоненті представлення схожий на вибір представлення в контролері, але з двома важливими відмінностями: механізм Razor шукає представлення в інших місцезнаходженнях та застосовує інше стандартне ім'я представлення, якщо воно не вказано.

Повернення фрагментів HTML-розмітки

Клас ContentViewComponentResult застосовується для включення фрагментів HTML-розмітки до батьківського представлення, не використовуючи представлення.

Екземпляри класу ContentViewComponentResult створюються із застосуванням методу Content(), успадкованого від базового класу ViewComponent, який приймає значення string.

2.4. Отримання даних контексту

Деталі про поточний запит та батьківське представлення компонент представлення отримує через властивості класу ViewComponentContext.

Базовий клас ViewComponent пропонує набір зручних властивостей, що полегшують доступ до специфічної інформації контексту.

Отримання даних контексту з батьківського представлення з використанням аргументів

Батьківські представлення можуть надавати додаткові дані контексту у вигляді аргументів для виразу @await Component.InvokeAsync(). Таку функціональну можливість можна застосовувати для отримання даних з моделі батьківського представлення або для повідомлення інструкції про тип вмісту, який повинен видавати компонент представлення.

2.5. Створення асинхронних компонентів представлень

Всі розглянуті до цих пір приклади були синхронними компонентами представлень, які легко впізнати за тим, що вони визначали метод Invoke(). Якщо компонент представлення покладається на асинхронні API-інтерфейси, то можна створити асинхронний компонент представлення, визначивши метод InvokeAsync(), який повертає об'єкт Task. Коли механізм Razor отримує об'єкт Task з методу InvokeAsync(), він буде очікувати його завершення і потім вставить результат у головне представлення.

3. Створення гібридних класів контролерів та компонентів представлень

Компоненти представлень часто забезпечують зведення або коротку характеристику функціональності, що підтримується контролером. Наприклад, компоненти представлень, що відображають зведення по кошику для покупок, нерідко містять посилання, націлене на контролер, який виводить докладний список товарів у кошику і може використовуватися для завершення покупок та переходу до оплати.

У такій ситуації можна створити клас, що є одночасно контролером і компонентом представлення, що дозволяє згрупувати разом пов'язану функціональність і скоротити дублювання коду.

Атрибут ViewComponent застосовується до класів, які не успадковані від базового класу ViewComponent і не мають у своїх іменах суфікс ViewComponent, нормальний процес виявлення не зможе зарахувати їх до категорії компонентів представлень. Властивість Name встановлює ім'я, за яким на клас можна посилатися під час його використання у виразі @Component.Invoke() всередині батьківського представлення.

Оскільки гібридні класи не успадковуються від базового класу ViewComponent, вони не мають доступу до зручних методів для створення об'єктів реалізації IViewComponentResult, що означає необхідність в створенні об'єкта ViewViewComponentResult безпосередньо, як вимагалося б у компоненті представлення POCO.

3.1. Створення гібридних представлень

Гібридний клас вимагає двох наборів представлень: того, що візуалізується у разі застосування класу як контролера, і того, що візуалізується при використанні класу як компонента представлення.

3.2. Застосування гібридного класу

Фінальний крок передбачає застосування гібридного класу як компонента представлення в компоновці, що розділяється, з використанням імені, яке було зазначено за допомогою атрибуту ViewComponent.

Контрольні питання

1. Дайте визначення поняттю компонента представлення для застосування ASP.NET Core MVC.
2. Поясніть необхідність використання компоненти представлення для застосування ASP.NET Core MVC.
3. Перерахуйте способи створення компоненти представлення для застосування ASP.NET Core MVC.
4. Дайте визначення поняттю компонента представлення POCO для застосування ASP.NET Core MVC.
5. Поясніть особливості реалізації компоненти представлення POCO для застосування ASP.NET Core MVC.
6. Дайте визначення поняттю компонента представлення, яка є похідним класом від класу ViewComponent для застосування ASP.NET Core MVC.
7. Поясніть переваги використання компоненти представлення, яка є похідним класом від класу ViewComponent для застосування ASP.NET Core MVC.
8. Назвіть клас, який реалізує інтерфейс IViewComponentResult, екземпляр якого повертається компонентою представлення для повідомлення механізму Razor про необхідність візуалізації часткового представлення для застосування ASP.NET Core MVC.
9. Поясніть механізм функціонування для механізму Razor пошуку розташування часткового представлення, що запитується компонентою представлення для застосування ASP.NET Core MVC.
10. Назвіть клас, який реалізує інтерфейс IViewComponentResult, екземпляр якого повертається компонентою представлення для включення фрагментів HTML-розмітки до батьківського представлення, не використовуючи представлення для застосування ASP.NET Core MVC.
11. Поясніть необхідність повернення фрагмента надійної HTML-розмітки компонентою представлення для застосування ASP.NET Core MVC.
12. Перерахуйте властивості класу ViewComponentContext, які дозволяють отримати дані контексту для компоненти представлення для застосування ASP.NET Core MVC.
13. Перерахуйте властивості класу ViewComponent, які дозволяють отримати дані контексту для компоненти представлення для застосування ASP.NET Core MVC.
14. Поясніть необхідність використання асинхронної компоненти представлення для застосування ASP.NET Core MVC.
15. Поясніть механізм функціонування асинхронної компоненти представлення для застосування ASP.NET Core MVC.
16. Поясніть необхідність використання гібридного класу контролера та компоненти представлення для застосування ASP.NET Core MVC.
17. Поясніть недолік реалізації гібридного класу контролера та компоненти представлення для застосування ASP.NET Core MVC.
18. Поясніть механізм функціонування асинхронного гібридного класу контролера та компоненти представлення для застосування ASP.NET Core MVC.
19. Перерахуйте типи даних класів, які може повертати компонента представлення, як складова гібридного класу контролера та компоненти представлення для застосування ASP.NET Core MVC.
20. Перерахуйте складові проекту, для яких може бути застосований гібридний клас контролера та компоненти представлення для застосування ASP.NET Core MVC.

Лекція №11. Допоміжні функції дескрипторів

1. Створення допоміжної функції дескриптора
   1.1. Визначення класу допоміжної функції дескриптора
   1.2. Реєстрація допоміжних функцій дескрипторів
   1.3. Використання допоміжної функції дескриптора
   1.4. Управління областю дії допоміжної функції дескриптора
2. Удосконалені можливості допоміжних функцій дескрипторів
   2.1. Створення елементів для скорочення
   2.2. Вставка перед та після вмісту та елементів
   2.3. Отримання даних контексту представлення та використання впровадження залежностей
   2.4. Робота з моделлю представлення
   2.5. Узгодження допоміжних функцій дескрипторів
   2.6. Пригнічення вихідного елемента

Допоміжні функції дескрипторів – це засіб, що з'явився у ASP.NET Core MVC. Вони є класами, які трансформують HTML-елементи у представленні. Поширені випадки використання допоміжних функцій дескрипторів включають генерування URL для форм із застосуванням конфігурації маршрутизації застосування, забезпечення узгодженої стилізації елементів специфічного типу та заміна спеціальних елементів для скорочення, ходовими фрагментами вмісту. У даному розділі буде показано, як працюють допоміжні функції дескрипторів і яким чином створювати та використовувати спеціальні допоміжні функції дескрипторів.

1. Створення допоміжної функції дескриптора

Як і з багатьма засобами MVC, розумінню допоміжних функцій дескрипторів найкраще всього сприяє створення одного такого класу, що дозволить виявити особливості їхньої роботи та узгодження з рештою частин застосування. У наведених далі розділах буде продемонстровано процес створення та використання допоміжної функції дескриптора.

1.1. Визначення класу допоміжної функції дескриптора

Допоміжні функції дескрипторів можна визначати будь-де в проекті, але їх зручно тримати разом, оскільки на відміну від більшості компонентів MVC перед застосуванням вони потребують реєстрації.

Допоміжні функції дескрипторів є похідними від класу TagHelper, який визначено у просторі імен Microsoft.AspNetCore.Razor.TagHelpers.

У класі TagHelper визначено метод Process(), який перевизначається підкласами для реалізації поведінки, що трансформує елементи. Ім'я допоміжної функції дескриптора утворюється з імені елемента, що трансформується, і суфікса TagHelper. Область дії допоміжної функції дескриптора може бути розширена або звужена з використанням атрибутів, описаних у розділі "Управління областю дії допоміжної функції дескриптора" далі у розділі, але така його стандартна поведінка.

Отримання даних контексту

Допоміжні функції дескрипторів отримують інформацію про елемент, що трансформується через екземпляр класу TagHelperContext, який передається в якості аргументу методу Process().

Генерування виводу

Метод Process() трансформує елемент шляхом конфігурування об'єкта TagHelperOutput, який отримується у аргументі. Об'єкт TagHelperOuput починає своє існування з опису

HTML-елемента в тому вигляді, як він з'являється в представленні Razor, і модифікує його за допомогою властивостей та методу.

1.2. Реєстрація допоміжних функцій дескрипторів

Допоміжні функції дескрипторів можна застосовувати лише після того, як вони були зареєстровані з використанням Razor-виразу @addTagHelper. Набір представлень, до яких застосовуватиметься допоміжна функція дескриптора, залежить від того, де використовується вираз @addTagHelper. Для окремого представлення згаданий вираз може міститись у самому представленні. Для підмножини представлень у застосуванні вираз @addTagHelper знаходиться всередині файлу _ViewImports.cshtml в папці, яка містить представлення, або в батьківській папці.

1.3. Використання допоміжної функції дескриптора

Зрештою, допоміжну функцію дескриптора можна застосувати для трансформації елемента.

1.4. Управління областю дії допоміжної функції дескриптора

Допоміжні функції дескрипторів застосовуються до всіх елементів заданого типу. Це не завжди придатне.

Звуження області дії для допоміжної функції дескриптора
Вирішення проблеми полягає в тому, щоб дозволити допоміжній функції дескриптора визначати обмеження на її використання, звужуючи область дії, в якій вона застосовуватиметься. Обмеження допоміжної функції дескриптора вказуються за допомогою атрибуту HtmlTargetElement.

Розширення області дії для допоміжної функції дескриптора
Атрибут HtmlTargetElement може також використовуватися для розширення області дії допоміжної функції дескриптора, щоб вона охоплювала більш широкий діапазон елементів. Це корисно, коли одну і ту ж трансформацію необхідно виконувати для декількох типів елементів, що йде врозріз з вихідною умовою зіставлення елементів на основі імені допоміжної функції дескриптора.

2. Удосконалені можливості допоміжних функцій дескрипторів

У попередньому розділі було продемонстровано, яким чином створювати та використовувати базову допоміжну функцію дескриптора, але це лише мала частка доступних можливостей. У наступних розділах будуть показані складніші випадки застосування допоміжних функцій дескрипторів та засобів, які вони надають.

2.1. Створення елементів для скорочення

Допоміжні функції дескрипторів не обмежуються трансформуванням стандартних HTML-елементів і можуть також використовуватися для заміни спеціальних елементів ходовим вмістом. Дана можливість дозволяє зробити представлення більш короткими та прояснити їх задум.

2.2. Вставка перед та після вмісту та елементів

Клас TagHelperOutput пропонує чотири властивості, які полегшують впровадження нового вмісту в представлення, так що воно оточує елемент або його вміст.

Додавання вмісту навколо вихідного елемента

Перші дві властивості класу TagHelperOuput, PreElement та PostElement, застосовуються для вставки елементів у представлення перед та після вихідного елемента.

Вставка вмісту всередину вихідного елемента

Властивості PreContent і PostContent використовуються для вставки вмісту всередину вихідного елементу, оточуючи його вихідний вміст.

2.3. Отримання даних контексту представлення та використання впровадження залежностей

Одним із найпоширеніших застосувань класів допоміжних функцій дескрипторів, включаючи вбудовані класи, є трансформація елементів так, щоб вони містили деталі поточного запиту або поточної моделі представлення.

Для отримання даних контексту знадобиться додати властивість і декорувати її двома атрибутами.

Атрибут ViewContext вказує на те, що значення властивості повинно присвоюватися об'єкту ViewContext, коли створюється новий екземпляр класу. Клас ViewContext надає деталі про представлення, що візуалізується, про дані маршрутизації і про поточний HTTP-запит.

Атрибут HtmlAttributeNotBound запобігає присвоєнню інфраструктурою MVC значення властивості, якщо HTML-елемент має атрибут view-context. Це рекомендований прийом, особливо у випадку створення допоміжних функцій дескрипторів для використання іншими розробниками.

Допоміжні функції дескрипторів можуть оголошувати у своїх конструкторах залежності від служб, які розпізнаються з використанням засобу впровадження залежностей.

Наприклад, оголошено залежність від служби IUrlHelperFactory, яка дозволяє створювати вихідні URL з даних маршрутизації (і є службою, яка лежить в основі властивості Url, яка надається класом Controller). Всередині методу Process() допоміжна функція дескриптора застосовує метод IUrlHelperFactory.GetUrlHelper() для отримання об'єкта реалізації IUrlHelper, який конфігурується за допомогою об'єкта ViewContext і потім використовується при створенні URL для атрибуту action вихідного елемента.

2.4. Робота з моделлю представлення

Допоміжні функції дескрипторів можуть оперувати на моделі представлення, підганяючи виконувані ними трансформації або створюваний вивід.

Клас ModelExpression застосовується, коли потрібно оперувати з частиною моделі представлення.

Значення атрибуту helper-for – це властивість із класу Model, що виявляється інфраструктурою MVC та надається допоміжній функції дескриптора у вигляді об'єкта ModelExpression.

Клас ModelExpression тут докладно не розглядається, тому що будь-який аналіз типів призводить до нескінченних списків класів та властивостей. Крім того, в MVC є зручний набір вбудованих допоміжних функцій дескрипторів, що використовуються моделлю представлення для трансформації елементів, створювати власні класи такого роду не доведеться.

2.5. Узгодження допоміжних функцій дескрипторів

Властивість TagHelperContext.Items надає словник, який використовується для узгодження допоміжних функцій дескрипторів, що оперують на елементах та їх дочірніх елементах.

2.6. Пригнічення вихідного елемента

Допоміжні функції дескрипторів можуть використовуватися для того, щоб запобігти включенню елемента в HTML-розмітку, що надсилається браузеру, за рахунок виклику методу SuppressOuput() на об'єкті TagHelperOutput, який отримується в якості аргументу методу Process().

Контрольні питання

1. Поясніть призначення допоміжної функції дескриптора для застосування ASP.NET Core MVC.
2. Назвіть базовий клас і метод, який повинен бути перевизначений, для класу допоміжної функції дескриптора для застосування ASP.NET Core MVC.
3. Поясніть призначення класу TagHelperContext і перерахуйте кілька його членів для застосування ASP.NET Core MVC.
4. Поясніть призначення класу TagHelperOutput і перерахуйте кілька його членів для застосування ASP.NET Core MVC.
5. Поясніть способи реєстрації допоміжної функції дескриптора для застосування ASP.NET Core MVC.
6. Поясніть спосіб використання допоміжної функції дескриптора для застосування ASP.NET Core MVC.
7. Поясніть необхідність застосування звуження області дії для допоміжної функції дескриптора для застосування ASP.NET Core MVC.
8. Поясніть необхідність застосування розширення області дії для допоміжної функції дескриптора для застосування ASP.NET Core MVC.
9. Поясніть призначення класу допоміжної функції дескриптора, який створює дескриптор для скорочення для застосування ASP.NET Core MVC.
10. Поясніть призначення класу TagHelperContent і перерахуйте кілька його членів для застосування ASP.NET Core MVC.
11. Поясніть механізм додавання вмісту навколо цільового дескриптора, в результаті створення класу допоміжної функції дескриптора для застосування ASP.NET Core MVC.
12. Поясніть механізм додавання вмісту в цільовий дескриптор, в результаті створення класу допоміжної функції дескриптора для застосування ASP.NET Core MVC.
13. Поясніть необхідність використання впровадження залежностей для допоміжної функції дескриптора для застосування ASP.NET Core MVC.
14. Поясніть механізм використання впровадження залежностей для допоміжної функції дескриптора для застосування ASP.NET Core MVC.
15. Поясніть необхідність використання моделі представлення для допоміжної функції дескриптора для застосування ASP.NET Core MVC.
16. Поясніть призначення класу ModelExpression і перерахуйте кілька його членів для застосування ASP.NET Core MVC.
17. Поясніть необхідність реалізації узгодження допоміжних функцій дескриптора для застосування ASP.NET Core MVC.
18. Поясніть необхідність використання властивості TagHelperContext.Items для допоміжної функції дескриптора для застосування ASP.NET Core MVC.
19. Поясніть необхідність запобігання включення дескриптора за допомогою допоміжної функції дескриптора для застосування ASP.NET Core MVC.
20. Поясніть механізм функціонування запобігання включення дескриптора за допомогою допоміжної функції дескриптора для застосування ASP.NET Core MVC.

Лекція №12. Допоміжні функції дескрипторів для форм

1. Робота з елементами form
   1.1. Встановлення мети форми
   1.2. Використання засобу протидії підробці
2. Робота з елементами input
   2.1. Конфігурування елементів input
   2.2. Форматування значень даних
3. Робота з елементами label
4. Робота з елементами select та option
   4.1. Використання джерела даних для заповнення елемента select
   4.2. Генерування елементів option з перерахування
5. Робота з елементами textarea
6. Допоміжні функції дескрипторів для перевірки достовірності форм

Інфраструктура MVC пропонує набір вбудованих допоміжних функцій дескрипторів, які використовуються для виконання часто потрібних трансформацій у відношенні HTML-елементів. У даному розділі розглядаються допоміжні функції дескрипторів, які оперують на HTML-формах, включаючи елементи form, input, label, select, option та textarea.

1. Робота з елементами form

Клас FormTagHelper реалізує вбудовану допоміжну функцію дескриптора для елементів form, яка використовується при управлінні конфігурацією HTML-форм, так що їх можна націлювати на правильний метод дії, ґрунтуючись на конфігурації маршрутизації застосування.

1.1. Встановлення мети форми

Основним призначенням класу FormTagHelper є встановлення атрибуту action елемента form із застосуванням конфігурації маршрутизації застосування, гарантуючи тим самим, що ці дані форми завжди надсилаються коректному URL, навіть коли схема маршрутизації змінюється.

1.2. Використання засобу протидії підробці

Підробка міжсайтових запитів (cross-site request forgery – CSRF) – це спосіб зловмисної експлуатації веб-застосування, спрямований на те, щоб задіяти у своїх інтересах метод аутентифікації запитів користувача. У більшості веб-застосувань, включаючи створювані з використанням ASP.NET Core, ідентифікація запитів, що стосуються специфічного сеансу, з яким зазвичай асоційований користувач, здійснюється із застосуванням cookie-наборів.

Атака CSRF (також звана містифікацією сеансу) спирається на ситуацію, коли користувач відвідує зловмисний веб-сайт після роботи зі своїм веб-застосуванням, явно не закінчивши свої сеанси натисканням на кнопці виходу з застосування. Застосування, як і раніше, вважає сеанс користувача активним, а термін дії cookie-набору, який браузер зберіг, поки що не закінчився. Зловмисний веб-сайт містить код JavaScript, що надсилає застосуванню запит форми, який виконує операцію без згоди користувача; при цьому природа операції залежатиме від застосування, що піддається атаці. Оскільки код JavaScript виконується браузером користувача, запит до застосування включає cookie-набір сеансу і застосування виконує операцію без будь-якого повідомлення користувача або згоди з його боку.

Якщо елемент form не містить атрибут action (через те, що він генерується системою маршрутизації за допомогою атрибутів asp-controller і asp-acton), клас FormTagHelper автоматично включає засіб протистояння атакам CSRF. Зазначений засіб додає до форми прихований елемент input з маркером безпеки всередині HTML-розмітки, яка відправляється

клієнту поряд із cookie-набором. Застосування буде обробляти запит, тільки якщо він містить і cookie-набір, та приховане значення форми, до якого зловмисний веб-сайт не має доступу. Кожен запит до форми створює новий унікальний набір маркерів безпеки.

Скориставшись інструментами браузера, доступними за натисканням клавіші <F12>, можна також побачити відповідний cookie-набір, який додається до відповіді. Додавання маркерів безпеки до HTML-відповідей – лише частина процесу; вони повинні також перевірятись контролером.

Атрибут ValidateAntiForgeryToken гарантує, що запит містить допустимі маркери протидії атакам CSRF, і генеруватиме виняток, якщо вони відсутні або не мають очікуваного значення.

Клас FormTagHelper пропонує атрибут asp-antiforgery для перевизначення стандартної поведінки протидії атакам CSRF. При установці цього атрибуту в true маркери безпеки включатимуться в запити, навіть якщо елемент form має атрибут action. Коли значення атрибуту asp-antiforgery є false, маркери безпеки будуть відключені.

2. Робота з елементами input

Елемент input є основою HTML-форм та надає головні засоби, за допомогою яких користувач може забезпечувати застосування неструктурованими даними. Клас InputTagHelper використовується для трансформування елементів input, щоб вони відображали тип даних та формат властивості моделі представлення, що застосовується для збору інформації, з використанням атрибутів.

2.1. Конфігурування елементів input

Атрибут asp-for встановлюється в ім'я властивості моделі представлення, яке потім використовується для встановлення атрибутів name, id, type та value елемента input.

Атрибут type елемента input повідомляє браузеру про те, як відображати елемент у формі.

Типи float, double та decimal виробляють елементи input з атрибутом type, встановленим у text, оскільки не всі браузери дозволяють використовувати повний діапазон символів для вираження допустимих значень зазначених типів. Щоб сприяти користувачеві, допоміжна функція дескриптора додає до елемента input атрибути, які застосовуються із засобом перевірки достовірності моделі.

2.2. Форматування значень даних

Коли метод дії забезпечує представлення об'єктом моделі представлення, допоміжна функція дескриптора застосовує значення властивості, вказане в атрибуті asp-for, для встановлення атрибута value елемента input. Атрибут asp-format використовується для вказівки, яким чином форматувати це значення даних.

Атрибут asp-format набуває значення, яке буде передано стандартній системі форматування строк C#.

Застосування форматування через клас моделі

Якщо необхідно завжди використовувати для властивості моделі одне і те ж форматування, тоді можна декорувати клас C# атрибутом DisplayFormat, який визначений у просторі імен System.ComponentModel.DataAnnotations. Для форматування значення даних атрибут DisplayFormat потребує двох аргументів: аргумент DataFormatString вказує строчку формату, а аргумент ApplyFormatInEditMode – що форматування повинно застосовуватись при редагуванні значень.

3. Робота з елементами label

Елемент label трансформується класом LabelTagHelper, який використовує клас моделі представлення, щоб переконатися в тому, що мітки не містять помилок і є узгодженими. Існує лише один підтримуваний атрибут.

Допоміжна функція дескриптора буде використовувати ім'я властивості моделі представлення для встановлення значення атрибута for та вмісту елемента label.

4. Робота з елементами select та option

Елементи select та option застосовуються для надання користувачеві фіксованого набору варіантів замість відкритого запису даних, який можливий за допомогою елемента input. Клас SelectTagHelper відповідає за трансформування елементів select та підтримує атрибути.

Атрибут asp-for встановлює значення атрибутів for та id.

4.1. Використання джерела даних для заповнення елемента select

Явне визначення елементів option для елемента select – зручний підхід для варіантів вибору, які завжди мають одні й ті ж значення. Однак він нічим не допоможе в ситуації, коли необхідно надати варіанти, що беруться з даних моделі, або коли потрібен однаковий набір варіантів у кількох представленнях, але небажано підтримувати вручну дубльований вміст.

4.2. Генерування елементів option з перерахування

Якщо є фіксований набір варіантів для надання користувачеві, але важливо не дублювати його в представленнях скрізь у застосуванні, тоді можна застосувати перерахування.

У випадку використання перерахування найкращий спосіб генерації елементів option передбачає забезпечення атрибута asp-items об'єктом SelectList, який заповнений іменами значень перерахування. "За лаштунками" клас SelectTagHelper генерує елементи option із колекції IEnumerable<SelectListItem>, а клас SelectList реалізує цей інтерфейс.

Генерування елементів option із моделі

Якщо необхідно генерувати елементи option для відображення даних у моделі, то найпростіший підхід полягає у наданні даних, необхідних для генерації елементів, за допомогою об'єкта ViewBag.

Використання спеціальної допоміжної функції дескриптора для генерації елементів option із моделі

Проблема з передачею даних для елементів option через об'єкт ViewBag пов'язана з тим, що потрібно не забути генерувати дані в кожному методі дії, який візуалізує представлення, де застосовується допоміжна функція дескриптора. Це призводить до дублювання коду, і ускладнює тестування та супровід контролера. Ефективніший підхід передбачає створення спеціальної допоміжної функції дескриптора, яка доповнює вбудований клас SelectTagHelper.

5. Робота з елементами textarea

Елемент textarea застосовується для запиту у користувача великого обсягу тексту і зазвичай використовується для неструктурованих даних, таких як коментарі або зауваження. Клас TextAreaTagHelper відповідає за трансформування елементів textarea.

Клас TextAreaTagHelper відносно простий; значення, що надається в атрибуті asp-for, використовується для встановлення атрибутів id та name елемента textarea.

6. Допоміжні функції дескрипторів для перевірки достовірності форм

Є ще дві допоміжні функції дескрипторів, які мають відношення до HTML-форм. Ці допоміжні класи використовуються для надання користувачеві відгуку, коли введені ним дані не задовольняють очікування застосування.

Контрольні питання

1. Поясніть призначення допоміжних функцій дескрипторів форм для застосування ASP.NET Core MVC.
2. Назвіть клас, який реалізує допоміжні функції дескриптора form для застосування ASP.NET Core MVC.
3. Поясніть переваги використання допоміжних функцій дескриптора form asp-action та asp-controller для застосування ASP.NET Core MVC.
4. Назвіть механізм, за допомогою якого організовано ідентифікацію запитів, що відносяться до специфічного сеансу, з яким зазвичай асоційований користувач для застосування ASP.NET Core MVC.
5. Поясніть механізм функціонування атаки підробки міжсайтових запитів для застосування ASP.NET Core MVC.
6. Поясніть необхідність використання атрибуту asp-for для дескриптора input для застосування ASP.NET Core MVC.
7. Перерахуйте атрибути для HTML-розмітки, які генеруються в результаті використання представлення, що містить форму з дескриптором input та атрибутом asp-for для застосування ASP.NET Core MVC.
8. Поясніть відповідність між типами даних C# та значеннями атрибуту type дескриптора input для застосування ASP.NET Core MVC.
9. Поясніть необхідність використання атрибуту UIHint для моделі представлення для застосування ASP.NET Core MVC.
10. Поясніть необхідність використання атрибуту asp-format для дескриптора input для застосування ASP.NET Core MVC.
11. Поясніть перевагу використання атрибуту DisplayFormat для властивості моделі представлення для застосування ASP.NET Core MVC.
12. Поясніть необхідність використання атрибуту asp-for для дескриптора label для застосування ASP.NET Core MVC.
13. Перерахуйте атрибути для HTML-розмітки, які генеруються в результаті використання представлення, що містить форму з дескриптором label та атрибутом asp-for для застосування ASP.NET Core MVC.
14. Назвіть клас, який реалізує допоміжні функції дескриптора select для застосування ASP.NET Core MVC.
15. Перерахуйте атрибути для розмітки HTML, які генеруються в результаті використання представлення, що містить форму з дескриптором select і атрибутом asp-for для застосування ASP.NET Core MVC.
16. Перерахуйте можливі джерела даних, які використовуються при генерації дескрипторів option для дескриптора select для застосування ASP.NET Core MVC.
17. Поясніть, можливі проблеми, що виникають при використанні ViewBag в якості джерела даних при генерації дескрипторів option для дескриптора select для застосування ASP.NET Core MVC.
18. Назвіть клас, який реалізує допоміжні функції дескриптора textarea для застосування ASP.NET Core MVC.
19. Перерахуйте атрибути для розмітки HTML, які генеруються в результаті використання представлення, що містить форму з дескриптором textarea і атрибутом asp-for для застосування ASP.NET Core MVC.
20. Поясніть необхідність використання допоміжних функцій дескрипторів для перевірки достовірності форм для застосування ASP.NET Core MVC.

Лекція №13. Спеціалізовані допоміжні функції дескрипторів

1. Використання допоміжної функції дескриптора для середовища розміщення
2. Використання допоміжних функцій дескрипторів для JavaScript та CSS
   2.1. Управління файлами JavaScript
   2.2. Управління таблицями стилів CSS
3. Робота з якірними елементами
4. Робота з елементами img
5. Використання кешу даних
   5.1. Встановлення часу закінчення для кешу
   5.2. Використання варіацій кешу
6. Використання URL, відносних до застосування

Допоміжні функції дескрипторів, які були описані в попередньому розділі, зосереджені на випуску HTML-форм, але вони не є єдиними вбудованими допоміжними функціями дескрипторів, пропонованими інфраструктурою ASP.NET Core MVC. У даному розділі розглядаються допоміжні функції дескрипторів, які управляють файлами JavaScript і таблицями стилів CSS, створюють URL для якірних елементів, надають можливість анулювання кешу для елементів зображень і підтримують кешування даних. Додатково буде описана допоміжна функція дескриптора, що забезпечує доступ для URL, відносних до застосування, які допомагають гарантувати, що браузери можуть звертатися до статичного вмісту, коли застосування розгорнуто в середовищі, що розділяється з іншими застосуваннями.

1. Використання допоміжної функції дескриптора для середовища розміщення

Клас EnvironmentTagHelper застосовується до спеціального елемента environment і визначає, чи включається область вмісту в HTML-розмітку, що надсилається браузеру, на основі середовища розміщення.

2. Використання допоміжних функцій дескрипторів для JavaScript та CSS

Наступна категорія вбудованих допоміжних функцій дескрипторів застосовується для управління файлами JavaScript і таблицями стилів CSS за допомогою елементів script і link, які зазвичай включаються в компоновку, що розділяється.

2.1. Управління файлами JavaScript

Клас ScriptTagHelper – це вбудована допоміжна функція дескриптора для елементів script. Клас ScriptTagHelper призначений для управління включенням файлів JavaScript у представлення.

Вибір файлів JavaScript

Атрибут asp-src-include застосовується для включення файлів JavaScript у представлення з використанням шаблонів універсалізації імен, які підтримують набір групових символів, що застосовуються для зіставлення з іменами файлів.

Звуження шаблону універсалізації імен

Багато пакетів надають звичайні та мініфіковані версії своїх файлів JavaScript, і якщо необхідно використовувати лише мініфіковані версії, тоді можна обмежити набір файлів, які відповідатимуть шаблону універсалізації імен. Це хороший підхід, коли не планується робота над налагодженням бібліотеки jQuery, яка реалізована ефективно і не викликає особливих проблем, або ж відомо, що цільові браузери підтримують відображення на вихідний код.

Вилучення файлів

Звуження шаблону для файлів JavaScript допомагає, коли потрібно вибрати файл, ім'я якого містить специфічний елемент типу slim. Воно не корисно, якщо ім'я файлу не має такого елемента, як у ситуації, коли необхідна повна версія мініфікованого файлу. На щастя, за допомогою атрибуту asp-src-exclude можна видаляти файли зі списку, побудованого атрибутом asp-src-include.

Використання середовища розміщення для вибору файлів

Загальний підхід передбачає застосування звичайних файлів JavaScript на етапі розробки, що полегшує налагодження, та використання мініфікованих файлів у виробничому середовищі для скорочення витрати смуги пропускання. У такому випадку можна задіяти елемент environment, щоб вибірково включати елементи script, ґрунтуючись на середовищі розміщення.

Анулювання кешу

З кешуванням пов'язана одна проблема: клієнти не отримують нові версії статичних файлів негайно після їх розгортання, тому що запити клієнтів продовжують обслуговуватися із застосуванням вмісту, що раніше кешувався.

Згадана проблема вирішується за допомогою анулювання кешу. Ідея полягає в тому, щоб дозволити кешам обробляти статичний вміст, але негайно відображати будь-які зміни, що відбулися на сервері.

Робота з мережами доставки вмісту

Мережа доставки вмісту застосовується для передачі запитів до вмісту застосування серверам, що знаходяться поблизу від користувача. Замість запиту файлу JavaScript у серверів браузер запитує його у імені хоста, яке перетворюється на географічно місцевий сервер, що зменшує час завантаження файлів і скорочує ширину смуги пропускання, необхідну застосуванню.

2.2. Управління таблицями стилів CSS

Клас LinkTagHelper – це вбудована допоміжна функція дескриптора для елементів link, яка застосовується при управління включенням таблиць стилів CSS у представлення.

Вибір таблиць стилів

Клас LinkTagHelper поділяє з класом ScriptTagHelper багато засобів, у тому числі підтримку шаблонів універсалізації імен для вибору або виключення файлів CSS, так що їх не потрібно вказувати окремо. Наявність можливості точного вибору файлів CSS має таку ж важливість, як і для файлів JavaScript, оскільки таблиці стилів теж надходять у вигляді звичайних і мініфікованих версій, додатково підтримуючи відображення на вихідний код.

Робота з мережами доставки вмісту

Допоміжний клас LinkTag пропонує набір атрибутів для повернення до локального вмісту, коли мережа CDN не доступна, хоча процес перевірки, чи завантажилася таблиця стилів, трохи складніше, ніж подібна перевірка у відношенні файлу JavaScript.

3. Робота з якірними елементами

Елемент a є базовим інструментом для навігації в рамках застосування та надсилання запитів GET застосуванню з метою отримання різноманітного вмісту. Клас AnchorTagHelper використовується для трансформації атрибуту href елементів a, щоб цільові URL генерувалися за участю системи маршрутизації.

Клас AnchorTagHelper простий та передбачуваний; він полегшує генерацію URL в елементах a, які використовують конфігурацію маршрутизації застосування.

4. Робота з елементами img

Клас ImageTagHelper пропонує засіб анулювання кешу для зображень через атрибут src елементів img, в результаті застосування отримує можливість застосовувати у своїх інтересах кешування та гарантувати негайне відображення будь-яких змін у зображеннях. Клас ImageTagHelper оперує з елементами img, які визначають атрибут asp-append-version.

Як і з засобами анулювання кешу для файлів JavaScript та таблиць стилів CSS, включена до URL контрольна сума залишається незмінною до тих пір, поки файл не буде модифікований.

5. Використання кешу даних

Інфраструктура MVC підтримує в пам'яті кеш, який може застосовуватися для кешування фрагментів вмісту з метою прискорення візуалізації представлень. Вміст, що підлягає кешуванню, позначається у файлі представлення з використанням елемента cache, який обробляється класом CacheTagHelper.

Елемент cache застосовується для оточення вмісту, яке має бути додано у кеш.

Застосування елемента cache без будь-яких атрибутів повідомляє інфраструктуру MVC про те, що для задоволення всіх майбутніх запитів необхідно повторно використовувати зазначений вміст. Тепер після запуску застосування вміст, згенерований компонентом представлення, кешується, тому навіть при перезавантаженні сторінки буде відображатися те ж саме показання часу.

5.1. Встановлення часу закінчення для кешу

Атрибути expires-* дозволяють вказувати час, коли закінчується термін існування кешованого вмісту, який виражається як абсолютний час, час відносно поточного часу або проміжок часу, протягом якого кешований вміст не запитується. За допомогою атрибуту expires-after вказується, що вміст повинен кешуватися протягом визначеного проміжку часу.

В результаті запуску застосування виявиться, що термін існування кешованих даних закінчується через зазначений проміжок часу, після чого перезавантаження сторінки призводить до виклику компонента представлення та кешування нового вмісту протягом наступного проміжку часу.

Встановлення фіксованого моменту закінчення

За допомогою атрибуту expires-on, що приймає значення DateTime, можна вказувати фіксований момент часу, при настанні якого закінчиться термін існування кешованого вмісту.

Встановлення періоду закінчення з моменту останнього використання

Атрибут expires-sliding застосовується для вказівки проміжку часу, через який термін існування вмісту спливає, якщо він не вилучався з кеша.

5.2. Використання варіацій кешу

За замовчуванням всі запити отримують один і той же кешований вміст. Клас CacheTagHelper здатний підтримувати різні версії кешованого вмісту та застосовувати їх для задоволення різних типів HTTP-запитів, що вказуються з використанням одного з атрибутів, імена яких починаються з vary-by.

6. Використання URL, відносних до застосування

Остання вбудована допоміжна функція дескриптора – клас UrlResolutionTagHelper – застосовується для забезпечення підтримки URL, відносних до застосування, які представляють собою URL, забезпечені префіксом у вигляді тильди (символу ~).

Контрольні питання

1. Поясніть призначення допоміжної функції дескриптора для середовища розміщення для застосування ASP.NET Core MVC.
2. Назвіть клас, який реалізує допоміжні функції дескриптора для середовища розміщення для застосування ASP.NET Core MVC.
3. Назвіть клас, який реалізує допоміжні функції дескриптора для управління включенням файлів JavaScript у представлення для застосування ASP.NET Core MVC.
4. Поясніть необхідність використання в компоновці атрибуту допоміжної функції дескриптора включення файлів JavaScript у представлення, з використанням шаблону універсалізації імен для застосування ASP.NET Core MVC.
5. Поясніть необхідність використання в компоновці атрибуту допоміжної функції дескриптора виключення файлів JavaScript у представленні, з використанням шаблону універсалізації імен для застосування ASP.NET Core MVC.
6. Поясніть перевагу використання мережі доставки вмісту для застосування ASP.NET Core MVC.
7. Назвіть клас, який реалізує допоміжні функції дескриптора для управління включенням файлів CSS у представлення для застосування ASP.NET Core MVC.
8. Поясніть необхідність використання в компоновці атрибуту допоміжної функції дескриптора включення файлів CSS у представлення, з використанням шаблону універсалізації імен для застосування ASP.NET Core MVC.
9. Поясніть необхідність використання в компоновці атрибуту допоміжної функції дескриптора виключення файлів CSS у представленні, з використанням шаблону універсалізації імен для застосування ASP.NET Core MVC.
10. Поясніть недоліки використання мережі доставки вмісту для застосування ASP.NET Core MVC.
11. Назвіть клас, який реалізує допоміжні функції дескриптора для навігації для застосування ASP.NET Core MVC.
12. Назвіть клас, який реалізує допоміжні функції дескриптора для зображення для застосування ASP.NET Core MVC.
13. Поясніть необхідність використання кешування фрагментів вмісту для застосування ASP.NET Core MVC.
14. Назвіть клас, який реалізує допоміжні функції дескриптора для управління кешуванням фрагментів вмісту для застосування ASP.NET Core MVC.
15. Поясніть необхідність використання анулювання кешування фрагментів вмісту для застосування ASP.NET Core MVC.
16. Поясніть механізм функціонування анулювання кешування фрагментів вмісту для застосування ASP.NET Core MVC.
17. Поясніть необхідність встановлення періоду закінчення часу для кешу вмісту представлення для застосування ASP.NET Core MVC.
18. Поясніть необхідність встановлення фіксованого часу для кешу вмісту представлення для застосування ASP.NET Core MVC.
19. Поясніть необхідність встановлення періоду закінчення часу з моменту останнього використання для кешу вмісту представлення для застосування ASP.NET Core MVC.
20. Поясніть необхідність встановлення періоду закінчення часу для кешу вмісту представлення для кожного запиту для застосування ASP.NET Core MVC.

Лекція №14. Прив'язка моделей

1. Поняття прив'язки моделей
   1.1. Стандартні значення прив'язки
   1.2. Прив'язка простих типів
   1.3. Прив'язка складних типів
   1.4. Прив'язка масивів та колекцій
   1.5. Прив'язка колекцій складних типів
2. Вказівка джерела даних прив'язки моделей
   2.1. Вибір стандартного джерела даних прив'язки
   2.2. Використання заголовків в якості джерела даних прив'язки
   2.3. Використання тіл запитів в якості джерела даних прив'язки

Прив'язка моделей – це процес створення об'єктів .NET з використанням даних із HTTP-запиту для постачання методів дій аргументами, в яких вони мають потребу. У даному розділі буде описано роботу системи прив'язки моделей, пояснено, як вона прив'язує прості типи, складні типи та колекції, а також продемонстровано, яким чином взяти на себе контроль над процесом, щоб вказувати частину запиту, яка надає значення даних, які потрібні методам дій.

1. Поняття прив'язки моделей

Прив'язка моделей – елегантний міст між HTTP-запитом та методами дій C#. На прив'язку моделей в тій чи іншій мірі покладається більшість застосувань MVC Framework.

Система прив'язки моделей спирається на зв'язувачі моделей, які є компонентами, відповідальними за надання значень даних із однієї частини запиту або застосування. Стандартні зв'язувачі моделей шукають значення даних у наступних трьох розташуваннях:

1. надіслані дані форми;
2. змінні маршрутизації;
3. строчки запитів.

Кожне джерело даних перевіряється по порядку до тих пір, поки не буде виявлено значення для аргументу.

1.1. Стандартні значення прив'язки

Прив'язка моделей є негарантованим засобом, оскільки MVC буде застосовувати його у спробі отримати значення, необхідні для виклику методу дії, але все одно викличе метод, навіть якщо значення даних надати не вдалося. Це може призвести до непередбачуваної поведінки.

1.2. Прив'язка простих типів

Коли доступне потрібне значення, воно перетворюється на значення типу C#, яке може використовуватися при виклику методу дії. Прості типи – це значення, що походять з одного елемента даних у запиті, який можна розібрати з строчки. До їх складу входять числові значення, булеві значення, дати і, звичайно же, значення string.

Якщо значення із запиту не може бути перетворене (наприклад, для параметра, що вимагає значення int, вказано значення apple), тоді процес прив'язки моделей виявиться нездатним надати значення застосуванню, і буде застосовуватись стандартне значення.

У результаті виникає проблема, оскільки метод дії отримуватиме стандартне значення 0 у двох ситуаціях. Перша з них – коли запит містить значення, яке не може бути перетворено на тип аргументу, як у URL виду /Home/Index/Apple. Друга ситуація – коли запит містить значення, яке піддається перетворенню, але дорівнює 0, як у URL виду /Home/Index/0.

Більшості застосувань потрібна можливість розрізнення таких ситуацій, і легше всього це робиться з використанням для аргументу методу дії типу, що допускає null.

1.3. Прив'язка складних типів

Коли параметр методу дії має складний тип (іншими словами, будь-який тип, який не може бути розібраний з одиночного строкового значення), процес прив'язки моделей застосовує рефлексію для отримання набору відкритих властивостей цільового типу і виконує процес прив'язки у відношення кожного з них по черзі.

Створення легко прив'язуваної HTML-розмітки

Використання префіксів означає, що представлення повинні включати інформацію, яка цікавить зв'язувач моделі. Це легко робиться із застосуванням допоміжних функцій дескрипторів, які автоматично додають необхідні префікси до елементів, що трансформуються.

Коли використовується допоміжна функція дескриптора, вкладене ім'я властивості вказується із застосуванням угод C#, так що зовнішні та вкладені імена властивостей відокремлюються точками.

Вказівка спеціальних префіксів

Трапляються ситуації, коли генерована HTML-розмітка відноситься до одного типу об'єкта, але її необхідно прив'язати до іншого типу. Це означає, що префікси, що містяться в представленні, не будуть відповідати структурі, на яку очікує зв'язувач моделі, і дані не зможуть бути правильно оброблені.

Виборча прив'язка властивостей

Припустимо, що властивість деякого класу є критично важливою і користувач не повинен мати можливість вказувати для нього значення. Перше, що можна зробити – перешкодити перегляду та редагуванню властивості користувачем, упевнившись у відсутності всередині представлень застосування будь-яких HTML-елементів, які посилаються на дану властивість. Тим не менш, зловмисник може просто відредагувати дані форми, що посилаються серверу, під час відправки форми і вказати потрібне йому значення для властивості. Насправді зв'язувачу моделі необхідно повідомити про те, щоб він не прив'язував властивість до значення із запиту. Для цього можна конфігурувати атрибут Bind на параметрі методу дії, вказавши імена лише тих властивостей, які повинні прив'язуватись.

Перший аргумент атрибуту Bind являє собою список імен властивостей, що розділяються комами, які повинні бути включені в процес прив'язки моделей.

Коли атрибут Bind застосовується до параметра методу дії, він впливає лише на екземпляри даного класу, які прив'язуються для цього методу дії; решта методів дій продовжать спроби прив'язати всі властивості, визначені типом параметра. Якщо потрібно отримати більший ефект, тоді атрибут Bind можна застосувати до самого класу моделі.

Властивості можна також явно виключати, декоруючи їх атрибутом BindNever, але тоді нові властивості, що додаються до класу моделі, включатимуться в процес прив'язки моделей, тільки якщо не забувати про застосування до них згаданого атрибуту.

1.4. Прив'язка масивів та колекцій

Процес прив'язки моделей має низку елегантних можливостей для прив'язки даних запитів до масивів та колекцій, які будуть описані в наступних розділах.

Прив'язка масивів

Однією з елегантних можливостей стандартного зв'язувача моделі є його підтримка для методів дій параметрів, що представляють собою масиви.

Коли форма відправляється, процес прив'язки моделей виявляє, що цільовий метод дії приймає масив, і виконує пошук елементів даних, які мають таке ж ім'я, як у параметра методу дії. Це означа, що всі значення з елементів input з атрибутом name, встановленим в names, збираються разом для створення масиву, який буде використовуватися в якості аргументу при виклику методу дії.

Прив'язка колекцій

Процес прив'язки моделей здатний створювати не лише масиви. Він також підтримує класи колекцій.

1.5. Прив'язка колекцій складних типів

Індивідуальні значення даних можна також прив'язувати до масиву складних типів, що дозволяє збирати з єдиного запиту множину об'єктів.

2. Вказівка джерела даних прив'язки моделей

Як пояснювалося на початку, стандартний процес прив'язки моделей шукає значення у трьох місцях: значення даних форми, змінні маршрутизації та строчки запитів.

Стандартна послідовність пошуку не завжди корисна або тому, що дані повинні завжди надходити зі специфічної частини запиту, або тому, що потрібно використовувати джерело даних, в якому пошук за замовчуванням не проводиться. Засіб прив'язки моделей включає набір атрибутів, які застосовуються для перевизначення стандартної поведінки пошуку.

2.1. Вибір стандартного джерела даних прив'язки

Атрибути FromForm, FromRoute та FromQuery дозволяють вказувати, що дані прив'язки моделі будуть отримуватись зі стандартного розташування, але без нормальної послідовності пошуку.

2.2. Використання заголовків в якості джерела даних прив'язки

Атрибут FromHeader дозволяє використовувати заголовки HTTP-запиту як джерела даних прив'язки.

Не всі імена HTTP-заголовків можуть легко вибиратися, покладаючись на ім'я параметра методу дії, тому що система прив'язки моделей не перетворює угоди за іменуванням C# у згоди, прийняті в HTTP-заголовках. У таких ситуаціях конфігурувати атрибут FromHeader необхідно із застосуванням властивості Name з метою вказівки імені заголовка.

Прив'язка складних типів із заголовків

Незважаючи на те, що подібне потрібно рідко, прив'язку складних типів можна виконувати, використовуючи значення заголовків, за рахунок застосування атрибуту FromHeader до властивостей класу моделі.

Процес прив'язки моделей досліджуватиме властивості складних типів у пошуку атрибутів, описаних у таблиці. Це дає можливість за допомогою атрибуту FromHeader визначити складний тип, властивості якого є моделлю, що прив'язується із заголовків.

2.3. Використання тіл запитів в якості джерела даних прив'язки

Не всі дані, що надсилаються клієнтами, надсилаються у вигляді даних форми, як відбувається в ситуації, коли клієнт JavaScript відправляє дані JSON контролеру API. Атрибут

FromBody вказує, що тіло запиту має бути декодовано і застосовуватися в якості джерела даних прив'язки моделі.

Атрибут FromBody можна застосовувати для прив'язки лише одного параметра методу дії, і у разі використання цього атрибуту більше одного разу в окремо взятому методі генерується виняток. Якщо потрібно отримати кілька об'єктів моделі з тіла запиту, тоді доведеться створити простий клас передачі даних, який має всі необхідні властивості і застосовує дані, що в ньому містяться, для створення об'єктів, потрібних усередині методу дії.

Контрольні питання

1. Дайте визначення поняттю прив'язка моделей для застосування ASP.NET Core MVC.
2. Дайте визначення поняттю зв'язувачі моделей для застосування ASP.NET Core MVC.
3. Поясніть спосіб призначення значення моделі прив'язкою моделей для методу дії, у випадку, якщо користувач не заповнив дані форми, не передав сегменти строчки запиту, у шаблоні маршрутизації не вказані значення за замовчуванням для застосування ASP.NET Core MVC.
4. Поясніть наслідки відсутності можливості отримання значення моделі прив'язкою моделей для методу дії для застосування ASP.NET Core MVC.
5. Перерахуйте прості типи даних, які можуть бути використані для моделей прив'язкою моделей для застосування ASP.NET Core MVC.
6. Поясніть механізм функціонування прив'язки моделей для моделей простих типів даних для застосування ASP.NET Core MVC.
7. Перерахуйте складні типи даних, які можуть бути використані для моделей прив'язкою моделей для застосування ASP.NET Core MVC.
8. Поясніть механізм функціонування прив'язки моделей для складних типів даних для застосування ASP.NET Core MVC.
9. Поясніть механізм функціонування прив'язки моделей для типів даних масивів для застосування ASP.NET Core MVC.
10. Поясніть механізм функціонування прив'язки моделей для типів даних колекцій для застосування ASP.NET Core MVC.
11. Поясніть механізм функціонування прив'язки моделей для моделей типів даних колекцій складних типів для застосування ASP.NET Core MVC.
12. Поясніть механізм візуалізації представлення, в якості моделі якого визначається колекція складних типів даних для застосування ASP.NET Core MVC.
13. Поясніть необхідність використання атрибутів для джерел даних прив'язки моделей для застосування ASP.NET Core MVC.
14. Поясніть необхідність зміни стандартного джерела даних прив'язки моделей для застосування ASP.NET Core MVC.
15. Поясніть спосіб зміни стандартного джерела даних прив'язки моделей для застосування ASP.NET Core MVC.
16. Поясніть спосіб зміни порядку стандартного джерела даних прив'язки моделей для застосування ASP.NET Core MVC.
17. Поясніть спосіб звернення до конкретного імені заголовка HTTP-запиту, який використовується в якості джерела даних прив'язки моделей для застосування ASP.NET Core MVC.
18. Поясніть переваги використання атрибутів для властивостей моделі, які вказують в якості джерела даних імена заголовка HTTP-запиту для прив'язки моделей для застосування ASP.NET Core MVC.
19. Поясніть необхідність використання тіла HTTP-запиту в якості джерела даних прив'язки моделей для застосування ASP.NET Core MVC.
20. Поясніть спосіб використання атрибуту FromBody для кількох параметрів методу дії, в якості джерела даних прив'язки моделей для застосування ASP.NET Core MVC.

Лекція №15. Перевірка достовірності моделей

1. Необхідність у перевірці достовірності моделі
2. Явна перевірка достовірності моделі
   2.1. Відображення користувачу помилок перевірки достовірності
   2.2. Відображення повідомлень про помилки перевірки достовірності
   2.3. Відображення повідомлень про помилки перевірки достовірності на рівні властивостей
   2.4. Відображення повідомлень про помилки перевірки достовірності на рівні моделі
3. Вказівка правил перевірки достовірності за допомогою метаданих
   3.1. Створення спеціального атрибута перевірки достовірності для властивості
4. Виконання перевірки достовірності на стороні клієнта
5. Виконання віддаленої перевірки достовірності

У попередньому розділі було показано, як інфраструктура MVC створює об'єкти моделей із HTTP-запитів за допомогою процесу прив'язки моделей. У ній повсюдно передбачалося, що дані, що надаються користувачем, були допустимими. Реальність така, що користувачі часто вводять дані, які є не допустимими і не можуть використовуватися, а тому темою даного розділу буде перевірка достовірності моделей.

Перевірка достовірності моделей представляє собою процес, який забезпечує придатність даних, одержуваних застосуванням, для прив'язки моделей. Якщо дані не підходять, то користувачам надається корисна інформація, яка допомагає усунути проблему.

Перша частина процесу – перевірка отриманих даних – є одним із основних способів оберігати цілісність моделі предметної області. Відхилення даних, які не мають сенсу в контексті предметної галузі, може запобігти виникненню дивних та небажаних станів у застосуванні. Друга частина – допомога користувачам у коригуванні проблеми – важлива у рівному ступені. Без такої інформації та зворотного зв'язку, необхідної для взаємодії з застосуванням, користувачі будуть незадоволені та спантеличені. У загальнодоступних застосуваннях це означає, що користувачі просто припинять користуватися ними. У корпоративних застосуваннях це означає порушення робочого потоку користувачів. Ні той, ні інший результат не є бажаним, але, на щастя, інфраструктура MVC надає широку підтримку перевірки достовірності моделей.

1. Необхідність у перевірці достовірності моделі

Перевірка достовірності моделі – це процес примусового застосування вимог, які застосування накладає на дані, отримані від клієнтів. Без перевірки достовірності застосування намагатиметься оперувати з будь-якими отриманими даними, що може призвести до генерації винятків і непередбаченої поведінки, яка відобразить негайні або довготривалі проблеми, які поступово накопичуються в міру того, як сховище наповнюється неправильними, незавершеними або зловмисними даними.

2. Явна перевірка достовірності моделі

Найпряміший спосіб перевірки достовірності моделі передбачає її виконання у методі дії.

У коді перевіряються значення, які зв'язувач моделі надав властивостям об'єкта параметра. Будь-які виявлені помилки реєструються із застосуванням об'єкта ModelStateDictionary, який повертається властивістю ModelState, успадкованою з базового класу Controller.

Як підказує його ім'я, клас ModelStateDictionary – це словник, який використовується для відстеження деталей стану об'єкта моделі з акцентом на помилках перевірки достовірності.

Система прив'язки моделей також застосовує об'єкт ModelStateDictionary для реєстрації будь-яких проблем із знаходженням та присвоєнням значень властивостям моделі. Метод

GetValidationState() використовується з метою з'ясування, чи зареєстровані для властивості моделі якісь помилки процесом прив'язки моделей або викликом AddModelError() під час явної перевірки достовірності у методі дії. Метод GetValidationState() повертає значення перерахування ModelValidationState.

2.1. Відображення користувачу помилок перевірки достовірності

Мати справу з помилкою перевірки достовірності, викликаючи метод View(), може здатися дивним підходом, але дані контексту, якими інфраструктура MVC забезпечує представлення, містять деталі помилок, що виникли при перевірці достовірності моделі. Такі деталі автоматично виявляються та застосовуються допоміжною функцією дескриптора, яка використовується для трансформації елементів input.

Допоміжна функція дескриптора додає елементи, чиї значення не пройшли перевірку достовірності, до класу input-validation-error, який потім можна стилізувати з метою виділення проблеми.

Робити це можна за рахунок визначення спеціальних стилів CSS у таблиці стилів, але якщо потрібно задіяти вбудовані стилі для перевірки, що надаються CSS бібліотеками на кшталт Bootstrap, то доведеться виконати невелику додаткову роботу. Оскільки ім'я класу, яке додається до елементів форми, не можна змінювати, знадобиться код JavaScript для порівняння імені, застосованого інфраструктурою MVC, і класами помилок CSS, запропонованими бібліотекою Bootstrap.

2.2. Відображення повідомлень про помилки перевірки достовірності

Класи CSS, які допоміжні функції дескрипторів застосовують до елементів input, вказують на наявність проблеми з полем форми, але не повідомляють користувачеві, у чому конкретно полягає проблема. Надання користувачеві додаткової інформації вимагає використання іншої допоміжної функції дескриптора, яка додає до представлення звіт з проблем.

Клас ValidationSummaryTagHelper виявляє атрибут asp-validation-summary в елементах div і реагує додаванням повідомлень, які описують будь-які помилки перевірки достовірності, виявлені методом дії. В атрибуті asp-validation-summary вказується одне із значень перерахування ValidationSummary.

2.3. Відображення повідомлень про помилки перевірки достовірності на рівні властивостей

Незважаючи на те, що спеціальне повідомлення про помилку виразніше, ніж стандартне повідомлення, користь від нього, як і раніше, невелика, воно не вказує ясно користувачеві на проблему. Для помилок такого роду відображати повідомлення про помилки перевірки достовірності більш практично поруч з HTML-елементами, які містять проблемні дані, що можна робити із застосуванням класу допоміжної функції дескриптора ValidationMessageTag. Він шукає елементи span з атрибутом asp-validation-for, що використовується для вказівки властивості моделі, до якого повинні відноситися повідомлення про помилки, що відображаються.

2.4. Відображення повідомлень про помилки перевірки достовірності на рівні моделі

Може здатися, що звіт з перевірки достовірності в застосуванні надмірний, він просто дублює повідомлення рівня властивостей, які в цілому корисніші користувачеві, оскільки вони знаходяться поруч з елементами форми, де повинні бути усунені проблеми. Але звіт дозволяє зробити зручний трюк, який полягає в можливості відображати повідомлення, які застосовуються до всієї моделі, а не лише до окремих властивостей.

3. Вказівка правил перевірки достовірності за допомогою метаданих

Одна з проблем із розташуванням логіки перевірки достовірності всередину методу дії пов'язана з тим, що в результаті вона дублюється у кожному методі дії, що отримує дані від користувача. Щоб допомогти зменшити дублювання, процес перевірки достовірності підтримує використання атрибутів. Атрибути дозволяють висловлювати правила перевірки достовірності моделей прямо у класі моделі, гарантуючи застосування одного й того ж набору правил незалежно від методу, що використовується для обробки запиту.

Використовуються два атрибути перевірки достовірності – Required та Range. Атрибут Required вказує, що помилка перевірки достовірності виникає, якщо користувач не надіслав значення для властивості. Атрибут Range задає підмножину прийнятних значень.

3.1. Створення спеціального атрибута перевірки достовірності для властивості

Процес перевірки достовірності може бути розширено за рахунок створення атрибуту, який реалізує інтерфейс IModelValidator.

В інтерфейсі IModelValidator визначено властивість IsRequired, яка застосовується для вказівки, чи потрібна перевірка достовірності за допомогою цього класу (що трохи вводить в оману, значення, яке повертається даною властивістю, просто використовується для впорядкування атрибутів перевірки достовірності, щоб обов'язкові атрибути виконувались першими). Метод Validate() застосовується для перевірки достовірності та отримує інформацію через екземпляр класу ModelValidationContext.

Метод Validate() повертає послідовність об'єктів ModelValidationResult, кожен із яких описує одиночну помилку перевірки достовірності.

4. Виконання перевірки достовірності на стороні клієнта

Усі продемонстровані до цих пір прийоми перевірки були прикладами перевірки достовірності на стороні сервера. Це означає, що користувач надсилає свої дані серверу, сервер перевіряє дані і відправляє назад результати перевірки (або ознака успішності, або список помилок, що підлягають виправленню).

У веб-застосуваннях користувачі зазвичай очікують негайного відгуку перевірки достовірності без необхідності надсилання чого-небудь серверу. Така можливість відома як перевірка достовірності на стороні клієнта та реалізується із застосуванням JavaScript. Дані, що вводяться користувачем, перевіряються на предмет достовірності перед їх відправкою серверу, забезпечуючи користувача негайним відкликом і шансом скоригувати будь-які проблеми.

Інфраструктура MVC підтримує ненав'язливу перевірку достовірності на стороні клієнта. Термін "ненав'язлива" означає, що правила перевірки достовірності виражаються з використанням атрибутів, які додаються до HTML-елементів, що генеруються представленням. Атрибути інтерпретуються бібліотекою JavaScript, що входить до складу інфраструктури MVC, яка, у свою чергу, конфігурує бібліотеку jQuery Validation, яка виконує дійсну роботу з перевірки достовірності.

5. Виконання віддаленої перевірки достовірності

На завершення розділу буде розглянуто віддалену (дистанційну) перевірку достовірності. Це прийом перевірки достовірності на стороні клієнта, при якому до виконання перевірки викликається метод дії на сервері.

Розповсюджений приклад віддаленої перевірки достовірності передбачає з'ясування доступності імені користувача у застосуваннях, де воно повинно бути унікальним, коли користувач надсилає дані та проводиться перевірка достовірності на стороні клієнта. В якості

частини такого процесу серверу надсилається запит Ajax для перевірки імені користувача. Якщо ім'я користувача вже було видано, тоді з'явиться повідомлення про помилку перевірки, і користувачеві надається можливість ввести інше ім'я.

Контрольні питання

1. Дайте визначення поняттю перевірка достовірності моделі для застосування ASP.NET Core MVC.
2. Поясніть необхідність виконання перевірки достовірності моделі для застосування ASP.NET Core MVC.
3. Поясніть призначення класу ModelStateDictionary та перерахуйте кілька його членів для застосування ASP.NET Core MVC.
4. Поясніть призначення перерахування ModelValidationState і перерахуйте кілька його значень для застосування ASP.NET Core MVC.
5. Поясніть функціонування механізму призначення стилізованих змін полів введення представлення, за допомогою таблиць стилів CSS, в результаті виконання перевірки достовірності моделі для застосування ASP.NET Core MVC.
6. Поясніть функціонування механізму призначення стилізованих змін полів введення представлення, за допомогою бібліотеки Bootstrap, в результаті виконання перевірки достовірності моделі для застосування ASP.NET Core MVC.
7. Поясніть функціонування механізму відображення звіту інформаційних повідомлень для представлення, в результаті виконання перевірки достовірності моделі для застосування ASP.NET Core MVC.
8. Поясніть призначення класу DefaultModelBindingMessageProvider і перерахуйте кілька його членів для застосування ASP.NET Core MVC.
9. Поясніть функціонування механізму відображення інформаційного повідомлення поруч із полем введення для представлення, в результаті виконання перевірки достовірності моделі для застосування ASP.NET Core MVC.
10. Поясніть необхідність використання звіту інформаційних повідомлень для представлення, в результаті виконання перевірки достовірності моделі для застосування ASP.NET Core MVC.
11. Поясніть призначення перерахування ValidationSummary та перерахуйте кілька його значень для застосування ASP.NET Core MVC.
12. Поясніть переваги використання атрибутів у моделі для виконання перевірки достовірності моделі для застосування ASP.NET Core MVC.
13. Перерахуйте кілька вбудованих атрибутів моделі для виконання перевірки достовірності моделі для застосування ASP.NET Core MVC.
14. Поясніть необхідність розробки класу, який реалізує спеціальний атрибут для виконання перевірки достовірності моделі для застосування ASP.NET Core MVC.
15. Назвіть інтерфейс, який повинен бути реалізований класом, який реалізує спеціальний атрибут для виконання перевірки достовірності моделі для застосування ASP.NET Core MVC.
16. Поясніть переваги виконання перевірки достовірності моделі на стороні сервера для застосування ASP.NET Core MVC.
17. Поясніть переваги виконання перевірки достовірності моделі на стороні клієнта для застосування ASP.NET Core MVC.
18. Поясніть функціонування механізму віддаленої перевірки достовірності моделі на стороні клієнта для застосування ASP.NET Core MVC.
19. Поясніть переваги виконання віддаленої перевірки достовірності моделі на стороні клієнта для застосування ASP.NET Core MVC.
20. Поясніть основну відмінність виконання віддаленої перевірки достовірності моделі на стороні клієнта від перевірки достовірності моделі на стороні сервера для застосування ASP.NET Core MVC.

Навчальне видання
Конспект лекцій з дисципліни «Програмування та підтримка веб-застосувань» для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 124 Системний аналіз (частина 2) / Укл.: К. О. Трифонова. − Одеса, 2025. − 34 с.

Укладач: Трифонова К. О., ст. викл.

Підписано до друку _____. Формат 60х84/16. Папір газетний. Друк офсетний. 0,87 ум. друк. арк. 0,94 обл. - вид. арк.
Тираж 100 пр. Зам. №

Національний університет «Одеська політехніка»
65044, Одеса, пр. Шевченка, 1