Фреймворк ASP.NET Core

Що таке ASP.NET Core?

ASP.NET Core — це кросплатформна структура програм (фреймворк) із відкритим кодом, яку можна використовувати для швидкого створення динамічних веб-програм. Ви можете використовувати ASP.NET Core для створення серверних веб-додатків, серверних додатків, HTTP API, які можуть використовуватися мобільними додатками, і багато іншого. ASP.NET Core працює на .NET 7, що є останньою версією .NET Core — високопродуктивного кросплатформного середовища виконання з відкритим кодом.

ASP.NET Core забезпечує структуру, допоміжні функції та структуру для створення програм, що позбавляє вас від необхідності самостійно писати велику частину цього коду. Потім код фреймворка ASP.NET Core звертається до ваших обробників, які, у свою чергу, викликають методи бізнес-логіки вашої програми, як показано на малюнку 1.1. Ця бізнес-логіка є основою вашої програми. Тут ви можете взаємодіяти з іншими службами, такими як бази даних або віддалені API, але ваша бізнес-логіка зазвичай не залежить безпосередньо від ASP.NET Core.

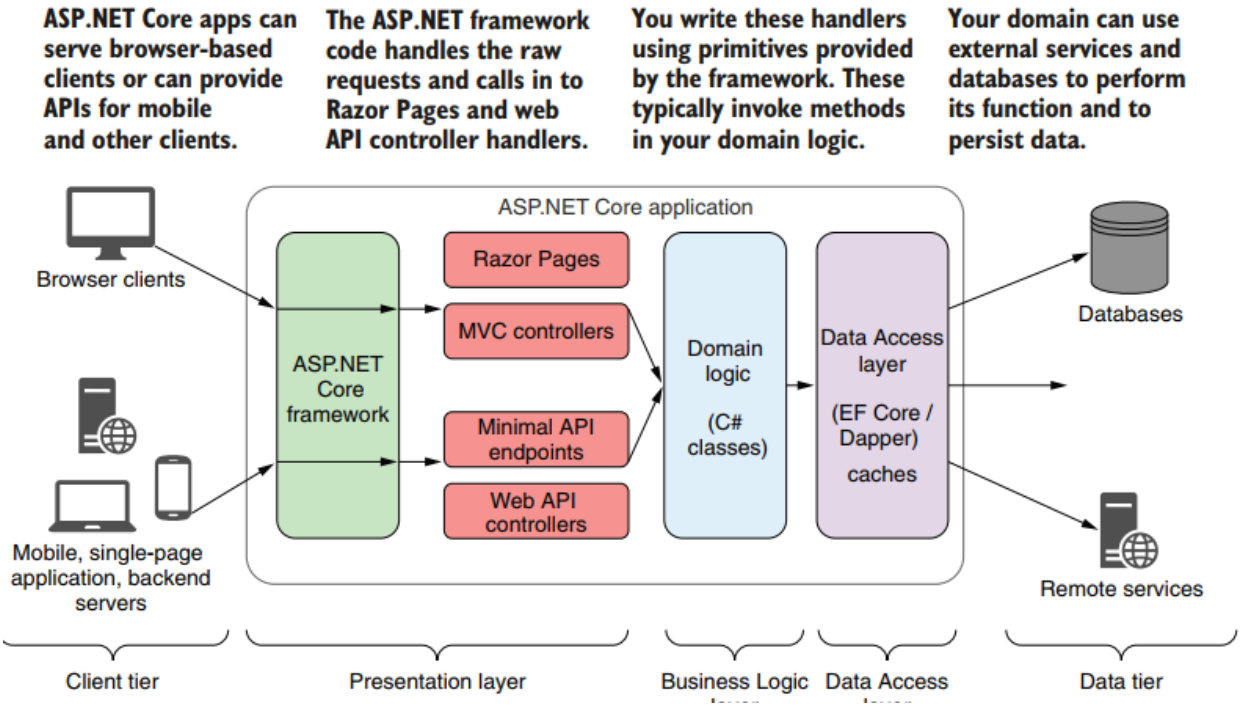

Рис. 1 - Типова програма ASP.NET Core складається з кількох рівнів. Код фреймворка ASP.NET Core обробляє запити від клієнта, маючи справу зі складним мережевим кодом. Потім фреймворк викликає обробники (наприклад, Razor Pages і контролери Web API), які ви пишете за допомогою примітивів, наданих фреймворком. Нарешті, ці обробники викликають логіку домену вашої програми — як правило, класи та об'єкти C# без будь-яких залежностей, специфічних для ASP.NET Core.

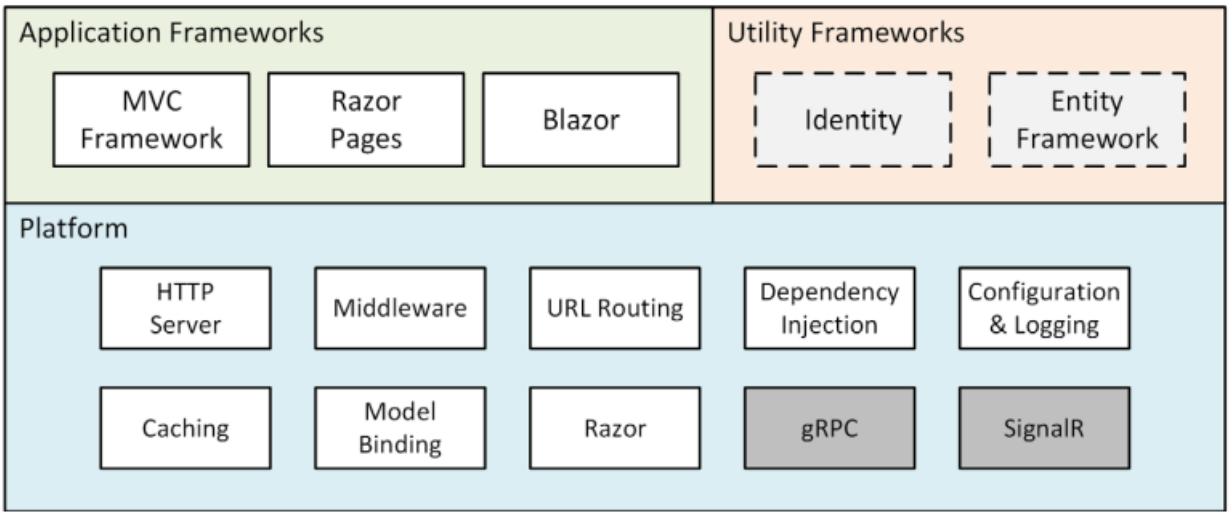

Рис. 2 - Структура ASP.NET Core

Які типи програм ви можете створити?

ASP.NET Core надає узагальнену веб-платформу, яку можна використовувати для створення різноманітних програм. ASP.NET Core містить API, які підтримують багато парадигм:

 Minimal APIs— прості HTTP API, які можуть використовуватися мобільними програмами або односторінковими програмами (SPA) на основі браузера..

 Web APIs — альтернативний підхід до створення HTTP-інтерфейсів API, який додає більше структури та функцій, ніж мінімальні API.

 API gRPC — використовується для створення ефективних бінарних API для зв'язку між серверами за допомогою протоколу gRPC.

 Razor Pages — використовуються для створення додатків, які відображаються сервером на основі сторінок.

 MVC controllers— схожі на Razor Pages. Додатки контролера Model-View-Controller (MVC) призначені для програм на основі сервера, але без парадигми на основі сторінок.

 Blazor WebAssembly — фреймворк односторінкової програми на основі браузера, який використовує стандарт WebAssembly, подібний до фреймворків JavaScript, таких як Angular, React і Vue.

 Blazor Server— використовується для створення додатків із збереженням стану, відтворених на сервері, які надсилають події інтерфейсу користувача та оновлення сторінок через WebSockets, щоб створити відчуття односторінкової програми на стороні клієнта, але з легкістю розробки програми, відтвореної сервером.

Усі ці парадигми базуються на тих самих будівельних блоках ASP.NET Core, таких як бібліотеки конфігурації та журналювання, а потім розміщують додаткову функціональність зверху. Найкраща парадигма для вашої програми залежить від багатьох факторів, зокрема вимог до API, деталей існуючих програм, з якими вам потрібно взаємодіяти, деталей браузерів і операційного середовища ваших клієнтів, а також вимог до масштабованості та часу безвідмовної роботи. Вам не потрібно вибирати лише одну з цих парадигм; ASP.NET Core може поєднувати кілька парадигм в одній програмі.

Розуміння .NET Core, .NET Framework і .NET

Якщо ви ніколи не працювали у великій корпорації, у вас може скластися враження, що Microsoft — це дисциплінована організація з чіткою стратегією та армією програмістів, які разом працюють над розробкою складних продуктів, таких як ASP.NET Core.

Насправді Microsoft - це хаотична сукупність неблагополучних племен, які постійно намагаються підірвати одне одного, щоб отримати престиж і просування по службі. Продукти випускаються під час затишшя в боях, і успіхи часто бувають абсолютно несподіваними. Це не унікально для Microsoft — це справедливо для будь-якої великої компанії, — але це має особливе значення для ASP.NET Core і плутанини з іменами, яку створила Microsoft.

Кілька років тому частина Microsoft, відповідальна за ASP.NET, створила власну версію платформи .NET, що дозволило оновлювати ASP.NET частіше, ніж решту .NET. Було створено ASP.NET Core та .NET Core, що дозволило кросплатформну розробку та використовувало підмножину

оригінальних API .NET, багато з яких були специфічними для Windows. Це був болісний перехід, але це означало, що веб-розробка могла розвиватися незалежно від «застарілої» розробки лише для Windows, яка тривала під перейменованою .NET Framework.

Але ніхто не хоче бути в «спадковому» племені, тому що немає ніякої слави в тому, щоб тримати світло в Microsoft. За .NET Core явно майбутнє, і одна за одною групи .NET у Microsoft стверджували, що їх технологія та API повинні бути частиною .NET Core. API .NET Core поступово розширювалися, і в результаті виник непослідовний безлад із половинчастими спробами розмежувати .NET Core і .NET Framework і стандартизувати API.

Щоб навести порядок, Microsoft об'єднала .NET Core і .NET Framework у .NET, вилучивши частину назви Core. «.NET» — це ім'я, яке, на мою думку, було вибрано, коли виходили з офісу у святкові вихідні, але я підозрюю, що воно є результатом багатьох місяців гарячої суперечки.

Проблема з вилученням Core з назви полягає в тому, що це не можна виконувати послідовно. Назва ASP.NET Core спочатку позначала версію ASP.NET .NET Core, і повернення до цієї назви було б ще більш заплутаним.

У результаті навіть Microsoft не може вирішити, яке ім'я використовувати. Ви побачите термін ASP.NET Core у багатьох документах для розробників, і це назва, яку я використовую в цій книзі, але ви також побачите ASP.NET Core у .NET, особливо в прес-релізах і маркетингових матеріалах. Незрозуміло, яка назва переможе, але доки не буде ясності, вам слід уважно визначити, чи використовуєте ви .NET Framework, .NET Core чи .NET.

Чому було створено ASP.NET Core

Розробка Microsoft ASP.NET Core була мотивована бажанням створити веб-платформу з п'ятьма основними цілями:

- Запуск і розробка кросплатформенних додатків
- Мати модульну архітектуру для полегшення обслуговування
- Повністю розроблятися як програмне забезпечення з відкритим кодом
- Дотримуватися веб-стандартів
- Бути застосовним до сучасних тенденцій веб-розробки, таких як клієнтські програми та розгортання в хмарних середовищах

Щоб досягти всіх цих цілей, Microsoft була потрібна платформа, яка могла б надати базові бібліотеки для створення основних об'єктів, таких як списки та словники, а також для виконання таких завдань, як прості операції з файлами. До цього моменту розробка ASP.NET завжди була зосереджена — і залежала — від .NET Framework лише для Windows. Для ASP.NET Core Microsoft створила полегшену платформу, яка працює на Windows, Linux і macOS під назвою .NET Core (згодом .NET), як показано на малюнку 2.2.

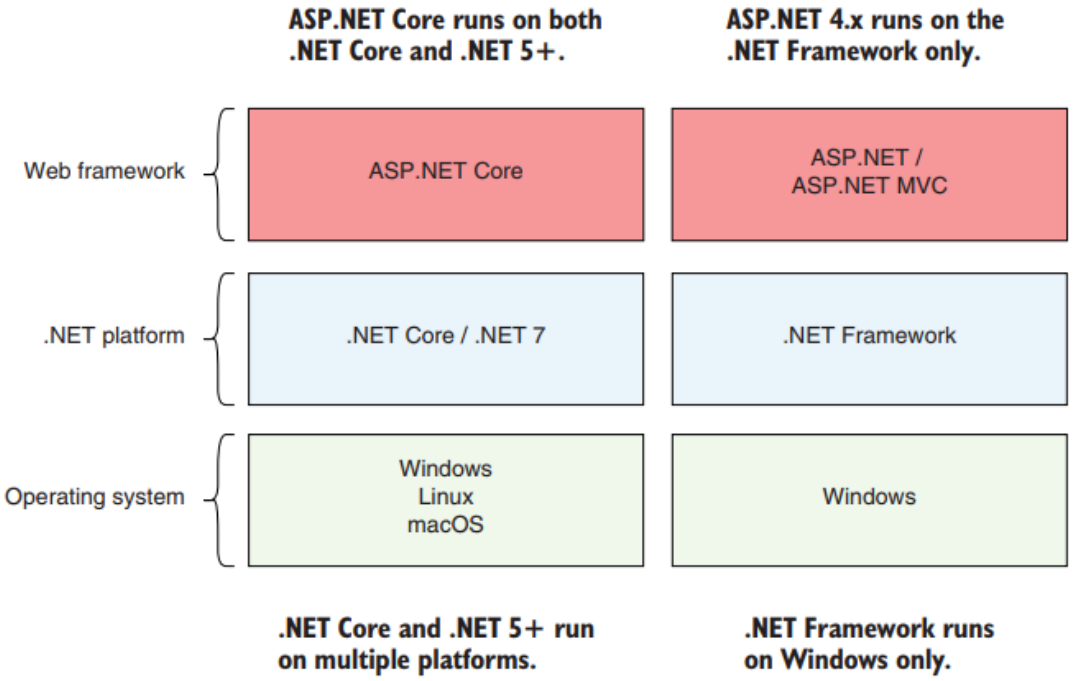

Рис. 3 - Зв'язки між ASP.NET Core, ASP.NET, .NET Core/.NET 5+ і .NET Framework. ASP.NET Core працює на .NET Core та .NET 5+, тому може працювати на різних платформах. Навпаки, ASP.NET працює лише на .NET Framework, тому він прив'язаний до ОС Windows.

Розуміння фреймворків додатків

Коли ви починаєте використовувати ASP.NET Core, вас може збентежити те, що існують різні доступні фреймворки додатків. Як ви дізнаєтесь, ці інфраструктури є доповнювальними та вирішують різні проблеми, або, для деяких функцій, вирішують ті самі проблеми різними способами. Розуміння зв'язку між цими фреймворками означає розуміння мінливих шаблонів проектування, які підтримує Microsoft, як я поясню в наступних розділах.

Розуміння структури MVC

MVC Framework було представлено на початку ASP.NET, задовго до появи .NET Core і новішої .NET. Оригінальний ASP.NET спирався на модель розробки під назвою Web Forms, яка відтворювала досвід написання настільних програм, але призводила до створення громіздких веб-проектів, які погано масштабувалися. MVC Framework було представлено разом із веб-формами з моделлю розробки, яка охопила характер HTTP і HTML, а не намагалася приховати це.

MVC означає Model-View-Controller, який є шаблоном проектування, який описує форму програми. Патерн MVC наголошує на поділі проблем, де області функціональності визначаються незалежно, що було ефективною протиотрутою проти нечіткої архітектури, до якої призвели веб-форми.

Ранні версії MVC Framework були створені на основі ASP.NET, яка спочатку була розроблена для веб-форм, що призвело до деяких незручних функцій і обхідних шляхів. З переходом на .NET Core ASP.NET став ASP.NET Core, а MVC Framework було перебудовано на відкритій, розширюваній і кросплатформній основі.

MVC Framework залишається важливою частиною ASP.NET Core, але спосіб його використання змінився з появою односторінкових програм (SPA). У SPA браузер робить один HTTP-запит і отримує HTML-документ, який надає розширений клієнт, зазвичай написаний у фреймворку JavaScript, наприклад Angular або React. Перехід до SPA означає, що чітке розділення, для якого спочатку розроблялася MVC Framework, не є таким важливим, і наголос на дотриманні шаблону MVC більше не є важливим, навіть якщо MVC Framework залишається корисним (і використовується для підтримки SPA). через веб-сервіси, як описано в розділі 19).

Розуміння сторінок Razor

Одним із недоліків MVC Framework є те, що для цього може знадобитися багато підготовчої роботи, перш ніж програма почне створювати вміст. Незважаючи на структурні проблеми, одна з переваг Web Forms полягала в тому, що прості програми можна було створювати за пару годин.

Razor Pages використовує дух розробки веб-форм і реалізує його за допомогою функцій платформи, спочатку розроблених для MVC Framework.

Код і вміст змішуються, щоб сформувати самодостатні сторінки; це відтворює швидкість розробки Web Forms без деяких основних технічних проблем (хоча масштабування складних проектів все ще може бути проблемою).

Сторінки Razor можна використовувати разом із MVC Framework, саме так я зазвичай їх використовую. Я пишу основні частини програми за допомогою MVC Framework і використовую Razor Pages для додаткових функцій, таких як інструменти адміністрування та звітування. Ви можете побачити цей підхід у розділах 7–11, де я розробляю реалістичну програму ASP.NET Core під назвою SportsStore.

Розуміння Blazor

Розвиток клієнтських фреймворків JavaScript може стати перешкодою для розробників C#, які повинні вивчити іншу — і дещо своєрідну — мову програмування. Я полюбив JavaScript, який такий же гнучкий і виразний, як C#. Але щоб навчитися володіти новою мовою програмування, особливо тією, яка має фундаментальні відмінності від C#, потрібен час і зусилля.

Blazor намагається подолати цю прогалину, дозволяючи використовувати C# для написання програм на стороні клієнта. Існує дві версії Blazor: Blazor Server і Blazor WebAssembly. Blazor Server покладається на постійне HTTP-з'єднання з сервером ASP.NET Core, де виконується код C# програми. Blazor WebAssembly йде на крок далі і виконує код C# програми в браузері. Жодна з версій Blazor не підходить для всіх ситуацій, як я пояснюю в розділі 33, але вони обидві дають відчуття напрямку для майбутнього розвитку ASP.NET Core.

Розуміння корисних структур

Два фреймворки тісно пов'язані з ASP.NET Core, але не використовуються безпосередньо для створення HTML-контенту чи даних. Entity Framework Core — це структура об'єктно-реляційного відображення (ORM) Microsoft, яка представляє дані, що зберігаються в реляційній базі даних, як об'єкти .NET. Entity Framework Core можна використовувати в будь-якій програмі .NET, і вона зазвичай використовується для доступу до баз даних у програмах ASP.NET Core.

ASP.NET Core Identity — це платформа автентифікації та авторизації Microsoft, яка використовується для перевірки облікових даних користувача в програмах ASP.NET Core та обмеження доступу до функцій програми.

У цій книзі я описую лише основні функції обох фреймворків, зосереджуючись на можливостях, необхідних більшості програм ASP.NET Core. Але це обидва складні фреймворки, які занадто великі, щоб детально описати їх у великій книзі про ASP.NET Core.

Розуміння платформи ASP.NET Core

Платформа ASP.NET Core містить функції низького рівня, необхідні для отримання й обробки HTTP-запитів і створення відповідей. Існує інтегрований HTTP-сервер, система компонентів проміжного програмного забезпечення для обробки запитів і основні функції, від яких залежать фреймворки додатків, такі як маршрутизація URL-адрес і механізм перегляду Razor.

Більшу частину вашого часу розробки витрачатиметься на фреймворки додатків, але ефективне використання ASP.NET Core вимагає розуміння потужних можливостей, які надає платформа, без яких фреймворки вищого рівня не зможуть функціонувати. У другій частині цієї книги я детально продемонструю, як працює платформа ASP.NET Core, і поясню, як її функції лежать в основі кожного аспекту розробки ASP.NET Core.

У цій книзі я не описував дві помітні функції платформи: SignalR і gRPC. SignalR використовується для створення каналів зв'язку з низькою затримкою між програмами. Він забезпечує основу для інфраструктури Blazor Server, яку я описую в частині 4 цієї книги, але SignalR рідко використовується безпосередньо, і є кращі альтернативи для тих небагатьох проектів, які потребують обміну повідомленнями з низькою затримкою, наприклад Azure Event Grid або Azure Service Bus.

gRPC — це новий стандарт для міжплатформних віддалених викликів процедур (RPC) через HTTP, який спочатку був створений Google (g у gRPC) і пропонує переваги ефективності та масштабованості. gRPC може бути майбутнім стандартом для веб-сервісів, але його не можна використовувати у веб-додатках, оскільки він вимагає низькорівневого контролю HTTP-повідомлень, які він надсилає, а браузери цього не дозволяють. (Існує бібліотека браузера, яка дозволяє використовувати gRPC через проксі-сервер,

але це підриває переваги використання gRPC.) Поки gRPC не можна буде використовувати в браузері, його включення в ASP.NET Core цікаво лише для проектів, які використовувати його для зв'язку між внутрішніми серверами, наприклад, у розробці мікросервісів. Я можу розповісти про gRPC у наступних виданнях цієї книги, але лише тоді, коли його можна буде використовувати у браузері.

Встановлення Visual Studio

Для ASP.NET Core 7 потрібна Visual Studio 2022. Я використовую безкоштовну Visual Studio 2022 Community Edition, яку можна завантажити з www.visualstudio.com . Запустіть програму встановлення, і ви побачите підказку, показану на Рис. 4.

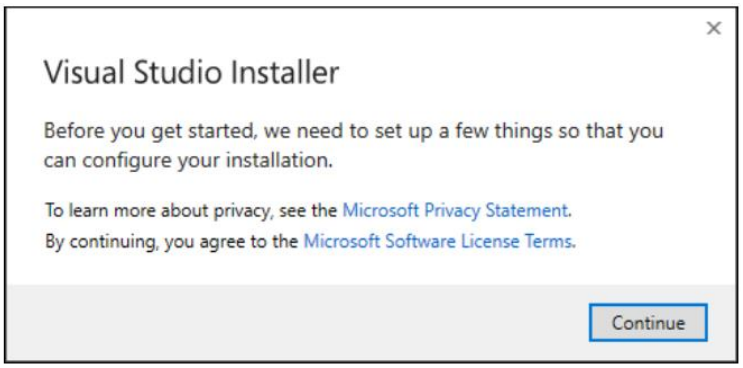

Рис. 4 – Запуск інсталятора Visual Studio

Натисніть кнопку «Continue», і програма встановлення завантажить файли встановлення, як показано на Рис. 5.

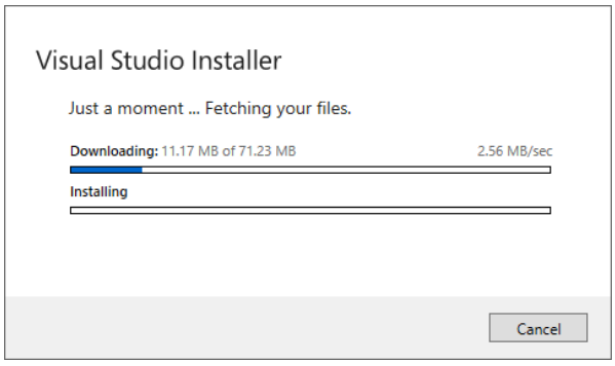

Рис. 5 – Завантаження файлів інсталятора Visual Studio

Коли файли інсталятора будуть завантажені, вам буде запропоновано набір параметрів інсталяції, згрупованих у робочі навантаження.

Переконайтеся, що позначено робоче навантаження «ASP.NET and web development», як показано на Рис. 6.

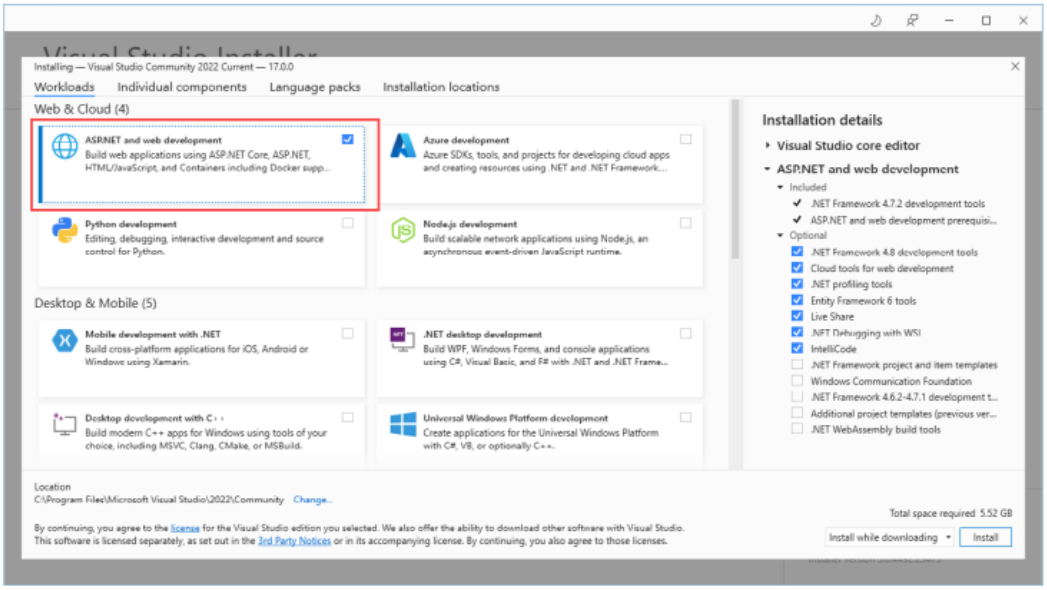

Рис. 6 – Вибір навантаження

Виберіть розділ «Individual components» у верхній частині вікна та переконайтеся, що параметр SQL Server Express 2019 LocalDB позначено, як показано на Рис. 7. Це компонент бази даних, який ми будемо використовувати для зберігання даних у наступних розділах.

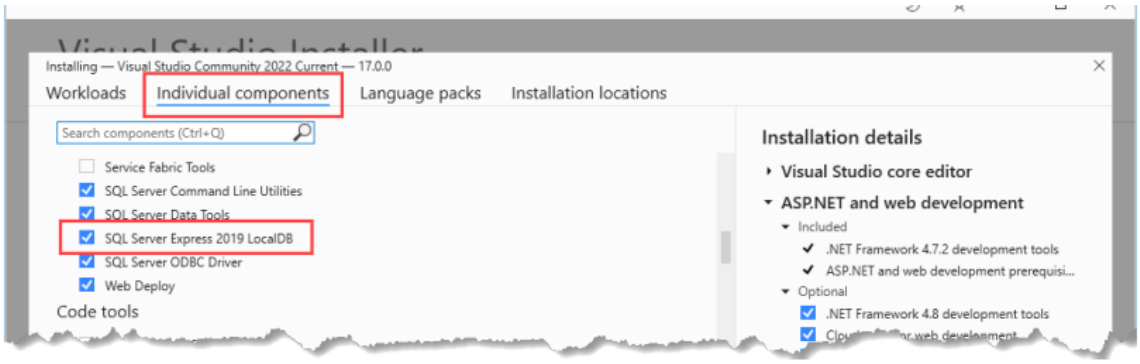

Рис. 7 – Переконайтеся, що встановлено LocalDB

Натисніть кнопку «Встановити», і файли, необхідні для вибраного робочого навантаження, будуть завантажені та встановлені. Для завершення встановлення може знадобитися перезавантаження.

ПРИМІТКА Ви також повинні інсталювати SDK, як описано в наступному розділі.

Встановлення .NET SDK

Інсталятор Visual Studio встановить .NET Software Development Kit (SDK), але він може не встановити версію, необхідну для прикладів у цій книзі. Перейдіть на сторінку https://dotnet.microsoft.com/download/dotnet-core/7.0 і завантажте програму встановлення для версії 7.0.0 .NET SDK, яка є поточною версією на момент написання статті.

Запустіть інсталятор; після завершення інсталяції відкрийте новий командний рядок PowerShell із меню «Пуск» Windows і виконайте команду, показану в лістингу 2.1, який відображає список встановлених .NET SDK.

Listing 2.1 Listing the Installed SDKs

```
dotnet --list-sdks
```

Ось результат нової інсталяції на машині Windows, яка не використовувалася для .NET:

```
7.0.100 [C:\Program Files\dotnet\sdk]

. . . .
```

Створення проекту ASP.NET Core

Найпростіший спосіб створити проект - це використовувати командний рядок. Відкрийте новий командний рядок PowerShell у меню «Пуск» Windows, перейдіть до папки, де ви хочете створити проекти ASP.NET Core, і виконайте команди, показані в лістингу 2.3.

ПОРАДА Ви можете завантажити приклад проекту для цього розділу — і для всіх інших розділів у цій книзі — з https://github.com/manningbooks/pro-asp.net-core-7 . Перегляньте розділ 1, щоб дізнатися, як отримати допомогу, якщо у вас виникли проблеми з виконанням прикладів.

Лістинг 2.3 Створення проєкта FirstProject

```
dotnet new globaljson --sdk-version 7.0.100 --output FirstProject
dotnet new mvc --no-https --output FirstProject --framework net7.0
dotnet new sln -o FirstProject
dotnet sln FirstProject add FirstProject
```

Перша команда створює папку під назвою FirstProject і додає до неї файл під назвою global.json, який визначає версію .NET, яку використовуватиме проект; це гарантує, що ви отримаєте очікувані результати, дотримуючись

прикладів. Друга команда створює новий проект ASP.NET Core. .NET SDK містить ряд шаблонів для запуску нових проектів, а шаблон mvc є одним із варіантів, доступних для програм ASP.NET Core. Цей шаблон проекту створює проект, налаштований для MVC Framework, яка є одним із типів програм, які підтримуються ASP.NET Core. Нехай вас не лякає ідея вибору фреймворка і не хвилюйтеся, якщо ви ще не чули про MVC — до кінця книги ви зрозумієте, які функції пропонує кожний з них і як вони поєднуються. Інші команди створюють файл рішення, який дозволяє використовувати декілька проектів разом.

ПРИМІТКА Це один із небагатьох розділів, у яких я використовую шаблон проекту, який містить вміст-заповнювач (placeholder). Мені не подобається використовувати попередньо визначені шаблони проектів, оскільки вони спонукають розробників розглядати важливі функції, такі як автентифікація, як чорні ящики. Моя мета в цій книзі — дати вам знання, щоб зрозуміти та керувати кожним аспектом ваших програм ASP.NET Core, тому я починаю з порожнього проекту ASP.NET Core. Цей розділ розповідає про те, як швидко розпочати роботу, для чого добре підходить шаблон mvc.

Відкриття проекту за допомогою Visual Studio

Запустіть Visual Studio та натисніть кнопку «Open a project or solution», як показано на Рис. 8.

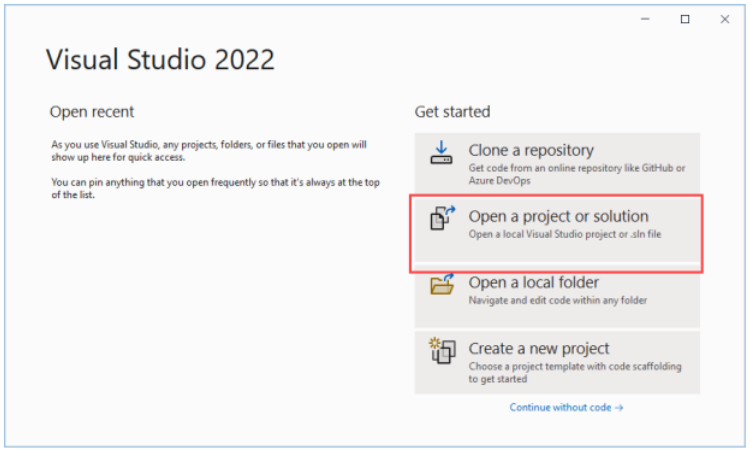

Рис. 8 – Відкриття проекту ASP.NET Core

Перейдіть до папки FirstProject, виберіть файл FirstProject.sln і натисніть кнопку «Відкрити». Visual Studio відкриє проект і відобразить його вміст у

вікні Solution Explorer, як показано на Рис. 9. Файли в проекті були створені за шаблоном проекту.

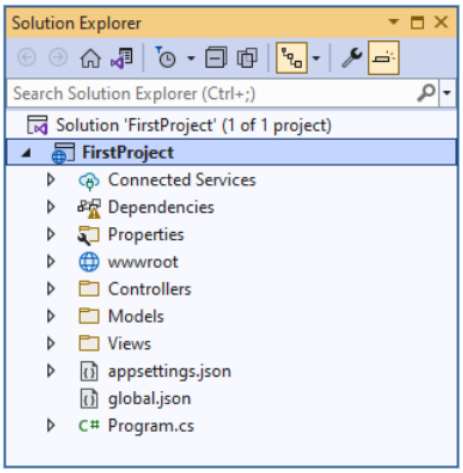

Рис. 9 – Відкриття проекту в Visual Studio

Запуск програми ASP.NET Core

Як Visual Studio, так і Visual Studio Code можуть запускати проекти безпосередньо, але я використовую інструменти командного рядка в цій книзі, оскільки вони надійніші та працюють послідовніше, допомагаючи отримати очікувані результати від прикладів.

Коли проект створюється, у папці Properties створюється файл launchSettings.json, і саме цей файл визначає, який HTTP-порт ASP.NET Core використовуватиме для прослуховування HTTP-запитів. Відкрийте цей файл у вибраному редакторі та змініть порти в URL-адресах, які він містить, на 5000, як показано в лістингу 2.4.

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:5000",
      "sslPort": 0
    }
  },
  "profiles": {
    "FirstProject": {
      "commandName": "Project",
```

```
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Лише URL-адреса в розділі профілів впливає на інструменти командного рядка .NET, але я змінив обидва, щоб уникнути проблем. Відкрийте новий командний рядок PowerShell у меню «Пуск» Windows; перейдіть до папки проекту FirstProject, яка містить файл FirstProject.csproj; і запустіть команду, показану в лістингу 2.5.

```
dotnet run
```

Команда dotnet run компілює та запускає проект. Після запуску програми відкрийте нове вікно браузера та надішліть запит http://localhost:5000, що дасть відповідь, показану на Рис. 10.

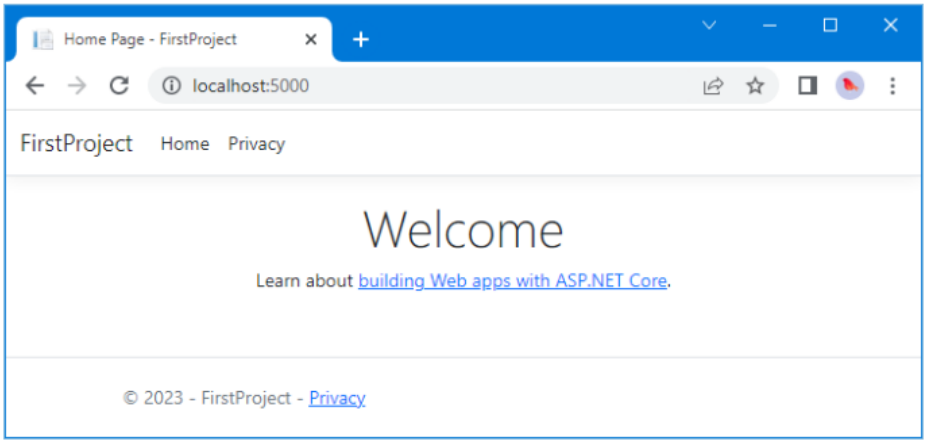

Рис. 10 – Запуск прикладу проекту

Завершивши, натисніть Control+C, щоб зупинити програму ASP.NET Core.

Розуміння кінцевих точок

У програмі ASP.NET Core вхідні запити обробляються кінцевими точками. Кінцевою точкою, яка створила відповідь на Рис.10, є дія, яка є методом, написаним на C#. Дія визначається в контролері, який є класом C#, похідним від класу Microsoft.AspNetCore.Mvc.Controller, вбудованого базового класу контролера.

Кожен публічний метод, визначений контролером, є дією, що означає, що ви можете викликати метод дії для обробки запиту HTTP. Конвенція в проектах ASP.NET Core передбачає розміщення класів контролерів у папці з іменем Controllers, яка була створена шаблоном, використаним для налаштування проекту.

Шаблон проекту додав контролер до папки Controllers, щоб допомогти швидко розпочати розробку. Контролер визначено у файлі класу з назвою HomeController.cs. Класи контролерів містять назву, після якої йде слово Controller, що означає, що коли ви бачите файл під назвою HomeController.cs, ви знаєте, що він містить контролер під назвою Home, який є контролером за замовчуванням, який використовується в програмах ASP.NET Core.

ПОРАДА Не хвилюйтеся, якщо терміни «контролер» і «дія» не мають сенсу. Просто продовжуйте слідувати прикладу, і ви побачите, як HTTP-запит, надісланий браузером, обробляється кодом C#.

Знайдіть файл HomeController.cs на панелі Solution Explorer або Explorer і клацніть його, щоб відкрити для редагування. Ви побачите такий код:

```
using System.Diagnostics;
using Microsoft.AspNetCore.Mvc;
using FirstProject.Models;
namespace FirstProject.Controllers;
public class HomeController : Controller {
  private readonly ILogger<HomeController> _logger;

  public HomeController(ILogger<HomeController> logger) {
_  logger = logger;
  }
  public IActionResult Index() {
    return View();
  }
```

```
  public IActionResult Privacy() {
    return View();
  }
  [ResponseCache(Duration = 0, Location =
ResponseCacheLocation.None,
  NoStore = true)]
  public IActionResult Error() {
    return View(new ErrorViewModel { RequestId =
Activity.Current?.Id
  ?? HttpContext.TraceIdentifier });
  }
}
```

За допомогою редактора коду замініть вміст файлу HomeController.cs так, щоб він відповідав лістингу 2.6. Я видалив усі методи, крім одного, змінив тип результату та його реалізацію, а також видалив оператори використання для невикористаних просторів імен.

```
using Microsoft.AspNetCore.Mvc;
namespace FirstProject.Controllers {
    public class HomeController : Controller {
        public string Index() {
            return "Hello World";
        }
    }
}
```

У результаті контролер Home визначає одну дію під назвою Index. Ці зміни не справляють драматичного ефекту, але створюють гарну демонстрацію. Я змінив метод під назвою Index, щоб він повертав рядок Hello World.

За допомогою підказки PowerShell ще раз запустіть команду dotnet run у папці FirstProject і за допомогою браузера надішліть запит http://localhost:5000 . Конфігурація проекту, створена за допомогою шаблону в лістингу 2.3, означає, що запит HTTP буде оброблено дією Index, визначеною контролером Home. Іншими словами, запит буде оброблено методом Index, визначеним класом HomeController. Рядок, створений методом Index, використовується як відповідь на HTTP-запит браузера, як показано на малюнку Рис. 11.

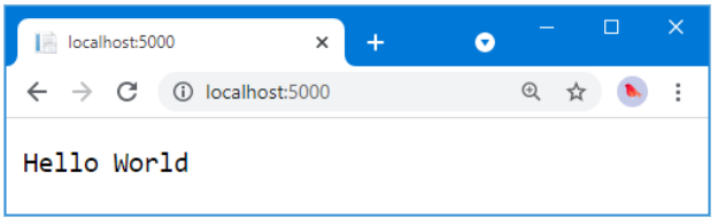

Рис. 11 – Виведення з методу дії

Розуміння маршрутів

Система маршрутизації ASP.NET Core відповідає за вибір кінцевої точки, яка оброблятиме запит HTTP. Маршрут — це правило, яке використовується для визначення способу обробки запиту. Під час створення проекту для початку було створено правило за замовчуванням. Ви можете запитати будь-яку з наведених нижче URL-адрес, і вони будуть надіслані до дії Index, визначеної домашнім контролером:

```
¡ /
¡ /Home
¡ /Home/Index
```

Отже, коли браузер запитує http://yoursite/ або http://yoursite/Home , він повертає вихід із методу Index HomeController. Ви можете спробувати це самостійно, змінивши URL-адресу в браузері. На даний момент це буде http://localhost:5000/ . Якщо ви додасте /Home або /Home/Index до URL-адреси та натиснете Return, ви побачите той самий результат Hello World від програми.

Розуміння відтворення HTML

Результатом попереднього прикладу був не HTML, а просто рядок Hello World. Щоб створити HTML-відповідь на запит браузера, мені потрібне представлення, яке повідомляє ASP.NET Core, як обробити результат, створений методом Index, у відповідь HTML, яку можна надіслати в браузер.

Створення та рендеринг подання (view)

Перше, що мені потрібно зробити, це змінити метод дії Index, як показано в лістингу 2.7. Зміни виділено жирним шрифтом, що є умовністю, якої я дотримуюся в цій книзі, щоб полегшити наслідування прикладів.

Лістинг 2.7 Відтворення представлення у файлі HomeController.cs у папці Controllers

```
using Microsoft.AspNetCore.Mvc;
namespace FirstProject.Controllers {
  public class HomeController : Controller {

    public ViewResult Index() {
      return View("MyView");
    }
  }
}
```

Коли я повертаю об'єкт ViewResult із методу дії, я даю вказівку ASP.NET Core відобразити представлення. Я створюю ViewResult, викликаючи метод View, вказуючи назву представлення, яке я хочу використати, яке є MyView.

Використовуйте Control+C, щоб зупинити ASP.NET Core, а потім використовуйте команду dotnet run, щоб скомпілювати та запустити його знову. Використовуйте браузер для запиту http://localhost:5000 , і ви побачите, що ASP.NET Core намагається знайти подання, як показано в повідомленні про помилку, яке відображається на Рис. 12.

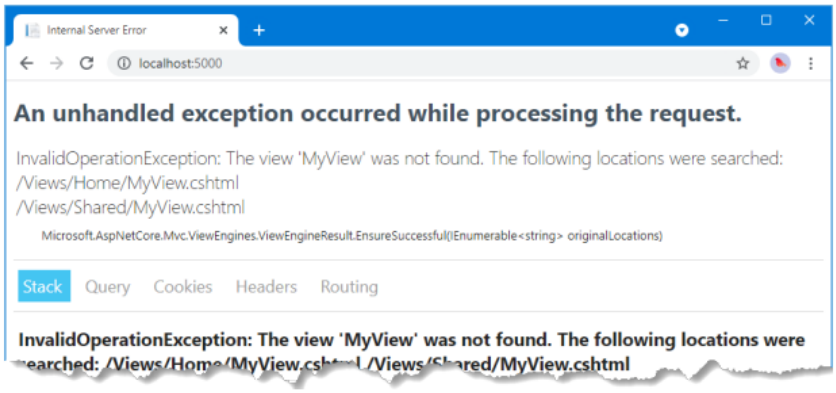

Рис. 12 – Намагаючись знайти view

Це корисне повідомлення про помилку. У ньому пояснюється, що ASP.NET Core не вдалося знайти представлення, яке я вказав для методу дії, і пояснюється, де він шукав. Подання зберігаються в папці «Views», упорядкованій у підпапки. Перегляди, пов'язані з контролером Home, наприклад, зберігаються в папці під назвою Views/Home. Перегляди, які не стосуються окремого контролера, зберігаються в папці під назвою Views/Shared. Шаблон, який використовувався для створення проекту,

автоматично додав папки «Home» та «Shared» та додав деякі подання покажчиків місця заповнення для запуску проекту.

Якщо ви використовуєте Visual Studio, клацніть правою кнопкою миші папку Views/Home в Solution Explorer і виберіть Add > New Item у спливаючому меню. Visual Studio надасть вам список шаблонів для додавання елементів до проекту. Знайдіть елемент Razor View - Empty, який можна знайти в розділі ASP.NET Core > Web > ASP.NET, як показано на Рис. 13.

Для Visual Studio вам може знадобитися натиснути кнопку «Show All Templates», перш ніж відобразиться список шаблонів. Назвіть новий файл MyView.cshtml і натисніть кнопку «Додати». Visual Studio додасть файл із назвою MyView.cshtml до папки Views/Home і відкриє його для редагування. Замініть вміст файлу вмістом, показаним у лістингу 2.8.

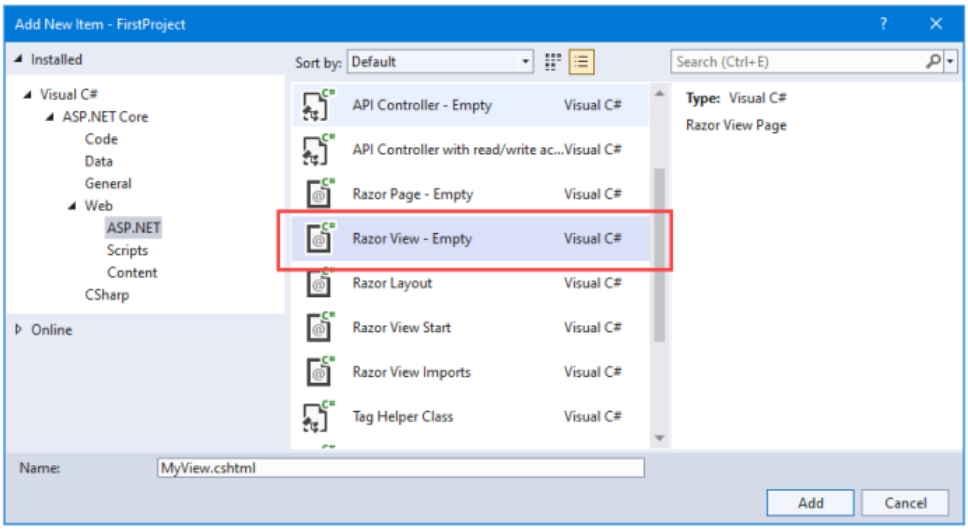

Рис. 13 – Вибір шаблону елемента Visual Studio

ПОРАДА Легко створити файл перегляду не в тій папці. Якщо ви не отримали файл під назвою MyView.cshtml у папці Views/Home, перетягніть файл у потрібну папку або видаліть файл і повторіть спробу.

Лістинг 2.8 Вміст файлу MyView.cshtml у папці Views/Home

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
```

```
    <title>Index</title>
</head>
<body>
    <div>
        Hello World (from the view)
    </div>
</body>
</html>
```

Новий вміст файлу перегляду є переважно HTML. Винятком є частина, яка виглядає так:

```
...
@{
    Layout = null;
}
...
```

Це вираз, який інтерпретуватиме Razor, який є компонентом, який обробляє вміст переглядів і генерує HTML, який надсилається в браузер. Razor — це механізм перегляду, а вирази в представленнях відомі як вирази Razor.

Вираз Razor у лістингу 2.8 повідомляє Razor, що я вирішив не використовувати макет, який схожий на шаблон для HTML, який буде надіслано в браузер (і який я описую в розділі 22). Щоб побачити ефект від створення представлення, використовуйте Control+C, щоб зупинити ASP.NET Core, якщо він запущений, і використайте команду dotnet run, щоб скомпілювати та запустити програму знову. Використовуйте браузер для запиту http://localhost:5000 , і ви побачите результат, показаний на Рис. 14.

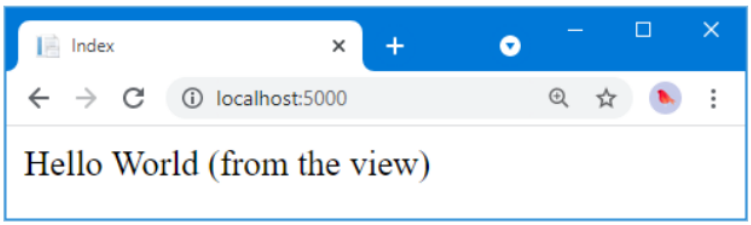

Рис.14 – Рендеринг перегляду

Коли я вперше редагував метод дії Index, він повернув рядкове значення. Це означало, що ASP.NET Core нічого не робив, окрім передачі рядкового значення в браузер. Тепер, коли метод Index повертає ViewResult, Razor використовується для обробки представлення та відтворення HTML-відповіді. Razor зміг знайти подання, оскільки я дотримувався стандартної угоди про

іменування, тобто поміщати файли перегляду в папку, ім'я якої збігалося з контролером, який містить метод дії. У цьому випадку це означало розміщення файлу перегляду в папці Views/Home, оскільки метод дії визначається контролером Home.

Я можу повертати інші результати з методів дії, окрім рядків і об'єктів ViewResult. Наприклад, якщо я поверну RedirectResult, браузер буде перенаправлено на іншу URL-адресу. Якщо я повертаю HttpUnauthorizedResult, я можу запропонувати користувачеві ввійти. Ці об'єкти спільно відомі як результати дії. Система результатів дій дозволяє інкапсулювати та повторно використовувати загальні відповіді в діях. Я розповім вам більше про них і поясню різні способи їх використання в розділі 19.

Додавання динамічного виведення

Суть веб-додатку полягає в створенні та відображенні динамічного виведення. Завдання методу дії полягає в створенні даних і передачі їх у представлення, щоб їх можна було використовувати для створення вмісту HTML на основі значень даних. Методи дії надають дані представленням, передаючи аргументи методу View, як показано в лістингу 2.9. Дані, надані в представленні, називають моделлю представлення.

Лістинг 2.9 Використання моделі перегляду у файлі HomeController.cs у папці Controllers

```
using Microsoft.AspNetCore.Mvc;
namespace FirstProject.Controllers {
    public class HomeController : Controller {

        public ViewResult Index() {
            int hour = DateTime.Now.Hour;
            string viewModel =
                hour < 12 ? "Good Morning" : "Good Afternoon";
            return View("MyView", viewModel);
        }
    }
}
```

Модель представлення в цьому прикладі — це рядок, і він надається представленню як другий аргумент методу View. Лістинг 2.10 оновлює

представлення таким чином, щоб воно отримувало та використовувало модель представлення у створеному HTML.

Лістинг 2.10 Використання моделі перегляду у файлі MyView.cshtml у папці Views/Home

```
@model string
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
        @Model World (from the view)
    </div>
</body>
</html>
```

Тип моделі перегляду вказується за допомогою виразу @model з маленькою літерою m. Значення моделі представлення включається до вихідних даних HTML за допомогою виразу @Model із великою літерою M. (Спочатку може бути важко запам'ятати, що у нижньому, а що у верхньому регістрі, але незабаром це стає другорядним).

Коли подання відтворюється, дані моделі подання, надані методом дії, вставляються у відповідь HTML. Використовуйте Control+C, щоб зупинити ASP.NET Core, і використовуйте команду dotnet run, щоб створити та запустити його знову. Використовуйте браузер для запиту http://localhost:5000 , і ви побачите результат, показаний на Рис. 15 (хоча ви можете побачити ранкове привітання, якщо ви дотримуєтеся цього прикладу до полудня).

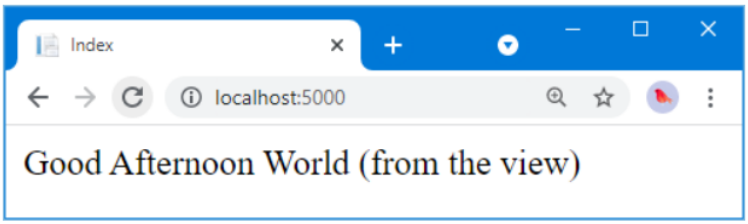

Рис. 15 – Створення динамічного контенту

Складання частин разом

Це простий результат, але цей приклад розкриває всі будівельні блоки, необхідні для створення простого веб-додатку ASP.NET Core і генерації динамічної відповіді.

Платформа ASP.NET Core отримує HTTP-запит і використовує систему маршрутизації для відповідності URL-адреси запиту кінцевій точці. Кінцевою точкою в цьому випадку є метод дії Index, визначений контролером Home. Метод викликається та створює об'єкт ViewResult, який містить назву представлення та об'єкт моделі представлення. Механізм перегляду Razor знаходить і обробляє представлення, оцінюючи вираз @Model, щоб вставити дані, надані методом дії, у відповідь, яка повертається в браузер і відображається користувачеві. Звичайно, є багато інших доступних функцій, але це суть ASP.NET Core, і варто пам'ятати про цю просту послідовність, коли ви читатимете решту книги.

Резюме

Розробку ASP.NET Core можна виконувати за допомогою Visual Studio або Visual Studio Code, або ви можете вибрати власний редактор коду.

Більшість редакторів коду забезпечують інтегроване збирання коду, але найнадійніший спосіб отримати узгоджені результати для різних інструментів і платформ — це використовувати команду dotnet.

ASP.NET Core покладається на кінцеві точки для обробки запитів HTTP.

Кінцеві точки можуть бути написані повністю на C# або використовувати HTML, який був анотований виразами коду.

Ваша перша програма ASP.NET Core

Цей розділ охоплює

- Використання ASP.NET Core для створення програми, яка приймає відповіді RSVP

- Створення простої моделі даних

- Створення контролера та подання, яке представляє та обробляє форму

- Перевірка даних користувача та відображення помилок перевірки

- Застосування стилів CSS до HTML, створеного програмою

Тепер, коли ви налаштовані на розробку ASP.NET Core, настав час створити просту програму. У цьому розділі ви створите додаток для введення даних за допомогою ASP.NET Core. Моя мета — продемонструвати ASP.NET Core у дії, тому я трохи прискорю темп і пропущу деякі пояснення того, як все працює за лаштунками. Але не хвилюйтеся; Я ще раз перегляну ці теми в наступних розділах.

Оформлення сцени

Уявіть, що подруга вирішила влаштувати новорічну вечірку та попросила мене створити веб-програму, яка дозволить її запрошеним електронною мовою відповідати на запрошення (RSVP). Вона попросила ці чотири ключові характеристики:

- Домашня сторінка з інформацією про вечірку
- Форма, яку можна використовувати для відповіді RSVP
- Перевірка форми RSVP, яка відображатиме сторінку подяки
- Сторінка підсумків, яка показує, хто прийде на вечірку

У цьому розділі я створюю проект ASP.NET Core і використовую його для створення простої програми, яка містить ці функції; коли все запрацює, я застосую деякі стилі, щоб покращити зовнішній вигляд готової програми.

Створення проекту

Відкрийте командний рядок PowerShell у меню «Пуск» Windows, перейдіть у зручне розташування та виконайте команди в Лістингу 3.1, щоб створити проект під назвою PartyInvites.

ПОРАДА Ви можете завантажити приклад проекту для цього розділу — і для всіх інших розділів у цій книзі — з https://github.com/manningbooks/pro-asp.net-core-7 . Перегляньте розділ 1, щоб дізнатися, як отримати допомогу, якщо у вас виникли проблеми з виконанням прикладів.

Лістинг 3.1 Створення нового проекту

```
dotnet new globaljson --sdk-version 7.0.100 --output PartyInvites
dotnet new mvc --no-https --output PartyInvites --framework net7.0
dotnet new sln -o PartyInvites
dotnet sln PartyInvites add PartyInvites
```

Це ті самі команди, які я використовував для створення проекту в розділі 2. Ці команди гарантують, що ви отримаєте правильну початкову точку проекту, яка використовує необхідну версію .NET.

Підготовка проекту

Відкрийте проект (відкривши файл PartyInvites.sln у Visual Studio або папку PartyInvites у Visual Studio Code) і змініть вміст файлу launchSettings.json у папці Properties, як показано в лістингу 3.2, щоб встановити порт, який буде використовуватися для прослуховування запитів HTTP.

Лістинг 3.2 Налаштування портів у файлі launchSettings.json у папці Properties

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:5000",
      "sslPort": 0
    }
  },
  "profiles": {
    "PartyInvites": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
  },
  "IIS Express": {
    "commandName": "IISExpress",
    "launchBrowser": true,
    "environmentVariables": {
      "ASPNETCORE_ENVIRONMENT": "Development"
    }
  }
```

```
  }
}
```

Замініть вміст файлу HomeController.cs у папці Controllers на код, показаний у лістингу 3.3.

Лістинг 3.3 Новий вміст файлу HomeController.cs у папці Controllers

```
using Microsoft.AspNetCore.Mvc;
namespace PartyInvites.Controllers {
    public class HomeController : Controller {
        public IActionResult Index() {
          return View();
        }
    }
}
```

Це забезпечує чисту відправну точку для нової програми, визначаючи єдиний метод дії, який вибирає перегляд за замовчуванням для візуалізації. Щоб надіслати вітальне повідомлення запрошеним на вечірку, відкрийте файл Index.cshtml у папці Views/Home і замініть вміст вмістом, показаним у лістингу 3.4.

Лістинг 3.4 Заміна вмісту файлу Index.cshtml у папці Views/Home

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Party!</title>
</head>
<body>
    <div>
      <div>
        We're going to have an exciting party.<br />
        (To do: sell it better. Add pictures or something.)
      </div>
    </div>
```

```
</body>
</html>
```

Виконайте команду, показану в лістингу 3.5, у папці PartyInvites, щоб скомпілювати та виконати проект.

Лістинг 3.5 Компіляція та виконання проекту

```
dotnet watch
```

Після запуску проекту відкриється нове вікно браузера, і ви побачите деталі вечірки (ну, заповнювач для деталей, але ви зрозуміли), як показано на Рис. 16.

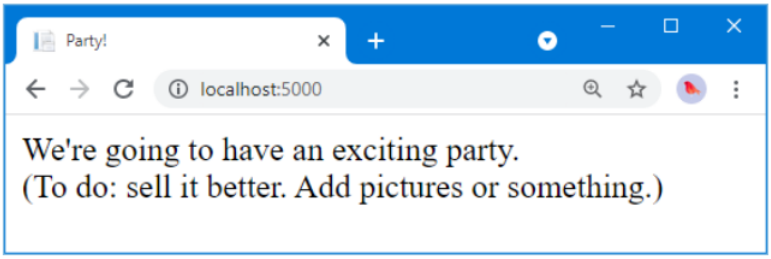

Рис. 16 – Додавання до перегляду HTML

Залиште команду dotnet watch запущеною. Коли ви вносите зміни в проект, ви побачите, що код автоматично перекомпільовується, а зміни автоматично відображаються у браузері.

Якщо ви зробите помилку, дотримуючись прикладів, ви можете виявити, що команда dotnet watch вказує на те, що вона не може автоматично оновлювати браузер. Якщо це станеться, виберіть опцію перезапуску програми.

Додавання моделі даних

Модель даних є найважливішою частиною будь-якої програми ASP.NET Core. Модель — це представлення об’єктів реального світу, процесів і правил, які визначають предмет програми, відомий як домен. Модель, яку часто називають моделлю домену, містить об’єкти C# (відомі як об’єкти домену), які складають всесвіт програми, і методи, які ними керують. У більшості проектів завдання програми ASP.NET Core полягає в тому, щоб надати користувачеві доступ до моделі даних і функцій, які дозволяють користувачеві взаємодіяти з нею.

Конвенція для програми ASP.NET Core полягає в тому, що класи моделі даних визначаються в папці з назвою Models, яка була додана до проекту за допомогою шаблону, використаного в лістингу 3.1.

Мені не потрібна складна модель для проекту PartyInvites, оскільки це такий простий додаток. Мені потрібен лише один доменний клас, який я називатиму GuestResponse. Цей об'єкт представлятиме відповідь на запрошення від запрошеного.

Якщо ви використовуєте Visual Studio, клацніть правою кнопкою миші папку Models і виберіть Add > Class у спливаючому меню. Назвіть клас GuestResponse.cs і натисніть кнопку «Додати». Якщо ви використовуєте Visual Studio Code, клацніть правою кнопкою миші папку Models, виберіть New File і введіть GuestResponse.cs як ім'я файлу. Використовуйте новий файл для визначення класу, показаного в лістингу 3.6.

Лістинг 3.6 Вміст файлу GuestResponse.cs у папці Models

```
namespace PartyInvites.Models {
    public class GuestResponse {
        public string? Name { get; set; }
        public string? Email { get; set; }
        public string? Phone { get; set; }
        public bool? WillAttend { get; set; }
    }
}
```

Зауважте, що всі властивості, визначені класом GuestResponse, мають значення NULL. Я поясню, чому це важливо, у розділі «Додавання перевірки» далі в цьому розділі.

Перезапуск автоматичної збірки

Ви можете побачити попередження, створене командою dotnet watch про те, що гаряче перезавантаження не можна застосувати. Команда dotnet watch не може впоратися з кожним типом змін, а деякі зміни призводять до збою процесу автоматичного відновлення. Ви побачите цю підказку в командному рядку:

```
watch : Do you want to restart your app
- Yes (y) / No (n) / Always (a) / Never (v)?
```

Натисніть a, щоб завжди перебудувати проект. Корпорація Майкрософт часто вдосконалює команду dotnet watch, тому дії, які викликають цю проблему, змінюються.

Створення другої дії та перегляду

Однією з цілей моєї програми є включення форми RSVP, що означає, що мені потрібно визначити метод дії, який може отримувати запити на цю форму. Один клас контролера може визначати кілька методів дій, і прийнято групувати пов'язані дії в одному контролері. Лістинг 3.7 додає новий метод дії до контролера Home. Контролери можуть повертати різні типи результатів, які пояснюються в наступних розділах.

Лістинг 3.7 Додавання дії у файлі HomeController.cs у папці Controllers

```
using Microsoft.AspNetCore.Mvc;
namespace PartyInvites.Controllers {
    public class HomeController : Controller {
        public IActionResult Index() {
            return View();
        }

        public ViewResult RsvpForm() {
            return View();
        }
    }
}
```

Обидва методи дії викликають метод View без аргументів, що може здатися дивним, але пам'ятайте, що механізм перегляду Razor використовуватиме назву методу дії під час пошуку файлу перегляду, як пояснюється в розділі 2. Це означає, що результат з Index методу дії повідомляє Razor шукати представлення під назвою Index.cshtml, тоді як результат від методу дії RsvpForm повідомляє Razor шукати представлення під назвою RsvpForm.cshtml.

Якщо ви використовуєте Visual Studio, клацніть правою кнопкою миші папку Views/Home і виберіть Add > New Item у спливаючому меню. Виберіть елемент «Razor View – Empty», встановіть назву RsvpForm.cshtml і натисніть кнопку «Додати», щоб створити файл. Замініть вміст вмістом, показаним у лістингу 3.8.

Лістинг 3.8 Вміст файлу RsvpForm.cshtml у папці Views/Home

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>RsvpForm</title>
</head>
<body>
    <div>
        This is the RsvpForm.cshtml View
    </div>
</body>
</html>
```

На даний момент цей вміст є просто статичним HTML. Використовуйте браузер для запиту http://localhost:5000/home/rsvpform . Механізм перегляду Razor знаходить файл RsvpForm.cshtml і використовує його для створення відповіді, як показано на Рис. 17.

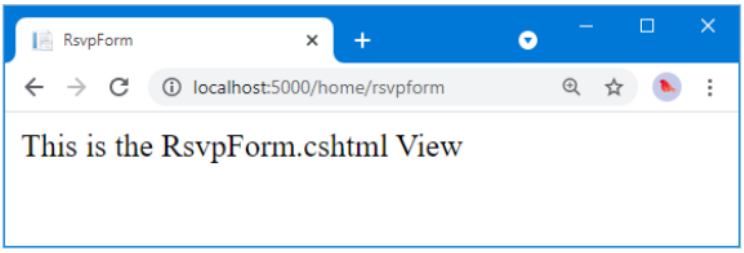

Рис. 17 – Візуалізація другого вигляду

Пов'язка методів дії

Я хочу мати можливість створити посилання з подання Index, щоб гості могли бачити подання RsvpForm без необхідності знати URL-адресу, яка націлена на певний метод дії, як показано в лістингу 3.9.

Лістинг 3.9 Додавання посилання у файл Index.cshtml у папці Views/Home

```
@{
    Layout = null;
}
```

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Party!</title>
</head>
<body>
    <div>
        <div>
            We're going to have an exciting party.<br />
           (To do: sell it better. Add pictures or something.)
        </div>
        <a asp-action="RsvpForm">RSVP Now</a>
   </div>
</body>
</html>
```

Доповненням до списку є елемент, який має атрибут asp-action. Атрибут є прикладом атрибута допоміжного тегу (*tag helper*), який є інструкцією для Razor, яка виконуватиметься під час рендерингу перегляду. Атрибут asp-action — це вказівка додати атрибут href до елемента a, який містить URL-адресу для методу дії.

Я пояснюю, як працюють помічники тегів у розділах 25–27, але цей помічник тегів повідомляє Razor вставити URL-адресу для методу дії, визначеного тим самим контролером, для якого відображається поточне представлення. Використовуйте браузер для запиту http://localhost:5000 , і ви побачите посилання, створене помічником, як показано на Рис. 18.

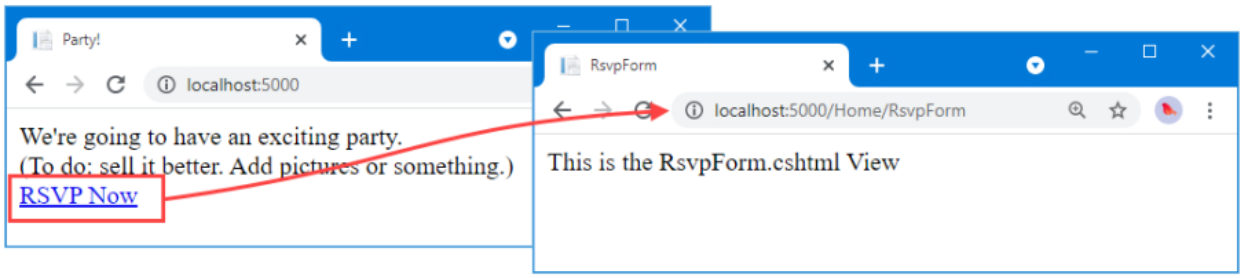

Рис. 18 – Зв'язок між методами дій

Наведіть курсор миші на посилання «RSVP Now, Відповісти зараз» у браузері. Ви побачите, що посилання вказує на http://localhost:5000/Home/RsvpForm.

Тут діє важливий принцип, який полягає в тому, що ви повинні використовувати функції, надані ASP.NET Core, для створення URL-адрес, а не жорстко кодувати їх у своїх представленнях. Коли помічник тегів створив атрибут href для елемента a, він перевірив конфігурацію програми, щоб визначити, якою має бути URL-адреса. Це дозволяє змінювати конфігурацію програми для підтримки різних форматів URL-адрес без необхідності оновлення переглядів.

Побудова форми

Тепер, коли я створив подання та можу отримати доступ до нього з подання Index, я збираюся створити вміст файлу RsvpForm.cshtml, щоб перетворити його на HTML-форму для редагування об'єктів GuestResponse, як показано в лістингу 3.10.

Лістинг 3.10 Створення представлення форми у файлі RsvpForm.cshtml у папці Views/Home

```
@model PartyInvites.Models.GuestResponse
@{
  Layout = null;
}
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>RsvpForm</title>
</head>
<body>
  <form asp-action="RsvpForm" method="post">
    <div>
      <label asp-for="Name">Your name:</label>
      <input asp-for="Name" />
    </div>
    <div>
      <label asp-for="Email">Your email:</label>
      <input asp-for="Email" />
    </div>
    <div>
```

```
      <label asp-for="Phone">Your phone:</label>
      <input asp-for="Phone" />
    </div>
    <div>
      <label asp-for="WillAttend">Will you attend?</label>
      <select asp-for="WillAttend">
        <option value="">Choose an option</option>
        <option value="true">Yes, I'll be there</option>
        <option value="false">No, I can't come</option>
      </select>
    </div>
    <button type="submit">Submit RSVP</button>
  </form>
</body>
</html>
```

Вираз @model вказує, що представлення очікує отримати об'єкт GuestResponse як свою модель перегляду. Я визначив мітку (label) та елемент введення (input) для кожної властивості класу моделі GuestResponse (або, у випадку властивості WillAttend, елемент вибору select). Кожен елемент пов'язується з властивістю моделі за допомогою атрибута asp-for, який є іншим атрибутом допоміжного тегу. Допоміжні атрибути тегів налаштовують елементи, щоб прив'язати їх до об'єкта моделі представлення. Ось приклад HTML, який створюють помічники тегів:

```
<p>
    <label for="Name">Your name:</label>
    <input type="text" id="Name" name="Name" value="">
</p>
```

Атрибут asp-for елемента label встановлює значення атрибута for. Атрибут asp-for елемента input встановлює елементи id і name. Це може здатися не особливо корисним, але ви побачите, що зв'язування елементів із властивістю моделі пропонує додаткові переваги, оскільки функціональність програми визначена.

Більш безпосереднього використання має атрибут asp-action, застосований до елемента форми, який використовує конфігурацію маршрутизації URL-адреси програми, щоб встановити атрибут action на URL-адресу, яка буде націлена на певний метод дії, наприклад:

```
<form method="post" action="/Home/RsvpForm">
```

Як і у випадку з допоміжним атрибутом, який я застосував до елемента a, перевага цього підходу полягає в тому, що коли ви змінюєте систему URL-адрес, яку використовує програма, вміст, створений допоміжними тегами, автоматично відображатиме ці зміни.

Використовуйте браузер для запиту http://localhost:5000 і клацніть посилання RSVP Now, щоб побачити форму, як показано на Рис. 19.

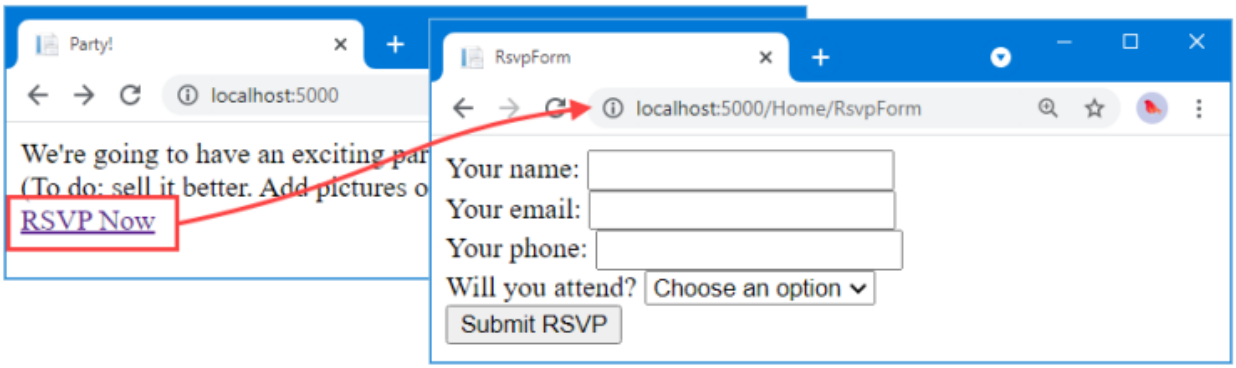

Рис. 19 – Додавання HTML-форми до програми

Отримання даних форми

Я ще не сказав ASP.NET Core, що я хочу робити, коли форму буде опубліковано на сервері. У поточному стані натискання кнопки «Submit RSV, Надіслати відповідь» просто видаляє всі значення, які ви ввели у форму. Це пов'язано з тим, що форма відправляє повідомлення назад до методу дії RsvpForm у контролері Home, який просто знову відображає перегляд. Для отримання та обробки надісланих даних форми я збираюся використовувати важливу функцію контролерів. Я додам другий метод дії RsvpForm, щоб створити наступне:

- Метод, який відповідає на HTTP-запити GET. Запит GET – це те, що зазвичай видає браузер щоразу, коли хтось натискає посилання. Ця версія дії відповідатиме за відображення початкової порожньої форми під час першого відвідування /Home/RsvpForm.
- Метод, який відповідає на запити HTTP POST: Елемент форми, визначений у лістингу 3.10, встановлює атрибут методу на post, що змушує дані форми надсилатися на сервер як запит POST. Ця версія дії буде відповідати за отримання надісланих даних і рішення, що з ними робити.

Обробка запитів GET і POST окремими методами C# допомагає підтримувати порядок у коді контролера, оскільки ці два методи мають різні обов'язки. Обидва методи дії викликаються однією URL-адресою, але ASP.NET Core забезпечує виклик відповідного методу залежно від того, із запитом GET чи POST я маю справу. У лістингу 3.11 показано зміни в класі HomeController.

Лістинг 3.11 Додавання методу у файл HomeController.cs у папці Controllers

```
using Microsoft.AspNetCore.Mvc;
using PartyInvites.Models;
namespace PartyInvites.Controllers {
  public class HomeController : Controller {
    public IActionResult Index() {
      return View();
    }
    [HttpGet]
    public ViewResult RsvpForm() {
      return View();
    }
    [HttpPost]
    public ViewResult RsvpForm(GuestResponse guestResponse) {
      // TODO: store response from guest
      return View();
    }
  }
}
```

Я додав атрибут HttpGet до існуючого методу дії RsvpForm, який заявляє, що цей метод слід використовувати лише для запитів GET. Потім я додав перевантажену версію методу RsvpForm, яка приймає об'єкт GuestResponse. Я застосував атрибут HttpPost до цього методу, який оголошує, що він оброблятиме запити POST. У наступних розділах я поясню, як працюють ці доповнення до списку. Я також імпортував простір імен PartyInvites.Models — це лише для того, щоб я міг посилатися на тип моделі GuestResponse без необхідності кваліфікувати назву класу.

Розуміння прив'язки моделі

Перше перевантаження методу дії RsvpForm відтворює те саме подання, що й раніше — файл RsvpForm.cshtml — для створення форми, показаної на

Рис. 19. Друге перевантаження більш цікаве через параметр, але враховуючи, що метод дії буде викликано у відповідь на запит HTTP POST і що тип GuestResponse є класом C#, як вони пов'язані?

Відповідь полягає в *зв'язуванні моделі* (model binding), корисній функції ASP.NET Core, за допомогою якої вхідні дані аналізуються, а пари ключ-значення в запиті HTTP використовуються для заповнення властивостей типів моделі домену.

Зв'язування моделі — це потужна та настроювана функція, яка усуває складність безпосередньої обробки HTTP-запитів і дозволяє працювати з об'єктами C#, а не працювати з окремими значеннями даних, які надсилає браузер. Об'єкт GuestResponse, який передається як параметр методу дії, автоматично заповнюється даними з полів форми. Я заглиблюся в подробиці прив'язки моделі в розділі 28.

Щоб продемонструвати, як працює зв'язування моделі, мені потрібно виконати певну підготовчу роботу. Однією з цілей програми є надання сторінки підсумків із детальною інформацією про відвідувачів вечірки, що означає, що мені потрібно відстежувати відповіді, які я отримую. Я збираюся зробити це, створивши в пам'яті колекцію об'єктів. Це не корисно в реальній програмі, оскільки дані відповіді буде втрачено, коли програму зупинено або перезапущено, але цей підхід дозволить мені зосередитися на ASP.NET Core і створити програму, яку можна легко скинути до її початкового стану. У наступних розділах буде продемонстровано постійне зберігання даних.

Додайте файл класу під назвою Repository.cs до папки Models і використовуйте його для визначення класу, показаного в лістингу 3.12.

Лістинг 3.12 Вміст файлу Repository.cs у папці Models

```
namespace PartyInvites.Models {
  public static class Repository {
    private static List<GuestResponse> responses = new();
    public static IEnumerable<GuestResponse> Responses =>
responses;
    public static void AddResponse(GuestResponse response) {
      Console.WriteLine(response);
      responses.Add(response);
    }
  }
}
```

Клас Repository та його члени є статичними, що полегшить мені зберігання та отримання даних із різних місць у програмі. ASP.NET Core пропонує більш складний підхід для визначення загальної функціональності, що називається ін'єкцією залежностей, яку я описую в розділі 14, але статичний клас є хорошим способом почати роботу для простої програми, як ця.

Якщо ви використовуєте Visual Studio, збереження вмісту файлу Repository.cs викличе попередження, створене командою dotnet watch про те, що гаряче перезавантаження не можна застосувати. Ви побачите цю підказку в командному рядку:

```
watch : Do you want to restart your app
- Yes (y) / No (n) / Always (a) / Never (v)?
```

Натисніть a, щоб завжди перебудовувати проект.

Зберігання відповідей

Тепер, коли у мене є де зберігати дані, я можу оновити метод дії, який отримує запити HTTP POST, як показано в лістингу 3.13.

Лістинг 3.13 Оновлення дії у файлі HomeController.cs у папці Controllers

```
using Microsoft.AspNetCore.Mvc;
using PartyInvites.Models;
namespace PartyInvites.Controllers {
  public class HomeController : Controller {
    public IActionResult Index() {
      return View();
    }
    [HttpGet]
    public ViewResult RsvpForm() {
      return View();
    }
    [HttpPost]
    public ViewResult RsvpForm(GuestResponse guestResponse) {
      Repository.AddResponse(guestResponse);
      return View("Thanks", guestResponse);
    }
  }
}
```

Перед викликом POST-версії методу RsvpForm функція зв’язування моделі ASP.NET Core витягує значення з форми HTML і призначає їх властивостям об’єкта GuestResponse. Результат використовується як аргумент, коли метод викликається для обробки HTTP-запиту, і все, що мені потрібно зробити, щоб працювати з даними форми, надісланими в запиті, це працювати з об’єктом GuestResponse, який передається в метод дії — у цьому випадку, щоб передати його як аргумент методу Repository.AddResponse, щоб можна було зберегти відповідь.

Додавання перегляду подяки

Виклик методу View у методі дії RsvpForm створює ViewResult, який вибирає представлення під назвою Thanks і використовує об’єкт GuestResponse, створений зв’язувачем моделі, як модель перегляду. Додайте Razor View із назвою Thanks.cshtml до папки Views/Home із вмістом, показаним у лістингу 3.14, щоб надати відповідь користувачеві.

Лістинг 3.14 Вміст файлу Thanks.cshtml у папці Views/Home

```
@model PartyInvites.Models.GuestResponse
@{
  Layout = null;
}
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Thanks</title>
</head>
<body>
  <div>
    <h1>Thank you, @Model?.Name!</h1>
    @if (Model?.WillAttend == true) {
      @:It's great that you're coming.
      @:The drinks are already in the fridge!
    } else {
      @:Sorry to hear that you can't make it,
      @:but thanks for letting us know.
    }
  </div>
  Click <a asp-action="ListResponses">here</a> to see who is
coming.
```

```
</body>
</html>
```

HTML-код, створений поданням Thanks.cshtml, залежить від значень, призначених моделі подання GuestResponse, яку надає метод дії RsvpForm. Щоб отримати доступ до значення властивості в об'єкті домену, я використовую вираз @Model.<PropertyName>. Тому, наприклад, щоб отримати значення властивості Name, я використовую вираз @Model.Name. Не хвилюйтеся, якщо синтаксис Razor не має сенсу — я поясню це більш детально в розділі 21.

Тепер, коли я створив подання подяки, у мене є базовий робочий приклад роботи з формою. Використовуйте браузер, щоб надіслати запит http://localhost:5000 , натисніть посилання «RSVP Now, Відповісти зараз», додайте деякі дані до форми та натисніть кнопку «Submit RSVP». Ви побачите відповідь, показану на Рис. 20 (хоча вона буде іншою, якщо вас не звуть Джо або ви сказали, що не можете бути присутніми).

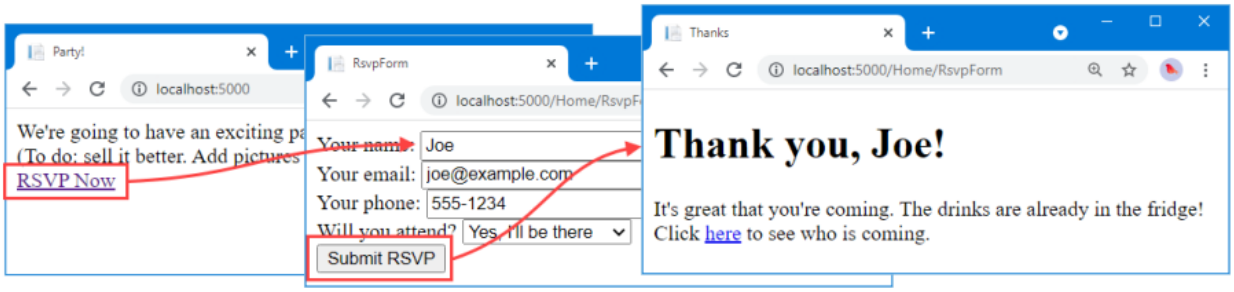

Рис. 20 – Перегляд подяки

Відображення відповідей

У кінці перегляду Thanks.cshtml я додав елемент a для створення посилання для відображення списку людей, які прийдуть на вечірку. Я використовував допоміжний атрибут тегу asp-action для створення URL-адреси, націленої на метод дії під назвою ListResponses, наприклад:

```
...
Click <a asp-action="ListResponses">here</a> to see who is
coming.
...
```

Якщо ви наведете курсор миші на посилання, яке відображається в браузері, ви побачите, що воно націлено на URL-адресу /Home/ListResponses.

Це не відповідає жодному з методів дій у домашньому контролері, і якщо ви клацнете посилання, ви побачите відповідь про помилку 404 Not Found.

Щоб додати кінцеву точку, яка оброблятиме URL-адресу, мені потрібно додати інший метод дії до контролера Home, як показано в лістингу 3.15.

Лістинг 3.15 Додавання дії у файлі HomeController.cs у папці Controllers

```
using Microsoft.AspNetCore.Mvc;
using PartyInvites.Models;
namespace PartyInvites.Controllers {
  public class HomeController : Controller {
    public IActionResult Index() {
      return View();
    }
    [HttpGet]
    public ViewResult RsvpForm() {
      return View();
    }
    [HttpPost]
    public ViewResult RsvpForm(GuestResponse guestResponse) {
      Repository.AddResponse(guestResponse);
      return View("Thanks", guestResponse);
    }
    public ViewResult ListResponses() {
      return View(Repository.Responses
          .Where(r => r.WillAttend == true));
    }
  }
}
```

Новий метод дії називається ListResponses, і він викликає метод View, використовуючи властивість Repository.Responses як аргумент. Це призведе до того, що Razor відтворить типовий вигляд, використовуючи ім’я методу дії як ім’я файлу перегляду, і використає дані зі сховища як модель перегляду. Дані моделі подання фільтруються за допомогою LINQ, щоб на подання надходили лише позитивні відповіді.

Додайте представлення Razor під назвою ListResponses.cshtml до папки Views/Home із вмістом, показаним у лістингу 3.16.

Лістинг 3.16 Відображення даних у файлі ListResponses.cshtml у папці Views/Home

```
@model IEnumerable<PartyInvites.Models.GuestResponse>
@{
  Layout = null;
}
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Responses</title>
</head>
<body>
  <h2>Here is the list of people attending the party</h2>
  <table>
    <thead>
      <tr><th>Name</th><th>Email</th><th>Phone</th></tr>
    </thead>
    <tbody>
      @foreach (PartyInvites.Models.GuestResponse r in Model!) {
        <tr>
            <td>@r.Name</td>
            <td>@r.Email</td>
            <td>@r.Phone</td>
        </tr>
      }
    </tbody>
  </table>
</body>
</html>
```

Файли перегляду Razor мають розширення файлу .cshtml для позначення суміші коду C# та елементів HTML. Ви можете побачити це в лістингу 3.16, де я використовував вираз @foreach для обробки кожного з об'єктів GuestResponse, які метод дії передає в представлення за допомогою методу View. На відміну від звичайного циклу foreach C#, тіло виразу @foreach у Razor містить елементи HTML, які додаються до відповіді, яка буде надіслана назад у браузер. У цьому поданні кожен об'єкт GuestResponse генерує елемент tr, який містить елементи td, заповнені значенням властивості об'єкта.

Використовуйте браузер, щоб надіслати запит http://localhost:5000 , натисніть посилання «RSVP Now» і заповніть форму. Надішліть форму, а потім клацніть посилання, щоб переглянути зведення даних, які було введено з моменту запуску програми, як показано на Рис. 21. Подання не представляє

даних у привабливий спосіб, але на даний момент цього достатньо, і я розповім про стиль програми пізніше в цьому розділі.

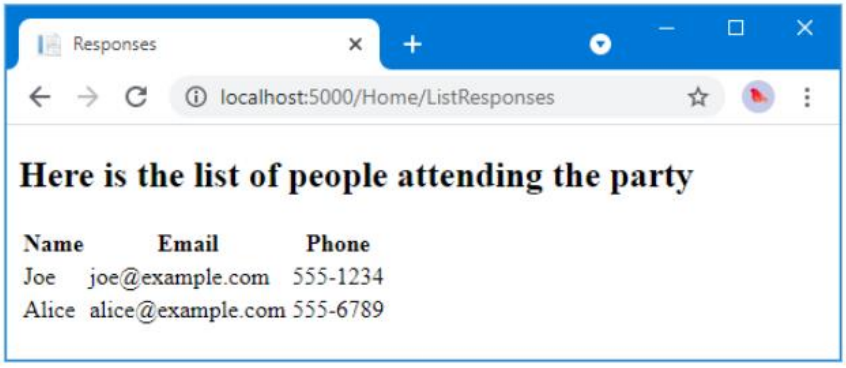

Рис. 21 – Показ списку учасників вечірки

Додавання перевірки

Тепер я можу додати перевірку даних до програми. Без перевірки користувачі можуть вводити безглузді дані або навіть надсилати порожню форму. У програмі ASP.NET Core правила перевірки визначаються шляхом застосування атрибутів до класів моделі, що означає, що ті самі правила перевірки можна застосовувати в будь-якій формі, яка використовує цей клас. ASP.NET Core покладається на атрибути з простору імен System.ComponentModel.DataAnnotations, які я застосував до класу GuestResponse в лістингу 3.17.

Лістинг 3.17 Застосування перевірки у файлі GuestResponse.cs у папці Models

```
using System.ComponentModel.DataAnnotations;
namespace PartyInvites.Models {
  public class GuestResponse {
    [Required(ErrorMessage = "Please enter your name")]
    public string? Name { get; set; }
    [Required(ErrorMessage = "Please enter your email address")]
    [EmailAddress]
    public string? Email { get; set; }
    [Required(ErrorMessage = "Please enter your phone number")]
    public string? Phone { get; set; }
    [Required(ErrorMessage = "Please specify whether you'll
attend")]
    public bool? WillAttend { get; set; }
```

```
  }
}
```

ASP.NET Core виявляє атрибути та використовує їх для перевірки даних під час процесу прив'язки моделі. Як зазначалося раніше, для визначення властивостей GuestResponse я використовував типи, що допускають значення NULL. Це корисно для позначення властивостей, яким не можуть бути призначені значення, але воно має спеціальне значення для властивості WillAttend, оскільки дозволяє працювати атрибуту Required validation. Якби я використовував звичайний логічний тип, який не допускає нуль, значення, отримане через прив'язку моделі, могло б бути лише істинним або хибним, і я б не зміг визначити, чи вибрав користувач значення. Тип nullable bool із можливістю обнулення має три можливі значення: істина, хибність і нуль. Значення властивості WillAttend буде нульовим, якщо користувач не вибрав значення, і через це атрибут Required повідомляє про помилку перевірки. Це гарний приклад того, як ASP.NET Core елегантно поєднує функції C# з HTML і HTTP.

Я перевіряю, чи виникла проблема перевірки, використовуючи властивість ModelState.IsValid у методі дії, який отримує дані форми, як показано в лістингу 3.18.

Лістинг 3.18 Перевірка на наявність помилок у файлі HomeController.cs у папці Controllers

```
using Microsoft.AspNetCore.Mvc;
using PartyInvites.Models;
namespace PartyInvites.Controllers {
  public class HomeController : Controller {
    public IActionResult Index() {
      return View();
    }
  [HttpGet]
  public ViewResult RsvpForm() {
    return View();
  }
  [HttpPost]
  public ViewResult RsvpForm(GuestResponse guestResponse) {
    if (ModelState.IsValid) {
      Repository.AddResponse(guestResponse);
      return View("Thanks", guestResponse);
    } else {
        return View();
```

```
    }
  }
  public ViewResult ListResponses() {
    return View(Repository.Responses
        .Where(r => r.WillAttend == true));
    }
  }
}
```

Базовий клас Controller надає властивість під назвою ModelState, яка надає деталі результату процесу зв'язування моделі. Якщо властивість ModelState.IsValid повертає значення true, тоді я знаю, що прив'язка моделі змогла задовольнити обмеження перевірки, які я вказав через атрибути класу GuestResponse. Коли це відбувається, я відображаю подяку, як і раніше.

Якщо властивість ModelState.IsValid повертає false, я знаю, що є помилки перевірки. Об'єкт, який повертає властивість ModelState, надає детальну інформацію про кожну проблему, з якою зіткнувся, але мені не потрібно вникати в цей рівень деталей, оскільки я можу покластися на корисну функцію, яка автоматизує процес запиту користувача про вирішення будь-якої проблеми шляхом виклику методу View без будь-яких параметрів.

Під час рендерингу перегляду Razor має доступ до деталей будь-яких помилок перевірки, пов'язаних із запитом, а помічники тегів можуть отримати доступ до деталей, щоб відобразити користувачеві помилки перевірки. У лістингу 3.19 показано додавання допоміжних атрибутів тегу перевірки до перегляду RsvpForm.

Лістинг 3.19 Додавання підсумку до файлу RsvpForm.cshtml у папці Views/Home

```
@model PartyInvites.Models.GuestResponse
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>RsvpForm</title>
</head>
<body>
    <form asp-action="RsvpForm" method="post">
```

```
      <div asp-validation-summary="All"></div>
      <div>
        <label asp-for="Name">Your name:</label>
        <input asp-for="Name" />
      </div>
      <div>
        <label asp-for="Email">Your email:</label>
        <input asp-for="Email" />
      </div>
      <div>
        <label asp-for="Phone">Your phone:</label>
        <input asp-for="Phone" />
      </div>
      <div>
        <label asp-for="WillAttend">Will you attend?</label>
        <select asp-for="WillAttend">
            <option value="">Choose an option</option>
            <option value="true">Yes, I'll be there</option>
            <option value="false">No, I can't come</option>
        </select>
      </div>
      <button type="submit">Submit RSVP</button>
    </form>
</body>
</html>
```

Атрибут asp-validation-summary застосовується до елемента div і відображає список помилок перевірки під час відтворення подання. Значення для атрибута asp-validation-summary — це значення з переліку під назвою ValidationSummary, яке вказує, які типи помилок підтвердження міститиме зведення. Я вказав All, що є гарною відправною точкою для більшості програм, і я описую інші значення та поясню, як вони працюють у розділі 29.

Щоб побачити, як працює підсумок перевірки, запустіть програму, заповніть поле Ім’я та надішліть форму, не вводячи жодних інших даних. Ви побачите підсумок помилок перевірки, як показано на Рис. 22.

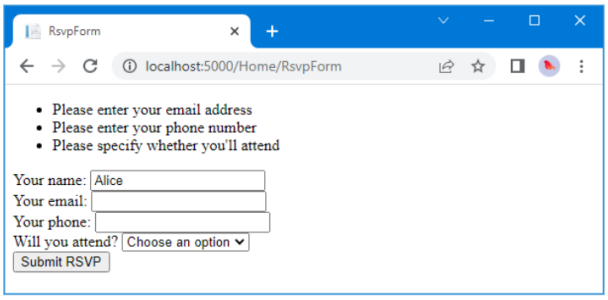

Рис. 22 – Відображення помилок перевірки

Метод дії RsvpForm не відображатиме подання «Дякую», доки не буде виконано всі обмеження перевірки, застосовані до класу GuestResponse. Зауважте, що дані, введені в поле «Ім'я», було збережено та відображено знову, коли Razor відтворив подання з підсумком перевірки. Це ще одна перевага зв'язування моделі, яка спрощує роботу з даними форми.

Виділення недійсних полів

Допоміжні атрибути тегів, які пов'язують властивості моделі з елементами, мають зручну функцію, яку можна використовувати разом із прив'язкою моделі. Якщо властивість класу моделі не пройшла перевірку, допоміжні атрибути створять дещо інший HTML. Ось елемент введення, який створюється для поля Телефон, коли немає помилки перевірки:

```
<input type="text" data-val="true"
   data-val-required="Please enter your phone number" id="Phone"
   name="Phone" value="">
```

Для порівняння, ось той самий елемент HTML після того, як користувач надіслав форму без введення даних у текстове поле (що є помилкою перевірки, оскільки я застосував атрибут Required до властивості Phone класу GuestResponse):

```
<input type="text" class="input-validation-error"
data-val="true" data-val-required="Please enter your phone
number" id="Phone"
name="Phone" value="">
```

Я підкреслив різницю: допоміжний атрибут тегу asp-for додав елемент input до класу під назвою input-validation-error. Я можу скористатися цією функцією, створивши таблицю стилів, яка містить стилі CSS для цього класу та інших, які використовують різні допоміжні атрибути HTML.

Угода в проектах ASP.NET Core полягає в тому, що статичний вміст розміщується в папці wwwroot і впорядковується за типом вмісту так, що таблиці стилів CSS потрапляють у папку wwwroot/css, файли JavaScript — у папку wwwroot/js тощо.

ПІДКАЗКА Шаблон проекту, використаний у лістингу 3.1, створює файл site.css у папці wwwroot/css. Ви можете ігнорувати цей файл, який я не використовую в цій главі.

Якщо ви використовуєте Visual Studio, клацніть правою кнопкою миші папку wwwroot/css і виберіть Додати > Новий елемент у спливаючому меню. Знайдіть шаблон елемента таблиці стилів, як показано на Рис. 23; встановіть ім'я файлу styles.css; і натисніть кнопку Додати.

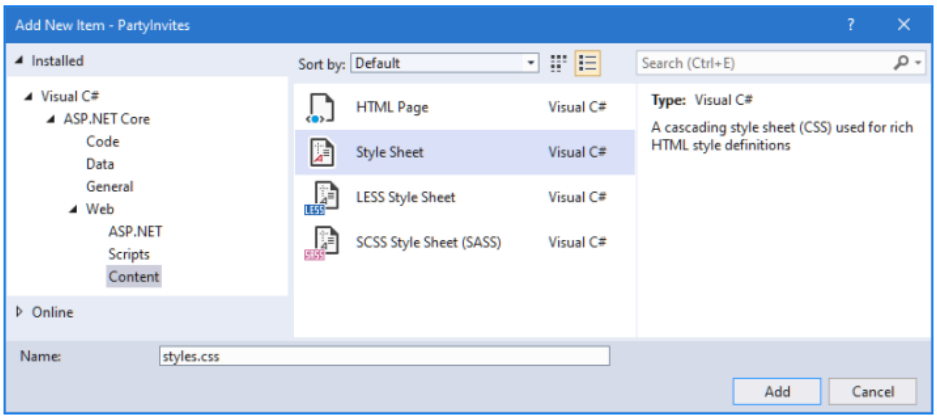

Рис. 23 – Створення таблиці стилів CSS

Замініть вміст файлу стилями, показаними в лістингу 3.20.

Лістинг 3.20 Вміст файлу styles.css у папці wwwroot/css

```
.field-validation-error {
    color: #f00;
}
.field-validation-valid {
    display: none;
}
.input-validation-error {
    border: 1px solid #f00;
    background-color: #fee;
}
.validation-summary-errors {
    font-weight: bold;
    color: #f00;
```

```
}
.validation-summary-valid {
    display: none;
}
```

Щоб застосувати цю таблицю стилів, я додав елемент посилання до головного розділу перегляду RsvpForm, як показано в лістингу 3.21.

Лістинг 3.21 Застосування таблиці стилів у файлі RsvpForm.cshtml у папці Views/Home

```
...
<head>
    <meta name="viewport" content="width=device-width" />
    <title>RsvpForm</title>
    <link rel="stylesheet" href="/css/styles.css" />
</head>
...
```

Елемент посилання використовує атрибут href для визначення розташування таблиці стилів. Зверніть увагу, що в URL-адресі пропущено папку wwwroot. Конфігурація за замовчуванням для ASP.NET включає підтримку обслуговування статичного вмісту, наприклад зображень, таблиць стилів CSS і файлів JavaScript, і автоматично відображає запити на папку wwwroot. Із застосуванням таблиці стилів більш очевидна помилка перевірки відображатиметься, коли надсилатимуться дані, які викликають помилку перевірки, як показано на Рис. 24.

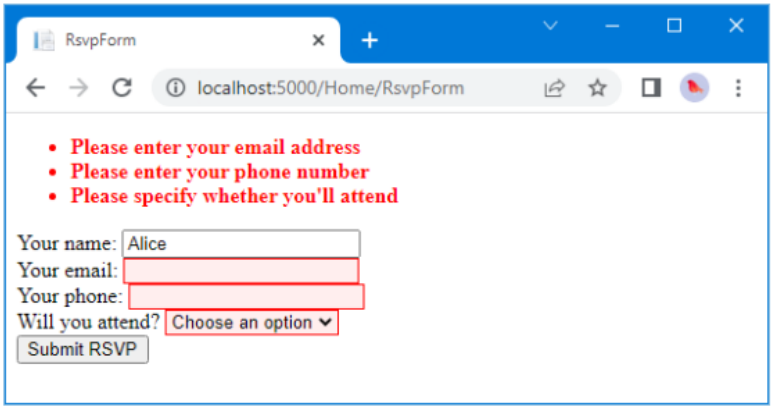

Рис. 24 – Автоматично підсвічуються помилки перевірки

Стилізація контенту

Усі функціональні цілі програми виконані, але загальний вигляд програми поганий. Коли ви створюєте проект за допомогою шаблону mvc, як я робив для прикладу в цій главі, інсталюються деякі звичайні пакети розробки на стороні клієнта. Хоча я не прихильник використання шаблонних проектів, мені подобаються клієнтські бібліотеки, які обрала Microsoft. Один із них називається Bootstrap, який є хорошим фреймворком CSS, спочатку розробленим Twitter, який став великим проектом з відкритим вихідним кодом і основою розробки веб-додатків.

Стилізуйте вітальний вигляд

Основні функції Bootstrap працюють шляхом застосування класів до елементів, які відповідають селекторам CSS, визначеним у файлах, доданих до папки wwwroot/lib/bootstrap. Ви можете отримати повну інформацію про класи, які визначає Bootstrap, на http://getbootstrap.com , але ви можете побачити, як я застосував деякі базові стилі до файлу перегляду Index.cshtml у лістингу 3.22.

Лістинг 3.22 Додавання Bootstrap до файлу Index.cshtml у папці Views/Home

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <link rel="stylesheet"
href="/lib/bootstrap/dist/css/bootstrap.css" />
  <title>Index</title>
</head>
<body>
    <div class="text-center m-2">
      <h3> We're going to have an exciting party!</h3>
      <h4>And YOU are invited!</h4>
      <a class="btn btn-primary" asp-action="RsvpForm">RSVP
Now</a>
    </div>
```

```
</body>
</html>
```

Я додав елемент посилання, атрибут href якого завантажує файл bootstrap.css із папки wwwroot/lib/bootstrap/dist/css. Угодою є те, що сторонні пакети CSS і JavaScript встановлюються в папку wwwroot/lib, і я описую інструмент, який використовується для керування цими пакетами, у розділі 4.

Після імпорту таблиць стилів Bootstrap мені потрібно стилізувати свої елементи. Це простий приклад, тому мені потрібно використовувати лише невелику кількість класів CSS Bootstrap: textcenter, btn і btn-primary.

Клас text-center центрує вміст елемента та його дітей. Клас btn стилізує кнопку, вхід або елемент як красиву кнопку, а клас btn-primary вказує, якого з діапазону кольорів я хочу, щоб кнопка була. Ви можете побачити ефект, запустивши програму, як показано на Рис. 25.

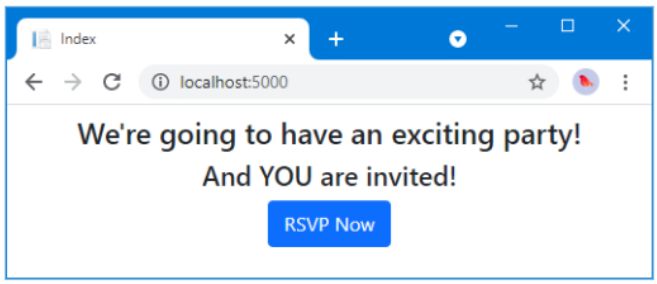

Рис. 25 – Стилізація вигляду

Стилізуйте вигляд форми

Bootstrap визначає класи, які можна використовувати для стилізації форм. Я не збираюся вдаватися в подробиці, але ви можете побачити, як я застосував ці класи в лістингу 3.23.

Лістинг 3.23 Додавання стилів до файлу RsvpForm.cshtml у папці Views/Home

```
@model PartyInvites.Models.GuestResponse
@{
  Layout = null;
}
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
```

```
  <title>RsvpForm</title>
  <link rel="stylesheet"
href="/lib/bootstrap/dist/css/bootstrap.css" />
  <link rel="stylesheet" href="/css/styles.css" />
</head>
<body>
  <h5 class="bg-primary text-white text-center m-2 p-
2">RSVP</h5>
  <form asp-action="RsvpForm" method="post" class="m-2">
    <div asp-validation-summary="All"></div>
      <div class="form-group">
        <label asp-for="Name" class="form-label">Your
name:</label>
        <input asp-for="Name" class="form-control" />
      </div>
      <div class="form-group">
        <label asp-for="Email" class="form-label">Your
email:</label>
        <input asp-for="Email" class="form-control" />
      </div>
      <div class="form-group">
        <label asp-for="Phone" class="form-label">Your
phone:</label>
        <input asp-for="Phone" class="form-control" />
      </div>
      <div class="form-group">
        <label asp-for="WillAttend" class="form-label">
           Will you attend?
        </label>
        <select asp-for="WillAttend" class="form-select">
          <option value="">Choose an option</option>
          <option value="true">Yes, I'll be there</option>
          <option value="false">No, I can't come</option>
        </select>
      </div>
      <button type="submit" class="btn btn-primary mt-3">
        Submit RSVP
      </button>
  </form>
</body>
</html>
```

Класи Bootstrap у цьому прикладі створюють заголовок лише для надання структури макету. Для стилізації форми я використовував клас form-

group, який використовується для стилізації елемента, який містить мітку та пов’язаний елемент введення або вибору, призначений класу form-control. Ви можете побачити вплив стилів на Рис. 26.

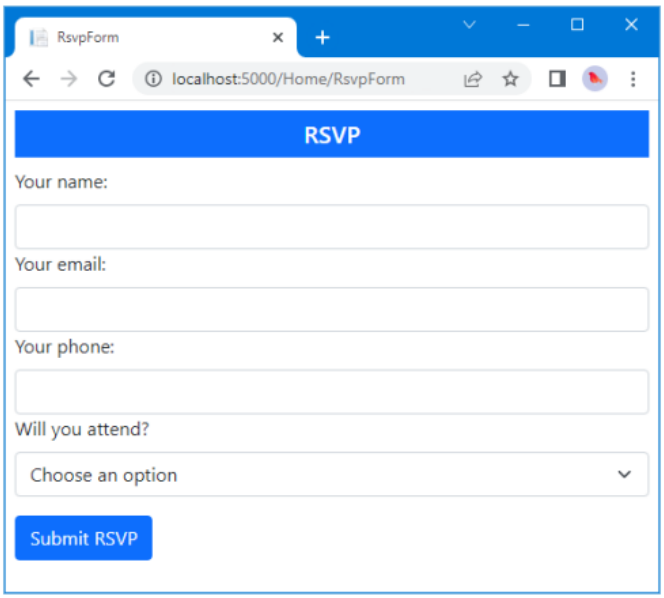

Рис. 26 – Стилізуйте вигляд RsvpForm

Стилізація вигляду подяки

Наступний файл перегляду для стилізації — Thanks.cshtml, і ви можете побачити, як я це зробив у лістингу 3.24, використовуючи класи CSS, подібні до тих, які я використовував для інших переглядів. Щоб полегшити керування програмою, радимо уникати дублювання коду та розмітки, де це можливо. ASP.NET Core надає кілька функцій, які допомагають зменшити дублювання, про які я опишу в наступних розділах. Ці функції включають макети Razor (розділ 22), часткові види (розділ 22) і компоненти перегляду (розділ 24).

Лістинг 3.24 Застосування стилів до файлу Thanks.cshtml у папці Views/Home

```
@model PartyInvites.Models.GuestResponse
@{
  Layout = null;
}
<!DOCTYPE html>
<html>
<head>
```

```
  <meta name="viewport" content="width=device-width" />
  <title>Thanks</title>
  <link rel="stylesheet"
href="/lib/bootstrap/dist/css/bootstrap.css" />
</head>
<body class="text-center">
  <div>
    <h1>Thank you, @Model?.Name!</h1>
    @if (Model?.WillAttend == true) {
      @:It's great that you're coming.
      @:The drinks are already in the fridge!
    } else {
      @:Sorry to hear that you can't make it,
      @:but thanks for letting us know.
    }
  </div>
  Click <a asp-action="ListResponses">here</a> to see who is
coming.
</body>
</html>
```

На Рис. 27 показано дію стилів.

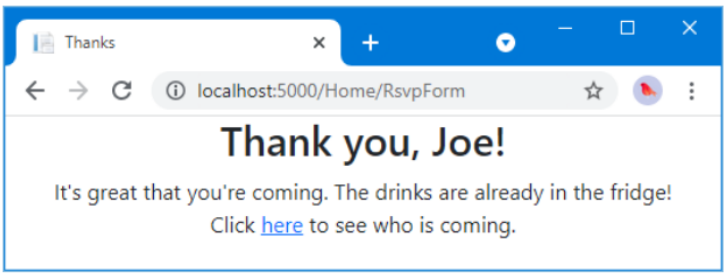

Рис. 27 – Стилізуйте режим подяки

Стилізуйте вигляд списку

Останнім видом стилю є ListResponses, який представляє список учасників. Стилізація вмісту дотримується того самого підходу, який використовується для інших переглядів, як показано в лістингу 3.25.

Лістинг 3.25 Додавання стилів до файлу ListResponses.cshtml у папці Views/Home

```
@model IEnumerable<PartyInvites.Models.GuestResponse>
@{
  Layout = null;
```

```
}
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Responses</title>
  <link rel="stylesheet"
href="/lib/bootstrap/dist/css/bootstrap.css" />
</head>
<body>
  <div class="text-center p-2">
    <h2 class="text-center">
      Here is the list of people attending the party
    </h2>
    <table class="table table-bordered table-striped table-sm">
      <thead>
        <tr><th>Name</th><th>Email</th><th>Phone</th></tr>
      </thead>
      <tbody>
     @foreach (PartyInvites.Models.GuestResponse r in Model!)  {
        <tr>
          <td>@r.Name</td>
          <td>@r.Email</td>
          <td>@r.Phone</td>
        </tr>
      }
       </tbody>
      </table>
    </div>
</body>
</html>
```

На Рис. 28 показано спосіб представлення таблиці учасників. Додавання цих стилів до подання завершує приклад програми, яка тепер відповідає всім цілям розробки та має покращений зовнішній вигляд.

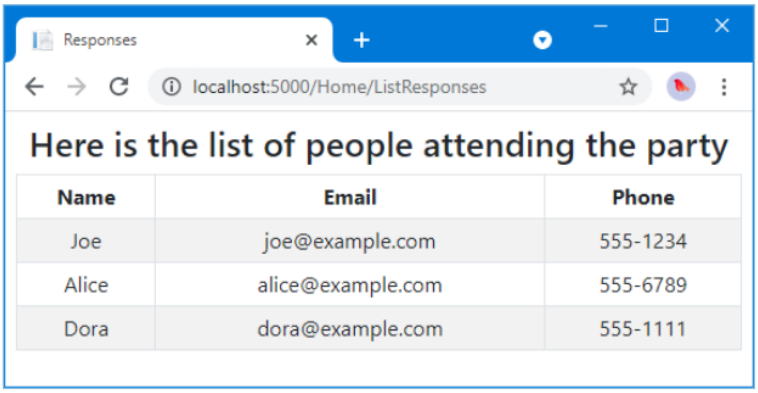

Рис. 28 – Стилізуйте представлення ListResponses

Резюме

Проекти ASP.NET Core створюються за допомогою команди dotnet new.

Контролери визначають методи дій, які використовуються для обробки запитів HTTP.

Представлення створюють вміст HTML, який використовується для відповіді на запити HTTP.

Представлення можуть містити елементи HTML, прив'язані до властивостей моделі даних.

Зв'язування моделі — це процес, за допомогою якого дані запиту аналізуються та призначаються властивостям об'єктів, які передаються методам дії для обробки.

Дані в запиті можуть піддаватися перевірці, а помилки можуть відображатися користувачеві в тій самій формі HTML, яка використовувалася для надсилання даних.

Вміст HTML, створений представленнями, можна стилізувати за допомогою тих самих функцій CSS, які застосовуються до статичного вмісту HTML.