Parte I

Primi passi con ASP.NET Core

Le applicazioni web sono ovunque di questi tempi, dai social media ai siti di notizie alle app dello smartphone. Dietro le quinte, c'è quasi sempre un server sul quale è in funzione un'applicazione web o un'API HTTP. Le applicazioni web sono scalabili indefinitamente, distribuibili via cloud e molto performanti. Un'introduzione all'argomento può essere davvero corposa, e date le elevate aspettative, la sfida sembra quasi impossibile.

La buona notizia è che ASP.NET Core è stato progettato per rispondere a tutti questi requisiti. Che dobbiate creare un semplice sito web, una complessa web app di e-commerce o un web distribuito di microservizi, potrete impiegare ASP.NET Core per realizzare ottime web app adatte alle vostre esigenze. ASP.NET Core vi consente di creare ed eseguire web app in Windows, Linux o macOS. È anche molto modulare, permettendo di impiegare i soli componenti necessari, aspetto che rende l'app sempre la più compatta e performante possibile.

Nella Parte I partiremo dall'inizio e ci impegneremo nella realizzazione di applicazioni web e API. Il Capitolo 1 offre una panoramica di alto livello di ASP.NET Core, che troverete particolarmente utile se non avete una grande esperienza nello sviluppo web in generale. Nel Capitolo 2 osserverete una completa applicazione ASP.NET Core e questo ci darà modo di descrivere ciascun componente dell'app per vedere come collaborano per generare una risposta. Il Capitolo 3 tratta in dettaglio la pipeline del middleware, che definisce come vengono elaborate le richieste web in arrivo e come viene generata una risposta. Osserveremo vari elementi standard del

In questa parte

middleware e vedremo come si colloca il framework Razor Pages nella pipeline di lavoro. Nei Capitoli da 4 a 8 ci concentreremo su Razor Pages, che rappresenta l'approccio principale alla generazione di risposte nelle app ASP.NET Core. Nei Capitoli da 4 a 6 esamineremo il comportamento del framework Razor Pages, del routing e dell'impiego di modelli. Nei Capitoli 7 e 8 osserveremo come realizzare l'interfaccia utente per l'applicazione impiegando la sintassi Razor e i Tag Helper, per permettere agli utenti di navigare e interagire con l'app. Infine, nel Capitolo 9 esploreremo specifiche funzionalità di ASP.NET Core che consentono di realizzare API web e vedremo come ciò differisce dalla programmazione di applicazioni basate su un'interfaccia utente.

La Parte I è ricca di contenuti, ma entro la fine sarete in grado di realizzare semplici applicazioni con ASP.NET Core. Inevitabilmente, dovrò sorvolare su alcuni degli aspetti più complessi della configurazione del framework, ma dovreste riuscire ad avere una comprensione sufficiente del framework Razor Pages e di come impiegarlo per realizzare web app dinamiche. Nelle parti successive di questo libro approfondiremo il framework ASP.NET Core, e così imparerete a configurare l'applicazione e ad aggiungerle altre funzionalità, come i profili utente.

Capitolo 1

Iniziare a lavorare con ASP.NET Core

Scegliere di studiare e poi di iniziare a sviluppare con un nuovo framework è un grosso investimento; pertanto, è importante stabilire presto se questa è la strada giusta. In questo capitolo, vi fornisco alcune informazioni di base su ASP.NET Core: che cos'è, come funziona e perché dovreste considerarlo per la realizzazione delle vostre applicazioni web.

Se siete alle prime armi nel campo dello sviluppo in .NET, questo capitolo vi aiuterà a comprendere il mondo .NET. Per chi, invece, è già uno sviluppatore .NET, fornirò indicazioni utili per valutare se sia il caso di considerare il passaggio a .NET Core e .NET 5.0 e sui vantaggi di ASP.NET Core rispetto alle versioni precedenti di ASP.NET.

Entro la fine di questo capitolo, avrete una buona panoramica del mondo .NET, del ruolo di .NET 5.0 e dei meccanismi di base del funzionamento di ASP.NET Core.

In questo capitolo

Introduzione ad ASP.NET Core

ASP.NET Core è un framework applicazioni web multipiattaforma e open source, che potete impiegare per programmare rapidamente applicazioni dinamiche e server-side. Potete impiegare ASP.NET Core anche per creare API HTTP utilizzabili da applicazioni di tipo mobile, da applicazioni monopagina per browser come Angular e React o da altre applicazioni backend, operanti sul server.

ASP.NET Core fornisce struttura, funzioni di supporto e un framework per la realizzazione di applicazioni web, che evita di dover scrivere molto di questo

codice. Il codice del framework ASP.NET Core richiama i vostri *handler*, "agganci", che, a loro volta, richiamano i metodi presenti nella logica operativa della vostra applicazione, come illustrato nella Figura 1.1. Questa logica operativa è il nucleo dell'applicazione. Potete anche interagire con altri servizi, come database o API remote, ma tipicamente la logica operativa della vostra applicazione non dipende *direttamente* da ASP.NET Core. Questa parte del capitolo tratta i seguenti argomenti.

- I motivi per utilizzare un framework web.
- I vantaggi e i limiti dei precedenti framework ASP.NET.
- Che cos'è ASP.NET e quali sono le sue motivazioni.

Alla fine di questa parte del capitolo avrete un'idea del perché è stato creato ASP.NET Core, dei suoi obiettivi progettuali e del perché può essere il caso di utilizzarlo.

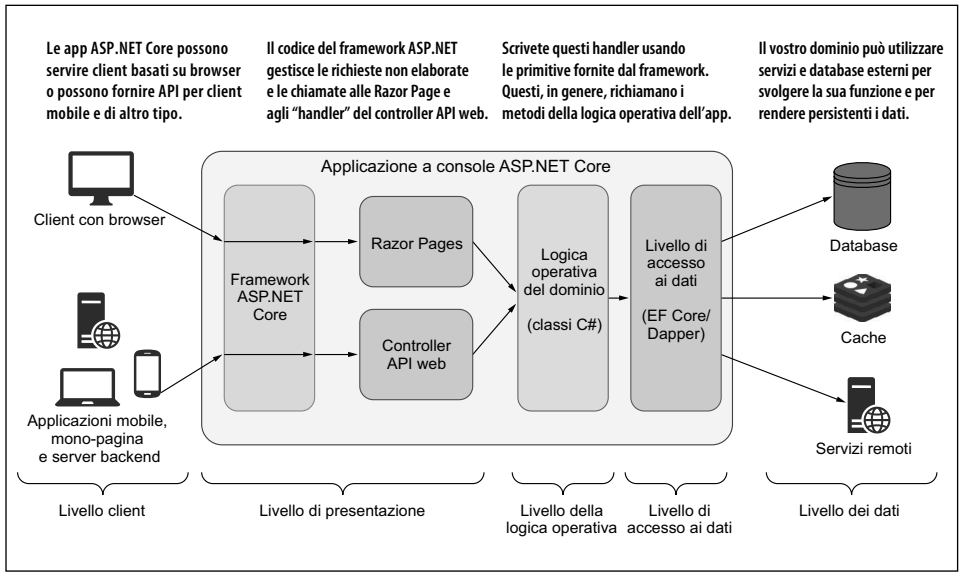

Figura 1.1 Una tipica applicazione ASP.NET Core è costituita da vari livelli. Il codice del framework ASP.NET Core gestisce le richieste da un client, occupandosi del complesso codice di gestione della rete. Poi il framework richiama gli handler (Razor Pages e controller delle API web) che avete scritto utilizzando le primitive fornite dal framework. Infine, questi handler richiamano la logica operativa della vostra applicazione, tipicamente costituita da classi e oggetti C# senza dipendenze specifiche da ASP.NET Core.

Utilizzo di un framework web

Se siete alle prime armi nel campo dello sviluppo web, può essere scoraggiante affrontare un campo così ricco di gergo e con una pletora di prodotti in continua evoluzione. Vi starete forse chiedendo se sono tutti necessari: ma… quanto può essere difficile chiedere a un server di restituire un file?

Be', è certamente possibile costruire un'applicazione web statica senza l'uso di alcun framework web, ma le sue funzionalità saranno giocoforza limitate. Non appena vorrete introdurre una qualsiasi forma di sicurezza o comportamento dinamico, probabilmente incontrerete difficoltà, e quella apparente semplicità che vi aveva allettato svanirà davanti ai vostri occhi.

Esattamente come i framework di sviluppo desktop o mobile possono aiutarvi a creare applicazioni native, ASP.NET Core rende la scrittura di applicazioni web più veloce, più semplice e più sicura rispetto a cercare di creare tutto da zero. ASP.NET Core contiene librerie per attività comuni come le seguenti.

- Creare pagine web il cui contenuto cambia dinamicamente.
- Consentire agli utenti di fare login alla vostra app web.
- Consentire agli utenti di utilizzare il proprio account Facebook per fare login alla vostra app web utilizzando OAuth.
- Costituire una struttura comune per la programmazione di applicazioni di facile manutenibilità.
- Leggere i file di configurazione.
- Fornire file di immagini.
- Eseguire il logging delle richieste giunte alla vostra app web.

Un fattore chiave per qualsiasi applicazione web moderna è la capacità di generare pagine web dinamiche. Una pagina web dinamica può visualizzare dati differenti a seconda dell'utente che ha fatto login o può visualizzare i contenuti inviati dagli utenti. Senza un framework dinamico non sarebbe possibile effettuare il login a siti web o visualizzare dati personalizzati su una pagina. In breve, siti web come Amazon, eBay e Stack Overflow (vedi la Figura 1.2) non sarebbero possibili.

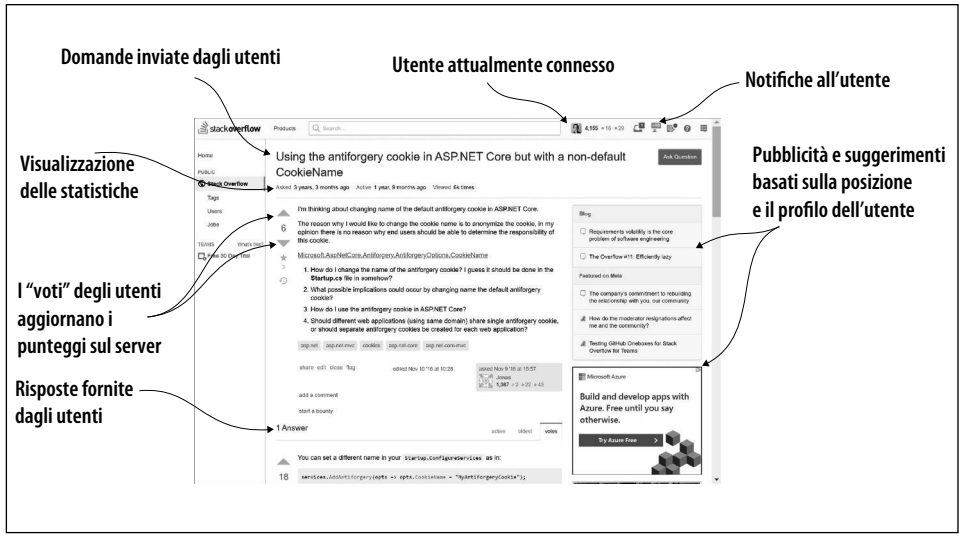

Figura 1.2 Il sito web Stack Overflow (https://stackoverflow.com) è stato creato utilizzando ASP.NET e offre quasi interamente contenuti dinamici.

I vantaggi e i limiti di ASP.NET

ASP.NET Core è l'ultima evoluzione del noto framework web Microsoft ASP.NET, rilasciato a giugno 2016. Le versioni precedenti di ASP.NET hanno visto emergere molti aggiornamenti incrementali, concentrati sul miglioramento della produttività degli sviluppatori e dando comunque una priorità alla compatibilità con le versioni precedenti. ASP.NET Core spinge ancora più in là questa tendenza apportando modifiche significative all'architettura, con un ripensamento sul modo in cui il framework web è stato progettato e realizzato.

ASP.NET Core deve molto ad ASP.NET e molte delle sue funzionalità sono evoluzioni delle versioni precedenti, ma ASP.NET Core è un framework interamente nuovo. L'intero stack di tecnologie è stato riscritto, comprendendo sia il framework web sia la piattaforma sottostante.

Al cuore di questi cambiamenti c'è la filosofia secondo cui ASP.NET dovrebbe confrontarsi a testa alta con altri framework moderni, ma lasciando agli attuali sviluppatori .NET un certo senso di familiarità con l'ambiente.

Per capire *perché* Microsoft abbia deciso di creare un nuovo framework, è importante comprendere i vantaggi e i limiti del precedente framework web ASP.NET.

La prima versione di ASP.NET è stata rilasciata nel 2002 nell'ambito di .NET Framework 1.0, in risposta agli ambienti di scripting allora convenzionali: ASP e PHP. ASP.NET Web Forms ha consentito agli sviluppatori di creare rapidamente applicazioni web utilizzando un ambiente di progettazione grafico e un semplice modello a eventi che rispecchiava le tecniche di programmazione di applicazioni desktop.

Il framework ASP.NET ha consentito agli sviluppatori di creare rapidamente nuove applicazioni, ma nel tempo l'ecosistema di sviluppo web è cambiato. È diventato sempre più evidente che ASP.NET Web Forms soffriva di molti problemi, specialmente nella programmazione di applicazioni più grandi. In particolare, la mancanza di verificabilità, un modello a stati complesso e un controllo limitato sull'HTML generato (aspetto che complica lo sviluppo lato client) hanno portato gli sviluppatori a valutare altre opzioni.

In risposta, nel 2009 Microsoft ha rilasciato la prima versione di ASP.NET MVC, basata sul pattern Model-View-Controller, un modello di web design ormai comune, utilizzato da altri framework come Ruby on Rails, Django e Java Spring. Questo framework ha consentito di separare gli elementi dell'interfaccia utente dalla logica dell'applicazione, ha semplificato i test e ha fornito un controllo più diretto sul processo di generazione del codice HTML.

ASP.NET MVC ha attraversato altre quattro iterazioni dalla sua prima versione, tutte incentrate sullo stesso framework sottostante, rappresentato dal file `System.Web.dll`. Questa libreria fa parte di .NET Framework, quindi si trova preinstallata in tutte le versioni di Windows e contiene tutto il codice di base utilizzato da ASP.NET quando si realizza un'applicazione web.

Questa dipendenza offre sia vantaggi sia svantaggi. Da un lato, il framework ASP.NET è una piattaforma affidabile e collaudata, adatta alla programmazione di applicazioni web su Windows. Fornisce un'ampia varietà di funzionalità, utilizzate da molti anni e ben note a quasi tutti gli sviluppatori web che lavorano in Windows.

Ma d'altro canto, questa dipendenza è limitante: le modifiche al file `System.Web.dll` sottostante sono di vasta portata e, di conseguenza, la loro implementazione è lenta. Ciò limita la misura in cui ASP.NET è libero di evolversi e si traduce in cicli di rilascio che si misurano in anni. Vi è, inoltre, un accoppiamento esplicito con l'host web Windows, *Internet Information Service* (IIS), il che ne preclude l'utilizzo su piattaforme non Windows.

Recentemente, Microsoft ha dichiarato che .NET Framework è "terminato". Non verrà rimosso né sostituito, ma non riceverà nemmeno nuove funzionalità. Di conseguenza, nemmeno ASP.NET, basato su `System.Web.dll`, riceverà nuove funzionalità o aggiornamenti.
Negli ultimi anni, molti sviluppatori web hanno iniziato a considerare altri framework web multipiattaforma che possono essere eseguiti su Windows, Linux e macOS. Microsoft ha così deciso che era giunto il momento di creare un framework che non fosse più legato alla sua eredità Windows, ed è così che è nato ASP.NET Core.

Che cos'è ASP.NET Core?

Lo sviluppo di ASP.NET Core è stato motivato dal desiderio di creare un framework web che avesse quattro obiettivi principali.

- Consentisse l'esecuzione e lo sviluppo multipiattaforma.
- Offrisse un'architettura modulare, per garantire la facilità di manutenzione.
- Venisse sviluppato completamente come software open source.
- Fosse in linea con le tendenze attuali dello sviluppo web, come le applicazioni client-side e la distribuzione in ambienti cloud.

Per raggiungere tutti questi obiettivi, Microsoft aveva bisogno di realizzare una piattaforma che potesse fornire librerie di base per creare oggetti fondamentali, come elenchi e dizionari, e per eseguire, per esempio, semplici operazioni sui file. Fino a quel momento, lo sviluppo in ASP.NET era sempre stato focalizzato e dipendente da .NET Framework, solo per Windows. Per ASP.NET Core, Microsoft ha creato una piattaforma leggera operante su Windows, Linux e macOS chiamata .NET Core (e successivamente .NET 5.0), come rappresentato nella Figura 1.3.

DEFINIZIONE

.NET 5.0 è la versione successiva di .NET Core, dopo la versione 3.1. Rappresenta un'unificazione di .NET Core e di altre piattaforme .NET in un unico runtime e framework. I termini .NET Core e .NET 5.0 sono spesso usati come sinonimi, ma per coerenza con il linguaggio di Microsoft, preferisco utilizzare il termine .NET 5.0 per fare riferimento alla versione più recente di .NET Core e utilizzare .NET Core quando mi riferisco alle versioni precedenti.

.NET Core (e il suo successore, .NET 5.0) utilizza molte delle API di .NET Framework, ma è più modulare e implementa solo un sottoinsieme delle funzionalità di .NET Framework, con l'obiettivo di fornire un'implementazione e una programmazione più semplici. È una piattaforma completamente distinta, più che una fork di .NET Framework, sebbene utilizzi codice simile per molte delle sue API.
Con il solo .NET 5.0 è possibile creare applicazioni a console multipiattaforma. Microsoft ha creato ASP.NET Core come un livello aggiuntivo che si colloca sopra le applicazioni a console, in modo tale che la conversione in un'applicazione web comporti l'aggiunta e la composizione di librerie, come rappresentato nella Figura 1.4.

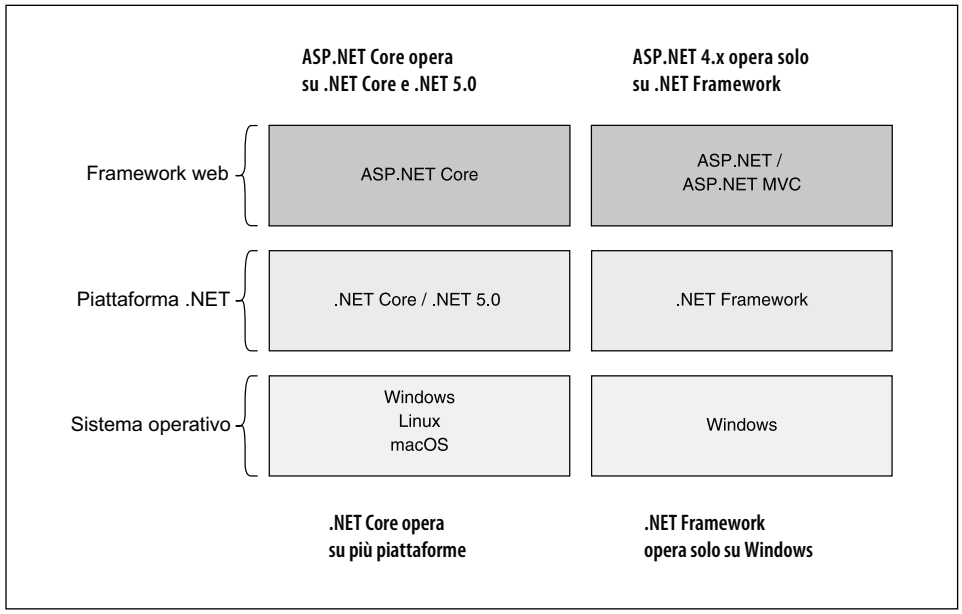

Figura 1.3 La relazione tra ASP.NET Core, ASP.NET, .NET Core/.NET 5.0 e .NET Framework. ASP.NET Core opera su .NET Core e .NET 5.0 e pertanto può essere eseguito su più piattaforme. Al contrario, ASP.NET opera solo su .NET Framework, quindi è legato al sistema operativo Windows.

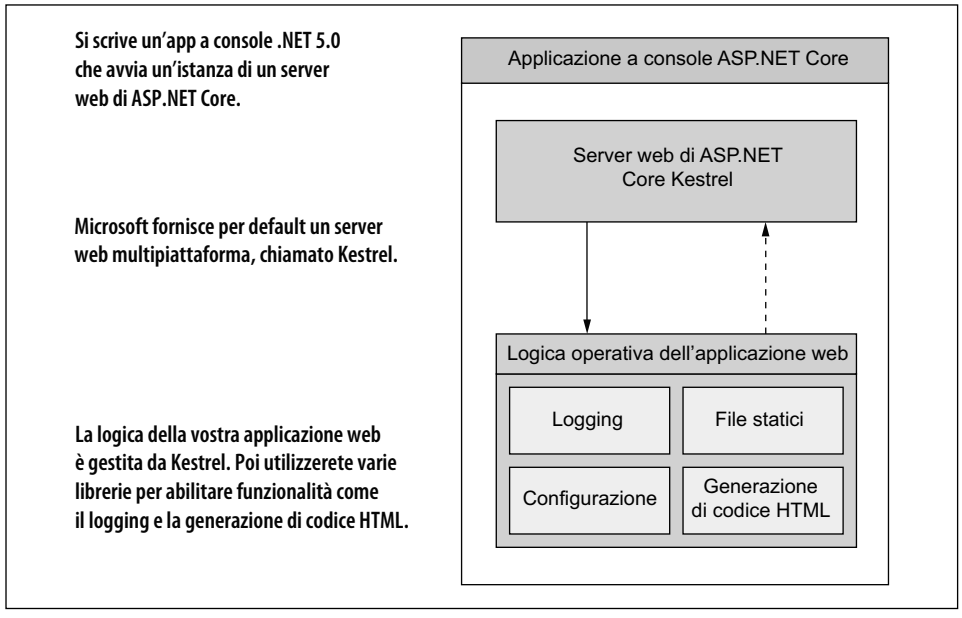

Figura 1.4 Il modello di un'applicazione ASP.NET Core. La piattaforma .NET 5.0 fornisce un modello base per un'applicazione a console per l'esecuzione di app dalla riga di comando. L'aggiunta di una libreria server web la converte in un'app web ASP.NET Core. Le funzionalità aggiuntive, come la configurazione e il logging, vengono aggiunte tramite librerie aggiuntive.

Aggiungendo alla vostra app .NET 5.0 un server web di ASP.NET Core, la vostra applicazione potrà essere eseguita come un'applicazione web. ASP.NET Core è composto da tante piccole librerie, tra le quali è possibile scegliere quelle utili a fornire alla vostra applicazione le funzionalità desiderate. Raramente avrete bisogno di avere a disposizione tutte le librerie, e aggiungerete solo quelle di cui avete bisogno. Alcune delle librerie sono molto comuni e appariranno praticamente in ogni vostra applicazione: per esempio quelle per leggere i file di configurazione o per eseguire il logging. Altre librerie sfruttano queste funzionalità di base per fornire operazioni specifiche dell'applicazione, come il logging tramite account Facebook o Google.

La maggior parte delle librerie che utilizzerete in ASP.NET Core è disponibile su GitHub, nei repository di Microsoft ASP.NET Core all'indirizzo `https://github.com/dotnet/aspnetcore`. Vi potete trovare tutte le librerie principali, come le librerie di autenticazione e logging, così come molte altre librerie accessorie, come quelle di autenticazione da terze parti.

Tutte le applicazioni ASP.NET Core avranno un comportamento simile per la configurazione di base, come suggerito dall'impiego di librerie comuni, ma in generale il framework è flessibile, cosa che vi lascia liberi di creare le vostre convenzioni di programmazione. Queste librerie comuni, le librerie di estensioni che si basano su di esse e le convenzioni di progettazione che promuovono sono tutte coperte dal termine alquanto nebuloso ASP.NET Core.

Quando scegliere ASP.NET Core?

Ora, dovreste avere una conoscenza almeno generale di che cos'è ASP.NET Core e di come è stato progettato. Ma la domanda rimane: perché usarlo? Microsoft raccomanda che tutto il nuovo sviluppo web .NET utilizzi ASP.NET Core, ma il passaggio a un nuovo stack web, e il suo apprendimento, rappresentano un grande impegno per qualsiasi sviluppatore o azienda. In questa parte del capitolo mi occupo dei seguenti argomenti.

- Che tipo di applicazioni potete sviluppare con ASP.NET Core.
- Alcuni degli aspetti più importanti di ASP.NET Core.
- Perché dovreste considerare l'utilizzo di ASP.NET Core per le vostre nuove applicazioni.
- Aspetti da considerare prima di convertire le attuali applicazioni ASP.NET in ASP.NET Core.

Quali tipi di applicazioni potete creare?

ASP.NET Core fornisce un framework web generalizzato che può essere usato per un'ampia varietà di applicazioni. Ovviamente può essere utilizzato per creare siti web ricchi e dinamici, siano essi siti di e-commerce, siti di contenuti o grandi applicazioni multilivello, più o meno come la versione precedente di ASP.NET.

Quando è stato rilasciato .NET Core, erano disponibili poche librerie di terze parti per la creazione di applicazioni così complesse. Grazie a diversi anni di sviluppo attivo, non è più così. Molti sviluppatori hanno aggiornato le proprie librerie per funzionare con ASP.NET Core e molte altre librerie sono state create per essere specificatamente destinate ad ASP.NET Core. Per esempio, il sistema di gestione dei contenuti (CMS) open source

Orchard (il codice sorgente del progetto Orchard è in: `https://github.com/OrchardCMS`) è stato riprogettato come Orchard Core (`www.orchardcore.net`) codice sorgente su `https://github.com/OrchardCMS/OrchardCore`) per poter operare in ASP.NET Core. Addirittura, il progetto CMS cloudscribe (`http://www.cloudscribe.com`, codice sorgente su `https://github.com/cloudscribe`) (Figura 1.5) è stato scritto appositamente per ASP.NET Core.

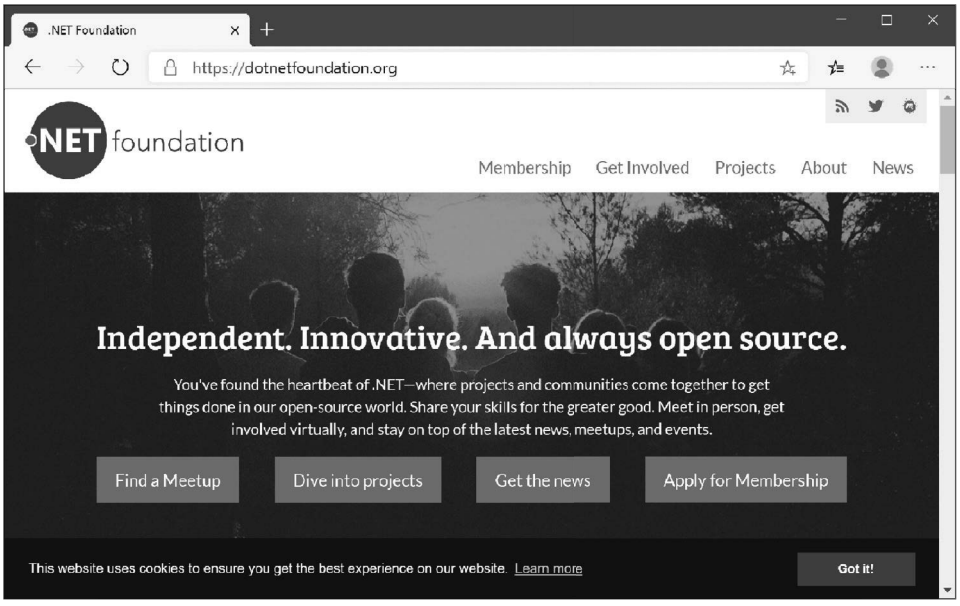

Figura 1.5 Il sito web di .NET Foundation (https://dotnetfoundation.org/) è stato creato utilizzando il CMS cloudscribe e ASP.NET Core.

Le applicazioni web tradizionali, con rendering lato server e basate su pagine sono il pane quotidiano dello sviluppo in ASP.NET, sia con la versione precedente di ASP.NET sia con ASP.NET Core. Inoltre, le applicazioni mono-pagina, che usano un framework client-side che comunemente comunica con un server REST, sono facili da sviluppare con ASP.NET Core. Sia che stiate usando Angular, Vue, React o qualche altro framework client-side, scoprirete che è facile creare un'applicazione ASP.NET Core che si comporti come un'API server-side.

DEFINIZIONE

REST sta per *REpresentational State Transfer*. Le applicazioni *RESTful*, in genere, utilizzano chiamate HTTP leggere e senza stato per leggere, pubblicare (creare/aggiornare) ed eliminare i dati.

ASP.NET Core non si limita a creare servizi RESTful. È facile creare un servizio web o un servizio in stile RPC (*Remote Procedure Call*) per la vostra applicazione, a seconda di quali sono le vostre esigenze, come rappresentato nella Figura 1.6. Nel caso più semplice, la vostra applicazione potrebbe esporre solo un singolo *endpoint*, restringendo il suo ambito d'azione fino a rendersi un microservizio. ASP.NET Core è perfettamente in grado di creare servizi semplici, grazie al supporto multipiattaforma e al design leggero.

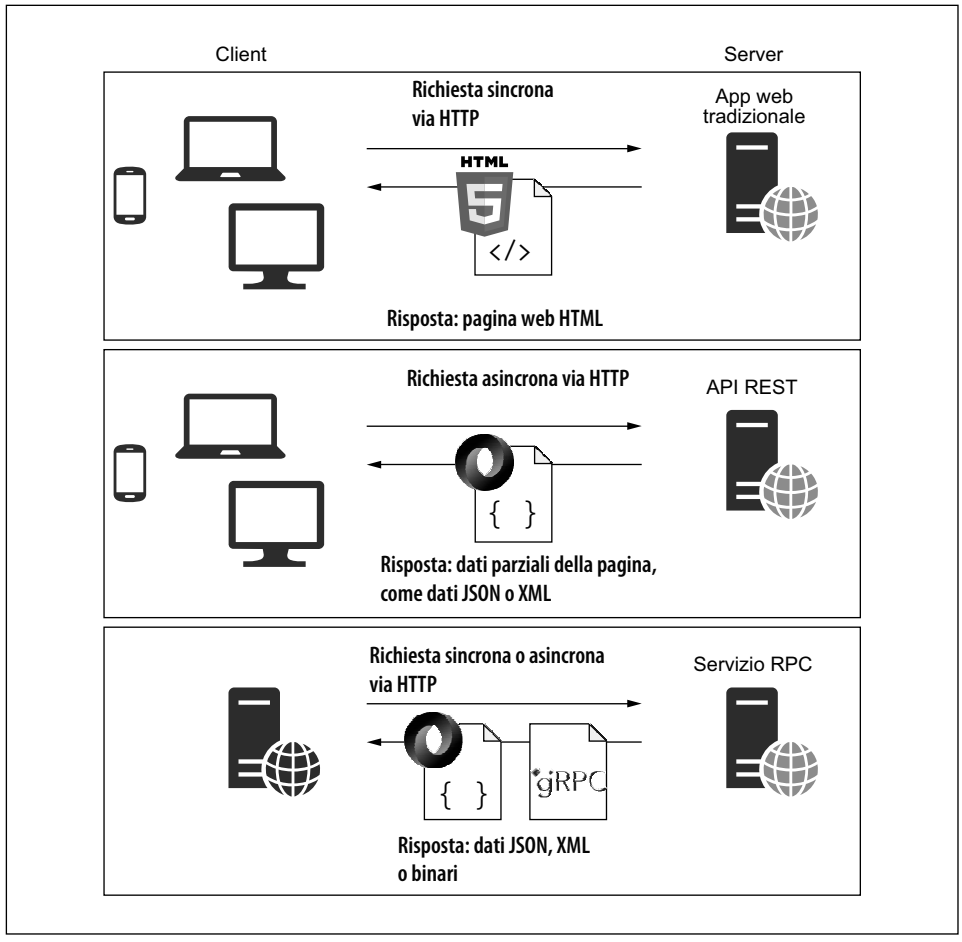

Figura 1.6 ASP.NET Core può fungere da applicazione server-side per un'ampia varietà di client: può servire pagine HTML per applicazioni web tradizionali, può fungere da API REST per applicazioni mono-pagina client-side o può fungere da servizio RPC ad hoc per applicazioni client.

> **NOTA**
>
> In questo libro mi concentro sulla programmazione di applicazioni web tradizionali, basate su pagine, con rendering server-side e API web RESTful. Nel Capitolo 22 mostro anche come creare servizi worker "headless".

Dovreste considerare vari fattori, non tutti tecnici, quando scegliete una piattaforma. Uno di questi fattori è il livello di supporto che potete aspettarvi di ricevere dai suoi creatori. Per alcune organizzazioni, questo può essere uno dei principali ostacoli all'adozione di un certo software open source. Fortunatamente, Microsoft si è impegnata a fornire un supporto completo per le versioni LTS (*Long Term Support*) di .NET Core e ASP.NET Core per almeno tre anni dal momento del loro rilascio (Consultate la policy di supporto in: `https://dotnet.microsoft.com/platform/support/policy/dotnet-core`). E poiché tutto

lo sviluppo avviene in piena luce, potrete ottenere risposte alle vostre domande anche dalla community in generale, oltre che direttamente da Microsoft.
Per decidere se utilizzare ASP.NET Core, dovete considerare due aspetti, in particolare: se siete già sviluppatori .NET e se state creando una nuova applicazione o state cercando di convertirne una esistente.

Se siete alle prime armi nello sviluppo .NET

Un paragrafo dedicato a chi non ha mai sviluppato in .NET e sta prendendo in considerazione ASP.NET Core. Microsoft sta promuovendo ASP.NET Core come un'opzione interessante anche per i principianti dello sviluppo web, ma la scelta multipiattaforma lo mette in competizione con molti altri framework, sfidandoli sul loro territorio. ASP.NET Core ha molti punti di forza rispetto ad altri framework web multipiattaforma.

- È un framework web moderno, ad alte prestazioni e open source.
- Utilizza modelli e paradigmi di progettazione ormai familiari.
- C# è un ottimo linguaggio (ma in alternativa potete impiegare VB.NET o F# se preferite).
- Potete sviluppare e utilizzarlo su qualsiasi piattaforma.

ASP.NET Core è una rivisitazione del framework ASP.NET, costruito con nuovi principi di progettazione software sulla nuova piattaforma .NET Core/.NET 5.0. Sebbene sia certamente nuovo, .NET Core vanta diversi anni di utilizzo diffuso in produzione e ha attinto in modo significativo al .NET Framework, la cui maturità, stabilità e affidabilità sono dimostrate da quasi due decenni di utilizzo. Questo significa che scegliendo ASP.NET Core e .NET 5.0 otterrete una piattaforma affidabile e un framework web completo. Molti dei framework web oggi disponibili usano pattern di sviluppo consolidati simili, e ASP.NET Core non fa eccezione. Per esempio, Ruby on Rails è noto per l'utilizzo del pattern MVC (*Model-View-Controller*); Node.js è noto per il modo in cui elabora le richieste utilizzando piccoli moduli discreti (chiamati *pipeline*); e la *dependency injection* è ormai offerta da un'ampia varietà di framework. Se queste tecniche vi sono familiari, non dovreste avere problemi a trasferirle in ASP.NET Core; se invece questi termini vi lasciano perplessi, non vedrete l'ora di utilizzare le migliori tecniche disponibili.

> **NOTA**
>
> Incontrerete una pipeline nel Capitolo 3, il pattern MVC nel Capitolo 4 e la *dependency injection* nel Capitolo 10.

Il linguaggio principale per lo sviluppo .NET, e in particolare ASP.NET Core, è C#. Questo linguaggio ha un enorme seguito, e per buoni motivi. Essendo un linguaggio basato sul C e orientato agli oggetti, risulta familiare a tutti coloro che sono abituati ai linguaggi C, Java e molti altri. Inoltre, offre molte e potenti funzionalità, come LINQ (*Language Integrated Query*), le *closure* e i costrutti per la programmazione asincrona. Inoltre, il linguaggio C# è progettato in modo aperto su GitHub, così come il compilatore C# di Microsoft, nome in codice Roslyn (il repository del codice sorgente GitHub per il linguaggio C# e .NET Compiler Platform sono disponibili all'indirizzo `https://github.com/dotnet/roslyn`).

NOTA

In tutto il libro utilizzerò C# e metterò in evidenza alcune delle nuove funzionalità fornite dal linguaggio, ma questo libro non ha lo scopo di insegnare a programmare in C#. Se il linguaggio vi interessa, vi consiglio di acquistare *C# in Depth*, quarta edizione di Jon Skeet (Manning, 2019) e *Code like a Pro in C#* di Jort Rodenburg (Manning, 2021).

Uno dei principali punti di forza di ASP.NET Core e .NET 5.0 è la capacità di sviluppare e di operare su qualsiasi piattaforma. Che stiate usando un Mac, Windows o Linux, potrete eseguire le stesse app ASP.NET Core e svilupparle in più ambienti. Se siete utenti Linux, è supportata un'ampia varietà di distribuzioni (RHEL, Ubuntu, Debian, CentOS, Fedora e openSUSE, solo per citarne alcune), quindi siete praticamente certi di poterlo usare anche sul vostro sistema operativo. ASP.NET Core può essere eseguito anche sulla piccola distribuzione Alpine, per distribuzioni veramente compatte nei container.

Costruito pensando ai container

Tradizionalmente, le applicazioni web venivano distribuite direttamente su un server o, più recentemente, su una macchina virtuale. Le macchine virtuali consentono di installare i sistemi operativi in uno strato di hardware virtuale, astraendo pertanto l'hardware sottostante. Ciò presenta numerosi vantaggi rispetto all'installazione diretta, come la facilità di manutenzione, distribuzione e ripristino. Sfortunatamente, si tratta anche di soluzioni pesanti, in termini sia di dimensioni del file sia di utilizzo delle risorse.

È qui che entrano in gioco i container. I container sono molto più leggeri e non presentano il sovraccarico delle macchine virtuali. Sono formati da una serie di livelli e non richiedono l'avvio di un nuovo sistema operativo. Ciò significa che sono veloci da avviare e ottimi per il provisioning rapido. I container, e Docker in particolare, stanno rapidamente diventando la piattaforma di riferimento per la creazione di sistemi di grandi dimensioni e scalabili.

I container non sono mai stati un'opzione particolarmente interessante per le applicazioni ASP.NET, ma con ASP.NET Core, .NET 5.0 e Docker per Windows tutto sta cambiando. Un'applicazione ASP.NET Core leggera in esecuzione sul framework multipiattaforma .NET 5.0 è perfetta per le distribuzioni di container di piccole leggeri. Riparleremo delle opzioni di deployment nel Capitolo 16.

Oltre a funzionare su ogni piattaforma, uno dei punti di forza di .NET è la possibilità di scrivere e compilare il codice una volta sola. La vostra applicazione viene compilata in codice IL (*Intermediate Language*), che è indipendente dalla piattaforma. Se nel sistema di destinazione è installato .NET 5.0 Runtime, potrete eseguire il codice IL compilato da qualsiasi piattaforma. Ciò significa che, per esempio, potete sviluppare su un computer Mac o Windows e distribuire esattamente gli stessi file su computer Linux. La promessa della singola compilazione di codice eseguibile ovunque si è finalmente realizzata con ASP.NET Core e .NET Core/.NET 5.0.

Se siete sviluppatori .NET Framework e dovete creare una nuova applicazione

Se siete già sviluppatori .NET, la scelta se investire in ASP.NET Core per realizzare nuove applicazioni è prevalentemente una questione di tempistica. Le prime versioni di .NET Core erano prive di alcune funzionalità che ne complicavano l'adozione. Con il rilascio di .NET Core 3.1 e .NET 5.0, questo non è più un problema; Microsoft ora consiglia esplicitamente per tutte le nuove applicazioni .NET di utilizzare .NET 5.0. Microsoft si è impegnata a fornire correzioni di bug e a mantenere la sicurezza del vecchio framework ASP.NET, ma non fornirà più aggiornamenti delle funzionalità. .NET Framework non viene rimosso, quindi le vostre vecchie applicazioni continueranno a funzionare, ma non dovreste più usarlo per nuove applicazioni.
I principali vantaggi di ASP.NET Core rispetto al precedente framework ASP.NET sono i seguenti.

- Sviluppo e distribuzione multipiattaforma.
- Un focus sulle prestazioni.
- Un modello di hosting semplificato.
- Rilasci regolari con un ciclo di rilascio più breve.
- Open source.
- Funzionalità modulari.

Come sviluppatori .NET, se non utilizzate alcun costrutto specifico di Windows, come il Registro di sistema, allora la capacità di sviluppare e di eseguire il deployment di software multipiattaforma apre nuove porte a tutta una nuova serie di applicazioni: approfittate dell'hosting di macchine virtuali Linux più economico nel cloud, usate i container Docker per un'integrazione continua e ripetibile o scrivete codice .NET sul vostro Mac senza dover ricorrere a una macchina virtuale Windows. ASP.NET Core, in combinazione con .NET 5.0, rende possibile tutto questo.
.NET Core e .NET 5.0 sono ambienti intrinsecamente multipiattaforma, ma permettono comunque di usare le funzionalità specifiche della piattaforma, se necessario. Per esempio, le funzionalità specifiche di Windows, come il Registro di sistema o i Directory Services, possono essere abilitate con un pacchetto di compatibilità che rende queste API disponibili in .NET 5.0.

> **NOTA**
>
> Il Windows Compatibility Pack è progettato per agevolare il porting del codice da .NET Framework a .NET Core/.NET 5.0. Vedere `https://docs.microsoft.com/dotnet/core/porting/windows-compat-pack`.

Saranno però disponibili solo eseguendo .NET 5.0 su Windows, non su Linux o macOS; quindi, sarà necessario assicurarsi che tali applicazioni vengano eseguite solo in ambiente Windows o che gestiscano altrimenti le API potenzialmente mancanti.
Il modello di hosting del precedente framework ASP.NET era relativamente complesso e si basava su Windows IIS per fornire l'hosting del server web. In un ambiente multipiattaforma, questo tipo di relazione simbiotica non è possibile, quindi è stato adottato un altro modello di hosting, che separa le applicazioni web dall'host sottostante. Questa

opportunità ha portato allo sviluppo di Kestrel: un server HTTP veloce e multipiattaforma su cui può essere eseguito ASP.NET Core.
Invece della situazione precedente, in cui IIS richiama punti specifici dell'applicazione, quelle ASP.NET Core sono applicazioni a console, che ospitano autonomamente un server web e gestiscono direttamente le richieste, come rappresentato nella Figura 1.7. Questo modello di hosting è concettualmente molto più semplice e consente di sottoporre a test ed eseguire il debug delle applicazioni dalla riga di comando, sebbene non venga eliminata del tutto la necessità di richiamare IIS (o un server equivalente), come vedrete nel prossimo paragrafo.

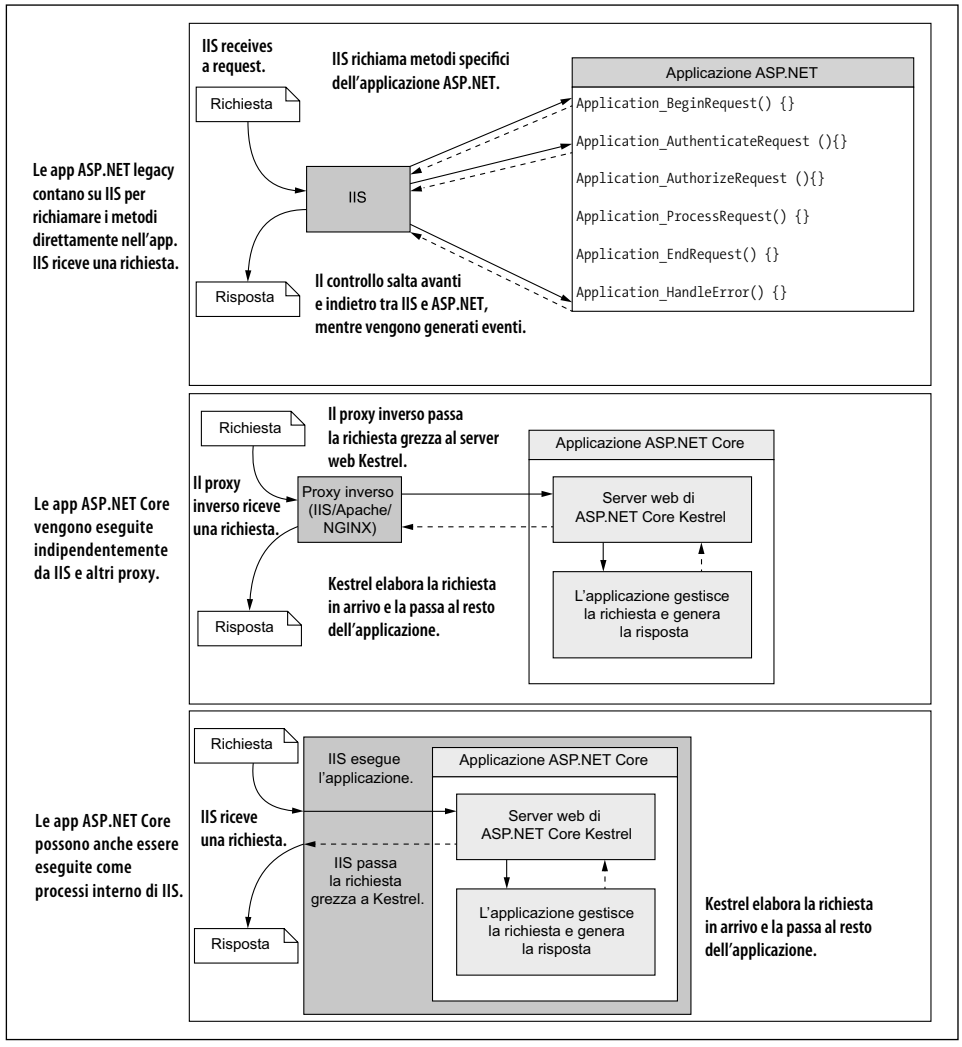

Figura 1.7 La differenza tra i modelli di hosting in ASP.NET (in alto) e ASP.NET Core (in basso). Con la versione precedente di ASP.NET, IIS rimane strettamente associato all'applicazione. Il modello di hosting in ASP.NET Core è più semplice; IIS passa la richiesta a un server web self-hosted nell'applicazione ASP.NET Core e riceve la risposta, ma non ha una conoscenza approfondita dell'applicazione.

NOTA

È anche possibile, opzionalmente, eseguire ASP.NET Core all'interno di IIS, come mostrato nella terza parte della Figura 1.7. Questo può offrire vantaggi in termini di prestazioni rispetto alla versione con proxy inverso. Si tratta principalmente di un dettaglio di distribuzione, che non cambia il modo in cui si compilano le applicazioni ASP.NET Core.

La modifica del modello di hosting, che prevede di utilizzare un server web HTTP integrato, ha creato un'altra opportunità. In passato le prestazioni sono state un punto dolente per le applicazioni ASP.NET. È certamente possibile creare applicazioni ad alte prestazioni (come testimonia Stack Overflow, `https://stackoverflow.com`), ma il framework web non è progettato avendo in mente alte prestazioni, quindi la cosa può finire per essere un po' un ostacolo.

Per renderlo competitivo come ambiente multipiattaforma, il team di ASP.NET si è concentrato sul rendere il server HTTP Kestrel il più veloce possibile. Ormai da diversi anni TechEmpower (`www.techempower.com/benchmarks`) esegue benchmark su un'intera gamma di framework web e con diversi linguaggi. Nel Round 19 dei benchmark con testo normale, TechEmpower ha annunciato che ASP.NET Core con Kestrel era il più veloce di oltre 400 framework sottoposti a test.

NOTA

Come sempre nello sviluppo web, la tecnologia è in costante divenire, quindi questi parametri di riferimento si evolveranno nel tempo. Sebbene ASP.NET Core possa non mantenere il suo slot nella top-ten, potete essere certi che le prestazioni sono uno dei punti focali chiave del team di ASP.NET Core.

Server web: nominare le cose può essere difficile

Uno degli aspetti più difficili della programmazione per il Web è la confusione fra terminologie, spesso conflittuali. Per esempio, se in passato avete utilizzato IIS, potreste averlo descritto come un server web o magari come un host web. Al contrario, se avete creato un'applicazione utilizzando Node.js, potreste aver definito quell'applicazione un server web. In alternativa, potreste aver chiamato server web la macchina fisica su cui operava la vostra applicazione. Allo stesso modo, potreste aver creato un'applicazione per Internet e averla chiamata sito web o applicazione web, probabilmente in modo un po' arbitrario e in base al comportamento dinamico che esibiva.

In questo libro, quando parlo di "server web" nel contesto di ASP.NET Core, mi riferisco al server HTTP che viene eseguito come parte dell'applicazione ASP.NET Core. Per default, questo è il server web Kestrel, ma questo non è un requisito. Sarebbe certamente possibile scrivere un server web alternativo e sostituirlo a Kestrel.

Il server web è responsabile della ricezione delle richieste HTTP e della generazione delle risposte. Nella versione precedente di ASP.NET, questo ruolo era coperto da IIS, mentre in ASP.NET Core il server web è Kestrel.

In questo libro userò il termine "applicazione web" solo per descrivere le applicazioni ASP.NET Core, indipendentemente dal fatto che contengano contenuti statici o che siano completamente dinamiche. In ogni caso, sono applicazioni a cui si accede tramite il Web, quindi quel nome sembra il più appropriato.

Molti dei miglioramenti prestazionali apportati a Kestrel non sono stati sviluppati dai membri del team ASP.NET, ma dai contributori al progetto open source su GitHub (il progetto GitHub server HTTP Kestrel è disponibile nel repository ASP.NET Core all'indirizzo `https://github.com/dotnet/aspnetcore`). Lo sviluppo "open" fa sì, in genere, che le correzioni e le nuove funzionalità arrivino in produzione più velocemente di quanto fosse possibile con la versione precedente di ASP.NET, che dipendeva da .NET Framework e Windows e, pertanto, prevedeva lunghi cicli di rilascio.
Al contrario, .NET 5.0, e quindi ASP.NET Core, è progettato per essere rilasciato in piccoli incrementi. Le versioni principali verranno rilasciate a una cadenza prevedibile, dove una nuova versione uscirà ogni anno e una nuova versione LTS (*Long Term Support*) verrà rilasciata ogni due anni (il piano di rilascio per .NET 5.0 e oltre si trova in `https://devblogs.microsoft.com/dotnet/introducing-net-5`). Inoltre, le correzioni di bug e gli aggiornamenti minori verranno aggiunti se e quando sono necessari. Le nuove funzionalità vengono fornite come pacchetti NuGet, indipendentemente dalla piattaforma .NET 5.0 sottostante.

NOTA

NuGet è un gestore di pacchetti per .NET che consente l'importazione di librerie nei progetti. È equivalente alle Ruby Gems, a npm per JavaScript o a Maven per Java.

Per consentire questo approccio alle release, ASP.NET Core è altamente modulare, prevedendo il minor accoppiamento possibile con altre funzionalità. Questa modularità si presta a un approccio *pay-for-play* alle dipendenze, in cui si inizia con un'applicazione essenziale e si aggiungono solo le librerie necessarie, al contrario dell'approccio "a vuotatoio" delle precedenti applicazioni ASP.NET. Perfino MVC è un pacchetto opzionale. Ma non temete: questo approccio non significa che ad ASP.NET Core manchino funzionalità; significa solo che dovete attivarle. Ecco alcuni dei miglioramenti chiave dell'infrastruttura.

- "Pipeline" middleware per definire il comportamento dell'applicazione.
- Supporto integrato della dependency injection.
- Infrastruttura combinata UI (MVC) e API (API web).
- Sistema di configurazione altamente estendibile.
- Scalabilità di default per piattaforme cloud utilizzando la programmazione asincrona.

Ognuna di queste funzionalità era possibile anche nella versione precedente di ASP.NET, ma richiedeva una discreta quantità di lavoro aggiuntivo per la configurazione. Con ASP.NET Core, tutto è lì, pronto e in attesa di essere impiegato.
Microsoft supporta completamente ASP.NET Core, quindi se volete creare un nuovo sistema, non c'è alcun motivo per non usarlo. L'ostacolo più grande che potreste incontrare è quello di voler usare modelli di programmazione non più supportati in ASP.NET Core, per esempio Web Forms o WCF Server, come vedremo nel prossimo paragrafo.
Spero che questa parte del capitolo abbia stuzzicato l'appetito con alcuni dei tanti motivi che spingono a utilizzare ASP.NET Core per la creazione di nuove applicazioni. Ma se siete già sviluppatori ASP.NET e state valutando se convertire un'applicazione ASP.NET in ASP.NET Core, questa è un'altra questione.

Conversione di un'applicazione ASP.NET in ASP.NET Core

A differenza delle nuove applicazioni, un'applicazione già preesistente presumibilmente svolge già un servizio, pertanto deve sempre esserci un vantaggio tangibile nel procedere a ciò che potrebbe rappresentare una riscrittura significativa, per la conversione da ASP.NET ad ASP.NET Core. I vantaggi dell'adozione di ASP.NET Core sono più o meno gli stessi delle nuove applicazioni: distribuzione multipiattaforma, funzionalità modulari e migliori prestazioni. Se tali vantaggi sono da considerarsi sufficienti dipende in gran parte dai dettagli della vostra applicazione, ma alcune caratteristiche sono chiari indicatori che spingono a non eseguire la conversione.

- La vostra applicazione usa ASP.NET Web Forms.
- La vostra applicazione usa WCF.
- La vostra applicazione è grande, e offre molte funzionalità MVC "avanzate".

Se disponete di un'applicazione ASP.NET Web Forms, non è consigliabile tentare di convertirla in ASP.NET Core. Web Forms è indissolubilmente legato al file `System.Web.dll` e come tale probabilmente non sarà mai disponibile in ASP.NET Core. La conversione di un'applicazione in ASP.NET Core implicherebbe sicuramente la riscrittura dell'applicazione da zero, con un cambio radicale non solo di framework, ma anche dei paradigmi di progettazione. Un approccio migliore sarebbe quello di introdurre lentamente i concetti delle API web e cercare di ridurre la dipendenza dai costrutti legacy di Web Forms, come `ViewData`. Potete trovare molte risorse online per aiutarvi con questo approccio, in particolare il sito web `https://www.asp.net/web-api`.

> **NOTA**
>
> Un approccio alternativo sarebbe considerare la conversione dell'applicazione in Blazor, utilizzando l'impegno collettivo della community per creare versioni Blazor dei componenti WebForms più comuni: `https://github.com/FritzAndFriends/BlazorWebFormsComponents`)

WCF (*Windows Communication Foundation*) è supportato solo parzialmente in ASP.NET Core (potete trovare le librerie client per l'utilizzo di WCF con .NET Core all'indirizzo `https://github.com/dotnet/wcf`). Potete utilizzare alcuni servizi WCF, ma il supporto è, nella migliore delle ipotesi, imprevedibile. Non esiste un modo supportato per ospitare un servizio WCF da un'applicazione ASP.NET Core, quindi se dovete assolutamente supportare WCF, per ora è meglio evitare ASP.NET Core.

> **SUGGERIMENTO**
>
> Se amate la programmazione in stile RPC di WCF, ma non avete requisiti rigidi per WCF, prendete in considerazione l'utilizzo di gRPC. gRPC è un moderno framework RPC dotato di molti concetti simili a WCF ed è supportato immediatamente da ASP.NET Core (potete trovare un eBook di Microsoft su gRPC per sviluppatori WCF all'indirizzo `https://docs.microsoft.com/enus/dotnet/architecture/grpc-for-wcf-developers`).

Se l'applicazione esistente è complessa e fa ampio uso di MVC, dei punti di estendibilità Web API o dei gestori di messaggi, il porting dell'applicazione in ASP.NET Core potrebbe essere particolarmente difficile. ASP.NET Core è dotato di molte funzionalità simili alla

versione precedente di ASP.NET MVC, ma l'architettura sottostante è differente. Molte delle funzionalità precedenti non hanno un sostituto diretto e quindi richiederanno un ripensamento radicale.
Più grande è l'applicazione, maggiore sarà la difficoltà che avrete nel convertire l'applicazione in ASP.NET Core. La stessa Microsoft suggerisce che il porting di un'applicazione da ASP.NET MVC ad ASP.NET Core è una riscrittura di dimensioni pari almeno al porting da ASP.NET Web Forms ad ASP.NET MVC. Se questo non vi spaventa, allora significa che siete pronti a tutto.
Se un'applicazione viene utilizzata raramente, non fa parte del vostro core business o non avrà bisogno di sviluppi significativi a breve termine, vi consiglio vivamente di non provare a convertirla in ASP.NET Core. Microsoft supporterà .NET Framework anche nel prossimo futuro (Windows stesso dipende da lui) ed è improbabile che valga la pena di convertire queste applicazioni "marginali".
Quindi, quando è il caso di convertire un'applicazione in ASP.NET Core? Come ho già detto, la migliore opportunità per iniziare è su progetti piccoli e nuovi piuttosto che su applicazioni esistenti. Detto questo, se l'applicazione esistente in questione è piccola o richiederà significativi sviluppi futuri, il porting potrebbe essere una buona opzione. È sempre meglio lavorare per piccole iterazioni, quando possibile, piuttosto che tentare di convertire l'intera applicazione in una sola volta. Ma se l'applicazione è costituita principalmente da controller MVC o API web e viste Razor associate, il passaggio ad ASP.NET Core potrebbe essere una buona scelta.

Come funziona ASP.NET Core?

A questo punto, dovreste avere una buona idea di che cosa sia ASP.NET Core e del tipo di applicazioni per cui dovreste usarlo. In questa parte del capitolo vedrete come funziona un'applicazione creata con ASP.NET Core, dal momento in cui l'utente richiede un URL alla visualizzazione della pagina nel browser. Per arrivarci, prima vedrete come funziona una richiesta HTTP per un qualsiasi server web, poi vedrete come ASP.NET Core estende il processo per creare pagine web dinamiche.

Come funziona una richiesta web HTTP?

Come sapete, ASP.NET Core è un framework per la programmazione di applicazioni web che "servono" dati da un server. Uno degli scenari più comuni per gli sviluppatori web è la creazione di un'app web che è possibile visualizzare in un browser web. Il processo di alto livello che potete aspettarvi da un server web è rappresentato nella Figura 1.8.
Il processo inizia quando un utente accede a un sito web o digita un URL nel browser. L'URL o l'indirizzo web è costituito da un *nome host* e da un *percorso* verso una risorsa nell'app web. La navigazione verso l'indirizzo dal browser prevede l'invio di una richiesta dal computer dell'utente al server su cui è ospitata l'app web, utilizzando il protocollo HTTP.

DEFINIZIONE

Il nome host di un sito web identifica univo≠camente la sua posizione in Internet e viene mappato, tramite un servizio DNS (*Domain Name Service*), su un indirizzo IP. Alcuni esempi sono `microsoft.com`, `google.it` e `facebook.com`.

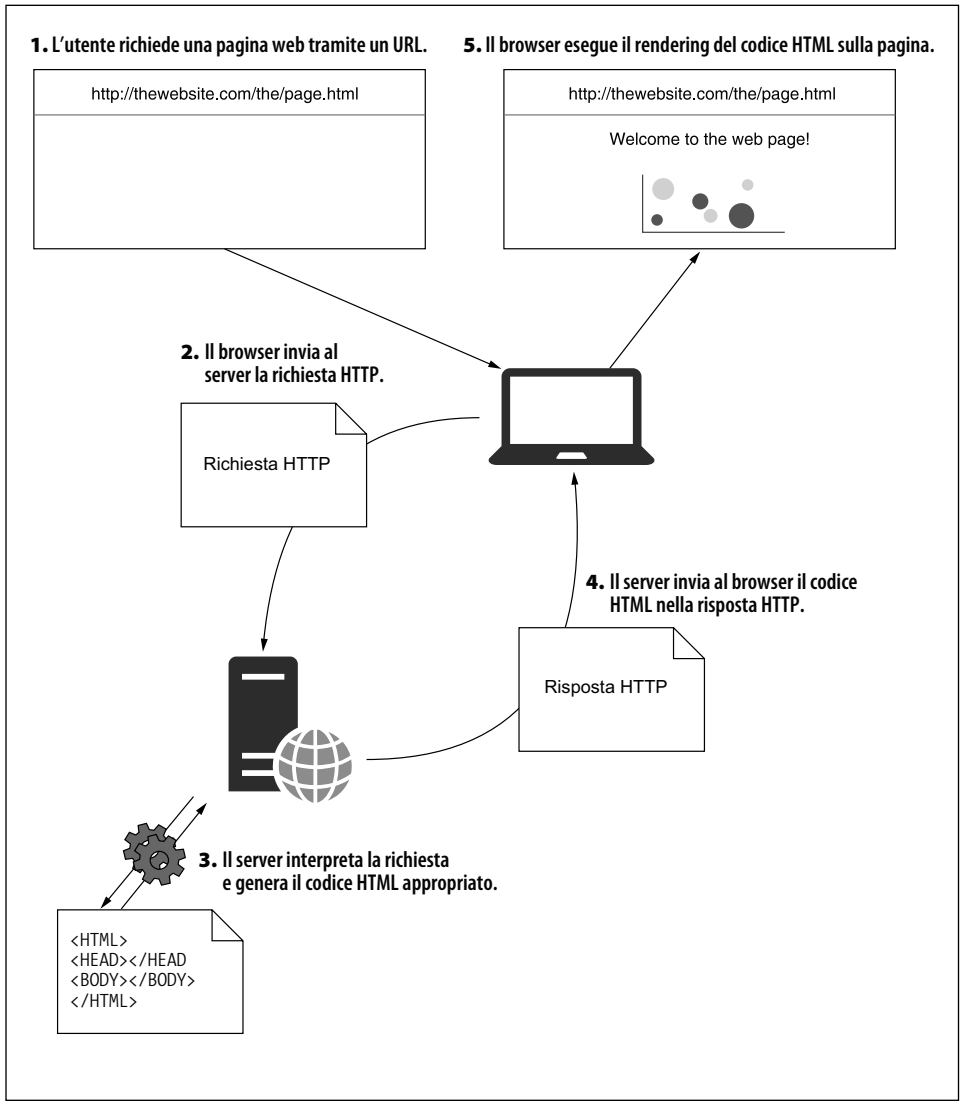

Figura 1.8 Richiesta di una pagina web. L'utente inizia richiedendo una pagina web. Ciò provoca l'invio al server di una richiesta HTTP. Il server interpreta la richiesta, genera il codice HTML necessario e lo restituisce in una risposta HTTP. Il browser può così visualizzare la pagina web.

Una breve introduzione su HTTP

HTTP (*Hypertext Transfer Protocol*) è il protocollo a livello di applicazione che alimenta il Web. È un protocollo a richiesta e risposta senza stato: una macchina client invia una richiesta a un server, che a sua volta invia una risposta.

Ogni richiesta HTTP è costituita da un *verbo* che indica il "tipo" della richiesta e da un *percorso* che indica la risorsa con la quale interagire. In genere le richieste includono anche delle *intestazioni*, che sono coppie chiave-valore, e in alcuni casi un *corpo*, che può essere il contenuto di un modulo, quando si inviano dati al server.
Una risposta HTTP contiene un *codice di stato*, che indica se la richiesta ha avuto esito positivo e, opzionalmente, le *intestazioni* e il *corpo*.
Per una descrizione più dettagliata del protocollo HTTP e per ulteriori esempi, consultate il paragrafo 1.3 *A quick introduction to HTTP* del libro *Go Web Programming* di Sau Sheong Chang (Manning, 2016), `https://livebook.manning.com/book/go-web-programming/chapter-1/point-9018-55-145-1`.

La richiesta attraversa Internet, giungendo potenzialmente dall'altra parte del mondo, fino a raggiungere il server associato al nome host specificato, sul quale è in esecuzione l'app web. La richiesta viene, potenzialmente, ricevuta e ritrasmessa da più router lungo il percorso, ma solo quando raggiunge il server associato al nome host la richiesta viene elaborata.
Quando il server riceve la richiesta, innanzitutto verifica che abbia senso e, in caso affermativo, genera una risposta HTTP. A seconda della richiesta, questa risposta potrebbe essere una pagina web, un'immagine, un file JavaScript o un semplice *acknowledgement*. Per questo esempio, presumo che l'utente abbia raggiunto la home page di un'app web, quindi il server risponde con del codice HTML, che viene aggiunto alla risposta HTTP, la quale viene poi rinviata attraverso Internet al browser che ha effettuato la richiesta.
Non appena il browser dell'utente inizia a ricevere la risposta HTTP, può iniziare a visualizzare il contenuto sullo schermo, ma la pagina HTML può anche fare riferimento ad altre pagine e link sul server. Per visualizzare la pagina web completa, invece di un file HTML statico, scarno e grezzo, il browser deve ripetere il processo di richiesta, recuperando ogni file previsto. HTML, immagini, CSS per lo stile e file JavaScript per comportamenti extra vengono tutti recuperati utilizzando lo stesso identico processo di richiesta HTTP.
Praticamente tutte le interazioni che avvengono in Internet celano questo stesso processo di fondo. Una pagina web di base potrebbe richiedere solo poche semplici richieste per offrire un rendering completo, mentre una pagina web moderna e di grandi dimensioni potrebbe richiederne anche centinaia. Al momento attuale, la homepage di Amazon.com (`www.amazon.com`), per esempio, effettua ben 606 richieste, incluse quelle per tre file CSS, dodici file JavaScript e ben 402 file di immagini.
Ora che avete un'idea del processo che si svolge per il caricamento di una pagina nel browser, vediamo come ASP.NET Core genera dinamicamente la risposta sul server.

In che modo ASP.NET Core elabora una richiesta?

Quando create un'applicazione web con ASP.NET Core, i browser continueranno a usare lo stesso protocollo HTTP descritto per comunicare con l'applicazione. ASP.NET Core si occupa di tutto ciò che avviene sul server per gestire una richiesta, inclusa la verifica della validità della richiesta, la gestione dei dettagli di accesso e la generazione del codice HTML.

Proprio come con l'esempio della pagina web generica, il processo di richiesta inizia quando il browser di un utente invia al server una richiesta HTTP, come rappresentato nella Figura 1.9.
La richiesta viene ricevuta attraverso la rete dall'applicazione ASP.NET Core. Ogni applicazione ASP.NET Core dispone di un server web incorporato, che è Kestrel per default, responsabile della ricezione delle richieste non elaborate e della creazione di una rappresentazione interna dei dati, un oggetto `HttpContext`, che può essere usato dal resto dell'applicazione.
Grazie a questa rappresentazione, la vostra applicazione dovrebbe avere tutti i dettagli necessari per creare una risposta adeguata alla richiesta. Può utilizzare i dettagli archiviati in `HttpContext` per generare una risposta appropriata, che potrebbe essere generare del codice HTML, restituire un messaggio "Access denied" o inviare un'e-mail, il tutto a seconda dei requisiti dell'applicazione.
Una volta che l'applicazione ha terminato di elaborare la richiesta, restituirà la risposta al server web. Il server web di ASP.NET Core convertirà la rappresentazione in una risposta HTTP grezza e la invierà alla rete, la quale a sua volta la inoltrerà al browser dell'utente. Per l'utente, questo processo ha lo stesso comportamento della richiesta HTTP generica rappresentata nella Figura 1.8: l'utente ha inviato una richiesta HTTP e ha ricevuto una risposta HTTP. Tutte le differenze sono sul lato server, all'interno della vostra applicazione.

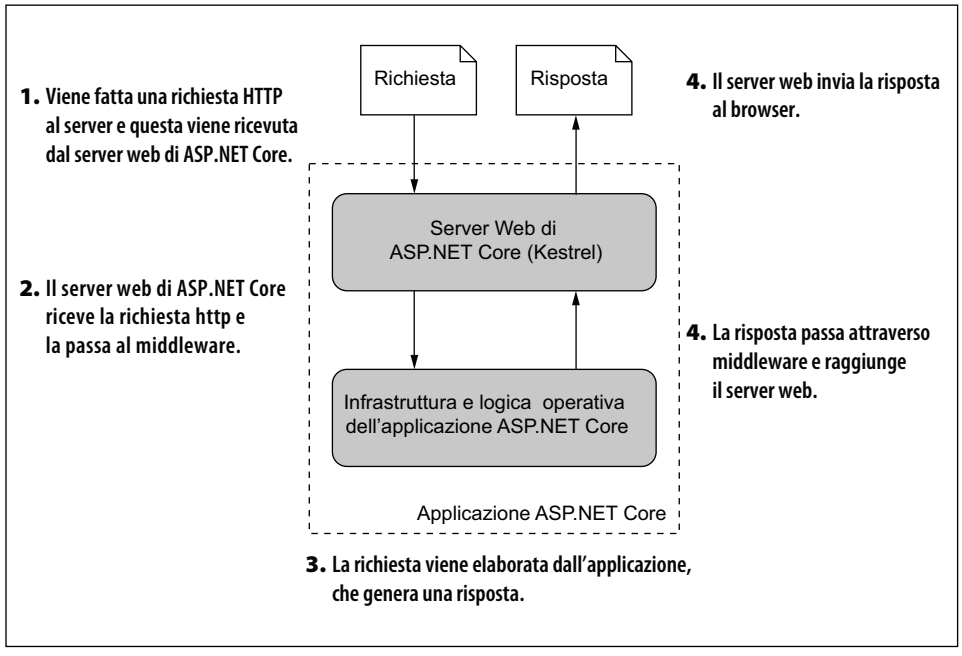

Figura 1.9 Il modo in cui un'applicazione ASP.NET Core elabora una richiesta. L'applicazione ASP.NET Core riceve una richiesta, e lancia un server web self-hosted. Il server web elabora la richiesta e la passa al corpo dell'applicazione, che genera una risposta e la restituisce al server web. Il server web invia questa risposta al browser.

ASP.NET Core e proxy inversi

È possibile esporre le applicazioni ASP.NET Core direttamente a Internet, in modo che Kestrel riceva le richieste direttamente dalla Rete. Tuttavia, è più comune utilizzare un proxy inverso tra la rete e l'applicazione. In Windows, il server proxy inverso sarà, in genere, IIS e su Linux o macOS potrebbe essere NGINX, HAProxy o Apache.

Un *proxy inverso* è un software responsabile della ricezione delle richieste e del loro inoltro al server web appropriato. Il proxy inverso è esposto direttamente a Internet, mentre il server web sottostante è esposto solo al proxy. Questa configurazione offre diversi vantaggi, principalmente in termini di sicurezza e prestazioni dei server web.

Potreste pensare che il fatto di avere un proxy inverso e un server web sia in qualche modo ridondante. Perché non avere l'uno o l'altro? Bene, uno dei vantaggi è il disaccoppiamento della vostra applicazione dal sistema operativo sottostante. Lo stesso server web di ASP.NET Core, Kestrel, può essere multipiattaforma e utilizzato dietro un'ampia varietà di proxy senza imporre vincoli a una particolare implementazione. In alternativa, se avete scritto un nuovo server web per ASP.NET Core, potete usarlo al posto di Kestrel senza dover modificare nient'altro sulla vostra applicazione.

Un altro vantaggio di un proxy inverso è che può essere rafforzato contro potenziali minacce provenienti da Internet. Un proxy inverso può anche essere dotato di aspetti aggiuntivi, come il riavvio di un processo bloccatosi. Kestrel può rimanere un semplice server HTTP e non doversi occupare di queste funzionalità grazie al fatto che viene utilizzato dietro un proxy inverso. Consideratela come una separazione dei compiti: Kestrel si occupa di generare le risposte HTTP; il proxy inverso si occupa della gestione della connessione a Internet.

Finora avete visto come le richieste e le risposte entrano ed escono da un'applicazione ASP.NET Core, ma non ho ancora parlato di come viene generata la risposta. Nella Parte I di questo libro esamineremo i componenti che costituiscono una tipica applicazione ASP.NET Core e le loro relazioni. Servono molte cose per generare una risposta in ASP.NET Core, e, in genere, tutto deve avvenire in una frazione di secondo, ma nel corso del libro esamineremo molto lentamente un'applicazione, trattando in dettaglio ciascuno dei suoi componenti.

Che cosa imparerete in questo libro

Questo libro vi guiderà in un tour approfondito del framework ASP.NET Core. Per sfruttare il libro nel migliore dei modi, dovreste avere una certa familiarità con C# o un linguaggio a oggetti analogo. Sarà utile anche una certa conoscenza di base dei concetti tipici del Web, come HTML e JavaScript. Ecco di che cosa ci occuperemo.

- Come creare applicazioni basate su pagine con Razor Pages.
- I concetti chiave di ASP.NET Core, come in *binding del modello*, la convalida e il routing.
- Come generare codice HTML per le pagine web utilizzando la sintassi Razor e i Tag Helper.

- L'utilizzo di funzionalità come la dependency injection, la configurazione e il logging a mano a mano che le applicazioni diventano più complesse.
- Come proteggere la vostra applicazione utilizzando le migliori tecniche di sicurezza.

In tutto il libro useremo un'ampia varietà di esempi per apprendere ed esplorare questi concetti. Gli esempi sono generalmente piccoli e autonomi, per permetterci di concentrarci su una singola caratteristica per volta.
Userò Visual Studio per la maggior parte degli esempi del libro, ma sarete in grado di seguirli anche usando il vostro editor o IDE preferito. L'Appendice A include molti dettagli sulla configurazione dell'editor o dell'IDE e sull'installazione di .NET 5.0 SDK. Anche se gli esempi presentati in questo libro impiegano l'ambiente e gli strumenti di Windows, tutto ciò che vedrete può essere ottenuto ugualmente bene su piattaforme Linux o Mac.

SUGGERIMENTO

Potete installare .NET 5.0 da `https://dotnet.microsoft.com/download`. L'Appendice A contiene ulteriori dettagli utili per configurare l'ambiente di sviluppo per lavorare con ASP.NET Core e .NET 5.0.

Nel prossimo capitolo, creerete la vostra prima applicazione da un template e poi la eseguirete. Esamineremo ciascuno dei componenti principali che compongono la vostra applicazione e vedremo come essi interagiscono tra loro per eseguire il rendering di una pagina web.

Riepilogo

- ASP.NET Core è un nuovo framework web creato con tecniche moderne di architettura software e avendo come obiettivo la modularizzazione.
- È particolarmente adatto a nuovi progetti.
- Le tecnologie legacy come WCF Server e Web Forms non possono essere usate con ASP.NET Core.
- ASP.NET Core viene eseguito sulla multipiattaforma .NET 5.0. È possibile accedere a funzionalità specifiche di Windows, come il Registro di sistema, utilizzando il pacchetto di compatibilità di Windows.
- .NET 5.0 è la versione successiva di .NET Core dopo .NET Core 3.1.
- L'ottenimento di una pagina web implica l'invio di una richiesta HTTP e la ricezione di una risposta HTTP.
- ASP.NET Core consente di creare in modo dinamico le risposte per una determinata richiesta.
- Un'applicazione ASP.NET Core contiene un server web, che funge da punto d'ingresso per una richiesta.
- Le app ASP.NET Core sono protette da Internet da un server proxy inverso, il quale inoltra le richieste all'applicazione.