Real-Time Web Application Development

With ASP.NET Core, SignalR, Docker, and Azure

Rami Vemula

Apress®

Real-Time Web Application Development

With ASP.NET Core, SignalR, Docker, and Azure

Rami Vemula

Apress®

Real-Time Web Application Development

Rami Vemula
Visakhapatnam, Andhra Pradesh, India

ISBN-13 (pbk): 978-1-4842-3269-9 ISBN-13 (electronic): 978-1-4842-3270-5
https://doi.org/10.1007/978-1-4842-3270-5

Library of Congress Control Number: 2017960937

Cover image by Freepik (www.freepik.com)

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Nikhil Karkal
Development Editor: Matthew Moodie
Technical Reviewer: Mohana Krishna G and Yogesh Sharma
Coordinating Editor: Prachi Mehta
Copy Editor: Sharon Wilkey

Dedicated to my maternal grandmother Kanaka Maha Lakshmi and in (loving) memories of my grandparents - Rowgi (maternal grandfather), Venkaiah (paternal grandfather) and Raghavamma (paternal grandmother).

Contents

About the Author

Rami Vemula is a technology consultant who has more than 7 years of experience is delivering scalable Web and Cloud solutions using Microsoft Technologies and platforms, which includes ASP.Net MVC/Web API, .NET Core, ASP.Net Core, JQuery, C#, Entity Framework, SQL Server and Azure. He is currently working for Deloitte US India Offices as Senior Consultant, where he leads a team of talented developers. As a part of his work, he architects, develops and maintains technical solutions to various clients in Public Sector domain.

Although web technologies are his primary area of focus, he also worked on other projects like Big Data Analytics using HDInsight, Universal Windows Platform Apps, DevOps etc. Now a days he is promoting Open Source technologies, platforms and tools to build cross platform solutions. He is also interested in providing streamlined DevOps integration flows through which Development teams can achieve greater productivity.

He is a Microsoft Certified ASP.Net and Azure Developer. He was a Microsoft ASP.Net MVP from 2011 to 2014 and an active trainer. In his free time, he enjoys answering technical questions at StackOverflow and forums.asp.net. He loves to share his technical experiences through his blog at http://intstrings.com/ramivemula. A part from technology, his other interests include movies, drama and theatre arts.

He holds Master's Degree in Electrical Engineering from California State University, Long Beach, USA in 2009. He is married and lives with his wife, kid and parents in Hyderabad, India.

You can reach Rami at rami.ramilu@gmail.com or https://twitter.com/RamiRamilu.

About the Technical Reviewers

Yogesh Sharma is a software engineer who loves to work at the cutting edge of web technologies. He focuses his time and effort on the development of software that can help other people learn. His most recent project involving the MEAN stack is an NLP system that is currently under active development. He completed his bachelor's degree in Information Technology at the Vidyalankar School of Information Technology in Mumbai. He currently works for Mphasis as a senior infrastructure engineer on Microsoft Azure Cloud Services. He can be reached at `www.linkedin.com/in/yogisharma24/`.

Mohana Krishna Gundumogula is a full-stack developer in a multinational software engineering firm. He is a Microsoft Certified Professional for Developing Microsoft SharePoint Server 2013 Core Solutions. His expertise includes Microsoft web technologies, collaborative platforms such as Microsoft SharePoint, cloud platforms such as Microsoft Azure, as well as the AngularJS and ReactJS frameworks.

Acknowledgments

I would like to thank my parents, Ramanaiah and RajaKumari, my wife, Sneha, my two year old daughter Akshaya and rest of the family - Aunts, Uncles, Cousins, Kids, for their patience and support throughout my life and making me achieve all wonderful milestones and accomplishments. Their consistent encouragement and guidance gave me strength to overcome all the hurdles and kept me moving forward.

I would like to thank my longtime friend Sanjay Mylapilli for always pushing me towards great heights and excellence. Thanks to Mahadevu Durga Prasad for always encouraging to explore new opportunities. Speial thanks to VSN Sastry uncle for believing and supporting me.

I would not imagine achieving all this success and passion in my life without the coaching and help from my mentors. I would like to thank Srikanth Pragada for introducing me to the world of .NET and teaching me how to deliver technical solutions with utmost discipline. Special thanks to Venu Yerra and RK Vadlani of Idea Entity Tech Solutions Pvt. Ltd. for trusting me and giving me first professional opportunity where I was introduced to real world problems and technical challenges.

My heartfelt thanks to Vishwas Lele, Rajesh Agarwal and Nasir Mirza of Applied Information Sciences (AIS) for introducing me to Microsoft's Azure platform. The extensive training provided by Vishwas not only matured me in providing cloud optimized solutions but also changed my thought process in architecting cloud solutions. Special thanks to Gaurav Mantri (Cerebrata) for introducing me to the fantastic world of Cloud Technology.

I would like to thank all my friends at Deloitte for having lengthy conversations around modern technologies and supporting me in reaching my goals.

Thanks to Nikhil Karkal, Prachi Mehta, Matt and other amazing people at APress for this wonderful opportunity and making this a memorable journey. Thanks to Technical reviewers, Yogesh and Mohana for valuable suggestions.

Sincere thanks to all friends, especially C O Dinesh Reddy, developers and architects at Microsoft and Open source contributors who are extensively working in develivering great products, frameworks and next generation platforms which created different opportunities to innonvate. The consistent effort of Microsoft communities in keeping up the document up-to-date and the prompt responses to technical queries is unprecedented.

Last but not least, I would like to thank all my readers for spending their time and effort in reading this book. Please feel free to share your feedback on this book which will help me to deliver better content in future. I look forward for comments and suggestions.

Introduction

In this modern era of technology, businesses strive to grow and expand at a rapid pace by exploring different technical opportunities to cater to diverse customer base. With the evolution of Open Source Community, technologies are evolving at a faster cadence and new ones are quickly emerging to bridge the gaps between different platforms. While Cloud platforms continued to advance in offering the most affordable and greatest computational power, technologies focus is being shifted to develop and deliver cross platform solutions by leveraging hybrid cloud infrastructure.

"Real Time Web Application Development with ASP.Net Core, SignalR, Docker and Azure" is going to take you through the journey of designing, developing and deploying a real world web application by leveraging modern day open source technologies. ASP.Net Core is the latest open source web framework from Microsoft which is capable of building cross platform web applications and SignalR enriches the application by enabling real time communication between server and clients. The Material Design concepts will deliver the state of the art look and feel for the application which supports multiple devices with different form factors. Azure is Microsoft's cloud computing platform which is leveraged in this book to demonstrate data storage and hosting capabilities.

The concepts of code version control using GitHub along with Travis CI Builds will help Developers, Software Engineers, System Administrators, and Architects in building a robust and reliable development ecosystem. Docker's container technology and its seamless integration with GitHub is used to package the application and provide continuous deployment to Azure's IaaS platform.

This book will empower you to gain deeper insights in developing cross platform applications by combining different open source technologies. By end of this book, you will be equipped to take on any real world challenge and provide reliable solution. The rich code samples from this book can be used to retrofit or upgrade existing ASP.Net Core applications.

This book will deep dive into following topics:

- Design and Develop Real world ASP.Net Core application with Materialize CSS and Azure Storage.
- Implement security and data persistence using OAuth 2.0 External Social Logins and Azure Storage.
- Real time communication orchestration using SignalR.
- Source version control with GitHub and continuous integration with Travis CI Build service.
- Containerize and Continuous Deployment using Docker and Azure Linux Virtual Machines.

Chapter 1 will introduce you to the real world 'Automobile Service Center' application which we are going to build through out this book. Chapter 1 also discuss the technology stack and prerequisites which will be used in building the application along with technical architecture. Chapters 2 through 5 will take you to the basic concepts of .NET Core, ASP.NET Core, Material Design techniques, Azure Table Storage and xUnit.Net test cases. In Chapter 6, we will work on securing the application with ASP.NET Core Identity enable customer authentication using Gmail Provider using OAuth 2.0 protocol.

Chapters 7 and 8 will discuss the implementation of Master data, Caching, Exception handling and Logging. We will use Azure Table Storage to persist all the application data along with logs. In Chapter 9 and 10, we will develop the application pages using ASP.NET Core and related concepts like Data Validatiton, Internationalization etc. We will implement real time communication and notifications for the application in chapter 11 using SignalR framework and also enable SMS notifications using Twilio API.

In Chapter 12, we will integrate version control for application code at GitHub Repository using Git commands (and also using Visual Studio). Continuous Integration with Travis Build is described in Chapter 13 along with user notifications on successful/faild builds. Chapters 14 and 15 focus on Containarization of application code using Docker and deploying it to Azure Linux Virtual Machines.

As a last note, I encourage you to extend the Automobile Service Center application to support functionalities like used car sales, spare parts management, financial accounting services etc. As the digital world transforming itself into microservices to serve global audience, there is a need for technologies to collaborate and deliver high performance and scalable solutions. An optimistic collaboration and automation can happen if technologies are open source, cross platform, easy to adapt and cloud ready. This book narrates one of the most prospective collaboration, having said that, it is just the beginning of long journey.

CHAPTER 1

Designing a Modern Real-World Web Application

In this modern era of the digital world, technology is no longer an auxiliary system but an integral part of the entire social ecosystem. Our current-day sophisticated devices, along with cutting-edge technologies, have made the entire world into a global village. People from different parts of the world, in spite of their diversity, can collaborate and solve complex challenges. Over the past decade, the open source community has played a pivotal role in evolving technology and setting the pace for innovation.

The ever-changing cadence of technology and tools has prompted a new generation of developers and engineers to rely mostly on Internet resources to educate themselves and deliver solutions. Although that approach still works in delivering solutions, it doesn't help in providing a panoramic view of the software-building process. Although different software development life cycle (SDLC) methodologies follow different processes, the most important ones are detailing functional requirements, system design, implementation, testing, deployment, and maintenance. Except for the testing part, this book will provide a holistic view of the software development process using the latest open source technologies and tools.

In this chapter, you will learn the basic phases of the software development life cycle. We will simulate a real-world business use case through a fictitious US company, Automobile Service Center. We will brainstorm the typical day-to-day challenges faced by this company and propose prospective solutions through a web application that we'll build with the latest technologies throughout the rest of the book. We'll start by defining the technologies that we are going to use in building the application. Then we'll design the logical architecture followed by continuous integration and deployment pipeline of the application. Finally, we will conclude the chapter by exploring the software prerequisites for our development environment.

Note My primary motive in writing this book is to introduce you to the experience of end-to-end software application development using .NET Core, ASP.NET Core, SignalR, Docker, and Microsoft Azure. This book provides a high-level overview of software development methodologies, and a detailed exploration is beyond the scope of this book.

R. Vemula, *Real-Time Web Application Development*, https://doi.org/10.1007/978-1-4842-3270-5_1

Overview of Application Development Strategies and Processes

The incredible pace of the current generation's digital innovation is not only transforming the creative process, but also influencing the incubation period for the latest technologies. Because of so many technical options being available to solve any given unique business challenge, technologists need to be more careful than ever before in offering prospective solutions. The current marketplace requires any technical professional to be adept at three major tasks: exploring, understanding, and adopting.

Let’s look at an example of creating a next-generation e-commerce application. To serve a large global audience, this application should be highly scalable and deliver optimistic performance, be available on various devices, support a rich and clean user experience, include real-time data analytics and reports, be cost-effective and easy to maintain, and more. To deliver this e-commerce solution, we could turn to a wide range of technical options. We could use the traditional ASP.NET MVC in combination with JQuery, SQL Server, HDInsight, and Power BI. Other options include a LAMP stack with Hadoop Spark or a MEAN stack with R-powered analytics. The list goes on. We must carefully evaluate all potential solutions and then continue with the design and development process.

Many proven methodologies in the software industry can help us in the various phases of application development. These methodologies can be considered as frameworks for planning and controlling the process of developing software. Popular methodologies include the waterfall model, rapid application development, agile software development, and the prototype model. Different software organizations and projects use different methodologies based on their engineering and operational criteria. In fact, some organizations use their own tailored hybrid development methodologies and processes. Regardless of which software methodology we follow, we have to adhere to certain phases that are common to all methodologies. We will follow these phases, depicted in Figure 1-1, throughout this book as we journey through application development.

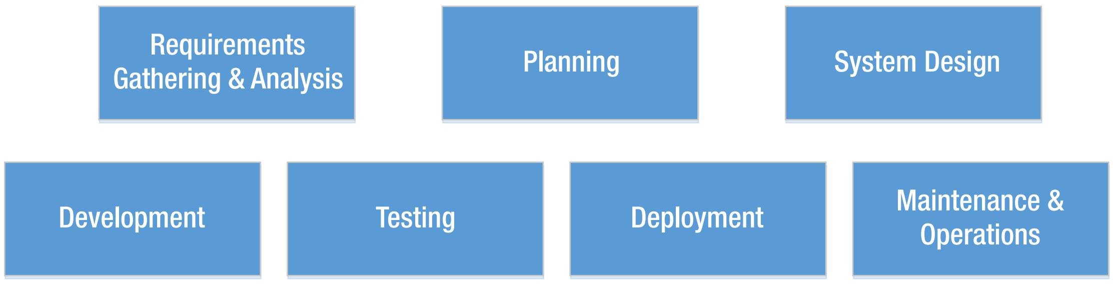


Figure 1-1. *Phases of the software development life cycle*

Requirements Gathering and Analysis

In this phase, we capture all the functional requirements of the software that we are planning to build. We create a brief functional specification document and then share it with all stakeholders.

Planning

During this phase, we start by augmenting staff and creating timelines for the project. Although it's not mandatory, we can organize all functionalities into logical groups and place them in different iterations, or *sprints*. We document the proposed project plan along with critical timelines as part of this phase's deliverables.

System Design

In this phase, we design the entire system's architecture. We have to list all technologies, tools, and commercial off-the-shelf (COTS) products that will be used during application development. The key deliverables of this phase are the specifications for the logical and physical architecture.

Development

The actual coding and development of the software is done during this phase. This phase may follow its own development strategies, such as test-driven development (TDD) or domain-driven development(DDD). In TDD, we write unit tests before developing the application code; so at first, all the unit tests will fail, and then we have to add or refactor application code to pass the unit tests. This is an iterative process that happens every time a new functionality is added or an existing one is modified.

Testing

During this phase, we integrate all the components of the software (for example, a web application might consume web API endpoints that are hosted differently, and these should be integrated as part of the testing phase). Then we test the application. The testing process usually comprises system integration testing (SIT) followed by user acceptance testing (UAT). Testing should be an iterative and ongoing activity that makes the application more stable and robust.

Deployment

Deployment activity includes all the procedures required for hosting the application on staging and production environments. Modern source-version-control systems support automated deployments to configured environments. Having said that, manual monitoring (for example, DEV, SIT, UAT, STAG, and PROD) should occur while promoting builds across various environments.

Maintenance and Operations

After the application is hosted and made available to the audience, unanticipated issues and errors (or even code-level exceptions) may arise, for multiple reasons. Maintenance is the phase when we closely monitor the hosted application for issues and then release hotfixes and patches to fix them. Operational tasks such as software upgrades on servers and backups of databases are performed as part of the maintenance activity.

As this book walks you through the end-to-end development of a web application, it will touch on every phase of application development except testing.

■ **Note** As mentioned previously, it is not my intention to proclaim that the cited methodologies are the *only* widely accepted options. But the preceding application development phases are the most common and frequently used practices for software projects.

Introduction to the Automobile Service Center Application

In this book, we'll take a near-real-world business requirement and deliver an end-to-end technical solution. I believe that this type of demonstration will help you understand the concepts better because of their practical implementations. The step-by-step process of building the web application will help you remember the concepts you've learned in each chapter.

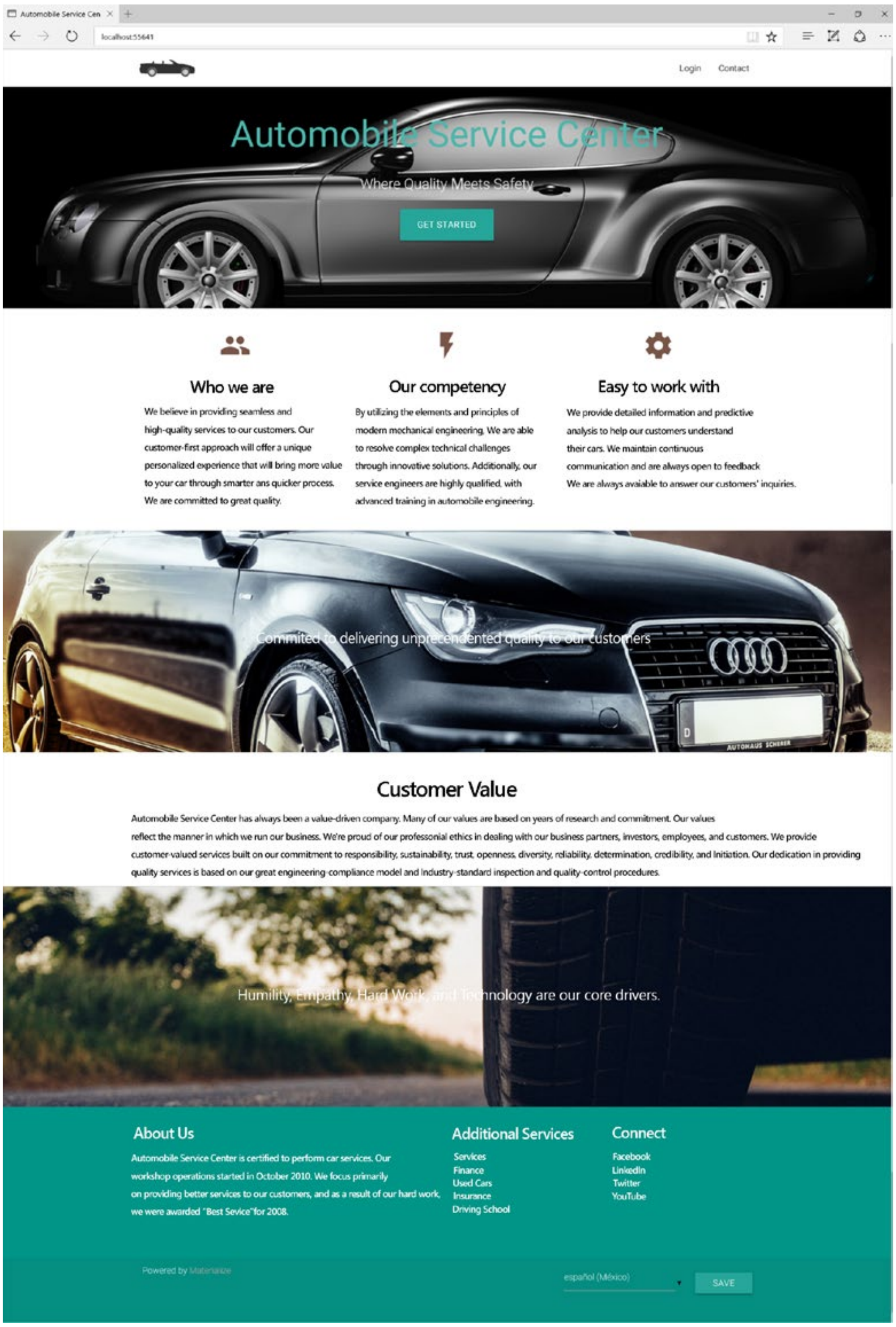
The Automobile Service Center is a fictitious US company that provides all types of car maintenance services to its customers. This company has more than 20 branches across the West Coast and serves more than 3,000 customers a day. It provides a personalized experience to its end customers in all areas related to car repairs and maintenance.

The Automobile Service Center provides car maintenance services in the following areas: Engine and Tires, Air Conditioning, Auto and Manual Transmissions, Electrical Systems, Batteries, Brakes, and Oil Changes. This Service Center also takes care of specific customer requests and complaints about the cars. At times it provides additional services including towing, pickup and drop, insurance, and financial-related offerings. This Service Center employs more than 1,500 professionals in various capacities. Most employees are service engineers who are responsible for the daily service operations, while other employees take care of responsibilities such as ordering spare parts, managing logistics, and running financial operations.

Today the Automobile Service Center is facing a major business challenge: maintaining good customer communication. The company lacks an affordable and reliable method of real-time communication with customers (for example, live chats between customers and service engineers, or e-mail/text notifications about service updates). A majority of customers want real-time service updates and automated service appointments. Currently, the company requires its customers to walk in to a Service Center branch and discuss the services with an available service engineer, who opens a service job card that documents the details. During the service work, all the communications between the service engineer and the customer occur through phone calls, and the points discussed are attached back to the job card. After the service is done, the customer pays the invoice amount, and the service engineer closes the job card. The Automobile Service Center has identified that this entire process is inefficient and is diminishing its brand value for the following reasons:

- Causes service delays because of a high dependency on the human factor
- Lack of transparency
- Creates opportunities for miscommunication
- Provides no precision in effort and estimations
- Doesn't enable the company to reach out to customers and promote great deals, discounts, and packages

Throughout this book, we'll help the Automobile Service Center overcome these challenges by creating a modern web application designed with cutting-edge technologies. By end of this book, we'll have a fully functional, production-ready web application, as shown in Figure 1-2.



Who we are

We believe in providing seamless and high-quality services to our customers. Our customer-first approach will offer a unique personalized experience that will bring more value to your car through smarter ans quicker process. We are committed to great quality.

Our competency

By utilizing the elements and principles of modern mechanical engineering, We are able to resolve complex technical challenges through innovative solutions. Additionally, our service engineers are highly qualified, with advanced training in automobile engineering.

Easy to work with

We provide detailed information and predictive analysis to help our customers understand their cars. We maintain continuous communication and are always open to feedback We are always avaiable to answer our customers' inquiries.

Commited to delivering unprecendented quality to our customers

Customer Value

Automobile Service Center has always been a value-driven company. Many of our values are based on years of research and commitment. Our values reflect the manner in which we run our business. We're proud of our professonial ethics in dealing with our business partners, investors, employees, and customers. We provide customer-valued services built on our commitment to responsibility, sustainability, trust, openness, diversity, reliability, determination, credibility, and Initiation. Our dedication in providing quality services is based on our great engineering-compliance model and Industry-standard inspection and quality-control procedures.

Humility, Empathy, Hard Work, and Technology are our core drivers.

About Us

Automobile Service Center is certified to perform car services. Our workshop operations started in October 2010. We focus primarily on providing better services to our customers, and as a result of our hard work, we were awarded "Best Sevice"for 2008.

Figure 1-2. *Home page of the Automobile Service Center application*

Throughout this book, we are going to use the same business use case and enhance the web application step-by-step. Let's get started by defining the scope of the application.

Scope of the Application

Any business challenge—whether it's simple or complex, big or small, mission-critical or optional—cannot be solved unless we have clear understanding of the functional requirements. Identifying the core features of an application is crucial for a successful implementation.

The Automobile Service Center application could have very complex requirements unless we carefully control them by defining their scope. Table 1-1 depicts the requirements of the Automobile Service Center application that are in the scope of this book's demonstration.

Table 1-1. *Scope of Work for Automobile Service Center Application*

Module	Functionality
User Administration	On the application's startup, an Admin user should be provisioned. Admin is a user who has access to the entire application and can manage other users; for example, provisioning service engineers. The Admin user can manage all customer information and the master data of the application.
	The application should be capable of allowing customers to log in by using their social media logins (for example, Gmail). On a user's first successful login, a local user account (specific to this application) should be created for moderation purposes.
	User management activities include customer registration, reminding users of or changing passwords, deactivating users (the Admin user can perform this action), signing out, assigning roles to service engineers and customers at registration.
	The Admin user should be able to provision other employees into the system.
Service Requests Management	The customer should be able to create a new service request. The Admin user should associate the service to a service engineer. Whenever the status of a service request changes, the customer should be notified via e-mail/text.
	Based on the service progress, the Admin user and service engineer can modify the job card details.
	The customer, service engineers, and Admin user should have respective dashboards displaying a summary of all activities and the latest updates.
Service Notifications	The customer, service engineer, and Admin user can view the summary of the job card.
	At any point during servicing, the customer, service engineer, and admin user can have two-way communication about the service progress and queries.
	The customer, service engineer, and Admin user can see the history of the job card (all the changes that have happened since the card's creation).
	The Admin user should be able to review services that are marked as completed by the service engineers.
	Note: Invoices and payment processing are beyond the scope of this book.
Promotions	The Admin user should be able to add details about new promotions and offers. Customers should receive real-time notifications with these details.
	Customers should be able to see all promotions and offers.

■ **Note** The requirements for this application are limited so that the application remains easy for you to understand. The limited scope of requirements also gives me the liberty to emphasize the technological concepts.

Technologies Used in Building the Application

The fundamental power of the software industry comes from its diverse technical advancements and offerings. During the last few years, the substantial growth of cloud computing and open source communities contributed to a significant number of technical programming languages, frameworks, and platforms. It is always difficult to learn a technology for a particular requirement because that requires a lot of research and comparisons.

Microsoft open sourced its ASP.NET MVC, Web API, and Web Pages frameworks under the Apache 2.0 license. Microsoft also established the .NET Foundation, an independent organization that aims to improve open source software development and collaboration around the .NET Framework. *ASP.NET Core*, the latest release of ASP.NET under the .NET Foundation, is capable of building cross-platform web and cloud solutions.

In this book, we'll use ASP.NET Core (Long Term Support version, 1.0.3) in combination with C# to build our web application for the Automobile Service Center. *Materialize*, an open source CSS framework based on Material Design, is used to create CSS themes for the Automobile Service Center application. JQuery will be used throughout this book to perform the necessary client-side activities. The *SignalR* library will provide real-time, two-way communication between server and clients (browsers, in this case). The OAuth 2.0 protocol is used to enable single sign-on from various external social media identity providers such as Gmail, Twitter, and Facebook.

Microsoft Azure Storage is an enterprise-grade cloud storage service that provides reliable, scalable, and durable data-storage services. In this book, we'll primarily use Azure Table storage to store all application data in NoSQL format. Figure 1-3 shows all the technologies and platforms that we'll use in designing and developing the Automobile Service Center application.

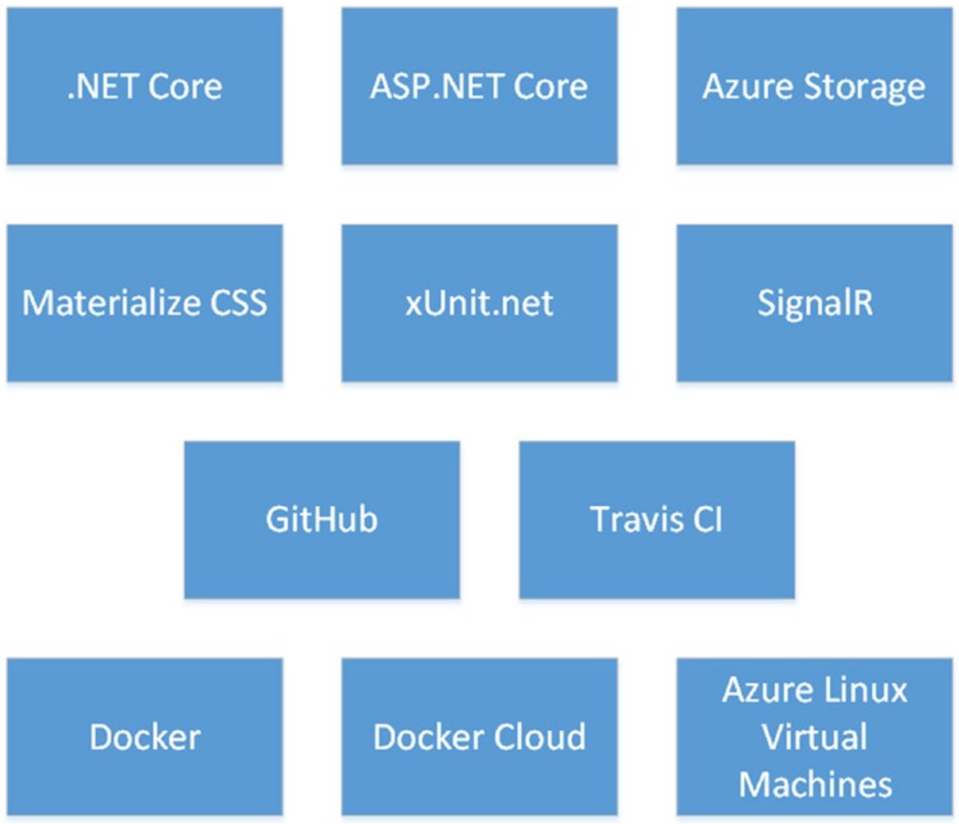


Figure 1-3. *Technologies used in building our Automobile Service Center application*

The Automobile Service Center application's code is versioned using *GitHub*, a distributed version-control and source-code management platform. To ensure code quality, we use *Travis CI* to build and test the source code from GitHub.

Docker is a container technology used to build software packages that can be deployed in any environment. The Automobile Service Center application is packaged using Docker by continuous integration with GitHub. *Docker Cloud*, which supports continuous deployment to any infrastructure, is used to deploy our web application containers to *Azure Linux Virtual Machines*.

Logical Architecture of the Application

The logical architecture is a design specification that depicts all the logical code components of the application and their interactions. Figure 1-4 illustrates the logical architecture of the Automobile Service Center technical solution.

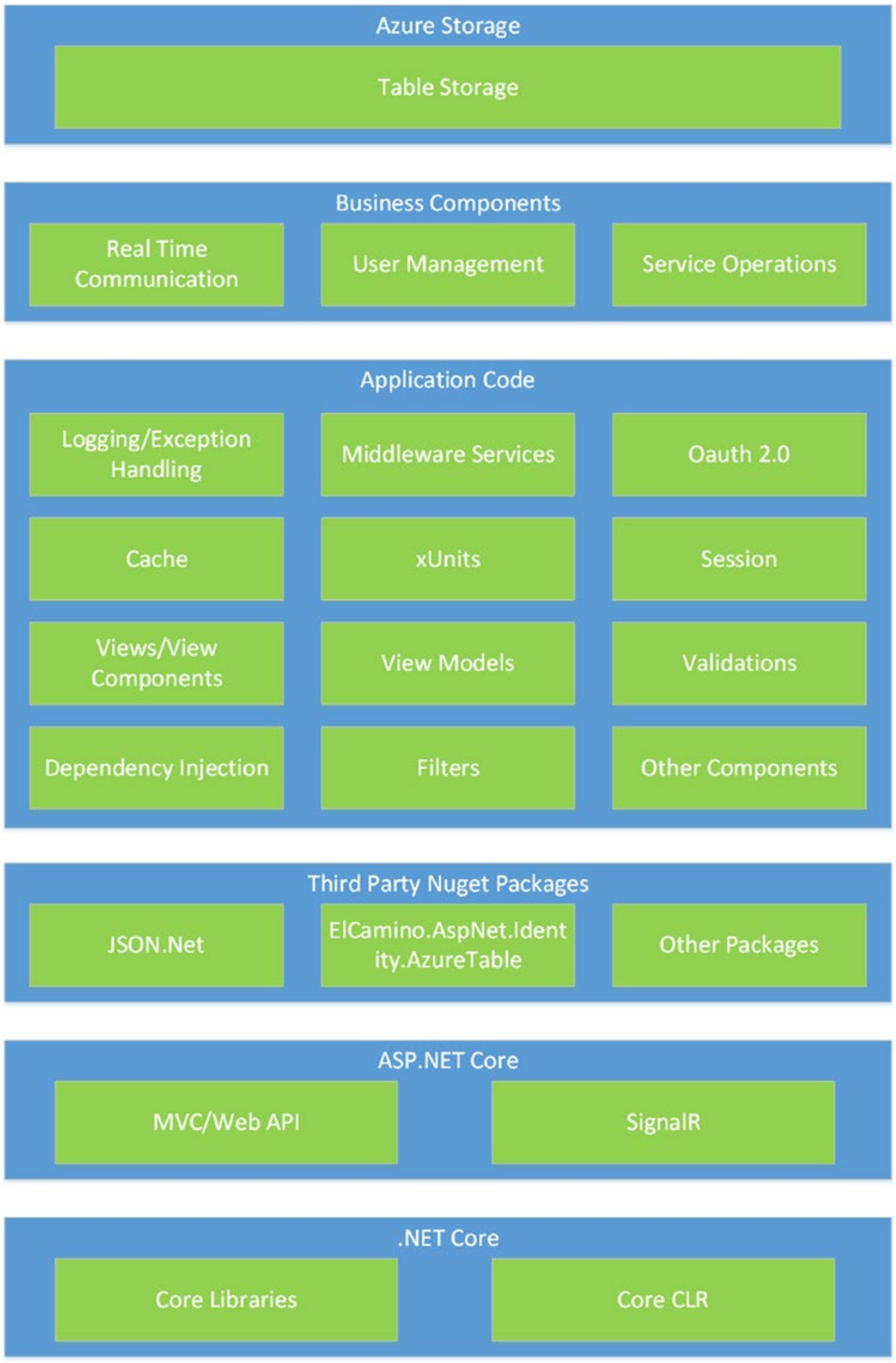


Figure 1-4. *Logical architecture of the Automobile Service Center technical solution*

The base foundation of the application is the Microsoft .NET Core runtime and its libraries. On top of .NET Core, we use the ASP.NET Core framework, which provides all the features related to web development. Microsoft designed the entire .NET Core ecosystem to be modular; everything is driven by NuGet packages, and we can choose which packages we need for the application. So the next step in the hierarchy is to get all the required NuGet packages to build the application.

The next layer in the hierarchy is custom application code, written by developers, that is required for web development. This layer contains all the artifacts related to MVC such as views, controllers, filters, and stylesheets. This layer also holds the key implementations such as session management, logging, and exception handling. This layer also houses the test project, which will hold all the xUnit.net test cases for the entire application.

Next is the Business Components layer, which holds all the code related to business requirements. This layer has a modular design by segregating the relevant code through interfaces into multiple physical projects. For example, storage operations and business functions are driven through interface design.

Azure Storage, the last layer of the hierarchy, is responsible for all data operations. This API is a C# wrapper for HTTP REST service calls to the Azure Storage infrastructure that's provided by Microsoft through a NuGet package.

The Automobile Service Center application is going to be a loosely coupled system; the Business Components layer requires Azure Storage dependencies, and the Web layer is dependent on business components. These dependencies are injected using dependency injection at runtime (ASP.NET Core supports dependency injection by default). This way, there is no need to create the instances of the dependencies within the code, as they are created and managed by the IoC containers.

■ **Note** The detailed implementation steps of each layer, along with dependency injection, are covered in Chapter 2.

Continuous Integration and Deployment Architecture

In today's application development, continuous integration (CI) and continuous deployment (CD) are the key DevOps practices. These operations will prevent manual effort and repetitive tasks on every build, and thereby reduce the possibility of human error, improve the quality of the build, and increase the efficiency of the overall life cycle of developing, testing, and deploying.

The Automobile Service Center application is going to use the latest CI and CD technologies and tools, as shown in Figure 1-5. The process begins with the developer, who designs and develops the application on a local machine by connecting to the local machine's Azure Storage emulator. On successful completion of a feature, the developer commits the code to the Dev branch (which is not depicted in the image) of the GitHub source-code repository. On a successful commit, the Travis CI service triggers a build to validate the entire source code for errors and to run xUnit test cases. If the build fails, the developer and all other stakeholders are notified via e-mail, and it is the developer's responsibility to correct the error and commit the code back to the Dev branch. This is an iterative process for all the developers during the development phase.

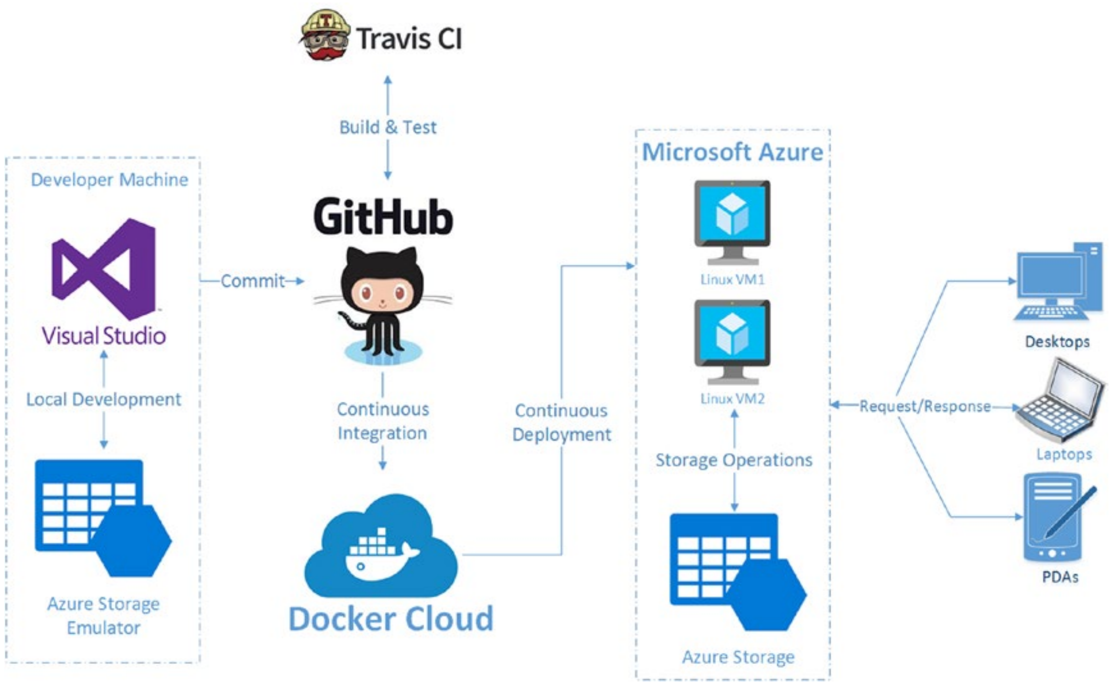


Figure 1-5. *Continuous integration and deployment architecture*

At a given logical time, when we decide to roll out the completed features to production servers, the Dev branch will be merged with the master branch. The Travis CI service builds again kicks off on master branch and validates the build. The same iterative process continues on the master branch as well.

■ **Note** The concepts described in this book are one way to achieve CI and CD. There are many ways to achieve CI and CD using different technologies. I advise developers to do thorough research and analysis of the contemporary CI and CD technologies and tools and to check their relevance in a project context before implementing these mission-critical strategies.

Deployments to production servers are triggered from the master branch. In this book, we use Docker technology to create application containers and deploy them to Azure Linux Virtual Machines. Whenever a merge or commit happens at the master branch at GitHub, Docker Cloud will take the latest source code, build a Docker image, create a Docker container, and finally push the container to the associated Azure Virtual Machine nodes.

■ **Note** The implementation details of CI and CD processes for an ASP.NET Core application are described in a later chapter.

Software Prerequisites

Before we start the development process, let's check all the software prerequisites and quickly set up the development machine.

■ **Note** The software prerequisites listed in this section are the preliminary and basic software required to get started. I will discuss in detail the required technologies and tools in their respective chapters. This way, every chapter is self-contained, and your need to search for relevant content throughout the book is reduced.

I am using the following development machine:

Dell Latitude E7450 x64-based PC with Intel Core i7-5600U CPU @ 2.60GHz, 2594MHz, 2 cores, 4 logical processors, 8GB physical memory, 476GB SSD hard drive.

■ **Note** It is not mandatory to have the same configuration as my development machine.

The operating system should be 64-bit Windows 10 Pro, Enterprise or Education (version 1511, November update, Build 10586 or later).

■ **Note** The Windows 10 OS Build version (Build 10586 or later) is crucial for running Docker on the Windows platform. One more important point is that Windows 10 Virtual Machine on Azure can't be used for Docker development because it doesn't support nested virtualization (at the time of writing this chapter).

Download and install Visual Studio Community 2017 from `www.visualstudio.com`. I am using the 15.1 (26403.7) release.

We need to install the following Visual Studio workloads: ASP.NET Web Development, Azure Development, and .NET Core Cross-Platform Development.

■ **Note** It is always advisable to sign into Visual Studio. Signing in will always sync your Visual Studio settings to your Microsoft account, and these settings can be retrieved later on any other machine by signing in with the same account. Starting from Visual Studio 2017, the new Roaming Extension Manager will keep track of all your favorite extensions across all development environments and sync them with Visual Studio sign-in.

We also need to have the following accounts created:

- GitHub (`https://github.com/`)
- Travis CI (`https://travis-ci.org/`): Associate Travis CI with GitHub by signing in with GitHub credentials.
- Docker (`www.docker.com`)
- Microsoft Azure Subscription (`https://azure.microsoft.com/`): Create a free Azure Account.

Other tools are required for development, including Docker for Windows, the latest .NET Core SDK, and the Azure Storage emulator. The details of these tools and SDKs are discussed in corresponding chapters.

Summary

In this chapter, you learned about the importance of, and paradigm shift toward, open source communities and cloud platforms in developing modern software applications. You also learned about the major software development strategies and procedures. Regardless of which software methodology we use, the basic processes remain the same, with a little tweak in implementation.

You were introduced to the business requirements of the fictitious Automobile Service Center. We defined the scope of the development work for the rest of the book.

You briefly looked at the technical stack—the combination of ASP.NET Core, SignalR, Azure Storage, GitHub, Travis CI, and Docker—which we are going to use to develop the application. We designed the logical architecture and CI/CD pipeline of the application. Finally, the chapter concluded with the details of the required environment and software prerequisites.

CHAPTER 2

The New Era of .NET Core

Over the last decade, the open source development model gained significant adoption and tremendous growth because of its ability to change and improve software products by making them available under a free license to the general public. The process of ideation and innovation in software development is at its peak because people around the world have been able to collaborate and create new products from open source code.

Microsoft's move toward the open source world started two years ago. Its strong commitment to the open source community was demonstrated by joining the Linux Foundation at the end of 2016. Microsoft's efforts— in adapting Linux on Azure Platform, running Bash on Windows, making the developer IDEs such as Visual Studio and VS Code available on various OSs, and enabling compatibility with SQL Server on Linux—made an extraordinary impact on open source communities and provided opportunities to create modern-day frameworks and tools. By far, Microsoft's decision to open source its .NET Framework was a game changer in the software industry, and .NET Core is the kickstarter of its open source journey.

In this chapter, you will learn the key features of .NET Core and ASP.NET Core, especially the advantages these frameworks offer compared to their predecessors. We will create an ASP.NET Core project that will serve as the baseline version of the Automobile Service Center application. You will learn the important project artifacts. You'll also see how to set up the configuration of the Automobile Service Center web project through the `Appsettings.json` file. Finally, we will conclude this chapter by exploring the new dependency injection system provided by the ASP.NET Core framework.

Introduction to .NET Core

.NET Core is an open source development platform maintained by Microsoft and the .NET community at GitHub under the .NET Foundation. (The .NET Foundation is an independent organization that supports open source development and collaboration around the .NET ecosystem.)

.NET Core possesses the following primary features:

- *Cross-platform*: Runs on Windows, macOS, and Linux. If the application's requirement is to run across multiple platforms, .NET Core is the obvious choice among Microsoft frameworks.
- *Compatible*: .NET Core is compatible with .NET Framework, Xamarin, and Mono, via the .NET Standard library.
- *Open source*: The .NET Core platform is open source under MIT and Apache 2 licenses.
- *Modular design*: In .NET Core, everything is a package. Unlike getting everything, even when it's not required for an application, we get only the required packages. All package dependencies can be retrieved from NuGet.

- *Command-line-style development*: .NET Core is designed for the command-line interface (CLI). It provides command-line tools available on all supported platforms, enabling developers to build and test applications with a minimal installation on developer and production machines.
- *Container support*: .NET Core by default provides support to container technologies such as Docker.
- *Performance*: .NET Core is designed to deliver optimistic performance and high scalability for an application.
- *Microservices-oriented design*: .NET Core is built to support microservices and can easily get along with other microservices built using .NET Framework or Java, for example.
- *Side-by-side support to other .NET versions at an application level*: .NET Core supports easy, side-by-side installation of different versions on the same machine. This allows us to run multiple services on the same server, targeting different versions of .NET Core.
- *Microsoft support*: .NET Core is supported by Microsoft.

■ **Note** Organizations should perform careful research and analysis before opting for .NET Core over .NET Framework.

As of now, .NET Core is a subset of .NET Framework, and not all the .NET Framework APIs are ported to .NET CoreFX libraries.

.NET Core doesn't support all the workloads, such as ASP.NET Web Forms or WPF.

.NET Core is relatively new, and most of the third-party NuGet packages and platforms don't support .NET Core yet.

Figure 2-1 shows the.NET ecosystem.

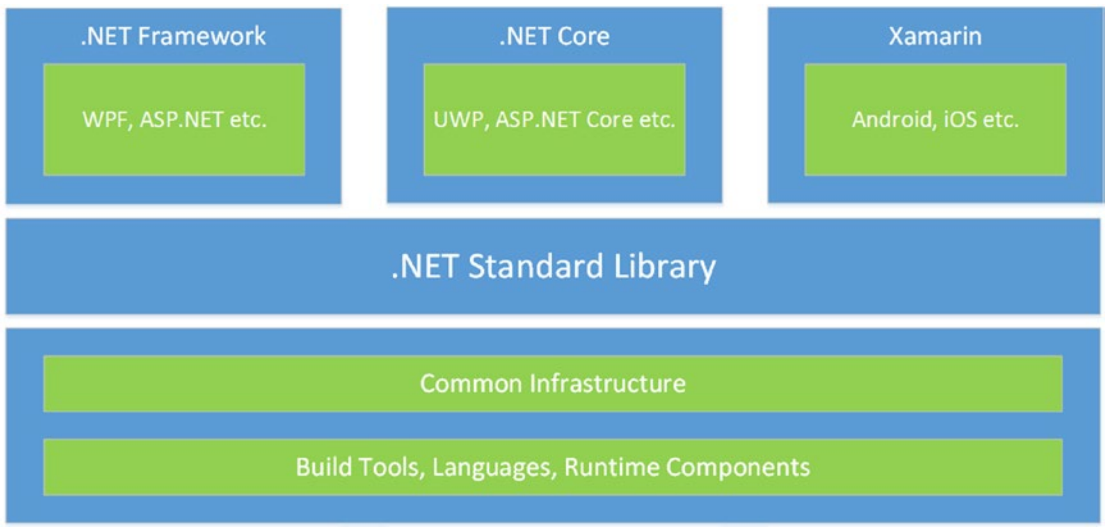


Figure 2-1. *.NET ecosystem architecture*

In this architecture diagram, the bottom layer corresponds to languages and their compilers, runtime components (for example, JIT and garbage collectors), and build and CLI tools required to build any .NET application.

On top of that layer, we have the .NET Standard library, which is a specification of .NET APIs that make up a uniform set of contracts with underlying implementations for each .NET runtime. Typically, in a traditional .NET system, this layer constitutes most of the APIs from `mscorlib.dll`. If application code targets a version of.NET Standard, that application is guaranteed to run on any .NET runtime that implements that version.

The top layer consists of various runtimes of the .NET ecosystem (for example, .NET Core, .NET Framework, and Mono for Xamarin). The .NET Core runtime supports ASP.NET Core and Universal Windows Platform workloads. Similarly, the traditional .NET Framework runtime supports ASP.NET, WPF, and Windows Forms. The Mono runtime is used by Xamarin to build cross-platform mobile applications for Android and iOS platforms.

Introduction to ASP.NET Core

ASP.NET Core is a new, open source, cross-platform framework for building modern web applications. It is designed from the ground up to empower applications for the Web, IoT, cloud, and mobile services. As depicted in Figure 2-2, the other major advantage of an ASP.NET Core application is its ability to run on either .NET Core or the full .NET Framework. ASP.NET Core uses a modular design and is driven through NuGet packages, so rather than housing all the unwanted references, we need to get only the packages that are required for our application.

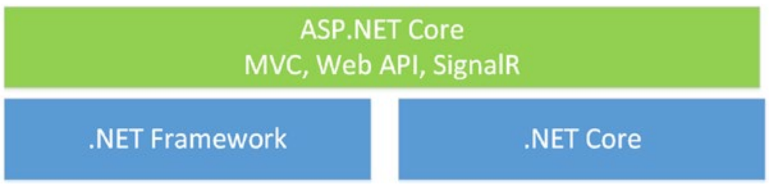


Figure 2-2. *Compatibility of ASP.NET Core with .NET Framework and .NET Core*

ASP.NET Core provides the following foundational improvements and capabilities when compared with the traditional .NET Framework–based ASP.NET Framework:

- *Open source*: ASP.NET Core is open source under the Apache 2 license.
- *Cross-platform support*: ASP.NET Core applications can be developed and run on Windows, macOS, and Linux.
- *Unified MVC and Web API approach*: In ASP.NET Core, both the MVC and Web APIs are driven with unified object model. A single namespace serves both the MVC and Web APIs, which removes confusion from older versions of ASP.NET.
- *Lightweight HTTP request pipeline*: Compared to previous versions of ASP.NET, most of the unwanted overhead has been removed from ASP.NET Core, resulting in a lightweight HTTP pipeline. We can construct the HTTP pipeline by using the default framework provided by middleware or through custom-developed middleware.

- *Modular design*: Starting with ASP.NET Core, there is no need to manage any unwanted packages or references in the project (there is no `System.Web.dll`). We need to get only those references that we want for the application. In fact, ASP.NET Core itself is a NuGet package.
- *Integration with client-side frameworks*: ASP.NET Core applications can easily integrate with client-side frameworks such as AngularJS, KnockoutJS, Gulp, Grunt, and Bower.
- *Improved performance*: ASP.NET Core's modular design and lightweight HTTP pipeline offer tremendous performance.
- *Dependency injection*: ASP.NET Core by default supports dependency injection.
- *Cloud-ready solutions*: ASP.NET Core solutions are optimized for cloud compatibility through an easy configuration system.
- *Runs on both .NET Framework and .NET Core*: ASP.NET Core applications can run on both .NET Framework and .NET Core runtimes. We have the flexibility of using traditional .NET Framework APIs that are not yet available in .NET Core. If the application is targeted for .NET Core runtime, it can support all the .NET Core versions available on the server.
- *Hosting*: Easy to self-host the ASP.NET Core application. It also supports IIS hosting.
- *Tooling*: ASP.NET Core applications can be built using Visual Studio Code, the Visual Studio IDE, or the .NET Core CLI. This new tooling enhances developer productivity in building, publishing, and creating Docker containers.

From a development perspective, many new features and groundbreaking changes have been introduced in ASP.NET Core. Some of the features are as follows.

- The ASP.NET Core Visual Studio project file is a rewrite from the ground up in a new format.
- There is no `Global.asax` or application startup events in ASP.NET Core. All configuration and streamlining of the HTTP pipeline is done at the `Startup.cs` class.
- Writing custom middleware is very easy in ASP.NET Core applications.
- The ASP.NET Core application is a pure console application. It has an application main entry point at the `Program.cs` file.
- ASP.NET Core uses a Kestrel server designed specifically for ASP.NET Core.
- The static content (such as JavaScript, stylesheets, and images) are now placed under the `wwwroot` folder of the application in ASP.NET Core.
- Bundling and minification is achieved through the `bundleconfig.json` file.
- The Roslyn compiler is used for in-memory compilation.
- Tag helpers and view components promote more robust, reusable, server-side HTML generation.
- Default support is provided for dependency injection, which can manage various object lifetime scopes.
- Attribute-based routing supports new controller and action tokens.

- Configuration based on `Appsettings.json.`
- New `_ViewImports.cshtml` file, which can be a common place for all namespaces across different views.
- New filter types and much better control on filter execution.
- Creating and managing unit tests is much easier.
- New way to manage secure information on development machine using user secrets.
- Better support for different caching providers.
- Data access using Entity Framework Core.
- Configuring authentication and authorization is easy using ASP.NET Core Identity.
- New improvement to internationalization, logging, and exception handling.

■ **Note** We cover most of these ASP.NET Core features in subsequent chapters.

Versions and Roadmap of .NET Core and ASP.NET Core

The development of .NET Core and ASP.NET Core is happening at a rigorous pace. There are many reasons for this fast-paced development; because .NET Core is a relatively new market offering, a lot of effort is going toward stabilizing the framework and adding new features and APIs. At the same time, .NET Core is open source and has to extend its footprint to multiple OS distributions.

■ **Note** My primary goal in writing this section is to help you understand the rapid cadence of .NET Core and familiarize you with the direction of its future releases. The complete release history of .NET Core, along with release notes, can be found at `https://github.com/dotnet/core/blob/master/release-notes/README.md`.

.NET Core has three major versions: 1.0., 1.1, and 2.0. The primary difference between .NET Core 1.0 and .NET Core 1.1 is the support for new operating system distributions and a new set of APIs added with the new .NET Standard. Both versions of .NET Core support side-by-side installation and adoption. .NET Core 2.0 targets .NET Standard 2.0, which has an extensive 32,000+ APIs. .NET Core 2 introduces a new development model to develop pages using Razor, which removes the dependency on ASP.NET controllers. There are lot of other performance improvements in .NET Core 2.0.

The rapid cadence in releasing new versions of .NET Core not only gave developers the flexibility of a new set of APIs, but also made them go through a roller coaster ride for the following reasons:

- The initial releases of .NET Core don’t cover the exhaustive list of .NET Framework APIs.
- VS tooling is not fully feature rich, compared to regular .NET Framework development.

- Major changes occurred in the .NET project system, such as moving away from a `Project.json`-based approach to a `csproj`-based approach.
- Different versions of .NET Core support different OS distributions.

Similar to .NET Core, ASP.NET Core versions significantly improved over time. The following are some of the major changes in ASP.NET Core 2.0:

- Support for NET Standard 2.0.
- `Microsoft.AspNetCore.All` meta package, which will house all the packages required to build an ASP.NET Core application.
- Support for .NET Core 2.0 Runtime Store and automatic Razor view precompilation during publish.
- Performance improvements to the Kestrel server.
- New improvements to ASP.NET Core Identity.
- Enhancements to configuration, caching, Razor, diagnostics, and logging.

Developers should watch out for GitHub repositories of both .NET Core and ASP.NET Core for new changes and should plan development strategies in advance. One way to plan things for upcoming releases is to become familiar with the daily builds of the .NET Core SDK (which gives the latest up-to-date .NET Core runtime) and MyGet packages (which are, again, daily builds of packages, and NuGet houses stable versions) and do proof-of-concepts in order to understand what is coming in subsequent releases.

■ **Note** Every developer should be aware that working with daily builds might result in unexpected behavior of an application. Bottlenecks can occur from VS tooling or the .NET Core CLI. Sporadic behavior could occur in the development environment, which might require uninstalling and reinstalling stable SDK versions and VS tooling.

Having said ups and downs, the recent releases of both .NET Core and ASP.NET Core are very stable, support numerous OS distributions, and offer an exhaustive set of APIs. The current releases were supported with great tooling from Visual Studio 2017 and the .NET Core CLI.

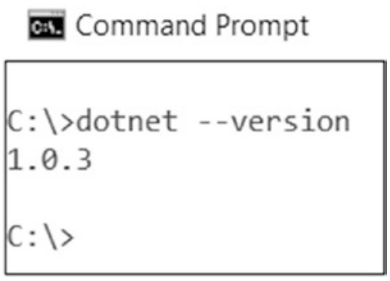
Creating the Automobile Service Center Application

In this section, we'll create an ASP.NET Core web application that will be used as the base solution for our Automobile Service Center application.

To get started with an ASP.NET Core application, we need the .NET Core SDK installed on a local machine.

By default, the .NET Core SDK will be installed with VS 2017. We can check the .NET Core SDK version as shown in Figure 2-3.

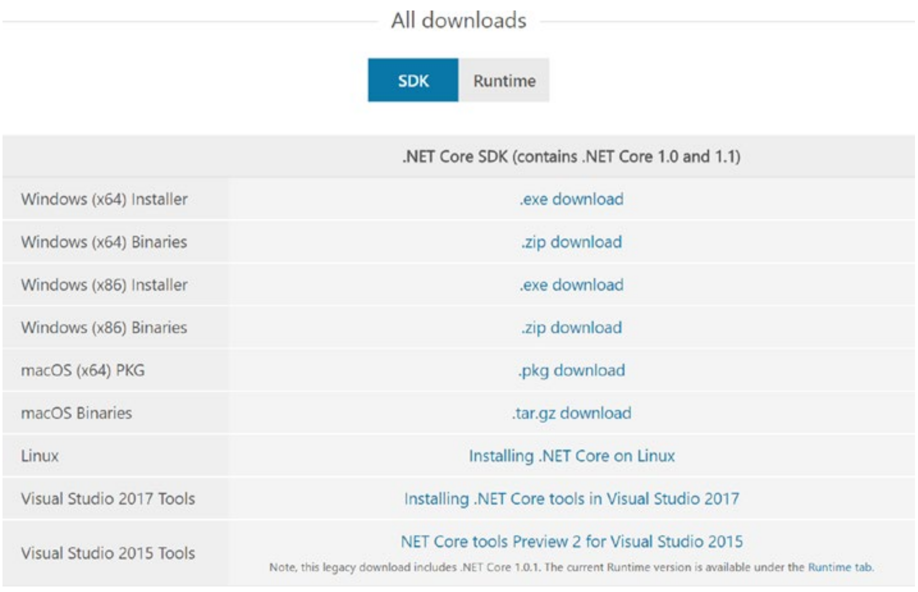
```
dotnet --version
```



```
C:\>dotnet --version
1.0.3

C:\>
```

Figure 2-3. *dotnet version*

■ **Note** New versions of the .NET Core SDK will be released on a regular basis. Install the new version by downloading it from `www.microsoft.com/net/download/core`. Download the x64 version of Windows Installer as shown in Figure 2-4.



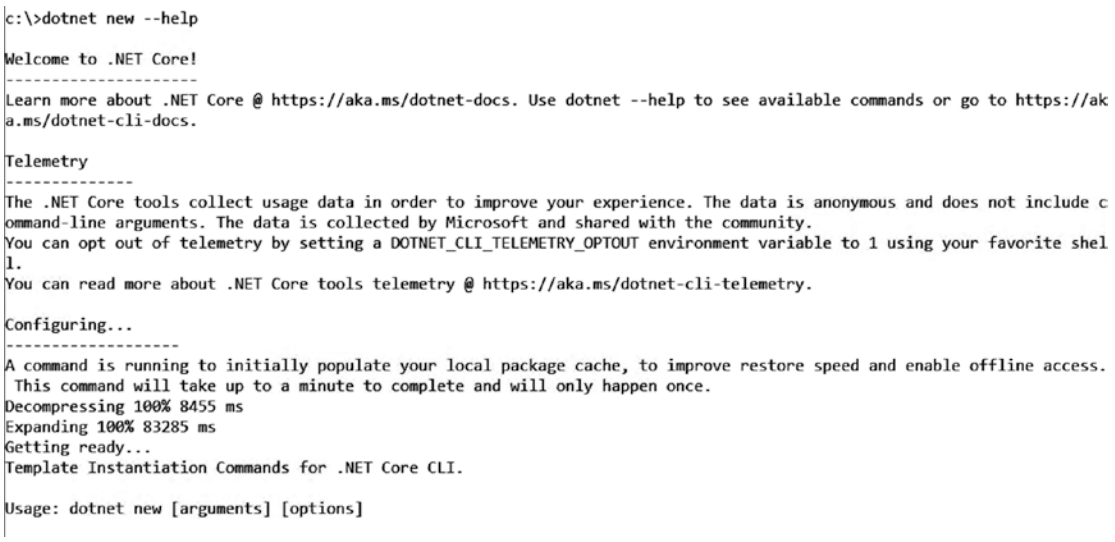
	.NET Core SDK (contains .NET Core 1.0 and 1.1)
Windows (x64) Installer	.exe download
Windows (x64) Binaries	.zip download
Windows (x86) Installer	.exe download
Windows (x86) Binaries	.zip download
macOS (x64) PKG	.pkg download
macOS Binaries	.tar.gz download
Linux	Installing .NET Core on Linux
Visual Studio 2017 Tools	Installing .NET Core tools in Visual Studio 2017
Visual Studio 2015 Tools	NET Core tools Preview 2 for Visual Studio 2015 Note, this legacy download includes .NET Core 1.0.1. The current Runtime version is available under the Runtime tab.

Figure 2-4. *.NET Core downloads (at the time of writing)*

Examine the sample usage of the `dotnet new` command shown in Figures 2-5 and 2-6.

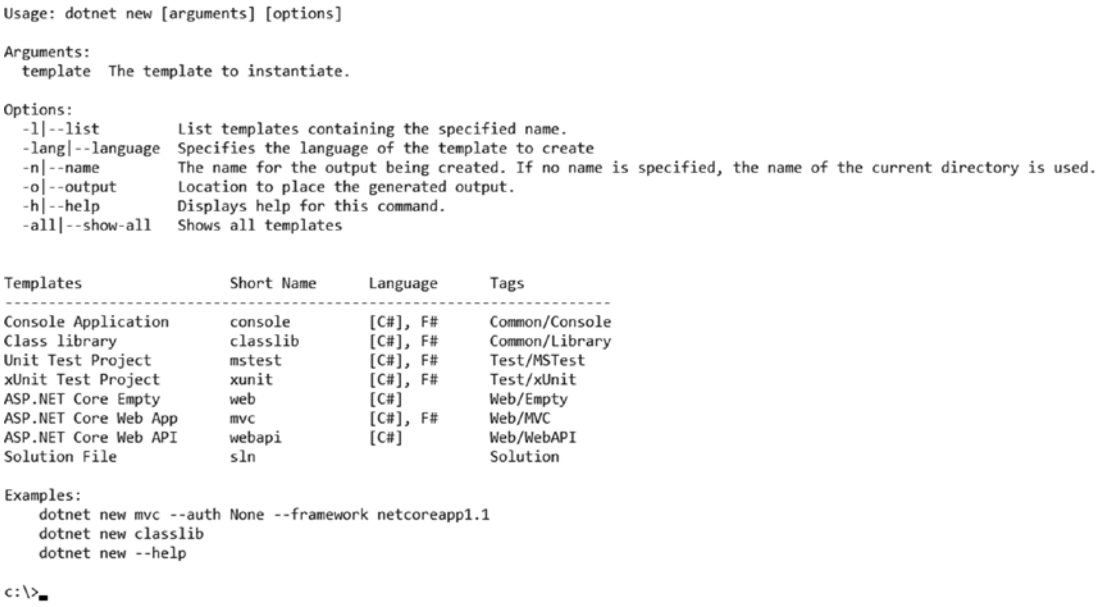
```
dotnet new --help
```

■ **Note** The initial run will populate the local package cache to improve performance.



```
c:\>dotnet new --help

Welcome to .NET Core!
---------------------
Learn more about .NET Core @ https://aka.ms/dotnet-docs. Use dotnet --help to see available commands or go to https://ak
a.ms/dotnet-cli-docs.

Telemetry
--------------
The .NET Core tools collect usage data in order to improve your experience. The data is anonymous and does not include c
ommand-line arguments. The data is collected by Microsoft and shared with the community.
You can opt out of telemetry by setting a DOTNET_CLI_TELEMETRY_OPTOUT environment variable to 1 using your favorite shel
l.
You can read more about .NET Core tools telemetry @ https://aka.ms/dotnet-cli-telemetry.

Configuring...
-------------------
A command is running to initially populate your local package cache, to improve restore speed and enable offline access.
 This command will take up to a minute to complete and will only happen once.
Decompressing 100% 8455 ms
Expanding 100% 83285 ms
Getting ready...
Template Instantiation Commands for .NET Core CLI.

Usage: dotnet new [arguments] [options]
```

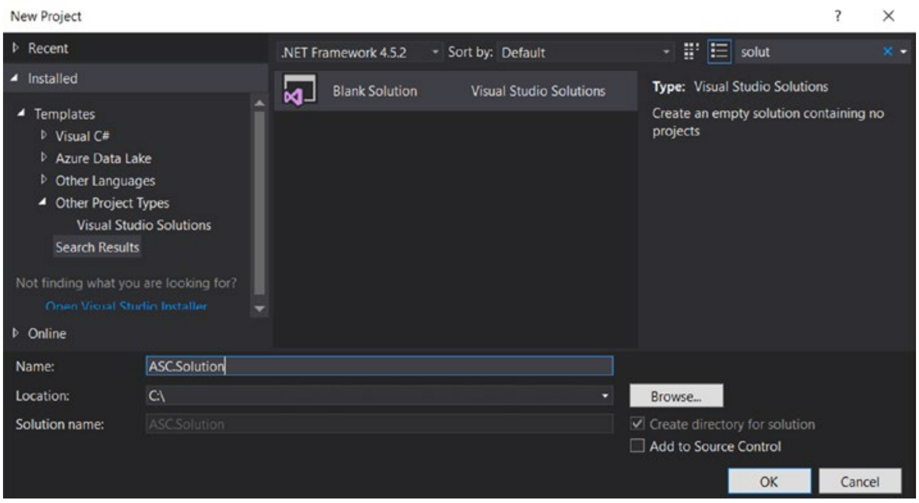
Figure 2-5. *dotnet new command usage*



```
Usage: dotnet new [arguments] [options]

Arguments:
  template  The template to instantiate.

Options:
  -l|--list         List templates containing the specified name.
  -lang|--language  Specifies the language of the template to create
  -n|--name         The name for the output being created. If no name is specified, the name of the current directory is used.
  -o|--output       Location to place the generated output.
  -h|--help         Displays help for this command.
  -all|--show-all   Shows all templates


Templates                 Short Name      Language      Tags
----------------------------------------------------------------------
Console Application       console         [C#], F#      Common/Console
Class library             classlib        [C#], F#      Common/Library
Unit Test Project         mstest          [C#], F#      Test/MSTest
xUnit Test Project        xunit           [C#], F#      Test/xUnit
ASP.NET Core Empty        web             [C#]          Web/Empty
ASP.NET Core Web App      mvc             [C#], F#      Web/MVC
ASP.NET Core Web API      webapi          [C#]          Web/WebAPI
Solution File             sln                           Solution

Examples:
    dotnet new mvc --auth None --framework netcoreapp1.1
    dotnet new classlib
    dotnet new --help

c:\>
```

Figure 2-6. *dotnet new command usage*

Open VS 2017. Create a new solution with the name `ASC.Solution`, as shown in Figure 2-7.

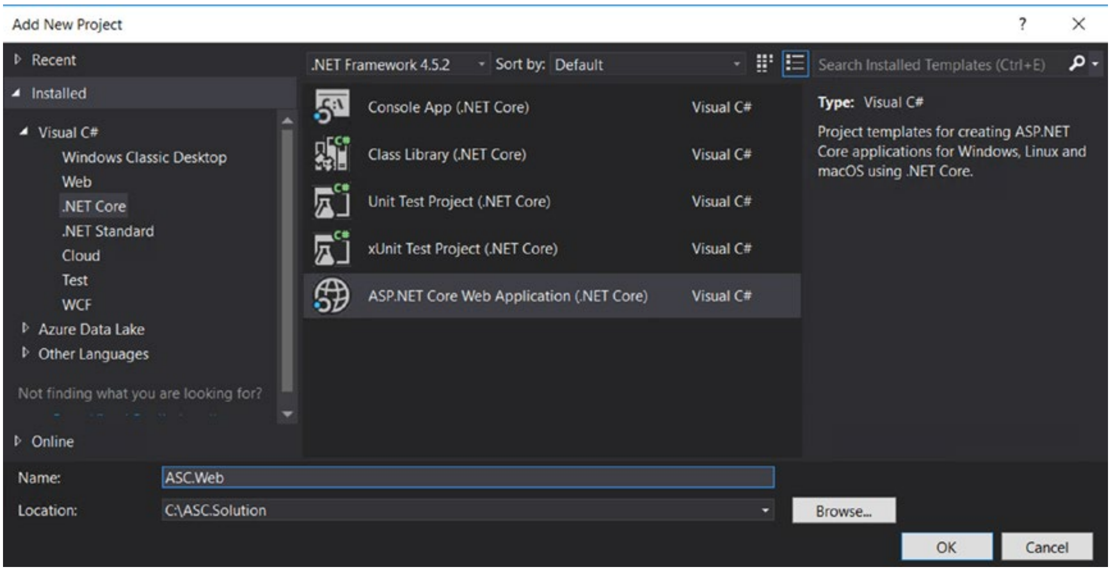


Type: Visual Studio Solutions
Create an empty solution containing no projects

Name: ASC.Solution
Location: C:\
Solution name: ASC.Solution
☑ Create directory for solution
☐ Add to Source Control

Figure 2-7. *New Visual Studio solution*

Create a new ASP.NET Core Web Application Project with the name `ASC.Web` in the solution we created (right-click the solution and select Add New Project, as shown in Figure 2-8).

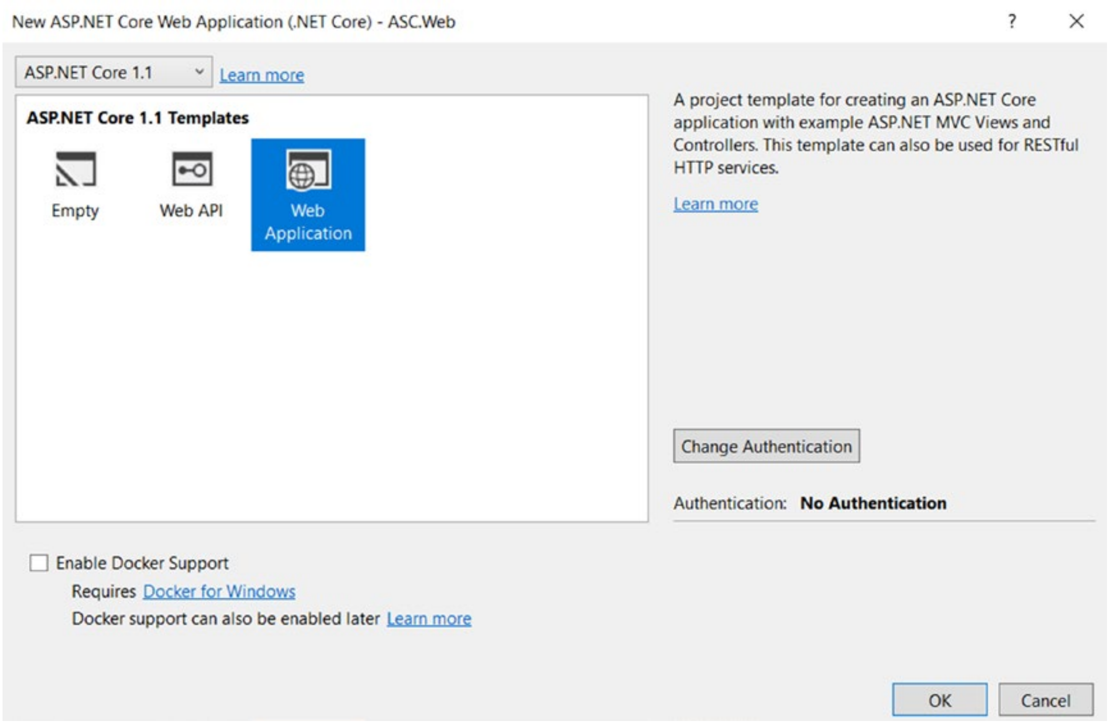


Type: Visual C#
Project templates for creating ASP.NET Core applications for Windows, Linux and macOS using .NET Core.

Name: ASC.Web
Location: C:\ASC.Solution

Figure 2-8. *New ASP.NET Core project in Visual Studio*

Click OK. A pop-up, shown in Figure 2-9, appears to allow the user to select the version of the ASP.NET Core framework and web template. Make sure to select ASP.NET Core 1.1 as the framework and Web Application template.


New ASP.NET Core Web Application (.NET Core) - ASC.Web

ASP.NET Core 1.1 Learn more

ASP.NET Core 1.1 Templates

A project template for creating an ASP.NET Core application with example ASP.NET MVC Views and Controllers. This template can also be used for RESTful HTTP services.

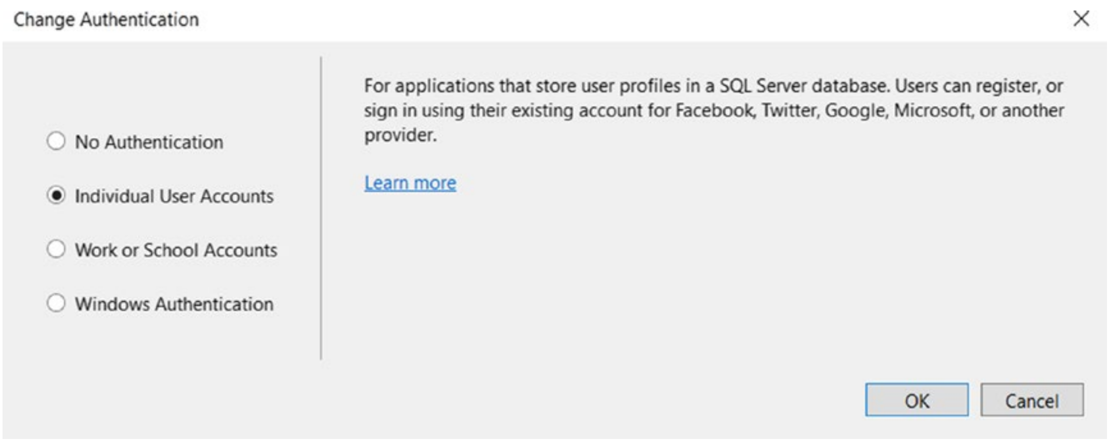
Learn more

Authentication: **No Authentication**

☐ Enable Docker Support
Requires Docker for Windows
Docker support can also be enabled later Learn more

Figure 2-9. *ASP.NET Core framework version and template*

Click the Change Authentication button to access the Change Authentication dialog box. Change the Authentication mode from none to individual user accounts, as shown in Figure 2-10. Then click OK.


Change Authentication

○ No Authentication
● Individual User Accounts
○ Work or School Accounts
○ Windows Authentication

For applications that store user profiles in a SQL Server database. Users can register, or sign in using their existing account for Facebook, Twitter, Google, Microsoft, or another provider.

Learn more

Figure 2-10. *Change authentication for the ASP.NET Core project*

After making sure all the settings are correct, click OK, and VS 2017 will create the web project that is shown in Figure 2-11.

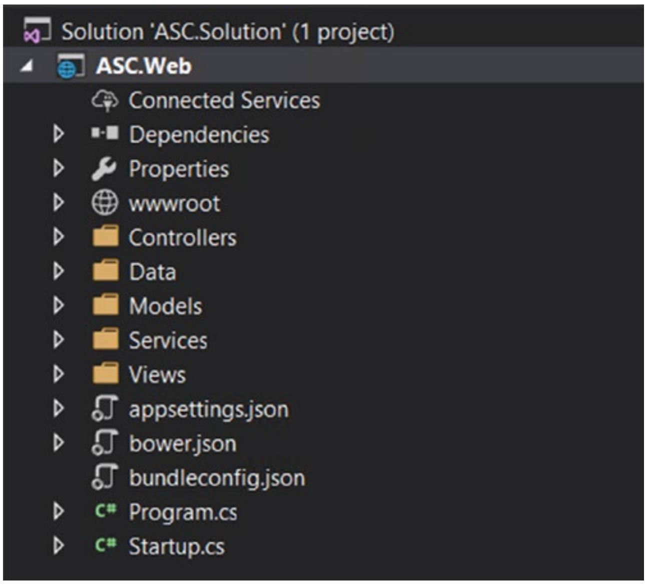


Figure 2-11. *ASP.NET Core project*

To run the web application, click IIS Express from the VS 2017 standard menu, as shown in Figure 2-12.

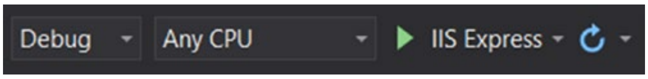
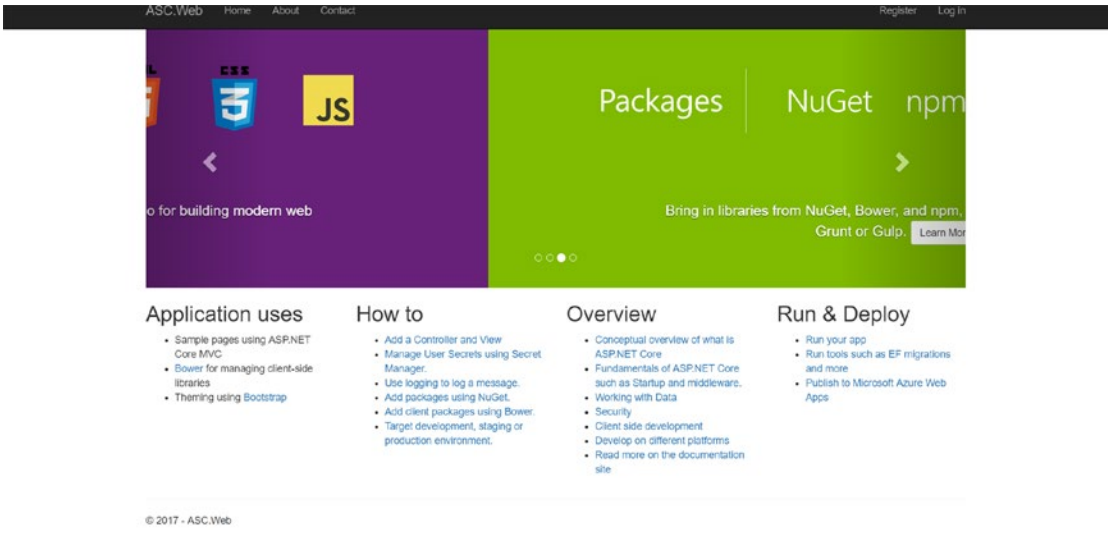


Figure 2-12. *Debug an ASP.NET Core project*

The application will be executed and opens in the default browser, as shown in Figure 2-13.



Application uses

- Sample pages using ASP.NET Core MVC
- Bower for managing client-side libraries
- Theming using Bootstrap

How to

- Add a Controller and View
- Manage User Secrets using Secret Manager.
- Use logging to log a message.
- Add packages using NuGet.
- Add client packages using Bower.
- Target development, staging or production environment.

Overview

- Conceptual overview of what is ASP.NET Core
- Fundamentals of ASP.NET Core such as Startup and middleware.
- Working with Data
- Security
- Client side development
- Develop on different platforms
- Read more on the documentation site

Run & Deploy

- Run your app
- Run tools such as EF migrations and more
- Publish to Microsoft Azure Web Apps

© 2017 - ASC.Web

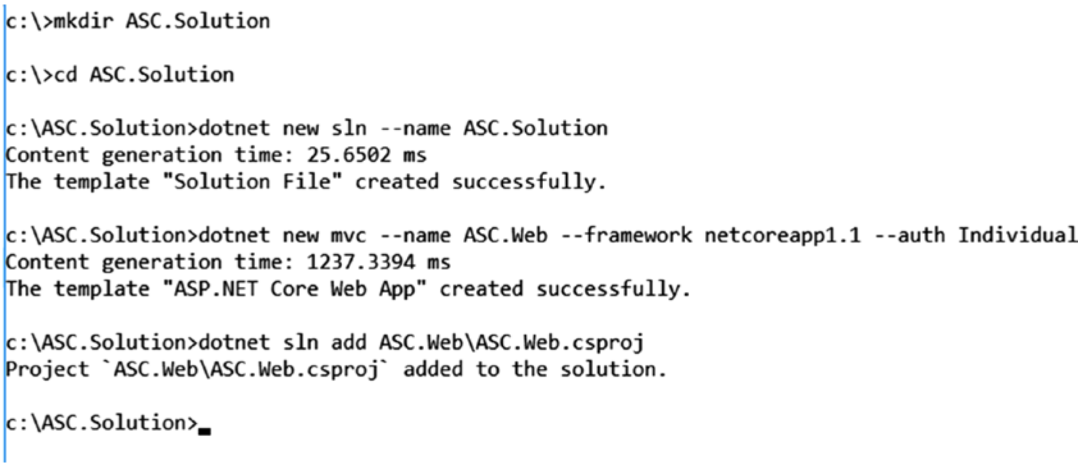
Figure 2-13. *Index page of ASP.NET Core application running from VS*

■ **Note** A web application project can also be created using the .NET Core CLI.

To create a project by using the .NET Core CLI, we need to execute the following commands in order (make sure to run CMD as Administrator):

```
mkdir ASC.Solution
cd ASC.Solution
dotnet new sln --name ASC.Solution
dotnet new mvc --name ASC.Web --framework netcoreapp1.1 --auth Individual
dotnet sln add ASC.Web\ASC.Web.csproj
```

Here we create a directory and set it as the working directory for the command-line prompt. Using the .NET Core CLI, we then create a blank solution and an ASP.NET Core Web project with the MVC template, ASP.NET Core 1.1 framework, and authentication mode set to Individual. Finally, add the web project to the solution file, as shown in Figure 2-14.



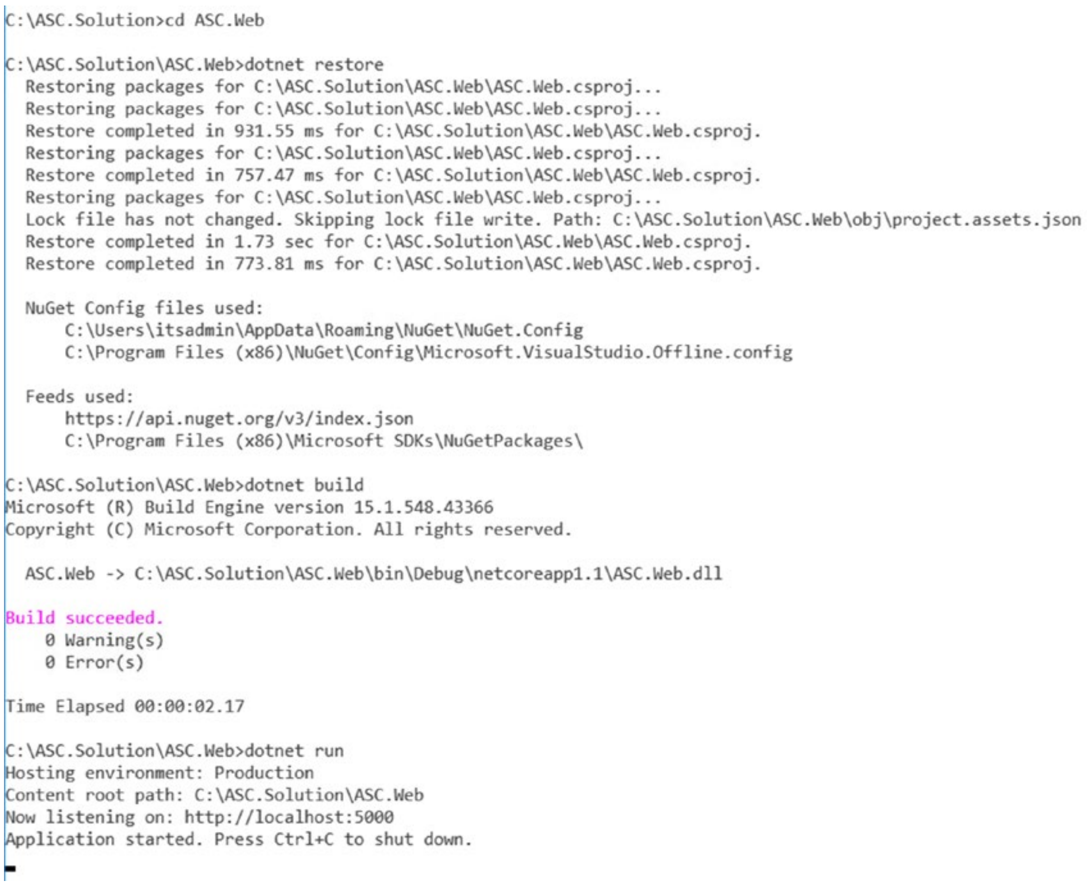
```
c:\>mkdir ASC.Solution

c:\>cd ASC.Solution

c:\ASC.Solution>dotnet new sln --name ASC.Solution
Content generation time: 25.6502 ms
The template "Solution File" created successfully.

c:\ASC.Solution>dotnet new mvc --name ASC.Web --framework netcoreapp1.1 --auth Individual
Content generation time: 1237.3394 ms
The template "ASP.NET Core Web App" created successfully.

c:\ASC.Solution>dotnet sln add ASC.Web\ASC.Web.csproj
Project `ASC.Web\ASC.Web.csproj` added to the solution.

c:\ASC.Solution>
```

Figure 2-14. *Creating the ASP.NET Core project using the .NET Core CLI*

To run the application, we need to execute the following commands:

```
cd ASC.Web
dotnet restore
dotnet build
dotnet run
```

First we need to restore all the dependencies from `nuget.org`; this is a one-time activity that is required to resolve all dependencies. Then we build the application to make sure there are no errors. Finally, we run the application as shown in Figure 2-15.



```
C:\ASC.Solution>cd ASC.Web

C:\ASC.Solution\ASC.Web>dotnet restore
  Restoring packages for C:\ASC.Solution\ASC.Web\ASC.Web.csproj...
  Restoring packages for C:\ASC.Solution\ASC.Web\ASC.Web.csproj...
  Restore completed in 931.55 ms for C:\ASC.Solution\ASC.Web\ASC.Web.csproj.
  Restoring packages for C:\ASC.Solution\ASC.Web\ASC.Web.csproj...
  Restore completed in 757.47 ms for C:\ASC.Solution\ASC.Web\ASC.Web.csproj.
  Restoring packages for C:\ASC.Solution\ASC.Web\ASC.Web.csproj...
  Lock file has not changed. Skipping lock file write. Path: C:\ASC.Solution\ASC.Web\obj\project.assets.json
  Restore completed in 1.73 sec for C:\ASC.Solution\ASC.Web\ASC.Web.csproj.
  Restore completed in 773.81 ms for C:\ASC.Solution\ASC.Web\ASC.Web.csproj.

  NuGet Config files used:
      C:\Users\itsadmin\AppData\Roaming\NuGet\NuGet.Config
      C:\Program Files (x86)\NuGet\Config\Microsoft.VisualStudio.Offline.config

  Feeds used:
      https://api.nuget.org/v3/index.json
      C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\

C:\ASC.Solution\ASC.Web>dotnet build
Microsoft (R) Build Engine version 15.1.548.43366
Copyright (C) Microsoft Corporation. All rights reserved.

  ASC.Web -> C:\ASC.Solution\ASC.Web\bin\Debug\netcoreapp1.1\ASC.Web.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:02.17

C:\ASC.Solution\ASC.Web>dotnet run
Hosting environment: Production
Content root path: C:\ASC.Solution\ASC.Web
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

Figure 2-15. *Restore, build, and run an ASP.NET Core application*

Navigate to `http://localhost:5000` and you should see our application up and running.

■ **Note** By default, the .NET Core CLI always runs the application at port 5000. To change the default port, follow the tutorial at `www.intstrings.com/ramivemula/articles/jumpstart-40-change-default-5000-port-of-dotnet-run-command/`.

Understanding the ASP.NET Core Web Project Artifacts

The .NET Core project system is a rewrite compared to previous versions of .NET Framework. In the new .NET Core project system and especially in the ASP.NET Core project, we have a lot of new files (such as `bundleconfig.json` and `Program.cs`) that were added and old files (such as `Global.asax`) that were removed. In this section, we will demystify the project structure of ASP.NET Core web applications so you can understand the importance and relevance of each artifact.

Dependencies

The `Dependencies` node displays all the dependencies of the project. These dependencies are categorized into client-side Bower dependencies, NuGet package dependencies, and the actual .NET Core framework. All the dependencies will be restored initially when the project is created and loaded for the first time. Whenever the `csproj` file changes, dependencies will be restored automatically.

Properties and launchSettings.json

This node holds the profiles that are used to launch the project. Typically, for a web project, we have executable profiles based on IIS Express and the .NET Core CLI. The Properties node can be used to maintain web server settings such as Enable Authentication. Project-specific environment variables can be set by double-clicking the Properties node. Any changes to project settings will be saved to the `launchSettings`.json file.

wwwroot

The `wwwroot` folder contains all static files of the project. These include HTML, image, JavaScript, and CSS files. These files will be served to the user directly. Custom JQuery files will go into the `js` folder, and all JQuery library files will go into the `lib` folder.

Controllers

The `Controllers` folder will hold all the MVC/Web API controllers in the ASP.Net Core project. It is the same as a traditional ASP.Net MVC 4 project.

Data

The `Data folder` will hold all the code files related to the Entity Framework Database Context and migrations.

■ **Note** As Individual Authentication mode is used while creating the project, the Data folder was created with the required ASP.NET Core Identity tables and migrations. During the course of this book, we are going to change the implementation to Azure Table Storage.

Models

All the web application View models are placed in this folder. A View model is typically used by the view for displaying data and submitting it.

Services

By default, all the custom interfaces and business logic will be created in this folder.

■ **Note** We will not use the `Services` folder to hold business logic. Instead, separate class libraries will be created for business and data storage operations.

Views

The `Views` folder holds all the `cshtml` files. It follows the same convention as does the traditional ASP.Net MVC 4 project, by placing views corresponding to a controller under the same folder with the controller name. Also, this folder holds shared views (such as partials) in the `Shared` folder. This folder contains a new file called `_ViewImports.cshtml`, which serves as a common place to add all namespaces that are used by all views, along with tag helpers. This folder contains `_ViewStart.cshtml`, which is used to execute the common view code at the start of each view's rendering.

appsettings.json

This file can be used to store configuration key/value pairs (just like `appsettings` in the `web.config` file of traditional ASP.Net projects) for the project. This file is also used to hold connection strings, log settings, and so forth.

bower.json

Client-side asset files (for example, JavaScript, stylesheets, images, and HTML) are managed through `bower.json`. By default, Bootstrap and JQuery files are loaded. The `.bowerrc` file holds the location (typically, `wwwroot/lib`) where Bower needs to save the downloaded packages.

bundleconfig.json

This file is used for the bundling and minification of JQuery and CSS files. In this file, we specify different input files and a corresponding output file. We need to refer to the `BundlerMinifier.Core` NuGet package and use dotnet bundle to trigger the bundle and minification operation.

Program.cs

`Program.cs` is the starting point of an ASP.Net Core web application. It contains the program entry point `Static Void Main()`, making the ASP.Net Core web application a truly console application. The ASP.Net Core application requires a host to start the application and manage the application's lifetime. `Program.cs` creates a new instance of `WebHostBuilder`, which is used to configure and launch the host. A Kestrel server is used to host the application. Finally, `WebHostBuilder`'s `Build` method is called to create the host, and then the `Run` method is used to start it.

Startup.cs

`Startup.cs` configures the environment for the ASP.Net Core web application. In its constructor, it configures the configuration of the application. In its `ConfigureServices` method, framework services (for example, EF, MVC, and authentication) and custom services (business specific) will be added and resolved using dependency injection. One more method, `Configure`, is used to build the ASP.Net HTTP pipeline with middleware.