ASP.NET Core Application Development

Building an application in four sprints

James Chambers
David Paquette
Simon Timms
ASP.NET Core Application Development: Building an application in four sprints (Developer Reference)

James Chambers
David Paquette
Simon Timms
I would like to dedicate this book to my loving wife. Thank you for your support and I look forward to spending more time outdoors with you once I finally emerge from this office.

—David Paquette

I would like to dedicate this book to my children who have been forced to play without me while I work feverishly pressing buttons in front of a glowing box and my wife who taught them to sing Cat’s In The Cradle outside my office door. Without their support I would be but a quarter of a person. I love them more than I have words to express.

—Simon Timms

I dedicate this book to my incredible, intelligent and striking wife who has supported me for over twenty years while I chase dreams and sometimes our pet dog who gets out in the middle of the night. I dedicate it to my children that I love endlessly and completely. I can’t wait to spend more time with them, even if they need more showers and should ask for rides less.

—James Chambers
This page intentionally left blank
# Contents at a Glance

*Introduction*  

**PART I  ALPINE SKI HOUSE**

- How We Got Here  
- Influencers  
- Models, Views, and Controllers  
- Scoping the Project  
- Builds  
- Deployment  

**PART 2  SPRINT RETRO: A JOURNEY OF 1000 STEPS**

- Building Web Applications with Microsoft Azure  
- Cross-Platform  
- Containers  
- Entity Framework Core  
- Razor Views  
- Configuration and Logging  

**PART 3  SPRINT RETRO: THE BELLY OF THE BEAST**

- Identity, Security, and Rights Management  
- Dependency Injection  
- Role of JavaScript  
- Dependency Management  
- Front End with Style  
- Caching
## PART 4  
**SPRINT RETRO: HOME STRETCH**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>REUSABLE COMPONENTS</td>
<td>295</td>
</tr>
<tr>
<td>TESTING</td>
<td>307</td>
</tr>
<tr>
<td>EXTENSIBILITY</td>
<td>325</td>
</tr>
<tr>
<td>INTERNATIONALIZATION</td>
<td>341</td>
</tr>
<tr>
<td>REFACTORING AND IMPROVING CODE QUALITY</td>
<td>353</td>
</tr>
<tr>
<td>ORGANIZING THE CODE</td>
<td>373</td>
</tr>
<tr>
<td>POSTFIX</td>
<td>383</td>
</tr>
</tbody>
</table>

*Index*  

387
# Table of Contents

*Introduction* .......................................................... xvii

## PART I  ALPINE SKI HOUSE  1

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>How We Got Here</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Active Server Pages</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>ASP.NET</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>ASP.NET MVC</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Web API</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>ASP.NET Core</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Influencers</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Backward compatibility</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Rails</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Node.js</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>Angular and React</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>Open source</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>OWIN</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>26</td>
</tr>
</tbody>
</table>

What do you think of this book? We want to hear from you!
Microsoft is interested in hearing your feedback so we can improve our books and learning resources for you. To participate in a brief survey, please visit:

http://aka.ms/tellpress
<table>
<thead>
<tr>
<th>Chapter 9</th>
<th>Containers</th>
<th>105</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeatable Environments</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>Docker</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>Windows Containers</td>
<td>114</td>
<td></td>
</tr>
<tr>
<td>Docker in Production</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>On the Cloud</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td>118</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 10</th>
<th>Entity Framework Core</th>
<th>119</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Framework Basics</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>Querying for a single record</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>Querying for multiple records</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>Saving Data</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>Change Tracking</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>Using Migrations to Create and Update Databases</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>The ApplicationDbContext</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>Extending ApplicationUserContext</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>Ski Card Context</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>Relationships crossing Context Boundaries</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>Wiring up the Controller</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>Pass Types</td>
<td>141</td>
<td></td>
</tr>
<tr>
<td>Passes and Validation</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>Events and Event Handlers</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td>146</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 11</th>
<th>Razor Views</th>
<th>147</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating Web Sites as a Developer Today</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>Building on Previous Successes and Learnings</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>Understanding Razor’s Role</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>Mastering Razor Essentials</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>Peeking Behind the Curtain</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>Writing Expressions with Razor Syntax</td>
<td>152</td>
<td></td>
</tr>
</tbody>
</table>
Switching to Code .......................................................... 153
Explicitly Using Markup .................................................. 154
Razor Parser Control Cheat Sheet ....................................... 154
Bringing in More C# Features ............................................. 155
Using C# Types in Your Views ............................................ 155
Defining a Model ............................................................ 156
Using View Data ............................................................. 156
Working with Layouts ....................................................... 158
Foundations of Layouts .................................................... 159
Including Sections from Views ........................................... 160
Defining and Consuming Partial Views ................................. 161
Enhancing Views with Advanced Razor Functionality ............ 162
Injecting Services into a View ............................................. 162
Working with Tag Helpers ................................................ 163
Avoiding Duplication in Your Views ..................................... 167
Using Alternate View Engines ............................................ 167
Summary ................................................................. 168

Chapter 12 Configuration and Logging ................................. 169
Moving Away from web.config ........................................... 170
Configuring Your Application ............................................. 170
Using Stock Configuration Providers ................................. 172
Building a Custom Configuration Provider ......................... 173
Employing the Options Pattern ......................................... 176
Logging as a First-Class Citizen ........................................ 177
Creating Logs That Provide Clarity .................................... 177
Setting Expectations for Exceptions ................................. 178
Logging as a Development Strategy ................................. 179
Logging Levels in ASP.NET Core ..................................... 180
Using Logging Scopes to Augment Logging ......................... 183
Using a Structured Logging Framework ............................ 185
Logging as a Service ..................................................... 186
Summary ................................................................. 188
PART 3  SPRINT RETRO: THE BELLY OF THE BEAST  189

Chapter 13  Identity, Security, and Rights Management  193
Defense in Depth ......................................................... 194
User Secrets .............................................................. 195
  Azure-Backed Authentication ................................. 197
Identity in ASP.NET Core MVC ................................. 202
  Local User Accounts ............................................. 206
Other Third-Party Authentication Providers .................. 208
  Enabling Security Through Attributes ....................... 210
Using Policies for Authorization .................................. 212
  Applying Policies Globally ....................................... 212
  Defining Policies for Selected Use ......................... 213
  Custom Authorization Policies ................................. 214
Protecting Resources ............................................... 215
  Cross-Origin Resource Sharing (CORS) ...................... 218
Summary .............................................................. 219

Chapter 14  Dependency Injection  221
What is Dependency Injection? ................................. 222
  Resolving Dependencies Manually ........................... 222
  Using a Service Container to Resolve Dependencies ....... 223
Dependency Injection in ASP.NET Core ....................... 225
  Using The Built-In Container ................................. 225
  Using a third-party container ................................. 228
Summary .............................................................. 230

Chapter 15  Role of JavaScript  231
Writing Good JavaScript ............................................. 232
Do We Even Need it? .................................................. 233
Organization .......................................................... 233
To SPA or not to SPA? .............................................. 234
Building JavaScript ................................................... 235
Chapter 16  Dependency Management  251

  NuGet ................................................................. 252
    Installing packages with NuGet. .................................. 252
  npm ................................................................. 255
    Adding Dependencies .............................................. 255
    Using npm modules .............................................. 256
    Visual Studio Integration ........................................ 257
  Yarn ........................................................................ 258

  Bower ................................................................. 260
    Adding Dependencies .............................................. 261
    Referencing Assets from Bower Packages ..................... 262

  Summary ..................................................................... 262

Chapter 17  Front End with Style  263

  Building Websites with Style Sheets. ................................ 264
    Digging up the Past ................................................. 264
    Creating Your Own Style Sheet .................................. 266

  Getting Sassy with Your Style ....................................... 268
    Basics of SCSS ...................................................... 269
    Creating Mixins ..................................................... 274
    Mixing Mixins and Directives ................................. 274
Establishing a Development Workflow ........................................... 275
Using Command Line Tools ......................................................... 275
Working in Visual Studio Code ...................................................... 276
Modifying the Project Build Tasks ................................................. 277
Using Third Party Frameworks ...................................................... 277
Extending a CSS Framework ......................................................... 277
Customizing a CSS Framework Footprint ...................................... 278
Leveraging CSS Frameworks for Custom Style Sheets .................... 279
Alternatives to CSS Frameworks ................................................... 280
Summary ...................................................................................... 280

Chapter 18  Caching ................................................................. 281
Cache Control Headers ................................................................. 283
Using the Data-Cache ................................................................... 285
In Memory Cache ......................................................................... 285
Distributed Cache ......................................................................... 287
How Much Cache is Too Much? ...................................................... 289
Summary ...................................................................................... 289

PART 4  SPRINT RETRO: HOME STRETCH .................................. 291

Chapter 19  Reusable Components .............................................. 295
Tag Helpers .................................................................................. 296
Anatomy of a Tag Helper ............................................................... 296
Scripts, Links, and Environment Tag Helpers ............................... 297
Cache Tag Helper .......................................................................... 298
Creating Tag Helpers .................................................................... 299
View Components ......................................................................... 302
Invoking View Components .......................................................... 303
Contact Customer Service View Component .................................. 303
Partial Views ................................................................................. 305
Summary ...................................................................................... 306
Chapter 23  Refactoring and Improving Code Quality  353
What is refactoring? .........................................................354
Measuring Quality ..........................................................355
Finding Time to Refactor ...................................................357
Refactoring with a Safety Net ............................................358
Data Driven Changes .......................................................365
A Code Cleanup Example ..................................................366
Tools That Want to Help ....................................................370
Getting to Quality ............................................................370
Summary ............................................................................370

Chapter 24  Organizing the Code  373
Repository Structure .........................................................374
Inside the Source ..............................................................374
Parallel Structure .............................................................376
MediatR ..............................................................................376
A Brief Introduction to the Messaging Pattern .......................377
Implementing a Mediator ...................................................377
Areas .................................................................................381
Summary .............................................................................382
Postfix .................................................................................383
Index .......................................................................................387

What do you think of this book? We want to hear from you!
Microsoft is interested in hearing your feedback so we can improve our books and learning resources for you. To participate in a brief survey, please visit:

http://aka.ms/tellpress
Introduction

ASP.NET Core MVC is Microsoft’s latest web framework for .NET developers. It is the next version of the now-familiar MVC Framework and aspires to cross boundaries by enabling cross-platform development and deployment. It leverages a wide assortment of open source libraries and is, itself built as open source software. ASP.NET Core MVC helps developers to separate concerns like business logic, routing, services, and views and provides new systems for configuration and extensibility. It uses the C# programming language and the Razor view engine. If you are an experienced .NET developer or a newcomer to the .NET platform, ASP.NET Core MVC is likely what your projects will be built from.

This book follows the first few sprints of an application being redeveloped by a team at a fictional company named Alpine Ski House. Each chapter contains a little bit of information about the challenges the team is facing and how they work to overcome them. Despite having a short story element to each chapter, the book dives deep to cover not only the features of ASP.NET Core MVC, but also the tooling around it that developers will use to build, maintain and deploy their applications.

In addition to its story elements and technical information around ASP.NET Core MVC, the book discusses the new version of Entity Framework, package management systems, and peripheral technologies that are used by modern web developers. Beyond the explanatory content, the book also comes with an accompanying project—the very same project that the developers at Alpine Ski House have built.

Who should read this book

The book takes a programmer through all the steps necessary to build a brand new application on ASP.NET Core and push it out so it is available on the Internet. There is still a great population of programmers who have yet to journey onto the web or have done so only using webforms, much less using the full gamut of tooling that is available today. This book will help put the skills and confidence needed in place to build modern applications on an emerging framework. It will help you explore application architecture, deployment and building applications that are designed for the cloud.
Assumptions

Readers should know how to program at an intermediate to senior level. Readers should be proficient in C#, have a background in web development, and understand fundamentals of working in Visual Studio. Experience with previous versions of MVC will be beneficial, but not required. Familiarity in working with a command line interface will be an asset. After completing this book you will be able to build a meaningful and relevant database-driven application and deploy it to a cloud-based infrastructure.

This book might not be for you if...

This book might not be for you if you are an advanced ASP.NET MVC developer who has been closely following or contributing to the development of ASP.NET Core MVC.

Organization of this book

This book offers the innovative approach of taking developers through individual sprints in the development of an application. It will cover not just the technology, but also the process of recovering from mistakes and adjusting to user feedback, starting from a blank canvas and ending with a real working product.

This book is divided into four sections:

- Part 1, “Alpine Ski House,” Covers background information that sets up the example app and fictional characters in the story used throughout the book
- Part 2, “Sprint Retro: A Journey of 1000 Steps,” focuses on the features required to get our application out the door, configuring the pipeline so that deployment happens on-the-fly in a way that the whole team understands.
- Part 3, “Sprint Retro: The Belly of the Beast,” focuses on the core features needed to get the business running with our sample application. Here we introduce data access using the Entity Framework Core, creating views using Razor, Configuration and Logging, Security and User Management, and finally Dependency Injection.
- Part 4, “Sprint Retro 3: Home Stretch” covers JavaScript and dependency management, along with building on top of the previous foundations.

Postfix covers some important topics such as testing, refactoring and extensibility.
Finding your best starting point in this book

The different sections of ASP.NET Core Application Development: Building an application in four sprints cover a wide range of technologies associated with the ASP.NET Core framework. Depending on your needs, and your existing understanding of Microsoft’s web stack, you may wish to focus on specific areas of the book. Use the following table to determine how best to proceed through the book.

<table>
<thead>
<tr>
<th>If you are</th>
<th>Follow these steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>New to ASP.NET Core development, or an existing ASP.NET Core developer</td>
<td>Focus on Parts I, II and III, or read through the entire book in order.</td>
</tr>
<tr>
<td>Familiar with earlier releases of ASP.NET</td>
<td>Briefly skim Chapter 1 and Chapter 2 if you need a refresh on the core concepts. Read up on the new technologies throughout the remainder of the book.</td>
</tr>
<tr>
<td>Interested in client side development</td>
<td>Read Chapters 15, 16 and 17 in Part IV. Skim the section on JavaScript services in Chapter 20.</td>
</tr>
<tr>
<td>Interested in cross-platform development</td>
<td>The entire book is applicable to cross platform development but Chapter 8 and 9 are specifically dedicated to the topic.</td>
</tr>
</tbody>
</table>

Most of the book’s chapters include hands-on samples that let you try out the concepts just learned. No matter which sections you choose to focus on, be sure to download and install the sample applications on your system.

Conventions and features in this book

This book presents information using conventions designed to make the information readable and easy to follow.

- The book includes samples for C# programmers and syntaxes such as HTML, CSS, SCSS and Razor.

- Boxed elements with labels such as “Note” provide additional information or alternative methods for completing a step successfully.

- A plus sign (+) between two key names means that you must press those keys at the same time. For example, “Press Alt+Tab” means that you hold down the Alt key while you press the Tab key.

- A vertical bar between two or more menu items (e.g. File | Close), means that you should select the first menu or menu item, then the next, and so on.
System requirements

You will need the following hardware and software to run the sample application accompanying this book:

- .NET Core 1.0 or newer, available cross platform from https://dot.net.
- Your code editor of choice. We use Visual Studio 2015 (any edition) or newer on Windows and Visual Studio Code on Windows, Mac and Ubuntu Linux.
- SQL Server LocalDB (included with Visual Studio 2015 or newer on Windows). Linux or Mac users will need access to a SQL Server database hosted either on a Windows machine or in Microsoft Azure.
- Computer that has a 1.6GHz or faster processor
- At least 1 GB of RAM
- 4 GB of available hard disk space
- Internet connection to download software and sample project

Depending on your Windows configuration, you might require Local Administrator rights to install or configure Visual Studio 2015.

Downloads: Sample Project

Most of the chapters in this book include snippets from the sample project. The sample project is available on GitHub:

https://github.com/AspNetMonsters/AlpineSkiHouse

Follow the instructions on the GitHub repository to download and run the sample project.

Note  In addition to the sample project, your system should have .NET Core 1.0 or newer installed.
Errata, updates, & book support

We've made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

https://aka.ms/ASPCoreAppDev/errata

If you discover an error that is not already listed, please submit it to us at the same page.

Get all code samples, including complete apps, at: https://aka.ms/ASPCoreAppDev/downloads.

If you need additional support, email Microsoft Press Book Support at mspinput@microsoft.com.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to http://support.microsoft.com.

Free ebooks from Microsoft Press

From technical overviews to in-depth information on special topics, the free ebooks from Microsoft Press cover a wide range of topics. These ebooks are available in PDF, EPUB, and Mobi for Kindle formats, ready for you to download at:

http://aka.ms/mspressfree

Check back often to see what is new!

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://aka.ms/tellpress

We know you’re busy, so we’ve kept it short with just a few questions. Your answers go directly to the editors at Microsoft Press. (No personal information will be requested.) Thanks in advance for your input!
Stay in touch

Let’s keep the conversation going! We’re on Twitter: http://twitter.com/MicrosoftPress
PART I

Alpine Ski House

CHAPTER 1  How We Got Here. ................................................. 5
CHAPTER 2  Influencers. ..................................................... 17
CHAPTER 3  Models, Views, and Controllers ......................... 27
CHAPTER 4  Scoping the Project ............................................ 37
CHAPTER 5  Builds ............................................................ 45
CHAPTER 6  Deployment ..................................................... 57

Here is some background information that introduces the fictional aspect covered in this book, including the fictional characters that are creating the Alpine Ski House application.

Even the most fervent riders had to admit it: the season was at an end. It hadn’t been the best season in memory but nor had it been the worst. It had been, in every way, unremarkable. There had been moments, a power outage in late February had forced the dusting off of an emergency plan which had long been practiced but never used. There had been reports on the local news station about children trapped on gondolas for hours but with the balmy weather nobody had ever truly been in danger. A smattering of free passes was all that was required to keep the skiers and riders coming.

The spring was a time for the permanent staff to regroup and the seasonal staff to return to wherever it is that lefties go in the summer. A rumor among the permanent staff was that at least half of the seasonal staff were rounded up by immigration as soon as they stepped off the hill and sent back to Australia.
elle couldn’t imagine why the young seasonal staff would resist being sent back to Australia. One thing was for sure, it was much more exciting in Australia than in the sleepy mountain town that reemerged from hibernation each winter.

It was still too early to plan the next year and Danielle was looking forward to a month or two of down time before the cycle began anew. She had been the lone developer for Alpine Ski House for close to a decade and every year was about the same. Most of her time involved keeping legacy systems running and adjusting whatever small things were needed for the next year’s activities. It wasn’t the most exciting job in the world but over the winter months it was expected that everybody would sneak off for a couple of hours skiing on nice days and that was a perk she thoroughly enjoyed.

Opening the door to the low-rise that Alpine Ski House called home she was surprised to see that things were buzzing. People she wouldn’t have expected to see in the office for a couple of hours were scattered about in huddles throughout the open plan office. Confused, Danielle dumped her bag and grabbed a coffee before looking for a huddle to join. The rest of the IT staff seemed to be orbiting Tim, the portly IT manager and her boss. Danielle headed over to join.

“Danielle! What do you think of the news, going to be an exciting time if you ask me,” boomed Tim.

“What news is this?” asked Danielle.

“Where have you been?” asked Arjun, “We just bought out Thunder Valley and Mount Ballyhoo. They’re merging operations and we’re all going to lose our jobs!"

The two other ski areas were a few miles down the road. Thunder Valley was a small operation with only three lifts but a loyal following of ski bums. It was a favorite for the locals who wanted a break from the crowds of tourists in the winter months. It couldn’t be more different from Mount Ballyhoo if it had been the output of Babbage’s difference machine. Mount Ballyhoo was a huge ski hill spanning three mountains with countless lifts and enough on hill accommodation to house everybody in town twice over. Every weekend they had bands playing on the hill,
and it was not uncommon to see famous people like Scott Gu and John Skeet there rubbing shoulders with the commoners.

“Now Arjun,” said Tim, “nobody has said anything about layoffs or redundancies or anything of the sort. Why at times like this the workload for IT usually increases because management wants systems integrated right away. We’re just going to have to wait and find out what the plans are.”

Danielle had to sit down. She was years still from retirement and didn’t want to find another job. How many jobs would there be for programmers in a seasonal town like this? “This is silly,” she told herself, “there is no use planning a move back to the big city based on this sort of uncertainty. Things will shake out in the next couple of weeks.”

As it turned out nobody was waiting a couple of weeks.

As soon as lunch, a dandelion and walnut salad, with balsamic sweet potato crisps, was over, Tim came knocking at her cubicle.

“We’re gathering in the big conference room. It sounds like the programmers from Thunder and Ballyhoo are here.”

Gulping down the rest of her goat’s milk, Danielle grabbed a pen and a big yellow legal pad and hurried towards the conference room. The pad and paper were just for show; she hadn’t taken notes at a meeting in years. It was easier to just engage people around the small office in person than plan things out way in advance on a notepad. Better to make a good impression right away with layoffs on the horizon.

The big conference room was rarely used outside of potlucks because there simply weren’t enough people to fill it. But today it was, if not packed, at least well used. Five young hipster looking individuals sat at one end of the table sipping on all manner of exotic looking smoothies. Danielle wondered how one would even go about importing rambutan and what sort of carbon footprint it would have. Still it was better than that time she had hacked her way into a durian only to have to defenestrate the offensive fruit.

Clearly divided from the hipsters were a group guys who would have been called “suits” in the big city. Out here they
looked like they had just stepped off a golf course. Somehow they were already tanned and relaxed looking.

Tim waited for everybody to settle and then addressed the room, “Good news, everybody, this is the team we’re moving forward with. If you’ve made it to this room, then your job is safe and you can relax. I’m sure you all have questions about that and you can come see me individually after this meeting if you want to talk.

“Management has asked me to keep a larger number of programmers on staff after the merge because they have some exciting new initiatives that they want us to embark upon. Over the next few years we’re going to be refreshing all the custom systems we have in place to run the hill. They recognize that this is a big undertaking and some of them have been reading CIO magazine and they’ve learned about agile and microservices. I can assure you that I’ve given instructions that all future copies of that magazine are to be burned before they reach management’s hands but for now we’re stuck with it.”

Tim had always had a bit of a rocky relationship with management’s great new ideas. He was the emergency brake on their crazy ideas. He continued on. “The first thing management want is a way for people to buy their lift tickets online. I’m told that it is 2016 and that we should have that in place already and that every other hill in the world has it.” Tim seemed exasperated by management’s generalizations; it must have been a fun discussion when these orders came down.

“Management wants to see a prototype in a month’s time. I think I can buy another week if we can show that we’re making progress.”

A month’s time! Danielle was beside herself. A month was how long Danielle liked to spend getting her head around a problem. She looked over at the hipster developers hoping they shared her ashen face. But the wheatgrass crew were nodding along happily.

Tim looked like he was coming to a conclusion and readying to step down from his soapbox. “Look guys, we need this to buy us some capital with management. I’ll clear any roadblocks in your way. Use whatever technology you think is best buy whatever tools you need. I trust you to get this done.”
It was a surprise when Adrian popped over to Danielle's cubicle. Maybe more surprising was the furrowed brow he was sporting. “Got a minute?” he asked in a hushed tone, and then walked away without waiting for an answer. She nodded a confused yes, mostly to herself, and slowly got up to follow him to a side room. She scanned around the development pit, but didn’t see anyone watching, much less interested, and couldn’t really get a read on why he was assuming the role of secret agent in this software docu-drama.

He closed the door behind her as she stepped into the room. “Look, I’m a CSS guy. I’m not going to sugar coat it. I know my way around jQuery well enough to troubleshoot, but I’m no programmer.” He looked tense and Danielle wondered what he’d been thinking. “Everyone else seems to be buying into this shift over to Core or whatever it’s called... but I’m... I mean, come on, I run Linux on my laptop and I don’t even use Office.” The room quickly drew silent.

“Are you still nervous, Adrian? Marc had mentioned that you were worried about the cuts.” Danielle was nervous too, truth be told. She had lost some good friends herself when the merge went through, but she wasn’t sure that was what he needed to hear at the moment.

“Well yeah, I guess,” he replied. “But my knowledge of MVC is that it stands for Model-View-Controller, and I haven’t taken a look any deeper than that. And you guys keep referring to it as a framework. If it’s a framework, but you have to make your own models, views, and controllers, then the MVC part seems more than a little misleading, don’t you think?”

He had a point. “Well, yeah, that’s actually pretty true,” said Danielle.

“I’m coming from a different view here; I just want to wrap my head around it. I want to learn, but I’m not sure I know where to start, either.” Adrian pounded back the rest of his coffee like it was about to expire. “I know they said we’re safe if we’re still here, but I don’t want to get caught with my feet standing still if they think there’s still room to shrink the team.”

“Okay, look,” said Danielle, “I’ve got a bit of time right now, let’s run through the basics and we’ll learn together. We’re all going to be fine.”
The M, the V, and the C of it

Let’s face it, the MVC Framework is a pretty boring name. The acronym used in the title is from the well-known Model-View-Controller pattern, and it helps to organize a project. If you’re familiar with it, the name literally spells out some of the original intent to separate concerns, and moves away from the other common pattern at the time known as Page-Controller. The name can also be misleading. The framework is much more than just models, views and controllers. ASP.NET Core MVC has a growing set of tooling and libraries available to that help developers create great applications, worthy of the modern web world.

Let’s do a quick recap on the aspects you should already understand, and then move into some more interesting aspects of the framework, officially known as ASP.NET Core MVC.

Diving into Models

First up is the letter M, so we’ll start with Models. The model represents the data that we need to properly render an experience, or part of an experience, for a customer. Customers are navigating to a page in the application that is data-driven, and models are the data part. However, as far as intent goes, the model in question is actually what you’ll be using to support the rendering of the view, and not the entity or entities in question in which you persist to the database.

Let’s consider this example from Alpine Ski House’s possible database design that deals with user account summaries, as shown in Figure 3-1. When you want to indicate to the user that she has a current season pass, you don’t want to return the list of season passes to the view and iterate over the collection to see if one exists that has not yet expired.
Returning all of this information to the view would be more than is required. Listing 3-1 contains a view model that might more closely approximate the information you would want to display to the user. As you can see, this is a POCO that sports the properties you can use to satisfy the view requirements without the view needing to make any decisions about what to display or any implementation of business logic. The view doesn’t need to know what qualifies as a current season pass nor does it need to sift through any of the purchase details or iterate through child records in related tables to make sense of the data.

LISTING 3-1 The AccountSummaryViewModel Class

```csharp
public class AccountSummaryViewModel
{
    public Guid UserId { get; set; }
    public int YearsOfMembership { get; set; }
    public bool IsCurrentSeasonPassHolder { get; set; }
    public List<string> IncludedFamilyMembers { get; set; }
}
```

The differentiation between what you craft for models on the front end, versus what you store in the database, is important not just for separating the concerns of the view and the business logic that supports it, but also for helping to prevent certain types of security issues. On the “write” side of things, when a view uses a database entity, the application becomes more likely to fall victim to overbinding bugs or attacks. Overbinding occurs when fields that weren’t anticipated from an incoming request are present in form or querystring parameters. The model binder sees the properties, doesn’t know that you hadn’t intended for them to be there, and kindly fills in your data for you. As an example, consider the class representing some kind of a digital asset in Listing 3-2.

LISTING 3-2 The AccountSummaryViewModel Class

```csharp
public class DigitalAsset
{
    public Guid AssetId { get; set; }
    public Guid AssetOwnerId { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public Uri AccessUri { get; set; }
}
```

This type of model can be used to display a list of resources made available to a user, and doing so is quite harmless. But, if you use the same object to receive edits for the record, a malicious user can exploit the fact that AssetOwnerId is a property and use that to take ownership of the asset. In fact, this is how Egor Homakov gained administrative privileges for the Ruby on Rails (RoR) repository on GitHub in 2012.\(^1\)

---

\(^1\) GitHub reinstates Russian who hacked site to expose flaw, John Leyden, March 5, 2012, http://www.theregister.co.uk.
mass assignment feature, an analog to automatic model binding in ASP.NET Core MVC. Thankfully, Homakov’s intentions were pure and no harm was done. We have learned from those binding conventions and habits of yore though. Today, we have many ways to protect ourselves, which we’ll cover later in Chapter 13, “Identity, Security and Rights Management,” but likely the easiest way is to make sure we’re using models that are appropriate to the task at hand.

Most of the examples you find for view models will likely use an entity directly as the model type for the view; however, the approach does not facilitate other aspects of software development, such as testing, nor does it help with separating concerns in your controllers. Using the entity directly in a view means that you’ve achieved an undesirable level of coupling from the database all the way up to the view.

A model should be everything you need to render your page after you’ve taken care of business logic and often has a flattened view of a denormalized record from several tables in the database. For these reasons, and considering the intent of the object you’re building up when you create a “model,” you should likely think of it as the “view model” due to its close relationship and responsibility to the view.

Views

Here, the view in question happens to start with V and is indeed the view we’re talking about in our new favorite acronym. Views in ASP.NET Core MVC are the files used to interleave parts of the model with the HTML needed in order to present the user with the intended user interface.

If you create a new project from the default application template you will find all the views in the Views folder, or you can search Solution Explorer with the term “.cshtml,” which is the extension used for Razor views.

Using the Razor view engine and syntax you’ve seen through the last few iterations of the MVC Framework, you can switch effortlessly between the syntax used to control flow or access our model or services, and the markup required to generate HTML.

In Listing 3-3 we have created an unordered list with values from the model’s IncludedFamilyMembers collection. Razor lets you use C# inline with the HTML and is pretty smart about how it interprets what you throw at it. A simple @ character is enough for the parser to know you’re flipping into C#, and since angle brackets can’t be used at the start of a valid C# statement, it can tell when you’ve switched back to HTML. We’ll be covering Razor in greater detail in Chapter 11, “Razor Views.”

**LISTING 3-3** An example of mixing C# and HTML in Razor Syntax.

```html
<ul>
    @foreach (var familyMember in Model.IncludedFamilyMembers)
    {
        <li>@familyMember</li>
    }
</ul>
```
Partial Views

Toolbars, authentication cues, shopping carts, parts of dashboards, and other similar components of your application often find themselves appearing on multiple pages, or even on all pages. In the name of Don’t Repeat Yourself (DRY), you can create these components using a partial view, which can in turn be used repeatedly from any other page. You’ll also see partial views referred to more simply as “partials.” We’ll use those terms interchangeably throughout the book.

Partials are not typically rendered on their own, but are used in composition of other views in your project. The first place you see this in any MVC application is likely to be in the _Layout.cshtml, where the view relies on partials to render the login status. Other common uses include using a partial view to render items in toolbars, shopping cart summaries like those you see at the top of an ecommerce site, or side bars with relevant data for the current page.

Child actions had to be rendered synchronously in previous versions of the MVC Framework, but the same ideas that made partials possible can now be used to construct view components and invoked asynchronously. We’ll talk about View Components more in Chapter 18, “Reusable Components,” which is important in certain scenarios to keep performance in check on the site. Complex generated views and partials that interact with services are examples of this, which we’ll talk about later in this chapter.

Before users can get the output of a view, and in order for you to load any kind of model into the view engine, we must talk a little bit about Controllers in your project.

Controllers (...and Actions!)

Controllers are the traffic cops of MVC applications, ensuring the right types of bits travel to and from the correct places. Controllers typically inherit from the base Controller class, but if you don’t need the functionality of the base class, you can also use the convention of ending your class name with “Controller,” such as in SeasonPassController.

The default convention assumes that you are putting your controllers in a folder called “Controllers” in the root of the project. This is no longer required because Core MVC actually does an assembly scan using the naming and inheritance conventions, but it’s still a recommended practice to organize your controllers in a recognized way. This helps other developers, including the future version of yourself, to easily manage and maintain the code base down the road.

As software developers, we use controllers as a container for related sets of handlers for incoming requests. These handlers are called actions and are implemented as methods in our controller class. Each method, or action, can accept zero or more parameters that are automatically filled in by the model binding step in the execution pipeline if they are presented by the incoming request.

As the authors of these “traffic cops,” our goal is to code our controllers using some well-accepted practices. The primary responsibility of an action is to process a request, validating the incoming parameters and creating an appropriate response.

From time to time, this also requires creating or requesting an instance of a model class, or producing an appropriate HTTP status code based response. You should try to avoid having any business logic
in your controller, which is the responsibility of your model or other components, as well as keeping data access or external calls out of your actions, which should be part of your application services. This is represented in a high level in Figure 3-2.

![Diagram of Controller, Model, and View](image)

**FIGURE 3-2** An illustration showing how controllers are responsible for invoking business logic that helps to generate an appropriate HTTP response

Keeping these services external might seem to make things more complex, or raise questions like, “Who will create these services for me?” This is a great question and one that we’ll answer in the “Dependency Injection” section later in this chapter.

**It’s Not Just About MVC**

As discussed earlier, there’s actually a lot more going on than just the models, views and controllers themselves in your solution. We’ll continue to explore these throughout the book, but here are some important ideas to have in the peripheral as you develop.

**Middleware**

Here is the secret about middleware in ASP.NET Core MVC: it’s pretty much all middleware. All of it! During application start-up you have the opportunity to load your configuration, configure your services, and then configure the request pipeline, which is where the concept of middleware is called into play. You can see this in the `Configure` method of the `Startup` class in the default project template.

Often, the description of middleware and the interpretation by the reader overcomplicates a fairly simple premise. The purpose of middleware is to allow an application to say, “Have each request processed by the following components in the order that I specify.” Middleware is a simplification over
previous incarnations of similar functionality in ASP.NET, namely HTTP Modules and Handlers. Middleware replaces both with a common approach in a fashion that is easier to implement.

There are several pieces of middleware that ship publically to handle most scenarios that you need to cover during the execution of your application, both in lower level environments such as staging and QA, as well as in production:

- **Diagnostics**: Provides exception handling and runtime helpers like database error pages and technical details to developers.
- **Static files**: Allows a short-circuit of the request pipeline to return a file from disk.
- **Identity and Authentication**: Allows applications to protect end points and assets of an application.
- **Routing**: Resolves which controller and action should be executed based on the incoming path and parameters.
- **CORS**: Handles injecting the correct headers for cross-origin resource sharing.
- **MVC itself**: Usually at the end of the configured middleware pipeline as it consumes requests.

Each middleware component has the option to execute code before and after the next component in the chain, or to short-circuit the execution and return a response. The name middleware likely comes from the idea that you can execute a piece of software in the middle of something else, as shown in Figure 3.3. In this instance, you see a series of different requests that are handled by the various middleware components in the default project template. In some cases, the request is handled by the static files middleware, returning an asset in wwwroot. At other times, the request is allowed to pass all the way through to the MVC execution pipeline where your controller is created and you can return a view.

![Figure 3-3](image-url)  
**Figure 3-3** An illustration showing examples of different request types as handled by middleware
You can bring in other middleware from third parties, additional helpers that are provided by Microsoft, or you can write your own to handle cross-cutting concerns that are needed throughout your application. The middleware pipeline can also be branched based on paths or predicates to allow dynamic, flexible rules around processing requests.

**Dependency Injection**

There are many written volumes covering dependency injection, but we'll recap the basics here for completeness.

Generally speaking, it's likely going to be a good idea for your code to be obvious about the dependencies that it takes on. In C# you tend to do this by putting the components and services you need in your constructor, such that any creator of the class has to provide your dependencies for you.

Let's consider the constructor of the HomeController class in Listing 3-3. The class requires that any time it is being created an instance of an ILogger implementation would be provided for it.

**LISTING 3-3 The HomeController Class Constructor**

```csharp
public class HomeController{
    ILogger _logger
    public HomeController (ILogger logger)
    {
        _logger = logger;
    }
}
```

HomeController doesn't need to know how to configure or create an ILogger; it doesn't need to know where to log to, or how it is to be done. But from any point after instantiation, HomeController is now able to add valuable information to your log files as required. This one simple parameter on the constructor explicitly defines your requirements and is referred to as the Explicit Dependencies Principle.

For this controller to be created by the pipeline, you need to have something in the runtime aware of how to resolve the ILogger requirement. You configure these services and components in a container, and then these types of dependencies are injected for you into the constructors at runtime. And voila, dependency injection! Being a broader topic, and also by virtue of ASP.NET Core MVC introducing some new ideas for Dependency Injection (DI), we're going to take a deeper look at the idea of
inverting control in Chapter 14, “Dependency Injection,” where we’ll also explore what is required when replacing the default container.

**Other Gems**

ASP.NET Core MVC contains some great improvements over previous versions and we are going to explore them throughout the book.

- **Configuration and Logging:** Long considered afterthoughts in the .NET space, these critical application aspects have been revamped, simplified, and made into first-class citizens. Read more about these in Chapter 12, “Configuration and Logging.”

- **Tag Helpers:** Along with a few other aspects of simplifying front end development, in Chapter 18, “Reusable Components,” we’ll examine Tag Helpers and how they more closely resemble the HTML we’re trying to return to the client.

- **Identity:** A user is more than just a name and a password. In Chapter 13, “Identity, Security & Rights,” we’ll explore the new features, uses, and components around security and rights management in ASP.NET Core MVC.

**Summary**

Each iteration of the MVC Framework has helped shape part of what it’s become today. Some lessons learned along the way have helped to bring better assets to developers and the models, views, and controllers we have at the heart of our applications today take up only a small part of our development efforts.

Before they knew it, the morning had escaped them and Danielle had all but plastered the whiteboard with dry erase marker. Adrian flopped back in his chair and said, “Geez, Danielle. Someone should write a book on this.”
This page intentionally left blank
Index

Symbols
37 Signals 18
201 status code 14
.aspx extension 8
@ character 152
@inject directive 346
HBoxLayout.cshtml 31
.NET Base Class Library 22
.NET Core 23–24
@type/chartjs package 258
@types/lodash 258

A
Access-Control-Allow-Origin headers 218
access keys 85
access rights 195
access tokens 194, 205
AccountController class 211
account lockouts 208
action methods 136–140
ActionResult 20
ActionResults 11–12
Active Directory (AD) 197
ActiveRecord 21
Active Scripting 6–7
Active Server Pages 18
Active Server Pages (ASP) 6–7
adapter pattern 43
AddConsole extension method 182
AddDbContext method 226
Add* extension method 225
AddMvc() 212
AddPolicy() 218
addTagHelper directive 300
admin view 42
AJAX 8
Alpine Ski House
build 51–56
AMD modules 241
Angular 23–24
anti-corruption layer 43
Any method 138
Apache 2.0 license 15
Apache Mesos 116
API keys
developer-level 194
ApplicationDbContext class 130–133
Application Lifecycle Management group 65
ApplicationModel 327
applications
configuration of 169–177
deployment 57–74, 106
hosting 68
in containers 108–118
internationalization of 341–352
isolating 106–107
isomorphic 338–339
MVC 10–13
packaging 67–68
portable mode 66
publishing 64–66
running inside of Linux container 110–114
self-contained mode 67
single page 10, 338
web. See web applications
ApplicationUser class 130–131, 132
ApplicationUserContext class 132–133
Apply method 327
App Services 70–73
appSettings.Development.json file 100
appsettings.json file 201
app.UseIdentity() 206
AppVeyor 65
areas 381–382
asm.js 235
.asmx files 13
AsNoTracking() method 124
ASP. See Active Server Pages
ASP.NET 7, 7–9, 18
  Web Forms 8–10
ASP.NET Core 15–16
  adding services in 225–227
  backward compatibility 18
  built-in container 225–227
  configuration in 170–177
  dependency injection in 225–229
  development of 15
  extensibility of 326–340
  logging in 177–188
  middleware 329–334
  routing in 335–337
  using third-party containers with 228–229
ASP.NETCORE_ENVIRONMENT variable 99
ASP.NET Core Identity
  202–205
    app.UseIdentity() 206
    claims in 205
    custom authorization policies 214–215
    local user accounts 206–208
    password hashing and 205
    referencing 205–206
AspNetCoreModule 61
ASP.NET Core MVC
  conventions 326–328
  MVC framework 28–36
ASP.NET MVC 10–13, 18, 20
  conventions 20–21
  directory structure 20, 373–382
  routing 21
ASP.NET Web Forms 18
async/await 137
Atlassian 51
attribute routing 336–337
attributes
  enabling security through 210–211
Atwood, Jeff 22
authentication 33, 195
  ASP.NET Core Identity 202–205
  Azure-backed 197–205
  middleware 201
  NTLM 61
  of users 207–208
  sign up policies 200–201
  third-party authentication providers 208–211
  through attributes 210–211
  two-factor 208
  authorization
    claims and 205
    policies for 212–215
    resource protection using 215–217
  AuthorizationHandler 216–217
  AuthorizationPolicyBuilder 213–214
  Authorize attribute 211
  autocomplete 246
  Autofac 228–229
  automated testing 358
Azure
  building web applications with 79–92
  cloud offerings in 81
  deployments 68–73
  resource groups 72
  storage queues 87–88
  WebJobs 84, 88–89
Azure Active Directory
  options for authentication in 197
Azure Active Directory B2C 198–202
  adding social identity to 199–200
  adding user-specific attributes 199–200
  creating B2C directory 198–199
  registering application in 200–201
Azure App Services 70–74
Azure-backed authentication 197–202
  configuration options 197
  external users and 197–201
Azure Portal 82–83
Azure Resource Explorer 82
Azure Resource Manager (ARM) templates 72–74
Azure Storage Tools 170

B

B2C directory
  creating 198–199
  backward compatibility 10, 18
Bamboo 51
Base Class Library 22
  base class library (BCL) 102
  Base Class Library (BCL) 7
BCL. See Base Class Library
  binary large objects (blobs) 83–84, 86–87
bin deployment 67
bin folder 234
BlobAttribute 89
blobs (binary large objects) 83–84, 86–87
block level scoping 245
Bootstrap 298
Bootstrap CSS framework 264, 277–280
bounded contexts 42, 43
Bower 260–262
  adding dependencies 261–262
  referencing assets from 262
bower install command 261
bower.json file 261
build environments 106
build pipeline 48–51
builds 45–56
  Alpine Ski House 51–56
  build definition page 53
  command line 46–47
  continuous integration and 48
  JavaScript 235–243
  modifying project build tasks 277
  nightly 48
  pipelines 239
  triggers for 56
build scripts 374
build servers 48, 49, 51
build types 66–67
Bundler & Minifier tool 236–237

C

C# 7, 232, 245
  Razor and 148, 149, 151, 153, 155–157
  types 155
C++ 235
cache busting 297–298
cache tag helper 298
caching 92, 281–290
  amount of 289
  cache control headers 283–285
  cache levels 282
  data-cache 285
  distributed 287–289
  in memory 285–286
  output 283
Can I Use website 232, 236, 241, 245, 249, 255, 257, 282
CaptureStartupErrors(true) directive 59
Cascading Style Sheet (CSS) 264–269.
See also CSS framework
  creating 266–268
  modules 266
  nesting 272–273
  pre-processors 269
  Sassy 268–275
C# code 46
CDNs. See Content Delivery Networks
change tracking 124–125
characterization tests 358–359
child actions 303
claims 205
ClaimsPrincipal 207
classes
  lack of, in JavaScript 232
Client ID 199
client side libraries 260
client side programming 231, 235
cloud
  containers on the 117–118
  cloud computing 69, 80–92
  service offerings 81
code
  cleanup 366–370
  tools for 370
  cyclometric complexity of 356–357
  legacy 358
  organization of 373–382
  quality 367–370
  measuring 355–357
  Razor 153–154
  readability 356
  refactoring 353–365
  source. See source code
testing 49–51, 55
  tightly coupled 223
code behind model 10
code duplication 295
code editors 98
CodePlex 24
CodeRush 367, 370
CoffeeScript 21, 235, 243
COM. See Component Object Model
close line builds 46–47
Command Line Interface (CLI)
  user secret commands 196
command line tools 68, 275
commands 377, 378
Common Gateway Interface (CGI) scripts 6
CommonJS modules 241
compilers 46
compile time errors 12
ComponentModel namespace 12
Component Object Model (COM) 6–7
components
  large number of 251
  partial views 305–306
  tag helpers 296–302
  view 302–305
view 31
computer language 45–46
configuration 35, 169–177
  custom providers 173–175
  hosting options 59
  options services 176–177
  order of loading and 171–172
  pipeline 329–330
  stock providers for 172–174
  web.config 170
Configuration.Azure.KeyVault 172
Configuration.Builder 170
Configuration.CommandLine 172
Configuration.EnvironmentVariables 173
Configuration.Ini 173
Configuration.Json 173
Configuration.Xml 173
ConfigureServices method 133, 176, 212, 213, 215, 225, 227, 228
Connected Service Wizard 84–85
constructor injection 224–225, 227
container deployments 73
containers 105–118
  advantages of 107–110
  base layer 107
  data storage 113
  Docker 110–117
  list of running 112
  Microsoft supplied 111
  networks in 109
  on the cloud 117–118
  orchestration system 109
  pausing 112
  repeatable environments 106–110
  resuming 112
  sharing 114–115
shutting down 112
third-party 228–229
Windows 114–116
Content Delivery Networks (CDNs) 262, 298
Content-Security Policy headers 232
context boundaries
  relationships across 134–135
continuous delivery 49
continuous deployment 48
continuous integration 48
continuous testing 310
Controller class 31
controllers 11, 31–32, 116
  loading external 334–335
  Rails 19
  testing 313–316
  wiring up 135–140
Controllers folder 326, 381
conventions 326
  ASP.NET MVC 20–21
  custom 21, 327–328
  default 326
  routing 336
  Ruby on Rails 19–21
cookies 212
CoreOS Linux distribution 107
CORS. See cross-origin resource sharing
Create action methods 137–139
CreateInitialSchema class 126–127
Create/Read/Update/Delete (CRUD) applications 81
critical logs 180
cross-origin resource sharing (CORS) 33, 218–219
cross-platform support 93–104
cross-site scripting attacks (XSS) 232–233
CSS files 233–234
CSS framework 263–280
  alternatives to 280
  Bootstrap 264, 277–280
  customizing 278–279
  custom style sheets and 279–280
  extending 277
  Foundation 278–279
  third-party 277–280
culture 342, 343
  setting current 348–351
culture-specific resource files 345–346
CurrentUserCulture 343
CurrentUICulture 343
custom conventions 21, 327–328
customer experience 39–41
custom style sheets 279–280
custom tag helpers 299–302
custom tags 23
cyclometric complexity 356–357

domain class

dapper 120
data
  change tracking 124–125
deleting 124
  saving 123
  view 156–157
data annotations 346–347
database providers
  migrations and 129–130
databases
  creating 124–125
  creating and updating
    using migrations 124–128
  migrations 124–130
  relationships crossing context boundaries 134–135
database servers 113
data-cache 285
datacenters 68, 69
data driven changes 365
data retrieval 119
  EF Core
    querying for single record 122
  Entity Framework (EF) Core
    querying for multiple records 123
data storage 119, 282
  containers 113
data types 127–128
DbContext 226
DbContext classes 123, 125, 131, 143
DbSet.Find method 122
DbSet.Remove method 124
debug logs 180, 181
definition files 246–247
dependencies 261
  adding 255–256, 261–262
  errors resolving 227
  manual resolution of 222–223
  restoring all, in package 255
  service containers to resolve 223–224
dependency injection 136, 162, 379
  alternative patterns 224
constructor injection 224–225
  in ASP.NET Core 225–229
  service containers 223–224
  third-party containers 228–229
  vs. manually resolving dependencies 222–223
dependency injection (DI) 34
dependency management 124, 251–262
  package managers for 252–262
    Bower 260–262
    npm 255–258, 259
    NuGet 252–254, 260
    Yarn 258–260
deployment 57–74
  Azure 68–73
  bin 67
  building packages 67–68
  build types 66–67
  container 73
  Kestrel 58–59
  Nginx 62–63
  publishing 64–66
  reverse proxy 59–61
tools 67
  web server for 58
deployment pipeline 49
desktop applications 7
developer-level API keys 194
development environments 106
development workflow 275–276
  --dev flag 259
devops scripts 374
diagnostics 33
directives 274
directory structure 20, 233–234, 373–382
  ASP.NET MVC 20
  Rails 19
Dispose method 122
distributed cache 287–289
Docker 110–114
  in production 116–117
  on Windows 114–116
Docker Registry 110, 114
Docker Swarm 116
documentation 374
document object model (DOM) 235
DOM, See document object model
domain
  defining 43
domain class
    adding properties to 127
Don’t Repeat Yourself (DRY)

Don’t Repeat Yourself (DRY) 31, 295
dotnet build command 96–97
dotnet command line interface (CLI) 94–98
dotnet command line tool 67, 309
dotnet command line tools 337
dotnet ef command 124, 125, 337
dotnet ef database update command 124, 125, 128, 129, 133
dotnet ef migrations add command 126, 128
dotnet ef migrations script command 129
dotnet new command 95
dotnet publish 55
dotnet restore command 96, 252
dotnet run command 97
Down() method 125–126
drag and drop functionality 8
Dropbox 70
duplication
  code 295
dynamically types languages 245

E

ECMAScript 243–245
Edge browser 243
Edit action methods 139–140
editor templates 12
EF. See Entity Framework (EF) Core
elastic computing 69
elastic scale 90–91
Electron 22
encrypted passwords 204
Entity Framework 21
Entity Framework (EF) Core 119–146, 120–146, 147–168
  ApplicationContext 130–133
  basics of 120–122
  change tracking 124–125
  data types 127–128
  events and event handlers 144–146
  mapping conventions in 121
  migrations
  adding 125–127
  database providers and 129–130
to create and update databases 124–128
  passes and validation 142–146
  pass types 141–142
  querying for multiple records 123
  querying for single record 122
  saving data 123
  SkiCardContext class 133–140
  System.Data 120
  environment tag helper 268, 297
  error logs 180
  errors
    dependency resolution 227
    ES2015 to ES5 compiler 243–245
    ES2016 243
  event handler model 7
  event handlers 135, 144–146, 378, 379–380
  events 144–146, 377
  exception handling 178
  exceptions 59, 178, 365
  exit costs
    new technology adoption and 247
  extensibility 325–340
    conventions and 326–328
    dotnet command line tools and 337
    isomorphic applications and 338–339
    loading external controller and views 334–335
    middleware 328–333
    routing 335–337
  extension methods 226–227
  external methods 208–211
  external threats 195

F

F# 245
façade pattern 43
Facebook 199–200, 208, 210
failed login attempts 208
failing fast 50
FAKE 47
fallbacks 298
F# code 46
feature folders 381–382
FileConfigurationSource 174
files property 322
filters 12
First method 122
folder structure
  within areas 381–382
foreign keys
  referential integrity without 134–135
Foundation 277
Freya 25
function level scoping 245
functions
  style sheet 273–274
G

Gartner’s Magic Quadrant Reports 69
GET action 11
GET action method 137
GET requests 41, 60, 283
GitHub 15, 24, 261
  VSTS and 54
global filters 210
globalization, of applications 342–343
glob patterns 297
Gmail 10
Google
  containers 116–118
grunt 255
  unit testing and 323
Grunt 22, 237–238
Gruntfile.js 238
grunt-ts library 238
gulp 239–240, 255
Gulp 22
  unit testing and 323
gulp-karma plugin 323

H

hashing passwords 204–205
headers
  Content-Security Policy 232
heat mapping 365
helper templates 12–13
Homakov, Egor 29
hosted services 69
hosting
  elastic 69
  external 69
  internal 68
hosting configuration options 59
HTML 148, 149, 153, 154
  JavaScript and 232
HTML content
  of tag helpers 301
HTTP protocol 283
HTTPRequest object 18
Hub 114
Hyper-V containers 115
HyperV hypervisor 110, 114

I

IConfigurationBuilder 175
IConfigurationProvider 174
IConfigurationSource 174
IControllerModelConvention 326
IDataReader 120
IDbCommand 120
IDbConnection 120
IDbTransaction 120
identity 33, 35
  ASP.NET Core 202–205
IdentityRoles 205
IdentityRoleUserClaims 205
IdentityUserClaim 205
IdentityUser class 130–131, 203
IdentityUserLogin 205
IdentityUserTokens 205
IgnoreBody 160
IIS. See Internet Information Server (IIS)
image files 234
ImageProcessing library 89
images
  optimization of 240
  storing in blob containers 86–87
Index action method 136–137, 138, 140
Index method 20
information
  protection of sensitive 85
information logs 180
infrastructure as a service (IaaS) 69
inheritance 271–272
INotificationHandler<PurchaseCompleted> interface 145
INotification interface 144
input tag helpers 296
installation
  packages
    with NuGet 252–254
integration tests 50, 324
IntelliSense
  for custom tag helpers 300
IntelliTest 359
internal threats 194–195
internal users 197
internationalization 341–352
  localizable text 343–348
  setting current culture 348–351
Internet Explorer 243
Internet Information Server (IIS) 58, 59, 64
  enabling hosting 61–62
  reverse proxy and 59–61
Internet of Things (IoT) 41
inversion of control (IoC) 223, 228
Invoke method 302, 303
IParameterModelConvention 326
IQueryable<T> 123
IServiceCollection 227, 228, 229
isolation
  performance and 106–107
isomorphic applications 338–339
IStringLocalizer service 344–345
IViewLocalizer 346

J
Jakarta Struts 18
Jasmine 320–321
Jasmine JavaScript unit framework 255
Java 245
JavaScript, 231–250, 221–230
  Angular 23–24
  build tools 235–243
  Bundler & Minifie 236–237
  choosing 243
  Grunt 237–238
  gulp 239–240
  WebPack 240–243
  code writing principles in 232–233
  external script files 233
  files 234
  frameworks for 249–250
  isomorphic applications and 338–339
  mixing with HTML 232
  module formats 241
  module loading 247–248
  need for 233
  organization 233–234
  popularity of 23
  React 23–24
  security vulnerabilities 232–233
  Single Page Application 234–235
  testing 233, 320–323
  TypeScript and 243–247
  variable scoping in 245
Jenkins 65, 71
jQuery 249, 298

JScript 6
JSON results 11
Just-In-Time compiler 66
Just-In-Time compiler (JITer) 102

K
Karma 321–323
Karma test runner 255, 256
Katana 25
Kestrel 25, 58–59
  Nginx and 62–63
  NTLM authentication 61
  reverse proxy and 59–61
key-value pairs 171
Kubernetes 116, 117

L
labels 116
lambda syntax 244
landing pages 282
layouts 13
  foundations of 159–161
  working with 158–161
legacy code 358
legacy support 18
LESS 269
let variable scoping operator 245
libraries 251–252
  base class 102
  client side 260
  definition files and 246–247
  portable class 102
  shared 108
link tag helper 297–299
Linux 93–94
  Alpine Ski House on 98–101
  code editors 98
  SQL Server and 100
  Ubuntu 94–98
Linux containers 110–114
load testing 324
localization middleware 348–349
localization, of applications 341–352
logging 35, 92, 177–188, 365
  as a service 186–188
  as development strategy 179–180
  exception handling 178
levels of 180–183
scopes 183–185
structured framework for 185–187
writing helpful log files 177

N

Ninject

MapRoute 335
MapWhen function 333
Marathon 116
master pages 8–9, 13, 158
MD5 Hashed Passwords 204
mediator class 377–378
mediator pattern 376–380
MediatR 376–380
MediatR library 144
MergeClassAttributeValue extension method 301
Mesosphere 116
micro-ORMs 120
microservices 355
Microsoft Account 199
Microsoft Azure. See Azure
Microsoft Developer Network (MSDN) 51
Microsoft Windows
  backward compatibility 18
middleware 13, 32–33, 328–333
    as class 331
    localization 348–349
    pipelines
      branching 332–333
      configuration 329–330
      writing your own 330–332
Migration class 125
migrations
  adding 125–127
  database providers and 129–130
  data types and 127–128
  to create and update databases 124–128
mixins 274–275
mkdirp 260
mobile devices 10
  browsers on 243
mocking 223
model binding 11
models 12
  defining, in Razor 156–157
  in MVC framework 28–30
Models folder 381
model state validation 138
Model View Controller (MVC) framework 27–36,
  35–36
    controllers 31–32
    models 28–30
    partial views 31–32
    views 30
Model-View-Controller (MVC) framework 10, 11–12
  extensibility of 326–340
Model View Controller (MVC) web frameworks
  Ruby on Rails 18–21
Model-View-Controller pattern 28
modular applications 22
module formats 241
module loaders 247–248
modules 229
MSBuild file 47
MSTest 308
multi-tenant authentication 197
  in Azure Active Directory 197
MvcOptions parameter 212
MyGet 254

N

namespaces 375
  lack of, in JavaScript 232
NancyHost 58
NDepend 370
nightly builds 48
Ninject 228
Node.js 22–24, 25
npm and 255–258
Nodejs 338, 339
node_modules directory
   restoring 259
node services 339
NotFound() method 139, 140
npm 255–258
   adding dependencies 255–256
   modules, using 256
   shortcomings of 259
       Visual Studio integration 257–258
npm init command 255
npm install command 255, 256, 257
NPM package manager 22
npm run-script command 256
NTLM authentication 61
NuGet 22, 85, 252–254, 260
   feeds 254
   installing packages with 252–254
       Visual Studio tooling 252–254
.nuget files 67
nuget packages 67
nUnit 308
nvarchar(max) datatype 127

O
object oriented programming 355
Object Relational Mapper (ORM) 120
   Entity Framework (EF) Core 120–146
       micro-ORMs 120
   System.Data 120
Octopus Deploy 67
OmniSharp 98
OneDrive 70
OnModelCreating method 142
OpenID Connect 201
open source 24
Open Web Interface for .NET (OWIN) 24–25, 58
Open Web Interface for .NET (OWIN) standard 13
operating systems 93, 107
options delegates 225
options pattern 176–177
orchestration system
   containers 109
Origin headers 218
ORM. See Object Relational Mapping
output caching 283
OWIN (Open Web Interface for .NET) 24–25

P
package.json file 255, 257, 258
package managers 252–262
   Bower 260–262
   npm 255–258, 259
   NuGet 252–254, 260
   Yarn 258–260
package restore 49, 55
packages
   building 67–68
       global installation of 256
Page-Controller pattern 28
parallel structure 375–376
partial views 31–32, 161, 305–306
Pass class 145
pass types 141–142
passwords
   encrypted 204
   hashing 204–205
   storing 204–205
   verifying 207
PayPass 39
PayWave 39
PCLs. See portable class libraries (PCLs)
performance
   isolation and 106–107
performance testing 50–51
Perl 6
PhantomJS 323
PHP 18
pipelines
   branching 332–333
       configuration 329–330
Pipes and Filters architecture 328
plain old CLR object (POCO) 12
Plain Old CLR Objects (POCOs) 120
platform as a service (PaaS) 69
Platform as a Service (PaaS) 117
Platform-as-a-Service (PaaS) 80–83, 88–89
platform services 80–82
   building applications using 83–89
pods 116
policies
   applying globally 212–213
       custom authorization 214–215
Radio Frequency Identification (RFID) chips 39–40
Rails. See Ruby on Rails
Razor 11–12, 18
   syntax 152, 154–155
Razor views 30, 147–168
   advanced functionality 162–166
   alternative view engines 167
   avoiding duplication in 167
   C# features 155–157
   code 153–154
   compilation 151
   defining model 156–157
   errors in 150
   essentials of 149–155
   including sections in 160–161
   injecting services into 162
   layouts 158–161
   localization of 346
   markup 154
   parser control cheat sheet for 154–155
   partial views 161, 305–306
   role of 149
   style sheets and 268
   tag helpers 296
   Tag Helpers 163–166
   view components and 302
   view data 156–157
   web development and 148–149
   writing expressions 152
React 23–24, 249–250
readme files 374
records
   change tracking 124–125
   querying for multiple 123
   querying for single 122
   saving 123
recycleOnFileChange setting 62
redirects 11
Redis 287
refactoring 353–365
   data driven changes and 365
   defined 354–355
   microservices 355
   time for 357–358
   with safety net 358–365
referential integrity 134–135
RefluxJS 249–250
registers 282
relationships
   across context boundaries 134–135
Remove Additional Files At Destination

Remove Additional Files At Destination 72
repeatable environments 106–110
repetition 21
repository structure 374
representational state transfer (REST) 13–14
requirements
  in authorization policies 214
resource files 343
  culture-specific 345–346
  sharing 347–348
resource groups 72
resources
  Azure Resource Explorer 82
  cross-origin resource sharing (CORS) 218–219
  protection of 215–217
  partial views 305–306
  tag helpers 296–302
  view components 302–305
reverse proxy 59–61
RFID. See Radio Frequency Identification (RFID) chips
RFID scanners 41–42
-r flag 68
rkt 110, 114–115
root directory 374
Roslyn 101
Rosslyn compiler 46
Route attribute 336–337
routing 21, 33, 335–337
  advanced 337
  attribute 336–337
  conventional 336
routing table 10
RT5500s 41–42
Ruby on Rails 18–21, 25
  conventions 19–21
  directory structure 19
runtime errors 12
RyuJIT project 102

S

SASS. See Syntactically Awesome Style Sheets
SASS script functions 273–274
Sassy CSS (SCSS) 268–275
  basics of 269–273
  directives 274
  imports and partials 270
  inheritance 271–272
  mixins 274–275
  nesting 272–273
  variables 270
SaveChanges method 123, 124–125
--save flag 255, 261
scaling
  web applications 89–92
Scoped lifetime 226
scopes 183–185
scripts 374
Scripts folder 234
script tag helper 297–299
SCSS. See Sassy CSS
search engine optimization 21
search engine optimization (SEO) 338
sections
  from views 160–161
security 193–220
  ASP.NET Core Identity 202–205
  authentication
    Azure-backed 197–202
  third-party authentication providers 208–211
  through attributes 210–211
  authorization
    policies for 212–215
  cross-origin resource sharing 218–219
  cross-site scripting attacks 232–233
  external threats 195
  internal threats 194–195
  passwords
    storing 204–205
  resource protection 215–217
  through attributes 210–211
  user secrets and 195–196, 201
selectors 267–268
self-contained packages 67, 68
Seq 187–188
Serilog library 185–187, 188
server push 22
servers
  build 48, 49, 51
  proxy 283, 284
  source control 49, 51
Server Side Includes 6
server side programming 231, 235–236
Service Locator pattern 224
services 117
adding 225–227
injection into views 162
lifetime of 226
localization 343–344
logging as 186–188
node 339
option 176–177
SetLanguage action method 349–350
SHA hashing function 204
shared libraries 108
SignalR 22
SignInManager class 206, 207–208, 208
sign up policies 200–201
Silverlight 102
Simple Object Access Protocol (SOAP) 13–14
Single Page Application (SPA) 234–235
single page applications (SPAs) 10, 338
single-tenant authentication 197
in Azure Active Directory 197
SkiCardContext class 133–140
SOAP. See Simple Object Access Protocol
social identity providers 199–200
Socket.io package 22
software
  continuous delivery of 49
  continuous deployment of 48
  inside containers 108
software releases 48
solution files 47
source code 38
  repository for 53, 54–55
  repository structure 374
source control 70
source control servers 49, 51
SPA. See Single Page Application
SPAs. See single page applications
SpaServices 338
spiky loads 69
Spring framework 18
SQL Server 100
distributed cache and 287–289
cache 288
cache busting 297–298
CDNs and fallbacks 298
creating 299–302
environment 297
glob patterns 297
handling existing attributes and contents 300–301
link 297–299
script 297–299
testing 316–319
Tag Helpers 163–166
Taglet 23

string localization 344–345
strongly typed languages 245
strongly-typed views 156
structured logging 185–187
StructureMap 228
StyleCop 370
Style folder 234
style sheets 263–280
  about 265
  creating 266–268
  custom 279–280
  directives 274
  imports and partials 270
  inheritance 271–272
  mixins 274–275
  nesting 272–273
  SASS script functions 273–274
  SCSS 268–275
  variables 270
Stylus 269
Symantec Ghost 106
Syntactically Awesome Style Sheets (SASS) 269
syntax
  Razor 152, 154–155
System.Data 120
System.Data.SqlClient 120
system.js module loader 247–248
system layers 108

table controls 9
tag helpers 35
tag helpers 296–302
  anatomy of 296–297
  attributes of 296
cache 298
  cache busting 297–298
  CDNs and fallbacks 298
creating 299–302
evironment 297
glob patterns 297
handling existing attributes and contents 300–301
link 297–299
script 297–299
testing 316–319
Tag Helpers 163–166
Taglet 23
tags

custom 23
target element 296
Task Runner Explorer 258
TeamCity 51, 65, 71
Team Foundation Server (TFS) 51–56, 71
technical debt 195
templates
  ARM 72–74
Test Explorer 310
testing 49–51, 55, 307–324
  automated 358
  characterization tests 358–359
  continuous 310
  controllers 313–316
  integration 324
  integration tests 50
  JavaScript 233, 320–323
  load 324
  mocking 223
  performance 50–51
  pyramid 50
  refactoring and 358–366
  tag helpers 316–319
  types of 324
  unit tests 49, 50, 55, 224, 308–323
  view components 319–320
text
  localizable 343–348
TFS. See Team Foundation Server
TFS Build 65
third-party authentication providers 208–211
third-party containers 228–229
third-party CSS frameworks 277–280
threads 106, 107
tightly coupled code 223
tokens 205
tools 374
trace logs 180
transpilers 244–246
Trello 10
triggers
  for builds 56
  tsconfig.json file 241
Twitter 10, 208, 210
two-factor authentication 208

TypeScript 22, 235, 237, 242, 243–247
  as ES2015 to ES5 compiler 243–245
  exit costs 247
  React and 249
  typing system 245–247
TypeScript compiler 255

Ubuntu 94–98
  dotnet command line interface (CLI) 94–98
  installing .NET Core 94
UglifyJsPlugin 242
unit tests 49, 50, 55, 224, 308–323
controllers 313–316
Jasmine 320–321
JavaScript 320–323
Karma 321–323
organizing 311–313, 321
running 309–310, 321–323
tag helpers 316–319
view components 319–320
xUnit 308–310, 316
Update Panels 8
update scripts
  for production servers 129–130
Up() method 125–126
URLs 21
UseCors() 218
UseIISIntegration 61
UseMvc() 218
user accounts
  ASP.NET Core Identity 206–208
  deleting 135
  lockout of 208
user controls 9, 12
user experience
  with Web Forms 9–10
user identity
  options for establishing 202
UserManager class 206
usernames
  verifying 207
users
  authentication of 207–208
  management of 206–207
user-secret command 196
user secrets 195–196, 201
V

validation
  passes and 142–146
variables 270
variable scoping 245
VB.NET 7
VBScript 6
-v flag 113
ViewBag 12
view component class 302
view components 31, 302–305
  example 303–304
  invoking 303
testing 319–320
view data 156–157
ViewData 12
view engines. See also Razor views
  alternative 167
_ViewImports 167
View method 20
view results 11
views 11, 30
  admin 42
  advanced functionality 162–166
  alternative view engines 167
  avoiding duplication in 167
  C# types in 155
  including sections from 160–161
  injecting services into 162
  loading external 334
  localization of 346
  partial 31–32, 161
  Rails 19
  Razor 30, 147–168
  strongly typed 156
  Tag Helpers 163–166
Views folder 381
ViewState container 10
view templates 12
virtual machines 69, 106, 107, 108
  scaling 89–90
Visual Basic 7, 8
Visual Studio 64–66
  Bower and 260
  Integrated Development Environment (IDE) 257
  npm integration with 257–258
  NuGet in 252–254
  Tag Helpers and 166
  Test Explorer 310

W

warning logs 180
WCF. See Windows Communication Foundation
Web API 13–15
web applications
  building
    using platform services 83–89
    with Azure 79–92
caching 92
developing 148–149
layouts for 158–161
outbound traffic 92
platform as a service and 80–83
  scaling 89–92
web browsers 243
  communication with servers 22
web.config 170
web.config file 61–62
Web Deployment 64
web development 148–149
  CSS and 264–280
  early days of 6
  project build tasks 277
  workflow for 275–276
Web development tools 22
WebDriver 324
Web Forms 8–10, 12, 18, 20, 21, 24
WebForms 158, 163, 374
WebHost 59
WebJobs 84, 88–89
WebJobs package 86
Webpack 240–243
web pages
  master pages 8–9, 13
  Web Forms 8–10
web servers 113
  caching on 282–290
  choosing 58
  communication with browsers 22
  IIS 64
  Kestrel 58–59

Visual Studio Code 22, 98
Visual Studio Code (VS Code) 276–277
Visual Studio Team Services (VSTS) 51–56, 65, 71–72
  GitHub and 54
VS Code 276–277
VSTS. See Visual Studio Team Services
WebSockets

load on 282
Nginx 62–63
reverse proxy and 59–61
WebSockets 22
Where method 123
Wikipedia 232
wildcard patterns 297
windows authentication tokens 61
Windows Communication Foundation (WCF) 13,
14–15
Windows containers 114–116
Windows Nano 107
containers 115
Windows Server Containers 115
Windows Server Core
containers 115
WinForms 8
WS-BPEL 13
WS-Security 13
wwwroot 33
wwwroot directory 233–234

X

XSS. See cross-site scripting attacks
xUnit 308–310, 316

Y

Yarn 258–260
yarn.lock file 259