

# Содержание

Об авторе	18
<b>Часть I. Введение в инфраструктуру ASP.NET Core MVC</b>	<b>19</b>
<b>Глава 1. Основы ASP.NET Core MVC</b>	<b>20</b>
История развития ASP.NET Core MVC	20
ASP.NET Web Forms	21
Первоначальная инфраструктура MVC Framework	22
Обзор ASP.NET Core	23
Основные преимущества ASP.NET Core MVC	24
Архитектура MVC	24
Расширяемость	24
Жесткий контроль над HTML и HTTP	25
Тестируемость	25
Мощная система маршрутизации	25
Современный API-интерфейс	26
Межплатформенная природа	26
Инфраструктура ASP.NET Core MVC имеет открытый код	27
Что необходимо знать?	27
Какова структура книги?	27
Часть I. Введение в инфраструктуру ASP.NET Core MVC	27
Часть II. Подробные сведения об инфраструктуре ASP.NET Core MVC	28
Что нового в этом издании?	28
Где можно получить код примеров?	28
Резюме	28
<b>Глава 2. Ваше первое приложение MVC</b>	<b>29</b>
Установка Visual Studio	29
Создание нового проекта ASP.NET Core MVC	31
Добавление первого контроллера	33
Понятие маршрутов	36
Визуализация веб-страниц	37
Создание и визуализация представления	37
Добавление динамического вывода	40
Создание простого приложения для ввода данных	42
Предварительная настройка	42
Проектирование модели данных	43
Создание второго действия и строго типизированного представления	44
Ссылка на методы действий	46
Построение формы	47
Получение данных формы	49
Добавление проверки достоверности	56
Стилизация содержимого	61
Резюме	67

<b>Глава 3. Паттерн MVC, проекты и соглашения</b>	68
История создания MVC	68
Особенности паттерна MVC	68
Понятие моделей	69
Понятие контроллеров	70
Понятие представлений	70
Реализация MVC в ASP.NET Core	70
Сравнение MVC с другими паттернами	71
Паттерн интеллектуального пользовательского интерфейса	71
Проекты ASP.NET Core MVC	75
Создание проекта	76
Соглашения в проекте MVC	79
Резюме	81
<b>Глава 4. Важные функциональные возможности языка C#</b>	82
Подготовка проекта для примера	83
Включение ASP.NET Core MVC	84
Создание компонентов приложения MVC	85
Использование null-условной операции	87
Связывание в цепочки null-условных операций	88
Комбинирование null-условной операции и операции объединения с null	89
Использование автоматически реализуемых свойств	90
Использование инициализаторов автоматически реализуемых свойств	91
Создание автоматически реализуемых свойств только для чтения	92
Использование интерполяции строк	93
Использование инициализаторов объектов и коллекций	94
Использование инициализатора индексированной коллекции	96
Использование расширяющих методов	97
Применение расширяющих методов к интерфейсу	98
Создание фильтрующих расширяющих методов	100
Использование лямбда-выражений	101
Определение функций	103
Использование методов и свойств в форме лямбда-выражений	105
Использование автоматического выведения типа и анонимных типов	107
Использование анонимных типов	108
Использование асинхронных методов	109
Работа с задачами напрямую	110
Применение ключевых слов async и await	111
Получение имен	113
Резюме	114
<b>Глава 5. Работа с Razor</b>	115
Подготовка проекта для примера	116
Определение модели	117
Создание контроллера	117
Создание представления	118
Работа с объектом модели	119
Использование файла импортирования представлений	121
Работа с компоновками	123
Создание компоновки	123

## 8 Содержание

Применение компоновки	125
Использование файла запуска представления	126
Использование выражений Razor	128
Вставка значений данных	129
Установка значений атрибутов	130
Использование условных операторов	131
Проход по содержимому массивов и коллекций	133
Резюме	135
<b>Глава 6. Работа с Visual Studio</b>	<b>136</b>
Подготовка проекта для примера	137
Создание модели	137
Создание контроллера и представления	139
Управление программными пакетами	141
Инструмент NuGet	141
Инструмент Bower	143
Итеративная разработка	147
Внесение изменений в представления Razor	147
Внесение изменений в классы C#	148
Использование средства Browser Link	156
Подготовка файлов JavaScript и CSS для развертывания	161
Включение доставки статического содержимого	161
Добавление в проект статического содержимого	162
Обновление представления	164
Пакетирование и минификация в приложениях MVC	166
Резюме	169
<b>Глава 7. Модульное тестирование приложений MVC</b>	<b>170</b>
Подготовка проекта для примера	171
Включение встроенных дескрипторных вспомогательных классов	171
Добавление действий к контроллеру	172
Создание формы для ввода данных	172
Обновление представления Index	173
Модульное тестирование приложений MVC	174
Создание проекта модульного тестирования	175
Изолирование компонентов для модульного тестирования	182
Улучшение модульных тестов	190
Параметризация модульного теста	190
Улучшение фиктивных реализаций	194
Резюме	200
<b>Глава 8. SportsStore: реальное приложение</b>	<b>201</b>
Начало работы	202
Создание проекта MVC	202
Создание проекта модульного тестирования	207
Проверка и запуск приложения	209
Начало работы с моделью предметной области	209
Создание хранилища	210
Создание фиктивного хранилища	210
Регистрация службы хранилища	211

Отображение списка товаров	212
Добавление контроллера	213
Добавление и конфигурирование представления	215
Установка стандартного маршрута	216
Запуск приложения	217
Подготовка базы данных	218
Установка Entity Framework Core	219
Создание классов базы данных	220
Создание класса хранилища	221
Определение строки подключения	222
Конфигурирование приложения	223
Создание и применение миграции базы данных	225
Добавление поддержки разбиения на страницы	226
Отображение ссылок на страницы	228
Улучшение URL	236
Стилизация содержимого	237
Установка пакета Bootstrap	238
Применение стилей Bootstrap к компоновке	238
Создание частичного представления	241
Резюме	242
<b>Глава 9. SportsStore: навигация</b>	<b>244</b>
Добавление навигационных элементов управления	244
Фильтрация списка товаров	244
Улучшение схемы URL	248
Построение меню навигации по категориям	252
Корректировка счетчика страниц	259
Построение корзины для покупок	262
Определение модели корзины	262
Создание кнопок добавления в корзину	266
Включение поддержки сеансов	267
Реализация контроллера для корзины	269
Отображение содержимого корзины	272
Резюме	275
<b>Глава 10. SportsStore: завершение построения корзины для покупок</b>	<b>276</b>
Усовершенствование модели корзины с помощью службы	276
Создание класса корзины, осведомленного о хранилище	276
Регистрация службы	277
Упрощение контроллера Cart	278
Завершение функциональности корзины	279
Удаление элементов из корзины	279
Добавление виджета с итоговой информацией по корзине	281
Отправка заказов	284
Создание класса модели	284
Добавление реализации процесса оплаты	285
Реализация обработки заказов	287
Завершение построения контроллера Order	291
Отображение сообщений об ошибках проверки достоверности	295
Отображение итоговой страницы	296
Резюме	297

<b>Глава 11. SportsStore: администрирование</b>	298
Управление заказами	298
Расширение модели	298
Добавление действий и представления	299
Добавление средств управления каталогом	302
Создание контроллера CRUD	303
Реализация представления списка	304
Редактирование сведений о товарах	306
Создание новых товаров	320
Удаление товаров	321
Резюме	324
<b>Глава 12. SportsStore: защита и развертывание</b>	325
Защита средств администрирования	325
Добавление в проект пакета Identity	325
Создание базы данных Identity	326
Применение базовой политики авторизации	330
Создание контроллера Account и представлений	332
Тестирование политики безопасности	336
Развертывание приложения	336
Создание баз данных	337
Подготовка приложения	338
Применение миграций баз данных	343
Процесс развертывания приложения	343
Резюме	346
<b>Глава 13. Работа с Visual Studio Code</b>	348
Настройка среды разработки	348
Установка Node.js	349
Проверка установки Node	350
Установка Git	350
Проверка установки Git	351
Установка Yeoman, Bower и Gulp	351
Установка .NET Core	351
Проверка установки .NET Core	352
Установка Visual Studio Code	353
Проверка установки Visual Studio Code	353
Установка расширения C# для Visual Studio Code	353
Создание проекта ASP.NET Core	355
Подготовка проекта с помощью Visual Studio Code	356
Добавление в проект пакетов NuGet	357
Добавление в проект пакетов клиентской стороны	358
Конфигурирование приложения	359
Построение и запуск проекта	359
Воссоздание приложения PartyInvites	360
Создание модели и хранилища	360
Создание базы данных	363
Создание контроллеров и представлений	365
Модульное тестирование в Visual Studio Code	369
Конфигурирование приложения	369
Создание модульного теста	371
Прогон тестов	371
Резюме	372

<b>Часть II. Подробные сведения об инфраструктуре ASP.NET Core MVC</b>	<b>373</b>
<b>Глава 14. Конфигурирование приложений</b>	<b>374</b>
Подготовка проекта для примера	376
Конфигурационные файлы JSON	377
Конфигурирование решения	378
Конфигурирование проекта	380
Класс Program	383
Класс Startup	385
Особенности использования класса Startup	386
Службы ASP.NET	387
Промежуточное программное обеспечение ASP.NET	390
Особенности вызова метода Configure()	400
Добавление оставшихся компонентов промежуточного программного обеспечения	408
Использование данных конфигурации	413
Конфигурирование служб MVC	419
Работа со сложными конфигурациями	421
Создание разных внешних конфигурационных файлов	421
Создание разных методов конфигурирования	422
Создание разных классов конфигурирования	424
Резюме	425
<b>Глава 15. Маршрутизация URL</b>	<b>426</b>
Подготовка проекта для примера	428
Создание класса модели	429
Создание контроллеров	430
Создание представления	431
Введение в шаблоны URL	432
Создание и регистрация простого маршрута	434
Определение стандартных значений	435
Определение встроенных стандартных значений	436
Использование статических сегментов URL	438
Определение специальных переменных сегментов	442
Использование специальных переменных в качестве параметров метода действия	445
Определение необязательных сегментов URL	446
Определение маршрутов переменной длины	448
Ограничение маршрутов	451
Ограничение маршрута с использованием регулярного выражения	454
Использование ограничений на основе типов и значений	455
Объединение ограничений	456
Определение специального ограничения	458
Использование маршрутизации с помощью атрибутов	460
Подготовка для маршрутизации с помощью атрибутов	460
Применение маршрутизации с помощью атрибутов	461
Применение ограничений к маршрутам	463
Резюме	464

<b>Глава 16. Дополнительные возможности маршрутизации</b>	465
Подготовка проекта для примера	467
Генерация исходящих URL в представлениях	468
Генерирование исходящих ссылок	468
Генерация URL (без ссылок)	478
Генерирование URL в методах действий	478
Настройка системы маршрутизации	479
Изменение конфигурации системы маршрутизации	479
Создание специального класса маршрута	481
Применение специального класса маршрута	484
Маршрутизация на контроллеры MVC	485
Работа с областями	491
Создание области	492
Создание маршрута для области	492
Заполнение области	493
Генерирование ссылок на действия в областях	496
Полезные советы относительно схемы URL	497
Делайте URL чистыми и понятными человеку	497
GET и POST: выбор правильного запроса	499
Резюме	499
<b>Глава 17. Контроллеры и действия</b>	500
Подготовка проекта для примера	501
Подготовка представлений	503
Понятие контроллеров	505
Создание контроллеров	506
Создание контроллеров POCO	506
Использование базового класса Controller	508
Получение данных контекста	509
Получение данных из объектов контекста	510
Использование параметров метода действия	514
Генерирование ответа	516
Генерирование ответа с использованием объекта контекста	516
Понятие результатов действий	517
Генерирование HTML-ответа	520
Передача данных из метода действия в представление	523
Выполнение перенаправления	528
Возвращение разных типов содержимого	535
Реагирование с помощью содержимого файлов	538
Возвращение ошибок и кодов HTTP	540
Другие классы результатов действий	542
Резюме	542
<b>Глава 18. Внедрение зависимостей</b>	543
Подготовка проекта для примера	544
Создание модели и хранилища	545
Создание контроллера и представления	547
Создание проекта модульного тестирования	548
Создание слабо связанных компонентов	549
Исследование сильно связанных компонентов	550

Введение в средство внедрения зависимостей ASP.NET	556
Подготовка к внедрению зависимостей	556
Конфигурирование поставщика служб	558
Модульное тестирование контроллера с зависимостью	560
Использование цепочек зависимостей	560
Использование внедрения зависимостей для конкретных типов	563
Жизненные циклы служб	565
Использование переходного жизненного цикла	566
Использование жизненного цикла, ограниченного областью действия	570
Использование жизненного цикла одиночки	571
Использование внедрения в действия	572
Использование атрибутов внедрения в свойства	573
Запрашивание объекта реализации вручную	574
Резюме	575
<b>Глава 19. Фильтры</b>	576
Подготовка проекта для примера	577
Включение SSL	579
Создание контроллера и представления	579
Использование фильтров	581
Понятие фильтров	584
Получение данных контекста	585
Использование фильтров авторизации	585
Создание фильтра авторизации	586
Использование фильтров действий	588
Создание фильтра действий	590
Создание асинхронного фильтра действий	591
Использование фильтров результатов	592
Создание фильтра результатов	593
Создание асинхронного фильтра результатов	595
Создание гибридного фильтра действий/результатов	596
Использование фильтров исключений	598
Создание фильтра исключений	599
Использование внедрения зависимостей для фильтров	600
Распознавание зависимостей в фильтрах	601
Управление жизненными циклами фильтров	605
Создание глобальных фильтров	608
Порядок применения фильтров и его изменение	610
Изменения порядка применения фильтров	612
Резюме	613
<b>Глава 20. Контроллеры API</b>	614
Подготовка проекта для примера	615
Создание модели и хранилища	615
Создание контроллера и представлений	617
Конфигурирование приложения	619
Роль контроллеров REST	621
Проблема скорости	621
Проблема эффективности	622
Проблема открытости	622



## 14 Содержание

Введение в REST и контроллеры API	623
Создание контроллера API	623
Тестирование контроллера API	628
Использование контроллера API в браузере	631
Форматирование содержимого	634
Стандартная политика содержимого	635
Согласование содержимого	636
Указание формата данных для действия	639
Получение формата данных из маршрута или строки запроса	639
Включение полного согласования содержимого	641
Получение разных форматов данных	643
Резюме	643
<b>Глава 21. Представления</b>	644
Подготовка проекта для примера	645
Создание специального механизма визуализации	647
Создание специальной реализации интерфейса IView	649
Создание реализации интерфейса IViewEngine	650
Регистрация специального механизма визуализации	651
Тестирование механизма визуализации	652
Работа с механизмом Razor	654
Подготовка проекта для примера	654
Прояснение представлений Razor	656
Добавление динамического содержимого к представлению Razor	660
Использование разделов компоновки	661
Использование частичных представлений	666
Добавление содержимого JSON в представления	669
Конфигурирование механизма Razor	671
Расширители местоположений представлений	672
Резюме	677
<b>Глава 22. Компоненты представлений</b>	678
Подготовка проекта для примера	679
Создание моделей и хранилищ	680
Создание контроллера и представлений	682
Конфигурирование приложения	685
Понятие компонентов представлений	686
Создание компонента представления	686
Создание компонентов представлений POCO	686
Наследование от базового класса ViewComponent	688
Понятие результатов компонентов представлений	690
Получение данных контекста	695
Создание асинхронных компонентов представлений	701
Создание гибридных компонентов контроллеров/представлений	703
Создание гибридных представлений	704
Применение гибридного класса	706
Резюме	707
<b>Глава 23. Дескрипторные вспомогательные классы</b>	708
Подготовка проекта для примера	709
Создание модели и хранилища	710

Создание контроллера, компоновки и представлений	711
Конфигурирование приложения	713
Создание дескрипторного вспомогательного класса	715
Определение дескрипторного вспомогательного класса	715
Регистрация дескрипторных вспомогательных классов	719
Использование дескрипторного вспомогательного класса	719
Управление областью действия дескрипторного вспомогательного класса	721
Усовершенствованные возможности дескрипторных вспомогательных классов	726
Создание сокращающих элементов	726
Вставка содержимого перед и после элементов	728
Получение данных контекста представления и использование внедрения зависимостей	732
Работа с моделью представления	734
Согласование дескрипторных вспомогательных классов	736
Подавление выходного элемента	738
Резюме	740
<b>Глава 24. Использование дескрипторных вспомогательных классов для форм</b>	741
Подготовка проекта для примера	743
Изменение регистрации дескрипторных вспомогательных классов	743
Переустановка представлений и компоновки	743
Работа с элементами form	745
Установка цели формы	746
Использование средства противодействия подделке	746
Работа с элементами input	749
Конфигурирование элементов input	749
Форматирование значений данных	751
Работа с элементами label	754
Работа с элементами select и option	756
Использование источника данных для заполнения элемента select	758
Генерирование элементов option из перечисления	758
Работа с элементами textarea	763
Дескрипторные вспомогательные классы для проверки достоверности форм	764
Резюме	765
<b>Глава 25. Использование других встроенных дескрипторных вспомогательных классов</b>	766
Подготовка проекта для примера	767
Использование вспомогательного класса для среды размещения	768
Использование вспомогательных классов для JavaScript и CSS	769
Управление файлами JavaScript	769
Управление таблицами стилей CSS	778
Работа с якорными элементами	781
Работа с элементами img	782
Использование кеша данных	783
Установка времени истечения для кеша	786
Использование вариаций кеша	788
Использование URL, относительных к приложению	789
Резюме	792

<b>Глава 26. Привязка моделей</b>	793
Подготовка проекта для примера	794
Создание модели и хранилища	795
Создание контроллера и представления	796
Конфигурирование приложения	798
Понятие привязки моделей	799
Стандартные значения привязки	801
Привязка простых типов	802
Привязка сложных типов	803
Привязка массивов и коллекций	813
Привязка коллекций сложных типов	817
Указание источника данных привязки моделей	820
Выбор стандартного источника данных привязки	820
Использование заголовков в качестве источников данных привязки	821
Использование тел запросов в качестве источников данных привязки	824
Резюме	827
<b>Глава 27. Проверка достоверности моделей</b>	828
Подготовка проекта для примера	830
Создание модели	831
Создание контроллера	831
Создание компоновки и представлений	832
Необходимость в проверке достоверности модели	834
Явная проверка достоверности модели	835
Отображение пользователю ошибок проверки достоверности	838
Отображение сообщений об ошибках проверки достоверности	839
Отображение сообщений об ошибках проверки достоверности на уровне свойств	844
Отображение сообщений об ошибках проверки достоверности на уровне модели	845
Указание правил проверки достоверности с помощью метаданных	849
Создание специального атрибута проверки достоверности для свойства	852
Выполнение проверки достоверности на стороне клиента	854
Выполнение удаленной проверки достоверности	857
Резюме	860
<b>Глава 28. Введение в ASP.NET Core Identity</b>	861
Подготовка проекта для примера	863
Создание контроллера и представления	864
Настройка ASP.NET Core Identity	866
Добавление пакета Identity в приложение	866
Создание класса пользователя	867
Создание класса контекста базы данных	869
Конфигурирование настройки строки подключения к базе данных	869
Конфигурирование служб и промежуточного программного обеспечения Identity	870
Создание базы данных Identity	872
Использование ASP.NET Core Identity	872
Перечисление пользовательских учетных записей	873
Создание пользователей	875
Проверка паролей	879
Проверка деталей, связанных с пользователем	886

Завершение средств администрирования	890
Реализация средства удаления	891
Реализация возможности редактирования	892
Резюме	897
<b>Глава 29. Применение ASP.NET Core Identity</b>	898
Подготовка проекта для примера	898
Аутентификация пользователей	899
Подготовка к реализации аутентификации	901
Добавление аутентификации пользователей	904
Тестирование аутентификации	907
Авторизация пользователей с помощью ролей	908
Создание и удаление ролей	909
Управление членством в ролях	914
Использование ролей для авторизации	919
Помещение в базу данных начальных данных	923
Резюме	925
<b>Глава 30. Расширенные средства ASP.NET Core Identity</b>	926
Подготовка проекта для примера	927
Добавление специальных свойств в класс пользователя	927
Подготовка миграции базы данных	931
Тестирование специальных свойств	931
Работа с заявками и политиками	932
Понятие заявок	933
Создание заявок	937
Использование политик	940
Использование политик для авторизации доступа к ресурсам	946
Использование сторонней аутентификации	951
Регистрация приложения в Google	951
Включение аутентификации Google	952
Резюме	957
<b>Глава 31. Соглашения по модели и ограничения действий</b>	958
Подготовка проекта для примера	959
Создание модели представления, контроллера и представления	960
Использование модели приложения и соглашений по модели	962
Модель приложения	963
Роль соглашений по модели	967
Создание соглашения по модели	968
Порядок выполнения соглашений по модели	973
Создание глобальных соглашений по модели	974
Использование ограничений действий	976
Подготовка проекта для примера	977
Ограничения действий	978
Создание ограничения действия	979
Распознавание зависимостей в ограничениях действий	984
Резюме	986
<b>Предметный указатель</b>	987

# ГЛАВА 13

## Работа с Visual Studio Code

**В** настоящей главе будет показано, как создать приложение ASP.NET Core MVC с применением Visual Studio Code — межплатформенного редактора с открытым кодом производства Microsoft. Несмотря на название, продукт Visual Studio Code не имеет отношения к Visual Studio и основан на инфраструктуре Electron, используемой редактором Atom, который популярен среди разработчиков, применяющих другие инфраструктуры для построения веб-приложений, такие как Angular.

Продукт Visual Studio Code поддерживает операционные системы Windows, OS X/macOS и наиболее популярные дистрибутивы Linux. Продукт Visual Studio Code находится на начальной стадии своего развития, поэтому не все средства работают должным образом; однако в Microsoft предлагают ежемесячные обновления и продвижение происходит довольно быстро. Некоторые текущие ограничения, такие как отсутствие поддержки отладки для приложений ASP.NET Core, могут быть устранены ко времени выхода этой книги, но выполнение всех примеров в данной книге по-прежнему требует Visual Studio и Windows.

Процесс разработки в Visual Studio Code менее автоматизирован, чем в полной версии Visual Studio, но он вполне осуществим, предлагая пристойную отправную точку для построения приложений ASP.NET Core MVC в средах других операционных систем или легковесную альтернативу Visual Studio 2015 в Windows.

---

**На заметку!** В Microsoft заявили, что инструментарий, используемый для создания приложений ASP.NET Core MVC, в будущем изменится. Проверяйте веб-сайт издательства на предмет обновлений, которые появятся после выпуска новых инструментов.

---

### Настройка среды разработки

Настройка Visual Studio Code требует выполнения определенной работы, поскольку некоторая функциональность, включенная в Visual Studio, обрабатывается здесь внешними инструментами. Одни инструменты идентичны тем, которые среда Visual Studio применяет “за кулисами”, но другие являются новыми в мире разработки для .NET и могут быть незнакомыми. Хорошая новость в том, что эти инструменты широко используются разработчиками, которые ориентируются на другие инфраструктуры для построения веб-приложений, причем качество и функциональные средства находятся на высоком уровне. В последующих разделах мы исследуем процесс установки Visual Studio Code наряду с важными инструментами и дополнениями, требующимися для разработки приложений MVC.

## Установка Node.js

В мире разработки клиентской стороны Node.js (или просто Node) представляет собой исполняющую среду, на которую полагаются многие популярные инструменты разработки. Node была создана в 2009 году как простая и эффективная исполняющая среда для серверных приложений, написанных на языке JavaScript. Она основана на механизме JavaScript, применяемом в браузере Chrome, и предлагает API-интерфейс для выполнения кода JavaScript за пределами среды браузера.

Исполняющая среда Node.js достигла определенных успехов в качестве сервера приложений, но в этой главе она интересна тем, что предоставляет основу для нового поколения межплатформенных инструментов построения и диспетчеров пакетов. Ряд интеллектуальных проектных решений, внедренных командой разработчиков Node, и межплатформенная поддержка, обеспечиваемая исполняющей средой JavaScript браузера Chrome, создали благоприятную возможность, которой воспользовались полные энтузиасты проектировщики инструментов, особенно те из них, кто желал поддерживать разработку веб-приложений.

---

**На заметку!** Доступны две версии Node.js. Версия LTS (Long Term Support — долгосрочная поддержка) предоставляет стабильный фундамент для развертывания в производственных средах, где изменения должны быть сведены к минимуму. Обновления версии LTS выпускаются каждые 6 месяцев и сопровождаются в течение 18 месяцев. Версия Current (текущая) является более часто изменяющимся выпуском, в котором преимущество отдается новым средствам взамен стабильности. В настоящей главе мы будем применять выпуск Current.

---

### Установка Node.js в Windows

Загрузите и запустите установщик Node.js для Windows из веб-сайта <http://nodejs.org>. При установке Node.js удостоверьтесь, что выбран вариант Add to PATH (Добавить в переменную PATH). На рис. 13.1 показан установщик для Windows, который предлагает модифицировать переменную среды PATH в качестве варианта установки.

### Установка Node.js в OS X/macOS

Установщик для OS X/macOS может быть загружен из веб-сайта <http://nodejs.org>. Запустите установщик и примите стандартные настройки. Когда установка завершится, удостоверьтесь в наличии пути `/usr/local/bin` в переменной `$PATH`.

### Установка Node.js в Linux

Самый простой способ установки Node.js в Linux предполагает использование диспетчера пакетов; команда разработчиков Node предоставила инструкции для основных дистрибутивов по адресу <http://nodejs.org/en/download/package-manager>. В среде Ubuntu для загрузки и установки Node.js применяются следующие команды:

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash
-sudo apt-get install -y nodejs
```

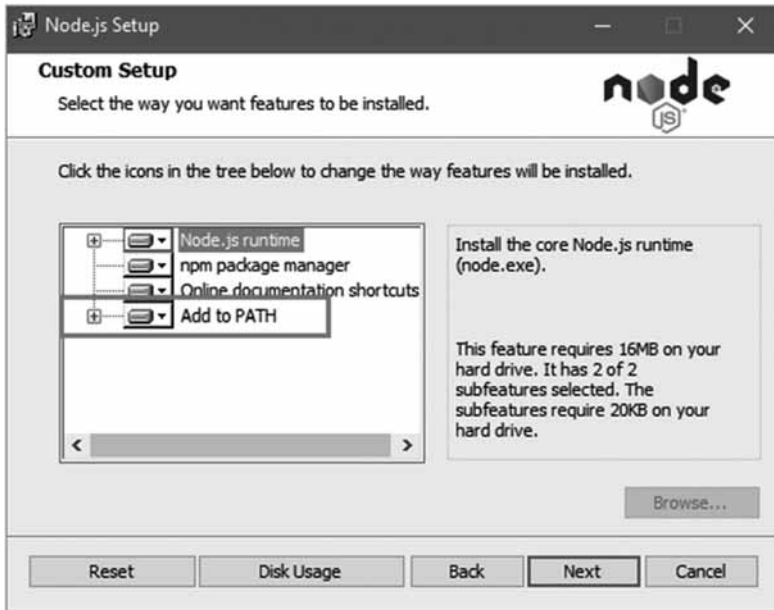


Рис. 13.1. Добавление в переменную среды PATH

## Проверка установки Node

После завершения установки откройте новое окно командной строки и запустите показанную ниже команду:

```
node -v
```

Если установка прошла успешно, а путь к Node был добавлен в переменную среды PATH, тогда вы увидите номер версии. На момент написания главы текущей версией Node являлась 6.3.0. Если во время проработки примеров, рассмотренных в главе, вы получаете неожиданные результаты, то попробуйте воспользоваться указанной конкретной версией.

## Установка Git

Продукт Visual Studio Code включает интегрированную поддержку Git, но требуется отдельная установка для поддержки инструмента Bower, который применяется при управлении пакетами клиентской стороны.

### Установка Git в Windows или OS X/macOS

Загрузите и запустите установщик из веб-сайта <https://git-scm.com/downloads>.

### Установка Git в Linux

В большинстве дистрибутивов Linux инструмент Git уже установлен. Если вы хотите установить его в любом случае, тогда ознакомьтесь с инструкциями по установке для вашего дистрибутива по адресу <https://git-scm.com/download/linux>.

В среде Ubuntu используется следующая команда:

```
sudo apt-get install git
```

## Проверка установки Git

После завершения установки запустите приведенную ниже команду в новом окне командной строки/терминала, чтобы проверить, установлен и доступен ли инструмент Git:

```
git --version
```

Команда выведет версию установленного пакета Git. На момент написания главы последней версией Git для Windows была 2.9.0, а для OS X/macOS/Linux — 2.8.1.

## Установка Yeoman, Bower и Gulp

Node.js поступает с диспетчером пакетов Node (Node Package Manager — NPM), который применяется для загрузки и установки пакетов разработки, написанных на JavaScript. Пакеты, полезные для разработки приложений ASP.NET Core MVC, описаны в табл. 13.1.

**Таблица 13.1. Пакеты NPM, полезные для разработки приложений ASP.NET Core**

Имя	Описание
yo	Пакет Yeoman (известный как yo) — это инструмент, который облегчает начало новых разрабатываемых проектов, устанавливая исходное содержимое, как демонстрируется в разделе “Создание проекта ASP.NET Core” далее в главе
bower	Это тот же самый инструмент Bower, описанный в главе 6, который используется для управления пакетами клиентской стороны
generator-aspnet	Этот пакет снабжает Yeoman информацией, необходимой для создания новых проектов ASP.NET Core MVC, как объясняется в разделе “Создание проекта ASP.NET Core” далее в главе

В среде Windows указанные пакеты устанавливаются с помощью такой команды:

```
npm install -g yo@1.8.4 bower@1.7.9 generator-aspnet@0.2.1
```

Ко времени выхода книги могут появиться более новые версии этих пакетов, но при создании примера применялись версии, упомянутые в командах установки. В средах Linux и OS X/macOS установка производится посредством команды sudo:

```
sudo npm install -g yo@1.8.4 bower@1.7.9 generator-aspnet@0.2.1
```

## Установка .NET Core

При разработке приложений ASP.NET Core MVC требуется исполняющая среда .NET Core. Для каждой поддерживаемой платформы предусмотрен собственный процесс установки, описание которого доступно по адресу [www.microsoft.com/net/core](http://www.microsoft.com/net/core). В Microsoft предлагают установщики для Windows и OS X/macOS, а также предоставляют инструкции для установки в Linux с использованием архивов tar.



## Установка .NET Core в Windows

Чтобы установить .NET Core в Windows, просто загрузите и запустите установщик .NET Core SDK (который отделен от установщика .NET Core для Visual Studio, необходимого в главе 2).

## Установка .NET Core в OS X/macOS

Перед запуском установщика .NET Core в среде macOS должна быть установлена последняя версия пакета OpenSSL; в Microsoft рекомендуют применять для этого диспетчер пакетов Homebrew. Откройте новое окно терминала и выполните следующую команду, чтобы установить Homebrew:

```
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Для обновления пакета OpenSSL запустите показанные ниже команды:

```
brew update
brew install openssl
brew link --force openssl
```

Загрузите установщик .NET Core, находящийся по адресу <https://go.microsoft.com/fwlink/?LinkID=809124>, и запустите его, чтобы добавить .NET Core в свою систему.

## Установка .NET Core в Linux

По адресу [www.microsoft.com/net/core](http://www.microsoft.com/net/core) предоставлены инструкции по установке .NET Core в большинстве популярных дистрибутивов Linux. В настоящей главе используется Ubuntu, и процесс требует первоначальной настройки новой подачи для apt-get с применением следующих команд:

```
sudo sh -c 'echo "deb [arch=amd64]
https://apt-mo.trafficmanager.net/repos/dotnet/ trusty
main" > /etc/apt/sources.list.d/dotnetdev.list'
sudo apt-key adv --keyserver apt-mo.trafficmanager.net
--recv-keys 417A0893
sudo apt-get update
```

Далее производится установка .NET Core:

```
sudo apt-get install dotnet-dev-1.0.0-preview2-003121
```

## Проверка установки .NET Core

Независимо от имеющейся платформы проверка того, что инфраструктура .NET Core установлена и готова к использованию, осуществляется одинаково. Откройте новое окно командной строки/терминала и выполните такую команду:

```
dotnet --version
```

Команда dotnet запускает исполняющую среду .NET, после чего отобразится номер версии установленного пакета .NET. На момент написания главы текущим выпуском был 1.0.0-preview2-003121, но ко времени выхода книги он наверняка будет заменен более новым выпуском.

## Установка Visual Studio Code

Самым важным шагом является загрузка и установка редактора Visual Studio Code, который доступен на веб-сайте <http://code.visualstudio.com>. Установочные пакеты предусмотрены для Windows, OS X/macOS и популярных дистрибутивов Linux. Загрузите и установите пакеты для предпочитаемой платформы.

---

**На заметку!** Компания Microsoft предлагает новые выпуски Visual Studio Code ежемесячно, а это значит, что устанавливаемая вами версия будет отличаться от версии 1.3, которая применяется в главе. Хотя основы должны остаться теми же самыми, выполнение некоторых примеров может потребовать определенной доли экспериментирования.

---

### Установка Visual Studio Code в Windows

Чтобы установить Visual Studio Code для Windows, просто запустите установщик. После завершения процесса Visual Studio Code запустится, и вы увидите окно редактора, представленное на рис. 13.2.

### Установка Visual Studio Code в OS X/macOS

Продукт Visual Studio Code предоставляется в виде архива zip для Mac, который доступен для загрузки по адресу <https://go.microsoft.com/fwlink/?LinkID=620882>. Раскройте архив и дважды щелкните на содержащемся в нем файле Visual Studio Code.app, чтобы запустить Visual Studio Code и получить окно редактора, показанное на рис. 13.2.

### Установка Visual Studio Code в Linux

Компания Microsoft предлагает файл .deb для Debian и Ubuntu и файл .rpm для Red Hat, Fedora и CentOS. Загрузите и установите файл для подходящего дистрибутива Linux. Поскольку в главе используется Ubuntu, понадобится загрузить файл .deb и установить его с помощью следующей команды:

```
sudo dpkg -i code_1.3.0-1467909982_amd64.deb
```

После завершения установки выполните приведенную ниже команду, чтобы запустить Visual Studio Code и получить окно редактора, показанное на рис. 13.2:

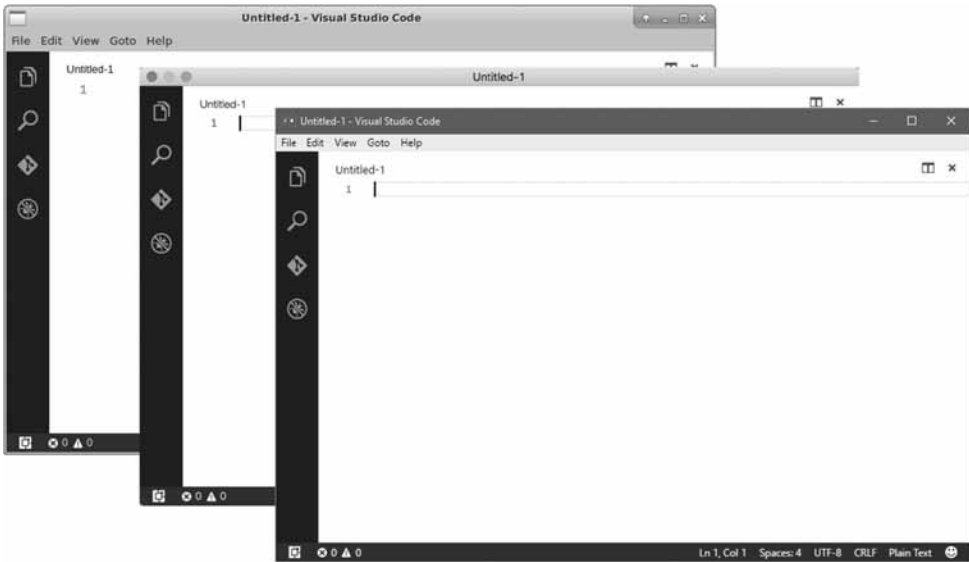
```
/usr/share/code/code
```

## Проверка установки Visual Studio Code

Тестирование успешности установки Visual Studio Code сводится к проверке возможности запуска приложения и открытию окна редактора (см. рис. 13.2). (Цветовая схема была изменена, т.к. стандартные темные цвета не очень хорошо подходят для получения экранных снимков.)

## Установка расширения C# для Visual Studio Code

Редактор Visual Studio Code поддерживает функциональность, специфичную для языка, через расширения, хотя это не те же самые расширения, которые поддерживаются средой Visual Studio 2015. Самое важное расширение, касающееся разработки приложений ASP.NET Core MVC, добавляет поддержку для C#, отсутствие которой может выглядеть как странное упущение в базовой установке.



**Рис. 13.2.** Выполнение Visual Studio Code в средах Windows, OS X/macOS и Ubuntu Linux

Тем не менее, такое положение дел отражает тот факт, что в Microsoft позиционировали Visual Studio Code как универсальный межплатформенный редактор, который поддерживает широчайший спектр языков и инфраструктур.

Чтобы установить расширение C#, выберите пункт Command Palette (Палитра команд) в меню View (Вид) редактора Visual Studio Code. Палитра команд предоставляет доступ ко всем командам, которые можно выполнять с помощью Visual Studio Code. Введите `ext` и нажмите на клавишу `<Return>`, в результате чего Visual Studio Code откроет окно Extensions (Расширения). Введите `csharp` и отыщите в списке расширение C# for Visual Studio Code (C# для Visual Studio Code), как показано на рис. 13.3.

Щелкните на кнопке Install (Установить), после чего Visual Studio Code загрузит и установит расширение. Щелкните на кнопке Enable (Включить), чтобы активизировать расширение (рис. 13.4).



**Рис. 13.3.** Нахождение расширения C#

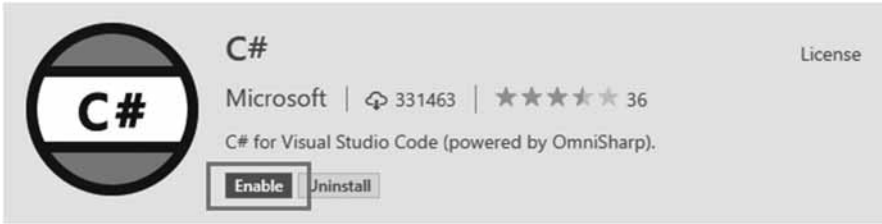


Рис. 13.4. Включение расширения C#

## Создание проекта ASP.NET Core

Редактор Visual Studio Code не имеет интегрированной поддержки для создания проектов ASP.NET Core и в настройке начальной структуры папок и файлов полагается на пакет Yeoman, который применяет шаблоны, предоставляемые пакетом `generator-aspnet`. Откройте новое окно командной строки/терминала, перейдите в каталог, где нужно создать проекты ASP.NET Core, и выполните следующую команду:

```
yo aspnet
```

Эта команда запускает пакет Yeoman и сообщает ему о необходимости создания нового проекта ASP.NET Core. Весь процесс настройки проекта производится через командную строку, переходя по параметрам с использованием клавиш со стрелками и делая выбор с применением клавиши <Return>. В табл. 13.2 описан набор шаблонов проектов, доступных для разработки приложений ASP.NET Core. (Есть еще несколько других шаблонов, которые здесь не перечислены, но они не предназначены для ASP.NET Core.)

**Таблица 13.2. Шаблоны проектов ASP.NET Core из Yeoman для разработки приложений ASP.NET Core**

Имя	Описание
Empty Web Application (Пустое веб-приложение)	Этот шаблон создает проект ASP.NET Core с минимальным начальным содержимым; он похож на шаблон Empty (Пустой) в Visual Studio 2015
Web Application (Веб-приложение)	Этот шаблон создает проект ASP.NET Core с начальным содержимым, которое включает контроллеры, представления и аутентификацию. Он похож на шаблон Web Application (Веб-приложение) в Visual Studio 2015 с включенной аутентификацией
Web Application Basic (Базовое веб-приложение)	Этот шаблон создает проект ASP.NET Core с начальным содержимым, которое включает контроллеры и представления, но не аутентификацию
Web API Application (Веб-приложение API)	Этот шаблон создает проект ASP.NET Core с контроллером API (который будет описан в главе 20). Он эквивалентен шаблону Web API в Visual Studio 2015

Выберите шаблон `Empty Web Application` и нажмите `<Return>`. В ответ на запрос имени для проекта введите `PartyInvites`. Вот вывод, который вы будете видеть во время создания проекта:

```
? What type of application do you want to create? Empty Web Application
? Какой тип приложения вы хотите создать?

? What's the name of your ASP.NET application? (EmptyWebApplication)
PartyInvites
? Каково имя вашего приложения ASP.NET? (EmptyWebApplication)
```

Нажмите `<Return>`; пакет `Yeoman` создаст папку `PartyInvites` и наполнит ее минимальным набором файлов, которые требуются для проекта `ASP.NET Core`.

## Подготовка проекта с помощью Visual Studio Code

Чтобы открыть проект в `Visual Studio Code`, выберите пункт `Open Folder` (Открыть папку) в меню `File` (Файл), перейдите в папку `PartyInvites` и щелкните на кнопке `Select Folder` (Выбрать папку). Редактор `Visual Studio Code` откроет проект, и по прошествии нескольких секунд вы увидите сообщение, предлагающее добавить элементы в проект (рис. 13.5).

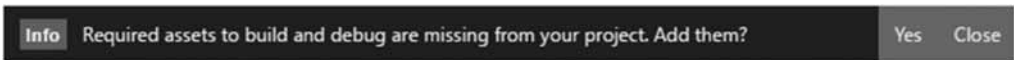


Рис. 13.5. Приглашение добавить активы в проект

Щелкните на кнопке `Yes` (Да). Редактор `Visual Studio Code` создаст папку `.vscode` и добавит в нее файлы, которые конфигурируют процесс построения. По умолчанию `Visual Studio Code` использует компоновку из трех разделов. Боковая панель, выделенная на рис. 13.6, предоставляет доступ к главным областям функциональности.

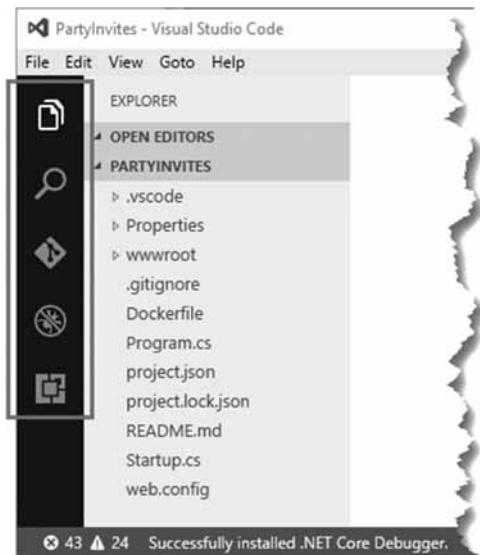


Рис. 13.6. Боковая панель Visual Studio Code

Самая верхняя кнопка позволяет открыть панель проводника, которая отображает содержимое ранее открытой папки. Остальные кнопки обеспечивают доступ к средству поиска, к интегрированному управлению исходным кодом, к отладчику и к набору установленных расширений. Щелчок на имени файла в панели проводника приводит к открытию файла для редактирования. Допускается редактировать несколько файлов одновременно, и вы можете создавать новые панели редактора, щелкая на кнопке `Split Editor` (Разделить окно редактора) в правой верхней части окна. Редактор `Visual Studio Code` вполне хорош, обладая неплохой поддержкой `IntelliSense` для файлов `C#` и представлений `Razor`, а также содействием в завершении имен и версий пакетов `NuGet` и `Bower`.

Кроме содержимого папки проекта панель проводника показывает, какие файлы в текущий момент редактируются. Это позволяет легко сосредоточиться на подмножестве файлов, с которыми производится работа, что является удобным дополнением, когда приходится иметь дело с поднабором связанных файлов в крупном проекте.

## Добавление в проект пакетов NuGet

Первый шаг связан с добавлением пакетов NuGet, которые содержат сборки .NET, требующиеся для приложений MVC. В панели проводника Visual Studio Code щелкните на имени файла `project.json` и с помощью редактора кода внесите в разделы `dependencies` и `tools` изменения, приведенные в листинге 13.1. Редактор Visual Studio Code выдаст предположения относительно имен и версий пакетов.

### Листинг 13.1. Добавление пакетов NuGet в файл `project.json`

```
...
{
  "dependencies": {
    "Microsoft.NETCore.App": {
      "version": "1.0.0",
      "type": "platform"
    },
    "Microsoft.AspNetCore.Diagnostics": "1.0.0",
    "Microsoft.AspNetCore.Server.IISIntegration": "1.0.0",
    "Microsoft.AspNetCore.Server.Kestrel": "1.0.0",
    "Microsoft.Extensions.Logging.Console": "1.0.0",
    "Microsoft.Extensions.Configuration.EnvironmentVariables": "1.0.0",
    "Microsoft.Extensions.Configuration.FileExtensions": "1.0.0",
    "Microsoft.Extensions.Configuration.Json": "1.0.0",
    "Microsoft.Extensions.Configuration.CommandLine": "1.0.0",
    "Microsoft.AspNetCore.Mvc": "1.0.0",
    "Microsoft.AspNetCore.StaticFiles": "1.0.0"
  },
  "tools": {
    "Microsoft.DotNet.Watcher.Tools": "1.0.0-preview2-final",
    "Microsoft.AspNetCore.Server.IISIntegration.Tools": "1.0.0-preview2-final"
  },
  ...
}
```

Пакеты в разделе `dependencies` добавляют поддержку для MVC и доставки статического содержимого, такого как файлы JavaScript и CSS. Пакет в разделе `tools` делает возможной итеративную разработку в Visual Studio Code, которая настраивается в разделе «Построение и запуск проекта» далее в главе.

Сохраните изменения в файле `project.json` и в окне командной строки/терминала выполните следующую команду, находясь внутри папки `PartyInvites`:

```
dotnet restore
```

Эта команда обрабатывает файл `project.json` и загружает указанные в нем пакеты NuGet. (Иногда редактор Visual Studio Code будет определять, что есть пакеты, подлежащие загрузке, и предлагать выполнить такую команду, но на момент написания главы это средство было ненадежным, т.к. не все изменения обнаруживались. Явный запуск команды гарантирует, что приложение может быть скомпилировано. Перед запуском команды `restore` может понадобиться закрыть файл `project.json` в Visual Studio Code.)

## Добавление в проект пакетов клиентской стороны

Как и в Visual Studio 2015, при управлении пакетами клиентской стороны в проектах Visual Studio Code применяется инструмент Bower, хотя для этого требуется дополнительная работа.

Первым делом необходимо добавить файл по имени `.bowerrc`, который используется для того, чтобы указать Bower, где устанавливать пакеты. Наведите курсор мыши на элемент `PARTYINVITES` в панели проводника и щелкните на значке `New File` (Новый файл), как показано на рис. 13.7.

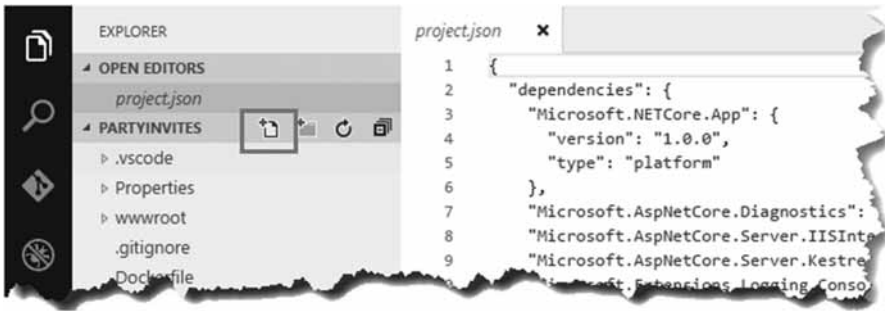


Рис. 13.7. Создание нового файла

Установите имя файла в `.bowerrc` (обратите внимание на наличие в имени двух букв `r`) и поместите в него содержимое, приведенное в листинге 13.2.

### Листинг 13.2. Содержимое файла `.bowerrc`

```
{
  "directory": "wwwroot/lib"
}
```

Далее создайте файл по имени `bower.json` с содержимым, представленным в листинге 13.3.

### Листинг 13.3. Содержимое файла `bower.json`

```
{
  "name": "PartyInvites",
  "private": true,
  "dependencies": {
    "bootstrap": "3.3.6",
    "jquery": "2.2.3",
    "jquery-validation": "1.15.0",
    "jquery-validation-unobtrusive": "3.2.6"
  }
}
```

При добавлении пакетов в файл `project.json` или `bower.json` редактор Visual Studio Code обеспечивает поддержку IntelliSense, что облегчает выбор нужных пакетов и указание применяемых версий.

Воспользуйтесь окном командной строки/терминала, чтобы выполнить в папке PartyInvites показанную ниже команду, которая применяет инструмент Bower для загрузки и установки пакетов клиентской стороны, указанных в файле `bower.json`:

```
bower install
```

## Конфигурирование приложения

Процесс инициализации проекта создал пустой проект без поддержки MVC. В листинге 13.4 приведены изменения, которые вносятся в файл `Startup.cs` для настройки MVC с использованием наиболее базовой конфигурации. Операторы в листинге применяют пакеты, которые были добавлены к проекту в листинге 13.1 и описаны в главе 14.

### Листинг 13.4. Добавление поддержки MVC в файле `Startup.cs`

---

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;

namespace PartyInvites {
    public class Startup {
        public void ConfigureServices(IServiceCollection services) {
            services.AddMvc();
        }

        public void Configure(IApplicationBuilder app,
            IHostingEnvironment env, ILoggerFactory loggerFactory) {
            app.UseStatusCodePages();
            app.UseDeveloperExceptionPage();
            app.UseStaticFiles();
            app.UseMvcWithDefaultRoute();
        }
    }
}
```

---

## Построение и запуск проекта

Чтобы построить и запустить проект, в окне командной строки/терминала перейдите в каталог PartyInvites и выполните следующую команду:

```
dotnet watch run
```

Редактор Visual Studio Code скомпилирует код в проекте и с помощью сервера приложений Kestrel, рассматриваемого в главе 14, запустит приложение, ожидая HTTP-запросы на порте 5000. Любые изменения в файлах C# будут инициировать автоматическую перекомпиляцию. (Если вы хотите запустить проект, игнорируя любые изменения, тогда используйте команду `dotnet run`.)



Редактор Visual Studio Code не предоставляет аналога для средства Browser Link и не открывает окно браузера автоматически. Чтобы протестировать приложение, откройте окно браузера и перейдите на URL вида `http://localhost:5000`. Вы получите ответ, представленный на рис. 13.8. Ошибка 404 связана с тем, что в текущий момент в проекте отсутствуют какие-либо контроллеры для обработки запросов.

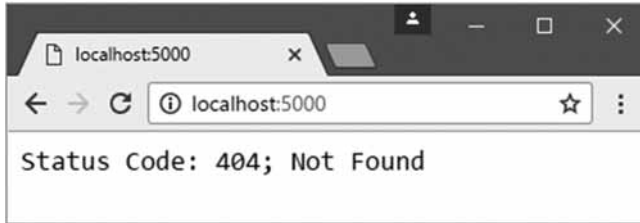


Рис. 13.8. Тестирование примера приложения

## Воссоздание приложения PartyInvites

Все подготовительные шаги завершены, а это значит, что мы можем заняться созданием приложения MVC. Мы воссоздадим простое приложение PartyInvites из главы 2, но с рядом изменений и дополнений, которые подчеркнут особенности работы с Visual Studio Code.

### Создание модели и хранилища

Первым делом наведите курсор мыши на элемент PARTYINVITES в панели проводника и щелкните на значке New Folder (Новая папка), как показано на рис. 13.9. В качестве имени папки укажите Models.

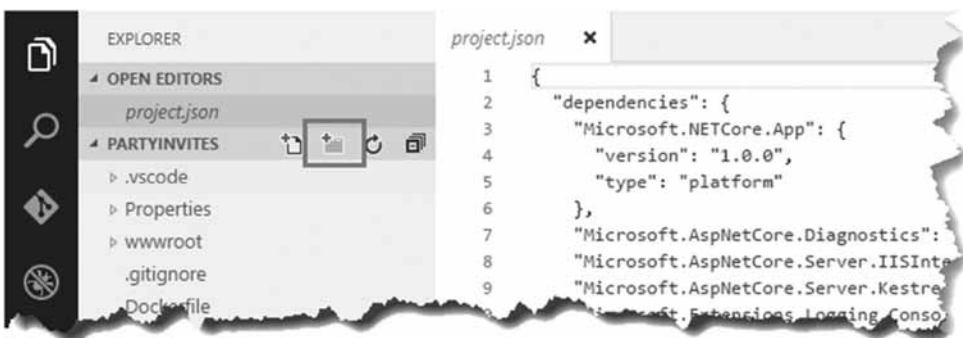


Рис. 13.9. Создание новой папки

Щелкните правой кнопкой мыши на папке Models в панели проводника, выберите в контекстном меню пункт New File (Новый файл), установите имя файла в `GuestResponse.cs` и поместите в файл код C# из листинга 13.5.

---

## Работа с редактором Visual Studio Code

---

Продукт Visual Studio Code (и расширение C#, установленное ранее в главе) предоставляет полнофункциональные средства для редактирования файлов C# и Razor, а также для файлов распространенных веб-форматов, подобных JavaScript, CSS и HTML. В принципе написание приложения MVC в Visual Studio Code имеет много общего с этим же процессом в редакторе Visual Studio 2015: имеется поддержка IntelliSense, кодирование цветом и подсветка ошибок (с советами по их исправлению).

Основной недочетом Visual Studio Code является отсутствие возможности настройки, особенно когда речь идет о форматировании кода. На момент написания главы были доступны варианты конфигурации для других языков, но расширение C# не допускает настройки, что несколько затрудняет работу, если предпочитаемый вами стиль написания кода не совпадает со стилем, предлагаемым по умолчанию. Однако в целом редактор отличается быстрой реакцией и легкостью в применении, так что написание приложений MVC в среде OS X/macOS или Linux не выглядит второсортным процессом.

---

### Листинг 13.5. Содержимое файла `GuestResponse.cs` из папки `Models`

---

```
using System.ComponentModel.DataAnnotations;
namespace PartyInvites.Models {
    public class GuestResponse {
        public int id { get; set; }
        [Required(ErrorMessage = "Please enter your name")]
        // Пожалуйста, введите свое имя
        public string Name { get; set; }
        [Required(ErrorMessage = "Please enter your email address")]
        // Пожалуйста, введите свой адрес электронной почты
        [RegularExpression(".+\\@.+\\.\\..+",
            ErrorMessage = "Please enter a valid email address")]
        // Пожалуйста, введите допустимый адрес электронной почты
        public string Email { get; set; }
        [Required(ErrorMessage = "Please enter your phone number")]
        // Пожалуйста, введите свой номер телефона
        public string Phone { get; set; }
        [Required(ErrorMessage = "Please specify whether you'll attend")]
        // Пожалуйста, укажите, примете ли участие
        public bool? WillAttend { get; set; }
    }
}
```

---

Добавьте в папку `Models` файл по имени `IRepository.cs` с определением интерфейса, приведенным в листинге 13.6. Самое важное отличие приложения в настоящей главе от приложения из главы 2 связано с тем, что мы собираемся хранить данные модели в постоянной базе данных. Интерфейс `IRepository` описывает, каким образом приложение будет получать доступ к данным модели, не указывая реализацию.

Добавьте в папку `Models` файл по имени `ApplicationDbContext.cs` и определите в нем класс контекста базы данных, как показано в листинге 13.7.

**Листинг 13.6. Содержимое файла IRepository.cs из папки Models**

---

```
using System.Collections.Generic;
namespace PartyInvites.Models {
    public interface IRepository {
        IEnumerable<GuestResponse> Responses {get; }
        void AddResponse(GuestResponse response);
    }
}
```

---

**Листинг 13.7. Содержимое файла ApplicationDbContext.cs из папки Models**

---

```
using Microsoft.EntityFrameworkCore;
namespace PartyInvites.Models {
    public class ApplicationDbContext : DbContext {
        public ApplicationDbContext() {}
        protected override void OnConfiguring(DbContextOptionsBuilder builder) {
            builder.UseSqlite("Filename=./PartyInvites.db");
        }
        public DbSet<GuestResponse> Invites {get; set;}
    }
}
```

---

Средство SQLite хранит данные в файле, который указывается классом контекста. В создаваемом примере приложения данные будут храниться в файле по имени `PartyInvites.db`, что определено в методе `OnConfiguring()`.

Чтобы завершить набор классов, необходимых для сохранения и доступа к данным модели, требуется реализация интерфейса `IRepository`, которая использует класс контекста базы данных. Добавьте в папку `Models` файл по имени `EFRepository.cs` и поместите в него код, представленный в листинге 13.8.

**Листинг 13.8. Содержимое файла EFRepository.cs из папки Models**

---

```
using System.Collections.Generic;
namespace PartyInvites.Models {
    public class EFRepository : IRepository {
        private ApplicationDbContext context = new ApplicationDbContext();
        public IEnumerable<GuestResponse> Responses => context.Invites;
        public void AddResponse(GuestResponse response) {
            context.Invites.Add(response);
            context.SaveChanges();
        }
    }
}
```

---

Класс `EFRepository` следует шаблону, который похож на тот, что применялся в главе 8 для настройки базы данных `SportsStore`. В листинге 13.9 видно, что к методу

`ConfigureServices()` класса `Startup` добавлен оператор конфигурирования, который сообщает инфраструктуре ASP.NET Core о необходимости создания экземпляра класса `EFRepository`, когда требуются реализации интерфейса `IRepository`, с помощью средства внедрения зависимостей (рассматриваемого в главе 18).

---

### Листинг 13.9. Конфигурирование хранилища в файле `Startup.cs`

---

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using PartyInvites.Models;

namespace PartyInvites {
    public class Startup {
        public void ConfigureServices(IServiceCollection services) {
            services.AddTransient<IRepository, EFRepository>();
            services.AddMvc();
        }

        public void Configure(IApplicationBuilder app,
            IHostingEnvironment env, ILoggerFactory loggerFactory) {
            app.UseStatusCodePages();
            app.UseDeveloperExceptionPage();
            app.UseStaticFiles();
            app.UseMvcWithDefaultRoute();
        }
    }
}
```

---

## Создание базы данных

В остальных главах книги всякий раз, когда нужно продемонстрировать функциональность, требующую постоянства данных, используется средство LocalDB, которое представляет собой упрощенную версию Microsoft SQL Server. Но средство LocalDB доступно только в среде Windows, поэтому при создании приложений ASP.NET Core MVC на других платформах понадобится какая-то альтернатива. Наилучшей альтернативой LocalDB является SQLite — не нуждающаяся в конфигурировании межплатформенная система управления базами данных, которая может встраиваться в приложения и для которой компания Microsoft включила поддержку в Entity Framework Core. В последующих разделах мы рассмотрим процесс добавления SQLite в проект и ее применение в качестве хранилища данных для ответов на приглашения поучаствовать в вечеринке.

---

## Использование SQLite при разработке

---

Одна из причин того, что LocalDB является настолько полезным инструментом, связана с возможностью разработки с применением механизма баз данных SQL Server, который делает переход в производственную среду SQL Server простым и почти полностью лишенным рисков. SQLite — великолепная база данных, но она не очень хорошо подходит для крупномасштабных веб-приложений, а значит, что при развертывании приложения MVC требуется переход на другую базу данных. Изменения в конфигурации могут быть упрощены с использованием средств конфигурирования, которые будут описаны в главе 14, но приложение должно быть тщательно протестировано в испытательной среде, чтобы выявить любые отличия, вносимые производственной базой данных.

Почитайте статью по адресу <https://www.sqlite.org/whentouse.html>, если вы не уверены в том, применять ли SQLite в производственной среде. Там приведен обзор ситуаций, когда база данных SQLite может быть хорошим вариантом, а когда нет.

Важно иметь в виду тот факт, что SQLite не поддерживает полный набор изменений схемы, которые Entity Framework Core может генерировать для других баз данных. В целом это не проблема, когда SQLite используется при разработке, поскольку вы можете удалить файл базы данных и сгенерировать новый файл с чистой схемой. Тем не менее, ситуация усложняется, если вы обдумываете развертывание приложения, в котором применяется SQLite.

Если вы хотите использовать ту же самую базу данных в среде разработки и производственной среде, тогда ознакомьтесь со списком поддерживаемых инфраструктурой Entity Framework Core баз данных по адресу <http://ef.readthedocs.io/en/latest/providers/index.html>. На момент написания главы список был коротким, но в Microsoft объявили о поддержке баз данных, которые больше, чем SQLite, подходят для процесса разработки и могут функционировать также на платформах, отличных от Windows.

---

### Добавление пакетов базы данных

Первый шаг для любого нового средства в проекте ASP.NET Core предусматривает добавление обязательных пакетов к файлу `project.json`, и в Visual Studio Code оно ничем не отличается. В листинге 13.10 показаны добавления к файлу `project.json` для инфраструктуры Entity Framework Core и ее поддержки SQLite.

#### Листинг 13.10. Добавление пакетов базы данных в файле `project.json`

---

```
...
"dependencies": {
  "Microsoft.NETCore.App": {
    "version": "1.0.0",
    "type": "platform"
  },
  "Microsoft.AspNetCore.Diagnostics": "1.0.0",
  "Microsoft.AspNetCore.Server.IISIntegration": "1.0.0",
  "Microsoft.AspNetCore.Server.Kestrel": "1.0.0",
  "Microsoft.Extensions.Logging.Console": "1.0.0",
  "Microsoft.Extensions.Configuration.EnvironmentVariables": "1.0.0",
  "Microsoft.Extensions.Configuration.FileExtensions": "1.0.0",
  "Microsoft.Extensions.Configuration.Json": "1.0.0",
  "Microsoft.Extensions.Configuration.CommandLine": "1.0.0",
  "Microsoft.AspNetCore.Mvc": "1.0.0",
  "Microsoft.AspNetCore.StaticFiles": "1.0.0",
```

```

"Microsoft.EntityFrameworkCore.Sqlite": "1.0.0",
"Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"
},
"tools": {
  "Microsoft.DotNet.Watcher.Tools": "1.0.0-preview2-final",
  "Microsoft.AspNetCore.Server.IISIntegration.Tools": "1.0.0-preview2-final",
  "Microsoft.EntityFrameworkCore.Tools": {
    "version": "1.0.0-preview2-final",
    "imports": [ "portable-net45+win8+dnxcore50", "portable-net45+win8" ]
  }
},
...

```

Сохраните изменения в файле `project.json`, откройте новое окно командной строки/терминала и выполните следующую команду в папке `PartyInvites`:

```
dotnet restore
```

### Создание и применение миграции базы данных

Создание базы данных следует похожему процессу в смысле команд, используемых средой Visual Studio 2015, хотя выполняется с применением инструмента `dotnet`. Чтобы создать начальную миграцию базы данных, выполните показанную ниже команду, находясь в папке `PartyInvites`:

```
dotnet ef migrations add Initial
```

Инфраструктура Entity Framework Core создаст папку по имени `Migrations`, содержащую файлы классов C#, которые будут использоваться для настройки схемы базы данных. Для применения этой миграции базы данных запустите в папке `PartyInvites` приведенную далее команду, которая создаст базу данных в папке `bin/Debug/netcoreapp1.0`:

```
dotnet ef database update
```

Редактор Visual Studio Code не включает поддержку для инспектирования баз данных SQLite, но на веб-сайте <http://sqlitebrowser.org> можно найти великолепный инструмент с открытым кодом для Windows, OS X/macOS и Linux.

### Создание контроллеров и представлений

В этом разделе мы добавим в приложение контроллер и представления. Начните с создания папки `Controllers` и добавьте в нее файл класса по имени `HomeController.cs` с определением из листинга 13.11.

---

**Совет.** Создание папки в редакторе Visual Studio Code может вызвать затруднения, т.к. щелчок на элементе `PARTYINVITES` в панели проводника скрывает содержимое папки, а не выбирает корневую папку. Щелкните на одном из файлов в корневой папке, таком как `project.json`, наведите курсор мыши поверх элемента `PARTYINVITES` и щелкните на значке `New Folder`.

---

**Листинг 13.11. Содержимое файла HomeController.cs из папки Controllers**

---

```

using System;
using Microsoft.AspNetCore.Mvc;
using PartyInvites.Models;
using System.Linq;

namespace PartyInvites.Controllers {
    public class HomeController : Controller {
        private IRepository repository;
        public HomeController(IRepository repo) {
            this.repository = repo;
        }

        public IActionResult Index() {
            int hour = DateTime.Now.Hour;
            ViewBag.Greeting = hour < 12 ? "Good Morning" : "Good Afternoon";
            return View("MyView");
        }

        [HttpGet]
        public IActionResult RsvpForm() {
            return View();
        }

        [HttpPost]
        public IActionResult RsvpForm(GuestResponse guestResponse) {
            if (ModelState.IsValid) {
                repository.AddResponse(guestResponse);
                return View("Thanks", guestResponse);
            } else {
                // Имеется ошибка проверки достоверности
                return View();
            }
        }

        public IActionResult ListResponses() {
            return View(repository.Responses.Where(r => r.WillAttend == true));
        }
    }
}

```

---

Чтобы установить встроенные дескрипторные вспомогательные классы, создайте папку Views и добавьте в нее файл по имени `_ViewImports.cshtml`, который содержит выражение, показанное в листинге 13.12.

**Листинг 13.12. Содержимое файла \_ViewImports.cshtml из папки Views**

---

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

---

Далее создайте папку Views/Home и добавьте в нее файл по имени `MyView.cshtml`, в котором определяется представление, выбираемое методом действия `Index()` из листинга 13.11. Поместите в него разметку, приведенную в листинге 13.13.

**Листинг 13.13. Содержимое файла MyView.cshtml из папки Views/Home**

---

```

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

```

---

Добавьте в папку Views/Home файл по имени RsvpForm.cshtml с содержимым, показанным в листинге 13.14. Это представление предлагает HTML-форму, которая будет заполняться пользователем, чтобы принять или отклонить приглашение на вечеринку.

**Листинг 13.14. Содержимое файла RsvpForm.cshtml из папки Views/Home**

---

```

@model PartyInvites.Models.GuestResponse
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>RsvpForm</title>
    <link rel="stylesheet" href="/lib/bootstrap/dist/css/bootstrap.css" />
</head>
<body>
    <div class="panel panel-success">
        <div class="panel-heading text-center"><h4>RSVP</h4></div>
        <div class="panel-body">
            <form class="p-a-1" asp-action="RsvpForm" method="post">
                <div asp-validation-summary="All"></div>
                <div class="form-group">
                    <label asp-for="Name">Your name:</label>
                    <input class="form-control" asp-for="Name" />
                </div>
                <div class="form-group">
                    <label asp-for="Email">Your email:</label>
                    <input class="form-control" asp-for="Email" />
                </div>
            </form>
        </div>
    </div>

```



```

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

```

Следующий файл представления называется `Thanks.cshtml`, также создается в папке `Views/Home` и имеет содержимое, приведенное в листинге 13.15, которое отображается, когда гость отправляет свой ответ.

### Листинг 13.15. Содержимое файла `Thanks.cshtml` из папки `Views/Home`

```

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

```

Финальный файл представления имеет имя `ListResponses.cshtml` и подобно другим представлениям в примере добавляется в папку `Views/Home`. Он отображает список ответов гостей, используя разметку из листинга 13.16.

---

### Листинг 13.16. Содержимое файла `ListResponses.cshtml` из папки `Views/Home`

---

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

---

Запущенная ранее в главе команда `dotnet watch` гарантирует компиляцию приложения всякий раз, когда редактируется какой-либо класс C#, и вы можете просмотреть завершенное приложение, перейдя на URL вида `http://localhost:5000` (рис. 13.10).

## Модульное тестирование в Visual Studio Code

Редактор Visual Studio Code не поддерживает отдельные проекты модульного тестирования, а это значит, что модульные тесты должны смешиваться с классами приложения MVC и конфигурироваться с применением того же самого файла `project.json`, с помощью которого настраиваются пакеты и инструменты ASP.NET. В последующих разделах мы добавим к приложению пакет тестирования xUnit, создадим простой модульный тест и прогоним его.

### Конфигурирование приложения

Первый шаг предусматривает добавление пакетов в файл `project.json` и указание деталей об используемом пакете тестирования (листинг 13.17).

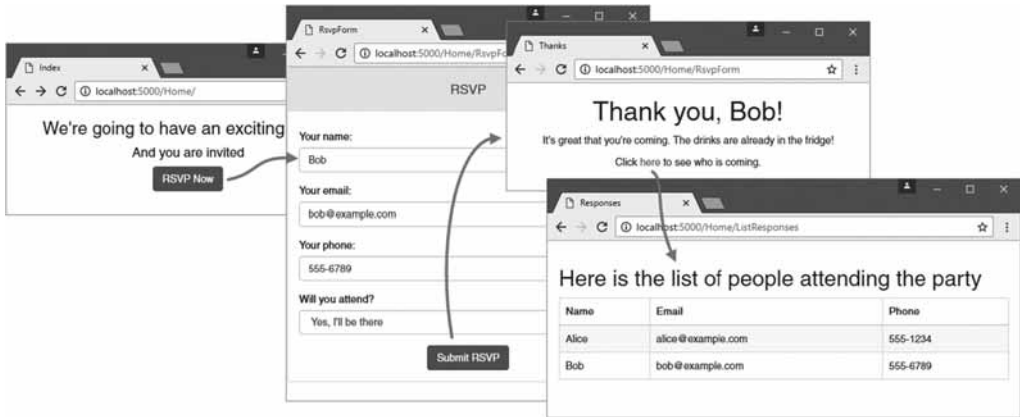


Рис. 13.10. Выполнение завершено приложения

**Листинг 13.17. Конфигурирование модульного тестирования в файле project.json**

```
{
  "dependencies": {
    "Microsoft.NETCore.App": {
      "version": "1.0.0",
      "type": "platform"
    },
    "Microsoft.AspNetCore.Diagnostics": "1.0.0",
    "Microsoft.AspNetCore.Server.IISIntegration": "1.0.0",
    "Microsoft.AspNetCore.Server.Kestrel": "1.0.0",
    "Microsoft.Extensions.Logging.Console": "1.0.0",
    "Microsoft.Extensions.Configuration.EnvironmentVariables": "1.0.0",
    "Microsoft.Extensions.Configuration.FileExtensions": "1.0.0",
    "Microsoft.Extensions.Configuration.Json": "1.0.0",
    "Microsoft.Extensions.Configuration.CommandLine": "1.0.0",
    "Microsoft.AspNetCore.Mvc": "1.0.0",
    "Microsoft.AspNetCore.StaticFiles": "1.0.0",
    "Microsoft.EntityFrameworkCore.Sqlite": "1.0.0",
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
    "xunit": "2.1.0",
    "dotnet-test-xunit": "2.2.0-preview2-build1029"
  },
  "testRunner": "xunit",
  "tools": {
    "Microsoft.DotNet.Watcher.Tools": "1.0.0-preview2-final",
    "Microsoft.AspNetCore.Server.IISIntegration.Tools": "1.0.0-preview2-final",
    "Microsoft.EntityFrameworkCore.Tools": {
      "version": "1.0.0-preview2-final",
      "imports": [ "portable-net45+win8+dnxcore50", "portable-net45+win8" ]
    }
  },
  // ...для краткости остальные разделы не показаны...
}
```

Выполните в папке `PartyInvites` следующую команду для установки пакетов тестирования:

```
dotnet restore
```

## Создание модульного теста

Модульные тесты создаются так, как было описано в главе 7, но тестовые классы должны быть частью проекта приложения. Создайте папку `Tests` и поместите в нее файл класса по имени `HomeControllerTests.cs` с содержимым, приведенным в листинге 13.18.

### Листинг 13.18. Содержимое файла `HomeControllerTests.cs` из папки `Tests`

---

```
using System;
using System.Collections.Generic;
using PartyInvites.Controllers;
using PartyInvites.Models;
using Xunit;
using Microsoft.AspNetCore.Mvc;
using System.Linq;

namespace PartyInvites.Tests {
    public class HomeControllerTests {
        [Fact]
        public void ListActionFiltersNonAttendees () {
            // Организация
            HomeController controller = new HomeController(new FakeRepository());
            // Действие
            ViewResult result = controller.ListResponses ();
            // Утверждение
            Assert.Equal(2, (result.Model as IEnumerable<GuestResponse>).Count());
        }
    }
}

class FakeRepository : IRepository {
    public IEnumerable<GuestResponse> Responses =>
        new List<GuestResponse> {
            new GuestResponse { Name = "Bob", WillAttend = true },
            new GuestResponse { Name = "Alice", WillAttend = true },
            new GuestResponse { Name = "Joe", WillAttend = false }
        };
    public void AddResponse(GuestResponse response) {
        throw new NotImplementedException();
    }
}
}
```

---

Это стандартный тест `xUnit`, который проверяет действие `ListResponses` в контроллере `Home` и фильтрует в хранилище объекты `GuestResponse` со значением свойства `WillAttend`, равным `false`.

## Прогон тестов

Редактор Visual Studio Code обнаруживает тесты и добавляет встроенную ссылку для их запуска (рис. 13.11).

```
[Fact]
0 references | run test | debug test
public void ListActionFiltersNonAttendees() {
    //Arrange
    HomeController controller = new HomeController(new FakeRepository());
    // Act
    ViewResult result = controller.ListResponses();
    // Assert
    Assert.Equal(2, (result.Model as IEnumerable<GuestResponse>).Count());
}
```

Рис. 13.11. Прогон теста внутри редактора кода

Щелчок на ссылке `run test` (прогнать тест) приведет к открытию окна вывода и отображению результата. (На момент написания главы ссылка `debug test` (отладить тест) не работала, а в ряде случаев вывод мог вообще не отображаться.)

Более надежный подход предусматривает прогон всех тестов в проекте. Выполните следующую команду в папке проекта:

```
dotnet test
```

Все тесты в проекте запустятся, и в результате отобразится вывод, подобный показанному ниже:

```
xUnit.net .NET CLI test runner (64-bit .NET Core win10-x64)
  Discovering: PartyInvites
  Discovered:  PartyInvites
  Starting:    PartyInvites
  Finished:   PartyInvites
=== TEST EXECUTION SUMMARY ===
  PartyInvites Total: 1, Errors: 0, Failed: 0, Skipped: 0, Time: 0.196s
SUMMARY: Total: 1 targets, Passed: 1, Failed: 0.
```

Прогнать все модульные тесты в проекте после каждого изменения классов C# можно также с помощью следующей команды:

```
dotnet watch test
```

Такая команда не может применяться одновременно с командой `dotnet watch run`, поскольку обе команды будут инициировать компиляцию проекта при наличии изменений и пытаться создавать те же самые выходные файлы.

## Резюме

В этой главе был предложен краткий обзор работы с редактором Visual Studio Code — легковесным инструментом разработки, который поддерживает создание приложений ASP.NET Core MVC в средах Windows, OS X/macOS и Linux. Редактор Visual Studio Code пока еще не является заменой полного продукта Visual Studio, но предоставляет основные средства, которые необходимы при построении приложений MVC, и расширяется компанией Microsoft в ежемесячных выпусках.

Итак, первая часть книги завершена. Во второй части мы начнем погружение в детали и посмотрим, как работают средства, которые использовались для создания приложения.