ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ ВЕБ-ТЕХНОЛОГИЙ В СИСТЕМАХ ПРИНЯТИЯ РЕШЕНИЙ

Екатеринбург
2022

Министерство науки и высшего образования Российской Федерации
Уральский государственный экономический университет

ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ ВЕБ-ТЕХНОЛОГИЙ В СИСТЕМАХ ПРИНЯТИЯ РЕШЕНИЙ

Рекомендовано
Советом по учебно-методическим вопросам и качеству образования
Уральского государственного экономического университета
в качестве учебного пособия

Екатеринбург
2022

УДК 004.9(075.8)
ББК 32.973.202-018.2я73
П42

Рецензенты:

Ученый совет Института радиоэлектроники и информационных технологий — РТФ Уральского федерального университета имени первого Президента России Б. Н. Ельцина (протокол № 1 от 1 июня 2022 г.)

генеральный директор ООО «Октоника»
К. Г. Ведьманов

Авторский коллектив:
В. П. Часовских, *В. Г. Лабунец*, *Е. Н. Стариков*,
Е. В. Кох, М. П. Воронов

П42 **Повышение эффективности веб-технологий в системах принятия решений** : учебное пособие / В. П. Часовских, В. Г. Лабунец, Е. Н. Стариков [и др.] ; Министерство науки и высшего образования Российской Федерации, Уральский государственный экономический университет. — Екатеринбург : УрГЭУ, 2022. — 128 с.

Цель учебного пособия — формирование теоретических и практических знаний о современных информационных веб-технологиях в системах принятия решений. Представлены общетеоретические аспекты формализации и использования различных моделей (от традиционных до новейших интеллектуальных систем) создания более экономных, оптимизированных под облачные технологии и адаптируемых к функционированию на мобильных устройствах веб-приложений для платформы .NET. Структура и содержание учебного пособия полностью соответствует требованиям федеральных государственных образовательных стандартов высшего образования и учебной программе дисциплины «Современные веб-технологии».

Для студентов очной и заочной форм обучения направлений бакалавриата 02.03.03 «Математическое обеспечение и администрирование информационных систем», 09.03.03 «Прикладная информатика», 09.03.01 «Информатика и вычислительная техника» и магистратуры 38.04.05 «Бизнес-информатика», 09.04.03 «Прикладная информатика».

УДК 004.9(075.8)
ББК 32.973.202-018.2я73

Введение

Мы живем в эпоху новых технологий — технологий четвертой промышленной революции. Во многом эти технологии взаимосвязаны: в том, как они расширяют цифровые возможности; в том, как масштабируются, развиваются, встраиваются в нашу жизнь; в том, как взаимно дополняют друг друга, а также в их способности концентрировать привилегии и бросать вызов существующим системам принятия решений. Особое место занимают веб-технологии в системах принятия управленческих решений.

Принятие управленческого решения, как правило, направлено на устранение не одной, а сразу нескольких видимых и (или) скрытых проблем в организации. В первую очередь важным представляется выбор метода принятия управленческого решения. На сегодняшний день это системно организованная совокупность методов и средств сбора, обработки, хранения, поиска, передачи и защиты данных, информации и знаний для решения задач управления на базе развитого программного обеспечения и средств вычислительной и телекоммуникационной техники. И комплексно это представлено именно в веб-технологиях.

Цель данного пособия — научить создавать более экономные, оптимизированные под облачные технологии и адаптируемые к функционированию на мобильных устройствах веб-приложения для платформы .NET для систем принятия решений. Будут изучены веб-технологии, позволяющие самостоятельно создавать веб-сайты среднего уровня сложности для самой последней ступени развития веб-платформы ASP.NET.

В этом учебном пособии помимо рассмотрения привычных презентаций будет практиковаться работа с кодом в интерактивных упражнениях, а закрепление навыков осуществляться в реальных проектах. Практическая часть курса построена на созда-

нии реального приложения «Выпускающая кафедра вуза», обеспечивающего понятные онлайн-взаимодействия обучающегося и преподавателя. Кроме того, будет создана функционально полная административная область, включающая средства создания, чтения, обновления, удаления и защищенная так, чтобы изменения могли вносить только администраторы.

В настоящее время управленческая деятельность и принятие решений сталкиваются с огромным количеством информации, что получило название big data. Веб-технологии позволяют преодолевать это препятствие.

Учебное пособие включает общетеоретические аспекты формализации и использования различных моделей данных, основных методов обработки больших наборов данных и функциональных возможностей веб-технологий. В учебном курсе студенты знакомятся с появлением и историей развития Active Server Pages (ASP) до ASP.NET Core, а также со связанным с ней языком программирования C#. Приведена иллюстрация применения многочисленных API-интерфейсов .NET, в числе которых доступ к базам данных с помощью ADO.NET и Entity Framework (EF), пользовательские интерфейсы, построенные посредством Windows Presentation Foundation (WPF), ориентированные на службы приложения, созданные с помощью Windows Communication Foundation (WCF), а также веб-службы и веб-сайты, реализованные посредством ASP.NET MVC и ASP.NET Core MVC 2.

В пособии рассмотрены общие принципы и подходы использования в ASP.NET Core MVC 2 платформы Ruby on Rails, предложившей революционную технологию применения архитектуры MVC совместно с протоколом HTTP. Рассматривается использование JavaScript в качестве основного языка программирования. Технология AJAX первой продемонстрировала важность JavaScript, технология jQuery показала, что этот язык может быть мощным и изящным, а JavaScript — механизм с открытым кодом V8 от Google. В настоящее время в ASP.NET Core MVC 2 язык программирования JavaScript стал серьезным средством серверной стороны и применяется в качестве универсального языка в серверной платформе Node.js, завоевавшей широкое признание, поскольку чрезвычайно эффективно использует системные ресурсы и позволяет обрабатывать десятки тысяч одновре-

менных запросов на одном центральном процессоре. Акцентируется внимание на том, что Angular обеспечивает удобную отправную точку для приложений ASP.NET Core с использованием Angular и Angular CLI для реализации многофункционального пользовательского интерфейса (UI) на стороне клиента.

Особенностью учебного пособия является системное последовательное представление моделей данных веб-технологий в разрезе их эволюции. Проводится сравнительный анализ веб-технологий и моделей данных для задач принятия решений. Существенную часть занимают вопросы практической реализации систем принятия решений в среде современных веб-технологий.

Структура учебного пособия включает основной теоретический материал и обобщенные выводы по нему в конце каждой главы. С целью повышения эффективности освоения материала по каждой главе сформулированы вопросы для самоконтроля. Авторы пособия рекомендуют после изучения теории для закрепления пройденного материала формулировать ответы на предложенные вопросы.

Пособие соответствует учебной программе дисциплины «Современные веб-технологии», изучаемой студентами направления 02.03.03 «Математическое обеспечение и администрирование информационных систем» (бакалавриат), очной и заочной форм обучения. Также может использоваться студентами бакалавриата направлений 09.03.03 «Прикладная информатика», 09.03.01 «Информатика и вычислительная техника», а также магистратуры 38.04.05 «Бизнес-информатика», 09.04.03 «Прикладная информатика» при освоении дисциплин «Разработка и проектирование информационных систем», «Базы данных», «Технологии обработки больших данных», «Разработка программных приложений».

Глава 1

Веб-технологии и платформа ASP.NET

1.1. Понятие веб-технологии

29 октября 1969 г. из сетевого узла лаборатории вычислительной техники Калифорнийского университета на сетевой узел в Стэнфордском исследовательском институте было отправлено сообщение.

Это день считается появлением современного Интернета — информационно-коммуникационной сети и всемирной системы объединенных компьютерных сетей для хранения и передачи информации.

Появились интернет-технологии — то, что позволяет осуществлять все работы в компьютерной сети Интернет. Прежде всего, многообразие сайтов, плюс чаты, форумы, электронная почта, Интернет вещей, социальные сети и масса всего, что создано для работы в Интернете или с использованием Интернета.

Интернет-технологии создаются определенными методами в соответствии с определенными правилами при помощи специальных технических средств (сетей, серверов и т. п.) и специальных программ.

В среде интернет-технологий стали выделять веб-технологии как их логическую составляющую.

Веб-технологии — это интернет-сервисы (WWW — Всемирная паутина).

Веб-технологии — это работа в Интернете (браузеры, поисковые системы, просмотр страниц в браузере).

Веб-технологии — это информационные ресурсы Интернета (веб-страницы, интернет-магазины, интернет-порталы, URL и протоколы передачи данных, адресация, создание сайтов, языки веб-программирования).

Основные понятия веб-технологий — веб-страница и веб-сайт.

Веб-страница — это минимальная единица сервиса WWW или документ, который является уникальным в WWW своим URL адресом.

Веб-сайт — это набор веб-страниц, связанных общей тематикой. Он находится на одном сервере (хостинге) и принадлежит одному владельцу. Может состоять из одной веб-страницы — сайт-визитка.

Основная цель в изучении веб-технологий — создание или изменение веб-страниц, которые будут правильно отражаться в браузерах.

В настоящее время существует множество платформ для разработки веб-приложений. Наибольший интерес представляет платформа ASP.NET, ей и будет посвящен основной материал учебного пособия.

1.2. Платформа ASP.NET

1.2.1. Что такое ASP.NET

Веб-технология ASP.NET (Active Server Pages) — это платформа для создания интерактивных веб-приложений, от простого сайта-визитки до огромных порталов в полном соответствии с веб-стандартами. Она поддерживает работу с несколькими языками программирования, входящими в сборку фреймворка: Basic NET, C# (2.47), J# и ряд прочих.

Профессиональная *ASP.NET технология* является бесплатной, но полностью поддерживаемой средой веб-разработки. Веб-технология построена на базе платформы программирования .NET, что позволяет программистам использовать огромное количество готовых классов при создании веб-проектов.

1.2.2. История ASP.NET

1996 — ASP — Active Server Pages, построение страниц на сервере на основе шаблонов. Шаблоны сочетали код на VB с HTML-разметкой.

2001 — ASP.NET — составная часть новой платформы .NET. Технология WebForms по аналогии с WinForms.

2009 — ASP.NET MVC. Аналогична уже существующим на рынке подходам: Java Spring 2002, Python Jango 2003 и др.

2013 — ASP.NET MVC 5.0 — октябрь, последняя версия 2 ASP.NET WebForms ASP.NET WebFormsMVC ASP.

1.2.3. Фреймворки на базе ASP.NET

ASP.NET — бесплатный фреймворк для построения больших веб-приложений с использованием HTML, CSS и JavaScript.

WebForms — технология построение веб-приложений из стандартных управляющих элементов и обработчиков событий.

ASP.NET MVC — построение веб-приложений на базе шаблона MVC с разделением ответственности и полным контролем над HTML кодом страниц.

Web Pages — быстрая разработка веб-сайтов согласно современным веб-стандартам.

ASP.NET по скорости работы значительно превосходит прочие скриптовые языки. Причина быстрой обработки заключается в том, что основа компилируется при первом подключении пользователя и сразу добавляется в кэш компьютера. Все последующие переходы по сайту используют уже кэшированный код, который просто извлекается из памяти, а не скачивается с сервера повторно. Такой подход сильно экономит время на парсинге, загрузке и обработке файлов.

1.2.4. Комплектация ASP.NET

ASP.NET включает широкий набор фреймворков (рабочих каркасов веб-сайтов): *WebForms*, *MVC*, сотни встроенных элементов управления, *AJAX* на основе *JavaScript* библиотек, таких как *jQuery* и *Microsoft Ajax*, упрощающие работу с *HTML DOM.* ASP.NET структурирует маршрутизацию веб-адресов, позволяя

с легкостью создавать URL понятные человеку и удобные для поисковых систем. Благодаря развитой и расширяемой архитектуре новые возможности в ASP.NET можно добавлять почти до бесконечности, гарантированно поддерживая предыдущие.

1.2.5. Возможности ASP.NET

ASP.NET позволяет создавать сайты сложной логики, при этом сохраняя четкость структуры программного кода и элементов. Мнение от том, что ASP.NET годится только для крупных веб-приложений, неверно. Данное суждение исходит прежде всего из того, что большинство крупных веб-разработок основывается на платформе ASP.NET, позволяющей в принципе создавать, расширять и полноценно поддерживать высокий уровень сложности крупных веб-порталов.

1.2.6. Изучение ASP.NET

В то же время изучение ASP.NET не такое уж простое дело. Если сравнивать PHP и ASP.NET, то первый выигрывает в своей простоте. ASP.NET требует более серьезной подготовки перед работой в ней. В то время как PHP язык веб-программирования, ASP.NET является платформой для программных модулей создания веб-приложений, как простых, так и огромных порталов и веб-служб. Но простой PHP имеет и серьезные недостатки, а по мере расширения веб-проекта простота языка PHP отходит на второй план.

1.2.7. Инструменты программирования для ASP.NET

Одно из главных преимуществ ASP.NET в сравнении с другими языками и платформами создания веб-приложений — это бесплатная доступность полноценных инструментов программирования. Ни одно бесплатное приложение для других веб-технологий не сравнится по возможностям и удобству работы с инструментами для ASP.NET.

Microsoft Visual Studio Community — бесплатная полнофункциональная и расширяемая интегрированная среда разработки для создания современных приложений для Windows,

Android и iOS, а также веб-приложений и облачных служб. Visual Studio Community создана для индивидуальных разработчиков, проектов с открытым кодом, научных исследований, образования и небольших групп специалистов. Программирование на C#, Visual Basic, F#, C++, HTML, JavaScript, Python и других языках.

Visual Studio Code — бесплатный редактор кода, построен на открытом исходном коде, поддерживает множество языков программирования: C#, Razor, PHP, HTML, XML, CSS, JavaScript, Sass, Python, Perl, F#, C++ и др. Отладка кода прямо из редактора, с помощью точек останова, стеков вызовов и интерактивной консоли. Visual Studio Code — расширяемый и настраиваемый. Для увеличения возможностей редактора устанавливаются расширения для добавления новых языков, тем, отладчиков и для подключения к дополнительным сервисам. Расширения выполняются в отдельных процессах, поэтому они не замедляют работу редактора. Visual Studio Code работает в Windows, Linux, Mac.

Microsoft Visual Studio 2019 Professional — коммерческая среда разработки для индивидуальных разработчиков и предприятий. Имеет расширенные возможности отладки, диагностики, тестирования и кроссплатформенной разработки.

1.2.8. Веб-сервер IIS, запуск и настройки

IIS (Internet Information Services) — комплекс служб Интернета от компании Майкрософт. IIS распространяется с операционными системами семейства Windows. Основным компонентом является веб-сервер, который обслуживает веб-сайты, расположенные на локальной машине и в сети Интернет. IIS поддерживает протоколы HTTP, HTTPS, FTP, POP3, SMTP. Позволяет использовать для сайтов программирование в любой веб-системе: ASP.NET, PHP, Python и др., использование любых баз данных, например, MSSQL, MySQL, PostgreSQL, SQLite. Характеризуется малым потреблением ресурсов процессора, является одним из самых быстрых серверов и, что важно, имеет централизованную поддержку компании Microsoft.

1.2.9. Установка IIS

Службы IIS не устанавливаются по умолчанию при установке Windows. Чтобы установить веб-сервер для работы с сайтами ASP.NET, необходимо включить компонент *Windows IIS*. Активация служб IIS:

Панель Управления → Программы → Включение и отключение компонентов Windows (поставить галочки).

В дальнейшем можно будет добавить необходимые настройки.

1.2.10. Дополнительные установки

Для работы веб-сайтов на основе ASP.NET, возможно, потребуется установка .NET Framework, старых и (или) новейших версий. Для веб-приложений .NET Core необходима установка пакета ASP.NET Core Hosting Bundle, включающего в себя среду выполнения .NET Core и среду выполнения ASP.NET Core. Для сервера IIS данный пакет добавит еще модуль ASP.NET Core IIS.

1.2.11. Тестирование сайтов на локальном компьютере

Настройка IIS для тестирования сайтов на локальном компьютере; операционные системы Windows 7, Windows 10. Перед опубликованием сайтов в Интернете очень желательно их тщательно протестировать у себя на компьютере. Для этого необходимо запустить и настроить службу IIS. Для повышения «производительности» выпуска сайтов IIS можно настроить на одновременную работу нескольких сайтов на одном компьютере.

1.3. Виды веб-проектов

Веб-проект ASP.NET можно создавать как *веб-приложение* и как *веб-сайт*. Конечная цель обоих проектов — рабочий сайт, размещенный на сервере. В *веб-приложении* весь программный код, находящийся в различных папках и файлах, компилируется в сборку .dll. При работе с *веб-сайтом* текстовые файлы с программным кодом размещаются непосредственно на сервере

для динамической компиляции во время загрузки веб-страниц. Выбрать ту или иную схему построения помогают теоретические знания и опыт работы с различных типами веб-проектов.

1.3.1. Видимые отличия проектов в Visual Studio

Просмотр содержимого веб-приложения и веб-сайта в MS Visual Studio. В *веб-приложении* файлы, исключенные из проекта, по умолчанию невидимы, и просмотреть их можно, нажав кнопку показа всех файлов, при этом исключенные файлы будут выделены прямоугольником из точек. Данная опция отключения видимости дает возможность сосредоточиться только на рабочих файлах проекта. В проекте *веб-сайта* видны все файлы, исключенные же из проекта файлы маркируются расширением .exclude, но не скрываются. При публикации файлы, исключенные из любого веб-проекта, на сервер не переносятся.

1.3.2. Особенности кодирования веб-приложения и веб-сайта

Работая над *веб-приложением,* программный код можно помещать в любые папки, но нежелательно использовать название для папки App_Code. Эта папка зарезервирована для веб-сайта, и при запуске веб-проекта в Visual Studio возможна двойная компиляция (хотя после публикации на сервер нормальная работа восстанавливается). Программный код *веб-сайта,* напротив, рекомендуется помещать только в папку App_Code. В веб-приложении и в веб-сайте можно использовать вложенность папок любой разумной глубины.

1.3.3. Инструменты программирования веб-приложений

Веб-приложения ASP.NET создаются в MS Visual Studio, MS Visual Studio Code. Все файлы классов с выделенным кодом и отдельные файлы классов в проекте компилируются в единую сборку, которая помещается в папку Bin проекта веб-приложения. Файлы *ASPX, ASCX, CSHTML* публикуются в неизменном виде и компилируются динамически на сервере, подобно функциональности веб-сайта.

1.3.4. Инструменты программирования веб-сайтов

Веб-сайты ASP.NET можно создавать и редактировать в MS Visual Studio, Visual Studio Code и редактировать, используя текстовый редактор типа Блокнот. Компилировать веб-сайт не требуется. Файлы веб-сайтов компилируются автоматически на сервере при запросе веб-страниц. Можно выбрать режим пакетной компиляции, в котором создается одна сборка для каждой папки, или режим фиксированной компиляции, в котором одна сборка для каждой страницы или пользовательского элемента управления. Данная настройка фиксируется в файле конфигурации веб-узла *web.config*.

1.3.5. Рекомендация выбора веб-приложения

Проекты веб-приложений желательно выбирать, когда:

– необходимо избежать размещения открытого исходного кода на рабочем сервере;

– с помощью компилятора требуется создать единую сборку для всего сайта;

– необходимо выполнять модульные тесты кода, находящегося в файлах классов, связанных со страницами ASP.NET;

– требуется ссылаться на классы, связанные со страницами и пользовательскими элементами управления из изолированных классов;

– требуется управление именем и номером версии сборки, созданной для сайта.

1.3.6. Рекомендации в пользу веб-сайта

Проекты веб-сайтов являются предпочтительным вариантом, когда:

– исходные файлы проекта копируются на сервер;

– в один веб-проект необходимо включить как код C#, так и код на другом языке .NET;

– необходимо открывать веб-сайт в Visual Studio, Visual Studio Code, WebMatrix и обновлять его в режиме реального времени;

– требуется возможность обновления отдельных файлов в рабочей среде путем простого копирования новых версий на рабочий сервер;

– требуется создать отдельную сборку для каждой страницы, папки или пользовательского элемента;

– нужно сохранить исходный код на рабочем сервере в качестве дополнительной резервной копии.

1.4. Инструменты работы с Web Forms

Основным инструментом создания и редактирования страниц *Web Forms ASP.NET* является Microsoft Visual Studio с .NET. При работе в Visual Studio веб-элементы управления могут добавляться на страницу вручную или путем перетаскивания из панели инструментов. На страницу автоматически добавляется тег *html form* в единственном экземпляре. Обрамляющая всю страницу html-форма дает возможность пользователям сайта интерактивно работать с интерфейсом веб-страницы. Работая с Web Forms, можно различными способами создавать HTML-разметку элементов управления. Все способы дают визуально одно и то же, но имеют свои особенности.

1.4.1. Среда HTML

Язык разметки гипертекста (Hyper Text Markup Language) является стандартным средством представления информации в среде Word Wide Web (WWW) в виде веб-страниц.

WWW — это часть Интернета, представляющая собой гигантский набор документов, хранящихся на компьютерах по всему миру. Сокращенно WWW называют просто Web (веб). Веб-страница — это отдельный документ Web, который способен содержать информацию различного вида — текст, рисунки, фото, видео, аудиозаписи.

URL (Uniform Resource Locator) — это уникальный адрес, которым обладает каждая веб-страница в сети. Если пользователю известен URL страницы, то он может ее отобразить у себя в браузере, набрав ее адрес в строке браузера. В общем случае

URL состоит из наименования используемого протокола, названия сервера и обозначения пути доступа к странице.

Протокол определяет правила обращения к веб-странице.

Название сервера (доменное имя) обозначает компьютер, содержащий данную веб-страницу.

Путь доступа указывает местоположение страницы на диске сервера.

Например, http://on-line-teaching.com/html/index.html:

http — наименование протокола;

on-line-teaching.com — наименование сервера или доменное имя;

/html/index.html — путь доступа к странице на самом сервере.

Язык HTML в том виде, в котором он существует сейчас, обладает большим потенциалом представления информации с расчетом не только на пользователей персональных компьютеров. Документы HTML можно просматривать на очень большом количестве различных по своим возможностям устройств: от черно-белого экрана мобильного телефона до телетайпа или терминала. Также в последние версии HTML включен ряд возможностей, облегчающих представление документов неграфическими средствами (например, речевыми синтезаторами), позволяющими пользоваться услугами WWW людям с ограниченными физическими возможностями.

Базовые понятия

Web-страница — это документ в формате HTML, содержащий текст и специальные теги (дескрипторы) HTML. По большому счету теги HTML необходимы для форматирования текста (придания ему нужного вида), который «понимает» браузер. Документы HTML хранятся в виде файлов с расширением .htm или .html.

Тег HTML сообщает браузеру информацию о структуре и особенностях форматирования веб-страницы. Каждый тег содержит определенную инструкцию и заключается в угловые скобки < >. Большинство тегов состоят из открывающей и закрывающей частей и воздействуют на текст, заключенный внутри.

Теги бывают одиночными и контейнерными. Контейнером называется пара: открывающий <ТЕГ> и закрывающий </ТЕГ>.

```
<ТЕГ> Контейнер </ТЕГ>
```

Атрибуты тега задают значения свойств данного объекта или объектов, помещенных в контейнер. Значения свойств, содержащие пробелы, берутся в кавычки, в остальных случаях кавычки можно опустить.

1.4.2. Структура HTML-документа

В идеальном случае HTML-документ состоит из трех частей:

– информация о версии используемого HTML;

– заголовок документа;

– тело документа.

Пример простейшего HTML-документа, содержащего все структурные элементы:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML> <HEAD>
<TITLE>HTML-документ</TITLE> </HEAD>
<BODY> <H1>Заголовок</H1> <P>Первый абзац <P>Второй абзац
</BODY> </HTML>
```

Первый тег, который должен находиться в любом HTML-документе, это <HTML> ... </HTML>. Этот тег указывает на то, что данный документ действительно содержит в себе HTML-текст. Все, что вы напишете в своем документе, должно находиться внутри данного тега:

```
<HTML> …
текст документа …
</HTML>
```

Заголовок документа

В заголовке (его еще называют «шапкой») HTML-документа содержатся сведения о документе: название (тема документа), ключевые слова (используются поисковыми машинами), а также ряд других данных, которые не являются содержимым документа.

Определение заголовка должно содержаться внутри тега <HEAD> ...</HEAD>:

```
<HTML> <HEAD> …
описание заголовка …
</HEAD> …
текст документа …
</HTML>
```

В разделе описания заголовка можно указать заглавие документа, для этого используется тег <TITLE> ... </TITLE>. Когда браузер встречает этот тег, он отображает все, что находится внутри него, как заглавие.

Определение тела документа

Весь остальной HTML-документ, включая весь текст, содержится внутри тега <BODY> ...</BODY> (тело). Теперь наш документ выглядит примерно так:

```
<HTML> <HEAD>
<TITLE>Это заглавие документа</TITLE> </HEAD>
<BODY> …
текст документа …
</BODY> </HTML>
```

Наиболее часто используемые атрибуты элемента BODY:

background — URL, указывающий расположение изображения для фона (обычно берется небольшое изображение, которое размножается для заполнения фона всего документа);

bgcolor — цвет фона HTML-документа;

text — цвет шрифта документа;

link — цвет непосещенных гиперссылок;

vlink — цвет посещенных гиперссылок;

alink — цвет гиперссылок при выборе их пользователем (при нажатии Enter произойдет переход по такой ссылке);

contenteditable — позволяет разрешить или запретить пользователю редактирование содержимого HTML-документа при просмотре его браузером (значения true, false, inherit).

Все атрибуты, позволяющие задать цвет, имеют тип %Color. Значения таких атрибутов могут задаваться шестнадцатеричными числами с символом # в начале каждого числа, например:

```
bgcolor = "#FF0005"
```

Также атрибутам задания цвета можно присваивать предопределенные идентификаторы некоторых наиболее часто употребляемых цветов.

Особенности ввода текста

Для того чтобы поместить простой текст на страницу, его достаточно ввести в тело документа. При этом браузер отобразит текст со шрифтом по умолчанию и цветом, заданным для текста тела страницы. Чтобы чтение информации, содержащейся в HTML-документе, стало приятным занятием, применяется форматирование текста.

Существуют специальные команды, выполняющие перевод строки и задающие начало нового абзаца. Кроме того, имеется команда, которая запрещает программе браузера каким-либо образом изменять форматирование текста и позволяет точно воспроизвести на экране заданный фрагмент текстового файла.

Тег перевода строки
 отделяет строку от последующего текста или графики. Тег абзаца <P> тоже отделяет строку, но еще добавляет пустую строку, которая зрительно выделяет абзац. Оба тега являются одноэлементными.

Задание начертания текста является, возможно, самым простым средством форматирования содержимого документа, которое доступно в HTML. Для изменения начертания текста в HTML-код вводятся элементы, обозначающие текст, написанный с соответствующим начертанием.

Задание шрифта текста

Если нужно отобразить некоторый текст с использованием определенного шрифта, а не применяемого браузером по умолчанию, то в HTML предусмотрен элемент FONT. Он вводится при помощи парных тегов <FONT> и </FONT>.

Параметры шрифта для элемента FONT устанавливаются заданием значений следующих его атрибутов:

face — задает название шрифта, например Arial или System;

size — задает размер шрифта (значение от 1 до 7, по умолчанию используется значение 3);

color — задает цвет шрифта.

Для атрибута size могут использоваться только 7 значений.

Списки

Нумерованные списки применяются для упорядочения приводимых данных. При нумерации элементов таких списков могут быть использованы как арабские, так и римские цифры, символы латинского алфавита. Для создания нумерованного списка используется комбинация двух пар дескрипторов. Дескрипторы <OL>…</OL> устанавливают начало и конец нумерованного списка, а дескрипторы <LI>…</LI> отмечают отдельные пункты списка. Тип нумерованного списка устанавливается с помощью атрибута TYPE дескриптора <OL>.

TYPE='1' 1,2,3,4 … (задан по умолчанию)
TYPE='i' i,ii,iii,iv,…
TYPE='I' I,II,III,IV,…
TYPE='a' a,b,c,d,…
TYPE='A' A,B,C,D.

Гиперссылки

Гиперссылки — ключевой, практически самый главный элемент гипертекста (текста, обладающего возможностями навигации). Гиперссылки позволяют быстро переходить к другим веб-страницам, исключая необходимость ввода URL страницы, к которой обращается пользователь. Гиперссылки, как правило, выделяются цветом и подчеркиванием. В качестве гиперссылки может выступать и графическое изображение, в таком случае оно обычно выделяется рамкой.

Простой переход к ресурсам

Для создания гиперссылки в тексте HTML-документа используется элемент A, который задается при помощи парных тегов <A> и </A>. Текст, изображение или другой элемент HTML-документа, заключенный между этими тегами, становится представлением гиперссылки в тексте.

Навигация между HTML-документами

Для создания простейшей гиперссылки, обеспечивающей переход к нужному HTML-документу, достаточно задать в качестве значения атрибута href элемента A адрес URI нужного ре-

сурса (файла HTML-документа). Допустим, что сайт, по которому нужно реализовать навигацию, состоит из пяти страниц. Файлы имеют свои имена. У сайта есть главная страница (index.html), на которой помещено оглавление. Элементы оглавления позволяют перейти к соответствующим страницам, т. е. являются гиперссылками.

Графика на веб-страницах

Для вставки изображения в текст HTML-документа используется элемент IMG. Для задания HTML-элемента применяется одиночный тег <IMG>. Список наиболее используемых атрибутов:

src — задает URL изображения;

alt — альтернативный текст, который отображается на месте изображения, если по каким-либо причинам само изображение не может быть показано;

border — задает толщину границы вокруг рисунка в пикселях;

align — определяет выравнивание изображения;

height — задает высоту изображения в пикселях;

width — определяет ширину изображения в пикселях;

vspace — задает величину свободного пространства между изображением и текстом сверху и снизу;

hspace — задает величину свободного пространства между изображением и текстом справа и слева;

name — позволяет идентифицировать изображение так, что на него могут ссылаться различные сценарии.

Таблицы

Таблица создается с помощью дескрипторов <TABLE> …</TABLE>. Эти дескрипторы создают объект таблицы в том месте текста, где они добавлены в коде HTML. Теперь в таблицу нужно добавить строки и столбцы. Для этого используются дескрипторы:

<TR>…</TR> — новая строка таблицы;

<TH>…</TH> — ячейка заголовка;

<TD>…</TD> — обычная ячейка таблицы.

Заголовок таблицы

Заголовок таблицы задается парными тегами <CAPTION> < /CAPTION > и помещается после тега <TABLE>. Для элемента CAPTION можно задать атрибут, определяющий положение заголовка относительно таблицы — align. Этот атрибут может принимать следующие значения:

top — заголовок показывается сверху таблицы (используется по умолчанию);

bottom — под таблицей;

left — слева от таблицы;

right — справа от таблицы.

Параметры отображения таблицы

Наиболее часто используемые атрибуты HTML-элемента TABLE:

align — задает положение таблицы в окне браузера;

bgcolor — задает цвет фона таблицы;

border — задает толщину внешней границы таблицы;

bordercolor — цвет границы таблицы;

cellpadding — размер пустого пространства между границами и содержимым ячеек таблицы;

cellspacing — размер пустого пространства между ячейками таблицы;

frame — задает отображаемые части внешней рамки таблицы, может принимать одно из перечисленных значений:

void — рамка не отображается (по умолчанию); above — отображается только верхняя граница; below — показывается только нижняя граница; hsides — отображается верхняя и нижняя границы; vsides — показывается правая и левая границы; lhs — отображается только левая граница; rhs — отображается только правая граница; box — рамка отображается полностью;

rules — задает, какие именно границы между ячейками должны отображаться, может принимать одно из следующих значений:

none — границы между ячейками не отображаются;

rows — отображаются только границы между строками;

cols — отображаются только границы между столбцами;
all — отображаются все границы между ячейками;
height — задает рекомендуемую высоту таблицы;
width — задает рекомендуемую ширину таблицы.

Формы

Назначение форм состоит в том, чтобы дать возможность посетителю управлять содержимым веб-страницы, вызывать на выполнение сценарии, а также вводить собственные данные и отправлять их на веб-сервер или по адресу электронной почты.

Создание формы

Для вставки формы в HTML-документ используется элемент FORM. Он задается парными тегами <FORM> </FORM>. Между этими тегами помещаются описания элементов управления формы. При создании формы используются следующие атрибуты:

action — обязательный для каждой формы параметр, URI программы-обработчика данных формы;

method — задает способ отправки данных, введенных в форму, может принимать значения get (используется по умолчанию) или post;

enctype — задает тип данных формы, если используется метод отправки post; по умолчанию имеет значение application/x-www-form-urlencoded; при необходимости передачи файлов используется значение multipart/form-data;

accept-charset — применяется при передаче файлов, позволяет указать, какие кодировки используются для каждого из файлов (список строковых значений — названий кодировок), по умолчанию устанавливается значение UNKNOWN (приложение на сервере должно само определять кодировку);

accept — описывает типы файлов (MIME-типы), передаваемые серверу; если этот параметр не использовать, то серверное приложение должно уметь само определять типы передаваемых ему файлов.

Стандартные элементы управления

Элементы управления, которые используются чаще всего: однострочное текстовое поле; поле для ввода пароля; флажки; переключатели; кнопки (как пользовательские, так и выполняющие стандартные действия); поля имен файлов. Все упомянутые элементы управления отображаются браузером Internet Explorer. Для обозначения всех этих элементов управления используется один HTML-элемент — INPUT. Этот элемент задается одиночным тегом <INPUT> и имеет следующие атрибуты:

type — принимает строку, задающую тип элемента управления (по умолчанию используется строка text и создается соответственно поле для ввода текста);

name — используется для задания имени элементу управления (строка, которая помимо идентификации элемента управления добавляется в данные, отсылаемые серверу);

value — начальное значение для полей ввода текста и полей для указания имен файлов, также используется как надпись таких элементов управления, как кнопки; необязательно для всех элементов управления, кроме флажков и переключателей;

checked — булев атрибут, если он установлен, то флажок или переключатель считается (и отображается браузером);

disabled — булев атрибут, установка которого не позволяет пользователю работать с элементом управления;

readonly — булев атрибут, позволяет запретить изменение состояния элемента управления (работает только для текстовых полей и поля выбора файла; для остальных элементов управления лучше использовать атрибут disabled, однако при этом данные деактивированных элементов управления не отправляются серверу);

size — задает размер элемента управления (единицы измерения и действие специфичны для разных элементов управления);

maxlenth — задает максимальную длину текста, который может быть введен в текстовые поля (положительное численное значение);

src — для элемента управления image задает расположение используемого изображения;

title — описание элемента управления (может отображаться браузерами как всплывающая подсказка);

align — задает горизонтальное выравнивание элемента управления, работает так же, как и для любого другого HTML-элемента, поддерживающего этот атрибут;

tabindex — номер элемента управления при навигации при помощи табуляции;

accesskey — «горячая» клавиша для элемента управления (для перехода к элементу управления нужно нажать Alt и заданную клавишу).

Значения атрибута type показаны в табл. 1.1

Таблица 1.1

Значения атрибута type

Значение атрибута	Особенности
text	Создается однострочное текстовое поле. Значение атрибута value отображается в качестве текста по умолчанию. Атрибут size воспринимается как количество символов, которое может отображаться в поле без необходимости горизонтальной прокрутки текста. Атрибут maxlength задает максимальное количество символов, которые могут быть введены в поле
password	Действие этого значения аналогично действию значения text, но дополнительно скрывается вводимый пользователем текст (заменяется символами «*» или другими). Используется для ввода конфиденциальной информации типа паролей
checkbox	Создает элемент управления «флажок». Задавать значение атрибута value нужно обязательно, так как именно это значение отправляется серверу, если флажок установлен. Можно использовать несколько элементов управления checkbox с одинаковым значением атрибута name для обеспечения возможности задания нескольких значений одного свойства
radio	Создает переключатель. Имеет смысл использовать как минимум два этих элемента управления вместе, объединенных в группу, с заданием одного и того же значения атрибута name. Тогда у пользователя появляется возможность выбора одного из нескольких вариантов (одновременно можно установить только один переключатель в группе,). Значение атрибута value задавать также обязательно, оно используется аналогично элементу управления checkbox
submit	Создает кнопку, при нажатии которой браузер отправит форму. Значение атрибута value задает надпись на кнопке

Окончание табл. 1.1

Значение атрибута	Особенности
file	Создает поле для ввода имени файла с возможностью выбора файла с помощью диалогового окна открытия файла или без него, что зависит от браузера. Содержимое выбранного файла или файлов пересылается вместе с формой. Для корректной работы необходима установка значения атрибута method формы в post, а enctype — в multipart/form-data. Значение атрибута value используется как имя файла по умолчанию. Значение атрибута size задается и работает аналогично элементу управления text
image	Создает изображение, при щелчке на котором произойдет отправка формы браузером. При этом на сервер передаются также координаты указателя мыши относительно левого верхнего угла изображения. Атрибут src задает расположение используемого изображения
reset	Создает кнопку, при нажатии которой значения всех элементов управления будут заменены значениями по умолчанию. Значение value задает надпись на кнопке
button	Создает пользовательскую кнопку. При нажатии или другом действии с кнопкой может выполняться ассоциированный сценарий. Значение атрибута value используется как надпись на кнопке
hidden	Создает скрытый элемент управления. Для него задаются атрибуты name и value. Пользователь не видит и не может изменить содержимое этого элемента управления. Однако данные скрытого элемента управления отправляются серверу вместе с остальными данными формы

Классический способ создания HTML-разметки

Классический *HTML-метод* создания элементов и обработки POST-запроса. В этом случае можно просто вручную добавлять HTML-теги и формировать ответ на POST-запрос перед повторной загрузкой веб-страницы в событии Load. Достоинство данного способа — возможность полной самостоятельной обработки запросов. В HTML-элементах формы обязательно проставляем атрибут name, ведь на нем строится механизм расшифровки запросов на основе ключ-значение. Программная связь поддерживается через свойства и методы возвращающих тип string. Надо добавить, что при данном способе вся ответственность за безопасность сайта ложится на вас и проверку входных данных будете осуществлять тоже вы.

Серверная обработка форм

Второй способ похож на первый с одной маленькой, но важной деталью кода: атрибутом runat="server". В этом случае элементы HTML-формы мы доверяем обрабатывать серверу и *.NET Framework*, для этого надо вместе с атрибутом runat="server" обязательно присвоить элементу идентификатор. После этого вроде бы сугубо HTML-элементы превращаются в объекты классов пространства имен System.Web.UI.HtmlControls, становясь серверными элементами управления HTML. Теперь можно программно управлять HTML-элементами и обращаться к их свойствам посредством значения атрибута *id*. В исходном коде веб-страницы атрибут *name* заимствуется от атрибута id. Заметьте, что у элемента с идентификатором Length атрибут name отличен от id, но после загрузки страницы в браузер name будет равен идентификатору.

Быстрый способ создания HTML-форм

Конечно, два способа создания HTML-кода страницы Web Forms, описанных выше, больше будут по душе тем, кто в HTML как рыба в воде. Но данными способами легко писать только небольшие по объему интерфейсы веб-страниц и утомительно будет писать сложные *HTML-группы*. Для таких целей и служит третий способ, оправдывающий характеристику *ASP.NET Web Forms* как технологии быстрой разработки веб-приложений. Третий способ и основной для ASP.NET Web Forms — это использование встроенных элементов управления. Сотни встроенных элементов и возможность создания своих многократно увеличивают скорость разработки веб-сайтов.

Вид кода элементов несколько непривычен для веб-программистов и первоначально замедляет скорость разработки. Методы работы с Web Forms более дружелюбны к desktop-программистам и предназначены для быстрого перехода на веб-платформу. В этом случае вы только формируете интерфейс, а каркас Web Forms действия пользователей превращает в соответствующие события. Веб-формы предоставляют возможность одним объектом класса создавать множество html-тегов для построения интерфейса. Кроме html-кода веб-компоненты могут генериро-

вать и необходимый *javascript* код. Есть возможность создавать собственные компоненты.

Рекомендации

В модели Web Forms, как правило, чистые HTML-теги применяются только для декоративных элементов страницы. Интерактивные же элементы управления, в целях безопасности, лучше доверять серверу и каркасу Web Forms. Используя встроенные или созданные самостоятельно веб-элементы, можно существенно повысить скорость разработки. Именно для быстрого создания веб-сайтов и предназначены веб-формы Web Forms.

1.4.3. Razor — интеллектуальный и понятный

Razor — это интеллектуальный обработчик программного кода динамических веб-страниц ASP.NET. Имеет простой, интуитивно понятный, синтаксис встраивания программного кода в веб-страницы. Razor — механизм визуализации (view engine), поддерживаемый .NET Framework, .NET Core в рамках ASP.NET и специально предназначенный для создания веб-приложений.

Движок Razor дает всю мощь ASP.NET, но использует упрощенный синтаксис, легкий для новичков и повышающий производительность для профессионалов. Он позволяет писать программный код без множества открывающих и закрывающих меток по всему шаблону, делая процесс разработки по-настоящему быстрым. Дизайн движка Razor использует минимальное количество символов для обозначения границ программного кода, не требуя явно обозначать серверные блоки в HTML-коде.

Краткая история Razor

Разработку движка Razor начали в июне 2010 г., и в январе 2011 г. он был выпущен для Microsoft Visual Studio 2010 как часть *ASP.NET MVC 3* и набора инструментов *Microsoft WebMatrix*. Целью создания нового механизма визуализации было обеспечение возможности разработчикам использовать свои существующие C# и Visual Basic языковые навыки. А также возможности быстрого интегрирования серверного кода в разметку HTML с минимальным количеством разграничивающих символов.

Свойства движка Razor

С помощью механизма визуализации Razor можно интуитивно легко создавать сложные композиции программного кода, гармонирующего с разметкой HTML. Когда сервер «читает страницу», движок Razor обрабатывает код веб-страницы прежде, чем сервер отправит сгенерированные данные в браузер. Загруженная в браузер веб-страница, порожденная серверным кодом, ничем не отличается от статического контента HTML. Файлы веб-страниц ASP.NET с синтаксисом Razor имеют специфическое расширение *.cshtml* (Razor с использованием C #) и *.vbhtml* (Razor с помощью Visual Basic).

Синтаксис Razor

Если работать в MS Visual Studio .NET и Visual Studio Code, то программирование на Razor можно освоить буквально за несколько часов. Встроенные в студии анализаторы кода оказывают существенную помощь в освоении механизма визуализации.

Блок объявления типов, методов, объектов и переменных, обозначенный как `@functions{…}`, дает возможность создавать любой синтаксически корректный код на языках .NET, что равносильно созданию кода в файле с расширением .cs. Программный код в блоке `@functions{…}` воспринимается как составляющая класса с именем идентичным названию файла .cshtml. Разместив данный файл в папке App_Code и создав переменные, методы и классы с модификатором `public`, можно получить доступ к этому коду с любой страницы сайта. Например, если вы создали файл Utility.cshtml, то код будет доступен на других страницах через `@Utility.название_метода()`, `@Utility.название_переменной` и т. д.

1.4.4. Веб-сайт на базе Web Forms

Веб-сайт построен на базе Web Forms платформы интернет-проектов ASP.NET, язык программирования C#. Дизайн сайта создан на основе свободно распространяемых шаблонов. Для большей реалистичности сайт состоит из нескольких страниц. Все веб-страницы размещены в корневом каталоге.

Функционирование сайта

Для загрузки веб-страниц используются натуральные адреса: как страница называется физически, так она обозначается и в адресной строке браузера. Достоинство такого подхода — простота и точность названия файлов веб-страниц, недостатки — сложно обслуживать сайт с большим количеством страниц.

Контент страниц веб-сайта хранится в базе данных. Роль базы данных играет файл .xml. Изменив текстовое содержание файла базы данных, можно изменить отображаемый текст на страницах сайта. Специальный класс унифицирует загрузку содержимого для каждой веб-страницы отдельно. Вместо файла .xml, конечно же, можно использовать любую двоичную базу данных.

Мастер-страницы Web Forms

Веб-сайт построен с помощью мастер-страницы и дочерних страниц с контентом. Это сделано для обеспечения однообразия вида страниц сайта. Мастер-страница содержит основной дизайн, в дочерних страницах только контент необходимый для отображения. При загрузке мастер-страница автоматически сливается с запрошенной дочерней страницей в единое целое.

По сути мастер-страница — обыкновенная страница ASPX, но со специальным расширением .master. Так же, как и любая страница ASPX, она состоит из двух страниц, одна с HTML-разметкой (плюс внедряемый код), другая с отделенным программным кодом на языках .NET: C# или Visual Basic.

Дочерние страницы

В отличие от мастер-страницы дочерняя сильно упрощена, что удобно для сосредоточенной работы с контентом данной веб-страницы. Обычно дочерние страницы имеют очень мало служебного кода.

1.4.5. Инфраструктура ASP.NET MVC

В ASP.NET применяется традиционная схема MVC.

Аббревиатура *MVC* происходит от слов *Model-View-Controller*. Model (Модель) — что будем показывать, View

(Вид) — как будем показывать, Controller (Контроллер) — кто будет управлять.

Представление (view) — это собственно визуальная часть или пользовательский интерфейс приложения. Как правило, html-страница, которую пользователь видит, зайдя на сайт.

Контроллер (controller) представляет класс, обеспечивающий связь между пользователем и системой, представлением и хранилищем данных. Он получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления.

Модель (model) представляет класс, описывающий логику используемых данных.

Для примера, пользователь запускает процесс регистрации и отправляет на сервер регистрационные данные. Контроллер интерпретирует действия человека и передает модели информацию о внесенных изменениях в статус пользователя. Модель реагирует на действия контроллера и работает с поставляемыми данными. Вид отвечает за отображение информации с модели.

Таким образом, модель отвечает за функционал, например соединение с базой данных или обработку с дополнительными данными. Вид отвечает за отображение данных на странице, а контроллер позволяет связать Вид и Модель между собой.

После отправки запроса на сервер его начинает обрабатывать контроллер, затем передает изменения в модель, которая реагирует на обновление и выдает все нужное для отображения сайта. Вид выполняет только роль отображения внешнего вида страницы — обычный HTML-шаблон.

Стандарт MVC наиболее распространенный стиль создания сайтов. Четкое разделение разных составляющих веб-проекта: дизайн, управление запросами пользователей, программная логика. Благодаря разбиению сложного на простые части повышается эффективность работы над веб-сайтом.

Существует несколько версий ASP.NET.

.NET Core — кроссплатформенная среда выполнения для приложений из веба или консольных программ. Программные продукты, разработанные на ней, могут успешно исполняться на Linux, Windows, MacOS.

Основные плюсы: мультиплатформенность, открытый исходный код.

.NET Framework — среда исполнения, которая предназначена исключительно для Windows. Помогает в разработке десктопных программ под Windows и веб-приложений ASP.NET под IIS.

Ключевые достоинства: огромный ассортимент готовых библиотек, значительно больше возможностей в сравнении с прошлой платформой.

Сегодня Microsoft усиленно продвигает и модернизирует Core, постоянно добавляя полезный функционал. Может быть, скоро эта среда сравняется с .NET Framework.

ASP.NET Core — лучше применять, если вы:

– желаете установить таргетинг в приложении на все популярные ОС;

– не страшитесь изучения нового;

– не боитесь уделять достаточно времени исправлениям и доработкам, ведь Core не дошел до статической точки, периодически меняется.

ASP .NET — идеальное решение, если вы:

– не испытываете необходимости организовывать кросс-платформенную поддержку веб-приложения;

– нуждаетесь в стабильной среде разработки;

– не имеете большого количества времени для разработки;

– уже занимаетесь разработкой или модернизацией существующей программы;

– входите в состав команды, обладающей опытом работы с ASP.NET.

Тем, кто сегодня только планирует начать обучение и в ближайший год начать работать на крупные компании, ASP.NET Core подходит идеально. С этой средой вы получаете много перспектив на будущее.

.NET Framework поддерживает множество языков. Самым популярным является C#.

Краткая история MVC

Впервые концепция MVC была сформулирована и описана в 1979 г. Трюгве Реенскаугом (норвежский ученый в сфере компьютерных наук и заслуженный профессор университета Осло), ра-

ботавшим в то время над языком программирования Smalltalk в Xerox PARC. Затем на практике была реализована версия MVC для библиотеки классов Smalltalk-80. Окончательная версия концепции была опубликована лишь в 1988 г. в журнале Journal of Object Technology. Впоследствии шаблон проектирования MVC стал развиваться и конкретизироваться.

Составляющие части MVC

В настоящее время трехсоставная схема MVC широко распространена и применяется в веб-разработках независимо от языка программирования, платформ и операционных систем.

Функции частей MVC:

Model — программная обработка и обращение к базе данных;

View — выводит результат в требуемом виде, к одной модели можно прикрепить несколько разных представлений;

Controller — распределяет внешние запросы пользователей между моделью и представлением.

Практический смысл MVC

Практическая польза от создания сайтов по концепции MVC:

– комфортность работы над сложной структурой сайта;

– сосредоточение в одном секторе программирования;

– возможность многократного использования отдельных частей приложения;

– возможность использования множества макетов для любой страницы сайта;

– небольшое количество шаблонов страниц, позволяющее создавать богатое разнообразное содержание сайта.

Упрощение работы над сложной структурой сайта

Используя шаблон MVC, изменить дизайн сайта или его логику работы гораздо проще и быстрее. Даже отдельный разработчик, не боясь запутаться в коде, может создавать веб-приложения высокой организованности. Работая в команде, прикладные программисты и веб-дизайнеры будут чувствовать себя каждый в своей родной стихии.

1.4.6. Среда Visual Studio

Для создания веб-приложений на платформе ASP.NET MVC 5 необходима среда разработки — Visual Studio 2017.

Разновидности проектов:

– ASP.NET Core Web Application — применяется для программирования приложений для веба на ASP.NET Core;

– ASP.NET Web Application (.NET Framework) — этот тип применим для классических проектов: Web Forms, MVC 5, Web Api, основанных на обычном .NET Framework.

Шаблоны

– Empty — создает чистый шаблон без дополнительных функций. Применяется для написания приложений с чистого листа;

– Web API — проект веб-службы;

– Web Application — в качестве обработчика запросов проект применяет Razor Pages;

– Web Application — проект, построенный на архитектуре модель-вид-контроллер;

– Angular — работает на Angular 2+;

– Reat JS — основан на React.JS;

– Reat JS and Redux — строится на React JS и Redux;

– Razor Class Library — проект, заточенный под разработку с Razor.

Обращаем внимание на запись «Configure HTTP». Если ее активировать, весь проект во время отладки и тестов будет работать через SSL-протокол.

Структура проекта MVC 5 показана на рис. 1.1.

Весь этот функционал обеспечивается следующей структурой проекта:

App_Data — содержит файлы, ресурсы и базы данных, используемые приложением;

App_Start — хранит ряд статических файлов, которые содержат логику инициализации приложения при запуске;

Content — содержит вспомогательные файлы, которые не включают код на C# или javascript и развертываются вместе с приложением (например, файлы стилей css);

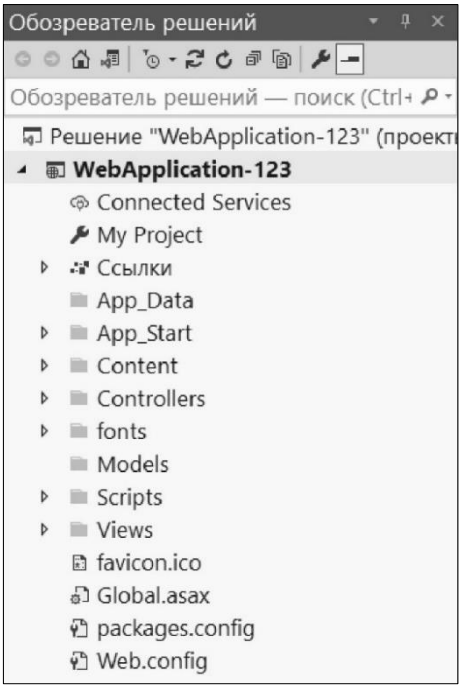

Рис. 1.1

Controllers — содержит файлы классов контроллеров. По умолчанию в эту папку добавляются два контроллера — HomeController и AccountController;

fonts — хранит дополнительные файлы шрифтов, используемых приложением;

Models — содержит файлы моделей. По умолчанию Visual Studio добавляет пару моделей, описывающих учетную запись и служащих для аутентификации пользователя;

Scripts — каталог со скриптами и библиотеками на языке javascript;

Views — здесь хранятся представления. Все представления группируются по папкам, каждая из которых соответствует одному контроллеру. После обработки запроса контроллер отправляет одно из этих представлений клиенту. Также здесь имеется каталог Shared, который содержит общие для всех представления;

Global.asax — файл, запускающийся при старте приложения и выполняющий начальную инициализацию. Как правило, здесь срабатывают методы классов, определенных в папке App_Start;

packages.config — файл, который содержит установленные в проект пакеты NuGet;

Web.config — файл конфигурации приложения.

1.4.7. Понятие модели

В аббревиатуре MVC буква «M» обозначает model (модель), которая является наиболее важной частью приложения.

Все сущности в приложении принято выделять в отдельные модели. В зависимости от поставленной задачи и сложности приложения можно выделить различное количество моделей.

Модель — это представление реальных объектов, процессов и правил, которые определяют сферу приложения, известную как предметная область.

Модель, которую часто называют моделью предметной области, содержит объекты C# (или объекты предметной области), которые образуют «вселенную» приложения, и методы, позволяющие манипулировать ими. Представления и контроллеры открывают предметную область клиентам в согласованной манере, и любое корректно разработанное приложение MVC начинается с хорошо спроектированной модели, которая затем служит центральным узлом при добавлении контроллеров и представлений.

Модели представляют собой простые классы и располагаются в проекте в каталоге Models. Они описывают логику данных.

Модель не обязательно состоит только из свойств, кроме того, она может иметь конструктор, какие-нибудь вспомогательные методы. Но главное — не перегружать класс модели и помнить, что его предназначение — описывать данные. Манипуляции с данными и бизнес-логика — это больше сфера контроллера.

Данные моделей, как правило, хранятся в базе данных. Для работы с базой данных очень удобно пользоваться фреймворком Entity Framework, который позволяет абстрагироваться от написания sql-запросов, от строения базы данных и полностью сосредоточиться на логике приложения.

Entity Framework

Для работы с данными в ASP.NET MVC рекомендуется использовать фреймворк Entity Framework, хотя это не обязательно и всецело зависит от предпочтений разработчика.

Entity Framework — это решение для работы с базами данных, которое используется в программировании на языках семейства .NET. Оно позволяет взаимодействовать с СУБД с помощью сущностей (entity), а не таблиц. Также код с использованием EF пишется гораздо быстрее. Преимущество этого фреймворка состоит в том, что он позволяет абстрагироваться от структуры конкретной базы данных и вести все операции с данными через модель.

Если проект не содержит библиотек EntityFramework, то чтобы их добавить в проект, необходимо воспользоваться пакетным менеджера NuGet. В окне Solution Explorer (Обозреватель решений) нажмем правой кнопкой мыши в структуре проекта на узел References и в появившемся меню выберем Manage NuGet Packages...

В окне управления пакетами NuGet в правом верхнем углу введем в поле поиска Entity Framework и нажмем Enter. После этого в среднем столбце будут отображены все найденные пакеты, которые имеют отношение к запросу, а самым первым будет пакет самого фреймворка EntityFramework, который нам и надо установить.

Если при создании проекта MVC 5 вы выберете в качестве типа аутентификации No Authentication, то после создания проекта в него надо будет подключить Entity Framework через пакетный менеджер NuGet.

Для хранения данных приложению нужна база данных. Мы можем использовать различные СУБД, но, как правило, в связке с ASP.NET MVC используется база данных MS SQL Server.

Можно создать базу данных прямо в проекте либо на сервере MS SQL. Для хранения баз данных в проекте предназначена папка App_Data.

Сегодня большое значение имеет работа с данными. Для хранения данных используются различные системы управления базами данных: MS SQL Server, Oracle, MySQL и т. д. И большин-

ство крупных приложений так или иначе используют для хранения данных эти СУБД. Однако, чтобы осуществлять связь между базой данных и приложением на C#, необходим посредник. Таким посредником является технология ADO.NET.

ADO.NET представляет собой технологию работы с данными, которая основана на платформе .NET Framework. Она предоставляет нам набор классов, через которые мы можем отправлять запросы к базам данных, устанавливать подключения, получать ответ от базы данных и производить ряд других операций.

Важно отметить, что систем управления базами данных может быть множество. В своей сущности они могут различаться. MS SQL Server, например, для создания запросов использует язык T-SQL, а MySQL и Oracle применяют язык PL-SQL. Разные системы баз данных могут иметь разные типы данных. Также могут различаться какие-то другие моменты. Однако функционал ADO.NET построен таким образом, чтобы предоставить разработчикам унифицированный интерфейс для работы с самыми различными СУБД.

Основу интерфейса взаимодействия с базами данных в ADO.NET представляет ограниченный круг объектов: Connection, Command, DataReader, DataSet и DataAdapter. С помощью Connection происходит установка подключения к источнику данных. Command позволяет выполнять операции с данными из БД. DataReader считывает полученные в результате запроса данные. DataSet предназначен для хранения данных из БД и позволяет работать с ними независимо от БД. DataAdapter является посредником между DataSet и источником данных. Главным образом через эти объекты и будет идти работа с базой данных.

Однако, чтобы использовать один и тот же набор объектов для разных источников данных, необходим соответствующий провайдер данных. Собственно, через провайдер данных в ADO.NET и осуществляется взаимодействие с базой данных. Причем для каждого источника данных в ADO.NET может быть свой провайдер, который и определяет конкретную реализацию вышеуказанных классов.

По умолчанию в ADO.NET имеются следующие встроенные провайдеры:

– для MS SQL Server;

– для OLE DB (предоставляет доступ к некоторым старым версиям MS SQL Server, а также к БД Access, DB2, MySQL и Oracle);

– для ODBC (провайдер для тех источников данных, для которых нет своих провайдеров);

– для Oracle;

– EntityClient (провайдер данных для технологии ORM Entity Framework);

– для сервера SQL Server Compact 4.0.

Кроме этих провайдеров, которые являются встроенными, существует также множество других, предназначенных для различных баз данных, например для MySQL.

Основные пространства имен, которые используются в ADO.NET:

System.Data — определяет классы, интерфейсы, делегаты, которые реализуют архитектуру ADO.NET;

System.Data.Common — содержит классы общие для всех провайдеров ADO.NET;

System.Data.Design — определяет классы, которые используются для создания своих собственных наборов данных;

System.Data.Odbc — определяет функциональность провайдера данных для ODBC;

System.Data.OleDb — определяет функциональность провайдера данных для OLE DB;

System.Data.Sql — хранит классы, которые поддерживают специфичную для SQL Server функциональность;

System.Data.OracleClient — определяет функциональность провайдера для баз данных Oracle;

System.Data.SqlClient — определяет функциональность провайдера для баз данных MS SQL Server;

System.Data.SqlServerCe — определяет функциональность провайдера для SQL Server Compact 4.0;

System.Data.SqlTypes — содержит классы для типов данных MS SQL Server;

Microsoft.SqlServer.Server — хранит компоненты для взаимодействия SQL Server и среды CLR.

1.4.8. Понятие контроллера

Контроллер является центральным компонентом в архитектуре MVC. Контроллер получает ввод пользователя, обрабатывает его и посылает обратно результат обработки, например, в виде представления.

При использовании контроллеров существуют некоторые условности. Так, по соглашениям об именовании названия контроллеров должны оканчиваться на суффикс Controller, остальная же часть (до этого суффикса) считается именем контроллера.

Чтобы обратиться к контроллеру из веб-браузера, надо в адресной строке набрать адрес_сайта/Имя_контроллера/. Так, по запросу адрес_сайта/Home/ система маршрутизации по умолчанию вызовет метод Index контроллера HomeController для обработки входящего запроса. Если мы хотим отправить запрос к конкретному методу контроллера, то нужно указывать этот метод явно: адрес_сайта/Имя_контроллера/Метод_контроллера. Например, адрес_сайта/Home/Buy — обращение к методу Buy контроллера HomeController.

Контроллер представляет обычный класс, который наследуется от базового класса System.Web.Mvc.Controller. В свою очередь, класс Controller реализует абстрактный базовый класс ControllerBase, а через него и интерфейс IController. Таким образом, формально, чтобы создать свой класс контроллера, достаточно создать класс, реализующий интерфейс IController и имеющий в имени суффикс Controller.

Интерфейс IController определяет один-единственный метод Execute, который отвечает за обработку контекста запроса.

При обращении к любому контроллеру система передает в него контекст запроса. В этот контекст запроса включается все: куки, отправленные данные форм, строки запроса, идентификационные данные пользователя и т. д. Реализация интерфейса IController позволяет получить этот контекст запроса в методе Execute через параметр RequestContext.

Хотя с помощью реализации интерфейса IController очень просто создавать контроллеры, в реальности чаще оперируют более высокоуровневыми классами, как, например, класс Controller, поскольку он предоставляет более мощные средства для обра-

ботки запросов. И если при реализации интерфейса IController мы имеем дело с одним методом Execute и все запросы к этому контроллеру будут обрабатываться только одним методом, то при наследовании класса Controller мы можем создавать множество методов действий, которые будут отвечать за обработку входящих запросов и возвращать различные результаты действий.

1.4.9. Понятие представления

Представление — это компонент View ASP.NET MVC.

Хотя работа приложения MVC управляется главным образом контроллерами, непосредственно пользователю приложение доступно в виде представления, которое и формирует его внешний вид. В ASP.NET MVC 5 представления — это файлы с расширением cshtml, которые содержат код пользовательского интерфейса в основном на языке HTML.

Хотя представление содержит главным образом код HTML, оно не является HTML-страницей. При компиляции приложения на основе требуемого представления сначала генерируется класс на языке C#, а затем этот класс компилируется.

Пути к файлам представлений

Все добавляемые представления, как правило, группируются по контроллерам в соответствующие папки в каталоге Views. Представления, которые относятся к методам контроллера Home, будут находиться в проекте в папке Views/Home. Однако при необходимости мы сами можем создать в каталоге Views папку с произвольным именем, где будем хранить дополнительные представления, не обязательно связанные с определенными методами контроллера.

1.4.10. Маршрутизация и веб-адреса шаблона ASP.NET MVC

В шаблон ASP.NET MVC встроена возможность создания человеко-понятных веб-адресов. При обработке запросов каркас ASP.NET MVC опирается на продвинутую систему маршрутизации, которая сопоставляет все входящие запросы с определен-

ными в системе маршрутами. Каждый отдельный маршрут указывает, какой контроллер (его метод) должен обработать входящий запрос. Встроенный в шаблон маршрут по умолчанию предполагает трехзвенную структуру: контроллер/метод/параметр, а непосредственно адрес выглядит: домен/каталог(имя_контроллера)/раздел(название_метода)/параметр. Возможно определение любой структуры маршрутов.

Еще раз о комфортности схемы MVC

По мере работы над сайтом можно возвращаться к каждой части шаблона модель-вид-контроллер и по необходимости дорабатывать и совершенствовать ее. Последовательность работы с контроллерами, моделями и представлениями любая, можно работать над каждой частью независимо от других частей, в чем и заключается достоинство концепции MVC.

1.4.11. Язык программирования C#

Язык программирования C# достаточно молодой. Он создавался в конце 1990-х гг. разработчиками из компании Microsoft. Одним из создателей считается Андерс Хейлсберг.

Если планируется программировать на C#, то программирование будет для Windows. Связано это не столько с языком C#, сколько с платформой .NET («дот нет»), под которую и разрабатывался язык — C# анонсирован как базовый язык для реализации в рамках технологии .NET.

Исполнительная среда .NET Framework предложена и поддерживается Microsoft как средство для выполнения приложений, компоненты (составные части) которых написаны на разных языках программирования. Язык программирования C# тесно связан с этой технологией, поскольку многие важные для C# библиотеки являются составной частью среды .NET Framework и, что более важно, откомпилированные C# программы выполняются под управлением этой среды.

Если на компьютере не установлена платформа .NET Framework, про программирование в C# можно забыть.

Совершено очевидно, что для совместной работы или совместного использования компонентов, написанных на разных языках, необходима «военная хитрость». Военная хитрость состоит в том, что при компиляции программного кода получается промежуточный псевдокод.

Промежуточный псевдокод называется общим промежуточным языком, или CIL — сокращение от Common Intermediate Language.

Псевдокод выполняется под управлением специальной системы, которая является составной частью платформы .NET Framework и называется CLR — сокращение от Common Language Runtime. Система CLR, в свою очередь, для выполнения промежуточного псевдокода вызывает специальный встроенный в среду компилятор. Компилятор переводит псевдокод в исполнительный код. Делается это непосредственно перед выполнением программы, что существенно оптимизирует время выполнения кода.

Если используется операционная система Windows и другие популярные продукты компании Microsoft, то, скорее всего, платформа .NET Framework уже установлена. Во всяком случае имеет смысл проверить систему на наличие файла csc.exe.

Объектно-ориентированное программирование

По сути, программа — это набор инструкций о том, какие данные и какими методами обрабатывать. Если и данных, и функций для их обработки много, то возможна путаница. Главная идея объектно-ориентированного программирования (ООП) состоит в том, чтобы объединить данные и функции для их обработки на одном из базовых уровней — на уровне тех «строительных блоков», из которых создается программа. Эта идея (объединение в одно целое данных и программного кода для их обработки) называется инкапсуляцией.

ООП базируется на трех понятиях:

– инкапсуляция;
– полиморфизм;
– наследование.

Инкапсуляция (от лат. *In capsule* — в оболочке) — это заключение данных и функционала в оболочку. В объектно-ориентированном программировании в роли оболочки выступают классы: они не только собирают переменные и методы в одном месте, но и защищают их от вмешательства извне (*сокрытие*).

Объект и класс в ОПП

Основными элементами программы являются не переменные и методы (процедуры), а объекты.

Объекты — это программные конструкции, включающие набор логически связанных свойств (данных) и методов.

Объекты создаются на основе шаблонов, которые называются классами, и являются экземплярами этих классов.

Создать объект без класса нельзя (во всяком случае в C#).

Класс в объектно-ориентированном программировании представляет собой шаблон для создания объектов, обеспечивающий начальные значения состояний: инициализация полей-переменных и реализация поведения функций или методов.

На основании одного класса можно создать много объектов, а можно не создать ни одного.

Класс напоминает описание типа данных, с той лишь принципиальной разницей, что кроме непосредственно данных в класс включаются и функции (как правило, предназначенные для обработки этих данных).

В ООП принято называть данные, относящиеся к классу, полями класса, а функции, относящиеся к классу — методами класса. Поля класса и методы класса называются членами класса. Помимо полей и методов, классы C# могут содержать свойства, индексаторы, события.

Описание класса начинается с ключевого слова class. После этого ключевого слова указывается имя класса. Непосредственно код класса указывается в блоке из фигурных скобок: открывающей { и закрывающей }.

Эта пара фигурных скобок очень часто используется в C# для выделения программных кодов. Место размещения фигурных скобок крайне демократично — их можно располагать где угодно, лишь бы последовательность скобок и команд была правильной.

Пример описания класса автомобилей:

```
class Автомобиль
{ // описание данных
// описание методов }
```

Создание класса

Что имеет автомобиль? В частности, это:

– марка;
– цвет;
– мощность (в л/с);
– максимальная скорость (км/ч);
– объем бака (л);
– расход топлива (л) на 100 км пути.

```
public class Car
{
  private string brand;
  private string color;
  private int power;
  private int maxSpeed;
  private int volumeOfTank;
  private double fuelConsumption;
}
```

Класс объявляется так: модификатор доступа (public), ключевое слово class и имя класса. Тело класса определяется фигурными скобками. Внутри класса объявлены его поля.

Модификаторы доступа private и public

Модификаторы доступа служат для определения полномочий доступа к членам класса извне. Если перед полем или методом стоит ключевое слово private, то обращаться к данному члену можно только внутри класса. Член с модификатором public доступен за пределами класса, т. е. другие классы могут напрямую получить или модифицировать значение поля (категорически не рекомендуется поле делать public, для безопасного получения и установки значения нужно использовать геттеры и сеттеры) либо вызвать публичный метод.

С помощью модификаторов доступа реализуется ключевой принцип ООП — инкапсуляция данных (их сокрытие).

Слова в именах членов класса пишутся слитно и каждое новое начинается с заглавной буквы. Например, numberOfPeople.

Существует ряд особенностей:

– имена классов (не объектов, а именно классов. Объект — это поле) всегда пишутся с Заглавной буквы;

– первая буква в имени поля всегда маленькая;

– в языке C# первая буква имени метода всегда Заглавная;

– в языке Java первую букву имени метода всегда принято писать маленькой.

Следует понимать, что класс — это каркас, иначе говоря, описание реального объекта. На основе этого «описания» создаются экземпляры реального объекта. Логично предположить, что необходим механизм для присваивания значениям полей характеристик объекта. Для этого существуют конструкторы класса.

Конструктор класса

Конструктор класса — это специальный метод, который вызывается при создании нового объекта и используется для инициализации полей класса значениями, а также для начальных вычислений, если они необходимы. После создания объекта конструктор вызвать нельзя. Кроме того, данный метод никогда не возвращает никакого значения.

Конструктор для инициализации полей в классе Car:

```
public class Car
{
  private string brand;
  private string color;
  private int power;
  private int maxSpeed;
  private int volumeOfTank;
  private double fuelConsumption;
  //конструктор класса
  public Car(string newBrand, string newColor, int
newPower, int newMaxSpeed,
    int newVolumeOfTank, double newFuelConsumption)
  {
    brand = newBrand;
    color = newColor;
    power = newPower;
    maxSpeed = newMaxSpeed;
    volumeOfTank = newVolumeOfTank;
```

```
        fuelConsumption = newFuelConsumption;
      }
    }
```

Конструктор объявляется так: `public Имя ([параметры])`. Наличие параметров не обязательно. Соответственно выделяют конструкторы класса:

– без параметров;

– с параметрами.

Модификатор доступа обязательно `public`, поскольку конструктор всегда вызывается вне класса.

Конструктор по умолчанию — это пустой конструктор без параметров. Он всегда присутствует в классе (если нет других конструкторов), даже если он не был объявлен явно. Конструктор по умолчанию вызывается автоматически всегда, когда отсутствуют другие конструкторы. Его код это (писать не обязательно):

```
public Car()
{

}
```

Класс может содержать несколько конструкторов с разными параметрами. При создании объекта будет вызван тот, который подходит по параметрам.

Ничто не запрещает написать в классе одновременно конструктор без параметров (явно; тогда им можно будет воспользоваться при создании нового экземпляра класса) и конструктор с параметрами.

Создание экземпляра класса

Для создания экземпляра (объекта) класса Car на примере автомобиля Форд понадобится оператор `new`:

– Ford (марка);

– Серый (цвет);

– 150 (мощность);

– 210 (максимальная скорость);

– 55 (объем бака);

– 6.4 (расход топлива в смешанном цикле).

```
Car c = new Car("Ford", "Серый", 150, 210, 55, 6.4);
```

Сам по себе класс является ссылочным типом данных.

Самая важная особенность ссылочных типов данных состоит в том, что они передаются не по значению, а по ссылке. Что это значит?

Ссылочные типы данных не являются примитивными, их размер не фиксирован и может быть произвольным, кроме того, они хранятся не на участке памяти переменной, а в совершенно другом месте памяти компьютера. Ссылочными типами, например, являются массивы. В объектно-ориентированных языках программирования это экземпляры классов, коллекции и т. п.

Если бы данный класс не содержал конструктор с параметрами, то создание нового объекта происходило бы так:

```
Car c = new Car();
```

Методы класса

В ООП методы класса схожи по назначению с функциями из процедурного программирования.

Общая схема объявления метода:

`[модификаторДоступа] типВозвращаемогоЗначения имяМетода ([аргументы]) { }`.

Тип возвращаемого значения может быть `void` (ничего не возвращает).

Наличие аргументов необязательно.

Например, создадим метод, который рассчитает, на сколько километров пути хватит полного бака бензина:

```
public double QuantityOfKilometers()
{
  double quantity = 100 * volumeOfTank / fuelConsumption;
  return quantity;
}
```

А также создадим метод с тем же именем, но с параметром «количество топлива»; рассчитаем, на сколько километров пути его хватит:

```
public double QuantityOfKilometers(double volume)
{
  double quantity = 100 * volume / fuelConsumption;
  return quantity;
}
```

Методы с одинаковым именем создавать можно, но они должны иметь разные аргументы (по количеству или типу данных; могут отсутствовать вовсе). Компилятор выберет тот, который подходит при вызове метода по аргументам. Методы с идентичными именами, но разными параметрами иллюстрируют полиморфизм в ООП.

Полиморфизм (от гр. *poly* — много и *morphe* — форма) — возможность объектов с одинаковой спецификацией иметь различную реализацию. Это один из главных столпов объектно-ориентированного программирования. Его суть заключается в том, что один фрагмент кода может работать с разными типами данных.

Полиморфизм позволяет вызывать методы и свойства объекта независимо от их реализации. Например, объект класса Водитель взаимодействует с объектом класса Автомобиль через открытый интерфейс. Если другой объект, например Грузовик или Гоночный_автомобиль, поддерживает такой открытый интерфейс, то объект класса Водитель сможет взаимодействовать и с ними (управлять ими), невзирая на различия в реализации интерфейса.

Наследование

Один класс может быть описан на основе уже имеющегося описания другого класса. В этом случае между классами задается отношение наследования.

Наследование позволяет создавать новые классы на основе существующих, при этом новые классы обладают всей функциональностью старых и при необходимости могут модифицировать их.

Класс, объявленный на основе некоторого (базового) класса, называется производным или классом-потомком.

У любого класса может быть только один прямой предок — его базовый класс.

Программа в C# может содержать (и обычно содержит) описание нескольких классов. Но всегда есть один класс (который иногда называют главным классом программы), в котором есть метод Main(). Этот метод будет реализован при выполнении программы.

Для того чтобы компилятор смог узнать класс, в самом начале программного кода подключают пространство имен.

Концепция использования пространства имен в C# позволяет структурировать и упорядочить все полезные классы, которые написаны специально для того, чтобы «жизнь стала проще». Все классы, которые идут в комплекте поставки с исполнительной системой, разбиты на группы, или пространства имен. Когда подключаем то или иное пространство, то фактически указываем компилятору, где ему следует искать те классы, на которые мы ссылаемся.

Для подключения пространства имен используют инструкцию using, после которой указывается имя пространства. Одно пространство может содержаться внутри другого. В этом случае иерархия пространства отображается с помощью точечного синтаксиса, как, например, в названии System.Windows.Forms. Обычно в программе подключается сразу несколько пространств имен.

Базовые типы данных и основные операторы

Чтобы понять, что в принципе можно делать с данными в программе, желательно сначала выяснить, какими эти данные могут быть. И в этом деле не обойтись без рассмотрения базовых типов данных. Более полное представление о базовых типах языка C# дает табл. 1.2.

Таблица 1.2

Базовые типы C#

Тип	Класс	Биты	Значения	Описание
byte	Byte	8	от 0 до 255	Целые неотрицательные числа
sbyte	SByte	8	от −128 до 127	Целые числа
short	Int16	16	от −32 768 до 32 767	Целые числа
ushort	UInt16	16	от 0 до 65 535	Целые неотрицательные числа
int	Int32	32	от −2 147 483 648 до 2 147 483 647	Целые числа
uint	UInt32	32	от 0 до 4 294 967 295	Целые неотрицательные числа

Окончание табл. 1.2

Тип	Класс	Биты	Значения	Описание
long	Int64	64	от −9 223 372 036 854 775 808 до 9 223 372 036 854 775 807	Целые числа
ulong	UInt64	64	от 0 до 18 446 744 073 709 551 615	Целые неотрицательные числа
float	Single	32	от 1.5E−45 до 3.4E+38	Действительные числа
double	Double	64	от 5E−324 до 1.7E+308	Действительные числа
decimal	Decimal	128	от 1E−28 до 7.9E+28	Действительные числа — специальный тип для выполнения особо точных (финансовых вычислений)
char	Char	16	от 0 до 65 535	Символьный тип
bool	Boolean	8	Значение true и false	Логический тип

Для каждого базового (примитивного) типа данных в C# есть класс-оболочка. Через такие классы реализуются данные соответствующих типов, но уже как объекты. Хотя наличие классов-оболочек на первый взгляд может показаться излишним, на практике это достаточно удобно, поскольку через такие классы реализуются многие полезные методы для работы с данными. В табл. 1.2 кроме прочего приведены и классы-оболочки для базовых типов данных.

bool — хранит значение true или false (логические литералы). Представлен системным типом System.Boolean.

```
bool alive = true;
bool isDead = false;
```

byte — хранит целое число от 0 до 255 и занимает 1 байт. Представлен системным типом System.Byte.

```
byte bit1 = 1;
byte bit2 = 102;
```

sbyte — хранит целое число от −128 до 127 и занимает 1 байт. Представлен системным типом System.SByte.

```
sbyte bit1 = -101;
sbyte bit2 = 102;
```

short — хранит целое число от −32768 до 32767 и занимает 2 байта. Представлен системным типом System.Int16.

```
short n1 = 1;
short n2 = 102;
```

ushort — хранит целое число от 0 до 65535 и занимает 2 байта. Представлен системным типом System.UInt16.

```
ushort n1 = 1;
ushort n2 = 102;
```

int — хранит целое число от −2147483648 до 2147483647 и занимает 4 байта. Представлен системным типом System.Int32. Все целочисленные литералы по умолчанию представляют значения типа int:

```
int a = 10;
int b = 0b101; // бинарная форма b =5
int c = 0xFF; // шестнадцатеричная форма c = 255
```

uint — хранит целое число от 0 до 4294967295 и занимает 4 байта. Представлен системным типом System.UInt32.

```
uint a = 10;
uint b = 0b101;
uint c = 0xFF;
```

long — хранит целое число от −9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 и занимает 8 байт. Представлен системным типом System.Int64.

```
long a = -10;
long b = 0b101;
long c = 0xFF;
```

ulong — хранит целое число от 0 до 18 446 744 073 709 551 615 и занимает 8 байт. Представлен системным типом System.UInt64.

```
ulong a = 10;
ulong b = 0b101;
ulong c = 0xFF;
```

float — хранит число с плавающей точкой от $-3.4*10^{38}$ до $3.4*10^{38}$ и занимает 4 байта. Представлен системным типом System.Single.

double — хранит число с плавающей точкой от $\pm 5.0*10^{-324}$ до $\pm 1.7*10^{308}$ и занимает 8 байт. Представлен системным типом System.Double.

decimal — хранит десятичное дробное число. Если употребляется без десятичной запятой, имеет значение от $\pm 1.0*10^{-28}$ до $\pm 7.9228*10^{28}$, может хранить 28 знаков после запятой и занимает 16 байт. Представлен системным типом System.Decimal.

char — хранит одиночный символ в кодировке Unicode и занимает 2 байта. Представлен системным типом System.Char. Этому типу соответствуют символьные литералы:

```
char a = 'A';
char b = '\x5A';
char c = '\u0420';
```

string — хранит набор символов Unicode. Представлен системным типом System.String. Этому типу соответствуют строковые литералы.

```
string hello = "Hello";
string word = "world";
```

object — может хранить значение любого типа данных и занимает 4 байта на 32-разрядной платформе и 8 байт на 64-разрядной платформе. Представлен системным типом System.Object, который является базовым для всех других типов и классов .NET.

```
object a = 22;
object b = 3.14;
object c = "hello code";
```

Например, определим несколько переменных разных типов и выведем их значения на консоль (рис. 1.2).

Для вывода данных на консоль здесь применяется интерполяция: перед строкой ставится знак $, и после этого мы можем вводить в строку в фигурных скобках значения переменных. Консольный вывод программы показан на рис. 1.3.

Использование суффиксов

При присвоении значений надо иметь в виду следующую тонкость: все вещественные литералы рассматриваются как значения типа `double`. И чтобы указать, что дробное число представ-

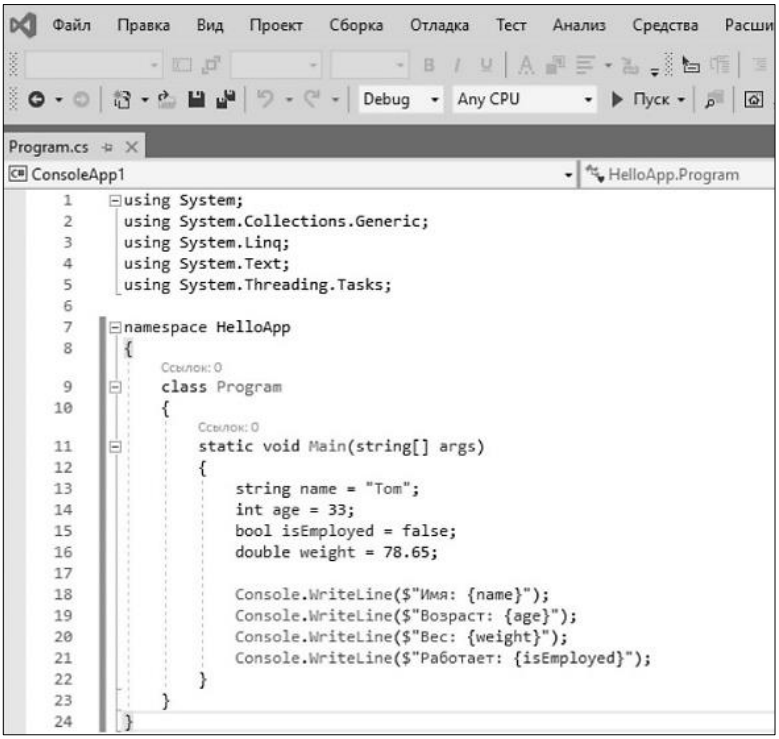

Рис. 1.2

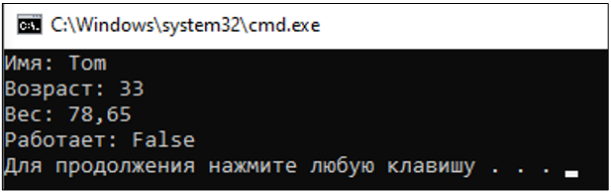

Рис. 1.3

ляет тип `float` или тип `decimal`, необходимо к литералу добавлять суффикс: `F/f` — для `float` и `M/m` — для `decimal`.

```
float a = 3.14F;
float b = 30.6f;
decimal c = 1005.8M;
decimal d = 334.8m;
```

Подобным образом все целочисленные литералы рассматриваются как значения типа `int`. Чтобы явным образом указать, что целочисленный литерал представляет значение типа `uint`, надо использовать суффикс `U/u`, для типа `long` — суффикс `L/l`, а для типа `ulong` — суффикс `UL/ul`:

```
uint a = 10U;
long b = 20L;
ulong c = 30UL;
```

Использование системных типов

Выше при перечислении всех базовых типов данных для каждого упоминался системный тип. По сути название встроенного типа представляет собой сокращенное обозначение системного типа. Например, следующие переменные будут эквивалентны по типу:

```
int a = 4;
System.Int32 b = 4;
```

Неявная типизация

Ранее мы явным образом указывали тип переменных (например, `int x;`). И компилятор при запуске уже знал, что `x` хранит целочисленное значение.

Однако мы можем использовать и модель неявной типизации:

```
var hello = "Hell to World";
var c = 20;
Console.WriteLine(c.GetType().ToString());
Console.WriteLine(hello.GetType().ToString());
```

Для неявной типизации вместо названия типа данных используется ключевое слово `var`. Затем уже при компиляции компилятор сам выводит тип данных исходя из присвоенного значения. В примере выше использовалось выражение `Console.WriteLine(c.GetType().ToString());`, которое позволяет нам узнать выведенный тип переменной `c`. Так как по умолчанию все целочисленные значения рассматриваются как значения типа `int`, то в итоге переменная `c` будет иметь тип `int` или `System.Int32`.

Эти переменные подобны обычным, однако они имеют некоторые ограничения.

Во-первых, мы не можем сначала объявить неявно типизируемую переменную, а затем инициализировать:

```
// этот код работает
int a;
a = 20;
 // этот код не работает
var c;
c= 20;
```

Во-вторых, мы не можем указать в качестве значения неявно типизируемой переменной `null`:

```
// этот код не работает
var c=null;
```

Так как значение `null`, то компилятор не сможет вывести тип данных.

double или decimal

Из вышеперечисленного списка типов данных очевидно, что если мы хотим использовать в программе числа до 256, то для их хранения мы можем использовать переменные типа `byte`. При использовании больших значений мы можем взять тип `short`, `int`, `long`. То же самое для дробных чисел: для обычных дробных чисел можно взять тип `float`, для очень больших дробных чисел — тип `double`. Тип `decimal` здесь стоит особняком в том плане, что, несмотря на большую разрядность по сравнению с типом `double`, тип `double` может хранить большее значение. Однако значение `decimal` может содержать до 28 знаков после запятой, тогда как значение типа `double` — 15–16 знаков после запятой.

`Decimal` чаще находит применение в финансовых вычислениях, тогда как `double` — в математических операциях.

Основную массу базовых (примитивных) типов составляют числовые.

Нечисловыми являются лишь логический тип `bool` и символьный `char`, да и тот представляет собой специальный числовой тип.

Тип `int` имеет некоторое идеологическое преимущество, которое основывается в первую очередь на правилах автоматического преобразования типов.

Среди двух типов (`float` и `double`), предназначенных для работы с действительными числами, приоритет остается за типом `double`: во-первых, диапазон допустимых значений у этого типа шире, а во-вторых, по умолчанию числа с плавающей точкой интерпретируются как `double`-значения.

Тип `decimal` предназначен для выполнения расчетов, в которых критичны ошибки округления. Обычно это финансовые расчеты.

Данные типа `char` — буква (или управляющие символы). Другими словами, значением переменной типа `char` может быть буква. В отличие от текста (объект класса `string`), который заключается в двойные кавычки, отдельный символ заключается в одинарные кавычки.

Если отдельный символ заключить в двойные кавычки, это уже будет текст, состоящий из одного символа. Например, `'A'` — это символьное значение (тип `char`), а `"A"` — текстовое значение (тип `string`).

Кроме непосредственно букв есть еще управляющие символы (последовательности символов):

`\n` — переход к новой строке;

`\t` — табуляция.

Каждая из этих конструкций считается одним символом — во всяком случае, соответствующее значение можно записать в переменную типа `char`.

`\a` — позволяет сгенерировать «бип» — программный писк;

`\'` — одинарная кавычка;

`\"` — двойные кавычки;

`\\` — косая черта;

`\b` — курсор вывода переводится на одну позицию назад.

Переменные логического типа (типа `bool`) могут принимать всего два значения: `true` (истина) и `false` (ложь). Обычно значения логического типа используются в условных операторах для проверки условий.

Консольный вывод

Для вывода информации на консоль используется встроенный метод `Console.WriteLine()` (рис. 1.4), т. е. если требуется

вывести некоторую информацию на консоль, то необходимо передать ее в метод `Console.WriteLine()`.

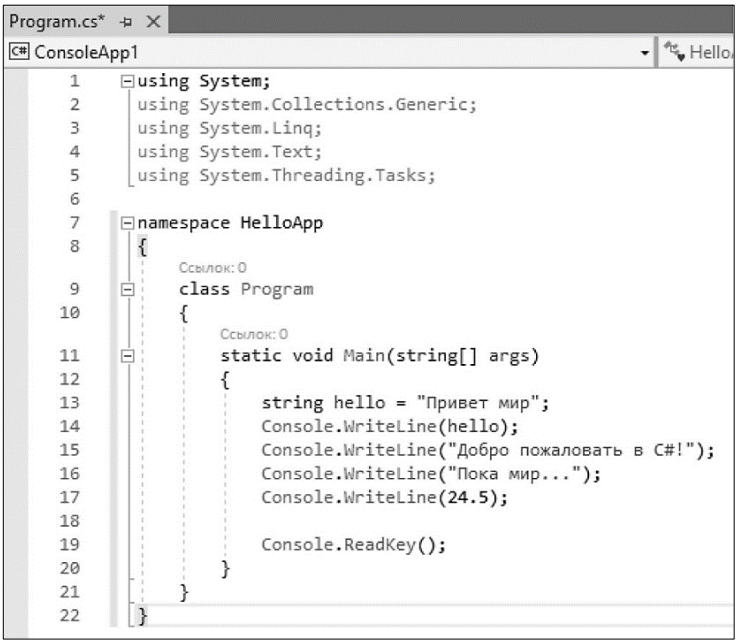

Рис. 1.4

Консольный вывод (рис. 1.5).

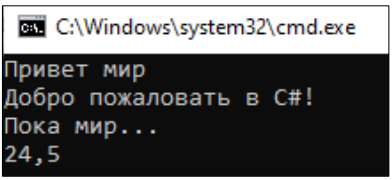

Рис. 1.5

Нередко возникает необходимость вывести на консоль в одной строке значения сразу нескольких переменных (рис. 1.6). В этом случае мы можем использовать прием, который называется интерполяцией.

```
Program.cs*
ConsoleApp1                                   HelloApp.Program
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Threading.Tasks;
 6
 7  namespace HelloApp
 8  {
        Ссылок: 0
 9      class Program
10      {
            Ссылок: 0
11          static void Main(string[] args)
12          {
13              string name = "Tom";
14              int age = 34;
15              double height = 1.7;
16              Console.WriteLine($"Имя: {name}  Возраст: {age}  Рост: {height}м");
17
18              Console.ReadKey();
19          }
20      }
21  }
```

Рис. 1.6

Для встраивания отдельных значений в выводимую на консоль строку используются фигурные скобки, в которые заключается встраиваемое значение. Это может быть значение переменной (`{name}`) или более сложное выражение (например, операция сложения `{4+7}`). А перед всей строкой ставится знак доллара `$`.

При выводе на консоль вместо помещенных в фигурные скобки выражений будут выводиться их значения (рис. 1.7).

```
C:\Windows\system32\cmd.exe
Имя: Tom  Возраст: 34  Рост: 1,7м
```

Рис. 1.7

Есть и другой способ вывода на консоль сразу нескольких значений (рис. 1.8).

Этот способ подразумевает, что первый параметр в методе `Console.WriteLine()` представляет выводимую строку (`"Имя: {0} Возраст: {2} Рост: {1}м"`). Все последующие параметры представляют значения, которые могут быть встроены в эту строку (`name, height, age`). При этом важен порядок подобных

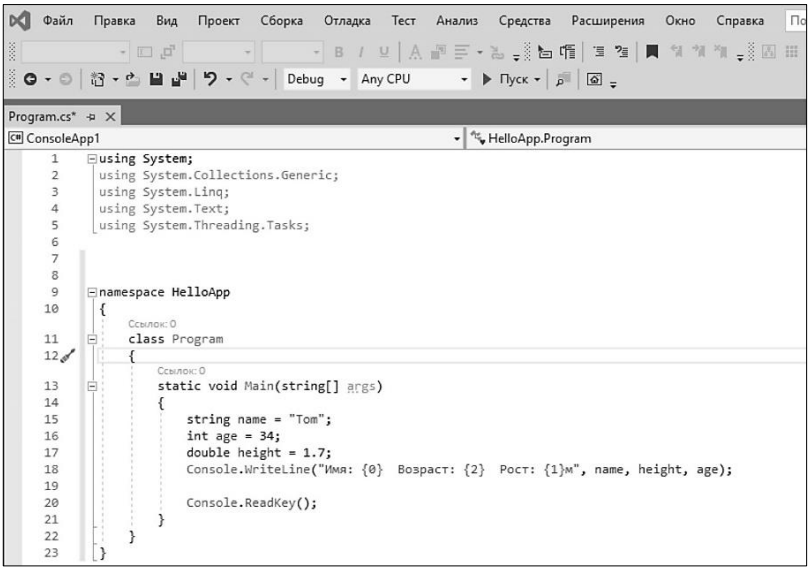

Рис. 1.8

параметров. Например, в данном случае вначале идет `name`, затем `height` и уже потом `age`. Поэтому значение переменной `name` будет представлять параметр с номером 0 (нумерация начинается с нуля), `height` имеет номер 1, а `age` — номер 2. Соответственно в строке `"Имя: {0} Возраст: {2} Рост: {1}м"` на место плейсхолдеров `{0}`, `{2}`, `{1}` будут вставляться значения соответствующих параметров.

Кроме `Console.WriteLine()` можно также использовать метод `Console.Write()`. Он работает точно так же, за тем исключением, что не осуществляет переход на следующую строку.

Консольный ввод

Кроме вывода информации на консоль мы можем получать информацию с консоли. Для этого предназначен метод `Console.ReadLine()`. Он позволяет получить введенную строку (рис. 1.9).

В данном случае все, что вводит пользователь, с помощью метода `Console.ReadLine()` передается в переменную `name`.

```
Program.cs*
ConsoleApp1
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7
8   namespace HelloApp
9   {
        Ссылок: 0
10      class Program
11      {
            Ссылок: 0
12          static void Main(string[] args)
13          {
14              Console.Write("Введите свое имя: ");
15              string name = Console.ReadLine();
16              Console.WriteLine($"Привет {name}");
17
18              Console.ReadKey();
19          }
20      }
21  }
```

Рис. 1.9

Пример работы программы показан на рис. 1.10.

```
C:\Windows\system32\cmd.exe
Введите свое имя: Иван
Привет Иван
```

Рис. 1.10

Таким образом мы можем вводить информацию через консоль. Однако минусом этого метода является то, что `Console.ReadLine()` считывает информацию именно в виде строки. Поэтому мы можем по умолчанию присвоить ее только переменной типа `string`. Как быть, если, допустим, мы хотим ввести возраст в переменную типа `int` или другую информацию в переменные типа `double` или `decimal`? По умолчанию платформа .NET предоставляет ряд методов, которые позволяют преобразовать различные значения к типам `int`, `double` и т. д.

Некоторые из этих методов:

```
Convert.ToInt32() (преобразует к типу int)
Convert.ToDouble() (преобразует к типу double)
Convert.ToDecimal() (преобразует к типу decimal)
```

Пример ввода значений показан на рис. 1.11.

```
7
8   namespace HelloApp
9   {
        Ссылок: 0
10      class Program
11      {
            Ссылок: 0
12          static void Main(string[] args)
13          {
14              Console.Write("Введите имя: ");
15              string name = Console.ReadLine();
16
17              Console.Write("Введите возраст: ");
18              int age = Convert.ToInt32(Console.ReadLine());
19
20              Console.Write("Введите рост: ");
21              double height = Convert.ToDouble(Console.ReadLine());
22
23              Console.Write("Введите размер зарплаты: ");
24              decimal salary = Convert.ToDecimal(Console.ReadLine());
25
26              Console.WriteLine($"Имя: {name}  Возраст: {age}  Рост: {height}м  Зарплата: {salary}$");
27
28              Console.ReadKey();
29          }
30      }
```

Рис. 1.11

При вводе важно учитывать текущую операционную систему. В одних культурах разделителем между целой и дробной частью является точка (США, Великобритания и др.), в других — запятая (Россия, Германия и др.). Например, если текущая операционная система русскоязычная, значит, надо вводить дробные числа с разделителем запятой. Если локализация англоязычная, значит, разделителем целой и дробной части при вводе будет точка.

Пример работы программы показан на рис. 1.12.

```
C:\Windows\system32\cmd.exe
Введите имя:Иван
Введите возраст:25
Введите рост в метрах:1,85
Введите размер зарплаты:1500
Имя: Иван Возраст: 25 Рост: 1,85м Зарплата: 1500$
```

Рис. 1.12

Основные операторы языка C#

Что касается основных операторов языка C#, то их традиционно делят на четыре группы:

– арифметические операторы, используемые в основном для выполнения операций с числовыми данными;

– операторы сравнения, которые позволяют сравнивать значения переменных;

– логические операторы, предназначенные для выполнения логических операций;

– побитовые, или поразрядные, операторы — группа операторов, которые позволяют выполнять преобразование на уровне побитового представления чисел.

Арифметические операторы представлены в табл. 1.3.

Таблица 1.3

Арифметические операторы C#

Оператор	Описание
+	Сложение: бинарный оператор. В результате вычисления выражения вида A+B в качестве результата возвращается сумма значений числовых переменных A и B. Если переменные текстовые, результатом является строка, полученная объединением текстовых значений переменных
−	Вычитание: бинарный оператор. В результате вычисления выражения вида A−B в качестве результата возвращается разность значений числовых переменных A и B. Оператор может также использоваться как унарный (перед переменной, например −A)
*	Умножение: бинарный оператор. В результате вычисления выражения вида A*B в качестве результата возвращается произведение значений числовых переменных A и B
/	Деление: бинарный оператор. В результате вычисления выражения вида A/B в качестве результата возвращается частное значений числовых переменных A и B. Если операнды целочисленные, деление выполняется нацело. Для вычисления результата на множестве действительных чисел (при целочисленных операндах) можно использовать команду в виде (double)A/B
%	Остаток от деления: бинарный оператор. Оператор применим не только к целочисленным операндам, но и к действительным числам. В результате вычисления выражения A%B возвращается остаток от целочисленного деления значения переменной A на значение переменной B

Окончание табл. 1.3

Оператор	Описание
++	Инкремент: унарный оператор. В результате вычисления выражения ++A (префиксная форма оператора инкремента) или A++ (постфиксная форма оператора инкремента) значение переменной A увеличивается на единицу. Оператор возвращает результат. Префиксная форма оператора инкремента возвращает новое (увеличенное на единицу) значение переменной. Постфиксная форма оператора инкремента возвращает старое значение переменной (значение переменной до увеличения на единицу)
--	Декремент: унарный оператор. В результате вычисления выражения --A (префиксная форма оператора инкремента) или A-- (постфиксная форма оператора инкремента) значение переменной A уменьшается на единицу. Оператор возвращает результат. Префиксная форма оператора инкремента возвращает новое (уменьшенное на единицу) значение переменной. Постфиксная форма оператора инкремента возвращает старое значение переменной (значение переменной до уменьшения на единицу)

Операции присваивания

На практике достаточно часто используются так называемые составные (или сокращенные) операторы присваивания, в которые, кроме прочего, могут входить и представленные выше бинарные операторы.

Операции присвоения устанавливают значение. В операциях присвоения участвуют два операнда, причем левый операнд может представлять только модифицируемое именованное выражение, например, переменную.

Как и во многих других языках программирования, в C# имеется базовая операция присваивания =, которая присваивает значение правого операнда левому операнду:

```
int number = 23;
```

Здесь переменной `number` присваивается число 23. Переменная `number` представляет левый операнд, которому присваивается значение правого операнда, т. е. числа 23.

Также можно выполнять множественное присваивание сразу нескольким переменным одновременно:

```
int a, b, c;
a = b = c = 34;
```

Стоит отметить, что операции присваивания имеют низкий приоритет. Вначале будет вычисляться значение правого операнда и только потом будет идти присваивание этого значения левому операнду. Например:

```
int a, b, c;
a = b = c = 34 * 2 / 4; // 17
```

Сначала будет вычисляться выражение 34 * 2 / 4, затем полученное значение будет присвоено переменным.

Кроме базовой операции присваивания в C# есть еще ряд операций:

+= — *присваивание после сложения*. Присваивает левому операнду сумму левого и правого операндов: выражение A += B равнозначно выражению A = A + B.

-= — *присваивание после вычитания.* Присваивает левому операнду разность левого и правого операндов: A -= B эквивалентно A = A−B

***=** — *присваивание после умножения.* Присваивает левому операнду произведение левого и правого операндов: A *= B эквивалентно A = A * B

/= — *присваивание после деления.* Присваивает левому операнду частное левого и правого операндов: A /= B эквивалентно A = A / B

%= — *присваивание после деления по модулю.* Присваивает левому операнду остаток от целочисленного деления левого операнда на правый: A %= B эквивалентно A = A % B

&= — *присваивание после поразрядной конъюнкции*. Присваивает левому операнду результат поразрядной конъюнкции его битового представления с битовым представлением правого операнда: A &= B эквивалентно A = A & B

|= — *присваивание после поразрядной дизъюнкции*. Присваивает левому операнду результат поразрядной дизъюнкции его битового представления с битовым представлением правого операнда: A |= B эквивалентно A = A | B

^= — *присваивание после операции исключающего ИЛИ*. Присваивает левому операнду результат операции исключаю-

щего ИЛИ его битового представления с битовым представлением правого операнда: A ^= B эквивалентно A = A ^ B

<<= — *присваивание после сдвига разрядов влево*. Присваивает левому операнду результат сдвига его битового представления влево на определенное количество разрядов, равное значению правого операнда: A <<= B эквивалентно A = A << B

>>= — *присваивание после сдвига разрядов вправо*. Присваивает левому операнду результат сдвига его битового представления вправо на определенное количество разрядов, равное значению правого операнда: A >>= B эквивалентно A = A >> B

Применение операций присваивания:

```
int a = 10;
a += 10; // 20
a -= 4; // 16
a *= 2; // 32
a /= 8; // 4
a <<= 4; // 64
a >>= 2; //
```

+ — *операция сложения двух чисел* (рис. 1.13, 1.14).

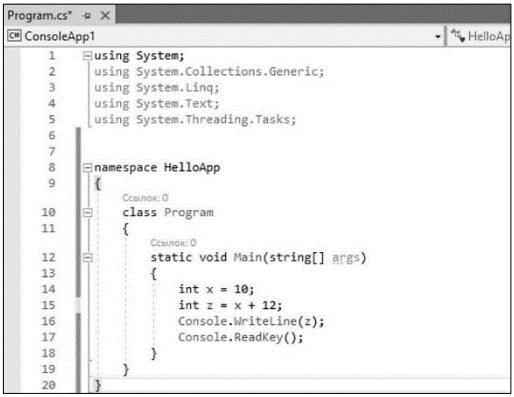

Рис. 1.13. Сложение двух чисел

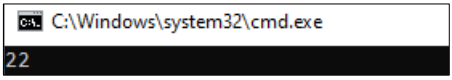

Рис. 1.14. Консольный вывод

- — *операция вычитания двух чисел:*

```
int x = 10;
int z = x - 6; // 4
```

***** — *операция умножения двух чисел:*

```
int x = 10;
int z = x * 5; // 50
```

/ — *операция деления двух чисел* (рис. 1.15, 1.16):

```
int x = 10;
int z = x / 5; // 2
double a = 10;
double b = 3;
double c = a / b; // 3.33333333
```

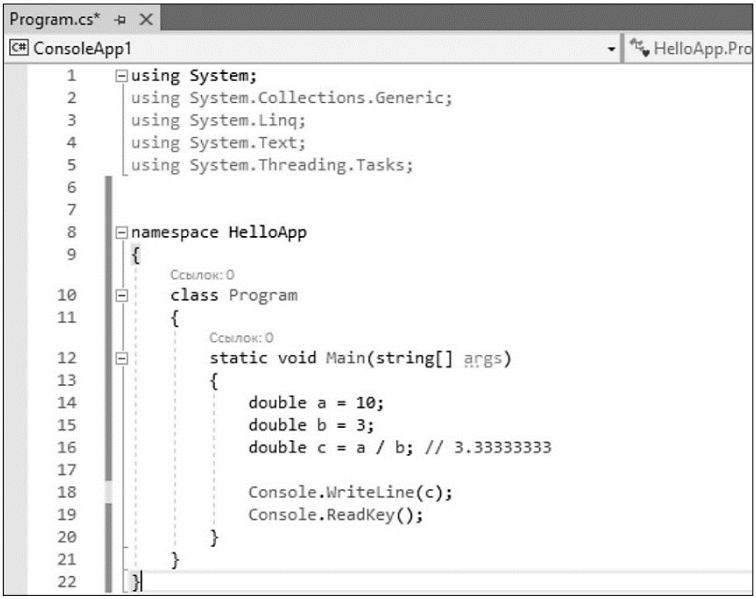

Рис. 1.15

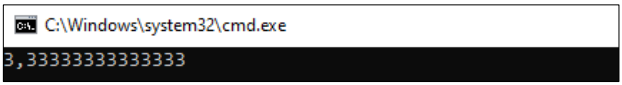

Рис. 1.16

При делении стоит учитывать, что если оба операнда представляют целые числа, то результат также будет округляться до целого числа:

```
double z = 10 / 4; // результат равен 2
```

Несмотря на то что результат операции в итоге помещается в переменную типа `double`, которая позволяет сохранить дробную часть, в самой операции участвуют два литерала, которые по умолчанию рассматриваются как объекты `int`, т. е. целые числа, и результат то же будет целочисленный.

Для выхода из этой ситуации необходимо определять литералы или переменные, участвующие в операции, именно как типы `double` или `float`:

```
double z = 10.0 / 4.0; // результат равен 2.5
```

% — *операция получения остатка от целочисленного деления двух чисел:*

```
double x = 10.0;
double z = x % 4.0; //результат равен 2
```

++ — *операция инкремента*

Инкремент бывает префиксным: `++x` — сначала значение переменной `x` увеличивается на 1, а потом ее значение возвращается в качестве результата операции (рис. 1.17, 1.18).

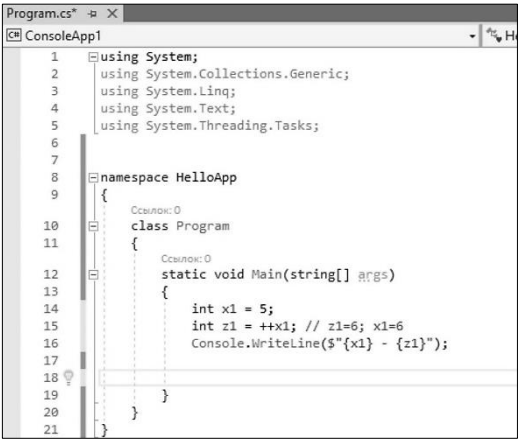

Рис. 1.17

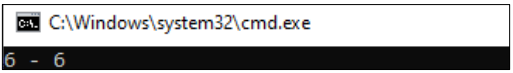

Рис. 1.18

Также существует постфиксный инкремент: `x++` — сначала значение переменной `x` возвращается в качестве результата операции, а затем к нему прибавляется 1 (рис. 1.19, 1.20).

```
int x2 = 5;
int z2 = x2++; // z2=5; x2=6
Console.WriteLine($"{x2} - {z2}");
```

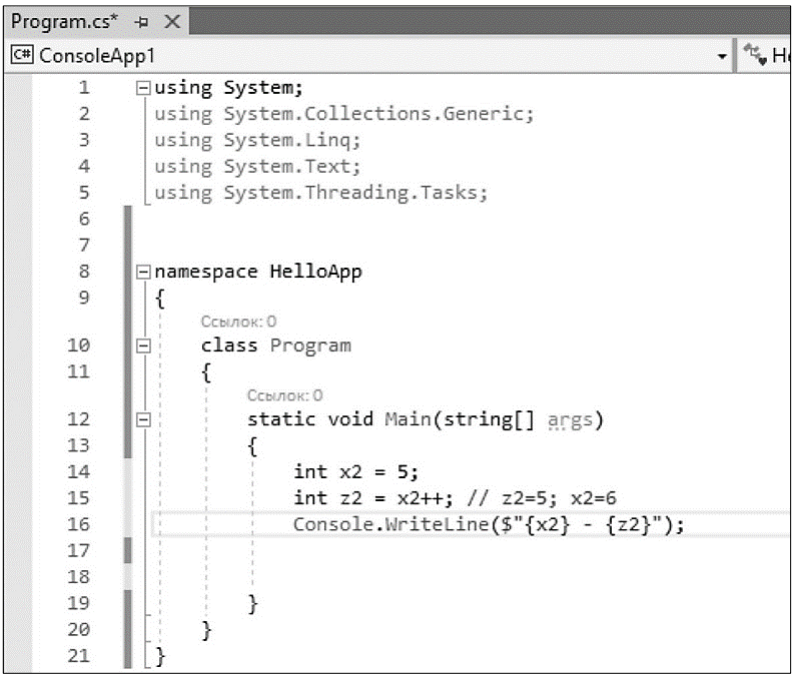

Рис. 1.19

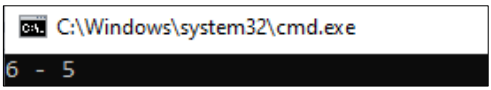

Рис. 1.20

-- — *операция декремента* или уменьшения значения на единицу. Также существует префиксная форма декремента (`--x`) и постфиксная форма (`x--`) (рис. 1.21, 1.22).

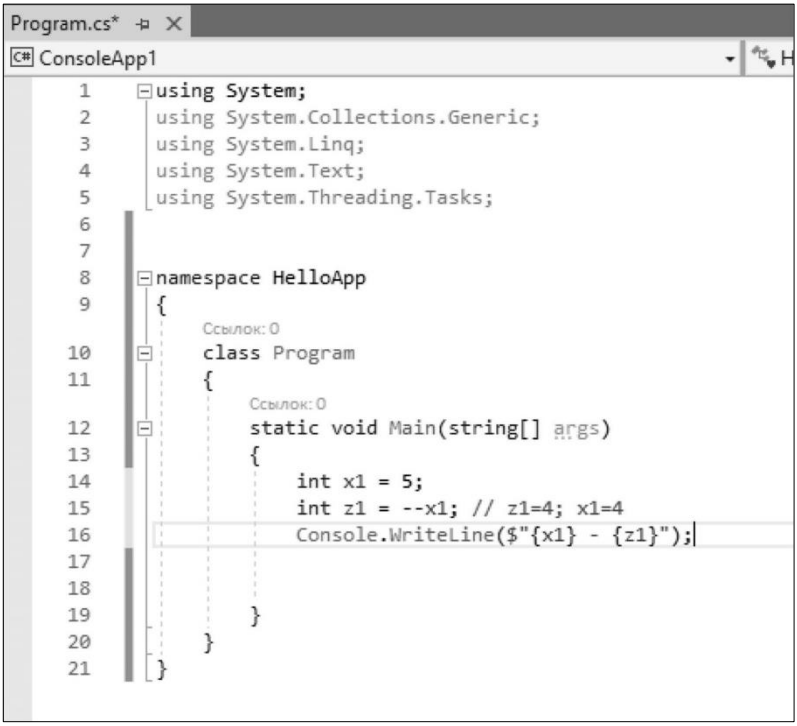

Рис. 1.21

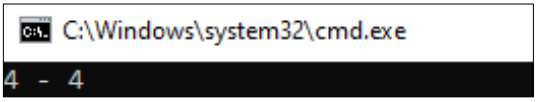

Рис. 1.22

При выполнении сразу нескольких арифметических операций следует учитывать порядок их выполнения. Приоритет операций от наивысшего к низшему:

– инкремент, декремент;

– умножение, деление, получение остатка;

– сложение, вычитание.

Для изменения порядка следования операций применяются скобки.

Рассмотрим набор операций (рис. 1.23, 1.24)

```
Program.cs*
ConsoleApp1                                   HelloApp.Progra
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Threading.Tasks;
 6
 7
 8  namespace HelloApp
 9  {
        Ссылок: 0
10      class Program
11      {
            Ссылок: 0
12          static void Main(string[] args)
13          {
14              int a = 3;
15              int b = 5;
16              int c = 40;
17              int d = c-- - b * a;    // a=3  b=5  c=39  d=25
18              Console.WriteLine($"a={a}  b={b}  c={c}  d={d}");
19
20
21          }
22      }
23  }
```

Рис. 1.23

```
C:\Windows\system32\cmd.exe
a=3  b=5  c=39  d=25
```

Рис. 1.24

Здесь мы имеем дело с тремя операциями: декремент, вычитание и умножение. Сначала выполняется декремент переменной `c`, затем умножение `b*a`, и в конце вычитание. То есть фактически набор операций выглядел так:

```
int d = (c--)-(b*a);
```

Но с помощью скобок мы могли бы изменить порядок операций, например, следующим образом:

```
int a = 3;
int b = 5;
int c = 40;
int d = (c-(--b))*a; // a=3 b=4 c=40 d=108
Console.WriteLine($"a={a} b={b} c={c} d={d}");
```

Операторы сравнения приведены в табл. 1.4.

Таблица 1.4

Операторы сравнения C#

Оператор	Описание
==	Равно: результатом выражения A==B является логическое значение true, если значения переменных A и B одинаковы, и false в противном случае
!=	Не равно: результатом выражения A!=B является логическое значение true, если значения переменных A и B разные, и false в противном случае
>	Больше: результатом выражения A>B является логическое значение true, если значение переменой A больше, чем значение переменной B, и false в противном случае
<	Меньше: результатом выражения A<B является логическое значение true, если значение переменой A меньше, чем значение переменной B, и false в противном случае
>=	Больше или равно: результатом выражения A>=B является логическое значение true, если значение переменой A не меньше, чем значение переменной B, и false в противном случае
<=	Меньше или равно: результатом выражения A<=B является логическое значение true, если значение переменой A не больше, чем значение переменной B, и false в противном случае

Результатом выражения с оператором сравнения является логическое значение (`true` или `false`). Такие выражения могут сами входить, как составная часть, в логические выражения. Операндами логических выражений являются значения логического типа.

== — *равно:*

```
int a = 10;
int b = 4;
bool c = a == b; // false
```

!= — *не равно*:

```
int a = 10;
int b = 4;
bool c = a != b; // true
bool d = a!=10; // false
```

< — *меньше чем*:

```
int a = 10;
int b = 4;
bool c = a < b; // false
```

> — *больше чем*:

```
int a = 10;
int b = 4;
bool c = a > b; // true
bool d = a > 25; // false
```

<= — *меньше или равно*:

```
int a = 10;
int b = 4;
bool c = a <= b; // false
bool d = a <= 25; // true
```

>= — *больше или равно*:

```
int a = 10;
int b = 4;
bool c = a >= b; // true
bool d = a >= 25; // false
```

Операции <, >, <=, >= имеют больший приоритет, чем == и !=.

Логические операторы приведены в табл. 1.5.

Таблица 1.5

Логические операторы C#

Оператор	Описание
&	Оператор логического «и» (бинарный). Результатом выражения A&B является логическое значение true, если оба логических операнда A и B равны true. Если хотя бы один из операндов равен false, результатом будет false
|	Оператор логического «или» (бинарный). Результатом выражения A|B является логическое значение true, если хотя бы один из логических операндов A и B равен true. Если оба операнда равны false, результатом будет false

Окончание табл. 1.5

Оператор	Описание
^	Оператор логического «исключающего или» (бинарный). Результатом выражения A^B является логическое значение true, если один из логических операндов A или B равен true, а другой равен false. Если оба операнда равны true или оба false, результатом будет false
&&	Сокращенная форма логического оператора «и». Отличие от обычной формы логического оператора «и» состоит в том, что при вычислении выражения A&&B второй операнд B вычисляется, только если первый операнд A равен true. Если первый операнд A равен false, то в качестве результата выражения A&&B возвращается значение false без вычисления второго операнда B
||	Сокращенная форма логического оператора «или». Отличие от обычной формы логического оператора «или» состоит в том, что при вычислении выражения A||B второй операнд B вычисляется, только если первый операнд A равен false. Если первый операнд A равен true, то в качестве результата выражения A||B возвращается значение true без вычисления второго операнда B
!	Оператор логического отрицания (унарный). Результатом выражения !A является значение true, если операнд A false. Если операнд A равен true, результатом выражения !A возвращается значение false

| — *операция логического сложения или логическое ИЛИ*:

```
bool x1 = (5 > 6) | (4 < 6); // 5 > 6 — false, 4 < 6 — true,
поэтому возвращается true
bool x2 = (5 > 6) | (4 > 6); // 5 > 6 — false, 4 > 6 —
false, поэтому возвращается false
```

& — *операция логического умножения или логическое И*:

```
bool x1 = (5 > 6) & (4 < 6); // 5 > 6 — false, 4 < 6 — true,
поэтому возвращается false
bool x2 = (5 < 6) & (4 < 6); // 5 < 6 — true, 4 < 6 — true,
поэтому возвращается true
```

|| — *операция логического сложения:*

```
bool x1 = (5 > 6) || (4 < 6); // 5 > 6 — false, 4 < 6 —
true, поэтому возвращается true
bool x2 = (5 > 6) || (4 > 6); // 5 > 6 — false, 4 > 6 —
false, поэтому возвращается false
```

&& — *операция логического умножения*:

```
bool x1 = (5 > 6) && (4 < 6); // 5 > 6 — false, 4 < 6 — true,
поэтому возвращается false
```

```
bool x2 = (5 < 6) && (4 < 6); // 5 < 6 — true, 4 < 6 — true,
поэтому возвращается true
```

! — *операция логического отрицания:*

```
bool a = true;
bool b = !a; // false
```

^ — *операция исключающего ИЛИ:*

```
bool x5 = (5 > 6) ^ (4 < 6); // 5 > 6 — false, 4 < 6 — true,
поэтому возвращается true
bool x6 = (50 > 6) ^ (4 / 2 < 3); // 50 > 6 — true, 4/2 < 3 —
true, поэтому возвращается false
```

Две пары операций | и ||, а также & и && выполняют похожие действия, однако они не равнозначны.

В выражении `z=x|y;` будут вычисляться оба значения — `x` и `y`.

В выражении `z=x||y;` сначала будет вычисляться значение `x`, и если оно равно `true`, то вычисление значения `y` уже смысла не имеет, так как в любом случае `z` будет равно `true`. Значение `y` будет вычисляться только в том случае, если `x` равно `false`.

Аналогично в выражении `z=x&y;` будут вычисляться оба значения — `x` и `y`.

В выражении `z=x&&y;` сначала будет вычисляться значение `x`, и если оно равно `false`, то вычисление значения `y` уже смысла не имеет, так как в любом случае `z` будет равно `false`. Значение `y` будет вычисляться только в том случае, если `x` равно `true`.

Операции || и && более удобны в вычислениях, так как позволяют сократить время на вычисление значения выражения и тем самым повышают производительность. А операции | и & больше подходят для выполнения поразрядных операций над числами.

Условные конструкции

Условные конструкции — один из базовых компонентов многих языков программирования; они направляют работу программы по одному из путей в зависимости от определенных условий.

В языке C# используются следующие условные конструкции: `if..else` и `switch..case`.

Конструкция if..else

Конструкция `if..else` проверяет истинность некоторого условия и в зависимости от результатов проверки выполняет определенный код (рис. 1.25, 1.26).

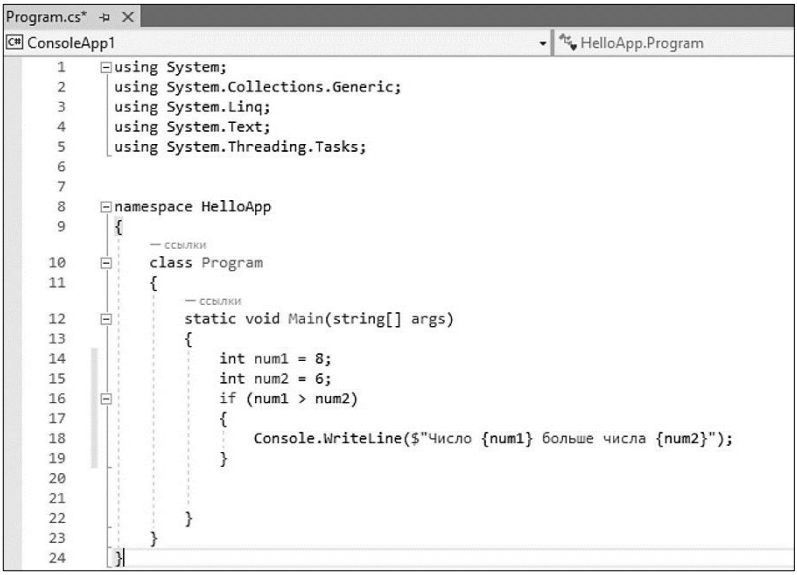

Рис. 1.25

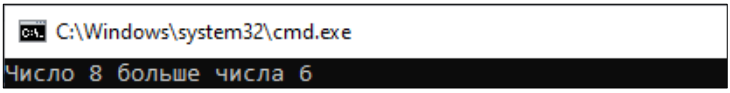

Рис. 1.26

После ключевого слова `if` ставится условие. И если это условие выполняется, то срабатывает код, который помещен далее в блоке `if` после фигурных скобок. В качестве условий выступают ранее рассмотренные операции сравнения.

В данном случае (см. рис. 1.25) у нас первое число больше второго, поэтому выражение `num1 > num2` истинно и возвращает

`true`, следовательно, управление переходит к строке `Console.WriteLine("Число {num1} больше числа {num2}");`

Если мы захотим, чтобы при несоблюдении условия также выполнялись какие-либо действия, мы можем добавить блок `else` (рис. 1.27, 1.28).

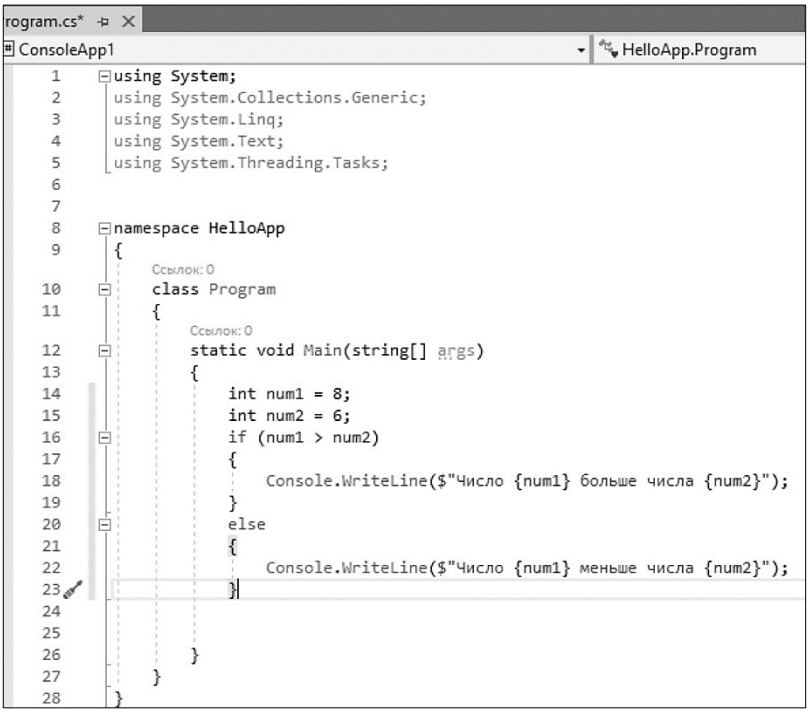

Рис. 1.27

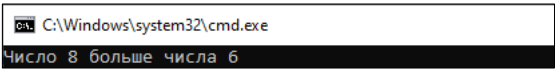

Рис. 1.28

Но при сравнении чисел возможны три состояния: первое число больше второго, первое число меньше второго и числа равны. Используя *конструкцию* `else..if`, мы можем обрабатывать дополнительные условия (рис. 1.29, 1.30).

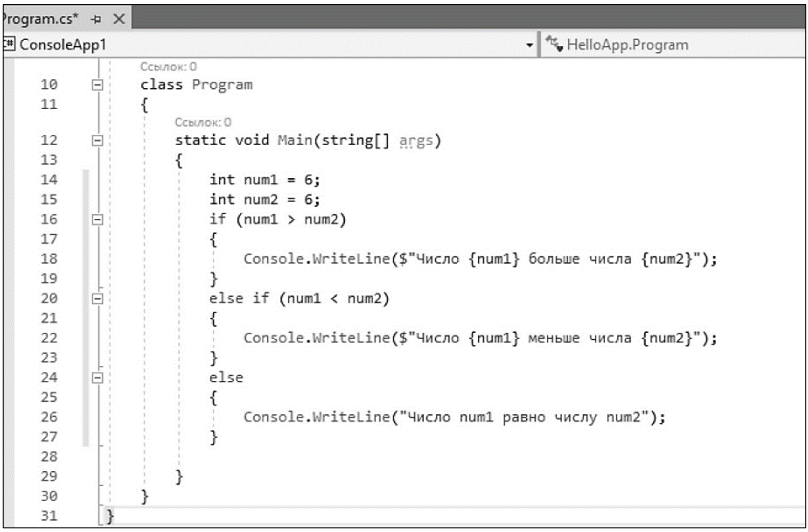

Рис. 1.29

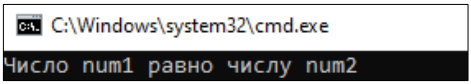

Рис. 1.30

Также можно соединить сразу несколько условий, используя логические операторы:

```
int num1 = 8;
int num2 = 6;
if(num1 > num2 && num1==8)
{
  Console.WriteLine($"Число {num1} больше числа {num2}");
}
```

В данном случае блок `if` будет выполняться, если `num1 > num2` равно `true` и `num1==8` равно `true`.

Конструкция switch

Конструкция `switch..case` аналогична конструкции `if..else`, так как позволяет обработать сразу несколько условий (рис. 1.31, 1.32).

```
rogram.cs
ConsoleApp1                                    HelloApp.Program
    using System.Text;
    using System.Threading.Tasks;


    namespace HelloApp
    {
        Ссылок: 0
        class Program
        {
            Ссылок: 0
            static void Main(string[] args)
            {
                Console.WriteLine("Нажмите Y или N");
                string selection = Console.ReadLine();
                switch (selection)
                {
                    case "Y":
                        Console.WriteLine("Вы нажали букву Y");
                        break;
                    case "N":
                        Console.WriteLine("Вы нажали букву N");
                        break;
                    default:
                        Console.WriteLine("Вы нажали неизвестную букву");
                        break;
                }


            }
        }
```

Рис. 1.31

```
C:\Windows\system32\cmd.exe
Нажмите Y или N
N
Вы нажали букву N
```

Рис. 1.32

После ключевого слова `switch` в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после оператора `case`. И если совпадение будет найдено, то будет выполняться определенный блок `case`.

В конце каждого блока `case` должен ставиться один из операторов перехода: `break`, `goto case`, `return` или `throw`. Как правило, используется оператор `break`. При его применении другие блоки `case` выполняться не будут.

Однако если необходимо, чтобы, наоборот, после выполнения текущего блока `case` выполнялся другой блок `case`, то мы можем использовать вместо `break` оператор `goto case` (рис. 1.33, 1.34).

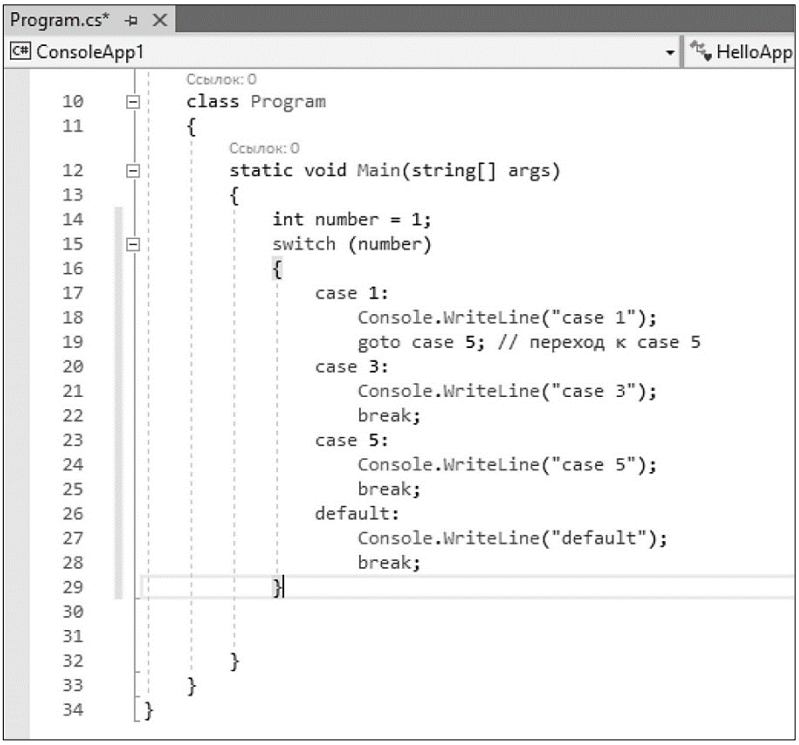

Рис. 1.33

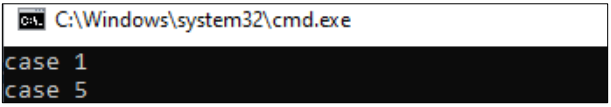

Рис. 1.34

Если мы хотим также обработать ситуацию, когда совпадения не будет найдено, то можно добавить блок `default`, как в примере выше.

Применение оператора `return` позволит выйти не только из блока `case`, но и из вызывающего метода. То есть, если в методе `Main` после конструкции `switch..case`, в которой используется оператор `return`, идут какие-либо операторы и выражения, то они выполняться не будут, а метод `Main` завершит работу.

Оператор `throw` применяется для выброса ошибок.

Тернарная операция

Тернарная операция имеет следующий синтаксис:

```
[первый операнд — условие] ? [второй операнд] : [третий операнд].
```

Здесь сразу три операнда. В зависимости от условия тернарная операция возвращает второй или третий операнд: если условие равно `true`, то возвращается второй операнд; если условие равно `false`, то третий (рис. 1.35, 1.36).

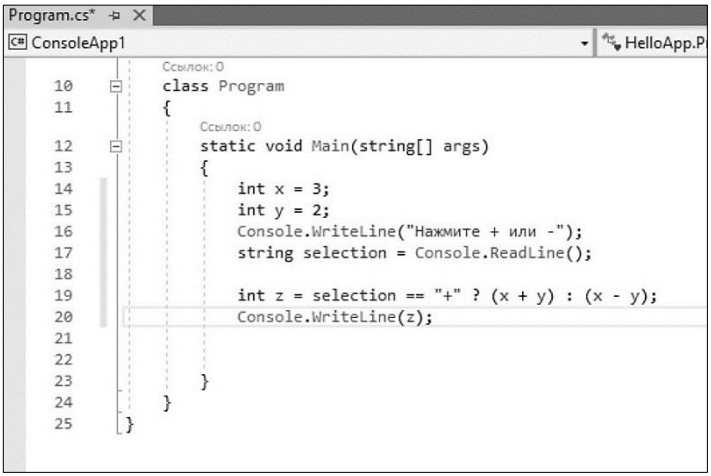

Рис. 1.35

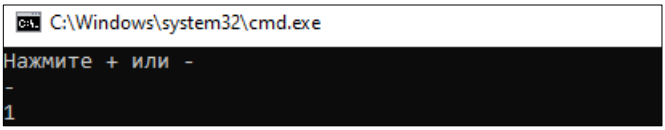

Рис. 1.36

Здесь результатом тернарной операции является переменная z. Если мы выше вводим "+", то z будет равно второму операнду — (x+y). Иначе z будет равно третьему операнду.

Выводы по главе 1

Существующие веб-технологии являются разновидностью интернет-технологий. Веб-технологии — это интернет-сервисы (WWW — Всемирная паутина); это работа в Интернет (браузеры, поисковые системы, просмотр страниц в браузере); это информационные ресурсы Интернет (веб-страницы, интернет-магазины, интернет-порталы, URL и протоколы передачи данных, адресация, создание сайтов, языки веб-программирования). Основные понятия веб-технологий — веб-страница и веб-сайт. Основная цель в изучении веб-технологий — создание или изменение веб-страниц, которые будут правильно отражаться в браузерах.

В настоящее время существует множество платформ для разработки веб-приложений. Наибольший интерес представляет платформа ASP.NET.

Современная версия ASP.NET реализована как ASP.NET Core MVC. Инфраструктура ASP.NET Core MVC генерирует ясную и соответствующую стандартам разметку. Она поощряет создание простой и элегантной разметки, стилизованной с помощью CSS. Работает в гармонии с HTTP. Дает контроль над запросами, передаваемыми между браузером и сервером, что позволяет точно настраивать пользовательский интерфейс по своему усмотрению. Инфраструктура ASP.NET Core является межплатформенной, как в отношении разработки, так и в плане развертывания. Она имеет открытый код. Большая часть разработки приложений ASP.NET Core MVC осуществляется с применением Visual Studio и, что очень важно, с использованием нового межплатформенного инструмента разработки под названием Visual Studio Code.

Концептуально показана основа ASP.NET и инструменты программирования. Рассмотрен веб-сервер IIS, его установка и настройка и возможности публикации проектов и веб-сайтов на локальном сервере и в облачных реализациях.

Проанализированы виды веб-проектов, инструменты их создания, особенности Web Forms и инструменты их создания.

Определена инфраструктура проектов ASP.NET MVC и база подобной технологии — модель, контроллер, представление.

Представлены базовые компоненты языка программирования C#.

Вопросы для самоконтроля

1. Что такое Интернет? Возможности интернет-технологий. Веб-технологии и их применение.
2. Понятие платформы для разработки в веб-приложения и необходимые компоненты для современной разработки сайтов.
3. Понятие веб-сервера, разновидности, сложность использования. Серверы Microsoft.
4. Отличия проектов в Visual Studio. Особенности кодирования веб-приложения и веб-сайта.
5. Рекомендация выбора веб-приложения. Особенности и преимущества веб-сайтов.
6. Язык программирования HTML. Возникновение и история развития.
7. Основные компоненты и инфраструктура ASP.NET MVC.
8. Взаимосвязь модели, контроллера и представления веб-проекта.
9. Маршрутизация и веб-адреса шаблона ASP.NET MVC.
10. Язык программирования C#. Возникновение и история развития. Преимущества и недостатки C#.

Глава 2

Среда разработки Visual Studio

2.1. Назначение Visual Studio

Visual Studio — это линейка продуктов фирмы Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментов.

Visual Studio позволяет разрабатывать как консольные приложения, так и игры и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, UWP а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, .NET Core, .NET, MAUI, Xbox, Windows Phone, .NET Compact Framework и Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода.

Встроенный отладчик может работать как отладчик уровня исходного кода и как отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных.

Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов

(например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

В учебном пособии будем изучать веб-технологии в среде Visual Studio 2019.

2.2. Окно запуска

При запуске Visual Studio 2019 появляется изображение, показанное на рис. 2.1.

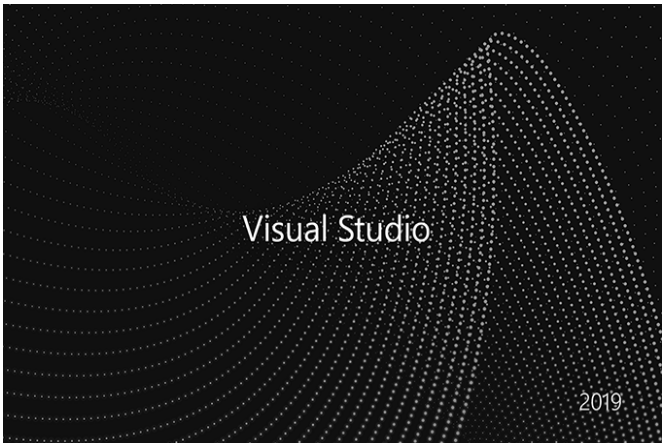

Рис. 2.1

Далее появляется начальное окно запуска Visual Studio 2019 (рис. 2.2).

Окно запуска помогает быстрее выбрать имеющиеся структуры веб-технологий, созданные ранее и вновь создаваемые, или просто открыть папку с файлами кода.

Начальное окно позволяет выбрать «Работу с репозитарием», «Открыть имеющийся проект», «Открыть локальную папку» или «Создание проекта».

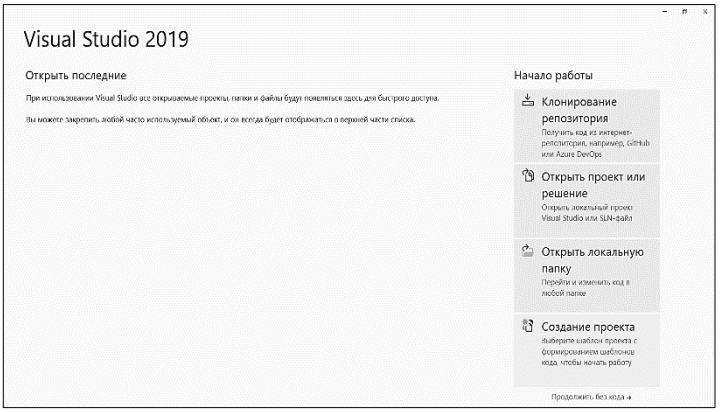

Рис. 2.2

Будем полагать, что Visual Studio 2019 запущена впервые и нам необходимо создать проект как среду веб-технологии.

2.3. Создание проекта

«Щелкнув мышкой», перейдем к функции Visual Studio 2019 *Создание проекта* (рис. 2.3).

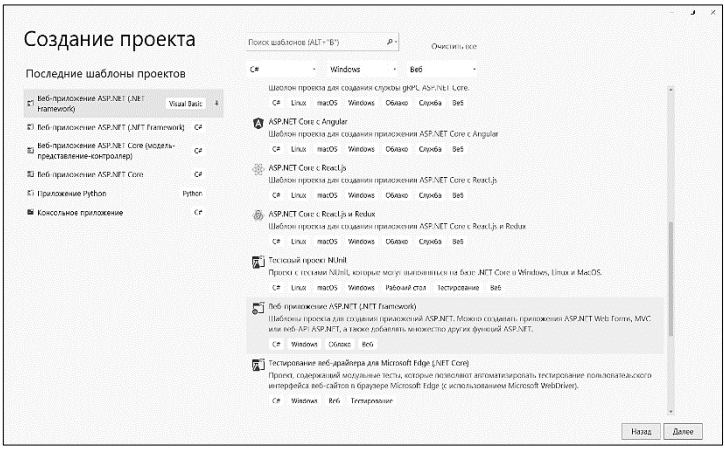

Рис. 2.3

В левой части окна показаны шаблоны проектов (рис. 2.4).

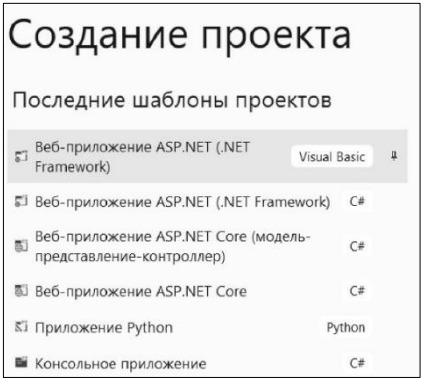

Рис. 2.4

В правой части окна показаны языки программирования, тип веб-технологии и операционная система (рис. 2.5).

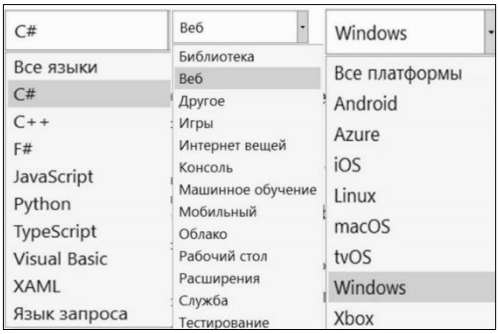

Рис. 2.5

Выбираем: язык программирования — C#; тип — Веб; операционная система — Windows и Веб-приложение ASP.NET (.NET Framework). Завершаем выбор кнопкой «Далее». Появляется новая форма (рис. 2.6).

Выбираем тип MVC, убираем флажок «Настроить для HTPPS» и завершаем кнопкой «Создать». Будет создан проект типа сайт.

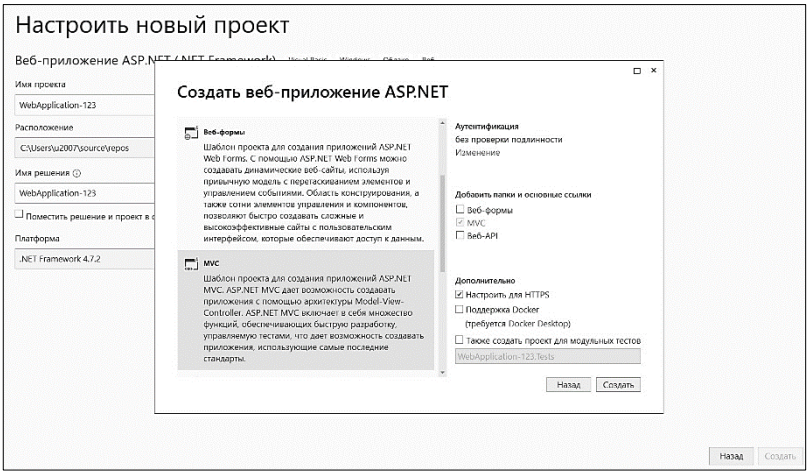

Рис. 2.6

2.4. Проект MVC

Результатом работы Visual Studio 2019 будет веб-проект (рис. 2.7).

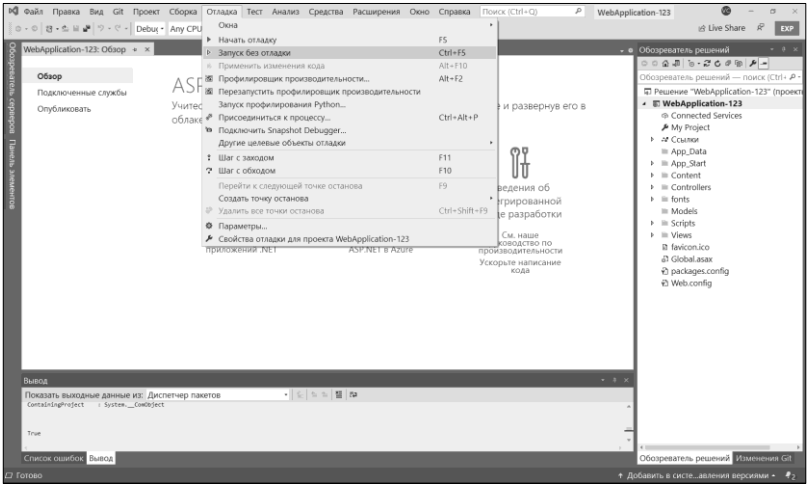

Рис. 2.7

В правой части рис. 2.7 в обозревателе решений показаны папки и файлы с программным кодом сайта как веб-технологии. Перечень папок с программным кодом приведен на рис. 2.8.

В верхней части рис. 2.7 приведен перечень функций и возможностей обработки проекта. Кнопка «Отладка» позволяет выполнить проект без отладки.

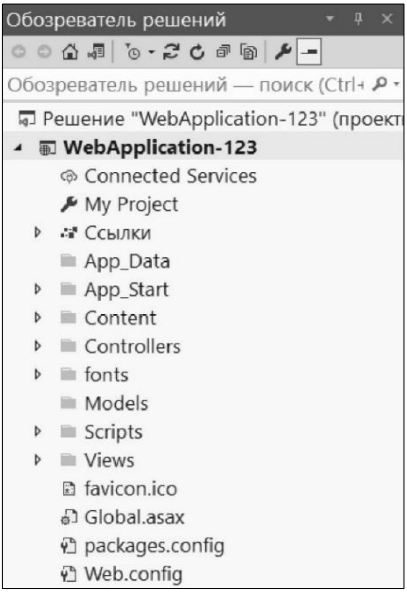

Рис. 2.8

Назначение папок и файлов проекта ASP.NET MVC следующее:

/App_Data — в эту папку помещаются закрытые данные, такие как XML-файлы или базы данных. Если используется SQL Server Express, SQLite или другие хранилища на основе файлов, веб-сервер IIS не будет обслуживать содержимое этой папки;

/App_Start — эта папка содержит ряд основных настроек конфигурации для проекта, в том числе определение маршрутов и фильтров, а также пакетов содержимого;

/Areas — области — это способ разделения крупного приложения на меньшие части;

/bin — сюда помещается скомпилированная сборка приложения MVC вместе со всеми ссылаемыми сборками, находящимися не в GAC. Чтобы увидеть папку bin в окне Solution Explorer (Проводник решения), понадобится щелкнуть на кнопке Show All Files (Показать все файлы). Поскольку все файлы в этой папке являются двоичными и генерируются в результате компиляции, обычно они не хранятся в системе управления исходным кодом;

/Content — сюда помещается статическое содержимое, такое как CSS-файлы и изображения. Это является необязательным соглашением. Статическое содержимое можно хранить в любом подходящем месте;

/Controllers — сюда помещаются классы контроллеров. Это является соглашением. Классы контроллеров могут размещаться где угодно, поскольку они компилируются в ту же самую сборку;

/Models — сюда помещаются классы моделей представлений и моделей предметной области, хотя все кроме простейших приложений выигрывают от определения модели предметной области в отдельном проекте. Это является соглашением. Классы моделей могут быть определены где угодно в текущем проекте или вообще вынесены в отдельный проект;

/Scripts — эта папка предназначена для хранения библиотек JavaScript, используемых в приложении. По умолчанию Visual Studio добавляет библиотеки jQuery и несколько других популярных JavaScript библиотек. Это является соглашением. Файлы сценариев могут находиться в любом месте, так как в действительности они представляют собой просто другой тип статического содержимого;

/Views — в этой папке хранятся представления и частичные представления, обычно сгруппированные вместе в папках с именами контроллеров, с которыми они связаны;

/Views/Shared — в этой папке хранятся компоновки и представления, не являющиеся специфичными для какого-либо контроллера;

/Views/Web.config — это конфигурационный файл. В нем содержится конфигурационная информация, которая обеспечивает обработку представлений с помощью ASP.NET и предотвра-

щает их обслуживание веб-сервером IIS, а также пространства имен, по умолчанию импортируемые в представления. Представления должны визуализироваться через методы действий;

/Global.asax — это глобальный класс приложения ASP.NET. В его файле отделенного кода (*Global.asax.cs*) регистрируется конфигурация маршрутов, а также предоставляется любой код, который должен выполняться при запуске или завершении приложения либо в случае возникновения необработанного исключения. В приложении MVC файл Global.asax играет ту же роль, что и в приложении Web Forms;

/Web.config — конфигурационный файл для приложения. В приложении MVC файл Web.config играет ту же роль, что и в приложении Web Forms.

Назначение папок и файлов проекта ASP.NET Core MVC следующее:

/Dependencies — все добавленные в проект пакеты и библиотеки;

/wwwroot — узел (на жестком диске ему соответствует одноименная папка), предназначенный для хранения статических файлов — изображений, скриптов javascript, файлов css и т. д., которые используются приложением. Цель добавления этой папки в проект по сравнению с другими версиями ASP.NET состоит в разграничении доступа к статическим файлам, к которым разрешен доступ со стороны клиента и к которым доступ запрещен;

/Controllers — папка для хранения контроллеров, используемых приложением;

/Models — каталог для хранения моделей;

/Views — каталог для хранения представлений;

/appsettings.json — хранит конфигурацию приложения;

/Program.cs — файл, определяющий класс Program, который инициализирует и запускает хост с приложением;

/Startup.cs — файл, определяющий класс Startup, с которого начинается работа приложения, т. е. это входная точка в приложение.

2.5. Выполнение проекта MVC

Для выполнения проекта MVC необходимо в средстве *Отладка* выполнить *Запуск без отладки*, как показано на рис. 2.9.

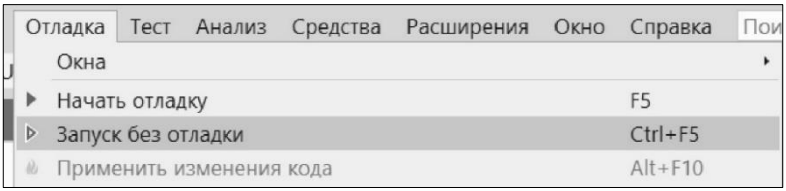

Рис. 2.9

Результаты выполнения показаны на рис. 2.10.

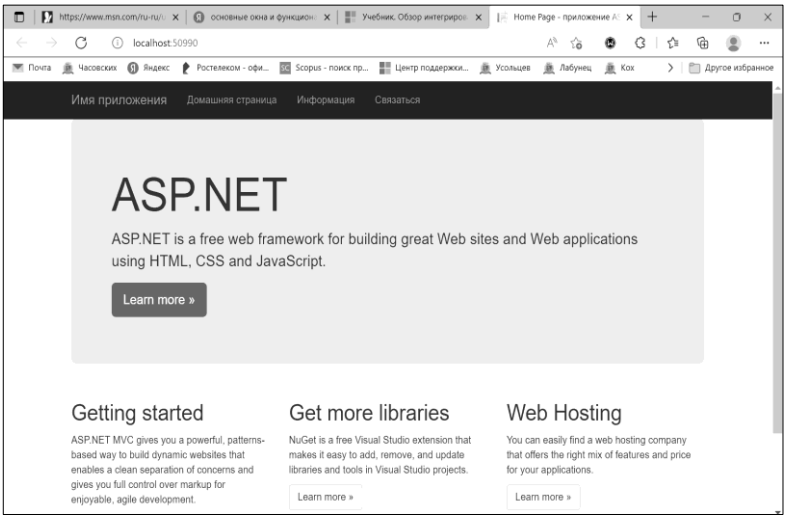

Рис. 2.10

Средствами Visual Studio 2019 был подготовлен функционально полный проект сайта как веб-технология. Программный код является синтаксически правильным в языке C#, а как следствие, проект может подготовить специалист без знания языка программирования C#.

Выводы по главе 2

Visual Studio — это линейка продуктов фирмы Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментов.

Visual Studio позволяет разрабатывать как консольные приложения, так и игры и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, UWP а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, .NET Core, .NET, MAUI, Xbox, Windows Phone, .NET Compact Framework и Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода.

Определены технологии создания проектов и веб-сайтов. Один проект может содержать несколько тысяч сайтов. Рассмотрена основная веб-технология — модель, контроллер, представление.

Вопросы для самоконтроля

1. Возникновение и история развития Visual Studio. Версии и возможности.
2. Возможности современной версии Visual Studio 2019. Можно ли в ней разрабатывать проекты искусственного интеллекта?
3. Visual Studio 2019, среда Python и машинное обучение.
4. Создание проекта, сохранность на внешних носителях, коррекция модулей и их отладка.
5. Запуск проекта. Окна, характеризующие процесс выполнения проекта.
6. Структуры проектов MVC. Создание и выполнение.

Глава 3

Веб-приложение MVC ASP.NET Core «Кафедра»

Цель настоящей главы — показать работу веб-технологии MVC на примере создания приложения, который максимально приближен к реальности. В рассматриваемом примере будем ориентироваться на ASP.NET Core MVC в среде Visual Studio 2019 с минимальным программированием, с концептуальным взаимодействием в реляционной базе данных SQL.

Мы будем рассматривать MVC-модель ASP.NET Core и Entity Framework Core с контроллерами и представлениями.

Будет создана административная область, которая включает в себя средства создания, просмотра, обновления и удаления (CRUD — create, read, update, delete) информационных моделей кафедры, и решены вопросы информационной безопасности информационной системы так, чтобы изменения могли вносить только зарегистрированные администраторы.

Мы будем использовать формы построения, генерируемые Visual Studio 2019 для ASP.NET Core MVC в расчете на удобный в сопровождении, расширяемый и хорошо структурированный код с поддержкой информационной безопасности.

Решение Visual Studio — это контейнер для нашего проекта.

Условное название проекта «Кафедра» с одной составляющей — дисциплина. Все остальное определяется по аналогии.

3.1. Создание проекта веб-приложения «Кафедра»

Запускаем Visual Studio 2019 (см. рис. 2.1, 2.2).

В начальном окне выбираем *Создание проекта* и дожидаемся формы выбора типа проекта (рис. 3.1, 3.2).

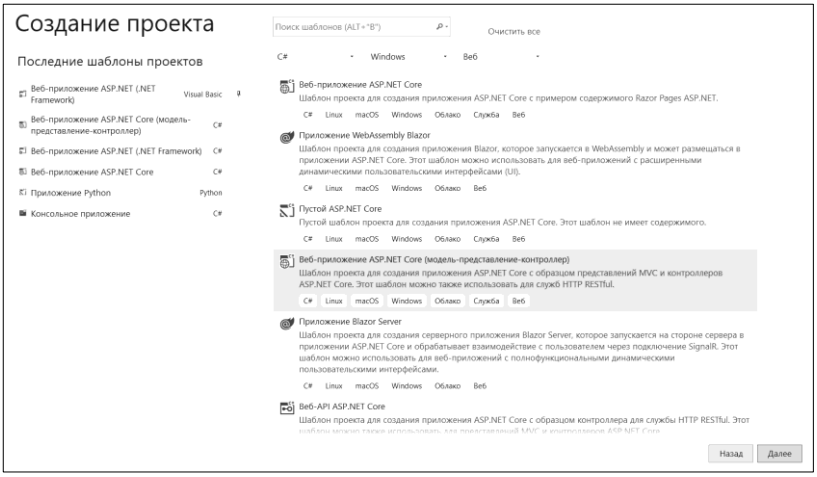

Рис. 3.1

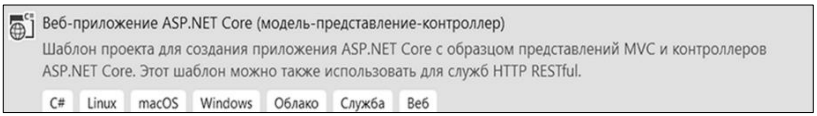

Рис. 3.2

После выбора типа проекта переходим «Далее». Дожидаемся формы *Настроить новый проект* (рис. 3.3).

Указываем Имя проекта — Kafedra. Определяем расположение папки для хранения проекта, в нашем случае папка «Кафедра» в папке Core3 рабочего стола. Имя решения Kafedra. Переходим «Далее». Дожидаемся формы *Дополнительные сведения* (рис. 3.4).

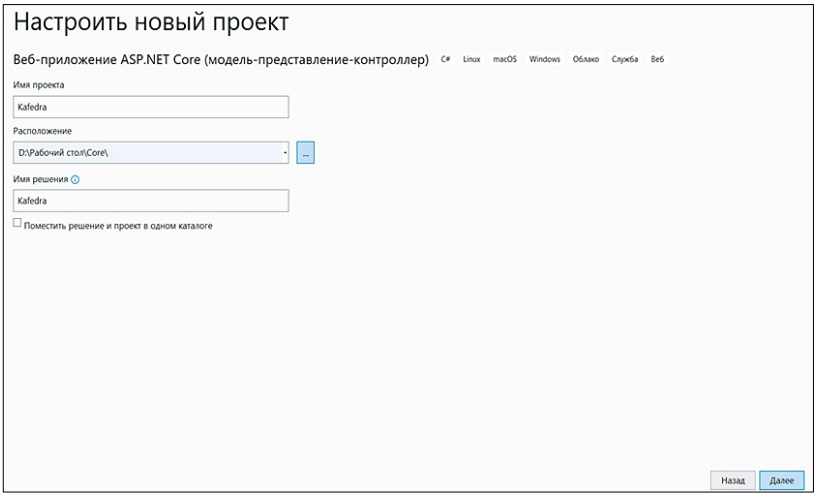

Рис. 3.3

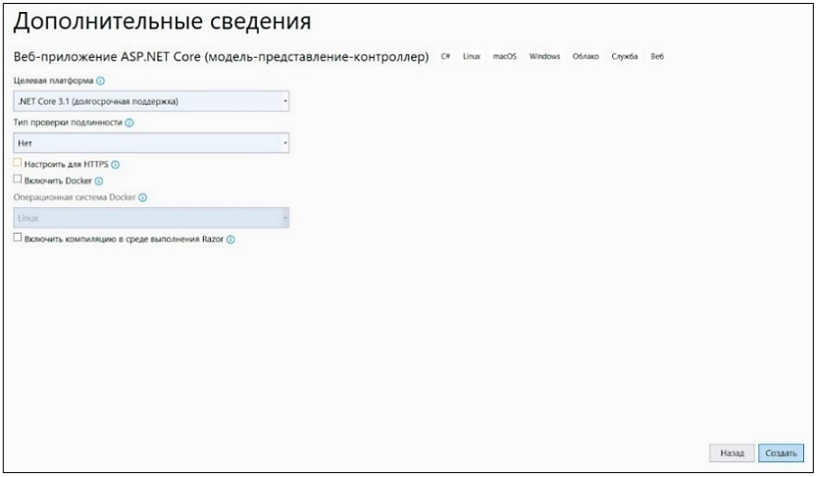

Рис. 3.4

В дополнительных сведениях квадратик «Настроить для HTTPS» должен быть **пуст**. Переходим «Создать». Дожидаемся созданного проекта Kafedra (рис. 3.5).

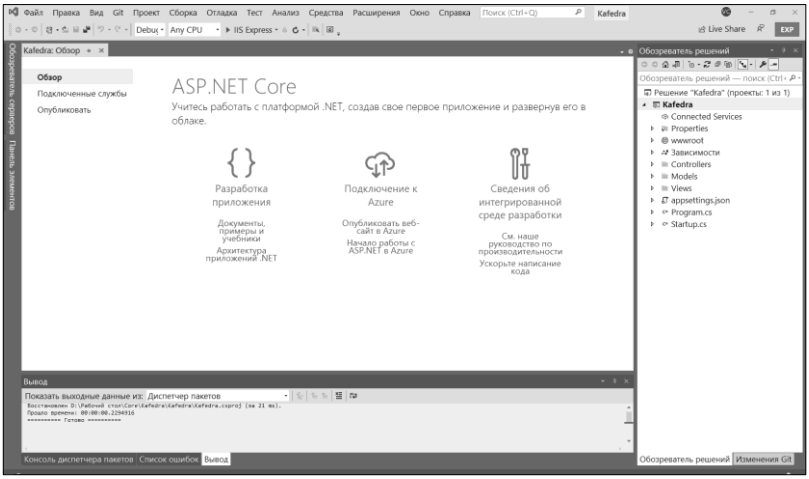

Рис. 3.5

Проект создан. Для проверки следует запустить проект. В верхней строчке выберите «Отладка» и «Запуск без отладки» (рис. 3.6).

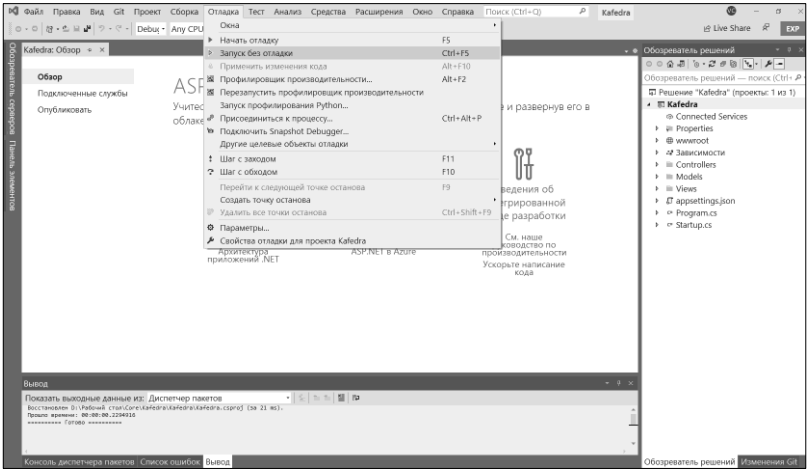

Рис. 3.6

Если нет ошибок, получим форму, показанную на рис. 3.7.

Рис. 3.7

3.2. Изменения в проекте

Сгенерированный Visual Studio 2019 проект Kafedra имеет функционально полную структуру сайта (рис. 3.8).

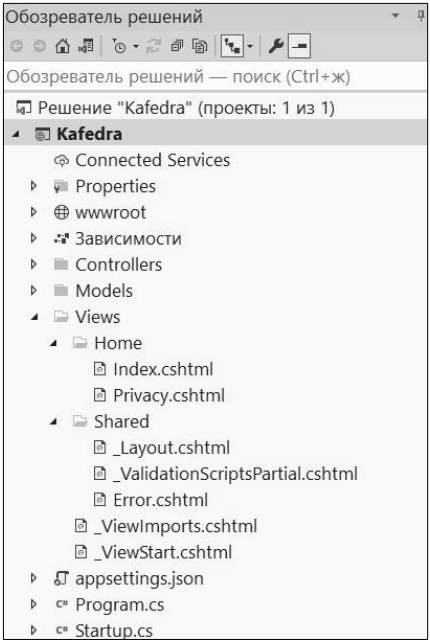

Рис. 3.8

Произведем изменения в файле _Layout.cshtml (файл формирования страниц компоновки представлений MVC). Открываем в редакторе файл _Layout.cshtml, щелкнув по названию (рис. 3.9).

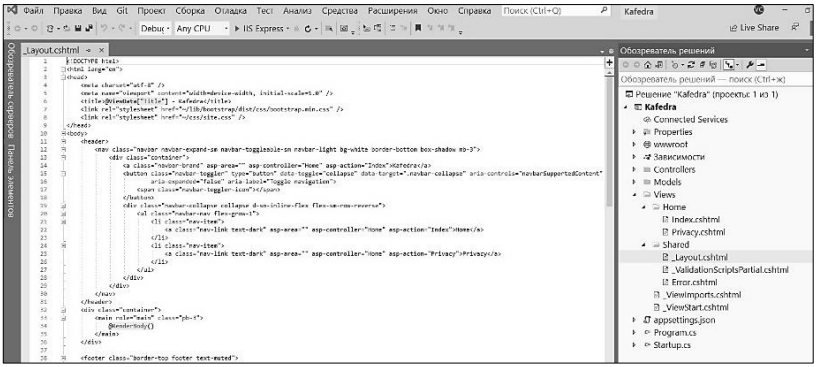

Рис. 3.9

И заменяем на следующее:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
  <title>@ViewData["Title"] - Kafedra03</title>
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/
bootstrap.min.css" />
  <link rel="stylesheet" href="~/css/site.css" />
</head>
<body>
  <header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm
navbar-light bg-white border-bottom box-shadow mb-3">
      <div class="container">
        <a class="navbar-brand" asp-area="" asp-controller=
"Home" asp-action="Index">Kafedra</a>
        <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target=".navbar-collapse" aria-con-
trols="navbarSupportedContent"
            aria-expanded="false" aria-label="Toggle navi-
gation">
          <span class="navbar-toggler-icon"></span>
        </button>

<div class="navbar-collapse collapse d-sm-inline-flex flex-
sm-row-reverse">

          <ul class="navbar-nav flex-grow-1">
```

```
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area=""
asp-controller="Home" asp-action="Index">Home</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area=""
asp-controller="Disziplinas" asp-action="Index">Дисциплина</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>
  </header>
  <div class="container">
    <main role="main" class="pb-3">
      @RenderBody()
    </main>
  </div>

  <footer class="border-top footer text-muted">
    <div class="container">
      &copy; 2022 - Kafedra - <a asp-area="" asp-controller=
"Home" asp-action="Privacy">Privacy</a>
    </div>
  </footer>
  <script src="~/lib/jquery/dist/jquery.min.js"></script>
  <script src="~/lib/bootstrap/dist/js/bootstrap.bun-
dle.min.js"></
script>
  <script src="~/js/site.js" asp-append-version="true">
</script>
  @RenderSection("Scripts", required: false)
</body>
</html>
```

В папке Home заменяем файл Index.cshtml на следующее:

```
@{
  ViewBag.Title = "Home Page";
}
<h1 style="text-align:center; color:#796310">
Учебное пособие «Повышение эффективности веб-технологий
в системах принятия решений» — создание сайта Kafedra
</h1>
```

Файл Privacy.cshtml заменяем на следующее:

```
@{
  ViewData["Title"] = "Политика конфиденциальности";
}
```

```
<h1>@ViewData["Title"]</h1>

<p>Обращайтесь к администратору ФИО.</p>
```

Для проверки следует запустить проект. В верхней строчке выберите «Отладка» и «Запуск без отладки». Если нет ошибок, получим форму, показанную на рис. 3.10.

Рис. 3.10

3.3. Создание модели проекта Kafedra

Включим в проект Kafedra предметную область, содержащую сущность Disziplina.

В папке Models проекта Kafedra создадим класс Disziplina, для этого правой кнопкой мыши щелкнем по папке Models, выберем «Добавить», «Класс» (рис. 3.11), укажем Disziplina.

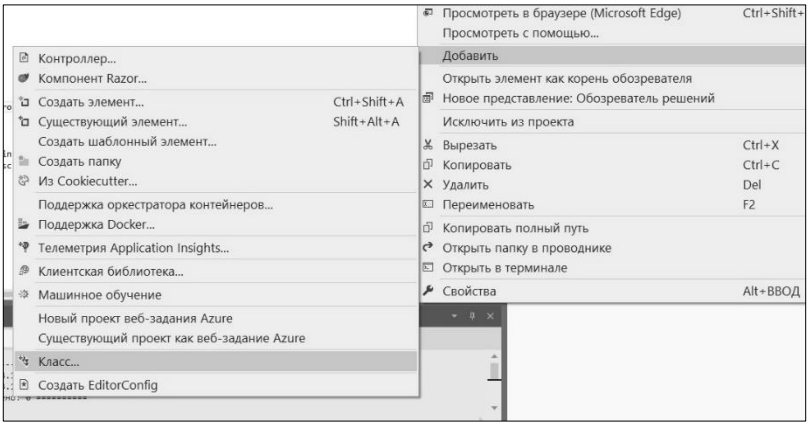

Рис. 3.11

Заменяем созданный сгенерированный код класса Disziplina на следующий текст:

```
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
namespace Kafedra
{
  public class Disziplina
  {
    [HiddenInput(DisplayValue = false)]
    [Display(Name = "Ключ записи")]
    public int DisziplinaID { get; set; }
    [Display(Name = "Название дисциплины")]
    public string Nazvanie { get; set; }
    [Display(Name = "Зачетные единицы")]
    public string ZachetEdin { get; set; }
    [Display(Name = "Курсовая работа")]
    public string KursRabota { get; set; }
    [Display(Name = "Контрольная работа")]
    public string KontRabota { get; set; }
    [Display(Name = "Форма получения оценки")]
    public string FormaOzenki { get; set; }
    [Display(Name = "Ссылка на рабочую  программу в папке
Uploads/RabProg")]
    public string ObrzavProgramma { get; set; }
    [Display(Name = "Имя файла рабочей программы")]
    public string DDopPole01 { get; set; }
    [Display(Name = "Читает кафедра")]
    public string DDopPole02 { get; set; }
    [Display(Name = "Направление - Бакалавр или Магистр или
Аспирант")]
    public string DDopPole03 { get; set; }
    [Display(Name = "Приобретаемые компетенции, записываются
через точку с запятой без пробелов")]
    public string DDopPole04 { get; set; }
    [Display(Name = "Доп. поле 05")]
    public string DDopPole05 { get; set; }
    public virtual ICollection<Disziplina> Disziplinas {
get; set; }
  }
}
```

Получилось описание дисциплины как таблицы базы данных SQL.

3.4. Создание контекста базы данных

Контекст базы данных обозначается **DbContext** — это основной класс, который координирует функциональные возможности Entity Framework (EF) для определенной модели данных. Этот класс является производным от класса Microsoft.EntityFrameworkCore.DbContext.

Производный класс DbContext указывает сущности, которые включаются в модель данных. Некоторые расширения функциональности EF можно настроить. В этом проекте соответствующий класс назовем KafedraContext.

В папке проекта создаем папку Data. В папке Data создаем класс KafedraContext для связи с базой данных SQL, содержащий следующий код:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using Kafedra;

namespace Kafedra.Data
{
  public class KafedraContext : DbContext
  {
    public KafedraContext (DbContextOptions<KafedraContext>
options)
      : base(options)
    {
    }
    public DbSet<Kafedra.Disziplina> Disziplina { get; set; }
  }
}
```

В тексте созданного класса KafedraContext появятся подчеркивания красным отсутствующих библиотек. Для их добавления выполним следующее.

Щелкнем мышкой по названию проекта Kafedra, получим форму, показанную на рис. 3.12.

Выбрав «Управление пакетами NuGet», получаем форму, показанную на рис. 3.13.

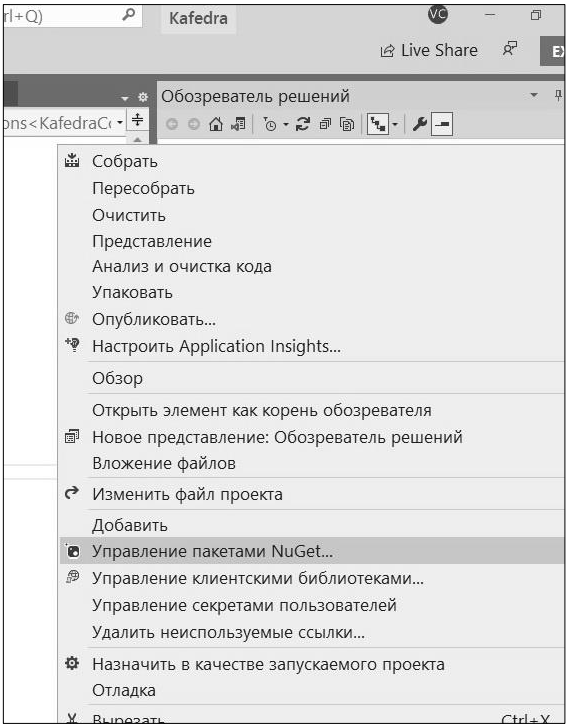

Рис. 3.12

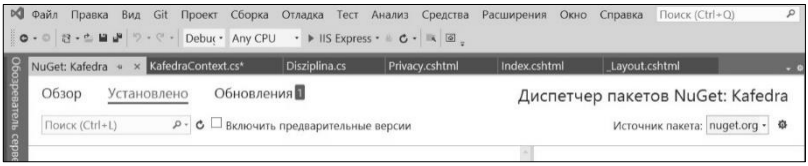

Рис. 3.13

Активируем «Установлено» и указываем «Источник пакета: Все». В поле «Поиск» указываем библиотеку Microsoft.EntityFrameworkCore.SqlServer.

По найденной библиотеке Microsoft.EntityFrameworkCore.SqlServer щелкаем мышкой, указываем версию 3.1.0 и переходим кнопкой «Установить». После установки все красные подчеркивания исчезают.

3.5. Стандартные настройки проекта

Далее заменяем следующие файлы в папке Views _ViewImports.cshtml и проекте appsettings.json, Program.cs, Startup.cs

_ViewImports.cshtml

```
@using Kafedra
@using Kafedra.Models
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "KafedraContext": "Server=(localdb)\\mssqllocaldb;Data-
base=KafedraContext-1;Trusted_Connection=True;MultipleActive
ResultSets=true",

  }
}
```

Program.cs

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Kafedra
{
  public class Program
  {
    public static void Main(string[] args)
```

```
    {
      CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[]
args) =>
      Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
        {
          webBuilder.UseStartup<Startup>();
        });
  }
}
```

Startup.cs

```
using Kafedra.Data;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Kafedra
{
  public class Startup
  {
    public Startup(IConfiguration configuration)
    {
      Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this
method to add services to the container.

    public void ConfigureServices(IServiceCollection ser-
vices)
    {
      services.AddControllersWithViews();

      services.AddDbContext<KafedraContext>(options =>
```

```
            options.UseSqlServer(Configuration.GetConnection-
String("KafedraContext")));
    }

    // This method gets called by the runtime. Use this
method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app,
IWebHostEnvironment env)
    {
      if (env.IsDevelopment())
      {
        app.UseDeveloperExceptionPage();
      }
      else
      {
        app.UseExceptionHandler("/Home/Error");
      }
      app.UseStaticFiles();

      app.UseRouting();

      app.UseAuthorization();

      app.UseEndpoints(endpoints =>
      {
        endpoints.MapControllerRoute(
          name: "default",
          pattern: "{controller=Home}/{action=Index}/{id?}");
      });
    }
  }
}
```

3.6. Создание контроллера модели Disziplina

Устанавливаем библиотеки для генерации контроллеров и программ обработки. Выбираем «Управление пакетами NuGet» и устанавливаем Microsoft.VisualStudio.Web.CodeGeneration.Design версия 3.1.0.

Для создания контроллера модели Disziplina необходимо в проекте Kafedra щелкнуть по папке Controllers. Откроется форма, показанная на рис. 3.14.

Выбираем действия «Добавить» и «Контроллер». Будет сгенерирована форма, показанная на рис. 3.15.

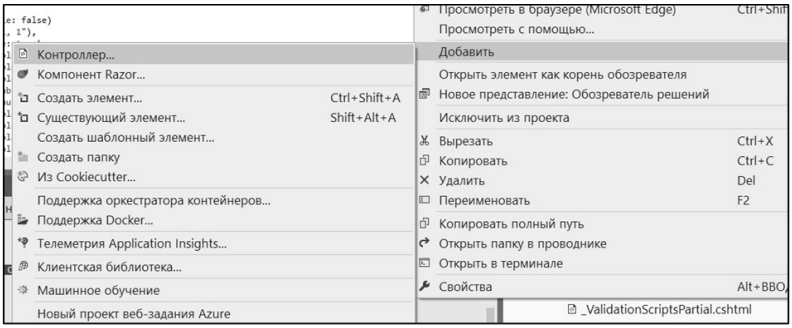

Рис. 3.14

Рис. 3.15

Выбираем шаблон контроллера для модели Disziplina (рис. 3.16).

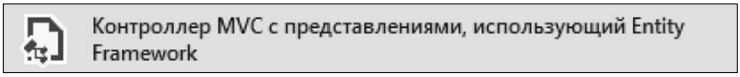

Рис. 3.16

В течение 5−8 секунд Visual Studio 2019 генерирует код контроллера и 5 программ для управление таблицей — моделью Disziplina в базе данных SQL.

В проект Kafedra, в папку Controllers добавлен код DisziplinasController.cs. Создана папка Disziplinas и в ней 5 программ для управление таблицей — моделью Disziplina в базе данных SQL.

Все изменения в проекте показаны на рис. 3.17.

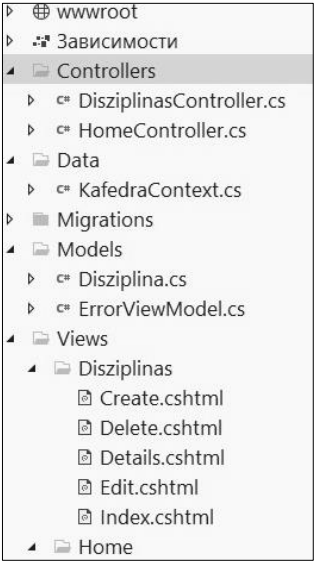

Рис. 3.17

3.7. Создание базы данных SQL для данных проекта Kafedra

Для всех моделей (их может быть любое количество) любого проекта создается их обобщенное описание, называемое «Миграция». В проекте создается папка Migrations для обобщенного описания всех моделей проекта.

Для создания «Миграции» и базы данных SQL выбираем в верхней строчке «Средства», далее «Диспетчер пакетов NuGet» и «Консоль диспетчера пакетов» (рис. 3.18).

Консоль диспетчера пакетов обозначается **PM>**. Вводим команду **Add-Migration InitialCreate** и далее **Update-Database**. На рис. 3.19 показано успешное создание папки «Миграция» и базы данных SQL.

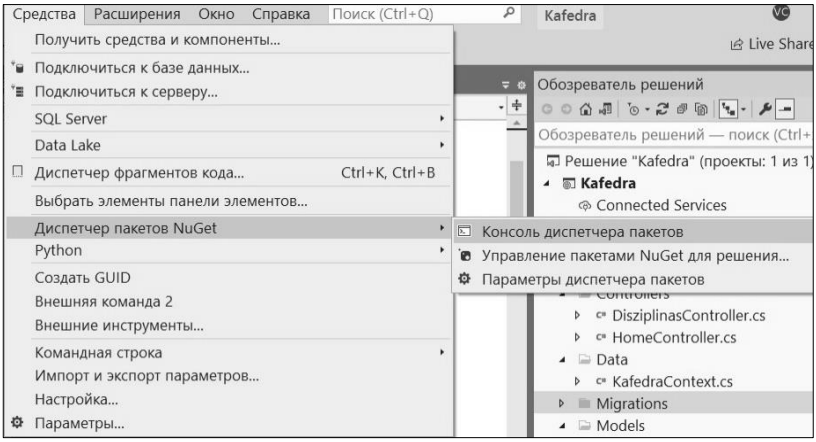

Рис. 3.18

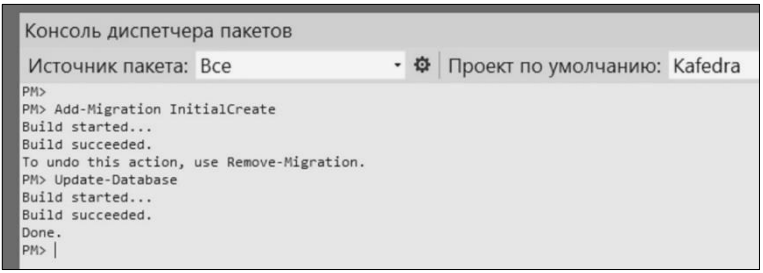

Рис. 3.19

Visual Studio 2019 в каждом проекте предоставляет эффективное средство для работы с базой данных. Доступ к этому средству показан на рис. 3.20 (Обозреватель серверов) и рис. 3.21 (Обработка баз данных SQL).

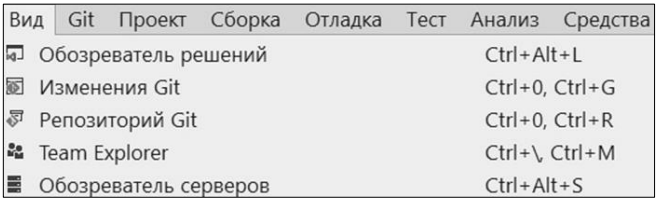

Рис. 3.20

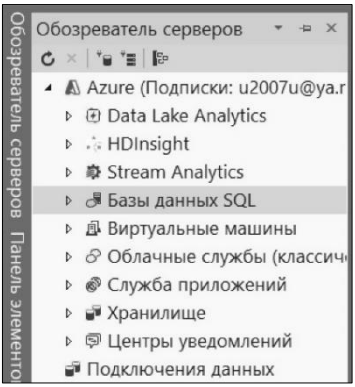

Рис. 3.21

3.8. Создание средств для пользователя и администратора сайта

При создании любого контроллера Visual Studio 2019 создает папку, называемую как контроллер. Эта папка создается в **Views** проекта и включает пять стандартных программ обработки таблицы SQL (соответствующая модель предметной области). Для контроллера Disziplina созданные программы показаны на рис. 3.22:

– **Create** — для ввода новых записей таблицы SQL, соответствующей модели предметной области;

– **Delete** — для удаления записей таблицы SQL, соответствующей модели предметной области;

– **Details** — для детального просмотра записей таблицы SQL, соответствующей модели предметной области;

– **Edit** — для редактирования записей таблицы SQL, соответствующей модели предметной области;

– **Index** — для просмотра записей таблицы SQL, соответствующей модели предметной области, ввода новых записей и выбора действий над записями — коррекции, удаления.

Все перечисленные программы требуют изменения дизайна визуализации результатов обработки.

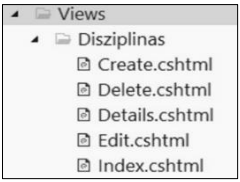

Рис. 3.22

3.8.1. Скрипт для табличного представления данных

Расширим функции библиотеки проекта Kafedra скриптом формирования таблиц и их обработки. Идентификатор скрипта таблиц FooTable-master. Расположение показано на рис. 3.23.

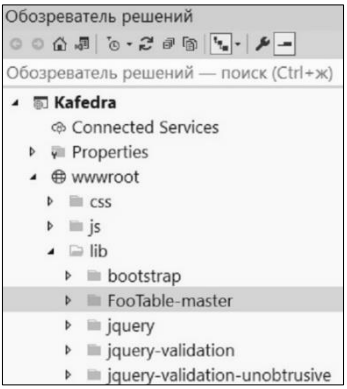

Рис. 3.23

3.8.2. Изменения в программе Index

Необходимо заменить текст сгенерированной Visual Studio 2019 программы Index в библиотеке Disziplinas на следующий, новый код.

Новый код Index

```
@model IEnumerable<Kafedra.Disziplina>

@{
  ViewData["Title"] = "Дисциплины и их характеристики";
  //Layout = "~/Views/Shared/_Layout.cshtml";
}
```

```
<h1>Index</h1>

<link href="~/lib/FooTable-master/css/footable-0.1.css"
rel="stylesheet" />
<script src="http://ajax.googleapis.com/ajax/libs/jquery/
1.8.2/jquery.min.js" type="text/javascript"></script>
<script src="js/footable.js" type="text/javascript">
</script>
<script src="~/lib/FooTable-master/js/footable.js"></script>
<script type="text/javascript">
  $(function () {
    $('table').footable();
  });
</script>
<p>
  <a asp-action="Create">Ввод новой дисциплины</a>
</p>

<form asp-action="Index" method="get">
  <div class="form-actions no-color">
    <p>
      Дисциплина или контрольная работа: <input type="text"
name="SearchString" value="@ViewData["CurrentFilter"]" />
      <input type="submit" value="Поиск" class="btn btn-de-
fault" /> |
      <a asp-action="Index"> Вернуться к исходной таблицы
t</a>
    </p>
  </div>
</form>
<table class="footable" style="width:100%; margin: 0 auto;">
  <thead>
    <tr>
      <th data-class="expand">
        @Html.DisplayNameFor(model => model.Nazvanie)
        <a asp-action="Index" asp-route-sortOrder="@View-
Data["NameSortParm"]">@Html.DisplayNameFor(model =>
model.Nazvanie)</a>
      </th>
      <th data-hide="phone,tablet">
        @Html.DisplayNameFor(model => model.ZachetEdin)
      </th>
      <th data-hide="phone,tablet">
        @Html.DisplayNameFor(model => model.KursRabota)
      </th>
      <th data-hide="phone,tablet">
        <a asp-action="Index" asp-route-sortOrder="@View-
Data ["DateSortParm"]">@Html.DisplayNameFor(model =>
model.KontRabota)</a>
```

```
        </th>
        <th>
          @Html.DisplayNameFor(model => model.FormaOzenki)
        </th>
        <th>
          @Html.DisplayNameFor(model => model.ObrzavProgramma)
        </th>
        <th>Действие</th>
      </tr>
    </thead>
    <tbody>
      @foreach (var item in Model)
      {
        <tr>
          <td>
            @Html.DisplayFor(modelItem => item.Nazvanie)
          </td>
          <td>
            @Html.DisplayFor(modelItem => item.ZachetEdin)
          </td>
          <td>
            @Html.DisplayFor(modelItem => item.KursRabota)
          </td>
          <td>
            @Html.DisplayFor(modelItem => item.KontRabota)
          </td>
          <td>
            @Html.DisplayFor(modelItem => item.FormaOzenki)
          </td>
          <td>
            @Html.DisplayFor(modelItem => item.ObrzavProgramma)
          </td>
          <td>
            <a asp-action="Edit" asp-route-id="@item.Diszip-
linaID">Изменить</a> |
            <a asp-action="Details" asp-route-id="@item.Dis-
ziplinaID">Подробно</a> |
            <a asp-action="Delete" asp-route-id="@item.Dis-
ziplinaID">Удалить</a>
          </td>
        </tr>
      }
    </tbody>
</table>
```

Результаты выполнения показаны на рис. 3.24.

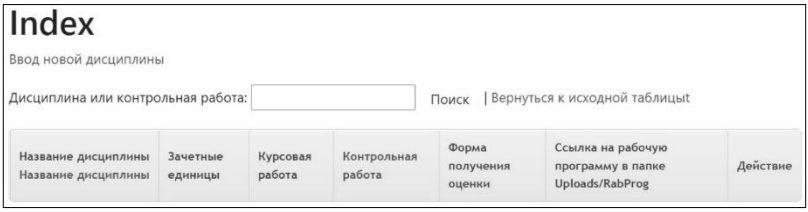

Рис. 3.24

3.8.3. Изменения в программе Create

Необходимо заменить текст сгенерированной Visual Studio 2019 программы Create в библиотеке Disziplinas на следующий, новый код.

Новый код Create

```
@model Kafedra.Disziplina
@{
  ViewData["Title"] = "Ввод дисциплины";
}
<h1>Ввести дисциплину и её характеристики</h1>
<h4>Программа Disziplina</h4>
<hr />
<div class="row">
  <div class="col-md-4">
    <form asp-action="Create">
      <div asp-validation-summary="ModelOnly" class="text-
danger"></div>
      <div class="form-group">
        <label asp-for="Nazvanie" class="control-label">
</label>
        <input asp-for="Nazvanie" class="form-control" />
        <span asp-validation-for="Nazvanie" class="text-
danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="ZachetEdin" class="control-label">
</label>
        <input asp-for="ZachetEdin" class="form-control" />
        <span asp-validation-for="ZachetEdin" class="text-
danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="KursRabota" class="control-label">
</label>
        <input asp-for="KursRabota" class="form-control" />
```

```
          <span asp-validation-for="KursRabota" class="text-
danger"></span>
        </div>
        <div class="form-group">
          <label asp-for="KontRabota" class="control-label">
</label>
          <input asp-for="KontRabota" class="form-control" />
          <span asp-validation-for="KontRabota" class="text-
danger"></span>
        </div>
        <div class="form-group">
          <label asp-for="FormaOzenki" class="control-label">
</label>
          <input asp-for="FormaOzenki" class="form-control" />
          <span asp-validation-for="FormaOzenki" class="text-
danger"></span>
        </div>
        <div class="form-group">
          <label asp-for="ObrzavProgramma" class="control-la-
bel"></label>
          <input asp-for="ObrzavProgramma" class="form-con-
trol" />
          <span asp-validation-for="ObrzavProgramma"
class="text-danger"></span>
        </div>
        <div class="form-group">
          <label asp-for="DDopPole01" class="control-label">
</label>
          <input asp-for="DDopPole01" class="form-control" />
          <span asp-validation-for="DDopPole01" class="text-
danger"></span>
        </div>
        <div class="form-group">
          <label asp-for="DDopPole02" class="control-label">
</label>
          <input asp-for="DDopPole02" class="form-control" />
          <span asp-validation-for="DDopPole02" class="text-
danger"></span>
        </div>
        <div class="form-group">
          <label asp-for="DDopPole03" class="control-la-
bel"></label>
          <input asp-for="DDopPole03" class="form-control" />
          <span asp-validation-for="DDopPole03" class="text-
danger"></span>
        </div>
        <div class="form-group">
          <label asp-for="DDopPole04" class="control-la-
bel"></label>
```

```
        <input asp-for="DDopPole04" class="form-control" />
        <span asp-validation-for="DDopPole04" class="text-
danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="DDopPole05" class="control-label">
</label>
        <input asp-for="DDopPole05" class="form-control" />
        <span asp-validation-for="DDopPole05" class="text-
danger"></span>
      </div>
      <div class="form-group">
        <input type="submit" value="Загрузить в БД"
class="btn btn-primary" />
      </div>
    </form>
  </div>
</div>
<div>
  <a asp-action="Index">Отказаться от ввода</a>
</div>
@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPar-
tial");}
}
```

Результаты выполнения показаны на рис. 3.25.

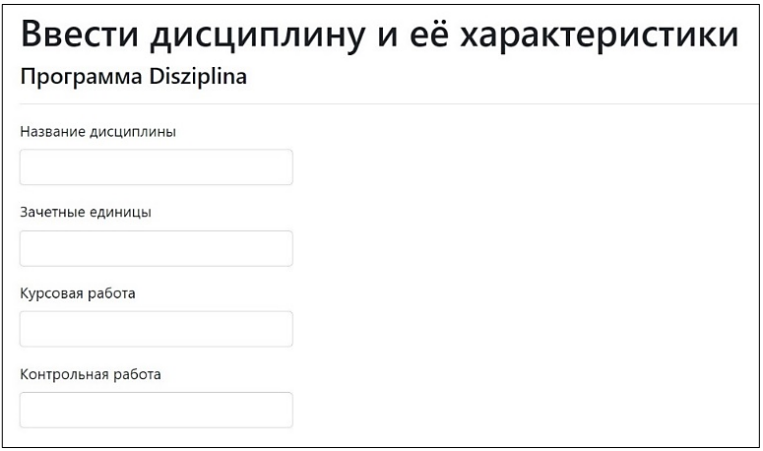

Рис. 3.25

3.8.4. Изменения в программе Edit

Необходимо заменить текст сгенерированной Visual Studio 2019 программы Edit в библиотеке Disziplinas на следующий, новый код.

Новый код Edit

```
@model Kafedra.Disziplina
@{
  ViewData["Title"] = "Изменить";
}
<h1>Редактировать характеристики дисциплины</h1>
<h4>Программа Disziplina</h4>
<hr />
<div class="row">
  <div class="col-md-4">
    <form asp-action="Edit">
      <div asp-validation-summary="ModelOnly" class="text-
danger"></div>
      <input type="hidden" asp-for="DisziplinaID" />
      <div class="form-group">
        <label asp-for="Nazvanie" class="control-label">
</label>
        <input asp-for="Nazvanie" class="form-control" />
        <span asp-validation-for="Nazvanie" class="text-
danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="ZachetEdin" class="control-label">
</label>
        <input asp-for="ZachetEdin" class="form-control" />
        <span asp-validation-for="ZachetEdin" class="text-
danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="KursRabota" class="control-label">
</label>
        <input asp-for="KursRabota" class="form-control" />
        <span asp-validation-for="KursRabota" class="text-
danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="KontRabota" class="control-label">
</label>
        <input asp-for="KontRabota" class="form-control" />
        <span asp-validation-for="KontRabota" class="text-
danger"></span>
      </div>
```

```
      <div class="form-group">
        <label asp-for="FormaOzenki" class="control-label">
</label>
        <input asp-for="FormaOzenki" class="form-control" />
        <span asp-validation-for="FormaOzenki" class="text-
danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="ObrzavProgramma" class="control-la-
bel"></label>
        <input asp-for="ObrzavProgramma" class="form-con-
trol" />
        <span asp-validation-for="ObrzavProgramma"
class="text-danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="DDopPole01" class="control-label">
</label>
        <input asp-for="DDopPole01" class="form-control" />
        <span asp-validation-for="DDopPole01" class="text-
danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="DDopPole02" class="control-label">
</label>
        <input asp-for="DDopPole02" class="form-control" />
        <span asp-validation-for="DDopPole02" class="text-
danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="DDopPole03" class="control-label">
</label>
        <input asp-for="DDopPole03" class="form-control" />
        <span asp-validation-for="DDopPole03" class="text-
danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="DDopPole04" class="control-label">
</label>
        <input asp-for="DDopPole04" class="form-control" />
        <span asp-validation-for="DDopPole04" class="text-
danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="DDopPole05" class="control-label">
</label>
        <input asp-for="DDopPole05" class="form-control" />
        <span asp-validation-for="DDopPole05" class="text-
danger"></span>
```

```
      </div>
      <div class="form-group">
        <input type="submit" value="Сохранить в БД"
class="btn btn-primary" />
      </div>
    </form>
  </div>
</div>

<div>
  <a asp-action="Index"> Отказаться от коррекции </a>
</div>
@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPar-
tial");}
```

3.8.5. Режим администрирования и обычного пользователя

Выше был приведен вариант проекта для администратора, поскольку рассматривались функции создания базы данных, а также ввод, коррекция и удаление записей из базы данных. Для разграничения доступа к режиму администрирования необходимо на этапе создания проекта указать защиту записей логином и паролями (рис. 3.26).

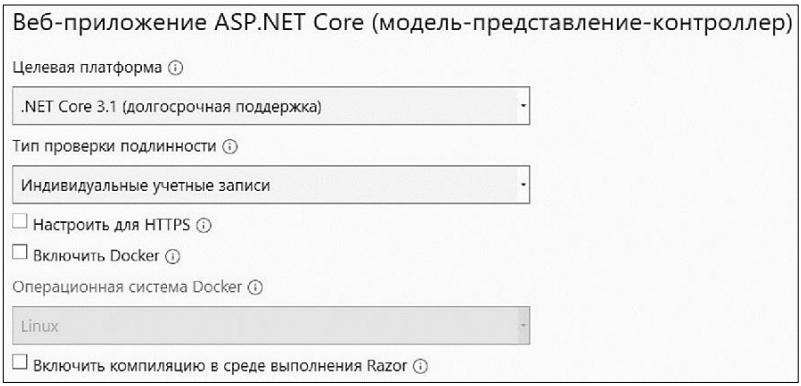

Рис. 3.26

В поле «Тип проверки подлинности» необходимо выбрать «Индивидуальные учетные записи». При создании проекта будет

сформирована система защиты записей базы данных и организован доступ с указанием логина и пароля.

Логин указывается для администратора в контроллере модели, таблица которой защищается.

```
[Authorize(Users = "admin@Chasovskikh.ru,
admin@Prepod01.ru")]

public class DisziplinasController : Controller
```

В параметре Authorize указываются конкретные логины, а при начале работы с проектом потребуется зарегистрировать пароль. Все допустимые варианты определяются системой защиты и не вызывают каких-либо затруднений.

Для пользователя создается отдельный контроллер, например UserController, и в библиотеке оставляется только одна программа с именем Index. И только эта программа будет доступна всем пользователям для просмотра.

Выводы по главе 3

В предшествующих главах мы рассмотрели концепцию веб-технологий и определили платформу для разработки веб-приложений ASP.NET. В качестве среды разработки выбрали Visual Studio 2019. Рассмотрели основные средства языка C#. Собрав все вместе, разработали несложное, но реалистичное приложение для кафедры вуза.

Для проектирования сайта в качестве модели выбрали одну составляющую «Дисциплина». Все остальные составляющие реальной кафедры в разработке полностью повторяют технологию для модели «Дисциплина».

Наше приложение под названием Kafedra следовало классическому подходу, который повсеместно используется в среде Visual Studio 2019, когда не требуется программирование всех действий и операций при взаимодействии веб-приложения с пользователем. Все необходимые программные модули генерировались средствами Visual Studio 2019. Разработка была ориентирована на ASP.NET Core MVC, поэтому интеграция с внешними системами, такими как база данных, была предельно упрощена и не требовала программирования в языке SQL.

Была создана административная область, которая включает в себя средства создания, чтения, обновления и удаления (create, read, update, delete — CRUD) для управления моделями кафедры. Классическая система защиты базы данных была сгенерирована средствами Visual Studio 2019 так, чтобы изменения могли вносить только зарегистрированные администраторы.

Были продемонстрированы возможности MVC на примере приложения, который максимально приближен к реальности.

Построение всех уровней необходимой инфраструктуры может показаться несколько медленным, но первоначальные трудозатраты при разработке приложения MVC окупаются, обеспечивая удобный в сопровождении, расширяемый и хорошо структурированный код.

Каждый шаг построения приложения был выделен, что позволило понять, каким образом средства MVC сочетаются друг с другом. Особое внимание было обращено на создание представлений.

Вопросы для самоконтроля

1. Типы веб-приложений в среде Visual Studio 2019.
2. Основные параметры веб-приложения. Возможности размещения приложения на внешних носителях.
3. Последовательность создания веб-приложения в среде Visual Studio 2019.
4. Создание контекста базы данных SQL.
5. Стандартные настройки веб-приложения ASP.NET Core MVC.
6. Создание программного кода контроллера веб-проекта.
7. Создание средств для администратора веб-приложения.
8. Создание технологии доступа пользователя к веб-приложению.
9. Создание логинов доступа к базе данных администратора веб-приложения.
10. Возможности представления табличных данных с помощью внешних скриптов.
11. Изменения в сгенерированных программах визуализации (представления) таблиц модели в базе данных SQL.
12. Контроллер для администратора веб-приложения и пользователя.

Заключение

В настоящее время веб-технологии стали эффективным средством решения сложных задач автоматизации, управления, проектирования и проведения экспериментов.

Научиться методам и подходам в данной области знаний как рабочему инструменту можно только при полном овладении приемами и технологией решения задач. Эту цель и преследует данное учебное пособие, предлагая необходимое количество теории для решения практических задач, при этом представленной таким образом, чтобы процесс реализации теоретических блоков в виде программ и алгоритмов на вычислительных машинах был наименее трудоемким.

Веб-технологии участвуют в глобальных преобразованиях технологической индустрии. Цифровая сфера развивается ускоренными темпами благодаря появлению искусственного интеллекта, компьютерному образованию, Интернету вещей и большим данным. Все эти составляющие представлены в платформе ASP.NET Core MVC и среде Visual Studio 2019, 2022.

Искусственный интеллект — главная тенденция развития веб-пространства и веб-технологий. Microsoft и другие ведущие компании делают общедоступными технологии, с помощью которых можно создавать собственные веб-проекты.

Среда Visual Studio 2019, 2022 позволяет просто и без особых навыков создавать достойные приложения в концепции Web 3.0. Это — концепция развития Интернета следующего поколения, которая строится вокруг идеи децентрализации.

Задание для самостоятельной работы

Создание проекта сайта «Кафедра»

Реальный проект сайта «Кафедра вуза» выполняется средствами и технологиями, рассмотренными в главах 1–3 учебного пособия.

Модель представлена сущностями: история кафедры, направления подготовки, преподаватель, студент, расписание, дисциплина, библиотека, наука, научные публикации, выпускники, достижения, сотрудничество (партнерство), персональная страница преподавателя, ЭИОС.

Платформа ASP.NET Core MVC в среде Visual Studio 2019.

Для запуска сайта «Кафедра вуза» необходимо его скопировать с http://vikchas.ru/Uploads/Biblioteka/Kafedra_1.rar и запустить.

Библиографический список

Использованная литература

Евдокимов А. П., Финков М. В. Создание сайтов своими руками на BOOTSTRAP. СПб.: Наука и Техника, 2017. 240 с.

Евдокимов П. В. C# на примерах. 2-е изд. СПб.: Наука и Техника, 2017. 320 с.

Кириченко А. В., Дубовик Е. В. Динамические сайты на HTML, CSS, Javascript и Bootstrap. Практика, практика и только практика. СПб.: Наука и Техника, 2018. 272 с.

Морето С. Bootstrap в примерах / пер. с англ. Р. Н. Рагимова; науч. ред. А. Н. Киселев. М.: ДМК Пресс, 2017. 314 с.

Прайс М. Дж. C# 7 и .NET Core. Кроссплатформенная разработка для профессионалов. 3-е изд. СПб.: Питер, 2018. 640 с.

Столбовский Д. Н. Основы разработки Web-приложений на ASP.NET. М.: Бином, 2014. 304 с.

Троелсен Э., Джепикс Ф. Язык программирования C# 7 и платформы .NET и .NET Core. 8-е изд.: пер. с англ. СПб.: Диалектика, 2018. 1328 с.

Фримен А. ASP.NET Core MVC 2 с примерами на C# для профессионалов: пер. с англ. 7-е изд. СПб.: Диалектика, 2019. 1008 с.

Фримен А. ASP.NET Core MVC с примерами на C# для профессионалов: пер. с англ. 6-е изд. СПб.: Альфа-книга, 2017. 992 с.

Фримен А. Entity Framework Core 2 для ASP.NET Core MVC для профессионалов: пер. с англ. СПб.: Диалектика, 2019. 624 с.

Фримен А. jQuery 2.0 для профессионалов: пер. с англ. М.: Вильямс, 2015. 1040 с.

Фримен А., Раттц-мл. Дж. С. LINQ: язык интегрированных запросов в C# 2010 для профессионалов: пер. с англ. М.: Вильямс, 2011. 656 с.

Рекомендуемая литература

Изучаем ASP.NET MVC 4. URL: https://metanit.com/sharp/mvc/.

Руководства по ASP.NET / Microsoft Learn. URL: https://docs.microsoft.com/ru-ru/aspnet/tutorials/.

Фримен А. ASP.NET MVC 5 с примерами на C# для профессионалов: пер. с англ. 5-е изд. М.: Вильямс, 2016. 736 с.

Хартл М. Ruby on Rails для начинающих / пер. с англ. А. Разуваева. М.: ДМК Пресс, 2017. 572 с.

Чедвик Дж., Снайдер Т., Панда Х. ASP.NET MVC 4: разработка реальных веб-приложений с помощью ASP.NET MVC: пер. с англ. М.: Вильямс, 2013. 432 с.

Эспозито Д. Разработка веб-приложений с использованием ASP.NET и AJAX. СПб.: Питер, 2012. 400 с.

Информация об авторах

Часовских Виктор Петрович — профессор кафедры шахматного искусства и компьютерной математики УрГЭУ, доктор технических наук, профессор

e-mail: u2007u@ya.ru

Лабунец Валерий Григорьевич — профессор кафедры шахматного искусства и компьютерной математики УрГЭУ, доктор технических наук, профессор

e-mail: vlabunets05@yahoo.com

Стариков Евгений Николаевич — заместитель заведующего кафедрой шахматного искусства и компьютерной математики УрГЭУ, кандидат экономических наук, доцент

e-mail: starikov_en@usue.ru

Кох Елена Викторовна — доцент департамента информационных технологий и автоматики УрФУ, кандидат сельскохозяйственных наук, доцент

e-mail: kohev@m.usfeu.ru

Воронов Михаил Петрович — доцент кафедры шахматного искусства и компьютерной математики УрГЭУ, кандидат технических наук, доцент

e-mail: mstrk@yandex.ru

Оглавление

Введение **3**

Глава 1. Веб-технологии и платформа ASP.NET **6**

1.1. Понятие веб-технологии 6
1.2. Платформа ASP.NET 7
1.2.1. Что такое ASP.NET 7
1.2.2. История ASP.NET 8
1.2.3. Фреймворки на базе ASP.NET 8
1.2.4. Комплектация ASP.NET 8
1.2.5. Возможности ASP.NET 9
1.2.6. Изучение ASP.NET 9
1.2.7. Инструменты программирования для ASP.NET 9
1.2.8. Веб-сервер IIS, запуск и настройки 10
1.2.9. Установка IIS 11
1.2.10. Дополнительные установки 11
1.2.11. Тестирование сайтов на локальном компьютере 11
1.3. Виды веб-проектов 11
1.3.1. Видимые отличия проектов в Visual Studio 12
1.3.2. Особенности кодирования веб-приложения и веб-сайта 12
1.3.3. Инструменты программирования веб-приложений 12
1.3.4. Инструменты программирования веб-сайтов 13
1.3.5. Рекомендация выбора веб-приложения 13
1.3.6. Рекомендации в пользу веб-сайта 13
1.4. Инструменты работы с Web Forms 14
1.4.1. Среда HTML 14
1.4.2. Структура HTML-документа 16
1.4.3. Razor — интеллектуальный и понятный 27
1.4.4. Веб-сайт на базе Web Forms 28
1.4.5. Инфраструктура ASP.NET MVC 29
1.4.6. Среда Visual Studio 33
1.4.7. Понятие модели 35
1.4.8. Понятие контроллера 39
1.4.9. Понятие представления 40
1.4.10. Маршрутизация и веб-адреса шаблона ASP.NET MVC 40
1.4.11. Язык программирования C# 41
Выводы по главе 1 ..... *81*
Вопросы для самоконтроля *82*

Глава 2. Среда разработки Visual Studio 83
2.1. Назначение Visual Studio 83
2.2. Окно запуска 84
2.3. Создание проекта 85
2.4. Проект MVC 87
2.5. Выполнение проекта MVC 91
Выводы по главе 2 ..... *92*
Вопросы для самоконтроля *92*

Глава 3. Веб-приложение MVC ASP.NET Core «Кафедра» ... 93
3.1. Создание проекта веб-приложения «Кафедра» 94
3.2. Изменения в проекте 97
3.3. Создание модели проекта Kafedra 100
3.4. Создание контекста базы данных 102
3.5. Стандартные настройки проекта 104
3.6. Создание контроллера модели Disziplina 106
3.7. Создание базы данных SQL для данных проекта Kafedra 108
3.8. Создание средств для пользователя и администратора сайта 110
3.8.1. Скрипт для табличного представления данных 111
3.8.2. Изменения в программе Index 111
3.8.3. Изменения в программе Create 114
3.8.4. Изменения в программе Edit 117
3.8.5. Режим администрирования и обычного пользователя 119
Выводы по главе 3 ..... *120*
Вопросы для самоконтроля *121*

Заключение 122

Задание для самостоятельной работы 123

Библиографический список 124

Учебное издание

Часовских Виктор Петрович,
Лабунец Валерий Григорьевич,
Стариков Евгений Николаевич
и др.

Повышение эффективности веб-технологий в системах принятия решений

Учебное пособие

Редактор и корректор *Л. В. Матвеева*
Компьютерная верстка *И. В. Засухиной*

Поз. 56. Подписано в печать 16.12.2022.
Формат 60 × 84 1/16. Бумага офсетная. Печать плоская.
Уч.-изд. л. 4,6. Усл. печ. л. 7,7. Печ. л. 8,3. Заказ 618. Тираж 24 экз.
Издательство Уральского государственного экономического университета
620144, г. Екатеринбург, ул. 8 Марта/Народной Воли, 62/45

Отпечатано с готового оригинал-макета в подразделении оперативной полиграфии
Уральского государственного экономического университета

55 лет

УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ
ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ