Приложения

A Ответы на проверочные вопросы

Это приложение содержит ответы на *проверочные вопросы*, приведенные в конце каждой главы.

Глава 1. Привет, C#! Здравствуй, .NET Core!

1. Почему на платформе .NET Core для разработки приложений программисты могут использовать разные языки, например C# и F#?

   Ответ: платформа .NET Core поддерживает разные языки, потому что для каждого из них есть компилятор, преобразующий исходный код в промежуточный язык (IL). Затем этот IL-код преобразуется в машинные инструкции для процессора CPU с помощью общеязыковой исполняющей среды (CLR).

2. Какие команды нужно ввести для создания консольного приложения?

   Ответ: `dotnet new console`.

3. Какие команды нужно ввести в окне консоли для сборки и запуска исходного кода на языке C#?

   Ответ: в папке с файлом `ProjectName.csproj` необходимо ввести команду `dotnet run`.

4. Какое сочетание клавиш используется для открытия в программе Visual Studio Code панели `TERMINAL` (Терминал)?

   Ответ: Ctrl+` (обратная кавычка) — для операционной системы macOS. Ctrl+' (одинарная кавычка) — для операционной системы Windows.

5. Среда разработки Visual Studio 2019 круче, чем Visual Studio Code?

   Ответ: нет. Каждая из них оптимизирована под разные задачи. Visual Studio 2019 мощна, многофункциональна и позволяет создавать программы с графическим пользовательским интерфейсом, к примеру мобильные приложения

Windows Forms, WPF-, UWP-приложения и Xamarin, но доступна только для операционной системы Windows. Visual Studio Code — меньше, оптимизирована под набор кода в командой строке и доступна для разных операционных систем. В 2021 году с выпуском .NET 6 и .NET *Multi-Platform App User Interface (MAUI)* среда разработки Visual Studio Code получит расширение, позволяющее создавать пользовательские интерфейсы для настольных и мобильных приложений.

6. Платформа .NET Core лучше .NET Framework?

   Ответ: зависит от того, что вам нужно. .NET Core — это сокращенная, кроссплатформенная версия полнофункциональной устаревшей версии платформы .NET Framework. NET Core периодически совершенствуется. Платформа .NET Framework стабильна и лучше поддерживает устаревшие приложения. Однако текущая версия 4.8 была последней основной версией. .NET Framework никогда не будет поддерживать некоторые языковые функции C# 8 и 9.

7. Что такое .NET Standard и почему эта технология так важна?

   Ответ: .NET Standard определяет API, который может реализовывать платформа .NET.

   Последние версии .NET Framework, .NET Core, NET 5 и Xamarin реализуют .NET Standard 2.0 для предоставления единого стандартного API, который разработчики смогут изучить и настроить. Платформы .NET Core 3.0 или более поздние версии, включая .NET 5, и Xamarin реализуют .NET Standard 2.1, который содержит некоторые новые функции, не поддерживаемые .NET Framework. Если необходимо создать новую библиотеку классов, поддерживающую все платформы .NET, вам потребуется, чтобы она была совместима с .NET Standard 2.0.

8. Как называется метод точки входа консольного приложения .NET и как его объявить?

   Ответ: точка входа консольного приложения .NET — метод `Main`. Рекомендуется использовать массив `string` для аргументов командной строки и тип возвращаемого значения `int`, хоть это и не обязательно. Это может быть объявлено так, как показано далее:

   ```
   public static void Main() // минимум
   public static int Main(string[] args) // рекомендуется
   ```

9. Как найти справочную информацию по ключевому слову C#?

   Ответ: на сайте Microsoft Docs. Ключевые слова C# также приведены на сайте https://docs.microsoft.com/ru-ru/dotnet/articles/csharp/language-reference/keywords/.

10. Как найти решения общих проблем программирования?

    Ответ: https://stackoverflow.com/.

Глава 2. Говорим на языке C#

Упражнение 2.1. Проверочные вопросы

Какой тип следует выбрать для каждого указанного ниже числа?

1. Телефонный номер.

   Ответ: `string`.

2. Рост.

   Ответ: `float` или `double`.

3. Возраст.

   Ответ: `int` для характеристики или `byte` (0–255) для размера.

4. Размер оклада.

   Ответ: `decimal`.

5. Артикул книги.

   Ответ: `string`.

6. Цена книги.

   Ответ: `decimal`.

7. Вес книги.

   Ответ: `float` или `double`.

8. Размер населения страны.

   Ответ: `uint` (от 0 до примерно 4 миллиардов).

9. Количество звезд во Вселенной.

   Ответ: `ulong` (от 0 до примерно 18 квадриллионов) или `System.Numerics.BigInteger` (допускает произвольно большие целые числа).

10. Количество сотрудников на каждом из предприятий малого или среднего бизнеса (примерно до 50 000 сотрудников на каждом предприятии).

    Ответ: поскольку существуют сотни тысяч малых и средних предприятий, нам необходимо использовать размер памяти в качестве определяющего фактора, поэтому выберите `ushort`, так как этот тип занимает всего 2 байта, в отличие от `int`, который занимает 4 байта.

Глава 3. Управление потоками и преобразование типов

Упражнение 3.1. Проверочные вопросы

Ответьте на следующие вопросы.

1. Что произойдет, если разделить переменную `int` на `0`?

   Ответ: вызов исключения `DivideByZeroException` при делении целого или дробного числа.

2. Что произойдет, если разделить переменную `double` на `0`?

   Ответ: тип `double` содержит особое значение `Infinity`. Экземпляры чисел с плавающей запятой могут иметь специальные значения — `NaN` (не число), `PositiveInfinity` и `NegativeInfinity`.

3. Что происходит при переполнении переменной `int`, то есть вы присваиваете ей значение, выходящие за пределы допустимого диапазона?

   Ответ: он будет работать в цикле, если вы не поместите оператор в блок `checked`. В последнем случае будет вызвано исключение `OverflowException`.

4. В чем разница между операторами `x = y++;` и `x = ++y;`?

   Ответ: в случае с операторами `x = y++;` будет присвоено значение `x`, а затем инкрементировано значение `y`. В случае с операторами `x = ++y;` сначала будет инкрементировано значение `y`, а затем результат присвоен переменной `x`.

5. В чем разница между ключевыми словами `break`, `continue` и `return` при использовании в операторах цикла?

   Ответ: оператор `break` завершает весь цикл и продолжает выполнение кода после цикла, оператор `continue` завершит текущую итерацию цикла и продолжает выполнение с начала цикла (следующая итерация), а оператор `return` завершит текущий вызов метода и продолжает выполнение после вызова метода.

6. Из каких трех частей состоит оператор `for` и какие из них обязательны?

   Ответ: три части оператора `for` — это инициализатор, условие и инкремент. Условие требуется обязательно и должно быть логическим выражением, которое возвращает значение `true` или `false`, а две другие части опциональны.

7. В чем разница между операторами `=` и `==`?

   Ответ: `=` — оператор присваивания для назначения значений переменным, а `==` — оператор равенства, возвращающий значение `true` или `false`.

8. Будет ли скомпилирован следующий оператор: `for ( ; true; ) ;`?

Ответ: Да. Для оператора `for` требуется только логическое выражение. Выражения `initializer` и `incrementer` опциональны и могут быть опущены. Оператор `for` будет выполняться непрерывно. Это пример бесконечного цикла.

9. Что означает символ подчеркивания `_` в выражении `switch`?

   Ответ: символ подчеркивания `_` представляет возвращаемое по умолчанию значение.

10. Какой интерфейс должен реализовать объект для перечисления с помощью оператора `foreach`?

    Ответ: объект должен реализовывать интерфейс `IEnumerable`.

Упражнение 3.2. Циклы и переполнение

Что произойдет при выполнении кода, приведенного ниже?

```
int max = 500;
for (byte i = 0; i < max; i++)
{
  WriteLine(i);
}
```

Ответ: код будет циклически работать без остановки, так как значение переменной `i` может быть только в диапазоне от `0` до `255`, поэтому, когда оно инкрементируется с шагом `255`, то возвращается к `0` и поэтому всегда будет меньше, чем `max (500)`.

Чтобы предотвратить непрерывное выполнение цикла, вы можете обернуть код оператором `checked`. Это приведет к тому, что исключение будет вызываться после достижения значения `255`, например:

```
254
255
System.OverflowException says Arithmetic operation resulted in an
overflow.
```

Упражнение 3.5. Проверка знаний операторов

Чему будут равны значения `x` и `y` после выполнения следующих операторов?

1. Чему равны значения `x` и `y` после выполнения следующих операторов?

   ```
   x = 3;
   y = 2 + ++x;
   ```

   Ответ: `x` равен `4`; `y` равен `6`.

2. Чему равны значения x и y после выполнения следующих операторов?

```
x = 3 << 2;
y = 10 >> 1;
```

Ответ: x равен 12; y равен 5.

3. Чему равны значения x и y после выполнения следующих операторов?

```
x = 10 & 8;
y = 10 | 7;
```

Ответ: x равен 8; y равен 15.

Глава 4. Разработка, отладка и тестирование функций

1. Что означает в языке C# ключевое слово void?

Ответ: ключевое слово void указывает на то, что метод не имеет возвращаемого значения.

2. В чем разница между императивным и функциональным стилями программирования?

Ответ: императивный стиль программирования — это процесс, который описывает процесс вычисления в виде инструкций. Написанный код сообщает среде выполнения, как именно выполнять задачу. Например, сначала необходимо выполнить шаг 1, затем шаг 2. Это парадигма программирования, в которой используются утверждения, означающие, что состояние программы может измениться в любой момент, в том числе вне текущей функции. Императивное программирование вызывает побочные эффекты, изменяя значение некоторого состояния вашей программы. Побочные эффекты сложно отладить. Стиль функционального программирования описывает то, чего вы хотите достичь, а не способ достижения результата. Данный стиль также можно назвать декларативным. Наиболее важный момент — это то, что во избежание побочных эффектов языки функционального программирования по умолчанию делают все состояния неизменяемыми.

3. В программе Visual Studio Code какова разница между сочетаниями клавиш F5, Ctrl или Cmd+F5, Shift+F5 и Ctrl или Cmd+Shift+F5?

Ответ: нажатие клавиши F5 сохраняет, компилирует, запускает и присоединяет отладчик, сочетания клавиш Ctrl или Cmd+F5 — сохраняет, компилирует и запускает отладчик, сочетания клавиш Shift+F5 — останавливает отладчик, а сочетания клавиш Ctrl или Cmd+Shift+F5 — перезапускает отладчик.

4. Куда записывает выходные данные метод `Trace.WriteLine`?

   Ответ: метод `Trace.WriteLine` записывает свои выходные данные в любые настроенные прослушиватели трассировки. По умолчанию это включает терминал Visual Studio Code или командную строку, но его можно настроить как текстовый файл или любой пользовательский прослушиватель.

5. Каковы пять уровней трассировки?

   Ответ: `0` = `None`, `1` = `Error` (Ошибка), `2` = `Warning` (Предупреждение), `3` = `Info` (Информация) и `4` = `Verbose` (Подробная информация).

6. В чем разница между классами `Debug` и `Trace`?

   Ответ: класс `Debug` активен только во время разработки. Класс `Trace` активен во время разработки и после выпуска в рабочую среду.

7. Как называются три А модульного теста?

   Ответ: Arrange, Act, Assert — размещение, действие, утверждение.

8. При написании модульного теста с использованием `xUnit` каким атрибутом вы должны дополнять методы тестирования?

   Ответ: `[Fact]`

9. Какая команда `dotnet` выполняет тесты `xUnit`?

   Ответ: `dotnet test`

10. Что такое TDD?

    Ответ: Test Driven Development — разработка через тестирование.

Вы можете прочитать о TDD на сайте https://docs.microsoft.com/ru-ru/dotnet/core/testing/.

Глава 5. Создание пользовательских типов с помощью объектно-ориентированного программирования

1. Какие шесть модификаторов доступа вы знаете и для чего они используются?

   Ответ: ниже в списке перечислены шесть модификаторов доступа и их значение:

   - `private`: доступ ограничен содержащим типом;
   - `internal`: доступ ограничен содержащим типом и любым другим типом в текущей сборке;
   - `protected`: доступ ограничен содержащим классом или типами, которые являются производными от содержащего класса;

- `internal protected`: доступ ограничен содержащим типом, производным классом или другим типом в текущей сборке;
- `private protected`: доступ ограничен содержащим типом или производным классом и другим типом в текущей сборке;
- `public`: неограниченный доступ.

2. В чем разница между разными ключевыми словами — `static`, `const` и `readonly`?

   Ответ: ниже в списке описана разница между ключевыми словами `static`, `const` и `readonly` применительно к элементу типа:

   - `static`: создает элемент, общий для всех экземпляров и с доступом из типа;
   - `const`: устанавливает поле как фиксированное литеральное значение, которое никогда не должно меняться, так как во время компиляции сборки операторы, использующие это поле, копируют литеральное значение;
   - `readonly`: создает поле, которое может быть назначено только во время выполнения с помощью конструктора.

3. Для чего используется конструктор?

   Ответ: конструктор служит для выделения памяти и инициализации значений полей.

4. Зачем с ключевым словом `enum` используется атрибут `[Flags]`, если требуется хранить комбинированные значения?

   Ответ: если не применить атрибут `[Flags]` к типу `enum` при сохранении скомбинированных значений, то сохраненное значение `enum`, представляющее собой комбинацию, будет возвращаться в качестве сохраненного целочисленного значения вместо списка текстовых значений, разделенных запятыми.

5. В чем польза ключевого слова `partial`?

   Ответ: вы можете использовать ключевое слово `partial` для разделения определения типа на несколько файлов.

6. Что вы знаете о кортежах?

   Ответ: это структура данных, состоящая из нескольких частей. Они используются при необходимости сохранить несколько значений, при этом не определяя тип.

7. Для чего служит ключевое слово C# `record`?

   Ответ: ключевое слово `record` определяет неизменяемую структуру данных для обеспечения более функционального стиля программирования. Как и класс, `record` может содержать свойства и методы, но значения свойств могут быть установлены только во время инициализации.

8. Что такое перегрузка?

   Ответ: перегрузка — это когда вы определяете более одного метода с одинаковым именем метода и разными входными параметрами.

9. В чем разница между полем и свойством?

   Ответ: поле — это место хранения данных, на которое можно ссылаться. Свойство — это один или пара методов, которые получают и/или устанавливают значение. Значение свойства часто хранится в закрытом поле.

10. Как сделать параметр метода необязательным?

    Ответ: вы можете сделать параметр метода необязательным, присвоив ему значение по умолчанию в сигнатуре метода.

Глава 6. Реализация интерфейсов и наследование классов

1. Что такое делегат?

   Ответ: делегат — это типобезопасный указатель на метод. Его можно использовать для выполнения любого метода с соответствующей сигнатурой.

2. Что такое событие?

   Ответ: событие — это поле, представляющее собой делегат, к которому применено ключевое слово `event`. Оно гарантирует, что используются только операторы `+=` и `-=` для безопасного объединения нескольких делегатов без замены каких-либо существующих обработчиков событий.

3. Как связаны базовый и производный классы и как производный класс может получить доступ к базовому классу?

   Ответ: производный класс (подкласс) — это класс, унаследованный от базового класса (суперкласса). Для доступа к классу, от которого наследуется подкласс, внутри производного класса необходимо использовать ключевое слово `base`.

4. В чем разница между ключевыми словами `is` и `as`?

   Ответ: оператор `is` возвращает значение `true`, если объект может быть приведен к типу; в противном случае возвращает значение `false`. Оператор `as` возвращает указатель, если объект может быть приведен к типу. В противном случае возвращается значение `null`.

5. Какое ключевое слово используется для предотвращения наследования класса и переопределения метода?

   Ответ: `sealed`.

Дополнительную информацию можно найти на сайте https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/sealed.

6. Какое ключевое слово используется для предотвращения создания экземпляра класса с помощью нового ключевого слова `new`?

   Ответ: `abstract`.

Дополнительную информацию можно найти на сайте https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/abstract.

7. Какое ключевое слово используется для переопределения члена?

   Ответ: `virtual`.

Дополнительную информацию можно найти на сайте https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/virtual.

8. Чем деструктор отличается от деконструктора?

   Ответ: *деструктор*, также известный как *финализатор*, должен использоваться для выпуска ресурсов, принадлежащих этому объекту. *Деконструктор* — новая функция C# 7 или более поздних версий, позволяющая разбивать сложный объект на более мелкие части. Это особенно полезно при работе с кортежами.

Дополнительную информацию можно найти на сайте https://docs.microsoft.com/ru-ru/dotnet/csharp/deconstruct.

9. Как выглядят сигнатуры конструкторов, которые должны иметь все исключения?

   Ответ: ниже приведены сигнатуры конструкторов, которые должны содержать все исключения:

   - конструктор без параметров;
   - конструктор со строковым параметром, обычно называемым `message`;
   - конструктор со строковым параметром, обычно называемым `message`, и параметром исключения, обычно называемым `innerException`.

10. Что такое метод расширения и как его определить?

    Ответ: метод расширения — это способ действия компилятора, который делает статический метод статического класса одним из членов типа. Расширяемый тип определяется с помощью префикса `this`.

Глава 7. Описание и упаковка типов .NET

1. В чем разница между пространством имен и сборкой?

   Ответ: *пространство имен* — это логический контейнер типа. *Сборка* — это физический контейнер типа. Чтобы использовать тип, разработчик должен ссылаться на его сборку. При желании разработчик может импортировать свое пространство имен или указать пространство имен при именовании типа.

2. Как вы ссылаетесь на другой проект в файле `.csproj`?

```
<ItemGroup>
  <ProjectReference Include="..\Calculator\Calculator.csproj" />
</ItemGroup>
```

3. В чем преимущество такого инструмента, как ILSpy?

   Ответ: преимущество такого инструмента, как ILSpy, заключается в том, чтобы научиться писать код на C# для платформы .NET, наблюдая за созданием других пакетов. Конечно, не используйте их интеллектуальную собственность. Но особенно полезно ознакомиться с тем, как разработчики Microsoft реализовали ключевые компоненты библиотек базовых классов.

4. Какой тип .NET представлен псевдонимом `float` в C#?

   Ответ: `System.Single`.

5. Какой инструмент следует использовать перед переносом приложения с .NET Framework на .NET 5?

   Ответ: перед переносом приложения с .NET Framework на .NET 5 следует использовать .NET Portability Analyzer.

6. В чем разница между платформозависимым и автономным развертыванием приложений .NET?

   Ответ: зависящие от платформы приложения .NET Core требуют наличия .NET Core для выполнения операционной системой. Автономные приложения .NET Core включают в себя все необходимое для самостоятельного выполнения.

7. Что означает RID?

   Ответ: RID — сокращение от Runtime Identifier (идентификатор среды выполнения). Значения RID используются для определения целевых платформ, на которых выполняется приложение .NET Core.

8. В чем разница между командами `dotnet pack` и `dotnet publish`?

   Ответ: с помощью команды `dotnet pack` создается пакет NuGet. С помощью команды `dotnet publish` приложение и его зависимости помещаются в папку для развертывания в хост-систему.

9. Какие типы приложений, написанных для .NET Framework, можно перенести на .NET Core?

Ответ: консоль, ASP.NET MVC, ASP.NET Web API, Windows Forms и Windows Presentation Foundation (WPF).

10. Можете ли вы использовать пакеты, написанные для .NET Framework, с .NET 5?

    Ответ: да, если они вызывают только API в .NET Standard 2.0.

Глава 8. Работа с распространенными типами .NET

1. Какое максимальное количество символов может быть сохранено в переменной типа `string`?

   Ответ: максимальный размер переменной `string` может быть равен 2 Гбайт, или примерно 1 миллиарду символов, потому что каждая переменная `char` использует 2 байта из-за внутреннего использования для символов кодировки Unicode (UTF-16).

2. В каких случаях и почему нужно использовать тип `SecureString`?

   Ответ: строковый тип хранит текстовые данные в памяти слишком долго, и при этом они не защищены. Тип `SecureString` шифрует текст и гарантирует немедленное освобождение памяти. Например, элемент `PasswordBox` сохраняет пароль в виде переменной `SecureString`, а при запуске нового процесса параметр `Password` должен быть переменной `SecureString`.

Для получения более подробной информации посетите сайт https://stackoverflow.com/questions/141203/when-would-i-need-a-securestring-in-net.

3. В каких ситуациях целесообразно применить тип `StringBuilder`?

   Ответ: при конкатенации свыше трех переменных `string` с помощью типа `StringBuilder` можно снизить потребление ресурсов памяти и улучшить производительность в сравнении со способами, предусматривающими использование метода `string.Concat` и оператора `+`.

4. В каких случаях следует задействовать `LinkedList<T>`?

   Ответ: каждый элемент в связанном списке содержит ссылку на предыдущие и следующие одноуровневые элементы, а также и на сам список, поэтому его следует использовать, когда элементы необходимо вставлять и удалять из позиций в списке, не перемещая элементы в памяти.

5. Когда класс `SortedDictionary<T>` нужно использовать вместо класса `SortedList<T>`?

   Ответ: класс `SortedList<T>` использует меньше памяти, чем `SortedDictionary<T>`, а `SortedDictionary<T>` быстрее выполняет операции добавления и удаления

неотсортированных данных. Если список заполняется с помощью уже отсортированных данных, `SortedList<T>` работает быстрее, чем `SortedDictionary<T>`.

Для получения более подробной информации посетите сайт https://stackoverflow.com/questions/935621/whats-the-difference-between-sortedlist-and-sorteddictionary.

6. Каков ISO-код языковых и региональных параметров ISO для валлийского языка?

   Ответ: `cy-GB`.

Полный список кодов языковых и региональных параметров можно найти по адресу https://lonewolfonline.net/list-net-culture-country-codes/.

7. В чем разница между локализацией, глобализацией и интернационализацией?

   Ответ

   - *Локализация* оказывает влияние на пользовательский интерфейс приложения. Локализация определяется нейтральными (только языковыми) или специфичными (языковыми и региональными) настройками. Вы предоставляете текст и другие значения в нескольких языковых версиях. К примеру, метка текстового поля с именем может быть отображена как *First name* на английском языке и *Prénom* на французском языке.
   - *Глобализация* определяет данные вашего приложения. Глобализация определяется языковыми и региональными настройками, к примеру `en-GB` для британской версии английского языка или `fr-CA` — для канадской версии французского языка. Эти настройки должны быть определены для правильного форматирования десятичных значений денежных единиц, например канадских долларов вместо французских евро.
   - *Интернационализация* — это сочетание локализации и глобализации.

8. Что означает символ `$` в регулярных выражениях?

   Ответ: в регулярных выражениях символ `$` представляет конец ввода.

9. Как в регулярных выражениях представить цифры?

   Ответ: в регулярном выражении вы можете представлять цифры с помощью `\d` или `[0-9]`.

10. Почему *нельзя* использовать официальный стандарт для адресов электронной почты при создании регулярного выражения для проверки адреса электронной почты пользователя?

Ответ: результат того не стоит. Проверка адреса электронной почты с использованием официальной спецификации не позволяет убедиться, действительно ли этот адрес существует или является ли человек, указавший адрес, его владельцем.

Для получения более подробной информации посетите сайты https://davidcel.is/posts/stop-validating-email-addresses-with-regex/ и https://stackoverflow.com/questions/201323/how-to-validate-an-email-address-using-a-regular-expression.

Глава 9. Работа с файлами, потоками и сериализация

1. Чем применение класса `File` отличается от использования класса `FileInfo`?

   Ответ: класс `File` содержит статические методы, поэтому его экземпляр не может быть создан. Он лучше всего подходит для одноразовых задач, таких как копирование файла. Класс `FileInfo` требует создания экземпляра объекта, представляющего файл. Его лучше всего использовать, когда нужно выполнить несколько операций с одним и тем же файлом.

2. В чем разница между методами потока `ReadByte` и `Read`?

   Ответ: метод `ReadByte` при каждом вызове возвращает один байт, а метод `Read` заполняет временный массив байтами до указанной длины. Обычно рекомендуется использовать метод `Read` для обработки сразу последовательности байтов.

3. В каких случаях применяются классы `StringReader`, `TextReader` и `StreamReader`?

   Ответ

   - Класс `StringReader` используется для эффективного чтения из строки, хранящейся в памяти.
   - `TextReader` — это абстрактный класс, который наследуют классы `StringReader` и `StreamReader` для совместной функциональности.
   - Класс `StreamReader` используется для чтения строк из потока, который может быть текстовым файлом любого типа, включая форматы XML и JSON.

4. Для чего предназначен тип `DeflateStream`?

   Ответ: тип `DeflateStream` реализует тот же алгоритм сжатия, что и GZIP, но без циклического избыточного кода, поэтому, хотя и создает меньшие по размеру сжатые файлы, он не может выполнять проверку целостности данных при распаковке.

5. Сколько байтов на символ затрачивается при использовании кодировки UTF-8?

   Ответ: количество байтов на символ, используемое кодировкой UTF-8, зависит от символа. Большинство символов латинского алфавита хранятся с использованием одного байта. Другим символам может потребоваться два и более байта для хранения.

6. Что такое граф объектов?

   Ответ: графом объектов является любой экземпляр классов в памяти, которые ссылаются друг на друга, тем самым формируя набор связанных объектов. Например, объект `Customer` может иметь свойство, которое ссылается на набор экземпляров `Order`.

7. Какой формат сериализации лучше всего подходит для минимизации затрат памяти?

   Ответ: объектная нотация JavaScript (JSON) имеет хороший баланс между требованиями к пространству и практическими факторами, такими как удобочитаемость. Однако буферы протокола лучше всего подходят для минимизации требований к пространству.

Дополнительную информацию вы можете найти на сайте https://stackoverflow.com/questions/52409579/protocol-buffer-vs-json-when-to-choose-one-over-another

8. Какой формат сериализации лучше всего подходит для кросс-платформенной совместимости?

   Ответ: расширяемый язык разметки (XML), хотя сегодня JSON еще более эффективен, особенно если вам необходимо интегрироваться с веб-системами или буферами протокола для лучшей производительности и использования минимальной полосы пропускания.

9. Почему не рекомендуется использовать строковое значение типа `"\Code\Chapter01"` для представления пути и что необходимо выполнить вместо этого?

   Ответ: неправильно использовать строковое значение, например `"\Code\Chapter01"`, для представления пути, так как предполагается, что символ «обратный слеш» используется в качестве разделителя папок во всех операционных системах. Вместо этого вы должны применять метод `Path.Combine` и передавать отдельные строковые значения для каждой папки, как показано в коде ниже:

```
string path = Path.Combine(new[] { "Code", "Chapter01" });
```

10. Где вы можете найти информацию о пакетах NuGet и их зависимостях?

    Ответ: дополнительную информацию о пакетах NuGet и их зависимостях можно найти на сайте https://www.nuget.org/.

Глава 10. Защита данных и приложений

1. Какой из алгоритмов шифрования, доступных на платформе .NET, лучше всего подойдет для симметричного шифрования?

   Ответ: для симметричного шифрования лучше всего подойдет алгоритм AES.

2. Какой из алгоритмов шифрования, доступных на платформе .NET, лучше всего подойдет для асимметричного шифрования?

   Ответ: для асимметричного шифрования лучше всего подойдет алгоритм RSA.

3. Что такое радужная атака?

   Ответ: радужная атака использует таблицу предварительно рассчитанных хешей паролей. Когда база данных хешей паролей украдена, злоумышленник может быстро сравнить хеши радужной таблицы и определить оригинальные пароли.

Дополнительную информацию о радужных таблицах вы можете найти на сайте https://learncryptography.com/hash-functions/rainbow-tables.

4. При использовании алгоритмов шифрования лучше использовать блоки большого или малого размера?

   Ответ: при использовании алгоритмов шифрования лучше использовать блоки малого размера.

5. Что означает хеширование данных?

   Ответ: хеш — это вывод фиксированного размера, который получается в результате ввода произвольного размера, обрабатываемого хеш-функцией. Хеш-функции — односторонние, это означает, что единственный способ воссоздать исходные данные — перебор всех возможных входных данных и сравнение результатов.

6. Что означает подписывание данных?

   Ответ: подпись — это значение, добавляемое к цифровому документу для подтверждения его подлинности. Действительная подпись сообщает получателю, что документ был создан известным отправителем.

7. В чем разница между симметричным и асимметричным шифрованием?

   Ответ: симметричное шифрование использует секретный общий ключ для шифрования и дешифрования. Асимметричное шифрование использует открытый ключ для шифрования и закрытый ключ для дешифрования.

8. Что означает RSA?

   Ответ: Rivest — Shamir — Adleman (Ривест — Шамир — Адлеман), фамилии трех создателей криптографического алгоритма, созданного в 1978 году.

9. Почему пароли должны быть засолены перед сохранением?

   Ответ: для замедления радужных словарных атак.

10. SHA1 — это алгоритм хеширования, разработанный Национальным агентством безопасности США. Почему не следует использовать данный алгоритм?

    Ответ: алгоритм SHA-1 — небезопасный. Все современные браузеры перестали принимать SSL-сертификаты алгоритма SHA-1.

Глава 11. Работа с базами данных с помощью Entity Framework Core

1. Какой тип вы бы использовали для свойства, представляющего таблицу, например для свойства `Products` контекста базы данных?

   Ответ: `DbSet<T>`, где `T` — тип объекта, например `Product`.

2. Какой тип вы бы применили для свойства, которое представляет отношение «один ко многим», скажем, свойство `Products` объекта `Category`?

   Ответ: `ICollection<T>`, где `T` — тип объекта, например `Product`.

3. Какое соглашение, касающееся первичных ключей, действует в EF?

   Ответ: предполагается, что свойство с именем `ID` или `ClassNameID` является первичным ключом. Если тип этого свойства является одним из следующих, то свойство также обозначается как столбец `IDENTITY`: `tinyint`, `smallint`, `int`, `bigint`, `guid`.

4. Когда бы вы воспользовались атрибутом аннотаций в классе сущности?

   Ответ: вы должны в классе элемента использовать атрибут аннотации, если соглашения не могут определить правильное сопоставление между классами и таблицами. Например, если имя класса не соответствует имени таблицы или имя свойства не соответствует имени столбца. Вы также можете определить ограничения, такие как максимальная длина символов в текстовом значении или диапазон числовых значений, добавив атрибуты проверки.

5. Почему вы предпочли бы использовать Fluent API, а не атрибуты аннотации?

   Ответ: вы можете выбрать Fluent API, а не атрибуты аннотации, если соглашения не позволяют определить правильное сопоставление между классами и таблицами или если требуется, чтобы классы элементов были чистыми и свободными от постороннего кода. К примеру, при создании библиотеки классов .NET Standard 2.0 для классов сущностей вы можете использовать только атрибуты проверки сущностей, чтобы эти метаданные поддерживались такими технологиями, как Entity Framework Core и проверка привязки модели ASP.NET Core, а также настольными Windows- и мобильными приложениями. Однако вы можете использовать Fluent API для определения функциональных

возможностей Entity Framework Core, таких как сопоставление с другой таблицей или именем столбца.

6. Что означает уровень изолированности транзакции Serializable?

   Ответ: максимальные блокировки применяются для обеспечения полной изоляции от любых других процессов, работающих с затронутыми данными.

7. Что возвращает в результате метод `DbContext.SaveChanges()`?

   Ответ: значение `int` для числа затронутых объектов.

8. В чем разница между жадной и явной загрузкой?

   Ответ: жадная загрузка означает, что связанные объекты включены в исходный запрос к базе данных, поэтому их не нужно загружать позже. Явная загрузка означает, что связанные объекты не включены в исходный запрос к базе данных и должны быть явно загружены непосредственно перед тем, как они понадобятся.

9. Как определить сущностный класс EF Core для соответствия следующей таблице?

```
CREATE TABLE Employees(
  EmpID INT IDENTITY,
  FirstName NVARCHAR(40) NOT NULL,
  Salary MONEY
)
```

   Ответ: используйте следующий класс:

```
public class Employee
{
  [Column("EmpID")]
  public int EmployeeID { get; set; }
  [Required] [StringLength(40)]
  public string FirstName { get; set; }
  [Column(TypeName = "money")]
  public decimal? Salary { get; set; }
}
```

10. Какую выгоду вы получаете от объявления свойств навигации как `virtual`?

    Ответ: вы можете включить ленивую загрузку, если объявите свойства навигации сущностей в качестве виртуальных.

Глава 12. Создание запросов и управление данными с помощью LINQ

1. Каковы две обязательные составные части LINQ?

   Ответ: поставщик данных LINQ и методы расширения LINQ. Для доступа к методам расширения LINQ вы должны импортировать пространство имен `System.`

`Linq`, а затем обращаться к сборке поставщика того типа данных, с которыми вы хотите работать.

2. Какой метод расширения LINQ вы использовали бы для возврата из типа набора свойств?

   Ответ: метод `Select` позволяет осуществлять проекцию (выбор) свойств.

3. Какой метод расширения LINQ вы применили бы для фильтрации последовательности?

   Ответ: метод `Where` позволяет выполнить фильтрование путем предоставления делегата (или лямбда-выражения), который возвращает логическое значение, показывающее, должно ли значение быть включено в результаты.

4. Перечислите пять методов расширения LINQ, выполняющих агрегацию данных.

   Ответ: любые пять из следующих: `Max`, `Min`, `Count`, `LongCount`, `Average`, `Sum` и `Aggregate`.

5. Чем различаются методы расширения `Select` и `SelectMany`?

   Ответ: метод `Select` возвращает именно то, что вы указали для возврата. Метод `SelectMany` проверяет, не являются ли выбранные вами элементы `IEnumerable<T>`, а затем разбивает их на более мелкие части. Например, если выбранный вами тип — строковое значение (то есть `IEnumerable<char>`), то метод `SelectMany` разобьет каждое возвращенное строковое значение на соответствующие отдельные значения `char`.

6. В чем разница между интерфейсом `IEnumerable<T>` и `IQueryable<T>`? и как вы переключаетесь между ними.

   Ответ: интерфейс `IEnumerable<T>` указывает поставщика LINQ, который будет выполнять запрос локально, как LINQ to Objects. Эти поставщики не имеют ограничений, но могут быть менее эффективными. Интерфейс `IQueryable<T>` указывает поставщика LINQ, который сначала создает дерево выражений для представления запроса, а затем преобразует его в другой синтаксис запроса перед его выполнением, подобно тому как Entity Framework Core преобразует LINQ to SQL. Эти поставщики иногда имеют ограничения и создают исключения. Вы можете преобразовать из поставщика `IQueryable<T>` в поставщика `IEnumerable<T>`, вызвав метод `AsEnumerable`.

7. Что представляет собой последний параметр типа в делегатах-дженериках `Func`?

   Ответ: последний параметр типа в общих делегатах `Func` представляет тип возвращаемого значения. Например, для `Func<string, int, bool>` используемая функция делегата или лямбда-функции должны возвращать логическое значение.

8. В чем преимущество метода расширения LINQ, который заканчивается оператором `OrDefault`?

Ответ: преимущество метода расширения LINQ, который заканчивается оператором `OrDefault`, заключается в том, что он возвращает значение по умолчанию, а не выдает исключение, если не может вернуть значение. Например, вызов метода `First` для последовательности значений `int` вызовет исключение, если коллекция пуста, но метод `FirstOrDefault` вернет в результате `0`.

9. Почему понятный синтаксис запросов необязателен?

   Ответ: синтаксис понимания запросов — необязательный, так как это просто синтаксический сахар. Он облегчает чтение кода разработчиками, но не добавляет никакой дополнительной функциональности.

Более подробно о ключевом слове let вы можете прочитать на сайте https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/let-clause.

10. Каким образом вы можете создать свои собственные методы расширения LINQ?

    Ответ: создайте класс `static` со статическим методом с параметром `IEnumerable<T>` с префиксом `this`.

```
namespace System.LINQ
{
  public static class MyLINQExtensionMethods
  {
    public static IEnumerable<T> MyChainableExtensionMethod<T>(
      this IEnumerable<T> sequence)
    {
    // вернуть что-нибудь IEnumerable<T>
    }
    public static int? MyAggregateExtensionMethod<T>(
      this IEnumerable<T> sequence)
    {
    // вернуть некоторое значение int
    }
  }
}
```

Глава 13. Улучшение производительности и масштабируемости с помощью многозадачности

1. Какую информацию о классе `Process` вы можете найти?

   Ответ: класс `Process` содержит ряд свойств, включая `ExitCode`, `ExitTime`, `Id`, `MachineName`, `PagedMemorySize64`, `ProcessorAffinity`, `StandardInput`, `StandardOutput`, `StartTime`, `Threads` и `TotalProcessorTime`.

Дополнительную информацию о классе Process вы можете найти на сайте https://docs.microsoft.com/ru-ru/dotnet/api/system.diagnostics.process.

2. Насколько точен класс `Stopwatch`?

   Ответ: класс `Stopwatch` может быть точным до наносекунды (миллиардной доли секунды), но все равно может быть погрешность.

Дополнительную информацию о том, как повысить точность, установив привязку процессора вы можете найти на сайте https://www.codeproject.com/Articles/61964/Performance-Tests-Precise-Run-Time-Measurements-wi.

3. По соглашению какой суффикс должен быть применен к методу, если он возвращает `Task` или `Task<T>`?

   Ответ: `Async`. Например, `OpenAsync` при использовании метода с именем `Open`.

4. Какое ключевое слово следует применить к объявлению метода, чтобы в нем можно было использовать ключевое слово `await`?

   Ответ: к объявлению метода следует применить ключевое слово `async`.

5. Как создать дочернюю задачу?

   Ответ: для создания дочерней задачи необходимо вызвать метод `Task.Factory.StartNew` с параметром `TaskCreationOptions.AttachToParent`.

6. Почему не следует использовать ключевое слово `lock`?

   Ответ: ключевое слово `lock` не позволяет указать время ожидания, что может привести к взаимоблокировке. Вместо этого используйте метод `Monitor.Enter` со структурой `TimeSpan` и метод `Monitor.Exit`.

7. В каких случаях нужно применять класс `Interlocked`?

   Ответ: класс `Interlocked` нужно использовать, если в коде используются целые числа и числа с плавающей запятой, распределяемые между несколькими потоками.

8. Когда класс `Mutex` следует задействовать вместо класса `Monitor`?

   Ответ: используйте класс `Mutex`, когда вам необходимо поделиться ресурсом через границы процесса. Класс `Monitor` работает только с ресурсами внутри текущего процесса.

9. В чем преимущество использования на сайте или в веб-сервисе методов `async` и `await`?

   Ответ: на сайте или в веб-сервисе использование методов `async` и `await` улучшает масштабируемость, но не производительность конкретного запроса, поскольку требуется дополнительная работа по обработке потоков.

10. Вы можете отменить задачу? Каким образом?

    Ответ: да, можно отменить задачу. Как это сделать, описано на сайте https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/async/cancel-an-async-task-or-a-list-of-tasks.

Глава 15. Разработка сайтов с использованием ASP.NET Core Razor Pages

1. Перечислите шесть методов, которые могут быть указаны в HTTP-запросе.

    Ответ: GET, HEAD, POST, PUT, PATCH, DELETE. Другие включают TRACE, OPTIONS и CONNECT.

Дополнительную информацию об определении HTTP-методов вы можете найти на сайте https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

2. Перечислите шесть кодов состояния и их описания, которые могут быть возвращены в HTTP-ответе.

    Ответ: 200 OK, 201 Created, 301 Moved Permanently, 400 Bad Request, 404 Not Found (отсутствует ресурс), 500 Internal Server Error. Другие включают 101 Switching Protocols (например, с HTTP на WebSocket), 202 Accepted, 204 No Content, 304 Not Modified, 401 Unauthorized, 403 Forbidden, 406 Not Acceptable (например, запрашивается формат ответа, который не поддерживается сайтом), 503 Service Unavailable.

Дополнительную информацию об определениях кода состояния вы можете найти на сайте https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

3. Для чего в ASP.NET Core используется класс `Startup`?

    Ответ: в ASP.NET Core класс `Startup` используется для добавления и настройки служб в конвейере запросов и ответов, таких как обработка ошибок, параметры безопасности, статические файлы, файлы по умолчанию, маршрутизация конечных точек, Razor Pages и MVC и контексты данных Entity Framework Core.

4. Что такое HSTS и для чего он предназначен?

    Ответ: HTTP Strict Transport Security (HSTS) — дополнительный механизм усовершенствования безопасности. Если сайт указывает его и браузер его поддерживает, то он заставляет все коммуникации по HTTPS и не позволяет пользователю задействовать недоверенные или недействительные сертификаты.

5. Как включить статические HTML-страницы для сайта?

   Ответ: чтобы включить статические HTML-страницы для сайта, необходимо добавить операторы в метод `Configure` класса `Startup`, чтобы использовать файлы по умолчанию, а затем — статические файлы (этот порядок важен!), как показано ниже:

```
app.UseDefaultFiles(); // index.html, default.html и т. д.
app.UseStaticFiles();
```

6. Как вставить код C# в HTML, чтобы создать динамическую страницу?

   Ответ: чтобы смешать код языка C# с HTML и создать динамическую страницу, вы можете создать файл Razor с расширением `.cshtml`. Затем вы можете добавить префикс любых выражений C# с символом `@`, а операторы C# заключить в фигурные скобки или создать раздел `@functions`, как показано ниже:

```
@page
@functions
{
  public string[] DaysOfTheWeek
  {
    get => System.Threading.Thread.CurrentThread
      .CurrentCulture.DateTimeFormat.DayNames;
  }

  public string WhatDayIsIt
  {
    get => System.DateTime.Now.ToString("dddd");
  }
}
<html>
<head>
  <title>Today is @WhatDayIsIt</title>
</head>
<body>
  <h1>Days of the week in your culture</h1>
  <ul>
@{
// чтобы добавить блок операторов, используйте фигурные скобки
foreach (string dayName in DaysOfTheWeek)
  {
  <li>@dayName</li>
  }
}
</ul>
</body>
</html>
```

7. Как можно определить общие макеты для Razor Pages?

   Ответ: чтобы определить общие макеты для Razor Pages, создайте как минимум два файла: `_Layout.cshtml` для определения разметки для общего макета и `_ViewStart.cshtml` для установки макета по умолчанию:

   ```
   @{
     Layout = "_Layout";
   }
   ```

8. Как можно отделить разметку от кода на странице Razor?

   Ответ: чтобы отделить разметку от кода на странице Razor, создайте два файла: `MyPage.cshtml`, который содержит разметку, и `MyPage.cshtml.cs`, содержащий класс, наследуемый от `PageModel`. В файле `MyPage.cshtml` установите модель для использования класса:

   ```
   @page
   @model MyProject.Pages.MyPageModel
   ```

9. Как настроить контекст данных Entity Framework Core для использования с сайтом ASP.NET Core?

   Ответ: чтобы настроить контекст данных Entity Framework Core для использования с сайтом ASP.NET Core, необходимо выполнить следующее:

   - в файле проекта укажите ссылку на сборку, определяющую класс контекста данных;
   - в классе `Startup` импортируйте пространства имен для `Microsoft.EntityFrameworkCore` и класс контекста данных;
   - в методе `ConfigureServices` добавьте следующий оператор, который настраивает контекст данных со строкой подключения к базе данных для использования с указанным поставщиком базы данных, таким как SQLite, как показано ниже:

     ```
     services.AddDbContext<MyDataContext>(options => options.UseSqlite("my
     database connection string"));
     ```

   - в классе модели Razor Page или в разделе `@functions` объявите скрытое поле для хранения контекста данных, а затем установите его в конструкторе, как показано ниже:

     ```
     private MyDataContext db;

     public SuppliersModel(MyDataContext injectedContext)
     {
       db = injectedContext;
     }
     ```

10. Как повторно использовать Razor Pages в ASP.NET Core 2.2 или более поздней версии?

    Ответ: чтобы повторно использовать Razor Pages в ASP.NET 2.2 или более поздней версии, все, что связано со страницей Razor, можно скомпилировать в библиотеку классов, используя команду `dotnet new razorclasslib -s`.

Глава 16. Разработка сайтов с использованием паттерна MVC

1. Зачем нужны файлы со специальными именами `_ViewStart` и `_ViewImports`, созданные в папке Views?

   Ответ

   - Файл `_ViewStart` содержит блок операторов, которые выполняются при вызове метода `View`, когда метод действия контроллера передает модель в представление, например, для установки макета по умолчанию.
   - Файл `_ViewImports` содержит операторы `@using` для импорта пространств имен для всех представлений, чтобы избежать необходимости добавлять одни и те же операторы импорта в верхней части всех представлений.

2. Что представляют собой имена трех сегментов, определенных по умолчанию в ASP. NET Core MVC, и какие из них необязательны?

   Ответ

   - `{controller}`: например, `/home` представляет класс контроллера для создания экземпляра `HomeController`. Это необязательно, так как он может использовать значение по умолчанию: `Home`.
   - `{action}`: например, `/index` представляет метод действия для выполнения `Index`. Это не обязательно, так как он может использовать значение по умолчанию: `Index`.
   - `{id}`: например, `/5` представляет параметр в методе действия `int id`. Это не обязательно, так как к нему добавляется суффикс `?`.

3. Что такое привязка моделей и какие типы данных она может обрабатывать?

   Ответ: связыватель модели по умолчанию устанавливает параметры в методе действия. Он может обрабатывать следующие типы данных:

   - простые типы, такие как `int`, `string`, `DateTime`, включая типы `nullable`;
   - сложные типы, такие как `Person`, `Product`;
   - типы коллекций, такие как `IEnumerable<T>`.

4. Как выводится содержимое текущего представления в общем файле макета, таком как `_Layout.cshtml`?

Ответ: чтобы вывести содержимое текущего представления в общем макете, вызовите метод `RenderBody`:

```
@RenderBody()
```

5. Как в общем файле макета, таком как `_Layout.cshtml`, задать секцию, для которой содержимое может быть отображено текущим представлением, и как это представление выдает содержимое для этой секции?

   Ответ: чтобы вывести содержимое раздела в общем макете, вызовите метод `RenderSection`, указав при необходимости имя раздела:

   ```
   @RenderSection("Scripts", required: false)
   ```

   Чтобы определить содержимое раздела в представлении, создайте именованный раздел:

   ```
   @section Scripts
   {
     <script> alert('hello');
     </script>
   }
   ```

6. Какие пути ищутся для представления по соглашению при вызове метода `View` внутри метода действия контроллера?

   Ответ: при вызове метода `View` внутри метода действия контроллера выполняется поиск трех путей для представления по умолчанию на основе сочетаний имен контроллера, метода действия и специальной общей папки, как показано в следующем выводе:

   ```
   InvalidOperationException: The view 'Index' was not found. The following
   locations were searched:
   /Views/Home/Index.cshtml
   /Views/Shared/Index.cshtml
   /Pages/Shared/Index.cshtml
   ```

   Это можно обобщить следующим образом:

   - `/Views/[controller]/[action].cshtml`;
   - `/Views/Shared/[action].cshtml`;
   - `/Pages/Shared/[action].cshtml`.

7. Как вы можете указать браузеру пользователя кэшировать ответ в течение 24 часов?

   Ответ: чтобы настроить браузер пользователя для кэширования ответа в течение 24 часов, дополните класс контроллера или метод действия атрибутом `[ResponseCache]` и установите для параметра `Duration` значение `86400` секунд, а для параметра `Location` — `ResponseCacheLocation.Client`.

8. Почему вы можете решить подключить Razor Pages, даже если сами их не создаете?

   Ответ: если вы использовали такие функции, как ASP.NET Core Identity UI, то для этого требуются Razor Pages.

9. Как ASP.NET Core MVC идентифицирует классы, которые могут действовать как контроллеры?

   Ответ: ASP.NET Core MVC идентифицирует классы, которые могут действовать как контроллеры, проверяя, не дополнен ли класс (или класс, из которого он происходит) атрибутом `[Controller]`.

10. Каким образом ASP.NET Core MVC облегчает тестирование сайта?

    Ответ: шаблон проектирования Model-View-Controller (MVC) разделяет технические аспекты формы данных (модели), исполняемых операторов для обработки входящего запроса и исходящего ответа, а затем генерирует ответ в формате, запрошенном пользовательским агентом, например HTML или JSON. Это облегчает написание модульных тестов. ASP.NET Core также упрощает реализацию шаблонов проектирования Inversion-of-Control (IoC) и Dependency Injection (DI) для удаления зависимостей при тестировании компонента, такого как контроллер.

Глава 17. Разработка сайтов с помощью системы управления контентом (CMS)

1. Каковы преимущества использования системы управления контентом для разработки сайта по сравнению с применением только ASP.NET Core?

   Ответ: система управления контентом CMS отделяет контент (значения данных) от шаблонов (макет, формат и стиль). CMS предоставляет пользовательский интерфейс управления контентом, чтобы нетехнические пользователи могли быстро и легко управлять контентом, сохраняя при этом профессионально выглядящий сайт.

   CMS уровня предприятия также будет обеспечивать такие функции, как SEO-URL, точный контроль аутентификации и авторизации, управление медиаресурсами, различные способы повторного использования контента, дизайнер визуальных форм, различные способы персонализации контента и поддержка нескольких версий и языков для контента.

2. Каков специальный относительный URL для доступа к UI управления Piranha CMS и какие имя пользователя и пароль настроены по умолчанию?

   Ответ: специальный относительный URL-путь для доступа к пользовательскому интерфейсу управления Piranha CMS — `/manager`, а имя пользователя и пароль, настроенные по умолчанию, — `admin` и `password`.

3. Что такое слаг?

   Ответ: слаг — это относительный URL-путь для элемента контента, такого как страница или сообщение.

4. В чем разница между сохранением контента и его публикацией?

   Ответ: разница между сохранением и публикацией контента заключается в том, что при сохранении сохраняются изменения в базе данных CMS, а публикация также делает эти изменения доступными для пользователей.

5. Каковы три типа контента Piranha CMS и для чего они используются?

   Ответ: существует три типа контента Piranha CMS: сайты, страницы и сообщения. Сайты предназначены для значений свойств, которые должны быть общими для всех страниц. Страницы предназначены для обычных веб-страниц. Сообщения предназначены для специальных страниц, которые могут отображаться только на странице архива с возможностью сортировки и фильтрации.

6. Каковы три типа компонентов Piranha CMS и для чего они применяются?

   Ответ: существует три типа компонентов Piranha CMS — это поля, регионы и блоки. Поля предназначены для простых значений данных, таких как строки и числа. Регионы предназначены для сложных значений данных, которые отображаются в фиксированном месте с типом контента, таким как страница. Блоки предназначены для сложных значений данных, которые отображаются в любом порядке и комбинации, определенной менеджером контента.

7. Перечислите три свойства, которые тип `Page` наследует от своих базовых классов, и объясните, для чего они используются.

   Ответ: три свойства, которые тип `Page` наследует от своих базовых классов, включают:

   - `ParentId` — значение `Guid`, которое ссылается на свой родительский элемент с «деревом контента»;
   - `Blocks` — упорядоченная коллекция ссылок на экземпляры блоков;
   - `Published` — значение `DateTime`, которое показывает, когда страница была опубликована для просмотра пользователями.

   Ниже приведен полный список свойств, унаследованных от базовых классов:

   - `GenericPage<T>`: `IsStartPage`;
   - `PageBase`: `SiteId`, `ParentId`, `Blocks`, `ContentType`, `SortOrder`, `NavigationTitle`, `IsHidden`, `RedirectUrl`, `RedirectType`, `OriginalPageId`;
   - `RoutedContent`: `Slug`, `Permalink`, `MetaKeywords`, `MetaDescription`, `Route`, `Published`;
   - `Content`: `Id`, `TypeId`, `Title`, `Created`, `LastModified`.

8. Каким образом определить пользовательский тип региона?

   Ответ: вы определяете пользовательский регион для типа страницы, создавая класс со свойствами, дополненными атрибутом `[Field]`, а затем добавляете свойство к типу контента с классом региона в качестве его типа и дополняете его с помощью атрибутов `[Region]` и `[RegionDescription]`:

```
namespace NorthwindCms.Models
{
  [PageType(Title = "Category Page", UseBlocks = false)]
  [PageTypeRoute(Title = "Default", Route = "/category")]
  public class CategoryPage : Page<CategoryPage>
  {
    [Region(Title = "Category detail")]
    [RegionDescription("The details for this category.")]
    public CategoryRegion CategoryDetail { get; set; }
```

9. Как определить маршруты для Piranha CMS?

   Ответ: чтобы определить маршруты для Piranha CMS, в классе `CmsController` добавьте метод действия и дополните его атрибутом `[Route]`:

```
[Route("category")]
public IActionResult Category(Guid id)
```

10. Каким образом получить страницу из базы данных Piranha CMS?

    Ответ: вы извлекаете страницу из базы данных Piranha CMS с помощью зависимостей службы `IApi`:

```
var model = await _api.Pages.GetByIdAsync<Models.CategoryPage>(id);
```

Глава 18. Разработка и использование веб-сервисов

1. Какой базовый класс вы должны унаследовать, чтобы создать класс контроллера для сервиса ASP.NET Core Web API?

   Ответ: чтобы создать класс контроллера для службы Web API ASP.NET Core, вы должны наследовать от класса `ControllerBase`. Не наследуйте от `Controller`, как в MVC, так как этот класс включает такие методы, как `View`, которые используют файлы Razor для визуализации HTML, которые не нужны для веб-службы.

2. Если вы дополнили свой класс `Controller` атрибутом `[ApiController]`, чтобы получить поведение по умолчанию, например автоматический ответ со статусом 400 для недопустимых моделей, то что еще должны сделать?

   Ответ: если вы дополните свой класс контроллера с помощью атрибута `[ApiController]`, то вы также должны вызвать метод `SetCompatibilityVersion` в классе `Startup`.

3. Что вы должны сделать, чтобы указать, какой метод действия контроллера будет выполняться в ответ на HTTP-запрос?

   Ответ: чтобы указать, какой метод действия контроллера будет выполняться в ответ на запрос, вы должны дополнить метод действия атрибутом. Например, чтобы ответить на HTTP-запрос `POST`, дополните метод действия атрибутом `[HttpPost]`.

4. Что вы должны сделать, чтобы указать, какие ответы следует ожидать при вызове метода действия?

   Ответ: чтобы указать, какие ответы следует ожидать при вызове метода действия, дополните метод действия атрибутом `[ProducesResponseType]`, как показано в коде ниже:

```
// GET: api/customers/[id]
[HttpGet("{id}", Name = nameof(Get))] // именованный маршрут
[ProducesResponseType(200, Type = typeof(Customer))]
[ProducesResponseType(404)]
public IActionResult Get(string id)
{
```

5. Перечислите три метода, которые можно вызывать для возврата ответов с разными кодами состояния.

   Ответ: три метода, которые можно вызвать для возврата ответов с разными кодами состояния, включают в себя следующие коды состояния:

   - `Ok` — возвращает код состояния 200 и объект, переданный в тело кода `Ok`;
   - `CreatedAtRoute` — возвращает код состояния 201 и объект, переданный этому методу в теле кода;
   - `NoContentResult` — возвращает код состояния 204 и пустое тело кода;
   - `BadRequest` — возвращает код состояния 400 и необязательное сообщение об ошибке;
   - `NotFound` — возвращает код состояния 404 и необязательное сообщение об ошибке.

6. Перечислите четыре способа тестирования веб-сервиса.

   Ответ: четыре способа тестирования веб-службы включают в себя следующее.

   - Использование браузера для проверки простых HTTP-запросов `GET`.
   - Установку клиентского расширения REST для Visual Studio Code.
   - Установку пакета Swagger NuGet в свой проект веб-службы, включение Swagger и применение пользовательского интерфейса тестирования Swagger.
   - Установку инструмента `Postman` по следующей ссылке: https://www.getpostman.com.

7. Почему не стоит оборачивать `HttpClient` в оператор `using`, чтобы очистить его ресурсы, когда закончите с ним работать, даже если он реализует интерфейс `IDisposable`, и что вы должны задействовать вместо этого?

   Ответ: HttpClient является общим, реентерабельным и частично поточно-ориентированным, поэтому его сложно использовать во многих сценариях. Следует использовать `HttpClientFactory`, представленный в .NET Core 2.1.

8. Что такое CORS и почему важно включить его в веб-сервис?

   Ответ: аббревиатура CORS расшифровывается как Cross-Origin Resource Sharing (совместное использование ресурсов между разными источниками). Это систему важно включить для веб-службы, поскольку для повышения безопасности по умолчанию политика для одного и того же браузера не позволяет коду, загруженному из одного источника, получать доступ к ресурсам, загруженным из другого источника.

9. Как вы можете разрешить клиентам определять, исправен ли ваш веб-сервис, в ASP.NET Core 2.2 и более поздних версиях?

   Ответ: чтобы клиенты могли определить, исправна ли ваша веб-служба, вы можете установить API проверки работоспособности, включая проверки работоспособности базы данных для контекстов данных Entity Framework Core. Чтобы сообщить клиенту подробную информацию, проверка работоспособности может быть расширена.

10. Какие преимущества обеспечивает маршрутизация на основе конечных точек?

    Ответ: маршрутизация в конечной точке обеспечивает улучшенную производительность при маршрутизации и выборе метода действия, а также службу генерации канала.

Глава 19. Разработка интеллектуальных приложений с помощью алгоритмов машинного обучения

1. Каковы четыре основных этапа жизненного цикла машинного обучения?

   Ответ: четыре основных этапа жизненного цикла машинного обучения — анализ проблем, сбор и обработка данных, моделирование и развертывание модели.

2. Каковы три подэтапа шага моделирования?

   Ответ: три подэтапа шага моделирования — идентификация функций, обучение модели и оценка модели.

3. Почему модели необходимо перетренировать после развертывания?

   Ответ: модели должны быть переобучены после развертывания, потому что их прогнозы могут изменяться и становиться хуже, так как данные могут со временем меняться.

4. Почему вы должны разделить свой набор данных на два: для обучения и для тестирования?

   Ответ: вам необходимо разделить свой набор данных на набор данных для обучения и набор данных для тестирования, так как, если вы используете весь набор данных для обучения, у вас не останется данных для тестирования вашей модели и вы не сможете использовать один и тот же набор данных для обучения и тестирования, потому что все будет работать идеально!

5. В чем разница между кластеризацией и классификацией задач машинного обучения?

   Ответ: некоторые из различий между задачами машинного обучения по кластеризации и классификации заключаются в следующем.

   - Кластеризация предназначена для группировки данных, когда вы еще не знаете, есть ли общие черты или какими должны быть метки. Данная техника машинного обучения — неконтролируемая.
   - Классификация предназначена для распределения данных по заранее определенным маркированным группам. Данная техника машинного обучения — контролируемая.

6. Какой класс вы должны создать, чтобы выполнить любую задачу машинного обучения?

   Ответ: для выполнения любой задачи машинного обучения необходимо создать экземпляр `MLContext`.

7. В чем разница между меткой и характеристикой?

   Ответ: характеристика представляет собой ввод, например токенизированный текст обзора Amazon. Метка — это значение, используемое для обучения модели, и может быть выводом, например положительным или отрицательным.

8. Что представляет собой интерфейс `IDataView`?

   Ответ: интерфейс `IDataView` представляет собой входные данные для задачи машинного обучения. Он неизменный, курсивный, лениво оцененный, разнородный и схематизированный.

9. Что представляет собой параметр `count` в атрибуте `[KeyType(count: 10)]`?

   Ответ: параметр `count` представляет собой максимально возможное значение, которое может быть сохранено в столбце.

10. Что означает оценка с матричной факторизацией?

    Ответ: оценка с матричной факторизацией означает вероятность быть положительным результатом, но это не означает вероятность в традиционном смысле. Чем больше значение оценки, тем выше вероятность.

Глава 20. Создание пользовательских веб-интерфейсов с помощью Blazor

1. Какие две основные модели хостинга существуют в Blazor и чем они отличаются?

   Ответ: две основные модели хостинга для Blazor — это Server и WebAssembly.

   Blazor Server выполняет код на стороне сервера. Это означает, что код имеет полный и простой доступ к ресурсам на стороне сервера, таким как базы данных. Это значительно упрощает реализацию функциональности. Обновления пользовательского интерфейса выполняются с использованием SignalR. Это означает, что между браузером и сервером необходимо постоянное соединение, что ограничивает масштабируемость.

   Blazor WebAssembly выполняет код на стороне клиента. Это означает, что код имеет доступ только к ресурсам в браузере, что значительно усложняет реализацию, так как обратный вызов на сервер должен выполняться всякий раз, когда требуются новые данные.

2. Какая дополнительная конфигурация требуется в классе Startup в проекте сайта Blazor Server по сравнению с проектом сайта ASP.NET Core MVC?

   Ответ: в классе `Startup` в методе `ConfigureServices` необходимо вызвать метод `AddServerSideBlazor`, а в методе `Configure` при настройке конечных точек необходимо вызвать `MapBlazorHub` и `MapFallbackToPage`.

3. Одним из преимуществ Blazor является возможность реализации компонентов пользовательского интерфейса с использованием C# и .NET вместо JavaScript. Необходима ли Blazor какая-либо реализация JavaScript?

   Ответ: да. Для компонентов Blazor требуется минимум кода JavaScript. Для Blazor Server этот вопрос можно решить, используя файл `_framework/blazor.server.js`. Для Blazor WebAssembly — файл `_framework/blazor.webassembly.js`. Blazor WebAssembly с PWA также использует файл JavaScript `service-worker.js`.

4. Какова роль файла `App.razor` в проекте Blazor?

   Ответ: файл `App.razor` настраивает маршрутизатор, используемый всеми компонентами Blazor в текущей сборке. Например, он устанавливает общий макет по умолчанию для компонентов, которые соответствуют маршруту, и представление, которое будет использоваться, если совпадения не обнаружены.

5. В чем преимущество использования компонента `<NavLink>`?

   Ответ: преимущество использования компонента `<NavLink>` заключается в том, что он интегрируется с системой маршрутизации Blazor. Затем NavigationManager может использоваться для программного перехода между компонентами.

6. Каким образом осуществляется передача значения компоненту?

   Ответ: вы можете передать значение в компонент, дополнив общедоступное свойство в компоненте атрибутом `[Parameter]`, а затем установив атрибут в компоненте при его использовании:

```
// определение компонента
@code {
  [Parameter]
  public string ButtonText { get; set; };

// использование компонента
<CustomerDetail ButtonText="Create Customer" />
```

7. В чем преимущество использования компонента `<EditForm>`?

   Ответ: преимущество использования компонента `<EditForm>` — сообщения автоматической проверки.

8. Каким образом выполняются некоторые операторы при установленных параметрах?

   Ответ: когда параметры установлены, вы можете выполнить некоторые операторы, определив метод `OnParametersSetAsync` для обработки события.

9. Каким образом выполняются некоторые операторы при появлении компонента?

   Ответ: при обнаружении компонента вы можете выполнить некоторые операторы, определив метод `OnInitializedAsync` для обработки события.

10. Назовите два ключевых различия в классе `Program` между сервером Blazor и проектом Blazor WebAssembly.

    Ответ: два ключевых различия в классе `Program` между Blazor Server и проектом Blazor WebAssembly: 1) использование `WebAssemblyHostBuilder` вместо `Host.CreateDefaultBuilder` и 2) регистрация `HttpClient` с базовым адресом серверной среды.

Глава 21. Разработка кросс-платформенных мобильных приложений

1. В чем разница между Xamarin и Xamarin.Forms?

   Ответ: платформа Xamarin позволяет разработчикам создавать настольные и мобильные приложения с использованием языка C# для Apple iOS (iPhone и iPad), *или* для MacOS, *или* для Google Android. Некоторой бизнес-логикой можно поделиться, но вы в основном будете создавать отдельные проекты для каждой платформы. Xamarin.Forms расширяет Xamarin, чтобы сделать еще более упростить кросс-платформенную разработку мобильных приложений,

разделяя большую часть уровня взаимодействия с пользователем, а также уровня бизнес-логики.

2. Каковы четыре категории компонентов пользовательского интерфейса Xamarin.Forms и что они собой представляют?

   Ответ: ниже перечислены четыре категории компонентов пользовательского интерфейса Xamarin.Forms.

   - *Страницы*: представляют кросс-платформенные экраны мобильных приложений.
   - *Макеты:* представляют структуру комбинации других компонентов пользовательского интерфейса.
   - *Представления*: представляют отдельный компонент пользовательского интерфейса.
   - *Ячейки*: представляют отдельный элемент в виде списка или таблицы.

3. Перечислите четыре типа ячеек.

   Ответ: `TextCell`, `SwitchCell`, `EntryCell` и `ImageCell`.

4. Каким образом вы можете разрешить пользователю выполнять действия с ячейками в `ListView`?

   Ответ: чтобы позволить пользователю выполнить действие с ячейкой в представлении списка, вы можете установить некоторые контекстные действия, представляющие собой пункты меню, которые вызывают событие, как показано в коде ниже:

```
<TextCell Text="{Binding CompanyName}" Detail="{Binding Location}">
  <TextCell.ContextActions>
    <MenuItem Clicked="Customer_Phoned" Text="Phone" />
    <MenuItem Clicked="Customer_Deleted" Text="Delete"
      IsDestructive="True" />
  </TextCell.ContextActions>
</TextCell>
```

5. Как определить сервис зависимостей в целях реализации функциональности, специфичной для платформы?

   Ответ: чтобы определить зависимости службы для реализации функциональности, специфичной для платформы, необходимо выполнить следующее.

   - Определите интерфейс для зависимостей службы.
   - Реализуйте зависимости службы для каждой платформы, например iOS и Android.
   - Дополните зависимости службы с помощью атрибута `[assembly: Dependency]` и передайте тип зависимостей службы в качестве параметра.

- В общем главном проекте получите зависимости службы:

```
var service = DependencyService.Get<IMyService>();
```

6. В каком случае элемент `Entry` используется вместо элемента `Editor`?

   Ответ: используйте `Entry` для одной строки текста и `Editor` — для нескольких строк текста.

7. В чем заключается эффект установки `IsDestructive` в значение `true` для пункта меню в контекстных действиях ячейки?

   Ответ: пункт меню окрашен в красный цвет в качестве предупреждения для пользователя.

8. Когда выполняется вызов методов `PushAsync` и `PopAsync` в мобильных приложениях Xamarin.Forms?

   Ответ: чтобы обеспечить переход между экранами со встроенной поддержкой для возврата к предыдущему экрану, при первом запуске приложения оберните первый экран оператором `NavigationPage`:

```
MainPage = new NavigationPage(new CustomersList());
```

   Чтобы перейти к следующему экрану, нажмите следующую страницу на объекте навигации:

```
await Navigation.PushAsync(new CustomerDetails(c));
```

   Чтобы вернуться к предыдущему экрану, откройте страницу из объекта `Navigation`:

```
await Navigation.PopAsync();
```

9. Как показать всплывающее модальное сообщение с простым выбором кнопок, таких как `Yes` (Да) или `No` (Нет)?

   Ответ: чтобы отобразить всплывающее модальное сообщение с простым выбором кнопок, например `Yes` (Да) или `No` (Нет), вы можете вызвать метод `DisplayAlert`:

```
bool response = await DisplayAlert("Apple Survey",
  "Do you like your iPhone 11 Pro?", "Yes", "No");
```

10. В чем заключается функция ATS от Apple и почему она важна?

    Ответ: функция ATS от Apple — это сокращение App Transport Security, и она важна, так как включена по умолчанию и призывает разработчиков использовать передовой опыт, включая безопасные соединения между приложением и веб-службой.

Приложение Б. Разработка настольных Windows-приложений

1. Платформа .NET 5 — кросс-платформенная. Приложения Windows Forms и WPF могут работать на .NET 5. Могут ли эти приложения работать для операционных систем macOS и Linux?

   Ответ: нет. Хотя приложения Windows Forms- и WPF-приложения могут работать на платформе .NET 5, им также необходимо выполнять вызовы Win32 API, поэтому они ограничены работой на основе операционной системы Windows.

2. Как приложение Windows Forms определяет свой пользовательский интерфейс и почему это сложная задача?

   Ответ: с помощью кода языка C#. Это сложно, поскольку приложение Windows Forms, предназначенное для .NET Framework, получает визуальный конструктор с набором инструментов и поддержкой перетаскивания в Visual Studio 2019. Однако приложение, ориентированное на .NET 5, в настоящее время не имеет визуальной поддержки.

3. Как WPF- или UWP-приложение может определять свой пользовательский интерфейс и почему это хорошо для разработчиков?

   Ответ: с помощью языка разметки XAML. Его понять легче, чем код на языке C#.

4. Назовите пять контейнеров макета и варианты расположения их дочерних элементов.

   Ответ

   - `StackPanel` — дочерние элементы размещаются путем их наложения по горизонтали или вертикали.
   - `Grid` — дочерние элементы располагаются по строкам и столбцам в пределах определенной сетки.
   - `DockPanel` — одиночный дочерний элемент размещается в его родительском контейнере путем выравнивания его с помощью значений `Top`, `Left`, `Bottom`, `Right` или заполнения оставшегося пространства.
   - `WrapPanel` — дочерние элементы размещаются путем их наложения горизонтально или вертикально, а затем при необходимости переносятся в другой столбец или строку.
   - `Canvas` — дочерние элементы позиционируются с помощью абсолютных значений `Top` и `Left` или `Bottom` и `Right`.

5. В чем разница между `Margin` и `Padding` для такого элемента, как `Button`?

   Ответ: `Margin` находится за пределами границы, а `Padding` — внутри границы.

6. Как обработчики событий присоединяются к объекту с помощью XAML?

   Ответ: устанавливая атрибут для имени события равным имени метода в классе с фоновым кодом:

   ```
   <Button Clicked="SaveButton_Clicked">
   ```

7. Как работают стили XAML?

   Ответ: позволяют устанавливать одно или несколько свойств.

8. Где вы можете определить ресурсы?

   Ответ: в любом элементе в зависимости от того, где хотите поделиться этими ресурсами. Для совместного использования ресурсов в приложении определите ресурсы в элементе `<Application.Resources>`. Чтобы поделиться ресурсами только внутри страницы, определите ресурсы в ее элементе `<Page.Resources>`. Что касается одного элемента, такого как `Button`, определите ресурсы в его элементе `<Button.Resources>`.

9. Каким образом вы можете связать свойство одного элемента со свойством другого?

   Ответ: с помощью выражения привязки для свойства, которое ссылается на другой элемент и его свойство:

   ```
   <Slider Value="18" Minimum="18" Maximum="65" Name="slider" />
   <TextBlock Text="{Binding ElementName=slider, Path=Value}" />
   ```

10. Почему вам может потребоваться реализовать интерфейс `IValueConverter`?

    Ответ: это возможно при необходимости выполнить привязку данных между двумя несовместимыми типами, например для преобразования значения `int` в значение `string`, которое определяет путь к изображению, или между значением с плавающей запятой, представляющим индекс массы тела (ИМТ) и цвет, который указывает на хорошее или плохое здоровье.

Б Разработка настольных Windows-приложений

Данное приложение посвящено созданию настольных Windows-приложений с помощью трех технологий: *Windows Forms*, *Windows Presentation Foundation* (WPF) и *универсальной платформы Windows* (Universal Windows Platform, UWP).

Windows Forms и WPF поддерживают использование .NET Core 3.0 и более поздней версии в качестве среды выполнения. Однако текущая поддержка времени разработки ограничена, и потому я рекомендую эти технологии, только если у вас имеются приложения Windows Forms или WPF, которые необходимо перенести на современную платформу .NET.

Значительная часть этого приложения будет посвящена приложениям UWP, которые используют современную среду выполнения Windows Runtime и могут выполнять приложения, созданные с помощью адаптированной версии .NET, которая компилируется в инструкции для ЦП.

Хотя в большей части данного приложения технически не применяется .NET 5, в ноябре 2021 года корпорация Microsoft выпустит версию .NET 6, которая станет единой унифицированной платформой для .NET, используемой всеми моделями приложений, включая ASP.NET Core для веб-разработки, Windows Forms, WPF, а также кросс-платформенные настольные и мобильные приложения с использованием пользовательского интерфейса кросс-платформенных приложений .NET (MAUI).

Более подробную информацию о выборе платформы для создания настольных Windows-приложений можно найти на сайте https://docs.microsoft.com/ru-ru/windows/apps/desktop/choose-your-platform.

Вы познакомитесь с некоторыми новыми функциями пользовательского интерфейса Fluent Design, представленными в обновлении Windows 10 Fall Creators, выпущенном в октябре 2017 года.

В одном приложении я смогу лишь немного рассказать о том, что достижимо с помощью .NET для разработки настольных Windows-приложений. Однако я надеюсь, что у вас появится желание узнать больше о новой возможности миграции

устаревших Windows-приложений на современную платформу .NET и о новой современной технологии UWP с интерфейсом Fluent Design, в том числе о шаблонизируемых элементах управления, привязке данных и анимации!

Несколько важных слов об этом приложении

UWP-приложения хотя и не кросс-платформенные, но задействуют принцип «кросс-девайсной» разработки, если устройства, на которых они запускаются, работают под управлением современных версий операционной системы Windows, то есть они могут работать на настольных ПК, ноутбуках, планшетах и устройствах виртуальной реальности наподобие гарнитуры HP Windows Mixed Reality. Чтобы использовать новейшие функции, такие как XAML Islands, вам понадобится операционная система Windows 10 с обновлением от мая 2019 года и среда разработки Visual Studio 2019 версии 16.3 или новее. Хотя для разработки приложения из этого приложения вам достаточно более старой версии Windows 10, выпущенной в апреле 2018 года.

В этом приложении:

- общие сведения об устаревших платформах Windows-приложений;
- общие сведения о современной платформе Windows;
- разработка современного Windows-приложения;
- использование ресурсов и шаблонов;
- привязка данных.

Общие сведения об устаревших платформах Windows-приложений

С момента выпуска Microsoft Windows 1.0 в 1985 году Windows-приложения можно было разрабатывать только на языке C, вызывая функции одной из трех основных DLL-библиотек: kernel, user и GDI. Как только появилась 32-битная Windows 95, к названиям этих библиотек добавился суффикс 32 и они стали известны как Win32 API.

В 1991 году корпорация Microsoft представила среду разработки Visual Basic, которая предоставила разработчикам возможность визуального перетаскивания элементов управления из набора инструментов в целях создания пользовательского интерфейса для Windows-приложений. Эта возможность стала очень популярной, и среда выполнения Visual Basic по-прежнему входит в состав операционной системы Windows 10.

В 2002 году корпорация Microsoft представила платформу .NET Framework, включающую Windows Forms для разработки Windows-приложений. Программный код

стало можно писать на языках Visual Basic или C#. Технология Windows Forms содержала похожий инструмент визуального перетаскивания, хотя для определения пользовательского интерфейса генерировала программный код на языках C# или Visual Basic, что может усложнить понимание и редактирование напрямую.

В 2006 году корпорация Microsoft представила платформу .NET Framework 3.0, включающую технологию WPF для создания пользовательских интерфейсов с помощью XAML. Эта технология проста для понимания и редактирования.

Поддержка .NET 5 для устаревших Windows-платформ

Пакет SDK, входящий в состав .NET 5, для операционных систем Linux и macOS занимает на диске 332 Мбайт и 337 Мбайт соответственно. Пакет SDK, входящий в состав .NET 5 для операционной системы Windows, занимает на диске 441 Мбайт. Это связано с тем, что в состав входит среда выполнения Windows Desktop, позволяющая запускать устаревшие платформы Windows-приложений Windows Forms и WPF в .NET 5.

Установка Microsoft Visual Studio 2019 для операционной системы Windows

С октября 2014 года корпорация Microsoft разработала версию Visual Studio профессионального уровня, доступную для всех бесплатно, — Community Edition.

Установим ее прямо сейчас.

1. Скачайте среду разработки Microsoft Visual Studio 2019 версии 16.8 или более поздней для операционной системы Windows по следующей ссылке: https://visualstudio.microsoft.com/downloads/.
2. Запустите установщик.
3. На вкладке Workloads (Рабочие нагрузки) выберите следующее:
   - Разработка настольных приложений .NET;
   - Разработка универсальной платформы Windows;
   - Кросс-платформенная разработка на основе .NET Core.
4. Нажмите кнопку Install (Установить) и дождитесь, пока установщик загрузит выбранное программное обеспечение и установит его.
5. После завершения установки нажмите кнопку Launch (Запустить).
6. При первом запуске Visual Studio вам будет предложено войти в систему. Если у вас уже есть учетная запись Microsoft, то можете использовать ее. В противном

случае для регистрации учетной записи вам необходимо пройти по следующей ссылке: signup.live.com.

7. При первом запуске Visual Studio вам будет предложено настроить вашу среду разработки. Для Development Settings (Параметры разработки) выберите язык Visual C#. В качестве цветовой темы я выбрал Blue (Синий), но вы можете выбрать любую другую.

Работа с Windows Forms

Разработку нового приложения Windows Forms начнем с использования инструмента командной строки `dotnet`. Затем задействуем среду разработки Visual Studio, чтобы создать приложение Windows Forms для .NET Framework в целях имитации устаревшего приложения, а затем перенесем это приложение в .NET 5.

Разработка нового приложения Windows Forms

На следующем примере вы убедитесь, что не стоит создавать приложения Windows Forms для .NET 5. Как правило, в .NET следует переносить только существующие приложения Windows Forms.

Следующие инструкции начинаются с создания папок для хранения файла решения и проекта. Вы можете задействовать другой инструмент для этого вместо командной строки, но вам необходимо по крайней мере создать новое решение и проект Windows Forms, используя инструмент командной строки `dotnet` в правильной папке.

1. В операционной системе Windows в меню Start (Пуск) откройте Command Prompt (Командная строка).
2. В командной строке введите команды для выполнения следующих задач:
   - переход в папку Code;
   - создание папки WindowsDesktopApps с новым файлом решения;
   - создание подпапки BasicWinForms с новым проектом Windows Forms.

```
cd C:\Code\
mkdir WindowsDesktopApps
cd WindowsDesktopApps
dotnet new sln
mkdir BasicWinForms
cd BasicWinForms
dotnet new winforms
```

3. Запустите программу Visual Studio 2019 и нажмите кнопку Open a project or solution (Открыть проект или решение).

4. Выберите каталог C:\Code\WindowsDesktopApps. Затем выберите файл решения `WindowsDesktopApps.sln` и нажмите кнопку Open (Открыть).
5. Выберите File ▸ Add ▸ Existing Project (Файл ▸ Добавить ▸ Существующий проект). Выберите проект `BasicWinForms.csproj` и нажмите кнопку Open (Открыть).
6. Выберите Project ▸ Edit Project File (Проект ▸ Свойства файла проекта) или щелкните правой кнопкой мыши на панели Solution Explorer (Обозреватель решений) и выберите пункт меню Edit Project File (Свойства файла проекта). Затем обратите внимание, что Target framework (Целевая платформа) — net5.0-windows, используется технология Windows Forms, а Output type (Тип вывода) — Windows-приложение:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net5.0-windows</TargetFramework>
    <UseWindowsForms>true</UseWindowsForms>
  </PropertyGroup>

</Project>
```

7. Закройте вкладку.
8. Дважды щелкните на панели Solution Explorer (Обозреватель решений) или щелкните правой кнопкой мыши на файле `Form1.cs` и выберите кнопку Open (Открыть). Обратите внимание: на момент написания книги в сентябре 2020 года конструктор Windows Forms по умолчанию не поддерживает .NET 5 (хотя версия 16.8, выпущенная в ноябре 2020 года, должна работать), как показано на рис. Б.1.

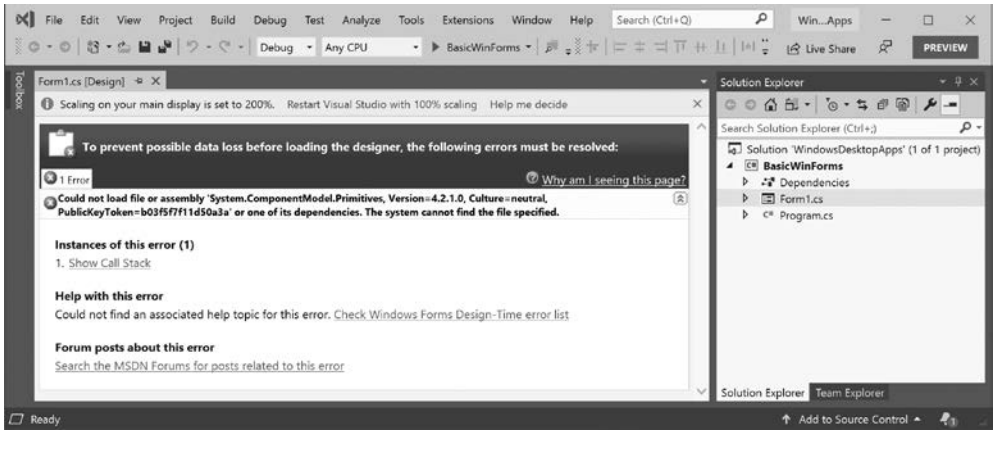

Рис. Б.1

Отслеживать прогресс конструктора Windows Forms можно на сайте https://github.com/dotnet/winforms/tree/master/docs/designer-releases.

9. Выберите команду меню Debug ▸ Start Without Debugging (Отладка ▸ Запуск без отладки) или нажмите сочетание клавиш Ctrl+F5. Вы увидите окно с изменяемым размером с заголовком Form1, а затем для выхода из приложения нажмите кнопку закрытия в правом верхнем углу.

Обзор нового приложения Windows Forms

Рассмотрим код в пустом приложении Windows Forms.

1. На панели Solution Explorer (Обозреватель решений) найдите и откройте файл `Program.cs`. Обратите внимание: он похож на консольное приложение с методом точки входа `Main`, создает экземпляр класса `Form1` и запускает его, как показано ниже:

```
static class Program
{
  [STAThread]
  static void Main()
  {
    Application.SetHighDpiMode(HighDpiMode.SystemAware);
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1());
  }
}
```

2. На панели Solution Explorer (Обозреватель решений) разверните файл `Form1.cs`, откройте класс `Form1` и обратите внимание, что он является частичным и что в его конструкторе вызывается метод `InitializeComponent`:

```
public partial class Form1 : Form
{
  public Form1()
  {
    InitializeComponent();
  }
}
```

3. На панели Solution Explorer (Обозреватель решений) разверните файл `Form1.cs`, найдите и откройте файл `Form1.Designer.cs`, разверните раздел Windows Form Designer generated code (Код, сгенерированный конструктором Windows Form Designer) и обратите внимание, что код, описывающий пустую форму, слишком путанный, чтобы понять и вручную менять его:

```
private void InitializeComponent()
{
  this.components = new System.ComponentModel.Container();
```

```
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(800, 450);
    this.Text = "Form1";
}
```

4. Закройте все окна редактирования.

Миграция устаревшего приложения Windows Forms

В данном примере вы добавите приложение Windows Forms для .NET Framework, чтобы можно было использовать визуальный конструктор Windows Forms, а затем перенесете его в .NET 5.

1. В программе Visual Studio 2019 выберите File ▸ Add ▸ New Project (Файл ▸ Добавить ▸ Создать проект).
2. В строке поиска введите `windows forms` и выберите пункт Windows Forms App (.NET Framework). Нажмите кнопку Next (Далее).
3. В качестве Project name (Имя проекта) введите `LegacyWinForms` и нажмите кнопку Create (Создать).
4. Выберите View ▸ Toolbox (Вид ▸ Панель инструментов) или нажмите сочетание клавиш Ctrl+W, X, а затем закрепите панель инструментов. Обратите внимание: у вас могут быть разные привязки клавиш. Если это так, то вы можете выбрать Tools ▸ Import and Export Settings (Инструменты ▸ Импорт и экспорт настроек), затем сбросьте настройки, чтобы установить язык Visual C# и использовать те же привязки клавиш, которые применяются в данном приложении.
5. На вкладке Toolbox (Панель инструментов) разверните пункт меню Common Controls (Общие элементы управления).
6. Перетащите некоторые элементы управления, такие как Button, MonthCalendar, Label и TextBox на Form1 (рис. Б.2).

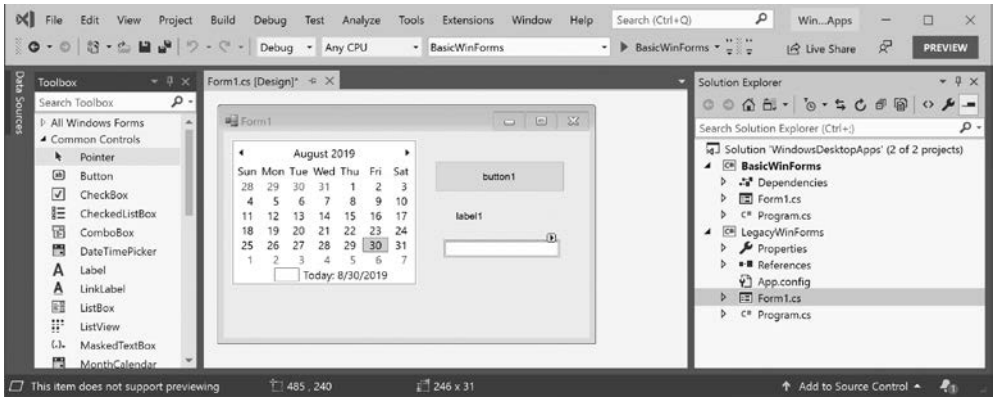

Рис. Б.2

7. Выберите команду меню View ▸ Properties Window (Вид ▸ Окно свойств) либо нажмите сочетание клавиш Ctrl+W, P или клавишу F4.
8. Выберите кнопку (элемент управления button), измените ее свойство (Name) на `btnGoToChristmas` и свойство Text на `Go to Christmas`.
9. Дважды нажмите кнопку. Появится обработчик событий. Далее введите операторы, чтобы установить календарь и выбрать в нем день Рождества 2020 года, как показано ниже:

```
private void btnGoToChristmas_Click(object sender, EventArgs e)
{
  DateTime christmas = new DateTime(2020, 12, 25);
  monthCalendar1.SelectionStart = christmas;
  monthCalendar1.SelectionEnd = christmas;
}
```

10. На панели Solution Explorer (Обозреватель решений) щелкните правой кнопкой мыши на решении и выберите Set StartUp Projects (Настроить проекты для запуска).
11. В диалоговом окне Solution 'WindowsDesktopApps' Properties Pages (Страницы свойств решения WindowsDesktopApps) выберите Current selection (Текущий выбор) для Startup Project (Проект для запуска) и нажмите кнопку OK.
12. На панели Solution Explorer (Обозреватель решений) выберите проект LegacyWinForms и обратите внимание: его имя выделено полужирным шрифтом — таким образом показывается, что это текущий выбор.
13. Выберите команду меню Debug ▸ Start Without Debugging (Отладка ▸ Запуск без отладки) или нажмите сочетание клавиш Ctrl+F5. Нажмите кнопку и обратите внимание: календарь анимируется, чтобы выбрать дату Рождества. Затем для выхода из приложения нажмите кнопку закрытия в правом верхнем углу.

Миграция приложения Windows Forms

Теперь мы можем выполнить перенос в проект .NET 5. Однако помните, что это временно. Как только конструктор Windows Forms (ныне тестируется) будет перенесен на .NET 5 и станет доступен в будущем выпуске Visual Studio 2019, миграция не потребуется.

1. Перетащите `Form1` из папки LegacyWinForms в папку BasicWinForms, которая предложит вам перезаписать следующие файлы:
   - `Form1.cs`;
   - `Form1.Designer.cs`;
   - `Form1.resx`.

2. В проекте `BasicWinForms` отредактируйте файл `Program.cs`, чтобы указать устаревшее пространство имен при создании экземпляра и запуске `Form1`, как показано ниже:

```
Application.Run(new LegacyWinForms.Form1());
```

3. На панели Solution Explorer (Обозреватель решений) выберите проект BasicWinForms.
4. Выберите Debug ▸ Start Without Debugging (Отладка ▸ Запуск без отладки) или нажмите сочетание клавиш Ctrl+F5. Нажмите кнопку и обратите внимание: календарь анимируется, чтобы выбрать день Рождества. Затем закройте приложение.
5. Закройте проект и решение.

Миграция приложений WPF в .NET 5

Если вам необходимо перенести существующие приложения WPF из .NET Framework в .NET 5, то, как и при миграции проектов Windows Forms, это возможно, но трудновыполнимо. Обновления будут доступны в течение следующего года, в частности в грядущих версиях Visual Studio 2019.

Более подробную информацию о переносе приложений WPF можно получить на сайте https://devblogs.microsoft.com/dotnet/migrating-asample-wpf-app-to-net-core-3-part-1/.

Миграция устаревших приложений с помощью Windows Compatibility Pack (пакет обеспечения совместимости Windows)

Пакет обеспечения совместимости Windows предоставляет доступ к API, которые ранее были доступны только для .NET Framework.

Более подробную информацию о пакете обеспечения совместимости Windows можно найти на сайте https://devblogs.microsoft.com/dotnet/announcing-the-windows-compatibility-pack-for-net-core/.

Общие сведения о современной платформе Windows

Корпорация Microsoft продолжает совершенствовать платформу Windows, включающую в себя такие технологии для создания современных приложений, как Universal Windows Platform (UWP) и Fluent Design System.

Общие сведения об универсальной платформе Windows

Универсальная платформа Windows (UWP) — новейшее решение Microsoft для разработки приложений, предназначенных для выполнения в операционных системах семейства Windows. Универсальная платформа Windows обеспечивает гарантированный уровень API для различных типов устройств. Вы создаете один пакет приложения, который загружается в единое хранилище и распространяется на все типы устройств, поддерживаемые приложением. К подобным устройствам относятся настольные компьютеры, ноутбуки и планшеты под управлением операционных систем Windows 10, Xbox One и более поздних версий, а также Mixed Reality Headsets (гарнитуры смешанной реальности), например Microsoft HoloLens.

Универсальная платформа Windows содержит механизмы, позволяющие определить возможности используемого устройства, а затем задействовать дополнительные функции вашего приложения, чтобы применять его максимально эффективно.

Универсальная платформа Windows предоставляет посредством XAML панели макетов, которые адаптируют способ отображения дочерних элементов управления, чтобы максимально использовать устройство, на котором они в данный момент работают. Это эквивалент адаптивного дизайна веб-страниц для приложений Windows. Кроме того, они содержат триггеры визуального состояния для изменения макета на основе динамических изменений, таких как горизонтальная или вертикальная ориентация планшета.

Fluent Design System

Новая система дизайна от корпорации Microsoft под названием Fluent Design System будет поставляться в несколько этапов, чтобы помочь разработчикам постепенно перейти от традиционных стилей пользовательского интерфейса к более современным.

Этап 1, доступный в обновлении Windows 10 Fall Creators, выпущенном в октябре 2017 года, и улучшенный при выпусках последующих этапов, входивших в выпускающиеся два раза в год обновления Windows 10, включал следующие функции:

- акриловый материал;
- связанные анимации;
- параллакс-эффекты (ParallaxView);
- подсвечивание.

Заполнение элементов пользовательского интерфейса акриловыми кистями

Акриловый материал (акрил) — полупрозрачная кисть с эффектом размытия, с помощью которой можно заполнять элементы пользовательского интерфейса, чтобы добавить глубину и перспективу в ваши приложения. Акрил может показать, что находится на заднем плане позади приложения, или элементы в приложении, находящиеся за панелью. Для акрила можно настроить различные цветов и уровень прозрачности.

Более подробно об использовании акрила можно прочитать на сайте https://docs.microsoft.com/ru-ru/windows/uwp/design/style/acrylic.

Связывание элементов пользовательского интерфейса с помощью анимации

При навигации по UI анимация элементов, помогающая проследить связь между экранами, помогает пользователям понять, где они находятся и как взаимодействовать с вашим приложением.

Более подробную информацию о применении связанной анимации можно найти на сайте https://docs.microsoft.com/ru-ru/windows/uwp/design/motion/connected-animation[1].

Параллакс-эффекты и подсвечивание

Параллакс-эффекты (эффекты Parallax) — это фоны, часто представленные изображениями, которые во время прокрутки перемещаются медленнее, чем передний план, чтобы создать ощущение глубины и придать вашим приложениям современный вид.

Более подробную информацию об эффектах Parallax можно получить на сайте https://docs.microsoft.com/ru-ru/windows/uwp/design/motion/parallax.

Подсвечивание помогает пользователю понять, какой из визуальных элементов в UI интерактивный. Элемент *подсвечивается*, когда пользователь наводит на него указатель мыши.

---

[1] На момент перевода книги в статье по ссылке термин Connected animation переведен как «подключенная анимация», однако этот перевод, возможно, сделан машинным переводчиком, что прямо указано вверху страницы. «Связанная аминация» лучше описывает суть явления.

Более подробную информацию о подсвечивании можно найти на сайте https://docs.microsoft.com/ru-ru/windows/uwp/design/style/reveal.

Разработка современного Windows-приложения

В первую очередь начнем с разработки простого приложения UWP с некоторыми общими элементами управления и современными функциями Fluent Design, такими как акриловый материал.

Включение режима разработчика

Для разработки UWP-приложения необходимо в операционной системе Windows 10 включить режим разработчика.

1. Выберите Start ▸ Settings ▸ Update & Security ▸ For developers (Пуск ▸ Параметры ▸ Обновление и безопасность ▸ Для разработчиков). Затем выберите пункт Developer mode (Режим разработчика).
2. Примите предупреждение о том, что даная операция could expose your device and personal data to security risk or harm your device (может подвергнуть ваше устройство и личные данные угрозе безопасности или повредить ваше устройство). Затем закройте приложение Settings (Параметры).

Разработка UWP-приложений

Теперь добавьте новый проект UWP-приложения в свое решение.

1. В программе Visual Studio 2019 откройте решение `WindowsDesktopApps`.
2. Выберите File ▸ Add ▸ New Project (Файл ▸ Добавить ▸ Создать проект).
3. В диалоговом окне Add a new project (Добавить новый проект) в строке поиска введите `uwp`, выберите шаблон Blank App (Universal Windows) (Пустое приложение (Universal Windows)) для языка C# и нажмите кнопку Next (Далее). Убедитесь, что выбрали шаблон для языка C#, а не Visual Basic!
4. В качестве имени проекта введите `FluentUwpApp` и нажмите кнопку Create (Создать).
5. В диалоговом окне New Universal Windows Project (Новый универсальный проект для Windows), в раскрывающемся списке выберите последнюю версию Windows 10 как Target Version (Целевую версию) и Minimum Version (Минимальную версию) и нажмите кнопку OK.

Поскольку Fluent Design System была впервые выпущена вместе с обновлением Window 10 Fall Creators (10.0; сборка 16299), вы сможете выбрать ее для поддержки функций, которые мы рассмотрим в этом приложении. Но на данный момент во избежание неожиданных несовместимостей лучше всего использовать последнюю версию. Разработчики, создающие UWP-приложения для широкой аудитории, должны выбирать последнюю версию Windows 10 в раскрывающемся списке Minimum Version (Минимальная версия). Разработчики, создающие корпоративные приложения, должны выбрать более старую минимальную версию. Сборка 10240 была выпущена в июле 2015 года и отлично подходит для максимальной совместимости. Однако у вас не будет доступа к современным функциям, таким как Fluent Design System.

6. На панели Solution Explorer (Обозреватель решений) дважды щелкните кнопкой мыши на файле `MainPage.xaml`, чтобы открыть его для редактирования. Вы увидите панель дизайна XAML, отображающую графический вид приложения, и панель XAML (рис. Б.3). Вы можете управлять этими панелями следующим образом:
   - панели XAML расположены по горизонтали, но вы можете переключиться на режим отображения по вертикали или свернуть одну из панелей, нажав соответствующую кнопку в правой части интерфейса между панелями;
   - вы можете поменять панели местами, нажав кнопку ↑↓ между панелями;
   - вы можете прокручивать и изменять масштаб обеих панелей.

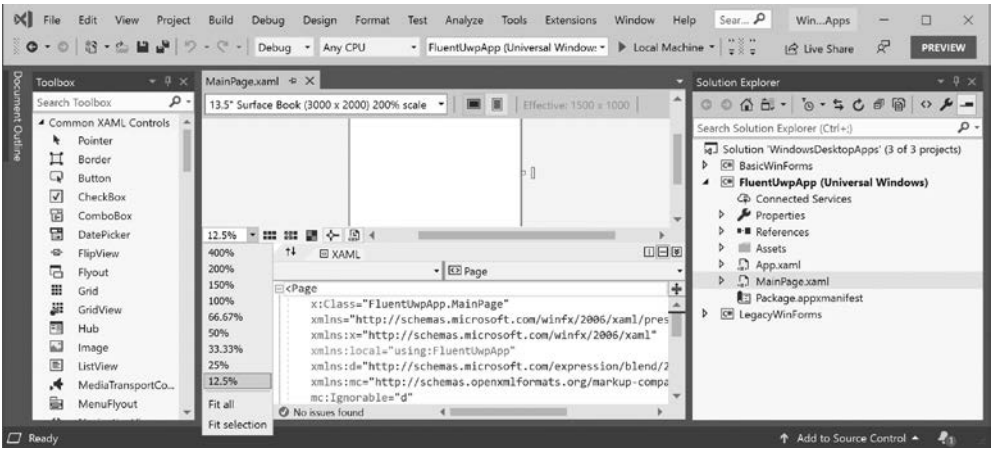

Рис. Б.3

7. На панели Design (Конструктор) измените устройство на 13,3" Desktop (1280 x 720) 100% scale и увеличьте до 100 %.

8. Выберите команду меню View ▸ Toolbox (Вид ▸ Панель инструментов) или нажмите сочетание клавиш Ctrl+W, X. Обратите внимание: панель содержит категории Common XAML Controls (Типовые элементы управления XAML), All XAML Controls (Все элементы управления XAML) и General (Общие).
9. В верхней части панели находится поле поиска. Введите в него буквы bu и обратите внимание, как фильтруется список элементов управления.
10. Перетащите элемент управления Button из панели Toolbox (Панель инструментов) в окно панели Design (Конструктор).
11. Измените размер элемента управления Button. Для этого необходимо нажать и удерживать кнопку мыши на любом из восьми прямоугольных маркеров, затем перетащить с помощью мыши.

    Обратите внимание: для кнопки зафиксированы ширина и высота, а также поля сверху (40 единиц) и слева (30 единиц) по отношению к положению и размерам по сетке (рис. Б.4).

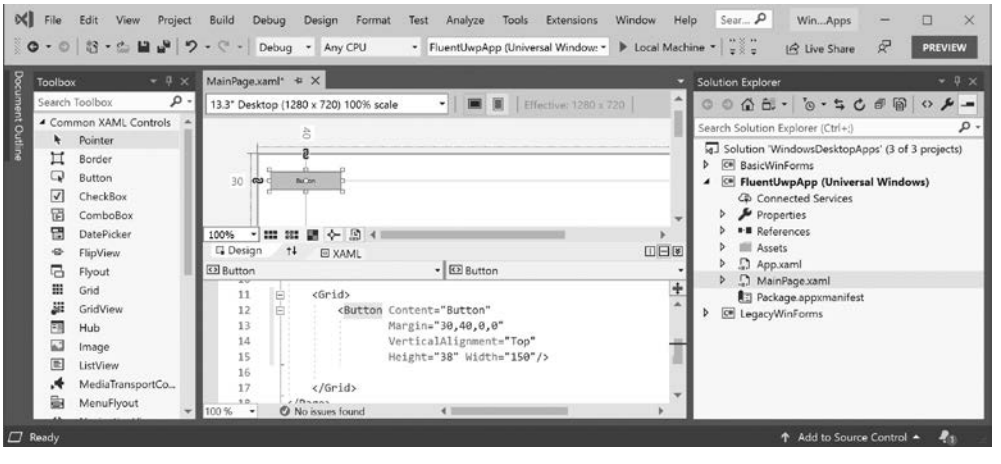

Рис. Б.4

Хотя вы можете перетаскивать элементы управления вручную, для компоновки макета лучше использовать панель XAML, где вы сможете расположить элементы относительно друг друга и реализовать более адаптивный дизайн.

12. На панели XAML найдите элемент `Button` и удалите его.
13. На панели XAML в элементе `Grid` добавьте следующую разметку:

```
<Button Margin="6" Padding="6" Name="ClickMeButton">
  Click Me
</Button>
```

14. Установите масштаб равным 50 % и обратите внимание, что кнопка Button автоматически меняет размер согласно своему содержимому, то есть тексту Click Me,

выравнивается по вертикали по центру и по горизонтали влево, даже если вы переключаетесь между вертикальной и горизонтальной раскладками телефона.

15. На панели XAML удалите элемент `Grid` и измените код, чтобы обернуть элемент `Button` элементом `StackPanel` с ориентацией по горизонтали, который находится в элементе `StackPanel` с ориентацией по вертикали (установленной по умолчанию). Обратите внимание на изменение компоновки макета, то есть на расположение в левом верхнем углу доступного пространства:

```
<StackPanel>
  <StackPanel Orientation="Horizontal" Padding="4"
              Background="LightGray" Name="toolbar">
    <Button Margin="6" Padding="6" Name="ClickMeButton">
      Click Me
    </Button>
  </StackPanel>
</StackPanel>
```

16. Измените код обработчика элемента `Button`, чтобы задать новый обработчик для события `Click` (рис. Б.5).

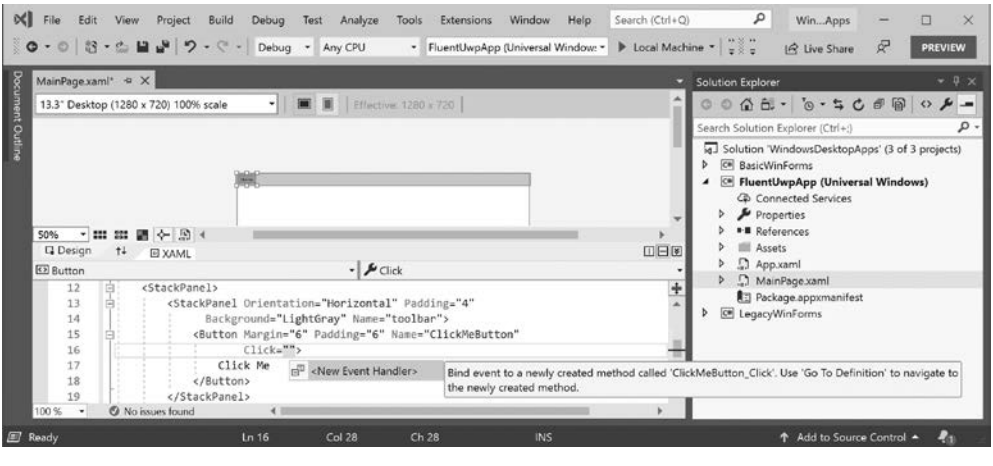

Рис. Б.5

17. Щелкните правой кнопкой мыши на имени обработчика событий и выберите пункт Go to Definition (Перейти к определению) в контекстном меню или нажмите клавишу F12.

18. Добавьте к обработчику событий оператор, устанавливающий содержимое элемента `Button` равным текущему времени:

```
private void ClickMeButton_Click(object sender, RoutedEventArgs e)
{
  ClickMeButton.Content = DateTime.Now.ToString("hh:mm:ss");
}
```

19. Выберите Build ▸ Configuration Manager (Сборка ▸ Диспетчер конфигурации) для проекта FluentUwpApp. Отметьте пункты Build (Сборка) и Deploy (Развертывание), выберите Platform of x64, а затем нажмите кнопку Close (Закрыть), как показано на рис. Б.6.

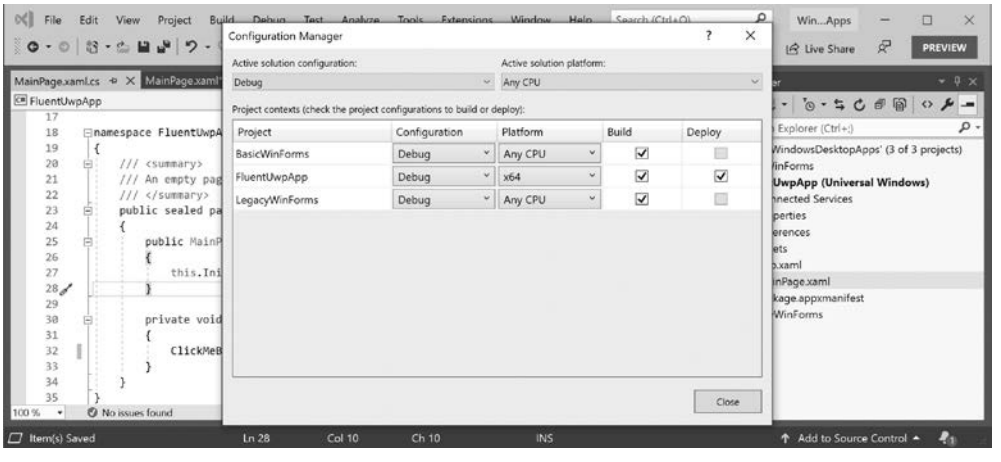

Рис. Б.6

20. Запустите приложение, выбрав Debug ▸ Start Without Debugging (Отладка ▸ Запуск без отладки) или нажав сочетание клавиш Ctrl+F5.

21. Нажмите кнопку Click Me. Обратите внимание: каждый раз, когда вы ее нажимаете, текст меняется на текущее время.

Типовые элементы управления и акриловая кисть

Теперь изучим некоторые типовые элементы управления и рисования акриловыми кистями.

1. Найдите и откройте файл `MainPage.xaml`. Установите фон панели `StackPanel` для применения системной акриловой кисти для окон и добавьте на панель после кнопки элементы для ввода пользователем имени, как показано ниже (выделено полужирным шрифтом):

```
<StackPanel
  Background="{ThemeResource SystemControlAcrylicWindowBrush}">
  <StackPanel Orientation="Horizontal" Padding="4"
              Background="LightGray" Name="toolbar">
    <Button Margin="6" Padding="6" Name="ClickMeButton"
            Click="ClickMeButton_Click">
```

```
        Click Me
      </Button>
      <TextBlock Text="First name:"
        VerticalAlignment="Center" Margin="4" />
      <TextBox PlaceholderText="Enter your name"
        VerticalAlignment="Center" Width="200" />
    </StackPanel>
  </StackPanel>
```

2. Запустите приложение, выбрав Debug ▸ Start Without Debugging (Отладка ▸ Запуск без отладки). Обратите внимание на тонированный акриловый материал, показывающий зеленую палатку и оранжевый закат над горами, изображенные на стандартных обоях Windows 10, сквозь фон окна приложения, как показано на рис. Б.7.

Рис. Б.7

Акрил использует большое количество системных ресурсов, поэтому отключается автоматически, если приложение теряет фокус или ваше устройство разряжено.

Подсвечивание

Подсвечивание включено по умолчанию для одних элементов управления, таких как `ListView` и `NavigationView`. Для других элементов вы можете включить его, применив стиль темы. В первую очередь мы добавим программный код XAML, чтобы определить пользовательский интерфейс калькулятора, состоящий из набора кнопок. Затем добавим обработчик события для события `Loaded` страницы, чтобы мы могли применить стиль темы Reveal и другие свойства, перечислив кнопки вместо необходимости вручную устанавливать атрибуты для каждого из них в программном коде XAML.

1. Откройте файл `MainPage.xaml`. Добавьте новую горизонтальную панель в стеке под панелью, которая используется в качестве панели инструментов, и для определения калькулятора добавьте сетку с кнопками:

```
<StackPanel
  Background="{ThemeResource SystemControlAcrylicWindowBrush}">
  <StackPanel Orientation="Horizontal" Padding="4"
              Background="LightGray" Name="toolbar">
    <Button Margin="6" Padding="6" Name="ClickMeButton"
            Click="ClickMeButton_Click">
      Click Me
    </Button>
    <TextBlock Text="First name:"
      VerticalAlignment="Center" Margin="4" />
    <TextBox PlaceholderText="Enter your name"
      VerticalAlignment="Center" Width="200" />
  </StackPanel>
  <StackPanel Orientation="Horizontal">
    <Grid Background="DarkGray" Margin="10"
          Padding="5" Name="gridCalculator">
      <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
      </Grid.RowDefinitions>
      <Button Grid.Row="0" Grid.Column="0" Content="X" />
      <Button Grid.Row="0" Grid.Column="1" Content="/" />
      <Button Grid.Row="0" Grid.Column="2" Content="+" />
      <Button Grid.Row="0" Grid.Column="3" Content="-" />
      <Button Grid.Row="1" Grid.Column="0" Content="7" />
      <Button Grid.Row="1" Grid.Column="1" Content="8" />
      <Button Grid.Row="1" Grid.Column="2" Content="9" />
      <Button Grid.Row="1" Grid.Column="3" Content="0" />
      <Button Grid.Row="2" Grid.Column="0" Content="4" />
      <Button Grid.Row="2" Grid.Column="1" Content="5" />
      <Button Grid.Row="2" Grid.Column="2" Content="6" />
      <Button Grid.Row="2" Grid.Column="3" Content="." />
      <Button Grid.Row="3" Grid.Column="0" Content="1" />
      <Button Grid.Row="3" Grid.Column="1" Content="2" />
      <Button Grid.Row="3" Grid.Column="2" Content="3" />
      <Button Grid.Row="3" Grid.Column="3" Content="=" />
    </Grid>
  </StackPanel>
</StackPanel>
```

2. В элемент Page добавьте новый обработчик событий для Loaded:

```
<Page
  ...
  Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
  Loaded="Page_Loaded">
```

3. Щелкните правой кнопкой мыши на Page_Loaded и в контекстном меню выберите пункт Go to Definition (Перейти к определению) или нажмите клавишу F12.
4. В метод Page_Loaded добавьте операторы для циклического перебора всех кнопок калькулятора, задав для них одинаковый размер, и примените стиль Reveal, как показано ниже:

```
private void Page_Loaded(object sender, RoutedEventArgs e)
{
  Style reveal = Resources.ThemeDictionaries[
    "ButtonRevealStyle"] as Style;

  foreach (Button b in gridCalculator.Children.OfType<Button>())
  {
    b.FontSize = 24;
    b.Width = 54;
    b.Height = 54;
    b.Style = reveal;
  }
}
```

5. Запустите приложение, выбрав Debug ▸ Start Without Debugging (Отладка ▸ Запуск без отладки). Обратите внимание, что кнопки калькулятора плоские, серого цвета.
6. Когда пользователь наводит указатель мыши на правый нижний угол кнопки 8, мы видим, что кнопка подсвечивается, как и фрагменты окружающих кнопок (рис. Б.8).
7. Закройте приложение.

Рис. Б.8

Реализация эффекта подсветки несколько изменялась по мере выхода последних полугодовых обновлений для операционной системы Windows 10, поэтому эффект в вашем приложении может немного отличаться.

Установка дополнительных элементов управления

В дополнение к десяткам встроенных элементов управления вы можете установить дополнительные элементы с помощью пакета NuGet. Один лучших пакетов — это *UWP Community Toolkit*.

Больше информации о UWP Community Toolkit можно получить на сайте https://docs.microsoft.com/en-us/windows/communitytoolkit/.

Один из элементов управления, содержащихся в этом пакете, — редактор для Markdown.

Больше информации о проекте Markdown можно найти на сайте https://daringfireball.net/projects/markdown/.

Установим UWP Community Toolkit и рассмотрим некоторые из его элементов управления.

1. В проекте `FluentUwpApp` щелкните правой кнопкой мыши на References (Ссылки) и выберите пункт Manage NuGet Packages (Управление пакетами NuGet).
2. Нажмите кнопку Browse (Обзор), найдите `Microsoft.Toolkit.Uwp.UI.Controls` и нажмите кнопку Install (Установить).
3. Проанализируйте изменения и примите лицензионное соглашение.
4. Чтобы восстановить пакеты (если это необходимо), выберите Build ▸ Build FluentUwpApp (Сборка ▸ Сборка FluentUwpApp).
5. Откройте файл `MainPage.xaml` и в элементе `Page` импортируйте пространство имен набора инструментов с помощью префикса `kit`:

```
<Page
  ...
  xmlns:kit="using:Microsoft.Toolkit.Uwp.UI.Controls"
  mc:Ignorable="d"
  Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
  Loaded="Page_Loaded">
```

6. Добавьте внутри второй горизонтальной панели после сетки калькулятора текстовое поле и текстовый блок с поддержкой Markdown так, чтобы их дан-

ные были связаны и текст в текстовом поле стал источником для отрисовки Markdown, как показано ниже (выделено полужирным шрифтом):

```
<StackPanel Orientation="Horizontal">
  <Grid Background="DarkGray" Margin="10"
        Padding="5" Name="gridCalculator">
  ...
  </Grid>
  <TextBox Name="markdownSource" Text="# Welcome"
           Header="Enter some Markdown text:"
           VerticalAlignment="Stretch" Margin="5"
           AcceptsReturn="True" />
  <kit:MarkdownTextBlock Margin="5"
    Text="{Binding ElementName=markdownSource, Path=Text}"
    VerticalAlignment="Stretch" HorizontalAlignment="Stretch" />
</StackPanel>
```

7. Запустите приложение, выбрав Debug ▸ Start Without Debugging (Отладка ▸ Запуск без отладки). Вы увидите, что пользователь может ввести в текстовое поле синтаксис Markdown и он отобразится в текстовом блоке Markdown, как показано на рис. Б.9.

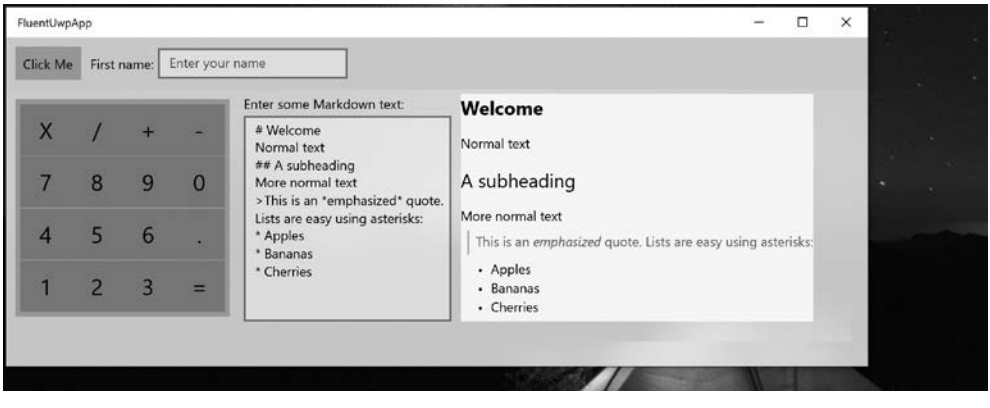

Рис. Б.9

Использование ресурсов и шаблонов

При разработке графических пользовательских интерфейсов вам часто понадобится задействовать ресурсы, например кисти, для окрашивания фона элементов управления. Эти ресурсы можно определить в одном расположении и сделать доступными всему приложению.

Общий доступ к ресурсам

Располагать определение общих ресурсов лучше всего на уровне приложения.

1. На панели Solution Explorer (Обозреватель решений) найдите и откройте файл `App.xaml`.
2. Добавьте следующую разметку к существующей в элементе `<Application>`, чтобы определить кисть с линейным градиентом с ключом `rainbow`, как показано ниже (выделено полужирным шрифтом):

```
<Application
  x:Class="FluentUwpApp.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:FluentUwpApp">

  <Application.Resources>
    <LinearGradientBrush x:Key="rainbow">
      <GradientStop Color="Red" Offset="0" />
      <GradientStop Color="Orange" Offset="0.1" />
      <GradientStop Color="Yellow" Offset="0.3" />
      <GradientStop Color="Green" Offset="0.5" />
      <GradientStop Color="Blue" Offset="0.7" />
      <GradientStop Color="Indigo" Offset="0.9" />
      <GradientStop Color="Violet" Offset="1" />
    </LinearGradientBrush>
  </Application.Resources>

</Application>
```

3. Выберите команду меню Build ▸ Build FluentUwpApp (Сборка ▸ Сборка FluentUwpApp).
4. В файле `MainPage.xaml` измените элемент `StackPanel` с именем `toolbar`, чтобы изменить его фон с `LightGray` на использование статического ресурса — кисти `rainbow`, как показано ниже (выделено полужирным шрифтом):

```
<StackPanel Orientation="Horizontal" Padding="4"
            Background="{StaticResource rainbow}" Name="toolbar">
```

5. Когда вы вводите ссылку на статический ресурс, подсказка IntelliSense отобразит ваш ресурс `rainbow` и встроенные ресурсы (рис. Б.10).

Ресурс может быть экземпляром любого объекта. Чтобы предоставить к нему общий доступ в приложении, определите его в файле App.xaml и присвойте ему уникальный ключ. Чтобы установить свойству элемента значение ресурса, используйте ключевое слово {StaticResource key}.

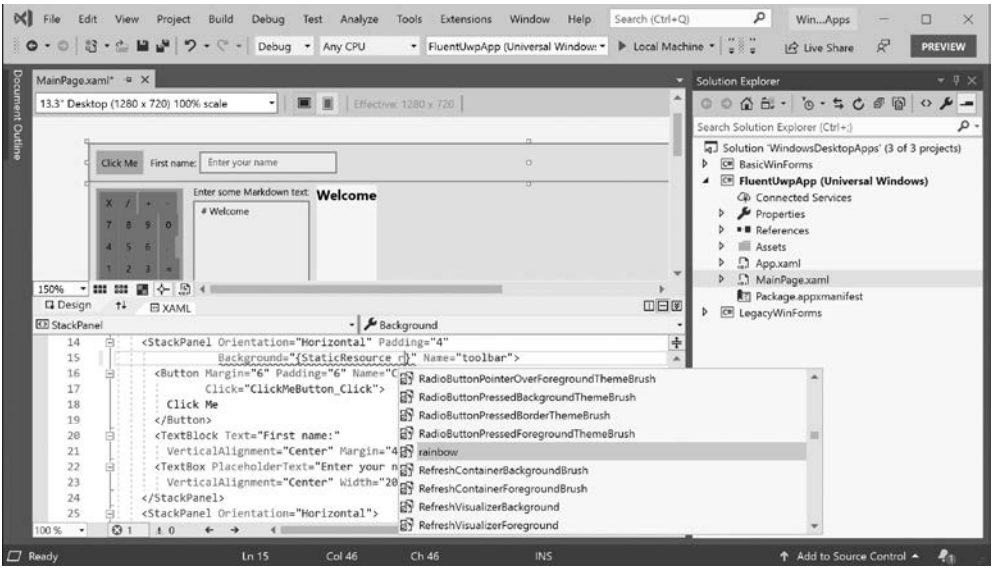

Рис. Б.10

Ресурсы могут быть определены и сохранены в любом элементе XAML, а не только на уровне приложения. Например, если ресурс необходим лишь на `MainPage`, то его можно определить на `MainPage`. Вы также можете динамически загружать файлы XAML во время выполнения.

Более подробную информацию о системе управления ресурсами можно найти на сайте https://docs.microsoft.com/en-us/windows/uwp/app-resources/.

Замена шаблона элемента управления

Вы можете изменить внешний вид элемента управления, заменив его шаблон по умолчанию. Шаблон управления по умолчанию для кнопки плоский и прозрачный.

Один из наиболее часто используемых общих ресурсов — `Style`, который позволяет определять сразу несколько свойств. Если стиль имеет уникальный ключ, то он должен указываться явно, как и в предыдущем примере с линейным градиентом. В случае отсутствия ключ будет применен автоматически на основе свойства `TargetType`.

1. В файле `App.xaml` добавьте следующую разметку в элемент `<Application.Resources>`. Обратите внимание: элемент `Style` автоматически установит для

всех элементов управления, имеющих тип `TargetType`, то есть кнопок, свойство `Template` равным определяемому нами шаблону, как показано ниже (выделено полужирным шрифтом):

```
<Application.Resources>

  <LinearGradientBrush x:Key="rainbow">
  ...
  </LinearGradientBrush>

  <ControlTemplate x:Key="DarkGlassButton"
                   TargetType="Button">
    <Border BorderBrush="#FFFFFFFF"
            BorderThickness="1,1,1,1" CornerRadius="4,4,4,4">
      <Border x:Name="border" Background="#7F000000"
              BorderBrush="#FF000000"
              BorderThickness="1,1,1,1"
              CornerRadius="4,4,4,4">
        <Grid>
          <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="*"/>
          </Grid.RowDefinitions>
          <Border Opacity="0"
                  HorizontalAlignment="Stretch" x:Name="glow"
                  Width="Auto" Grid.RowSpan="2"
                  CornerRadius="4,4,4,4">
          </Border>
          <ContentPresenter HorizontalAlignment="Center"
                            VerticalAlignment="Center"
                            Width="Auto"
                            Grid.RowSpan="2" Padding="4"/>
          <Border HorizontalAlignment="Stretch" Margin="0,0,0,0"
                  x:Name="shine" Width="Auto"
                  CornerRadius="4,4,0,0">
            <Border.Background>
              <LinearGradientBrush EndPoint="0.5,0.9"
                                   StartPoint="0.5,0.03">
                <GradientStop Color="#99FFFFFF" Offset="0"/>
                <GradientStop Color="#33FFFFFF" Offset="1"/>
              </LinearGradientBrush>
            </Border.Background>
          </Border>
          </Grid>
        </Border>
      </Border>
  </ControlTemplate>

  <Style TargetType="Button">
    <Setter Property="Template"
```

```
            Value="{StaticResource DarkGlassButton}" />
      <Setter Property="Foreground" Value="White" />
    </Style>

  </Application.Resources>
```

2. Перезапустите приложение и проанализируйте результат. Обратите внимание на эффект «затененного стекла» кнопки.

Эффект «затененного стекла» не влияет на кнопки калькулятора во время выполнения, поскольку мы заменяем их стили кодом после загрузки страницы.

Привязка данных

При разработке графических пользовательских интерфейсов часто требуется привязывать свойство одного элемента управления к другому или к определенным данным.

Привязка к элементам

Самый простой тип привязки — привязка между элементами.

1. В файле `MainPage.xaml`, после второй горизонтальной панели и внутри внешней вертикальной панели, добавьте текстовый блок с инструкциями, ползунковый регулятор с диапазоном значений от 0 до 360 градусов, сетку, содержащую панель и текстовые блоки, радиальный датчик из UWP Community Toolkit и красный квадрат, который будет вращаться:

```
<TextBlock Margin="10">
  Use the slider to rotate the square:
</TextBlock>
<Slider Value="180" Minimum="0" Maximum="360"
        Name="sliderRotation" Margin="10,0" />
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <StackPanel Orientation="Horizontal"
              VerticalAlignment="Center"
              HorizontalAlignment="Center">
    <TextBlock FontSize="30"
      Text="{Binding ElementName=sliderRotation, Path=Value}" />
    <TextBlock Text="degrees" FontSize="30" Margin="10,0" />
  </StackPanel>
  <kit:RadialGauge Grid.Column="1" Minimum="0" Maximum="360"
```

```
    Value="{Binding ElementName=sliderRotation, Path=Value}"
    Height="200" Width="200" />
  <Rectangle Grid.Column="2" Height="100" Width="100" Fill="Red">
    <Rectangle.RenderTransform>
      <RotateTransform
        Angle="{Binding ElementName=sliderRotation, Path=Value}" />
    </Rectangle.RenderTransform>
  </Rectangle>
</Grid>
```

2. Обратите внимание: текст блока, значение радиального датчика и угол поворота вращения привязаны к значению ползункового регулятора с помощью расширения разметки `{Binding}`, в котором указаны имя элемента и имя свойства привязки, также известное как путь.
3. Перезапустите приложение. Нажав и удерживая кнопку мыши, перетащите ползунковый регулятор, чтобы повернуть красный квадрат, как показано на рис. Б.11.

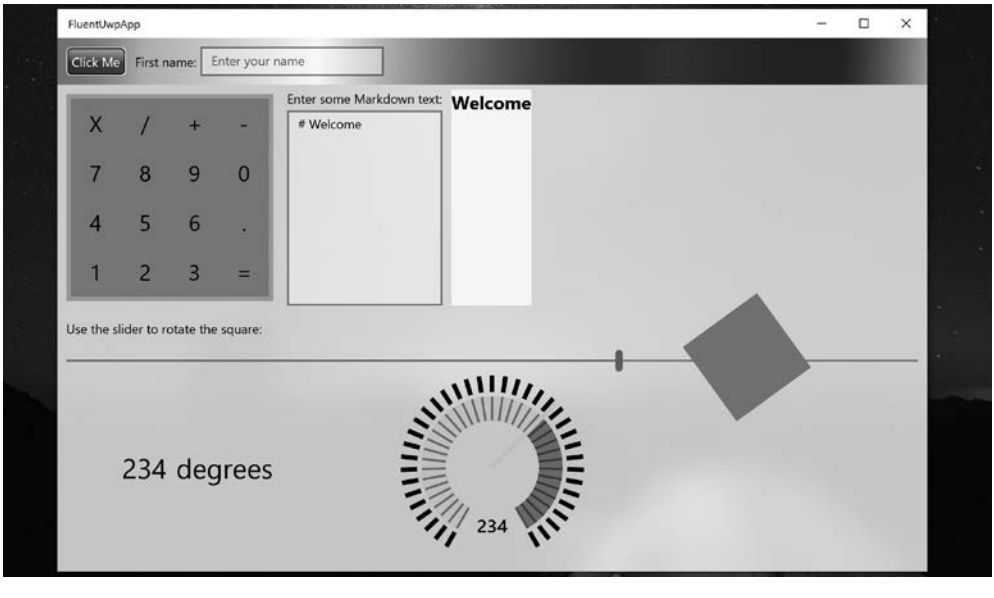

Рис. Б.11

Практические задания

Проверим полученные знания. Для этого ответьте на несколько вопросов, выполните приведенные упражнения и посетите указанные ресурсы, чтобы получить дополнительную информацию.

Упражнение Б.1. Проверочные вопросы

Ответьте на следующие вопросы.

1. Платформа .NET 5 — кросс-платформенная. Приложения Windows Forms и WPF могут работать на .NET 5. Могут ли эти приложения работать на операционных системах macOS и Linux?
2. Как приложение Windows Forms определяет свой пользовательский интерфейс и почему это сложная задача?
3. Как WPF- или UWP-приложение может определять свой пользовательский интерфейс и почему это хорошо для разработчиков?
4. Назовите пять контейнеров макета и варианты расположения их дочерних элементов.
5. В чем разница между `Margin` и `Padding` для такого элемента, как `Button`?
6. Как обработчики событий присоединяются к объекту с помощью XAML?
7. Как работают стили XAML?
8. Где вы можете определить ресурсы?
9. Каким образом вы можете связать свойство одного элемента со свойством другого?
10. Почему вам может потребоваться реализовать интерфейс `IValueConverter`?

Упражнение Б.2. Дополнительные ресурсы

Посетите следующие сайты, чтобы получить дополнительную информацию по темам, приведенным в этой главе:

- подготовка устройства к разработке: https://docs.microsoft.com/en-us/windows/uwp/get-started/enable-your-device-for-development;
- введение в работу с приложениями для операционной системы Windows 10: https://docs.microsoft.com/ru-ru/windows/uwp/get-started/;
- введение в работу с Windows Community Toolkit: https://docs.microsoft.com/ru-ru/windows/communitytoolkit/getting-started;
- проектирование и разработка Windows-приложений: https://docs.microsoft.com/ru-ru/windows/uwp/design/;
- разработка UWP-приложений: https://docs.microsoft.com/ru-ru/windows/uwp/develop/.

Резюме

В этом приложении вы узнали, что при создании настольных Windows-приложений платформа .NET 5 поддерживает более старые технологии, включая Windows Forms и WPF.

Вы узнали, как с помощью языка XAML и новой системы Fluent Design создавать графические пользовательские интерфейсы, включая такие функции, как Acrylic (акриловые материалы) и Reveal (подсвечивание).

Вы также узнали, как совместно использовать ресурсы в приложении, заменить шаблон элемента управления и как связать данные и элементы управления.