18 Создание и использование специализированных сервисов

В этой главе вы познакомитесь с основами нескольких сервисных технологий, которые используются в более специализированных сценариях, чем веб-сервисы общего назначения. Как только вы поймете концепции и преимущества каждой из них, вы сможете углубиться в те из них, которые вас интересуют больше всего.

В этой главе:

- специализированные сервисные технологии;
- предоставление данных в качестве веб-сервиса с помощью OData;
- предоставление данных как сервиса с помощью GraphQL;
- реализация сервисов с помощью gRPC;
- реализация взаимодействия в режиме реального времени с помощью SignalR;
- реализация бессерверных сервисов с помощью платформы Azure Functions;
- сервисы идентификации;
- обзор вариантов специализированных сервисов.

Специализированные сервисные технологии

ASP.NET Core Web API часто лучше всего подходит для реализации веб-сервиса общего назначения, но это не единственная технология для реализации сервисов или обмена данными между компонентами распределенного приложения.

Мы не будем подробно описывать эти технологии, однако вам необходимо знать, как они функционируют и когда их следует использовать.

Windows Communication Foundation (WCF)

В 2006 году Microsoft представила релиз .NET Framework 3.0 с несколькими новыми базовыми платформами, одной из которых была *Windows Communication Foundation (WCF)*. Эта платформа абстрагировала реализацию бизнес-логики сервиса от инфраструктуры коммуникационных технологий, чтобы в будущем можно было легко перейти на альтернативный вариант или даже иметь несколько механизмов для связи с сервисом.

В WCF интенсивно применяется конфигурация XML для декларативного определения конечных точек, включая их адрес (address), привязку (binding) и контракт (contract). Это так называемые ABC конечных точек WCF. Разобравшись, как это сделать, вы поймете, что WCF — это эффективная и в то же время гибкая технология.

Компания Microsoft решила официально не переносить WCF на современную .NET, но существует проект OSS, принадлежащий сообществу и называемый *Core WCF*, которым управляет .NET Foundation. Если вам нужно перенести существующий сервис из .NET Framework на современную .NET или создать клиент для сервиса WCF, то можете использовать Core WCF. Имейте в виду: Core WCF никогда не станет полноценным портом, поскольку части WCF зависят от операционной системы Windows.

Такие технологии, как WCF, позволяют создавать распределенные приложения. Клиентское приложение может выполнять *удаленные вызовы процедур* (remote procedure calls, RPC) серверного приложения. Вместо того чтобы использовать порт WCF, мы могли бы задействовать альтернативную технологию RPC, например gRPC, которая рассматривается далее в этой главе.

Предоставление данных в виде веб-сервиса с помощью OData

Одним из самых распространенных применений веб-сервиса является предоставление базы данных клиентам, которые не понимают, как работать с собственной базой данных. Другим распространенным способом использования является предоставление упрощенного или абстрактного API, который предоставляет только аутентифицированный интерфейс для подмножества данных.

В главе 10 вы узнали, как создать модель EF Core, чтобы предоставить доступ к базе данных любому приложению .NET или сайту. Но как насчет приложений и сайтов, не относящихся к .NET? Я знаю, что это сложно представить, но не все разработчики используют .NET!

К счастью, все платформы разработки поддерживают HTTP, чтобы можно было вызывать веб-сервисы, а в ASP.NET Core есть пакет, позволяющий сделать это легко и просто с помощью стандарта OData.

OData

OData (Open Data Protocol, Открытый протокол данных) — это стандарт OASIS, одобренный ISO/IEC, который определяет набор рекомендаций по созданию и использованию RESTful API. Компания Microsoft создала его в 2007 году и выпустила версии 1.0, 2.0 и 3.0 в рамках своей программы Microsoft Open Specification Promise. Версия 4.0 была стандартизирована в OASIS и выпущена в 2014 году.

В ASP.NET Core OData используется OData версии 4.0.

OData основан на HTTP и имеет несколько конечных точек для поддержки различных версий и наборов сущностей.

В отличие от традиционных Web API, в которых сервис определяет все методы и возвращаемые данные, OData использует строки запросов, позволяя клиенту иметь больший контроль над возвращаемыми данными, и сводит к минимуму круговые задержки. Например, клиенту может понадобиться только два поля данных, `ProductName` и `Cost`, и соответствующий объект `Supplier`, причем только для продуктов, в которых `ProductName` содержит слово `burger`, а стоимость меньше 4,95, как показано ниже:

```
GET https://example.com/v1/products?$filter=contains(ProductName, 'burger') and
Cost lt 4.95&$orderby=Country,Cost&$select=ProductName,Cost&$expand=Supplier
```

Создание веб-сервиса, поддерживающего OData

Не существует шаблона `dotnet new` для ASP.NET Core OData, и, кроме того, в нем используются классы контроллера, поэтому мы будем использовать шаблон проекта Web API, а затем добавим ссылки на пакеты для поддержки OData.

1. Откройте редактор кода и создайте проект с такими настройками:
   1) шаблон проекта: ASP.NET Core Web API/`webapi`;
   2) файл и папка рабочей области/решения: `PracticalApps`;
   3) файл и папка проекта: `Northwind.OData`;
   4) другие параметры Visual Studio: Authentication Type: None (Тип аутентификации: Нет), флажок Configure for HTTPS (Настроить для HTTPS) установлен, флажок Enable Docker (Включить Docker) снят, флажок Enable OpenAPI support (Включить поддержку OpenAPI) установлен.

В программе Visual Studio Code выберите `Northwind.OData` в качестве активного проекта OmniSharp.

2. Добавьте ссылку на пакет для ASP.NET Core OData, как показано ниже:

```
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.OData"
    Version="8.0.1" />
  <PackageReference Include="Swashbuckle.AspNetCore"
    Version="6.1.4" />
</ItemGroup>
```

Номера версий вышеуказанных пакетов NuGet, вероятно, изменятся в сторону увеличения после публикации книги. В качестве общего руководства рекомендуется использовать последнюю версию пакета.

3. Добавьте ссылку на проект контекста базы данных Northwind для SQLite или SQL Server:

```
<ItemGroup>
  <!-- при необходимости смените Sqlite на SqlServer -->
  <ProjectReference Include=
"..\Northwind.Common.DataContext.Sqlite\Northwind.Common.DataContext.
Sqlite.csproj" />
</ItemGroup>
```

4. В папке Northwind.OData удалите файл `WeatherForecast.cs`.
5. В папке Controllers удалите файл `WeatherForecastController.cs`.
6. В файле `Program.cs` настройте метод `UseUrls` на указание порта `5004` для HTTPS, как показано ниже (выделено жирным шрифтом):

```
var builder = WebApplication.CreateBuilder(args);

builder.WebHost.UseUrls("https://localhost:5004");
```

7. Соберите проект `Northwind.OData`.

Определение моделей OData для моделей EF Core

Первая задача — определить, что мы хотим предоставить в веб-сервисе в качестве моделей OData. Все под вашим контролем, поэтому, если у вас есть существующая модель EF Core, как у нас для Northwind, вам не обязательно раскрывать ее всю. Вам даже не обязательно использовать модели EF Core. Источник данных может быть любым, хотя в этой книге мы рассмотрим его использование только в EF Core, поскольку это наиболее распространенный случай применения разработчиками .NET.

Определим две модели OData: одну для отображения каталога продукции Northwind, то есть таблиц категорий и продуктов, а другую — для отображения клиентов, их заказов и связанных таблиц.

1. В файле `Program.cs` импортируйте пространства имен для работы с OData и нашу модель EF Core для базы данных Northwind:

```
using Microsoft.AspNetCore.OData; // метод расширения AddOData
using Microsoft.OData.Edm; // IEdmModel
using Microsoft.OData.ModelBuilder; // ODataConventionModelBuilder
using Packt.Shared; // NorthwindContext и модели сущностей
```

2. В конце файла `Program.cs` добавьте метод для определения и возврата модели OData для каталога Northwind, которая будет отображать только наборы сущностей, то есть таблицы для категорий, товаров и поставщиков:

```
IEdmModel GetEdmModelForCatalog()
{
  ODataConventionModelBuilder builder = new();
  builder.EntitySet<Category>("Categories");
  builder.EntitySet<Product>("Products");
  builder.EntitySet<Supplier>("Suppliers");
  return builder.GetEdmModel();
}
```

3. Добавьте метод для определения модели OData для заказов клиентов Northwind и обратите внимание, что один и тот же набор сущностей может использоваться в нескольких моделях OData, как в случае с `Products`:

```
IEdmModel GetEdmModelForOrderSystem()
{
  ODataConventionModelBuilder builder = new();
  builder.EntitySet<Customer>("Customers");
  builder.EntitySet<Order>("Orders");
  builder.EntitySet<Employee>("Employees");
  builder.EntitySet<Product>("Products");
  builder.EntitySet<Shipper>("Shippers");
  return builder.GetEdmModel();
}
```

4. В разделе конфигурации служб после вызова метода `AddControllers` добавьте вызов метода расширения `AddOData`, чтобы определить две модели OData и использовать такие функции, как проекция, фильтрация и сортировка:

```
builder.Services.AddControllers()
  .AddOData(options => options
    // регистрация моделей OData различных версий
    .AddRouteComponents(routePrefix: "catalog",
      model: GetEdmModelForCatalog())
```

```
    .AddRouteComponents(routePrefix: "ordersystem",
      model: GetEdmModelForOrderSystem())

    // включение параметра запросов
    .Select() // включение $select для проекции
    .Expand() // включение $expand для навигации по сущностям
    .Filter() // включение $filter
    .OrderBy() // включение $orderby
    .SetMaxTop(100) // включение $top
    .Count() // включение $count
  );
```

5. Добавьте операторы перед вызовом метода `AddControllers`, чтобы зарегистрировать контекст базы данных `Northwind`:

```
builder.Services.AddNorthwindContext();
```

6. В папке Properties откройте файл `launchSettings.json`.
7. В профиле `Northwind.OData` измените `applicationUrl`, чтобы использовать порт `5004`, как показано ниже:

```
"applicationUrl": "https://localhost:5004",
```

Тестирование моделей OData

Теперь мы можем проверить, правильно ли определены модели OData.

1. Запустите веб-сервис `Northwind.OData`.
2. Запустите браузер Google Chrome.
3. Перейдите по адресу https://localhost:5004/swagger и обратите внимание, что сервис `Northwind.OData v1` задокументирован.
4. В разделе Metadata (Метаданные) выберите вариант GET/catalog (Получить/каталог), затем вариант Try it out (Попробовать), нажмите кнопку Execute (Выполнить) и обратите внимание на тело ответа, которое показывает имена и URL трех наборов сущностей в модели OData каталога:

```
{
  "@odata.context": "https://localhost:5004/catalog/$metadata",
  "value": [
    {
      "name": "Categories",
      "kind": "EntitySet",
      "url": "Categories"
    },
    {
      "name": "Products",
```

```
      "kind": "EntitySet",
      "url": "Products"
    },
    {
      "name": "Suppliers",
      "kind": "EntitySet",
      "url": "Suppliers"
    }
  ]
}
```

5. Щелкните на GET/catalog (Получить/каталог), чтобы свернуть этот раздел.
6. Выберите GET/catalog/$metadata (Получить/каталог/$метаданные), затем Try it out (Попробовать), нажмите кнопку Execute (Выполнить) и обратите внимание, что модель подробно описывает такие сущности, как `Category`, со свойствами и ключами, включая аспекты навигации для продуктов в каждой категории (рис. 18.1).

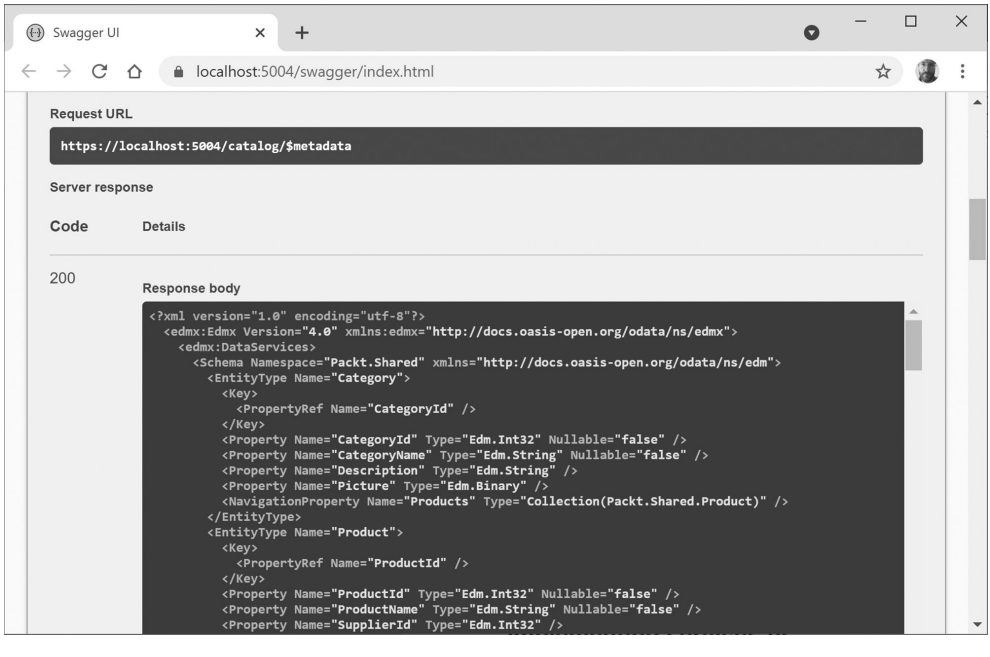

Рис. 18.1. Метаданные модели OData для каталога Northwind

7. Щелкните на GET/catalog/$metadata (Получить/каталог/$метаданные), чтобы свернуть этот раздел.
8. Закройте браузер Google Chrome и завершите работу веб-сервера.

Создание и тестирование контроллеров OData

Далее мы должны создать контроллеры OData, по одному для каждого типа сущностей, чтобы получить данные.

1. В папке Controllers добавьте пустой класс контроллера `CategoriesController`.
2. Измените его содержимое так, чтобы он наследовался от `ODataController`, получил экземпляр контекста базы данных Northwind с помощью введения параметров конструктора и определил два метода `Get` для извлечения всех категорий или одной категории по уникальному ключу:

```
using Microsoft.AspNetCore.Mvc; // IActionResult
using Microsoft.AspNetCore.OData.Query; // [EnableQuery]
using Microsoft.AspNetCore.OData.Routing.Controllers; // ODataController
using Packt.Shared; // NorthwindContext

namespace Northwind.OData.Controllers;

public class CategoriesController : ODataController
{
  private readonly NorthwindContext db;

  public CategoriesController(NorthwindContext db)
  {
    this.db = db;
  }

  [EnableQuery]
  public IActionResult Get()
  {
    return Ok(db.Categories);
  }

  [EnableQuery]
  public IActionResult Get(int key)
  {
    return Ok(db.Categories.Find(key));
  }
}
```

3. Повторите описанный выше шаг для `Products` и `Suppliers`. (На ваше усмотрение вы можете сделать то же самое для других сущностей, чтобы включить модель OData системы заказов. Обратите внимание, что `CustomerId` — это строка, а не целое число.)
4. Запустите веб-сервис `Northwind.OData`.
5. Запустите браузер Google Chrome.

6. Перейдите по адресу https://localhost:5004/swagger и обратите внимание, что наборы сущностей Categories, Products и Suppliers теперь можно использовать, поскольку вы создали для них контроллеры OData.
7. Выберите пункт GET/catalog/Categories (Получить/каталог/Категории), затем Try it out (Попробовать), нажмите кнопку Execute (Выполнить) и обратите внимание на тело ответа в формате JSON, содержащее все категории в наборе сущностей, как частично показано в этом выводе:

```
{
  "@odata.context": "https://localhost:5004/catalog/$metadata#Categories",
  "value": [
    {
      "CategoryId": 1,
      "CategoryName": "Beverages",
      "Description": "Soft drinks, coffees, teas, beers, and ales",
      "Picture": null
    },
    {
      "CategoryId": 2,
      "CategoryName": "Condiments",
      "Description": "Sweet and savory sauces, relishes, spreads,
      and seasonings",
      "Picture": null
    },
    ...
  ]
}
```

8. Закройте браузер Google Chrome и завершите работу веб-сервера.

Тестирование контроллеров OData с помощью REST Client

Использование пользовательского интерфейса Swagger для тестирования контроллеров OData может быстро стать неудобным. Лучшим инструментом является расширение для программы Visual Studio Code под названием REST Client.

1. Если вы еще не установили REST Client разработчика Huachao Mao (`humao.rest-client`), то установите его в программе Visual Studio Code прямо сейчас.
2. Откройте редактор кода и запустите веб-сервис проекта `Northwind.OData`.
3. В программе Visual Studio Code в каталоге PracticalApps создайте папку RestClientTests, если она еще не существует, а затем откройте ее.
4. В папке RestClientTests создайте файл `odata-catalog.http` и укажите в нем запрос на получение всех категорий:

```
GET https://localhost:5004/catalog/categories/ HTTP/1.1
```

5. Нажмите кнопку Send Request (Отправить запрос) и обратите внимание, что ответ совпадает с тем, который был возвращен Swagger, — документ JSON, содержащий все категории.
6. В файле `odata-catalog.http` добавьте дополнительные запросы, разделенные `###`, как показано в табл. 18.1.

Таблица 18.1. Дополнительные запросы

Запрос	Ответ
`https://localhost:5004/catalog/categories(3)`	`{ "@odata.context": "https://localhost:5004/catalog/$metadata#Categories/$entity", "CategoryId":3, "CategoryName":"Confections", "Description":"Desserts, candies, and sweet breads", "Picture":null }`
`https://localhost:5004/catalog/categories/3`	Аналогично тому, как указано выше
`https://localhost:5004/catalog/categories/$count`	`8`
`https://localhost:5004/catalog/products`	Документ JSON, содержащий все продукты
`https://localhost:5004/catalog/products/$count`	`77`
`https://localhost:5004/catalog/products(2)`	`{ "@odata.context": "https://localhost:5004/catalog/$metadata#Products/$entity", "ProductId": 2, "ProductName": "Chang", "SupplierId": 1, "CategoryId": 1, "QuantityPerUnit": "24 - 12 oz bottles", "UnitPrice": 19, "UnitsInStock": 17, "UnitsOnOrder": 40, "ReorderLevel": 25, "Discontinued": false }`
`https://localhost:5004/catalog/suppliers`	Документ JSON, содержащий данные обо всех поставщиках
`https://localhost:5004/catalog/suppliers/$count`	`29`

Запрос моделей OData

Для выполнения произвольных запросов к модели OData мы ранее включили выборку, фильтрацию и упорядочение. Одним из преимуществ OData является то, что в нем определены стандартные параметры запросов, как показано в табл. 18.2.

Таблица 18.2. Стандартные параметры запросов

Опция	Описание	Пример
`$select`	Выбирает свойства для каждой сущности	`$select=CategoryId,CategoryName`
`$expand`	Выбирает связанные сущности через навигационные свойства	`$expand=Products`
`$filter`	Выражение оценивается для каждого ресурса, и в ответ включаются только те объекты, для которых выражение истинно	`$filter=startswith(ProductName,'ch') or (UnitPrice gt 50)`
`$orderby`	Сортирует сущности по перечисленным свойствам в порядке возрастания (по умолчанию) или убывания	`$orderby=UnitPrice desc,ProductName`
`$top`	Пропускает указанное количество элементов. Берет указанное количество элементов	`$skip=40&$take=10`

По соображениям производительности пакетная обработка с помощью `$skip` и `$top` по умолчанию отключена.

Операторы OData

В OData есть операторы для использования с параметром `$filter`, как показано в табл. 18.3.

Таблица 18.3. Операторы для использования с параметром $filter

Операция	Описание
`eq`	Равно
`ne`	Не равно
`lt`	Меньше
`gt`	Больше
`le`	Меньше или равно
`ge`	Больше или равно

Продолжение ⇩

Таблица 18.3 *(продолжение)*

Операция	Описание
`and`	И
`or`	Или
`not`	Не
`add`	Арифметическое сложение для чисел и значений даты/времени
`sub`	Арифметическое вычитание для чисел и значений даты/времени
`mul`	Арифметическое вычитание для чисел
`div`	Арифметическое деление для чисел
`mod`	Арифметическое деление по модулю для чисел

Функции OData

В OData есть функции для использования с параметром `$filter`, как показано в табл. 18.4.

Таблица 18.4. Функции для использования с параметром $filter

Операция (-и)	Описание
`startswith(property, 'value')`	Текстовые значения, которые начинаются с указанного значения
`endswith(property, 'value')`	Текстовые значения, которые заканчиваются указанным значением
`concat(property, 'value')`	Конкатенация двух текстовых значений
`contains(property, 'value')`	Текстовые значения, содержащие указанное значение
`indexof(property, 'value')`	Возвращает позицию текстового значения
`length(property)`	Возвращает длину текстового значения
`substring`	Извлекает подстроку из текстового значения
`tolower`	Преобразует в строчные буквы
`toupper`	Преобразует в верхний регистр
`trim`	Обрезает пробелы до и после текстового значения
`now`	Текущая дата и время
`date`, `day`, `month`, `year`	Извлекает компоненты даты
`time`, `hour`, `minute`, `second`	Извлекает компоненты времени

Обзор запросов OData

Поэкспериментируем с некоторыми запросами OData.

1. В папке RestClientTests создайте файл `odata-catalog-queries.http` и добавьте в него запрос на получение всех категорий:

```
GET https://localhost:5004/catalog/categories/
  ?$select=CategoryId,CategoryName
```

2. Нажмите кнопку Send Request (Отправить запрос) и обратите внимание на ответ — документ JSON, содержащий все категории со свойствами `CategoryId` и `CategoryName`.
3. В файле `odata-catalog-queries.http` добавьте запрос на получение продуктов с названиями, начинающимися на Ch, таких как `Chai` и `Chef Anton's Gumbo Mix`, или имеющими цену за единицу более 50, например, `Mishi Kobe Niku` или `Sir Rodney's Marmalade`, как показано ниже:

```
GET https://localhost:5004/catalog/products/
  ?$filter=startswith(ProductName,'Ch') or (UnitPrice gt 50)
```

4. В файле `odata-catalog-queries.http` добавьте запрос для получения продуктов, отсортированных по цене, с самыми дорогими вверху, а затем отсортированных по названию продукта, и включите только свойства `ID`, названия и цены, как показано ниже:

```
GET https://localhost:5004/catalog/products/
  ?$orderby=UnitPrice desc,ProductName
  &$select=ProductId,ProductName,UnitPrice
```

5. В файле `odata-catalog-queries.http` добавьте запрос для получения категорий и связанных с ними продуктов:

```
GET https://localhost:5004/catalog/categories/
?$select=CategoryId,CategoryName
&$expand=Products
```

Логирование запросов OData

Как работают запросы OData? Выясним это, добавив функцию записи событий в контекст базы данных Northwind, чтобы увидеть реальные SQL-запросы, которые выполняются.

1. В проекте `Northwind.Common.DataContext.Sqlite` (и `SqlServer`) добавьте файл `ConsoleLogger.cs`.

2. Определите в файле три класса, один для реализации `ILoggerFactory`, один для реализации `ILoggerProvider` и один для реализации `ILogger`:

```
using Microsoft.Extensions.Logging;

using static System.Console;

namespace Packt.Shared;

public class ConsoleLoggerFactory : ILoggerFactory
{
  public void AddProvider(ILoggerProvider provider) { }

  public ILogger CreateLogger(string categoryName)
  {
    return new ConsoleLogger();
  }

  public void Dispose() { }
}

public class ConsoleLogger : ILogger
{
  public IDisposable BeginScope<TState>(TState state)
  {
    return null;
  }

  public bool IsEnabled(LogLevel logLevel)
  {
    switch(logLevel)
    {
      case LogLevel.Trace:
      case LogLevel.Information:
      case LogLevel.None:
        return false;
      case LogLevel.Debug:
      case LogLevel.Warning:
      case LogLevel.Error:
      case LogLevel.Critical:
      default:
        return true;
    };
  }

  public void Log<TState>(LogLevel logLevel,
    EventId eventId, TState state, Exception exception,
    Func<TState, Exception, string> formatter)
  {
    if (eventId.Id == 20100) // выполнить оператор SQL
    {
      Write($"Level: {logLevel}, Event Id: {eventId.Id}");
```

```
        // выводим состояние или исключение, только если оно существует
        if (state != null)
        {
          Write($", State: {state}");
        }

        if (exception != null)
        {
          Write($", Exception: {exception.Message}");
        }
        WriteLine();
      }
    }
}
```

3. В файле `NorthwindContextExtensions.cs` в методе `AddNorthwindContext` после вызова `UseSqlServer` или `UseSqlite` вызовите `UseLoggerFactory` для регистрации вашего пользовательского консольного логера (средства ведения журнала), как показано ниже:

```
services.AddDbContext<NorthwindContext>(options =>
  options.UseSqlServer(connectionString) // or UseSqlite(...)
  .UseLoggerFactory(new ConsoleLoggerFactory())
);
```

4. Запустите веб-сервис `Northwind.OData`.
5. Запустите браузер Google Chrome.
6. Перейдите по адресу https://localhost:5004/catalog/products/?$filter=startswith(ProductName,'Ch') или (UnitPrice gt 50)&$select=ProductId,ProductName,UnitPrice.
7. В браузере Google Chrome обратите внимание на результат:


```
{"@odata.context":"https://localhost:5004/catalog/$metadata#Products
(ProductId,ProductName,UnitPrice)","value":[{"ProductId":1,"ProductName":
"Chai","UnitPrice":18.0000},{"ProductId":2,"ProductName":"Chang",
"UnitPrice":19.0000},{"ProductId":4,"ProductName":"Chef Anton's Cajun
Seasoning","UnitPrice":22.0000},{"ProductId":5,"ProductName":"Chef
Anton's Gumbo Mix","UnitPrice":21.3500},{"ProductId":9,"ProductName":
"Mishi Kobe Niku","UnitPrice":97.0000},{"ProductId":18,"ProductName":
"Carnarvon Tigers","UnitPrice":62.5000},{"ProductId":20,"ProductName":
"Sir Rodney's Marmalade","UnitPrice":81.0000},{"ProductId":29,
"ProductName":"Th\u00fcringer Rostbratwurst","UnitPrice":123.7900},
{"ProductId":38,"ProductName":"C\u00f4te deBlaye","UnitPrice":263.5000},
{"ProductId":39,"ProductName":"Chartreuse verte","UnitPrice":18.0000},
{"ProductId":48,"ProductName":"Chocolade","UnitPrice":12.7500},
{"ProductId":51,"ProductName":"Manjimup Dried Apples","UnitPrice":
53.0000},{"ProductId":59,"ProductName":"Raclette Courdavault",
"UnitPrice":55.0000}]}
```

8. В командной строке или в терминале обратите внимание на зарегистрированный оператор SQL, например, при использовании поставщика базы данных SQL Server:

```
Level: Debug, Event Id: 20100, State: Executing DbCommand [Parameters=
[@__TypedProperty_0='?' (Size = 4000), @__TypedProperty_1='?' (DbType =
Decimal)], CommandType='Text', CommandTimeout='30']
SELECT [p].[ProductId], [p].[ProductName], [p].[UnitPrice]
FROM [Products] AS [p]
WHERE ((@__TypedProperty_0 = N'') OR (LEFT([p].[ProductName],
LEN(@__TypedProperty_0)) = @__TypedProperty_0)) OR ([p].[UnitPrice] >
@__TypedProperty_1)
```

Может показаться, что метод действия Get в ProductsController возвращает всю таблицу Products, но на самом деле он возвращает объект IQueryable<Products>. Другими словами, он возвращает запрос LINQ, а не его результаты. Мы дополнили метод действия Get атрибутом [EnableQuery]. Это позволяет OData расширить запрос LINQ фильтрами, проекциями, сортировкой и т. д., и только после этого он выполняет запрос, сериализует результаты и возвращает их клиенту. Это делает сервисы OData максимально гибкими и эффективными.

Контроль версий контроллеров OData

Рекомендуется планировать будущие версии ваших моделей OData, которые могут иметь различные схемы и поведение.

Чтобы сохранить обратную совместимость, вы можете использовать префиксы URL OData для указания номера версии.

1. В проекте `Northwind.OData` в файле `Program.cs` в разделе конфигурации сервисов после добавления двух моделей OData для `catalog` и `ordersystem` добавьте третью модель OData, которая имеет номер версии:

```
.AddRouteComponents(routePrefix: "catalog",
  model: GetEdmModelForCatalog())
.AddRouteComponents(routePrefix: "ordersystem",
  model: GetEdmModelForOrderSystem())
.AddRouteComponents(routePrefix: "v{version}",
  model: GetEdmModelForCatalog())
```

2. В файле `ProductsController.cs` статически импортируйте `Console`, а затем измените методы `Get`, добавив строковый параметр `version`, и используйте его для изменения поведения методов, если в запросе указана версия 2, как показано ниже:

```
[EnableQuery]
public IActionResult Get(string version = "1")
{
  WriteLine($"ProductsController version {version}.");
  return Ok(db.Products);
}

[EnableQuery]
public IActionResult Get(int key, string version = "1")
{
  WriteLine($"ProductsController version {version}.");
  Product? p = db.Products.Find(key);
  if (p is null)
  {
    return NotFound($"Product with id {key} not found.");
  }
  if (version == "2")
  {
    p.ProductName += " version 2.0";
  }
  return Ok(p);
}
```

3. Откройте редактор кода и запустите веб-сервис проекта `Northwind.OData`.
4. В программе Visual Studio Code в файле `odata-catalog-queries.http` добавьте запрос на получение продукта с ID `50` с помощью модели v2 OData:

```
GET https://localhost:5004/v2/products(50)
```

5. Нажмите кнопку Send Request (Отправить запрос) и обратите внимание, что в ответ будет получен продукт, к названию которого добавлено `version 2.0`, как показано ниже:

```
{
  "@odata.context": "https://localhost:5004/v2/$metadata#Products/
  $entity",
  "ProductId": 50,
  "ProductName": "Valkoinen suklaa version 2.0",
  "SupplierId": 23,
  "CategoryId": 3,
  "QuantityPerUnit": "12 - 100 g bars",
  "UnitPrice": 16.2500,
  "UnitsInStock": 65,
  "UnitsOnOrder": 0,
  "ReorderLevel": 30,
  "Discontinued": false
}
```

Добавление сущностей с помощью POST

Чаще всего с помощью OData предоставляется Web API, поддерживающий пользовательские запросы. Вы также можете поддерживать операции CRUD, такие как вставка.

1. В файле `ProductsController.cs` добавьте метод действия для ответа на `POST`-запросы:

```
public IActionResult Post([FromBody] Product product)
{
  db.Products.Add(product);
  db.SaveChanges();
  return Created(product);
}
```

2. Запустите веб-сервис.
3. Создайте файл `odata-catalog-insert-product.http`, как показано в следующем HTTP-запросе:

```
POST https://localhost:5004/catalog/products
Content-Type: application/json
Content-Length: 234

{
  "ProductName": "Impossible Burger",
  "SupplierId": 7,
  "CategoryId": 6,
  "QuantityPerUnit": "Pack of 4",
  "UnitPrice": 40.25,
  "UnitsInStock": 50,
  "UnitsOnOrder": 0,
  "ReorderLevel": 30,
  "Discontinued": false
}
```

4. Нажмите кнопку Send Request (Отправить запрос).
5. Обратите внимание на успешный ответ:

```
HTTP/1.1 201 Created
Connection: close
Content-Type: application/json; odata.metadata=minimal; odata.
streaming=true
Date: Sat, 17 Jul 2021 12:01:57 GMT
Server: Kestrel
Location: https://localhost:5004/catalog/Products(80)
```

```
Transfer-Encoding: chunked
OData-Version: 4.0

{
  "@odata.context": "https://localhost:5004/catalog/$metadata
  #Products/$entity",
  "ProductId": 78,
  "ProductName": "Impossible Burger",
  "SupplierId": 7,
  "CategoryId": 6,
  "QuantityPerUnit": "Pack of 4",
  "UnitPrice": 40.25,
  "UnitsInStock": 50,
  "UnitsOnOrder": 0,
  "ReorderLevel": 30,
  "Discontinued": false
}
```

Создание клиента для OData

В этом подразделе рассмотрим, как клиент может вызывать веб-сервис OData.

Если мы хотим запросить у сервиса OData продукты, начинающиеся с букв `Cha`, то нам нужно отправить запрос `GET` с относительным URL, подобным приведенному ниже:

```
catalog/products/?$filter=startswith(ProductName, 'Cha')&$select=ProductId,
ProductName,UnitPrice
```

OData возвращает данные в формате JSON со свойством `value`, которое содержит полученные продукты в виде массива, как показано в следующем документе JSON:

```
{
  "@odata.context": "https://localhost:5004/catalog/$metadata#Products",
  "value": [
    {
      "ProductId": 1,
      "ProductName": "Chai",
      "SupplierId": 1,
      "CategoryId": 1,
      "QuantityPerUnit": "10 boxes x 20 bags",
      "UnitPrice": 18,
      "UnitsInStock": 39,
      "UnitsOnOrder": 0,
      "ReorderLevel": 10,
      "Discontinued": false
    },
```

Мы создадим класс модели, чтобы облегчить десериализацию ответа.

1. В проекте Northwind.Mvc в папке Models создайте файл класса ODataProducts.cs:

```
using Packt.Shared; // Product

namespace Northwind.Mvc.Models;

public class ODataProducts
{
  public Product[]? Value { get; set; }
}
```

2. В файле Program.cs добавьте операторы для регистрации HTTP-клиента для сервиса OData:

```
builder.Services.AddHttpClient(name: "Northwind.OData",
  configureClient: options =>
  {
    options.BaseAddress = new Uri("https://localhost:5004/");
    options.DefaultRequestHeaders.Accept.Add(
      new MediaTypeWithQualityHeaderValue(
      "application/json", 1.0));
  });
```

Добавление страницы сервисов на сайт Northwind MVC

Далее мы создадим страницу сервисов.

1. В папке Controllers откройте файл HomeController.cs и добавьте новый метод действия для сервисов, который вызывает сервис OData для получения продуктов, начинающихся на Cha, и сохраняет результат в словаре ViewData:

```
public async Task<IActionResult> Services()
{
  try
  {
    HttpClient client = clientFactory.CreateClient(
      name: "Northwind.OData");

    HttpRequestMessage request = new(
      method: HttpMethod.Get, requestUri:
      "catalog/products/?$filter=startswith(ProductName, 'Cha')&$select=
      ProductId,ProductName,UnitPrice");

    HttpResponseMessage response = await client.SendAsync(request);
      ViewData["productsCha"] = (await response.Content
        .ReadFromJsonAsync<ODataProducts>())?.Value;
    }
    catch (Exception ex)
```

```
    {
      _logger.LogWarning($"Northwind.OData service exception: {ex.Message}");
    }

    return View();
  }
```

2. В разделе `Views/Shared` в файле `_Layout.cshtml` после элемента навигации `Privacy` добавьте новый элемент навигации, который ведет на страницу сервисов:

```
<li class="nav-item">
  <a class="nav-link text-dark" asp-area=""
    asp-controller="Home" asp-action="Services">Services</a>
</li>
```

3. В папке Views/Home добавьте новое пустое представление `Services.cshtml` и измените его содержимое для отображения продуктов, как показано ниже:

```
@using Packt.Shared
@using Northwind.Common
@{
  ViewData["Title"] = "Services";
  Product[]? products = ViewData["productsCha"] as Product[];
}
<div class="text-center">
  <h1 class="display-4">@ViewData["Title"]</h1>
  @if (ViewData["productsCha"] != null)
  {
    <h2>Products that start with Cha using OData</h2>
    <p>
      @if (products is null)
      {
        <span class="badge badge-info">No products found.</span>
      }
      else
      {
        @foreach (Product p in products)
        {
          <span class="badge badge-info">
            @p.ProductId
            @p.ProductName
            @(p.UnitPrice is null ? "" : p.UnitPrice.Value.ToString("c"))
          </span>
        }
      }
    </p>
  }
</div>
```

4. При необходимости запустите проект `Minimal.Web` без отладки. (Если вы используете Visual Studio 2022, то выберите проект на панели Solution Explorer (Проводник решений), чтобы сделать его текущим выбором, а затем перейдите в Debug ▸ Start Without Debugging (Запуск ▸ Запуск без отладки).)
5. Запустите проект `Northwind.OData` без отладки.
6. Запустите проект `Northwind.Mvc` без отладки.
7. Запустите браузер Google Chrome.
8. Перейдите на страницу Services (Сервисы), нажав меню или перейдя по следующей ссылке: https://localhost:5001/home/services.
9. Обратите внимание, что из сервиса OData возвращаются три продукта, как показано на рис. 18.2.

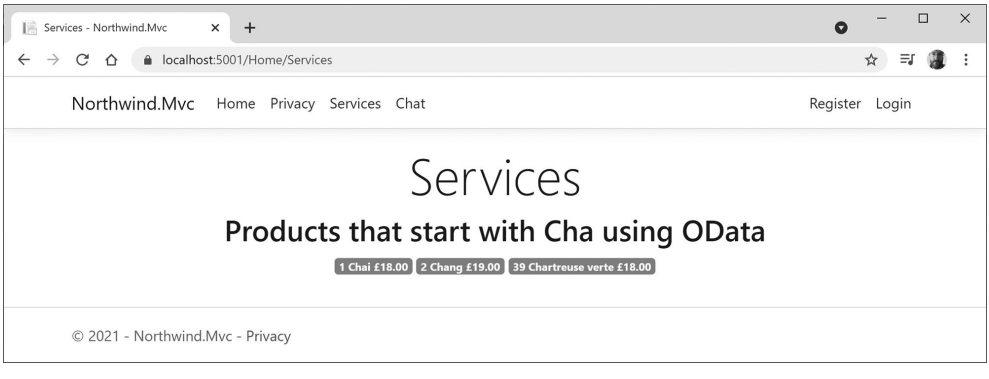

Рис. 18.2. Три названия продуктов начинаются с Cha, полученного из сервиса OData

10. Закройте браузер Google Chrome и завершите работу всех веб-серверов.

Предоставление данных как сервиса с помощью GraphQL

Если вы предпочитаете использовать более современную технологию для предоставления данных как сервиса, то альтернативой OData является GraphQL.

GraphQL

Как и OData, GraphQL — это стандарт для описания ваших данных и последующего запроса к ним, который позволяет клиенту контролировать именно то, что ему нужно. GraphQL был разработан компанией Facebook в 2012 году, а в 2015 году

стал проектом с открытым исходным кодом и в настоящее время управляется GraphQL Foundation.

Некоторые преимущества GraphQL перед OData заключаются в том, что он не требует HTTP, поскольку является транспортно-агностическим (независимым), поэтому вы можете использовать альтернативные транспортные протоколы, такие как WebSockets, а GraphQL имеет единственную конечную точку, обычно просто `/graphql`.

GraphQL использует собственный формат запросов, который немного похож на JSON, но запросы GraphQL не требуют запятых между именами полей, как показано ниже:

```
{
  product (productId: 23) {
    productId
    productName
    cost
    supplier {
      companyName
      country
    }
  }
}
```

Официальным типом данных для документов запросов GraphQL является application/graphql.

Создание сервиса, поддерживающего GraphQL

Для GraphQL не существует шаблона проекта `dotnet new`, поэтому мы будем использовать шаблон проекта Web API (хотя GraphQL не обязательно должен быть размещен в веб-сервисе), а затем добавим ссылки на пакеты для поддержки GraphQL.

1. Откройте редактор кода и создайте проект с такими настройками:
   1) шаблон проекта: ASP.NET Core Web API/`webapi`;
   2) файл и папка рабочей области/решения: `PracticalApps`;
   3) файл и папка проекта: `Northwind.GraphQL`;
   4) другие параметры Visual Studio: Authentication Type: None (Тип аутентификации: Нет), флажок Configure for HTTPS (Настроить для HTTPS) установлен, флажок Enable Docker (Включить Docker) снят, флажок Enable OpenAPI support (Включить поддержку OpenAPI) снят.

Поскольку GraphQL не является традиционным сервисом Web API, Swagger должен быть выключен. Если вы используете командную строку, то добавьте следующий переключатель: dotnet new webapi --no-openapi.

2. В программе Visual Studio Code выберите `Northwind.GraphQL` в качестве активного проекта OmniSharp.
3. Добавьте ссылки на пакеты для основных компонентов сервера GraphQL и пользовательского интерфейса GraphQL Playground, как показано ниже:

```
<ItemGroup>
  <PackageReference
    Include="GraphQL.Server.Transports.AspNetCore"
    Version="5.0.2" />
  <PackageReference
    Include="GraphQL.Server.Transports.AspNetCore.SystemTextJson"
    Version="5.0.2" />
  <PackageReference
    Include="GraphQL.Server.Ui.Playground"
    Version="5.0.2" />
</ItemGroup>
```

Удалите пакет пользовательского интерфейса GraphQL Playground перед развертыванием сервиса в рабочей среде. Хотя вы можете применять Playground только в режиме разработки, любые неиспользуемые пакеты увеличивают потенциальную вероятность атаки злоумышленников.

4. Добавьте ссылку на проект контекста базы данных Northwind для SQLite или SQL Server:

```
<ItemGroup>
  <!-- при необходимости смените Sqlite на SqlServer -->
  <ProjectReference Include=
"..\Northwind.Common.DataContext.Sqlite\Northwind.Common.DataContext.
Sqlite.csproj" />
</ItemGroup>
```

5. В папке Northwind.GraphQL удалите файл `WeatherForecast.cs`.
6. В папке Controllers удалите файл `WeatherForecastController.cs`.
7. Удалите папку Controllers.
8. В файле `Program.cs` добавьте вызов метода расширения `UseUrls`, чтобы указать порт `5005` для HTTPS:

```
var builder = WebApplication.CreateBuilder(args);

builder.WebHost.UseUrls("https://localhost:5005/");
```

9. В папке Properties откройте файл `launchSettings.json` и измените параметры `launchUrl` и `applicationUrl`:

```
"profiles": {
  "Northwind.GraphQL": {
    "commandName": "Project",
    "dotnetRunMessages": "true",
    "launchBrowser": true,
    "launchUrl": "ui/playground",
    "applicationUrl": "https://localhost:5005",
    "environmentVariables": {
      "ASPNETCORE_ENVIRONMENT": "Development"
    }
  },
```

10. Соберите проект `Northwind.GraphQL`.

Определение схемы GraphQL для Hello, World!

Первая задача — определить, что мы хотим представить в виде моделей GraphQL в веб-сервисе.

Определим модель GraphQL для самого простого примера — Hello, World!

1. В проект/папку `Northwind.GraphQL` добавьте файл класса `GreetQuery.cs`.
2. Измените класс, чтобы определить тип графа объектов `greet`, который в качестве ответа выдает обычный текст `"Hello, World!"`:

```
using GraphQL.Types; // ObjectGraphType

namespace Northwind.GraphQL;

public class GreetQuery : ObjectGraphType
{
  public GreetQuery()
  {
    Field<StringGraphType>(name: "greet",
      description: "A query type that greets the world.",
      resolve: context => "Hello, World!");
  }
}
```

3. В проект/папку `Northwind.GraphQL` добавьте файл класса `NorthwindSchema.cs`.
4. Измените класс, чтобы определить схему графа объектов, которая регистрирует класс `GreetQuery` как единственный тип запроса:

```
using GraphQL.Types; // схема

namespace Northwind.GraphQL;
```

```
public class NorthwindSchema : Schema
{
  public NorthwindSchema(IServiceProvider provider) : base(provider)
  {
    Query = new GreetQuery();
  }
}
```

Используйте внедрение параметров конструктора, чтобы получить IServiceProvider, который будет использоваться для получения зарегистрированных сервисов зависимостей.

5. В файле `Program.cs` импортируйте пространство имен для работы с GraphQL, а также запрос и схему GraphQL, которые вы только что определили:

```
using GraphQL.Server; // GraphQLOptions
using Northwind.GraphQL; // GreetQuery, NorthwindSchema
```

6. В разделе конфигурации сервисов после вызова метода `AddControllers` добавьте операторы для регистрации класса схемы в качестве сервиса зависимостей и для добавления поддержки GraphQL:

```
builder.Services.AddScoped<NorthwindSchema>();

builder.Services.AddGraphQL()
  .AddGraphTypes(typeof(NorthwindSchema), ServiceLifetime.Scoped)
  .AddDataLoader()
  .AddSystemTextJson(); // сериализовация ответов в формате JSON
```

Позже контекст базы данных Northwind будет зарегистрирован как scoped dependency service (сервис зависимостей с ограниченной областью действий), поэтому все сервисы, использующие его, также должны быть зарегистрированы как scoped, а не singleton.

7. В разделе настройки конвейера HTTP добавьте операторы для использования GraphQL со схемой Northwind и пользовательского интерфейса Playground, если вы находитесь в режиме разработки:

```
if (builder.Environment.IsDevelopment())
{
  app.UseGraphQLPlayground(); // путь по умолчанию: /ui/playground
}

app.UseGraphQL<NorthwindSchema>(); // путь по умолчанию: /graphql

app.UseHttpsRedirection();
```

8. Запустите проект сервиса `Northwind.GraphQL`.
9. Если вы используете Visual Studio Code, то запустите браузер Google Chrome и перейдите по адресу https://localhost:5005/ui/playground.
10. В левой части напишите безымянный запрос для запроса `greet`, как показано ниже:

```
{
  greet
}
```

11. Нажмите круглую серую кнопку воспроизведения и обратите внимание на URL конечной точки, используемый Playground, https://localhost:5005/graphql, и ответ, как показано в коде ниже и на рис. 18.3:

```
{
  "data": {
    "greet": "Hello, World!"
  },
  "extensions": {}
}
```

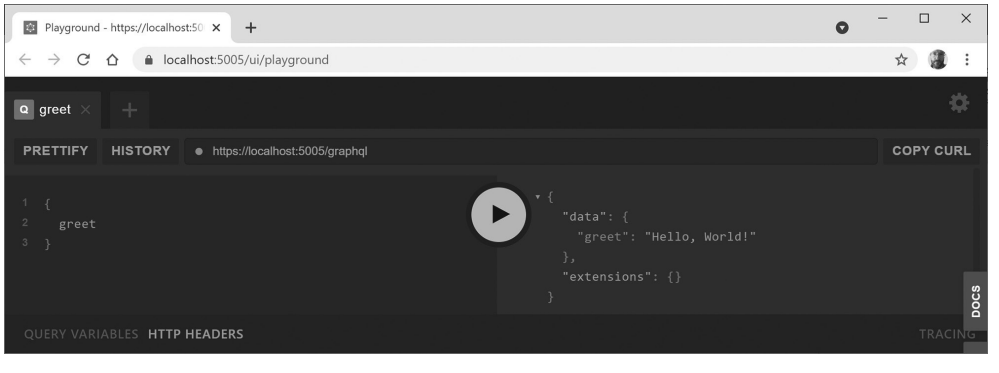

Рис. 18.3. Использование GraphQL Playground для выполнения приветственного запроса

12. Закройте браузер Google Chrome и завершите работу веб-сервера.

Мы также можем создать именованный запрос:

```
query QueryNameGoesHere {
  greet
}
```

Именованные запросы позволяют клиентам идентифицировать запросы и ответы в целях телеметрии, например, при размещении в облачных сервисах Microsoft Azure и мониторинге с помощью Application Insights.

Определение схемы GraphQL для моделей EF Core

Теперь, когда базовый сервис GraphQL успешно работает, добавим типы, чтобы обеспечить возможность запросов к базе данных Northwind.

1. В файле `Program.cs` импортируйте пространство имен для работы с моделью EF Core для базы данных Northwind:

```
using Packt.Shared; // метод AddNorthwindContext
```

2. Добавьте оператор в верхнюю часть раздела конфигурации сервисов, чтобы зарегистрировать класс контекста базы данных Northwind:

```
builder.Services.AddNorthwindContext();
```

3. В проект/папку `Northwind.GraphQL` добавьте файл класса `CategoryType.cs`. Он используется для описания класса сущности `Category` в системе GraphQL.
4. Измените класс `CategoryType`, чтобы определить тип графа объектов, соответствующий структуре модели сущности `Category`:

```
using GraphQL.Types; // ObjectGraphType<T>, ListGraphType<T>
using Packt.Shared; // Category

namespace Northwind.GraphQL;

public class CategoryType : ObjectGraphType<Category>
{
  public CategoryType()
  {
    Name = "Category";
    Field(c => c.CategoryId).Description("Id of the category.");
    Field(c => c.CategoryName).Description("Name of the category.");
    Field(c => c.Description).Description("Description of the category.");
    Field(c => c.Products, type: typeof(ListGraphType<ProductType>))
      .Description("Products in the category.");
  }
}
```

На данном этапе класс ProductType выдаст ошибку, поскольку мы его еще не создали.

5. В проект/папку `Northwind.GraphQL` добавьте файл класса `ProductType.cs`.
6. Определите в классе тип графа объектов, который соответствует структуре модели сущности `Product`:

```
using GraphQL.Types; // ObjectGraphType<T>, IntGraphType, DecimalGraphType
using Packt.Shared; // Category, Product

namespace Northwind.GraphQL;
```

```
public class ProductType : ObjectGraphType<Product>
{
  public ProductType()
  {
    Name = "Product";
    Field(p => p.ProductId).Description("Id of the product.");
    Field(p => p.ProductName).Description("Name of the product.");
    Field(p => p.CategoryId, type: typeof(IntGraphType))
      .Description("CategoryId of the product.");
    Field(p => p.Category, type: typeof(CategoryType))
      .Description("Category of the product.");
    Field(p => p.UnitPrice, type: typeof(DecimalGraphType))
      .Description("Unit price of the product.");
    Field(p => p.UnitsInStock, type: typeof(IntGraphType))
      .Description("Units in stock of the product.");
    Field(p => p.UnitsOnOrder, type: typeof(IntGraphType))
      .Description("Units on order of the product.");
  }
}
```

Простые типы, такие как int и string, автоматически распознаются GraphQL. Нулевые типы, такие как int? и decimal?, используемые свойствами CategoryId и UnitPrice, сложные типы, такие как Category, используемые свойством Category, и небольшие целые числа, такие как short, должны быть явно указаны, иначе GraphQL будет выбрасывать исключения во время выполнения.

7. В проект/папку `Northwind.GraphQL` добавьте файл класса `NorthwindQuery.cs`.
8. Измените класс, чтобы определить тип графа объектов, который имеет три типа запросов для возврата списка категорий, одной категории и товаров для категории:

```
using GraphQL; // метод расширения GetArgument
using GraphQL.Types; // ObjectGraphType, QueryArguments, QueryArgument<T>
using Microsoft.EntityFrameworkCore; // включить метод расширения
using Packt.Shared; // NorthwindContext

namespace Northwind.GraphQL;

public class NorthwindQuery : ObjectGraphType
{
  public NorthwindQuery(NorthwindContext db)
  {
    Field<ListGraphType<CategoryType>>(
      name: "categories",
      description: "A query type that returns a list of all categories.",
      resolve: context => db.Categories.Include(c => c.Products)
    );

    Field<CategoryType>(
```

```
        name: "category",
        description: "A query type that returns a category using its Id.",
        arguments: new QueryArguments(
          new QueryArgument<IntGraphType> { Name = "categoryId" }),
        resolve: context =>
        {
          Category? category = db.Categories.Find(
            context.GetArgument<int>("categoryId"));
          db.Entry(category).Collection(c => c.Products).Load();
          return category;
        }
      );

      Field<ListGraphType<ProductType>>(
        name: "products",
        arguments: new QueryArguments(
          new QueryArgument<IntGraphType> { Name = "categoryId" }),
        resolve: context =>
        {
          Category? category = db.Categories.Find(
            context.GetArgument<int>("categoryId"));
          db.Entry(category).Collection(c => c.Products).Load();
          return category.Products;
        }
      );
    }
  }
```

9. В файле `NorthwindSchema.cs` импортируйте пространство имен для получения сервиса с внедрением зависимостей и для контекста базы данных Northwind:

```
using Packt.Shared; // NorthwindContext
using Microsoft.Extensions.DependencyInjection; // GetRequiredService
```

10. В конструкторе закомментируйте оператор, который задает для `Query` использование `GreetQuery`, и добавьте оператор, который задает для него использование `NorthwindQuery`, получая и передавая требуемый контекст базы данных Northwind:

```
// Query = new GreetQuery();
Query = new NorthwindQuery(provider.GetRequiredService<NorthwindContext>());
```

Изучение запросов GraphQL с помощью Northwind

Теперь мы можем протестировать написание запросов GraphQL для базы данных Northwind.

1. Запустите проект сервиса `Northwind.GraphQL`.
2. Если вы работаете в программе Visual Studio Code, то запустите браузер Google Chrome и перейдите по адресу https://localhost:5005/ui/playground.

3. На Playground перейдите на вкладку Schema (Схема) с правой стороны и обратите внимание на определения схемы, запросов и типов, а также на подсказку IntelliSense, предоставляемую при вводе запроса (рис. 18.4).

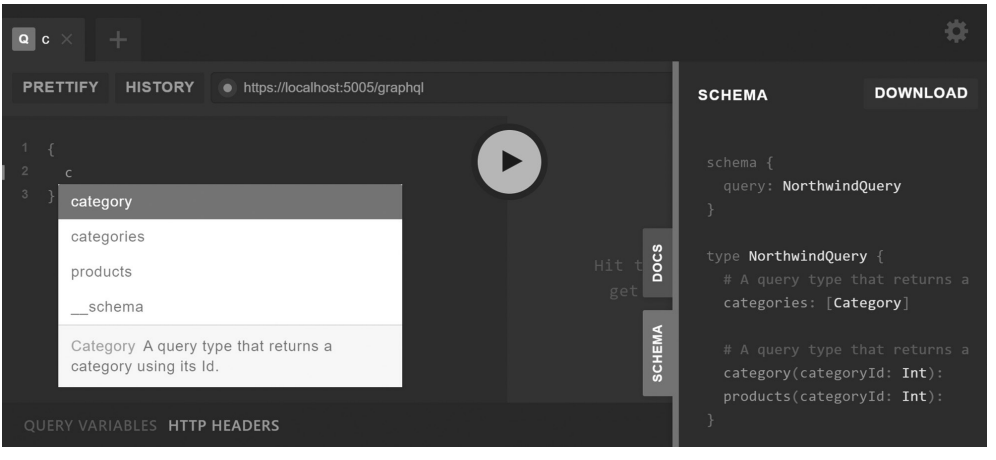

Рис. 18.4. Схема для запроса категорий и продуктов Northwind с помощью GraphQL

4. Перейдите на вкладку Docs (Документы), а затем щелкните на categories (категории), category(...) (категория(...)) и products(...) (продукты(...)), чтобы просмотреть документацию.
5. Снова перейдите на вкладку Docs (Документы), чтобы свернуть панель.
6. В левой части введите именованный запрос, чтобы запросить все категории, как показано ниже:

```
query AllCategories {
  categories {
    categoryId
    categoryName
    description
  }
}
```

7. Нажмите кнопку воспроизведения и обратите внимание на ответ, показанный в следующем частичном выводе:

```
{
  "data": {
    "categories": [
      {
        "categoryId": 1,
```

```
        "categoryName": "Beverages",
        "description": "Soft drinks, coffees, teas, beers, and ales"
      },
      {
        "categoryId": 2,
        "categoryName": "Condiments",
        "description": "Sweet and savory sauces, relishes, spreads,
        and seasonings"
      },
      ...
```

8. Нажмите вкладку +, чтобы открыть новую вкладку, напишите запрос, чтобы запросить категорию с `Id 2`, включая `Id`, название и цену ее продуктов, как показано ниже:

```
query Condiments {
  category (categoryId: 2) {
    categoryId
    categoryName
    products {
      productId
      productName
      unitPrice
    }
  }
}
```

Убедитесь, что буква I в categoryId прописная.

9. Нажмите кнопку воспроизведения и обратите внимание на ответ, показанный в следующем частичном выводе:

```
{
  "data": {
    "category": {
      "categoryId": 2,
      "categoryName": "Condiments",
      "products": [
        {
          "productId": 3,
          "productName": "Aniseed Syrup",
          "unitPrice": 10
        },
        {
          "productId": 4,
          "productName": "Chef Anton's Cajun Seasoning",
```

```
          "unitPrice": 22
        },
        ...
```

10. Нажмите вкладку +, чтобы открыть новую вкладку, введите запрос, чтобы запросить `Id`, название и единицы на складе товаров в категории с `Id 1`, как показано ниже:

```
query BeverageProducts {
  products (categoryId: 1) {
    productId
    productName
    unitsInStock
  }
}
```

11. Нажмите кнопку воспроизведения и обратите внимание на ответ, показанный в следующем частичном выводе:

```
{
  "data": {
    "products": [
      {
        "productId": 1,
        "productName": "Chai",
        "unitsInStock": 39
      },
      {
        "productId": 2,
        "productName": "Chang",
        "unitsInStock": 17
      },
      ...
```

12. Закройте браузер Google Chrome и завершите работу веб-сервера.

Мутации и подписки GraphQL

Помимо запросов, другими стандартными функциями GraphQL являются мутации и подписки.

- *Мутации* дают возможность создавать, обновлять и удалять ресурсы.
- *Подписки* позволяют клиенту получать уведомления об изменении ресурсов. Они лучше всего работают с альтернативными коммуникационными технологиями, такими как WebSockets.

Если вы хотите, чтобы я добавил описание этих функций в следующее издание, то, пожалуйста, свяжитесь со мной и дайте мне знать.

Создание клиента для GraphQL

Наконец, в этом подразделе мы посмотрим, как клиент может вызывать сервис GraphQL.

Мы создадим класс модели, чтобы упростить десериализацию ответа.

1. В проекте `Northwind.Mvc` в папке Models создайте файл класса `GraphQLProducts.cs`:

```
using Packt.Shared; // Product

namespace Northwind.Mvc.Models;

public class GraphQLProducts
{
  public class DataProducts
  {
    public Product[]? Products { get; set; }
  }

  public DataProducts? Data { get; set; }
}
```

2. В файле `Program.cs` добавьте операторы для регистрации HTTP-клиента для сервиса GraphQL:

```
builder.Services.AddHttpClient(name: "Northwind.GraphQL",
  configureClient: options =>
  {
    options.BaseAddress = new Uri("https://localhost:5005/");
    options.DefaultRequestHeaders.Accept.Add(
      new MediaTypeWithQualityHeaderValue(
      "application/json", 1.0));
  });
```

3. В папке Controllers в файле `HomeController.cs` импортируйте пространство имен для работы с текстовыми кодировками:

```
using System.Text; // кодировка
```

4. В методе действия `Services` добавьте операторы для вызова сервиса GraphQL и обратите внимание, что HTTP-запрос является `POST`-запросом, тип носителя — GraphQL, а запрос запрашивает все продукты в категории 8 (это морепродукты):

```
try
{
  HttpClient client = clientFactory.CreateClient(
    name: "Northwind.GraphQL");

  HttpRequestMessage request = new(
    method: HttpMethod.Post, requestUri: "graphql");
```

```
    request.Content = new StringContent(content: @"
      {
        products (categoryId: 8) {
          productId
          productName
          unitsInStock
        }
      }",
      encoding: Encoding.UTF8,
      mediaType: "application/graphql");

    HttpResponseMessage response = await client.SendAsync(request);

    if (response.IsSuccessStatusCode)
    {
      ViewData["seafoodProducts"] = (await response.Content
        .ReadFromJsonAsync<GraphQLProducts>())?.Data?.Products;
    }
    else
    {
      ViewData["seafoodProducts"] = Enumerable.Empty<Product>().ToArray();
    }
  }
  catch (Exception ex)
  {
    _logger.LogWarning($"Northwind.GraphQL service exception: {ex.Message}");
  }
```

5. В папке Views/Home в файле `Services.cshtml` добавьте новый раздел внутри самого контейнера `<div>` после раздела `@if`, который отображает продукты, начинающиеся с `Cha`, из сервиса OData, чтобы показать продукты из категории `Seafood`:

```
@{
  ViewData["Title"] = "Services";
  Product[]? products = ViewData["productsCha"] as Product[];
  Product[]? seafoodProducts = ViewData["seafoodProducts"] as Product[];
}
...
@if (seafoodProducts is not null)
{
  <h2>Seafood products using GraphQL</h2>
  <p>
    @foreach (Product p in seafoodProducts)
    {
      <span class="badge badge-success">
        @p.ProductId
        @p.ProductName
        -
        @(p.UnitsInStock is null ? "0" : p.UnitsInStock.Value) in stock
      </span>
    }
  </p>
}
```

6. При необходимости запустите проект `Minimal.Web` без отладки.
7. При необходимости запустите проект `Northwind.OData` без отладки.
8. Запустите проект `Northwind.GraphQL` без отладки.
9. Запустите проект `Northwind.Mvc`.
10. Перейдите на страницу Services (Сервисы): https://localhost:5001/home/services.
11. Обратите внимание, что продукты из категории `Seafood` успешно извлекаются с помощью GraphQL (рис. 18.5).

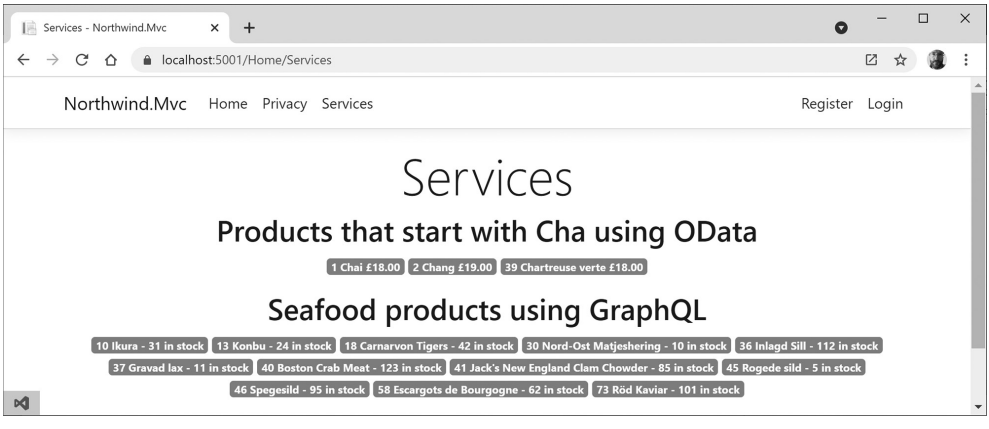

Рис. 18.5. Продукты в категории Seafood из сервиса GraphQL

12. Закройте браузер Google Chrome и завершите работу всех веб-серверов.

Реализация сервисов с помощью gRPC

gRPC — это современная высокопроизводительная RPC-платформа с открытым исходным кодом, которая может работать в любой среде.

gRPC

Клиент gRPC может вызывать методы в сервисе gRPC на другом сервере, как если бы это был локальный объект. Разработчик определяет интерфейс сервиса с методами, которые могут быть вызваны удаленно, включая их параметры и типы возврата. Сервер реализует этот интерфейс и запускает сервер gRPC для обработки клиентских вызовов. На стороне клиента сильно типизированный клиент gRPC предоставляет те же методы, что и на сервере.

Как и в случае с WCF, разработка API в gRPC построена на первичном определении контракта, что в данном случае позволяет избежать зависимости от языка

реализации. Вы определяете контракты в файлах `.proto`, которые обладают своим специфическим синтаксисом и затем используют инструменты для их преобразования в различные языки, такие как C#. Файлы `.proto` используются как сервером, так и клиентом для обмена сообщениями в правильном формате.

gRPC минимизирует использование сети, применяя двоичную сериализацию Protobuf, которая не читается человеком, в отличие от JSON или XML, используемых веб-сервисами. gRPC требует HTTP/2, который обеспечивает значительные преимущества в производительности по сравнению с предыдущими версиями, такие как двоичное кадрирование и сжатие, а также мультиплексирование вызовов HTTP/2 через одно соединение.

Основным ограничением gRPC является невозможность его использования в веб-браузерах, поскольку ни один браузер не обеспечивает уровень контроля, необходимый для поддержки клиента gRPC. Например, браузеры не позволяют вызывающей стороне требовать использования HTTP/2. Существует инициатива под названием *gRPC-Web*, которая добавляет дополнительный прокси-уровень, и прокси перенаправляет запросы на сервер gRPC.

Создание сервиса gRPC

Рассмотрим пример сервиса для управления поставщиками в базе данных Northwind.

1. Откройте редактор кода и создайте проект с такими настройками:
   1) шаблон проекта: ASP.NET Core gRPC Service/`grpc`;
   2) файл и папка рабочей области/решения: `PracticalApps`;
   3) файл и папка проекта: `Northwind.gRPC`.
2. В программе Visual Studio Code выберите `Northwind.gRPC` в качестве активного проекта OmniSharp.

Для работы с файлами .proto в программе Visual Studio Code необходимо расширение vscode-proto3 (zxh404.vscode-proto3).

3. В папке Protos откройте файл `greet.proto` и обратите внимание, что он определяет сервис `Greeter` с методом `SayHello`, который обменивается сообщениями `HelloRequest` и `HelloReply`:

```
syntax = "proto3";

option csharp_namespace = "Northwind.gRPC";

package greet;

// определение сервиса приветствия
service Greeter {
```

```
  // отправляем приветствие
  rpc SayHello (HelloRequest) returns (HelloReply);
}

// сообщение запроса, содержащее имя пользователя
message HelloRequest {
  string name = 1;
}

// ответное сообщение с приветствием
message HelloReply {
  string message = 1;
}
```

4. Откройте файл `Northwind.gRPC.csproj` и обратите внимание, что в нем содержится ссылка на пакет для реализации сервиса gRPC, размещенного в ASP.NET Core, а файл `.proto` зарегистрирован для использования на стороне сервера, как показано ниже:

```
<ItemGroup>
  <Protobuf Include="Protos\greet.proto" GrpcServices="Server" />
</ItemGroup>

<ItemGroup>
  <PackageReference Include="Grpc.AspNetCore" Version="2.32.0" />
</ItemGroup>
```

5. В папке Services откройте файл `GreeterService.cs` и обратите внимание, что в нем реализован контракт сервиса `Greeter`:

```
using Grpc.Core; // ServerCallContext
using Northwind.gRPC;

namespace Northwind.gRPC.Services;
public class GreeterService : Greeter.GreeterBase
{
  private readonly ILogger<GreeterService> _logger;

  public GreeterService(ILogger<GreeterService> logger)
  {
    _logger = logger;
  }

  public override Task<HelloReply> SayHello(
    HelloRequest request, ServerCallContext context)
  {
    return Task.FromResult(new HelloReply
    {
      Message = "Hello " + request.Name
    });
  }
}
```

6. В файле `Program.cs` в разделе конфигурации сервисов обратите внимание на вызов для добавления gRPC в коллекцию сервисов:

```
builder.Services.AddGrpc();
```

7. В файле `Program.cs` в разделе конфигурации HTTP-конвейера обратите внимание на вызов для сопоставления сервиса `Greeter`:

```
app.MapGrpcService<GreeterService>();
```

8. В файле `Program.cs` добавьте вызов метода расширения `UseUrls`, чтобы указать порт `5006` для HTTPS, как показано ниже:

```
var builder = WebApplication.CreateBuilder(args);

builder.WebHost.UseUrls("https://localhost:5006/");
```

9. В папке `Properties` откройте файл `launchSettings.json` и измените параметр `applicationUrl`, чтобы использовать порт `5006`:

```
{
  "profiles": {
    "Northwind.gRPC": {
      "commandName": "Project",
      "dotnetRunMessages": "true",
      "launchBrowser": false,
      "applicationUrl": "https://localhost:5006",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

10. Соберите проект `Northwind.gRPC`.

Создание клиента gRPC

Мы добавим пакеты клиента gRPC в проект сайта Northwind MVC, чтобы он мог вызывать сервис gRPC.

1. В проект `Northwind.Mvc` добавьте ссылки на пакеты для формата Google Protobuf, клиента gRPC и инструментов, как показано ниже:

```
<PackageReference Include="Google.Protobuf" Version="3.17.3" />
<PackageReference Include="Grpc.NET.Client" Version="2.38.0" />
<PackageReference Include="Grpc.Tools" Version="2.38.1">
  <PrivateAssets>all</PrivateAssets>
  <IncludeAssets>runtime; build; native; contentfiles;
    analyzers; buildtransitive</IncludeAssets>
</PackageReference>
```

Пакет Grpc.Tools используется только во время разработки, поэтому маркирован как PrivateAssets=all. Эта метка гарантирует, что инструментарий не будет опубликован на рабочем сайте.

2. Скопируйте папку Protos из проекта/папки `Northwind.gRPC` в проект/папку `Northwind.Mvc`. (В программе Visual Studio 2022 для копирования достаточно перетащить папку. В программе Visual Studio Code перетаскивайте папку, удерживая нажатой клавишу Ctrl или Cmd.)
3. В проекте `Northwind.Mvc` в файле `greet.proto` укажите пространство имен текущего проекта, чтобы автоматически создаваемые классы находились в том же пространстве имен:

```
option csharp_namespace = "Northwind.Mvc";
```

4. В файле проекта `Northwind.Mvc` добавьте группу элементов для регистрации файла `.proto` в качестве используемого на стороне клиента, как показано ниже:

```
<ItemGroup>
  <Protobuf Include="Protos\greet.proto" GrpcServices="Client" />
</ItemGroup>
```

Программа Visual Studio создаст для вас группу элементов, но по умолчанию для GrpcServices будет установлено значение Server, поэтому вы должны вручную изменить его на Client.

5. Соберите проект `Northwind.Mvc`, чтобы убедиться, что автоматически сгенерированные классы созданы.
6. В файле `HomeController.cs` импортируйте пространства имен для работы с gRPC в качестве клиента:

```
using Grpc.NET.Client; // GrpcChannel
```

7. В методе действия `Services` добавьте операторы для создания клиента gRPC и вызова метода `Greet`:

```
try
{
  using (GrpcChannel channel =
    GrpcChannel.ForAddress("https://localhost:5006"))
  {
    Greeter.GreeterClient greeter = new(channel);
    HelloReply reply = await greeter.SayHelloAsync(
      new HelloRequest { Name = "Henrietta" });
    ViewData["greeting"] = "Greeting from gRPC service: " + reply.Message;
  }
}
```

```
catch (Exception)
{
  _logger.LogWarning($"Northwind.gRPC service is not responding.");
}
```

8. В папке Views/Home в файле `Services.cshtml` добавьте код для отображения приветствия непосредственно под заголовком, перед отображением продуктов:

```
@if (ViewData["greeting"] != null)
{
  <p class="alert alert-primary">@ViewData["greeting"]</p>
}
```

Если вы очистите проект gRPC, то потеряете автоматически сгенерированные типы и увидите ошибки компиляции. Чтобы воссоздать их, внесите любые изменения в файл .proto или закройте и снова откройте проект/решение.

Тестирование клиента gRPC для сервиса gRPC

Теперь мы можем запустить сервис gRPC и проверить, может ли сайт Northwind MVC успешно вызвать его.

1. При необходимости запустите проект `Minimal.WebApi` без отладки.
2. При необходимости запустите проект `Northwind.OData` без отладки.
3. При необходимости запустите проект `Northwind.GraphQL` без отладки.
4. Запустите проект сервиса `Northwind.gRPC` без отладки.
5. Запустите проект `Northwind.Mvc`.
6. Перейдите на страницу Services (Сервисы): https://localhost:5001/home/services.
7. Обратите внимание на приветствие на странице сервисов (рис. 18.6).

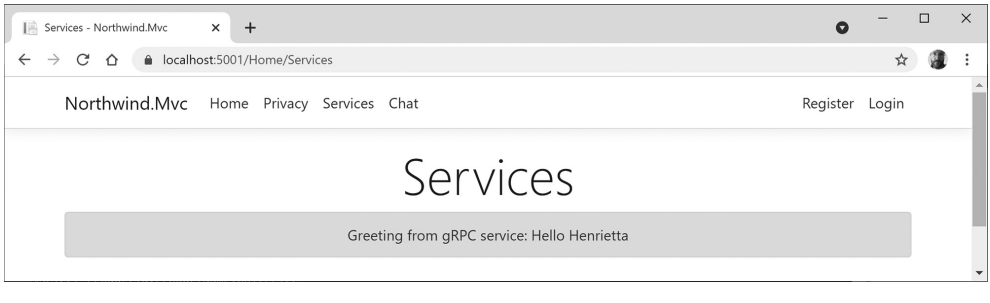

Рис. 18.6. Страница сервисов после вызова сервиса gRPC для получения приветствия

8. В командной строке или в терминале для сервиса gRPC обратите внимание на информационные сообщения, в которых сказано, что HTTP/2 `POST` был обработан конечной точкой `greet.Greeter/SayHello` примерно за 41 мс:

```
info: Microsoft.AspNetCore.Hosting.Diagnostics[1]
      Request starting HTTP/2 POST https://localhost:5006/greet.Greeter/
SayHello application/grpc -
info: Microsoft.AspNetCore.Routing.EndpointMiddleware[0]
      Executing endpoint 'gRPC - /greet.Greeter/SayHello'
info: Microsoft.AspNetCore.Routing.EndpointMiddleware[1]
      Executed endpoint 'gRPC - /greet.Greeter/SayHello'
info: Microsoft.AspNetCore.Hosting.Diagnostics[2]
      Request finished HTTP/2 POST https://localhost:5006/greet.Greeter/
SayHello application/grpc - - 200 - application/grpc 41.3434ms
```

9. Закройте браузер Google Chrome и завершите работу веб-сервера.

Реализация сервиса gRPC для модели EF Core

Теперь мы добавим в сервис gRPC поддержку работы с базой данных Northwind.

1. В проект `Northwind.gRPC` добавьте ссылку на проект контекста базы данных Northwind для SQLite или SQL Server:

```
<ItemGroup>
  <!-- при необходимости смените Sqlite на SqlServer -->
  <ProjectReference Include=
"..\Northwind.Common.DataContext.Sqlite\Northwind.Common.DataContext.
Sqlite.csproj" />
</ItemGroup>
```

2. В проекте `Northwind.gRPC` в папке Protos создайте файл `shipper.proto` (в программе Visual Studio шаблон элемента называется Protocol Buffer File):

```
syntax = "proto3";

option csharp_namespace = "Northwind.gRPC";

package shipr;

service Shipr {
  rpc GetShipper (ShipperRequest) returns (ShipperReply);
}

message ShipperRequest {
  int32 shipperId = 1;
}

message ShipperReply {
```

```
  int32 shipperId = 1;
  string companyName = 2;
  string phone = 3;
}
```

3. Откройте файл проекта и добавьте запись для включения файла `shipper.proto`:

```
<ItemGroup>
  <Protobuf Include="Protos\greet.proto" GrpcServices="Server" />
  <Protobuf Include="Protos\shipper.proto" GrpcServices="Server" />
</ItemGroup>
```

4. Соберите проект `Northwind.gRPC`.

5. В папке Services создайте файл класса `ShipperService.cs` и измените его содержимое, чтобы определить сервис доставки, который использует контекст базы данных Northwind для возврата грузоотправителей:

```
using Grpc.Core; // ServerCallContext
using Packt.Shared; // NorthwindContext, Shipper

namespace Northwind.gRPC.Services;

public class ShipperService : Shipr.ShiprBase
{
  private readonly ILogger<ShipperService> _logger;
  private readonly NorthwindContext db;

  public ShipperService(ILogger<ShipperService> logger,
    NorthwindContext db)
  {
    _logger = logger;
    this.db = db;
  }

  public override async Task<ShipperReply> GetShipper(
    ShipperRequest request, ServerCallContext context)
  {
    return ToShipperReply(
      await db.Shippers.FindAsync(request.ShipperId));
  }

  private ShipperReply ToShipperReply(Shipper? shipper)
  {
    return new ShipperReply
    {
      ShipperId = shipper?.ShipperId ?? 0,
      CompanyName = shipper?.CompanyName ?? string.Empty,
      Phone = shipper?.Phone ?? string.Empty
    };
  }
}
```

6. В файле `Program.cs` импортируйте пространство имен для контекста базы данных Northwind:

```
using Packt.Shared; // метод расширения AddNorthwindContext
```

7. В разделе конфигурации сервисов добавьте вызов для регистрации контекста базы данных Northwind:

```
builder.Services.AddNorthwindContext();
```

8. В разделе конфигурации HTTP-конвейера после вызова для регистрации сервиса `Greeter` добавьте оператор для регистрации сервиса доставки:

```
app.MapGrpcService<ShipperService>();
```

Реализация клиента gRPC для модели EF Core

Теперь мы можем добавить клиентские возможности на сайт Northwind MVC.

1. Скопируйте файл `shipper.proto` из папки Protos проекта `Northwind.gRPC` в папку Protos проекта `Northwind.MVC`. (Удерживайте нажатой клавишу Ctrl или Cmd при перетаскивании, если используете программу Visual Studio Code.)
2. В проекте `Northwind.Mvc` в файле `shipper.proto` измените пространство имен соответствующим образом, чтобы автоматически сгенерированные классы находились в том же пространстве имен:

```
option csharp_namespace = "Northwind.Mvc";
```

3. В файле проекта `Northwind.Mvc` добавьте запись для регистрации файла `.proto` как используемого на стороне клиента:

```
<ItemGroup>
  <Protobuf Include="Protos\greet.proto" GrpcServices="Client" />
  <Protobuf Include="Protos\shipper.proto" GrpcServices="Client" />
</ItemGroup>
```

4. В папке Controllers в файле `HomeController.cs` в методе действия `Services` добавьте операторы для вызова gRPC-сервиса Shipper:

```
try
{
  using (GrpcChannel channel =
    GrpcChannel.ForAddress("https://localhost:5006"))
  {
    Shipr.ShiprClient shipr = new(channel);

    ShipperReply reply = await shipr.GetShipperAsync(
      new ShipperRequest { ShipperId = 3 });

    ViewData["shipr"] = new Shipper
    {
```

```
      ShipperId = reply.ShipperId,
      CompanyName = reply.CompanyName,
      Phone = reply.Phone
    };
  }
}
catch (Exception)
{
  _logger.LogWarning($"Northwind.gRPC service is not responding.");
}
```

5. В папке Views/Home в файле `Services.cshtml` добавьте код для отображения информации о грузоотправителе после приветствия, как показано ниже:

```
@if (ViewData["shipr"] != null)
{
  Shipper? shipper = ViewData["shipr"] as Shipper;
  <p class="alert alert-danger">
    ShipperId: @shipper?.ShipperId, CompanyName: @shipper?.CompanyName,
Phone: @shipper?.Phone
  </p>
}
```

6. При необходимости запустите проект `Minimal.WebApi` без отладки.
7. При необходимости запустите проект `Northwind.OData` без отладки.
8. При необходимости запустите проект `Northwind.GraphQL` без отладки.
9. Запустите проект сервиса `Northwind.gRPC` без отладки.
10. Запустите проект `Northwind.Mvc`.
11. Перейдите на страницу Services (Сервисы): https://localhost:5001/home/services.
12. Обратите внимание на информацию о грузоотправителе на странице сервисов (рис. 18.7).
13. Закройте браузер Google Chrome и завершите работу веб-сервера.

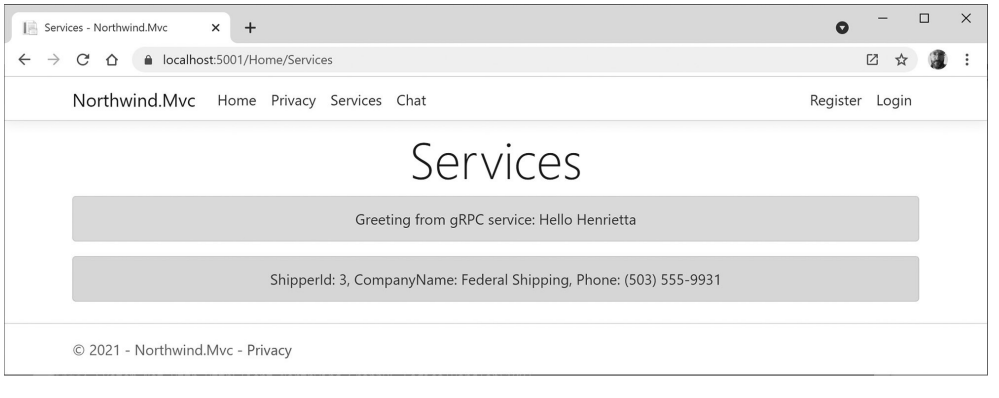

Рис. 18.7. Страница сервисов после вызова сервиса gRPC для получения грузоотправителя

Реализация взаимодействия в режиме реального времени с помощью SignalR

Интернет отлично подходит для создания сайтов и сервисов общего назначения, но он не был разработан для специализированных сценариев, требующих мгновенного обновления веб-страницы по мере поступления новой информации.

История взаимодействия в режиме реального времени в Интернете

Чтобы понять преимущества SignalR, необходимо знать историю HTTP и то, как организации работали над улучшением взаимодействия между клиентами и серверами в режиме реального времени.

На заре развития Интернета в 1990-х годах браузеры должны были отправлять полностраничный HTTP-запрос `GET` к веб-серверу, чтобы получить свежую информацию для показа посетителю.

XMLHttpRequest

В конце 1999 года Microsoft выпустила браузер Internet Explorer 5.0 с компонентом *XMLHttpRequest*, который мог выполнять асинхронные HTTP-запросы в фоновом режиме. Это, наряду с *динамическим HTML* (DHTML), позволяло плавно обновлять части веб-страницы при поступлении свежих данных.

Преимущества этой техники были очевидны, и вскоре все браузеры добавили такой же компонент.

AJAX

Компания Google максимально использовала данную возможность для создания умных веб-приложений, таких как Google Maps и Gmail. Несколько лет спустя эта техника стала известна как *асинхронный JavaScript и XML* (Asynchronous JavaScript and XML, AJAX).

Однако AJAX по-прежнему использует HTTP для обмена данными, и это имеет свои ограничения:

- во-первых, HTTP является протоколом взаимодействия «запрос — ответ»; это значит, сервер не может передавать данные клиенту. Он должен ждать, пока клиент сделает запрос;
- во-вторых, сообщения запроса и ответа HTTP содержат заголовки с большим количеством потенциально ненужных служебных данных;

- в-третьих, HTTP обычно требует создания нового базового TCP-соединения при каждом запросе.

WebSocket

WebSocket — полнодуплексная технология, то есть инициировать передачу новых данных может как клиент, так и сервер. WebSocket использует одно и то же TCP-соединение во время соединения. Она также более эффективна в плане размера отправляемых сообщений, поскольку минимальный размер фрейма составляет 2 байта.

WebSocket работает через HTTP-порты `80` и `443`, поэтому технология совместима с протоколом HTTP, а подтверждение WebSocket использует заголовок HTTP `Upgrade` для перехода от протокола HTTP к протоколу WebSocket.

Современные веб-приложения должны предоставлять актуальную информацию. Классическим примером является чат вживую, но существует множество других потенциальных приложений, от котировок акций до игр.

Всякий раз, когда серверу требуется передавать обновления на веб-страницу, вам нужна совместимая с Интернетом технология взаимодействия в режиме реального времени. Можно использовать WebSocket, но она поддерживается не всеми клиентами.

Знакомство с SignalR

ASP.NET Core SignalR — это библиотека с открытым исходным кодом, которая упрощает добавление веб-функций реального времени в приложения. Это своего рода абстракция над несколькими базовыми технологиями связи, что позволяет добавлять возможности взаимодействия в режиме реального времени с помощью сценариев на C#.

Разработчику не нужно понимать или реализовывать используемые базовые технологии, а SignalR будет автоматически переключаться между базовыми технологиями в зависимости от их поддержки браузером посетителя. Например, SignalR будет использовать WebSocket, когда эта технология доступна, а если нет, то плавно перейдет на другие, такие как длинный опрос AJAX, при этом код вашего приложения останется неизменным.

SignalR — это API для *удаленных вызовов процедур* (remote procedure calls, RPC) между сервером и клиентом. RPC вызывают функции JavaScript на клиентах из кода .NET на стороне сервера. SignalR имеет концентраторы для определения конвейера и автоматически обрабатывает отправку сообщений, используя два встроенных протокола концентраторов: JSON и бинарный протокол, основанный на MessagePack.

На стороне сервера SignalR работает везде, где поддерживается ASP.NET Core: серверы Windows, macOS или Linux. SignalR поддерживает следующие клиентские платформы:

- JavaScript-клиенты для современных браузеров, включая Chrome, Firefox, Safari, Edge и Internet Explorer 11;
- клиенты .NET, включая Blazor и Xamarin для мобильных приложений Android и iOS;
- Java 8 и более поздние версии.

Разработка сигнатур методов

При разработке сигнатур методов для сервиса SignalR лучше всего определять методы с одним параметром объекта, а не с несколькими параметрами простого типа. Например, определите класс с несколькими свойствами для использования в качестве типа одного параметра вместо передачи нескольких значений `string`:

```
// пример плохого варианта
public void SendMessage(string to, string body)

// пример лучшего решения
public class Message
{
  public string To { get; set; }
  public string Body { get; set; }
}

public void SendMessage(Message message)
```

Причина в том, что вы сможете вносить изменения в будущем, например добавить заголовок сообщения. В примере с плохим вариантом потребуется добавить третий строковый параметр `title`, и существующие клиенты будут получать ошибки, поскольку не отправляют дополнительное строковое значение. А использование примера лучшего решения не нарушит сигнатуру метода, поэтому существующие клиенты смогут продолжать вызывать его, как и до изменения. На стороне сервера дополнительное свойство `title` будет иметь нулевое значение, которое можно проверить и, возможно, установить в качестве значения по умолчанию.

Создание сервиса взаимодействия в реальном времени с помощью SignalR

Серверная библиотека SignalR включена в ASP.NET Core. Но клиентская библиотека JavaScript не включается в проект автоматически. Мы будем использовать инструмент *Library Manager CLI* для получения клиентской библиотеки из *unpkg*,

сети доставки контента (content delivery network, CDN), которая может доставлять все, что находится в менеджере пакетов Node.js.

Добавим серверный концентратор SignalR и клиентский JavaScript в проект Northwind MVC, чтобы реализовать функцию чата, позволяющую посетителям отправлять сообщения всем, кто в данный момент пользуется сайтом, динамически определяемым группам или одному указанному пользователю.

В рабочем решении было бы лучше разместить концентратор SignalR в отдельном веб-проекте, чтобы его можно было размещать и масштабировать независимо от остальной части сайта. Взаимодействие в реальном времени часто может создавать чрезмерную нагрузку на сайт.

Определение некоторых общих моделей

Сначала мы определим две общие модели, которые могут быть использованы как в серверном, так и в клиентском .NET-проектах, которые будут работать с нашим сервисом чата.

1. В проекте `Northwind.Common` добавьте файл класса `RegisterModel.cs` и измените его содержимое, чтобы определить модель для регистрации имени пользователя и групп, к которым он должен принадлежать:

```
namespace Northwind.Chat.Models;

public class RegisterModel
{
  public string? Username { get; set; }
  public string? Groups { get; set; }
}
```

2. В проекте `Northwind.Common` добавьте файл класса `MessageModel.cs` и измените его содержимое, чтобы определить модель сообщения со свойствами для тех, кому отправляется сообщение, и их тип (пользователь, группа или все), а также для тех, от кого было отправлено сообщение, и тела сообщения:

```
namespace Northwind.Chat.Models;

public class MessageModel
{
  public string? To { get; set; }
  public string? ToType { get; set; }
  public string? From { get; set; }
  public string? Body { get; set; }
}
```

Включение концентратора SignalR на стороне сервера

Далее мы включим концентратор SignalR на стороне сервера проекта Northwind MVC.

1. В проекте `Northwind.Mvc` добавьте ссылку на проект `Northwind.Common`, если вы не добавили ссылку на проект ранее.
2. В проекте `Northwind.Mvc` создайте папку `Hubs`.
3. В папке `Hubs` создайте файл класса `ChatHub.cs` и измените его содержимое так, чтобы он наследовался от класса `Hub` и реализовывал два метода, которые могут быть вызваны клиентом:

```
using Microsoft.AspNetCore.SignalR; // Hub
using Northwind.Chat.Models; // RegisterModel, MessageModel

namespace Northwind.Mvc.Hubs;

public class ChatHub : Hub
{
  // создаем новый экземпляр ChatHub для обработки каждого метода,
  // поэтому мы должны хранить имена пользователей и их идентификаторы
  // соединений в статическом поле
  private static Dictionary<string, string> users = new();

  public async Task Register(RegisterModel model)
  {
    // добавляем/обновляем словарь с именем пользователя и его connectionId
    users[model.Username] = Context.ConnectionId;

    foreach (string group in model.Groups.Split(','))
    {
      await Groups.AddToGroupAsync(Context.ConnectionId, group);
    }
  }

  public async Task SendMessage(MessageModel command)
  {
    MessageModel reply = new()
    {
      From = command.From,
      Body = command.Body
    };

    IClientProxy proxy;

    switch (command.ToType)
    {
      case "User":
        string connectionId = users[command.To];
        reply.To = $"{command.To} [{connectionId}]";
        proxy = Clients.Client(connectionId);
        break;
```

```
      case "Group":
        reply.To = $"Group: {command.To}";
        proxy = Clients.Group(command.To);
        break;

      default:
        reply.To = "Everyone";
        proxy = Clients.All;
        break;
    }

    await proxy.SendAsync(
      method: "ReceiveMessage", arg1: reply);
  }
}
```

Обратите внимание на следующие моменты:

- `ChatHub` имеет два метода, которые может вызвать клиент: `Register` и `SendMessage`;
- `Register` имеет один параметр типа `RegisterModel`. Имя пользователя и его идентификатор соединения хранятся в статическом словаре, чтобы имя пользователя можно было использовать для поиска идентификатора соединения и отправки сообщений непосредственно этому пользователю;
- `SendMessage` имеет один параметр типа `MessageModel`. Метод создает экземпляр класса `MessageModel` — сообщение, которое он отправляет одному или нескольким клиентам. Затем он действует в зависимости от типа получателя. Для пользователя он ищет идентификатор соединения по имени пользователя, а затем вызывает метод `Client`, чтобы получить прокси, который будет взаимодействовать только с этим клиентом. Для группы он вызывает метод `Group`, чтобы получить прокси, который будет взаимодействовать только с членами этой группы. Во всех остальных случаях он вызывает метод `All`, чтобы получить прокси, который будет взаимодействовать с каждым клиентом. Наконец, отправляет сообщение асинхронно, используя прокси.

4. В файле `Program.cs` импортируйте пространство имен для вашего концентратора SignalR:

```
using Northwind.Mvc.Hubs; // ChatHub
```

5. В разделе конфигурации сервисов добавьте оператор для добавления поддержки SignalR в коллекцию сервисов:

```
builder.Services.AddSignalR();
```

6. В разделе конфигурации HTTP-конвейера после вызова для отображения Razor Pages добавьте оператор для отображения относительного URL-пути/чата на ваш концентратор SignalR:

```
app.MapHub<ChatHub>("/chat");
```

Добавление клиентской JavaScript-библиотеки SignalR

Далее мы добавим клиентскую JavaScript-библиотеку SignalR, чтобы можно было использовать ее на веб-странице.

1. Откройте командную строку или терминал для проекта `Northwind.Mvc`.
2. Установите инструмент Library Manager CLI, как показано в следующей команде:

```
dotnet tool install -g Microsoft.Web.LibraryManager.Cli
```

Возможно, этот инструмент уже установлен. Чтобы обновить его до последней версии, повторите команду, но замените слово install (установить) на update (обновить).

3. Обратите внимание на сообщение об успешном завершении:

```
You can invoke the tool using the following command: libman
Tool 'microsoft.web.librarymanager.cli' (version '2.1.113') was
successfully installed.
```

4. Добавьте библиотеки `signalr.js` и `signalr.min.js` в проект из `unpkg`, как показано в следующей команде:

```
libman install @microsoft/signalr@latest -p unpkg -d wwwroot/js/signalr
--files dist/browser/signalr.js --files dist/browser/signalr.min.js
```

5. Обратите внимание на сообщение об успешном завершении:

```
Downloading file https://unpkg.com/@microsoft/signalr@latest/dist/browser/
signalr.js...
Downloading file https://unpkg.com/@microsoft/signalr@latest/dist/browser/
signalr.min.js...
wwwroot/js/signalr/dist/browser/signalr.js written to disk
wwwroot/js/signalr/dist/browser/signalr.min.js written to disk
Installed library "@microsoft/signalr@latest" to "wwwroot/js/signalr"
```

В программе Visual Studio есть графический интерфейс для добавления библиотек JavaScript на стороне клиента. Чтобы воспользоваться им, щелкните правой кнопкой мыши на веб-проекте и выберите команду меню Add ▸ Client Side Libraries (Добавить ▸ Библиотеки на стороне клиента).

Добавление страницы чата на сайт Northwind MVC

Далее мы создадим страницу чата.

1. В папке Controllers откройте файл `HomeController.cs` и добавьте новый метод действия для чата:

```
public IActionResult Chat()
{
  return View();
}
```

2. В папке Views/Shared в файле `_Layout.cshtml` добавьте новый элемент навигации после элемента навигации `Services`, который переходит на страницу чата, как показано ниже:

```
<li class="nav-item">
    <a class="nav-link text-dark" asp-area=""
      asp-controller="Home" asp-action="Chat">Chat</a>
</li>
```

3. В папке Views/Home добавьте новое пустое представление `Chat.cshtml` и измените его содержимое:

```
@{
  ViewData["Title"] = "Chat";
}
<div class="container">
  <h1>@ViewData["Title"]</h1>
  <div class="row">
    <div class="col-12">
      <h2>Register</h2>
    </div>
  </div>
  <div class="row">
    <div class="col-4">My name</div>
    <div class="col-8"><input type="text" id="from" /></div>
  </div>
  <div class="row">
    <div class="col-4">My groups</div>
    <div class="col-8"><input type="text" id="groups" value="Sales,IT"/>
    </div>
  </div>
  <div class="row">
    <div class="col-12">
      <input type="button" id="registerButton" value="Register" />
    </div>
  </div>
  <div class="row">
    <div class="col-12">
      <h2>Message</h2>
```

```
    </div>
  </div>
  <div class="row">
    <div class="col-4">To type</div>
    <div class="col-8">
      <select id="toType">
        <option selected>Everyone</option>
        <option>Group</option>
        <option>User</option>
      </select>
    </div>
  </div>
  <div class="row">
    <div class="col-4">To</div>
    <div class="col-8"><input type="text" id="to" /></div>
  </div>
  <div class="row">
  <div class="col-4">Body</div>
    <div class="col-8"><input type="text" id="body" /></div>
  </div>
  <div class="row">
    <div class="col-12">
      <input type="button" id="sendButton" value="Send" />
    </div>
  </div>
  <div class="row">
    <div class="col-12">
      <hr />
    </div>
  </div>
  <div class="row">
    <div class="col-12">
      <h2>Messages received</h2>
    </div>
  </div>
  <div class="row">
    <div class="col-12">
      <ul id="messages"></ul>
    </div>
  </div>
</div>
<script src="~/js/signalr/dist/browser/signalr.js"></script>
<script src="~/js/chat.js"></script>
```

Обратите внимание на следующие моменты:

- на странице есть три раздела: Register (Регистрация), Message (Сообщение) и Messages received (Полученные сообщения);
- раздел Register (Регистрация) содержит два ввода для имени посетителя и разделенного запятыми списка групп, членом которых он хочет быть, а также кнопку, которую нужно нажать для регистрации;

- раздел Message (Сообщение) содержит три ввода для типа получателя, имени получателя и тела сообщения, а также кнопку, которую нужно нажать для отправки сообщения;
- раздел Messages received (Полученные сообщения) содержит элемент неупорядоченного списка, который будет динамически заполняться элементами списка при получении сообщения;
- за двумя элементами сценария для клиентской библиотеки SignalR следует реализация клиента чата.

4. В каталоге wwwroot/js создайте JavaScript-файл `chat.js` и измените его содержимое:

```
"use strict";

var connection = new signalR.HubConnectionBuilder()
  .withUrl("/chat").build();

document.getElementById("registerButton").disabled = true;
document.getElementById("sendButton").disabled = true;

connection.start().then(function () {
  document.getElementById("registerButton").disabled = false;
  document.getElementById("sendButton").disabled = false;
}).catch(function (err) {
  return console.error(err.toString());
});

connection.on("ReceiveMessage", function (received) {
  var li = document.createElement("li");
  document.getElementById("messages").appendChild(li);
  // обратите внимание на использование обратной кавычки `,
  // чтобы включить форматированную строку
  li.textContent =
    `${received.from} says ${received.body} (sent to ${received.to})`;
});

document.getElementById("registerButton").addEventListener("click",
  function (event) {
  var registermodel = {
    username: document.getElementById("from").value,
    groups: document.getElementById("groups").value
  };
  connection.invoke("Register", registermodel).catch(function (err) {
    return console.error(err.toString());
  });
  event.preventDefault();
});

document.getElementById("sendButton").addEventListener("click",
```

```
  function (event) {
  var messageToSend = {
    to: document.getElementById("to").value,
    toType: document.getElementById("toType").value,
    from: document.getElementById("from").value,
    body: document.getElementById("body").value
  };
  connection.invoke("SendMessage", messageToSend).catch(function (err) {
    return console.error(err.toString());
  });
  event.preventDefault();
});
```

Обратите внимание на следующие моменты:

- сценарий создает конструктор соединения с концентратором SignalR, указывающий на относительный путь URL к концентратору чата на сервере/в чате;
- сценарий отключает кнопки Register (Регистрация) и Send (Отправить) до тех пор, пока соединение с концентратором на стороне сервера не будет успешно установлено;
- когда соединение получает вызов `ReceiveMessage` от концентратора на стороне сервера, оно добавляет элемент списка в неупорядоченный список `messages`. Содержимое элемента списка содержит подробную информацию о сообщении, такую как `from` (от кого), `to` (кому) и `body` (тело сообщения). Обратите внимание, что JavaScript использует верблюжий регистр;
- к кнопке Register (Регистрация) добавляется обработчик события нажатия кнопки мыши, который создает модель регистрации с именем пользователя и его группами, а затем вызывает метод `Register` на стороне сервера;
- обработчик события нажатия кнопки мыши добавляется к кнопке Send (Отправить), которая создает модель сообщения с параметрами `from`, `to`, `type` и `body`, а затем вызывает метод `SendMessage` на стороне сервера.

Тестирование функции чата

Теперь мы готовы попробовать отправить сообщения в чате между несколькими посетителями сайта.

1. Запустите сайт проекта `Northwind.Mvc`.
2. Запустите браузер Google Chrome.
3. Перейдите по адресу https://localhost:5001/home/chat.
4. Введите значения `Alice` для имени, `Sales, IT` для групп, а затем нажмите кнопку Register (Регистрация).
5. Откройте новое окно браузера или запустите другой браузер, например Firefox или Edge.

6. Перейдите по адресу https://localhost:5001/home/chat.
7. Введите значения `Bob` для имени, `Sales` для групп, а затем нажмите кнопку Register (Регистрация).
8. Откройте новое окно браузера или запустите другой браузер, например Firefox или Edge.
9. Перейдите по адресу https://localhost:5001/home/chat.
10. Введите значения `Charlie` для имени, `IT` для групп, а затем нажмите кнопку Register (Регистрация).
11. Расположите окна браузера так, чтобы вы могли видеть все три одновременно.
12. В браузере Алисы выберите Group (Группа), введите `Sales`, введите `Sell more!` и нажмите кнопку Send (Отправить).
13. Обратите внимание, что Алиса и Боб получили сообщение (рис. 18.8).

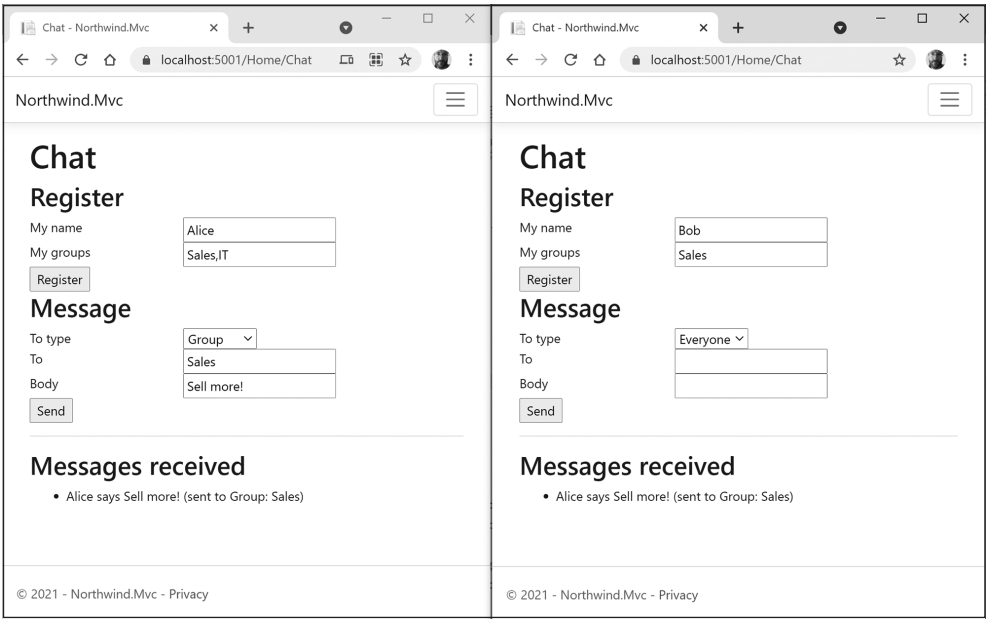

Рис. 18.8. Алиса отправляет сообщение группе Sales (Продажи)

14. В браузере Боба выберите Group (Группа), введите `IT`, введите `Fix more bugs!` и нажмите кнопку Send (Отправить).
15. Обратите внимание, что Алиса и Чарли получили сообщение (рис. 18.9).
16. В браузере Алисы выберите User (Пользователь), введите `Bob`, введите `Bonjour Bob!` и нажмите кнопку Send (Отправить).
17. Обратите внимание, что сообщение получает только Боб (рис. 18.10).

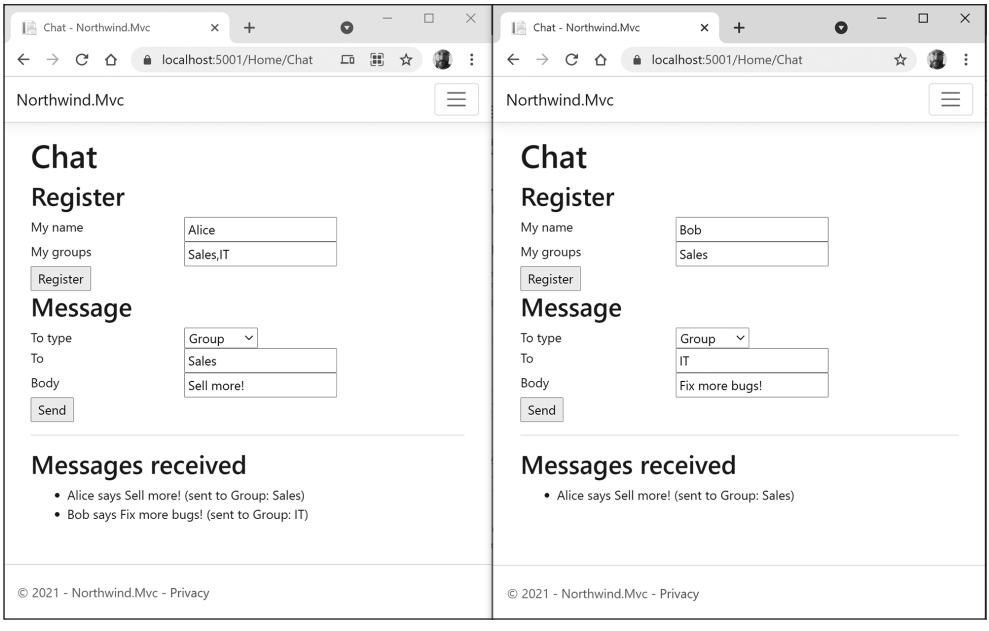

Рис. 18.9. Боб отправляет сообщение группе IT

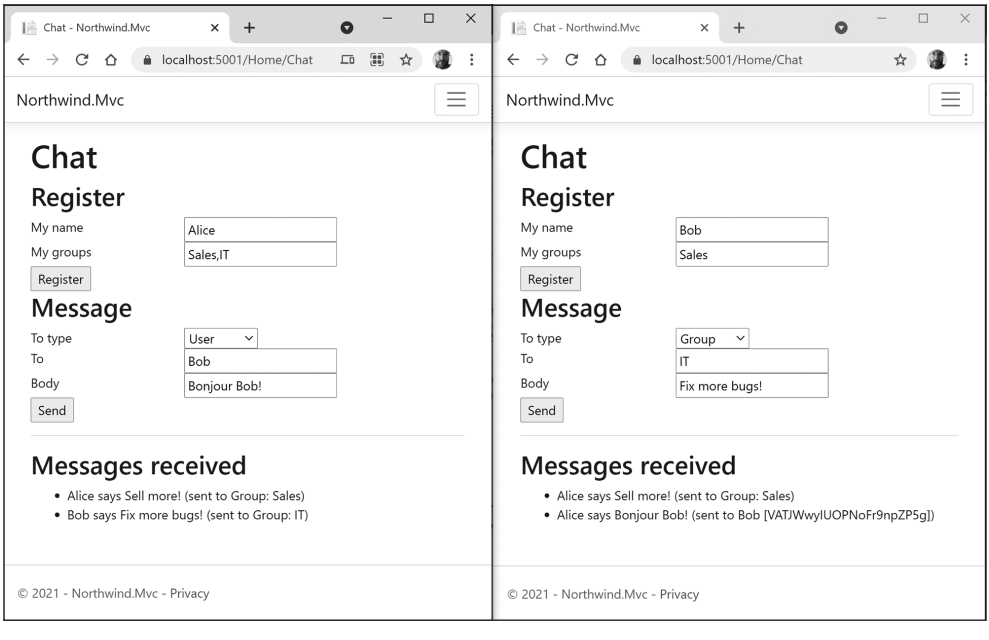

Рис. 18.10. Алиса отправляет сообщение Бобу

18. В браузере Чарли оставьте для параметра To type (Кому) значение Everyone (Все), для параметра To (Кому) оставьте значение пустым, введите любое сообщение, а затем нажмите кнопку Send (Отправить) и обратите внимание, что все получили сообщение.
19. Закройте браузеры и завершите работу веб-сервера.

Создание консольного приложения клиента чата

Теперь создадим клиент .NET для SignalR. Мы будем использовать консольное приложение, хотя для любого типа проекта .NET потребуется одна и та же ссылка на пакет и код реализации.

1. Откройте редактор кода и создайте проект с такими настройками:
   1) шаблон проекта: Console Application/`console`;
   2) файл и папка рабочей области/решения: `PracticalApps`;
   3) файл и папка проекта: `Northwind.SignalR.ConsoleClient`.
2. Добавьте ссылку на пакет для клиента ASP.NET Core SignalR и ссылку на проект для `Northwind.Common`:

```
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.SignalR.Client"
                    Version="6.0.0" />
</ItemGroup>
<ItemGroup>
  <ProjectReference
    Include="..\Northwind.Common\Northwind.Common.csproj" />
</ItemGroup>
```

3. В файле `Program.cs` импортируйте пространства имен для работы с SignalR в качестве клиента и модели чата, а затем добавьте операторы для создания соединения с концентратором, предложите пользователю ввести имя пользователя и группы для регистрации, а затем прослушивайте полученные сообщения:

```
using Microsoft.AspNetCore.SignalR.Client; // HubConnection
using Northwind.Chat.Models; // RegisterModel, MessageModel

using static System.Console;

Write("Enter a username: ");
string? username = ReadLine();
```

```
Write("Enter your groups: ");
string? groups = ReadLine();

HubConnection hubConnection = new HubConnectionBuilder()
  .WithUrl("https://localhost:5001/chat")
  .Build();

hubConnection.On<MessageModel>("ReceiveMessage", message =>
{
  WriteLine($"{message.From} says {message.Body} (sent to {message.To})");
});

await hubConnection.StartAsync();

WriteLine("Successfully started.");

RegisterModel registration = new()
{
  Username = username,
  Groups = groups
};

await hubConnection.InvokeAsync("Register", registration);

WriteLine("Successfully registered.");
WriteLine("Listening... (press ENTER to stop.)");
ReadLine();
```

4. Запустите сайт проекта `Northwind.Mvc` без отладки.
5. Запустите браузер Google Chrome.
6. Перейдите по адресу https://localhost:5001/home/chat.
7. Введите значения `Alice` для имени, `Sales, IT` для групп, а затем нажмите кнопку Register (Регистрация).
8. Запустите проект `Northwind.SignalR.ConsoleClient`, а затем введите свое имя и группы `Sales,Admins`.
9. Расположите окна браузера и консольного приложения так, чтобы вы могли видеть оба одновременно.
10. В браузере Алисы в разделе Message (Сообщения) выберите пункт Group (Группа), введите `Sales`, затем `Go team!`, нажмите кнопку Send (Отправить) и обратите внимание, что Алиса и вы получили сообщение.
11. Попробуйте отправить сообщения только себе, только членам группы `Admins` и всем, как показано на рис. 18.11.
12. В консольном приложении нажмите клавишу Enter, чтобы остановить его.
13. Закройте браузер Google Chrome и завершите работу веб-сервера.

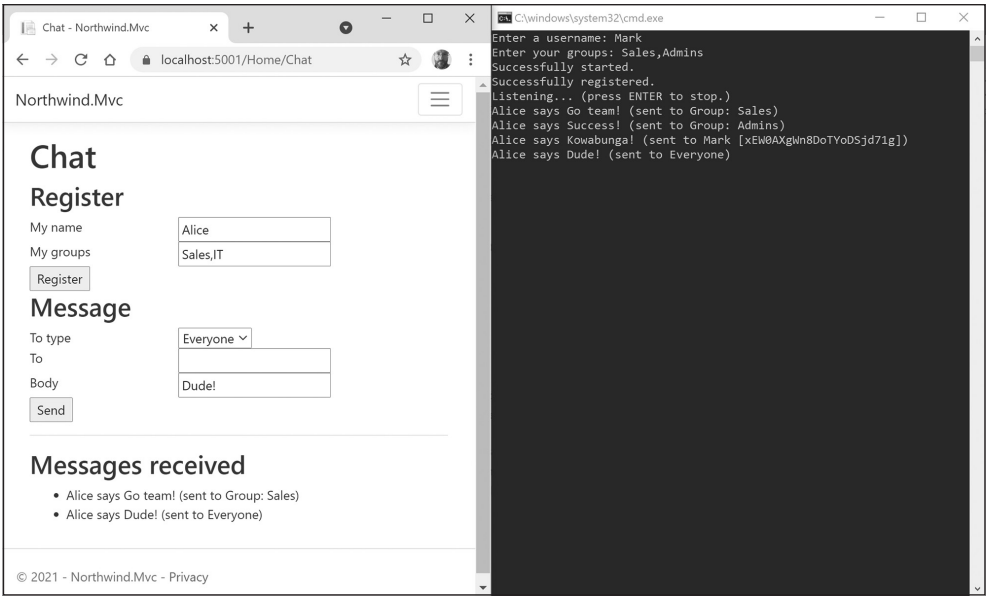

Рис. 18.11. Алиса отправляет сообщения различным типам получателей

Реализация бессерверных сервисов с помощью Azure Functions

Azure Functions — это управляемая событиями платформа для бессерверных вычислений. Вы можете создавать и отлаживать их локально, а затем развертывать в облаке Microsoft Azure. Платформу можно реализовать на многих языках, а не только на C# и .NET. Она имеет расширения для Visual Studio и Visual Studio Code.

Зачем нужно создавать сервис без сервера? «Бессерверный» не означает буквально «без сервера». «Бессерверный» означает отсутствие постоянно работающего сервера, как правило, бóльшую часть времени.

Например, в организациях часто есть бизнес-функции, которые необходимо выполнять только раз в месяц или на разовой основе. Возможно, организация печатает чеки для выплаты зарплаты своим сотрудникам в конце месяца. Для этих чеков может потребоваться конвертация сумм заработной платы в слова для печати на чеке. Функция для преобразования чисел в слова может быть реализована как бессерверный сервис.

Платформа Azure Functions — это нечто большее, чем просто отдельные функции. С помощью *Durable Functions* поддерживаются сложные рабочие процессы с отслеживанием состояния и решения, управляемые событиями. Мы не будем

рассматривать эти функции в данной книге, поэтому, если вам интересно, вы можете узнать о них больше здесь: https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp.

Модель Azure Functions

Платформа Azure Functions имеет модель программирования, основанную на триггерах и связях, которые позволяют вашим бессерверным приложениям реагировать на события и подключаться к другим сервисам, например к хранилищам данных.

Триггеры и привязки Azure Functions

Триггеры и привязки — ключевые понятия Azure Functions. Триггеры — это то, что заставляет функцию выполняться. Каждая функция должна иметь один и только один триггер. Наиболее распространенные триггеры представлены ниже:

- *HTTP* — реагирует на входящий HTTP-запрос;
- *Queue* — реагирует на поступление в очередь сообщения, готового к обработке;
- *Timer* — реагирует на наступление определенного времени;
- *Event Grid* — реагирует на наступление предопределенного события.

Привязки позволяют функциям иметь входы и выходы. Каждая функция может иметь ноль, одну или несколько привязок. Вот некоторые распространенные привязки:

- *Blob storage* — чтение или запись в любой файл, хранящийся в виде *двоичного большого объекта* (binary large object, BLOB);
- *Cosmos DB* — чтение или запись документов в облачное хранилище данных;
- *SignalR* — получение или выполнение удаленных вызовов методов;
- *Queue* — запись сообщения в очередь;
- *SendGrid* — отправка сообщения электронной почты;
- *Twilio* — отправка СМС;
- *IoT Hub* — запись на подключенное к Интернету устройство.

С полным списком поддерживаемых привязок вы можете ознакомиться здесь: https://docs.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings?tabs=csharp#supported-bindings.

Триггеры и привязки настраиваются по-разному для разных языков. Для C# и Java вы дополняете методы и параметры атрибутами. Для других языков вы настраиваете файл `function.json`.

Версии и языки Azure Functions

Azure Functions поддерживает четыре версии узлов выполнения и несколько языков (табл. 18.5).

Таблица 18.5. Версии узлов и языки, поддерживаемые Azure Functions

Язык (-и)	Версия 1 (v1)	Версия 2 (v2)	Версия 3 (v3)	Версия 4 (v4)
C#, F#	.NET Framework 4.8	.NET Core 2.1	.NET Core 3.1, .NET 5.0[2]	.NET 6.0[2]
JavaScript[1]	Node 6	Node 8, 10	Node 10, 12, 14	—
Java	—	Java 8	Java 8, 11	—
PowerShell	—	PowerShell Core 6	PowerShell Core 6, 7	—
Python	—	Python 3.6, 3.7	Python 3.6, 3.7, 3.8, 3.9	—

В этой книге мы рассмотрим только реализацию Azure Functions с помощью C# и .NET.

Модели хостинга Azure Functions

У Azure Functions есть две модели хостинга: внутрипроцессный и изолированный.

- *Внутрипроцессный* — ваша функция реализуется в библиотеке классов, которая запускается в том же процессе, что и хост. Ваши функции должны работать на той же версии .NET, что и среда выполнения Azure Functions.
- *Изолированный* — ваша функция реализована в консольном приложении, которое запускается в собственном процессе. Поэтому ваша функция может выполняться на текущих выпусках, таких как .NET 5.0, которые не поддерживаются средой выполнения Azure Functions, разрешающей только LTS-выпуски в процессе.

Azure Functions изначально поддерживает только одну LTS-версию .NET. Например, для Azure Functions v3 ваша функция должна использовать .NET Core 3.1 внутрипроцессно. Для Azure Functions v4 ваша функция должна использовать .NET 6.0 внутрипроцессно. Если вы создаете изолированную функцию, то можете выбрать любую версию .NET.

---

1 Azure Functions поддерживает язык TypeScript путем компиляции в код на другом языке программирования (преобразования/компиляции) в JavaScript.

2 .NET 5.0 поддерживается только в модели изолированного хостинга, поскольку это текущая версия. Платформа .NET 6.0 поддерживается как в изолированном, так и в текущем режиме, поскольку это выпуск с долгосрочной поддержкой.

Настройка локальной среды разработки для Azure Functions

Для начала, вам нужно установить последнюю версию *Azure Functions Core Tools*, которой на момент написания этой книги является v4, по следующей ссылке: https://www.npmjs.com/package/azure-functions-core-tools.

Azure Functions Core Tools (Основные инструменты от Azure Functions) предоставляет основную среду выполнения и шаблоны для создания функций, которые позволяют осуществлять локальную разработку на Windows, macOS и Linux с использованием любого редактора кода.

Azure Functions Core Tools включен в рабочую нагрузку разработки Azure в программе Visual Studio 2022, поэтому, возможно, он у вас уже установлен.

Создание проекта Azure Functions для локального запуска

Теперь мы можем создать проект Azure Functions. Хотя их можно создать в облаке с помощью портала Azure, разработчикам удобнее создавать и запускать их локально. После тестирования функции на собственном компьютере ее можно развернуть в облаке.

Каждый редактор кода имеет немного разные возможности для начала работы с проектом Azure Functions.

В программе Visual Studio 2022

Если вы предпочитаете Visual Studio, то создать проект Azure Functions можно следующим образом.

1. Откройте редактор кода и создайте проект с такими настройками:
   1) шаблон проекта: Azure Functions;
   2) файл и папка рабочей области/решения: `PracticalApps`;
   3) файл и папка проекта: `Northwind.AzureFuncs`.
2. В программе Visual Studio выберите `.NET 6 (Isolated)` (.NET 6 (Изолированный)), Http trigger (Триггер Http), Storage emulator (Эмулятор хранилища), а для Authorization level (Уровень авторизации) выберите Anonymous (Анонимный), затем нажмите Create (Создать) (рис. 18.12).

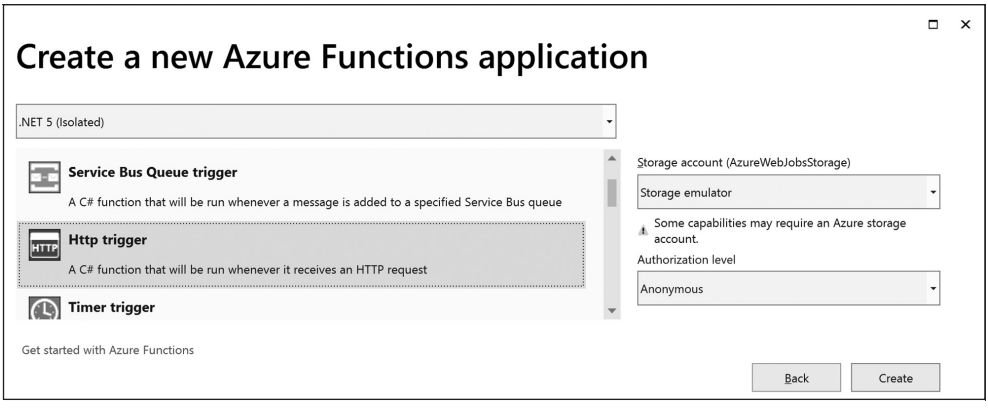

Рис. 18.12. Выбор параметров для проекта Azure Functions в программе Visual Studio 2022

В программе Visual Studio Code

Если вы предпочитаете Visual Studio Code, то создать проект Azure Functions можно следующим образом.

1. В программе Visual Studio Code перейдите в раздел Extensions (Расширения) и найдите расширение для Azure Functions (`ms-azuretools.vscode-azurefunctions`). Оно имеет зависимости от двух других расширений: Azure Account (`ms-vscode.azure-account`) и Azure Resources (`ms-azuretools.vscode-azureresourcegroups`), поэтому они тоже будут установлены.
2. В папке PracticalApps создайте папку Northwind.AzureFuncs и добавьте ее в рабочую область `PracticalApps`.
3. Закройте рабочую область `PracticalApps`, а затем откройте папку Northwind.AzureFuncs. (Следующие шаги допустимы только за пределами рабочей области.)
4. В расширении Azure в разделе FUNCTIONS (Функции) нажмите кнопку Create new project (Создать новый проект), а затем выберите папку Northwind.AzureFuncs (рис. 18.13).

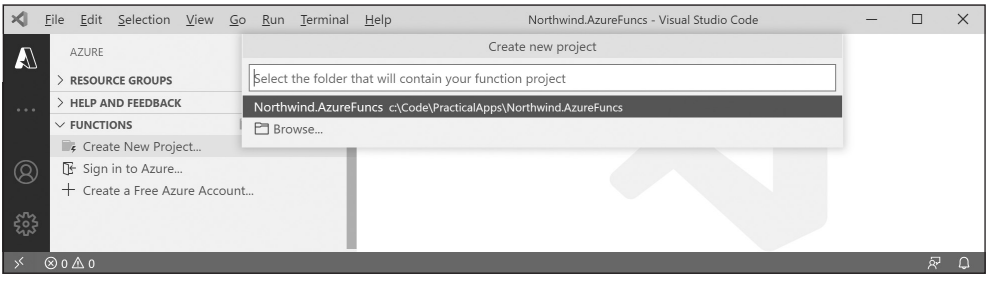

Рис. 18.13. Выбор папки для проекта Azure Functions

5. В подсказках сделайте следующее:
   1) выберите язык для проекта функций: C#;
   2) выберите .NET 6 LTS в качестве среды выполнения .NET, к сожалению не показанной на рис. 18.14, поскольку на момент написания этой книги она еще не была выпущена;

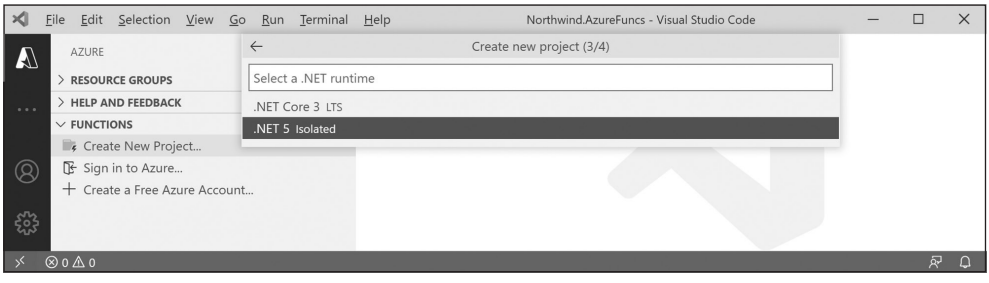

Рис. 18.14. Выбор целевой среды выполнения .NET для проекта Azure Functions

   3) выберите шаблон для первой функции проекта: HTTP trigger;
   4) укажите имя функции: `NumbersToWordsFunction`;
   5) укажите пространство имен: `Northwind.AzureFuncs`;
   6) выберите уровень авторизации: Anonymous (Анонимный).
6. В меню File (Файл) программы Visual Studio Code закройте папку.
7. Откройте рабочую область `PracticalApps`.

С помощью func CLI

Если вы предпочитаете командную строку и какой-либо другой редактор кода, то создать проект Azure Functions можно следующим образом.

1. В папке PracticalApps создайте папку Northwind.AzureFuncs и добавьте ее в рабочую область `PracticalApps`.
2. В командной строке или в терминале в папке Northwind.AzureFuncs создайте проект Azure Functions с помощью C#, как показано в следующей команде:

```
func init --csharp
```

3. В командной строке или в терминале в папке Northwind.AzureFuncs создайте функцию Azure Functions с помощью HTTP-триггера, которую можно вызывать анонимно:

```
func new --name NumbersToWordsFunction --template "HTTP trigger"
--authlevel "anonymous"
```

4. При необходимости вы можете запустить функцию локально, как показано в следующей команде:

```
func start
```

Обзор проекта

Прежде чем писать функцию, мы рассмотрим, из чего состоит проект Azure Functions.

1. Откройте файл проекта и обратите внимание на версию Azure Functions и ссылки на пакеты, необходимые для реализации Azure Function, отвечающей на HTTP-запросы:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <AzureFunctionsVersion>v4</AzureFunctionsVersion>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.NET.Sdk.Functions"
      Version="3.0.13" />
  </ItemGroup>
  <ItemGroup>
    <None Update="host.json">
      <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
    </None>
    <None Update="local.settings.json">
      <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
      <CopyToPublishDirectory>Never</CopyToPublishDirectory>
    </None>
  </ItemGroup>
</Project>
```

2. Откройте файл `local.settings.json` и обратите внимание, что во время локальной разработки ваш проект будет использовать локальное хранилище разработки и изолированный процесс:

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "UseDevelopmentStorage=true",
    "FUNCTIONS_WORKER_RUNTIME": "dotnet"
  }
}
```

Реализация функции

Теперь мы можем реализовать функцию для преобразования чисел в слова.

1. Если вы выполнили упражнение из главы 8 по написанию функции, преобразующей числа в слова, то используйте свою реализацию. Если нет, то используйте

класс, доступный по следующей ссылке: https://github.com/markjprice/cs10dotnet6/blob/master/vscode/PracticalApps/Northwind.AzureFuncs/NumbersToWords.cs.

2. Если вы используете Visual Studio, то в проекте `Northwind.AzureFuncs` щелкните правой кнопкой мыши на файле `Function1.cs` и переименуйте его в `NumbersToWordsFunction.cs`.

3. Откройте файл `NumbersToWordsFunction.cs` и измените его содержимое, чтобы реализовать Azure Function для преобразования суммы в виде числа в слова:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs; // [FunctionName], [HttpTrigger]
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using System.Numerics; // BigInteger
using Packt.Shared; // метод расширения ToWords
using System.Threading.Tasks; // Task

namespace Northwind.AzureFuncs;

public static class NumbersToWordsFunction
{
  [FunctionName(nameof(NumbersToWordsFunction))]
  public static async Task<IActionResult> Run(
    [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post")]
    HttpRequest req, ILogger log)
  {
    log.LogInformation($"C# HTTP trigger function processed a request.");

    string amount = req.Query["amount"];

    if (BigInteger.TryParse(amount, out BigInteger number))
    {
      return new OkObjectResult(number.ToWords());
    }
    else
    {
      return new BadRequestObjectResult($"Failed to parse: {amount}");
    }
  }
}
```

Тестирование функции

Теперь мы можем протестировать эту функцию.

1. Запустите проект `Northwind.AzureFuncs`. Если вы используете Visual Studio Code, то вам нужно перейти на панель Run and Debug (Запуск и отладка), убе-

диться, что выбрана опция Attach to .NET Functions (Подключиться к функциям .NET), а затем нажать кнопку Run (Выполнить).

2. Обратите внимание, что запускается эмулятор Azure Storage.
3. В Windows, если вы видите предупреждение о безопасности Windows от брандмауэра Windows Defender, нажмите Allow access (Разрешить доступ).
4. Обратите внимание, что Azure Functions Core Tools размещает вашу функцию обычно на порте 7071:

```
Azure Functions Core Tools
Core Tools Version:       4.0.3743 Commit hash:
44e84987044afc45f0390191bd5d70680a1c544e (64-bit)
Function Runtime Version: 4.0.16281

Functions:
        NumbersToWordsFunction: [GET,POST] http://localhost:7071/api/
NumbersToWordsFunction

For detailed output, run func with --verbose flag.
[2021-09-12T18:44:47.499Z] Worker process started and initialized.
[2021-09-12T18:44:51.038Z] Host lock lease acquired by instance ID '000000
0000000000000000011150C3D'.
```

5. Выберите URL вашей функции и скопируйте его в буфер обмена.
6. Запустите браузер Google Chrome.
7. Вставьте URL в адресное поле, добавьте строку запроса: ?amount=123456 и обратите внимание на успешный ответ (рис. 18.15).

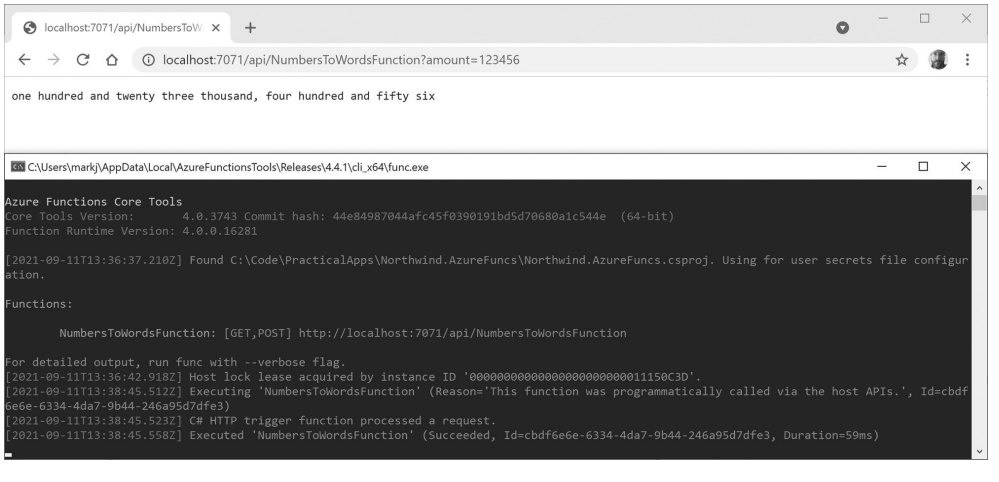

Рис. 18.15. Успешный вызов локально запущенной функции Azure Function

8. В командной строке или терминале обратите внимание, что функция была вызвана успешно, как показано ниже:

```
[2021-09-14T05:58:27.357Z] Executing 'Functions.NumbersToWordsFunction'
(Reason='This function was programmatically called via the host APIs.',
Id=c2c98c67-bf9f-4121-8f7b-701dbc9c0bad)
[2021-09-14T05:58:27.417Z] C# HTTP trigger function processed a request.
[2021-09-14T05:58:27.461Z] Executed 'Functions.NumbersToWordsFunction'
(Succeeded, Id=c2c98c67-bf9f-4121-8f7b-701dbc9c0bad, Duration=111ms)
```

9. Попробуйте вызвать функцию без суммы в строке запроса или с нецелым значением для суммы и обратите внимание, что функция возвращает код состояния `400`, указывающий на плохой запрос (рис. 18.16).

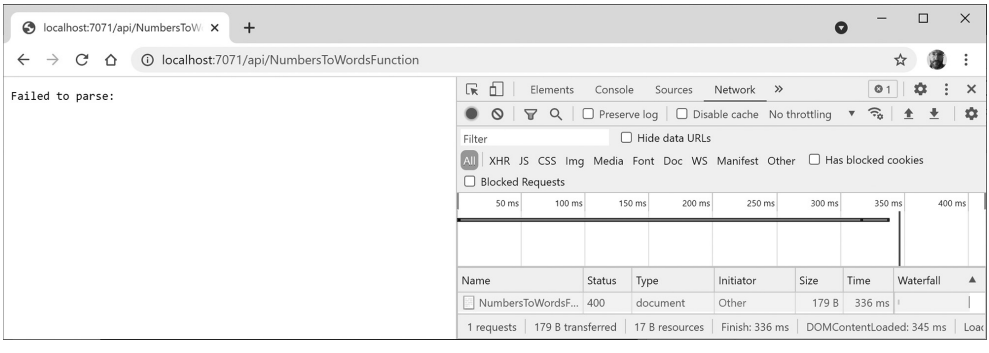

Рис. 18.16. Неверный запрос к локально запущенной функции Azure Function

10. Закройте браузер Google Chrome и завершите работу веб-сервера (или в программе Visual Studio Code остановите отладку).

Публикация проекта Azure Functions в облако

Теперь создадим приложение функции и связанные с ним ресурсы в подписке Azure, затем развернем вашу функцию в облаке и запустим ее там.

Если у вас еще нет учетной записи Azure, то вы можете бесплатно зарегистрироваться по следующей ссылке: https://azure.microsoft.com/en-us/free/.

В программе Visual Studio 2022

Visual Studio имеет графический пользовательский интерфейс для публикации в Azure.

1. На панели Solution Explorer (Проводник решений) щелкните правой кнопкой мыши на проекте `Northwind.AzureFuncs` и выберите команду меню Publish (Опубликовать).

2. Выберите Azure и нажмите кнопку Next (Далее).
3. Выберите Azure Function App (Windows) (Приложение Azure Function (Windows)) и нажмите кнопку Next (Далее).
4. Войдите в систему и введите свои учетные данные.
5. Выберите подписку.
6. В разделе Function Instance (Экземпляр функции) нажмите кнопку +, на которой есть всплывающая подсказка с надписью Create a new Azure Function (Создать новую функцию Azure).
7. Заполните диалоговое окно следующим образом (рис. 18.17):
   1) Name (Имя) — оно должно быть уникальным во всем мире;
   2) Subscription name (Имя подписки) — ваша подписка;
   3) Resource group (Группа ресурсов) — создайте группу ресурсов, чтобы потом упростить последующее удаление всего. Я ввел `cs10dotnet6projects`;
   4) Plan Type: Consumption (Тип плана: Потребление) (платите только за то, что используете);

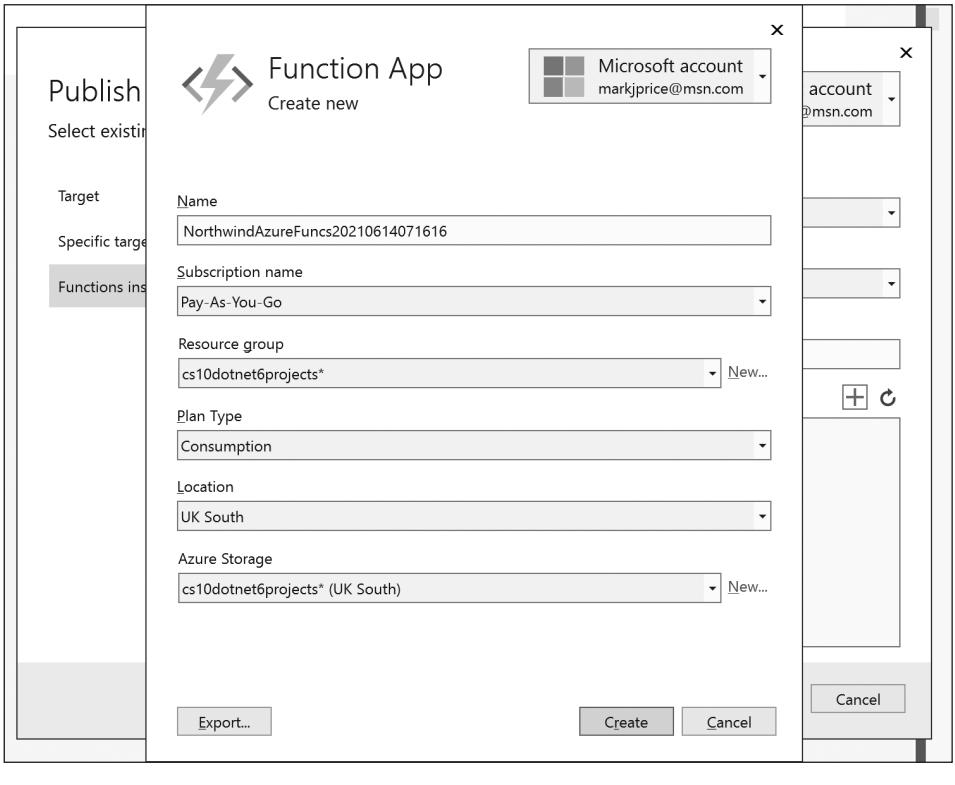

Рис. 18.17. Создание приложения Azure Function

5) Location (Местоположение) — ближайший к вам центр обработки данных. Я выбрал UK South (Юг Великобритании);
6) Azure Storage (Хранилище Azure) — создайте учетную запись `cs10dotnet6projects` (или другую, уникальную в мировом масштабе — попробуйте добавить свои инициалы) в ближайшем к вам центре обработки данных и выберите тип учетной записи Standard - Locally Redundant Storage (Стандартный — локально резервируемое хранилище).

8. Нажмите кнопку Create (Создать). Этот процесс может занять минуту или больше.
9. В диалоговом окне Publish (Публикация) нажмите Finish (Готово).
10. В окне Publish (Публикация) нажмите кнопку Publish (Опубликовать) (рис. 18.18).

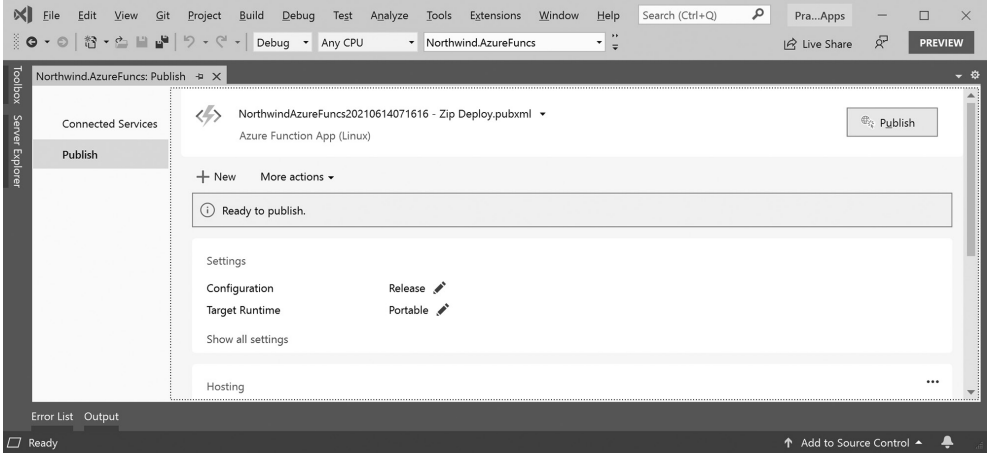

Рис. 18.18. Приложение Azure Function готово к публикации

11. Проанализируйте окно вывода:

```
Build started...
2>------ Publish started: Project: Northwind.AzureFuncs, Configuration:
Release Any CPU ------
2>Northwind.AzureFuncs -> C:\Code\PracticalApps\Northwind.AzureFuncs\bin\
Release\net6.0\Northwind.AzureFuncs.dll
2>Northwind.AzureFuncs -> C:\Code\PracticalApps\Northwind.AzureFuncs\obj\
Release\net6.0\PubTmp\Out\
2>Publishing C:\Code\PracticalApps\Northwind.AzureFuncs\obj\Release\
net6.0\PubTmp\Northwind.AzureFuncs - 20210911153432123.zip to https://
northwindazurefuncs20210911151522.scm.azurewebsites.NET/api/zipdeploy...
2>Zip Deployment succeeded.
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped
==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
Waiting for function app ready....
Finished waiting for function app to be ready
```

12. Протестируйте функцию в браузере (рис. 18.19).

Рис. 18.19. Вызов функции Azure Function в облаке

Очистка ресурсов Azure

Вы можете удалить функциональные приложения и связанные с ним ресурсы, чтобы избежать дальнейших расходов. Для этого выполните следующие шаги.

1. В программе Visual Studio Code выберите команду меню View ▸ Command Palette (Вид ▸ Набор команд).
2. Найдите и выберите пункт Azure Functions: Open in portal (Azure Functions: Открыть в портале).
3. Выберите функциональное приложение.
4. На портале Azure в окне функционального приложения Overview (Обзор) выберите Resource Group (Группа ресурсов).
5. Убедитесь, что она содержит только те ресурсы, которые вы хотите удалить.
6. Нажмите кнопку Delete resource group (Удалить группу ресурсов) и примите все остальные подтверждения.

Сервисы идентификации

Сервисы идентификации используются для аутентификации и авторизации пользователей. Важно, чтобы эти сервисы реализовывали открытые стандарты, чтобы вы могли интегрировать разрозненные системы. К распространенным стандартам относятся *OpenID Connect* и *OAuth 2.0*.

Популярной бесплатной реализацией этих стандартов идентификации с открытым исходным кодом является *IdentityServer4*. Он позволяет разработчикам интегрировать аутентификацию на основе токенов, единую регистрацию и контроль доступа к API на сайтах, в сервисах и приложениях.

Microsoft не планирует официально поддерживать IdentityServer4, поскольку «создание и поддержка сервера аутентификации подразумевает занятость 24/7, а у Microsoft уже есть команда и продукт в этой области, Azure Active Directory, который позволяет бесплатно использовать 500 000 объектов».

Вы можете прочитать документацию по IdentityServer4 по следующей ссылке: https://identityserver4.readthedocs.io/.

Краткое описание вариантов выбора специализированных сервисов

Используйте рекомендации для различных сценариев в качестве руководства (табл. 18.6).

Таблица 18.6. Рекомендации для различных сценариев

Сценарий	Рекомендации
Общедоступные сервисы	Сервисы REST, также известные как сервисы на основе HTTP, лучше всего подходят для сервисов, которые должны быть общедоступными, особенно если их нужно вызывать из браузера или даже с мобильного устройства
Общедоступные сервисы данных	OData и GraphQL хорошо подходят для раскрытия сложных иерархических наборов данных, которые могут поступать из различных хранилищ данных. OData разработан и поддерживается компанией Microsoft в официальных пакетах .NET. GraphQL разработан компанией Facebook и поддерживается пакетами сторонних разработчиков
Обмен данными между сервисами	gRPC разработан для взаимодействия с низкой задержкой и высокой пропускной способностью. Отлично подходит для легковесных внутренних микросервисов, где эффективность имеет решающее значение
Взаимодействие «точка — точка» в режиме реального времени	gRPC имеет отличную поддержку двунаправленной потоковой передачи. Службы gRPC могут передавать сообщения в режиме реального времени без опроса. SignalR также подходит для многих видов взаимодействия в режиме реального времени, хотя менее эффективен, чем gRPC
Широковещательное взаимодействие в режиме реального времени	SignalR имеет отличную поддержку для широковещательного взаимодействия в режиме реального времени со многими клиентами
Многоязычные среды	Инструментарий gRPC поддерживает все популярные языки разработки, благодаря чему gRPC хорошо подходит для мультиязычных и платформенных сред
Среды с ограниченной пропускной способностью	Сообщения gRPC сериализуются с помощью Protobuf, облегченного формата сообщений. Сообщение gRPC всегда меньше, чем эквивалентное сообщение JSON
Наносервисы	Azure Functions не нуждается в круглосуточном размещении, поэтому хорошо подходит для наносервисов, которые обычно не нуждаются в постоянной работе

Практические задания

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

Упражнение 18.1. Проверочные вопросы

Ответьте на следующие вопросы.

1. У вас есть приложение, которое взаимодействует с сервисом, созданным с помощью устаревшего сервиса Windows Communication Foundation. Каковы два возможных варианта переноса сервиса и клиента на современную .NET?
2. Какой транспортный протокол использует сервис OData?
3. Почему веб-сервис OData является более гибким, чем традиционный веб-сервис ASP.NET Core Web API?
4. Что нужно сделать с методом действия в контроллере OData, чтобы включить строки запроса для настройки того, что он возвращает?
5. Какой транспортный протокол использует сервис GraphQL?
6. Как определяются контракты в gRPC?
7. Благодаря каким трем преимуществам gRPC хорошо подходит для реализации сервисов?
8. Какие средства передачи данных использует SignalR и какой из них применяется по умолчанию?
9. В чем разница между моделями внутрипроцессного и изолированного хостинга для Azure Functions?
10. Что рекомендуется делать при разработке сигнатур методов RPC?

Упражнение 18.2. Дополнительные ресурсы

Воспользуйтесь ссылками на странице https://github.com/markjprice/cs10dotnet6/blob/main/book-links.md#chapter-18---building-and-consuming-other-services, чтобы получить дополнительную информацию по темам, приведенным в данной главе.

Резюме

В этой главе вы узнали, как создавать более специализированные типы сервисов с помощью различных технологий, включая gRPC, SignalR, OData, GraphQL и Azure Functions.

В следующей главе вы узнаете, как создавать кросс-платформенные мобильные и настольные приложения с помощью .NET MAUI.

19 Разработка мобильных и настольных приложений с помощью .NET MAUI

Эта глава посвящена тому, как создавать приложения с *графическим пользовательским интерфейсом* (graphical user interface, GUI) путем создания кросс-платформенного мобильного и настольного приложения для iOS и Android, macOS Catalyst и Windows с использованием .NET MAUI (пользовательского интерфейса мультиплатформенного приложения).

Внимание! Примеры из этой главы были протестированы с помощью .NET Release Candidate 2, .NET MAUI Preview 9 и Visual Studio 2022 Preview 5. В будущих предварительных версиях разработчики, вероятно, исправят некоторые недочеты, а возможно, удалят некий функционал, до выхода версии GA. Для получения последних обновлений, пожалуйста, ознакомьтесь с обновленной главой (на английском) в режиме онлайн по следующей ссылке: https://github.com/markjprice/cs10dotnet6/tree/main/docs/chapter19.

Вы узнаете, как *расширяемый язык разметки приложений* (eXtensible Application Markup Language, XAML) упрощает определение пользовательского интерфейса для графического приложения.

Рассказать все о кросс-платформенной разработке GUI в одной главе невозможно, однако, как и веб-разработка, она настолько важна в наши дни, что я хотел бы познакомить вас с ее основами. Рассматривайте эту главу как введение, в котором вы познакомитесь с базовыми положениями, а более подробную информацию сможете найти в книге, посвященной разработке мобильных и настольных приложений.

Приложение позволит вносить списки клиентов в базу данных Northwind и управлять ими. Мобильное приложение будет обращаться к веб-сервису Northwind, который вы создали с помощью ASP.NET Core Web API в главе 16. Если вы еще не сделали этого, то вернитесь к данной главе и создайте сейчас или скачайте из репозитория GitHub по ссылке: https://github.com/markjprice/cs10dotnet6.

Хотя вы можете создать проект .NET MAUI в командной строке, а затем изменить его с помощью Visual Studio Code, официальных инструментов, которые могли бы вам помочь, пока нет. Ожидается, что они появятся в .NET 7.0 в конце 2022 года.

В этой главе:

- замечания по поводу отложенного выпуска .NET MAUI;
- знакомство с XAML;
- знакомство с .NET MAUI;
- разработка мобильных и настольных приложений с помощью .NET MAUI;
- взаимодействие приложения .NET MAUI с веб-сервисами.

Замечания по поводу отложенного выпуска .NET MAUI

Четырнадцатого сентября 2021 года компания Microsoft объявила, что выпуск .NET MAUI будет отложен. «К сожалению, .NET MAUI не будет готов к выпуску вместе с .NET 6 GA в ноябре», — прокомментировал Скотт Хантер, директор по управлению программами .NET. Вы можете прочитать объявление Скотта по следующей ссылке: https://devblogs.microsoft.com/dotnet/update-on-dotnet-maui/.

Ниже приводится приблизительный график выпуска предварительных версий и релиз-кандидатов, а затем и выхода .NET MAUI во втором квартале 2022 года:

- 12 октября 2021 года: .NET MAUI Preview 9 и .NET 6 Release Candidate 2, которые использовались для этой главы, опубликованы в печатных и электронных изданиях этой книги;
- 9 ноября 2021 года: .NET MAUI Preview 10 и .NET 6 GA;
- декабрь 2021 года: .NET MAUI Preview 11;
- январь 2022 года: .NET MAUI Preview 12;
- февраль 2022 года: .NET MAUI Preview 13;
- март 2022 года: .NET MAUI Release Candidate 1;
- апрель 2022 года: .NET MAUI Release Candidate 2;
- май 2022 года: .NET MAUI General Availability на Microsoft Build;
- ноябрь 2022 года: .NET MAUI включен в .NET 7.

Я хотел включить эту главу в печатную книгу, несмотря на то что после публикации некоторые ее части могут измениться. Чтобы поддерживать актуальность главы по мере выхода предварительных версий .NET MAUI, я планирую обновлять ее в репозитории GitHub для этой книги вплоть до выхода GA. Онлайн-версия данной главы на английском языке доступна по следующей ссылке: https://github.com/markjprice/cs10dotnet6/tree/main/docs/chapter19.

Начнем с рассмотрения языка разметки, используемого в .NET MAUI.

Знакомство с XAML

В 2006 году Microsoft выпустила *Windows Presentation Foundation (WPF)*, которая стала первой технологией, использующей *XAML* (eXtensible Application Markup Language, расширяемый язык разметки приложений). Вскоре после этого была разработана технология Silverlight для веб- и мобильных приложений, но она больше не поддерживается Microsoft. WPF до настоящего времени используется для создания настольных приложений Windows; в качестве примера можно привести Visual Studio для Windows.

XAML можно использовать для создания частей таких приложений, как:

- *приложения .NET MAUI* для мобильных и настольных устройств, включая Android, iOS, Windows и macOS. Они являются продуктом развития технологии *Xamarin.Forms*;
- *приложения WinUI 3* для устройств с Windows 10 и 11;
- *приложения Universal Windows Platform (UWP)* для устройств с Windows 10 и 11, Xbox One и Mixed Reality;
- *WPF-приложения* для рабочего стола Windows, включая версии Windows 7 и более поздние;
- *приложения Avalonia и Uno Platform*, использующие кросс-платформенные технологии сторонних производителей.

Упрощение кода с помощью XAML

XAML упрощает код C#, особенно при создании пользовательского интерфейса.

Представьте, что для создания панели инструментов вам необходимо добавить две или более кнопки, расположенные горизонтально.

В C# вы можете написать следующий код:

```
StackPanel toolbar = new();
toolbar.Orientation = Orientation.Horizontal;

Button newButton = new();
newButton.Content = "New";
newButton.Background = new SolidColorBrush(Colors.Pink);
toolbar.Children.Add(newButton);

Button openButton = new();
openButton.Content = "Open";
openButton.Background = new SolidColorBrush(Colors.Pink);
toolbar.Children.Add(openButton);
```

В XAML данный код можно упростить. В процессе обработки этого XAML устанавливаются эквивалентные свойства и вызываются методы для достижения той же цели, которая стояла перед кодом C#:

```
<StackPanel Name="toolbar" Orientation="Horizontal">
  <Button Name="newButton" Background="Pink">New</Button>
  <Button Name="openButton" Background="Pink">Open</Button>
</StackPanel>
```

Вы можете рассматривать XAML как альтернативный и более легкий способ объявления и создания экземпляров типов .NET, особенно при определении пользовательского интерфейса и ресурсов, которые он использует.

XAML позволяет объявлять ресурсы, такие как кисти, стили и темы, на разных уровнях, например на уровне элемента UI, страницы или глобально для приложения, чтобы обеспечить совместное использование ресурсов.

XAML также позволяет выполнять привязку данных между элементами пользовательского интерфейса или между элементами UI и объектами и коллекциями.

Выбор общих элементов управления

Существует множество предопределенных элементов управления, с помощью которых вы можете создавать общие сценарии пользовательского интерфейса. Почти все диалекты XAML поддерживают эти элементы управления (табл. 19.1).

Таблица 19.1. Элементы управления

Элементы управления	Описание
`Button`, `ImageButton`, `Menu`, `Toolbar`	Выполнение определенных действий
`CheckBox`, `RadioButton`	Выбор вариантов
`Calendar`, `DatePicker`	Выбор дат
`ComboBox`, `ListBox`, `ListView`, `TreeView`	Выбор элементов из списков и иерархических деревьев
`Canvas`, `DockPanel`, `Grid`, `StackPanel`, `WrapPanel`	Контейнеры макета, которые по-разному влияют на свои дочерние элементы
`Label`, `TextBlock`	Отображение текста только для чтения
`RichTextBox`, `TextBox`	Редактирование текста
`Image`, `MediaElement`	Встраивание изображений, видео и аудиофайлов
`DataGrid`	Просмотр и редактирование данных максимально быстро и легко
`Scrollbar`, `Slider`, `StatusBar`	Различные элементы пользовательского интерфейса

Расширения разметки

Для поддержки некоторых дополнительных функций XAML использует расширения разметки. Некоторые из наиболее важных из них включают привязку элементов и данных, а также повторное использование ресурсов:

- `{Binding}` связывает элемент со значением из другого элемента или источника данных;
- `{StaticResource}` связывает элемент с общим ресурсом;
- `{ThemeResource}` связывает элемент с общим ресурсом, определенным в теме.

В данной главе вы изучите несколько практических примеров расширений разметки.

Знакомство с .NET MAUI

Создать мобильное приложение, работающее только на iPhone, вы можете с помощью Objective-C или Swift и UIKit, используя инструмент разработки Xcode.

Создать мобильное приложение, работающее только на телефонах с Android, вы можете с помощью Java или Kotlin и Android SDK, используя инструмент разработки Android Studio.

Но что, если вам необходимо создать мобильное приложение, которое может работать на iPhone *и* телефонах с Android? И вы хотите создать его только один раз, используя знакомый вам язык программирования и платформу разработки? А что, если, приложив чуть больше усилий для адаптации пользовательского интерфейса к экранам размером с настольный монитор, вы могли бы также ориентироваться на настольные компьютеры macOS и Windows?

.NET MAUI позволяет разработчикам создавать на языке C# кросс-платформенные мобильные приложения для Apple iOS (iPhone) и iPadOS, macOS с помощью Catalyst, для Windows с помощью WinUI 3 и для Google Android с помощью C# и .NET, которые затем компилируются в собственные API и запускаются на собственных телефонных и настольных платформах.

Уровень бизнес-логики можно написать один раз и использовать на всех платформах. Взаимодействие с пользовательским интерфейсом и API различается на разных мобильных и настольных платформах, поэтому уровень UI иногда настраивается для каждой платформы.

Подобно WPF- и UWP-приложениям, в .NET MAUI язык разметки XAML служит для однократного определения пользовательского интерфейса для всех платформ сразу, абстрагируя платформенно-зависимые компоненты UI. Приложения, созданные с помощью .NET MAUI, формируют UI на основе собственных виджетов

платформы, поэтому приложения удобны и привлекательны, а также идеально подходят под целевую мобильную платформу.

UI, созданный с помощью .NET MAUI, не будет идеально подходить для конкретной платформы так же, как если бы это было сделано на заказ с использованием собственных инструментов для этой платформы, но для мобильных и настольных приложений этого вполне достаточно.

Инструменты разработки для мобильных и облачных технологий

Мобильные приложения часто взаимодействуют с облачными сервисами.

Сатья Наделла, генеральный исполнительный директор Microsoft, однажды сказал: *«Для меня стратегия mobile first означает не мобильность устройств, а мобильность индивидуального опыта. [...] Единственный способ организовать мобильность этих приложений и данных — использовать облако».*

Как и раньше, воспользуемся Visual Studio Code для создания сервиса ASP.NET Core Web API для поддержки мобильного приложения. Создавать приложения .NET MAUI разработчики могут, используя программу Visual Studio 2022 для Windows или Visual Studio 2022 для Mac.

При установке Visual Studio 2022 необходимо установить флажок .NET MAUI (Preview), который относится к нагрузке Mobile development with .NET, как показано на рис. 19.1.

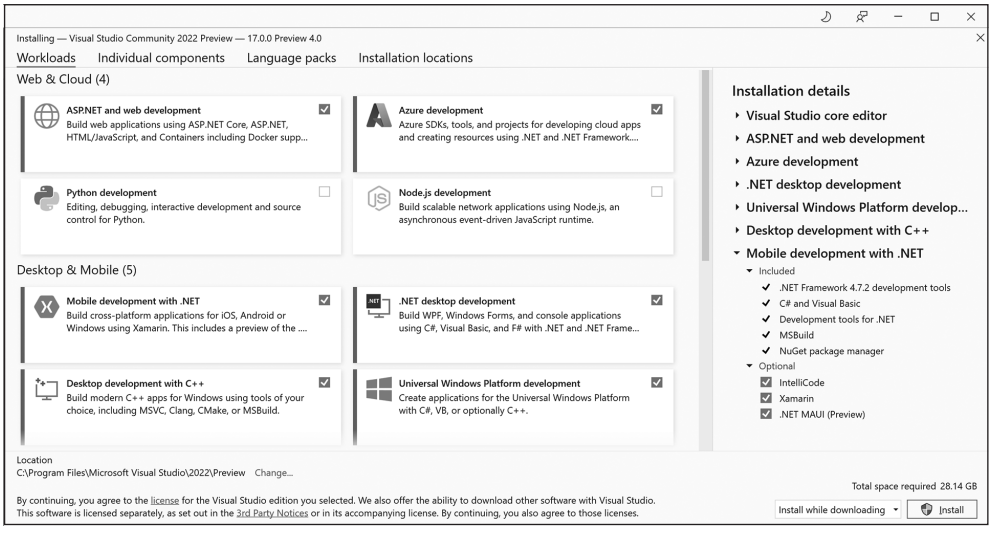

Рис. 19.1. Выбор рабочей нагрузки .NET MAUI для Visual Studio 2022

Создание приложений для iOS и macOS в Windows

Если вы хотите с помощью Visual Studio 2022 для Windows создавать мобильное приложение для iOS или настольное приложение для macOS Catalyst, то можете подключиться по сети к хосту сборки Mac. Инструкции можно найти по следующей ссылке: https://docs.microsoft.com/en-us/xamarin/ios/get-started/installation/windows/connecting-to-mac/.

Дополнительная функциональность

Создадим кросс-платформенное мобильное и настольное приложение, используя множество навыков и знаний, полученных в предыдущих главах. Кроме того, применим некоторые незнакомые функции.

MVVM

Model-View-ViewModel (MVVM) — это паттерн проектирования, подобный MVC. Буквы в аббревиатуре означают следующее:

- *Model* (Модель) — сущностный класс, который представляет объект данных в хранилище, например в реляционной базе данных;
- *View* (Представление) — способ представления данных в графическом пользовательском интерфейсе, включающий поля для отображения и редактирования данных, а также кнопки и другие элементы для взаимодействия с данными;
- *ViewModel* — класс, представляющий поля данных, действия и события, которые затем могут быть привязаны к таким элементам, как текстовые поля и кнопки в представлении.

В MVC модели, передаваемые в представление, доступны только для чтения, поскольку передаются в представление только в одном направлении. Вот почему неизменяемые записи хорошо подходят для моделей MVC. Но классы ViewModel — совсем другие. Они должны поддерживать двустороннее взаимодействие, и если исходные данные изменяются в течение жизни объекта, представление должно динамически обновляться.

Интерфейс INotifationPropertyChanged

Интерфейс `INotifyPropertyChanged` позволяет классу модели поддерживать двустороннюю привязку данных. Он требует от класса реализации события `PropertyChanged` с параметром типа `PropertyChangedEventArgs`:

```
namespace System.ComponentModel
{
  public class PropertyChangedEventArgs : EventArgs
```

```
  {
    public PropertyChangedEventArgs(string? propertyName);
    public virtual string? PropertyName { get; }
  }

  public delegate void PropertyChangedEventHandler(
    object? sender, PropertyChangedEventArgs e);

  public interface INotifyPropertyChanged
  {
    event PropertyChangedEventHandler PropertyChanged;
  }
}
```

Внутри каждого свойства в классе при установке нового значения вам необходимо инициировать событие (если оно не равно `null`) с экземпляром `PropertyChangedEventArgs`, содержащим имя свойства в виде значения `string`:

```
private string companyName;

public string CompanyName
{
  get => companyName;
  set
  {
    companyName = value; // сохраняем новое установленное значение
    PropertyChanged?.Invoke(this,
      new PropertyChangedEventArgs(nameof(CompanyName)));
  }
}
```

Будучи привязанным к свойству данных, элемент управления пользовательского интерфейса автоматически обновится, чтобы показать новое значение при его изменении.

Чтобы упростить реализацию, мы можем использовать функцию компилятора для получения имени свойства, дополнив параметр `string` атрибутом `[CallerMemberName]`:

```
private void NotifyPropertyChanged(
  [CallerMemberName] string propertyName = "")
{
  // если установлен обработчик события, вызываем делегат
  // и передаем имя свойства
  PropertyChanged?.Invoke(this,
    new PropertyChangedEventArgs(propertyName));
}

public string CompanyName
{
```

```
  get => companyName;
  set
  {
    companyName = value; // сохраняем новое установленное значение
    NotifyPropertyChanged(); // имя вызывающего абонента: "CompanyName"
  }
}
```

Класс ObservableCollection

С `INotifyPropertyChanged` связан интерфейс `INotifyCollectionChanged`, который реализуется классом `ObservableCollection<T>`. Он выдает уведомления при добавлении, удалении элементов или при обновлении коллекции. При привязке к элементам управления типа `ListView` или `TreeView` пользовательский интерфейс будет динамически обновляться, отражая изменения.

Сервисы зависимостей

Мобильные платформы, такие как iOS и Android, и настольные платформы, такие как Windows и macOS, по-разному реализуют общие функции, вследствие чего необходим способ реализовать общие функции на уровне платформы. Мы можем сделать это с помощью сервиса зависимостей.

- Определите интерфейс для общей функции, например `IDialer` для компонента набора телефонного номера на телефонном устройстве или `INotificationManager` для всплывающего локального уведомления на настольных и мобильных устройствах.
- Реализуйте интерфейс для всех платформ, которые вам необходимо поддерживать, например iOS и Android для компонента набора телефонного номера, и зарегистрируйте реализации с помощью атрибута:

```
[assembly: Dependency(typeof(PhoneDialer))]

namespace Northwind.Maui.iOS
{
  public class PhoneDialer : IDialer
```

- Получите платформенно-зависимую реализацию интерфейса, используя сервис зависимостей:

```
IDialer dialer = DependencyService.Get<IDialer>();
```

.NET MAUI Essentials включает компонент PhoneDialer, поэтому мы будем использовать его в нашем проекте вместо того, чтобы определять собственный сервис зависимости для набора телефонного номера.

Компоненты пользовательского интерфейса .NET MAUI

.NET MAUI содержит некоторые общие элементы управления, предназначенные для создания пользовательских интерфейсов. Эти элементы делятся на четыре категории:

- *страницы* представляют кросс-платформенные экраны приложений, например `ContentPage`, `NavigationPage`, `FlyoutPage`, и `TabbedPage`;
- *макеты* представляют структуру комбинации других компонентов пользовательского интерфейса, например `Grid`, `StackLayout` и `FlexLayout`;
- *представления* определяют отдельный компонент пользовательского интерфейса, например `CarouselView`, `CollectionView`, `Label`, `Entry`, `Editor` и `Button`;
- *ячейки* представляют отдельный элемент в списке или таблице, например `TextCell`, `ImageCell`, `SwitchCell` и `EntryCell`.

Вы можете отслеживать статус выполнения миграции компонентов .NET MAUI по следующей ссылке: https://github.com/dotnet/maui/wiki/Status.

Представление ContentPage

Представление `ContentPage` предназначено для простых пользовательских интерфейсов. Оно содержит свойство `ToolbarItems`, отображающее действия, которые пользователь может выполнять на платформе. Каждый параметр `ToolbarItem` может содержать значок и текст:

```
<ContentPage.ToolbarItems>
  <ToolbarItem Text="Add" Activated="Add_Activated"
    Order="Primary" Priority="0" />
  ...
</ContentPage.ToolbarItems>
```

Элемент управления ListView

Элемент управления `ListView` используется для длинных списков значений одного типа, связанных по данным. Может содержать верхние и нижние колонтитулы, а его элементы могут быть сгруппированы.

Этот элемент содержит ячейки для каждого элемента списка. Существует два встроенных типа ячеек: текст и изображение. Разработчики могут определять пользовательские типы ячеек.

Для ячеек могут быть определены действия контекста, которые появляются, когда ячейка пролистывается смахиванием влево на iPhone или долгим нажатием

в Android. Контекстное действие, которое является деструктивным, может быть отображено красным цветом.

```
<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>
```

Элементы управления Entry и Editor

Элементы управления `Entry` и `Editor` применяются для редактирования текстовых значений и часто привязаны по данным к свойству модели объекта:

```
<Editor Text="{Binding CompanyName, Mode=TwoWay}" />
```

Используйте элемент `Entry` для одной строки текста, а элемент `Editor` — для нескольких строк.

Обработчики .NET MAUI

В .NET MAUI элементы управления XAML определены в пространстве имен `Microsoft.Maui.Controls`. Компоненты, называемые *обработчиками*, сопоставляют эти общие элементы управления с нативными элементами управления на каждой платформе. В iOS обработчик сопоставляет элемент .NET MAUI `Button` с `UIButton`, определенным UIKit для iOS. В macOS `Button` сопоставляется с `NSButton`, определенным AppKit. В Android `Button` сопоставляется с нативным для Android элементом `AppCompatButton`.

Обработчики имеют свойство `NativeView`, которое раскрывает базовый нативный элемент управления. Это позволяет вам работать с платформенно-зависимыми функциями, такими как свойства, методы и события, и настраивать все экземпляры нативного элемента управления.

Написание платформенно-зависимого кода

Если вам нужно написать операторы, которые будут выполняться только для определенной платформы, например Android, вы можете использовать директивы компилятора.

Например, по умолчанию элементы управления вводом `Entry` в Android сопровождают символы подчеркивания.

Если вы хотите скрыть подчеркивание, то можете написать Android-зависимый код, чтобы получить обработчик для элемента управления `Entry`, использовать его

свойство `NativeView` для доступа к базовому нативному элементу управления, а затем установить свойство, управляющее этой функцией, в значение `false`:

```
#if __ANDROID__
  Handlers.EntryHandler.EntryMapper[nameof(IEntry.BackgroundColor)] = (h, v) =>
  {
    (h.NativeView as global::Android.Views.Entry).UnderlineVisible = false;
  };
#endif
```

Предопределенные константы компилятора включают в себя:

- `__ANDROID__`;
- `__IOS__`;
- `WINDOWS`.

Синтаксис оператора `#if` компилятора немного отличается от синтаксиса оператора `if` в C#:

```
#if __IOS__
  // iOS-зависимые операторы
#elif __ANDROID__
  // Android-зависимые операторы
#elif WINDOWS
  // Windows-зависимые операторы
#endif
```

Разработка мобильных и настольных приложений с помощью .NET MAUI

Создадим мобильное и настольное приложение для управления клиентами в Northwind.

Если вы никогда не запускали среду разработки Xcode, то запустите ее сейчас, чтобы увидеть окно Start (Пуск) и убедиться, что все необходимые компоненты установлены и зарегистрированы. Если не запускаете Xcode, то можете позже получить ошибки в среде Visual Studio для Mac.

Создание виртуального устройства Android для локального тестирования приложений

Для разработки под Android необходимо установить хотя бы один Android SDK. Установка по умолчанию Visual Studio с рабочей нагрузкой на разработку мобильных устройств уже включает в себя один Android SDK, но часто это довольно

старая версия, предназначенная для поддержки как можно большего количества устройств с Android.

Чтобы использовать последние функции .NET MAUI, вам необходимо установить более позднюю версию Android SDK.

1. В Windows запустите программу Visual Studio 2022.
2. Выберите команду меню Tools ▸ Android ▸ Android Device Manager (Инструменты ▸ Android ▸ Диспетчер устройств Android).
3. В Android Device Manager (Диспетчер устройств Android) нажмите кнопку + New (Создать), чтобы создать новое устройство.
4. В диалоговом окне New Device (Новое устройство) выберите следующие параметры:
   1) Base Device (Основное устройство): Pixel 2 (+ Store);
   2) Processor (Процессор): x86;
   3) OS (ОС): Pie 9.0 – API 28.
5. Нажмите кнопку Create (Создать).
6. Примите лицензионные соглашения.
7. Дождитесь скачивания всех необходимых файлов.
8. В Android Device Manager (Диспетчер устройств Android) в списке устройств в строке для устройства, которое вы только что создали, нажмите кнопку Start (Пуск).
9. Когда устройство Android завершит запуск, откройте браузер и проверьте, есть ли у него доступ к сети.
10. Закройте эмулятор.
11. Перезапустите Visual Studio 2022, чтобы убедиться, что программа поддерживает новый эмулятор.

Создание решения .NET MAUI

Сейчас мы создадим проект для кросс-платформенного мобильного и настольного приложения.

1. В программе Visual Studio для Windows создайте проект с такими настройками:
   1) шаблон проекта: .NET MAUI App (Preview)/`maui`;
   2) файл и папка рабочей области/решения: `PracticalApps`;
   3) файл и папка проекта: `Northwind.Maui.Customers`.

2. Откройте файл проекта и раскомментируйте элемент для включения таргетинга на Windows, как показано ниже:

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

  <PropertyGroup>
    <TargetFrameworks>net6.0-ios;net6.0-android;net6.0-maccatalyst</
    TargetFrameworks>
    <TargetFrameworks Condition="$([MSBuild]::IsOSPlatform('windows'))
    and '$(MSBuildRuntimeType)' == 'Full'">$(TargetFrameworks);net6.0-
    windows10.0.19041</TargetFrameworks>
    <OutputType>Exe</OutputType>
    <RootNamespace>Northwind.Maui.Customers</RootNamespace>
    <UseMaui>true</UseMaui>
    <SingleProject>true</SingleProject>
```

3. Справа от кнопки Run (Запуск) на панели инструментов присвойте параметру Framework значение net6.0-android и выберите образ эмулятора Pixel 2 - API 28 (Android 9.0 - API 28), который вы создали ранее (рис. 19.2).

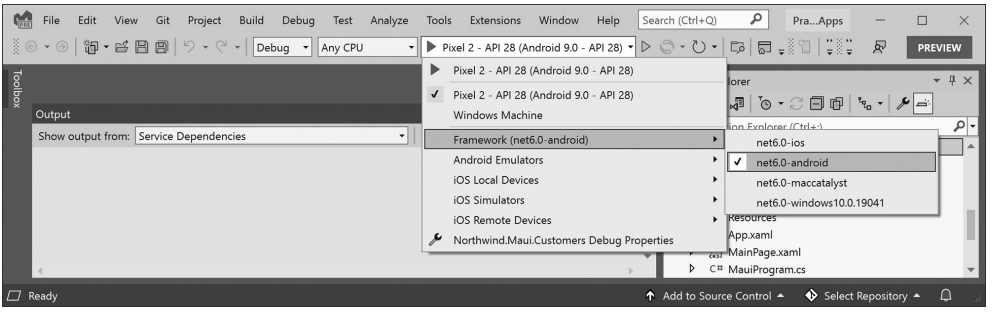

Рис. 19.2. Выбор Android в качестве цели для запуска

4. Нажмите кнопку Run (Запуск) на панели инструментов и дождитесь, пока эмулятор устройства запустит операционную систему Android и ваше мобильное приложение.

5. В приложении .NET MAUI нажмите кнопку Click me (Нажми меня), чтобы увеличить показания счетчика в три раза (рис. 19.3).

6. Обратите внимание на окно XAML Live Preview (Динамический просмотр XAML) в программе Visual Studio и на то, что подключена функция XAML Hot Reload (Горячая перезагрузка XAML), чтобы вы могли вносить изменения в XAML и видеть их отражение в приложении, не перезапуская его. Например, попробуйте изменить текст метки Hello, World! на какой-нибудь другой, сохраните XAML-файл и нажмите кнопку Hot Reload (Горячая перезагрузка) на панели инструментов.

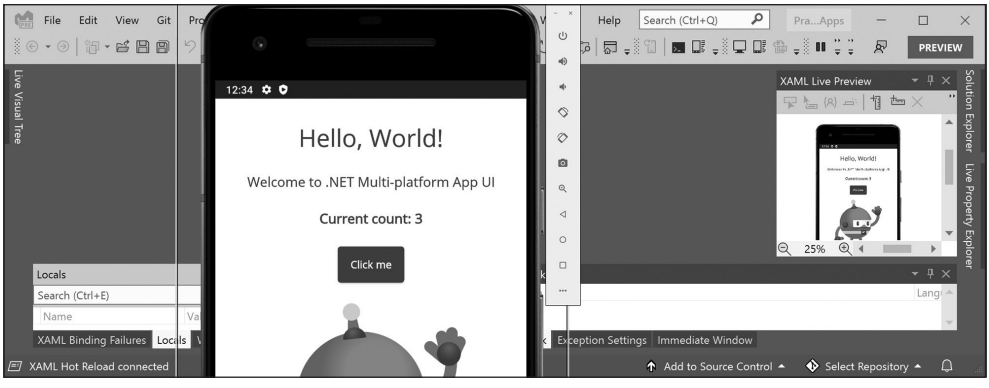

Рис. 19.3. Увеличение показаний счетчика в приложении Android .NET MAUI

7. Закройте симулятор устройства Android.
8. Выберите команду меню Build ▸ Configuration Manager (Сборка ▸ Диспетчер конфигурации).
9. В строке для проекта Northwind.Maui.Customers установите флажок Deploy (Развертывание) (рис. 19.4).

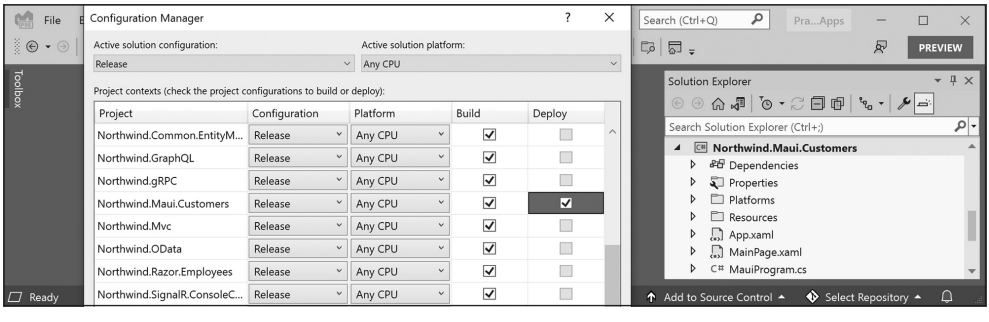

Рис. 19.4. Включение приложения Windows для развертывания на компьютере Windows

10. Справа от кнопки Run (Запуск) на панели инструментов присвойте параметру Framework значение net6.0-windows, а затем выберите Windows Machine.
11. Убедитесь, что выбрана конфигурация Debug (Отладка), а затем нажмите кнопку запуска с зеленым треугольником и меткой Windows Machine.
12. Через несколько мгновений обратите внимание, что приложение Windows отображается с той же кнопкой Click me (Нажми меня) и функцией счетчика (рис. 19.5).
13. Закройте приложение Windows.

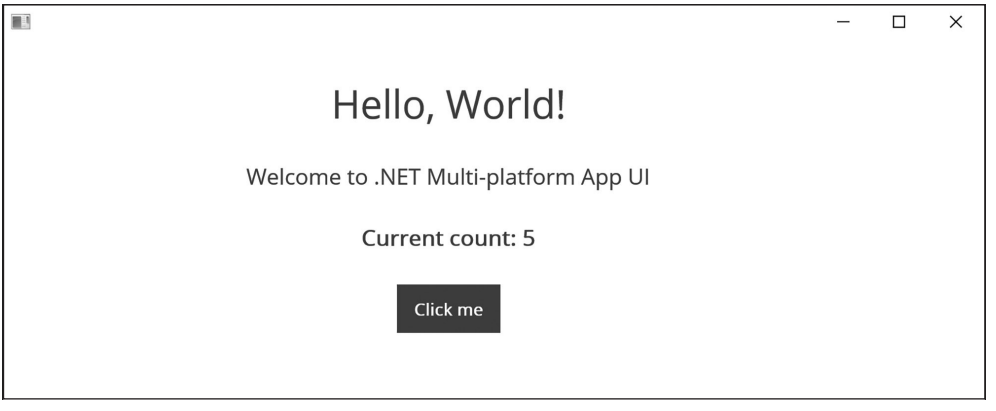

Рис. 19.5. Увеличение показаний счетчика в приложении Windows .NET MAUI

Создание модели представления с двусторонней привязкой данных

Нам нужно создать модель представления, которая позволит нам показывать и изменять сущностный класс `Customer`, поэтому он должен реализовать двустороннюю привязку данных.

1. В папке проекта `Northwind.Maui.Customers` создайте два класса, один — `CustomerDetailViewModel.cs` для отображения сведений об одном клиенте, а другой — `CustomersListViewModel.cs` для отображения списка клиентов.
2. В классе `CustomerDetailViewModel.cs` измените операторы так, чтобы определить класс, который реализует интерфейс `INotifyPropertyChanged` и содержит шесть свойств:

```
using System.ComponentModel; // INotifyPropertyChanged
using System.Runtime.CompilerServices; // [CallerMemberName]

namespace Northwind.Maui.Customers;

public class CustomerDetailViewModel : INotifyPropertyChanged
{
  public event PropertyChangedEventHandler PropertyChanged;

  private string customerId;
  private string companyName;
  private string contactName;
  private string city;
  private string country;
  private string phone;
```

```
// этот атрибут устанавливает параметр propertyName,
// используя контекст, в котором вызывается этот метод
private void NotifyPropertyChanged(
  [CallerMemberName] string propertyName = "")
{
  // если установлен обработчик события, вызываем делегат
  // и передаем имя свойства
  PropertyChanged?.Invoke(this,
    new PropertyChangedEventArgs(propertyName));
}

public string CustomerId
{
  get => customerId;
  set
  {
    customerId = value;
    NotifyPropertyChanged();
  }
}

public string CompanyName
{
  get => companyName;
  set
  {
    companyName = value;
    NotifyPropertyChanged();
  }
}

public string ContactName
{
  get => contactName;
  set
  {
    contactName = value;
    NotifyPropertyChanged();
  }
}

public string City
{
  get => city;
  set
  {
    city = value;
    NotifyPropertyChanged();
    NotifyPropertyChanged(nameof(Location));
  }
}
```

```
  public string Country
  {
    get => country;
    set
    {
      country = value;
      NotifyPropertyChanged();
      NotifyPropertyChanged(nameof(Location));
    }
  }

  public string Phone
  {
    get => phone;
    set
    {
      phone = value;
      NotifyPropertyChanged();
    }
  }

  public string Location
  {
    get => $"{City}, {Country}";
  }
}
```

Обратите внимание на следующие моменты:

- класс реализует интерфейс `INotifyPropertyChanged`, в связи с чем элемент управления с двухсторонней привязкой, такой как `Editor`, будет обновлять свойство, и наоборот. Событие `PropertyChanged` возникает всякий раз, когда одно из свойств изменяется с помощью частного метода `NotifyPropertyChanged`, упрощающего реализацию;
- помимо свойств для сохранения значений, полученных из HTTP-сервиса, класс определяет свойство `Location`, доступное только для чтения. Оно будет использоваться для привязки в сводном списке клиентов в целях отображения местоположения каждого из них. При изменении свойства `City` или `Country` нам также необходимо уведомлять об изменении `Location`, иначе все представления, привязанные к `Location`, будут обновляться некорректно.

3. В классе `CustomersListViewModel.cs` измените операторы, чтобы определить класс, который наследуется от `ObservableCollection<T>` и имеет метод для заполнения данных выборки:

```
using System.Collections.ObjectModel; // ObservableCollection<T>

namespace Northwind.Maui.Customers;

public class CustomersListViewModel :
```

```
  ObservableCollection<CustomerDetailViewModel>
{
  // тестирование перед вызовом веб-сервиса
  public void AddSampleData(bool clearList = true)
  {
    if (clearList) Clear();

    Add(new CustomerDetailViewModel
    {
      CustomerId = "ALFKI",
      CompanyName = "Alfreds Futterkiste",
      ContactName = "Maria Anders",
      City = "Berlin",
      Country = "Germany",
      Phone = "030-0074321"
    });

    Add(new CustomerDetailViewModel
    {
      CustomerId = "FRANK",
      CompanyName = "Frankenversand",
      ContactName = "Peter Franken",
      City = "München",
      Country = "Germany",
      Phone = "089-0877310"
    });

    Add(new CustomerDetailViewModel
    {
      CustomerId = "SEVES",
      CompanyName = "Seven Seas Imports",
      ContactName = "Hari Kumar",
      City = "London",
      Country = "UK",
      Phone = "(171) 555-1717"
    });
  }
}
```

Обратите внимание на следующие моменты:

- после загрузки из сервиса, который будет реализован позже в этой главе, клиенты кэшируются локально с помощью `ObservableCollection<T>`. Так поддерживаются уведомления для любых связанных компонентов пользовательского интерфейса, таких как `ListView`, чтобы UI мог перерисовать себя, когда базовые данные добавляют или удаляют элементы из коллекции;
- в целях тестирования, когда HTTP-сервис недоступен, существует статический метод для заполнения трех образцов клиентов.

Создание представлений для списка клиентов и подробной информации о клиенте

Заменим существующий код в файле `MainPage` на представление для отображения списка клиентов и сведений о клиенте.

1. В проекте `Northwind.Maui.Customers` удалите файл `MainPage.xaml`.
2. Откройте файл `App.xaml` и добавьте стиль для применения того же цвета фона и семейства шрифта к элементам управления `Entry`, которые применяются к элементам управления `Label`, как показано ниже:

```
<Style TargetType="Entry">
  <Setter Property="TextColor" Value="{DynamicResource PrimaryTextColor}"/>
  <Setter Property="FontFamily" Value="OpenSansRegular" />
  <Setter Property="HorizontalOptions" Value="StartAndExpand" />
  <Setter Property="WidthRequest" Value="300" />
</Style>
```

Реализация представления списка клиентов

Сначала мы создадим два представления для списка клиентов и показа подробной информации об одном клиенте, а затем реализуем список клиентов.

1. Щелкните правой кнопкой мыши на папке проекта `Northwind.Maui.Customers`, выберите команду меню Add ▸ New Item (Добавить ▸ Новый элемент), выберите Content Page (Страница содержимого), введите имя `CustomersListPage` и нажмите кнопку Add (Добавить) (рис. 19.6).

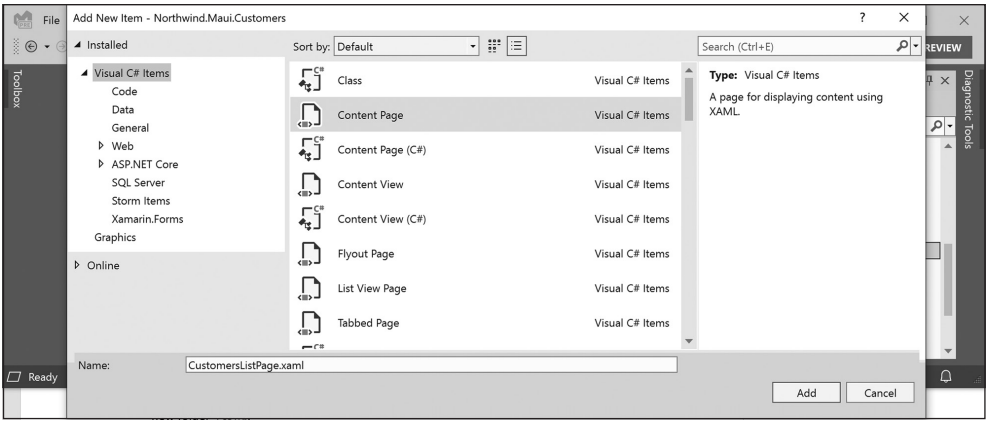

Рис. 19.6. Добавление нового элемента страницы содержимого XAML

2. Щелкните правой кнопкой мыши на папке Views, выберите команду меню Add ▸ New Item (Добавить ▸ Новый элемент), выберите Content Page (Страница содержимого), введите имя `CustomerDetailPage` и нажмите кнопку Add (Добавить).

На момент написания этой книги в программе Visual Studio 2022 нет шаблонов элементов проекта для .NET MAUI. Шаблон элемента проекта ContentPage предназначен для более старой версии Xamarin.Forms. На следующем этапе мы все равно заменим почти всю разметку и код, так что это не проблема.

3. Откройте файл `CustomersListPage.xaml` и измените его содержимое:

```
<ContentPage
  xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="Northwind.Maui.Customers.CustomersListPage"
  BackgroundColor="{DynamicResource PageBackgroundColor}"
  Title="List">
    <ContentPage.Content>
      <ListView ItemsSource="{Binding .}"
                VerticalOptions="Center"
                HorizontalOptions="Center"
                IsPullToRefreshEnabled="True"
                ItemTapped="Customer_Tapped"
                Refreshing="Customers_Refreshing">
        <ListView.Header>
          <StackLayout Orientation="Horizontal">
            <Label Text="Northwind Customers"
                   FontSize="Subtitle" Margin="10" />
            <Button Text="Add" Clicked="Add_Clicked" />
          </StackLayout>
        </ListView.Header>
        <ListView.ItemTemplate>
          <DataTemplate>
            <TextCell Text="{Binding CompanyName}"
                      Detail="{Binding Location}"
                      TextColor="{DynamicResource PrimaryTextColor}"
                      DetailColor="{DynamicResource PrimaryTextColor}" >
            <TextCell.ContextActions>
              <MenuItem Clicked="Customer_Phoned" Text="Phone" />
              <MenuItem Clicked="Customer_Deleted" Text="Delete"
                        IsDestructive="True" />
            </TextCell.ContextActions>
          </TextCell>
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>
  </ContentPage.Content>
</ContentPage>
```

Обратите внимание на следующие моменты:

- класс `ContentPage` содержит атрибут `Title` со значением `List`;
- класс `ListView` содержит атрибут `IsPullToRefreshEnabled` со значением `true`;
- были написаны обработчики для следующих событий:
  - `Customer_Tapped` — производится касание записи клиента, чтобы посмотреть сведения о нем;
  - `Customers_Refreshing` — список тянется вниз для обновления элементов;
  - `Customer_Phoned` — ячейка смахивается влево на iPhone или долго нажимается в Android, и далее производится касание кнопки Phone (Позвонить);
  - `Customer_Deleted` — ячейка смахивается влево на iPhone или долго нажимается в Android, и далее производится касание кнопки Delete (Удалить);
  - `Add_Clicked` — производится нажатие кнопки Add (Добавить);
- шаблон данных определяет способ отображения сведений о каждом клиенте: более крупный шрифт для названия компании и мелкий — для ее адреса;
- кнопка Add (Добавить) находится в заголовке представления списка, что позволяет пользователям перейти к представлению подробной информации и добавить нового клиента.

4. Найдите и откройте файл `CustomersListPage.xaml.cs` и измените его содержимое:

```
using Microsoft.Maui.Controls; // ContentPage, ListView
using Microsoft.Maui.Essentials; // PhoneDialer
using System;
using System.Threading.Tasks;

namespace Northwind.Maui.Customers;

public partial class CustomersListPage : ContentPage
{
  public CustomersListPage()
  {
    InitializeComponent();

    CustomersListViewModel viewModel = new();
    viewModel.AddSampleData();
    BindingContext = viewModel;
  }

  async void Customer_Tapped(object sender, ItemTappedEventArgs e)
  {
    if (e.Item is not CustomerDetailViewModel c) return;
```

```
    // переходим к подробному представлению и показываем
    // подключенного клиента
    await Navigation.PushAsync(new CustomerDetailPage(
      BindingContext as CustomersListViewModel, c));
  }

  async void Customers_Refreshing(object sender, EventArgs e)
  {
    if (sender is not ListView listView) return;

    listView.IsRefreshing = true;

    // имитируем обновление
    await Task.Delay(1500);

    listView.IsRefreshing = false;
  }

  void Customer_Deleted(object sender, EventArgs e)
  {
    MenuItem menuItem = sender as MenuItem;
    if (menuItem.BindingContext is not CustomerDetailViewModel c) return;
    (BindingContext as CustomersListViewModel).Remove(c);
  }

  async void Customer_Phoned(object sender, EventArgs e)
  {
    MenuItem menuItem = sender as MenuItem;
    if (menuItem.BindingContext is not CustomerDetailViewModel c) return;

    if (await DisplayAlert("Dial a Number",
      "Would you like to call " + c.Phone + "?",
      "Yes", "No"))
    {
      PhoneDialer.Open(c.Phone);
    }
  }

  async void Add_Clicked(object sender, EventArgs e)
  {
    await Navigation.PushAsync(new CustomerDetailPage(
      BindingContext as CustomersListViewModel));
  }
}
```

Обратите внимание на следующие моменты:

- значение `BindingContext` устанавливается на экземпляр `CustomersViewModel`, который заполняется данными образца в конструкторе страницы;
- при касании записи о клиенте в представлении списка пользователь попадает в представление сведений (которое вы реализуете на следующем шаге);

- опускаясь, представление списка запускает смоделированное обновление, которое занимает 1,5 с;
- при удалении из представления списка клиент удаляется из связанной модели представления клиентов;
- когда запись о пользователе в представлении списка смахивается, а кнопка Phone (Позвонить) нажата, в диалоговом окне пользователю будет предложено набрать номер. Если он согласится, то встроенная в платформу реализация будет получена с помощью разрешения зависимостей, а затем использована для набора номера;
- при нажатии кнопки Add (Добавить) пользователь перенаправляется на страницу сведений о клиенте для ввода сведений о новом клиенте.

Реализация подробного представления о клиенте

Далее мы реализуем подробное представление о клиенте.

1. Откройте файл `CustomerDetailPage.xaml` и измените его содержимое, как показано в коде ниже. Обратите внимание на следующие моменты:
   - элемент `Title` страницы содержимого имеет значение `Edit`;
   - макет построен на элементе `Grid` для вывода клиентов, который содержит два столбца и шесть строк;
   - представление `Entry` двусторонне связано по данным со свойствами класса `CustomerViewModel`;
   - для `InsertButton` определен обработчик событий для выполнения кода, добавляющего нового клиента.

```
<ContentPage
  xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="Northwind.Maui.Customers.Views.CustomerDetailPage"
  BackgroundColor="{DynamicResource PageBackgroundColor}"
  Title="Edit">

  <ContentPage.Content>
    <StackLayout VerticalOptions="Fill" HorizontalOptions="Fill">
      <Grid ColumnDefinitions="Auto,Auto"
            RowDefinitions="Auto,Auto,Auto,Auto,Auto,Auto">
        <Label Text="Customer Id" VerticalOptions="Center" Margin="6" />
        <Entry Text="{Binding CustomerId, Mode=TwoWay}" Grid.Column="1"
               MaxLength="5" TextTransform="Uppercase" />
        <Label Text="Company Name" Grid.Row="1"
               VerticalOptions="Center" Margin="6" />
        <Entry Text="{Binding CompanyName, Mode=TwoWay}"
               Grid.Column="1" Grid.Row="1" />
        <Label Text="Contact Name" Grid.Row="2"
```

```
                VerticalOptions="Center" Margin="6" />
          <Entry Text="{Binding ContactName, Mode=TwoWay}"
                Grid.Column="1" Grid.Row="2" />
          <Label Text="City" Grid.Row="3"
                VerticalOptions="Center" Margin="6" />
          <Entry Text="{Binding City, Mode=TwoWay}"
                Grid.Column="1" Grid.Row="3" />
          <Label Text="Country" Grid.Row="4"
                VerticalOptions="Center" Margin="6" />
          <Entry Text="{Binding Country, Mode=TwoWay}"
                Grid.Column="1" Grid.Row="4" />
          <Label Text="Phone" Grid.Row="5"
                VerticalOptions="Center" Margin="6" />
          <Entry Text="{Binding Phone, Mode=TwoWay}"
                Grid.Column="1" Grid.Row="5" />
        </Grid>
        <Button x:Name="InsertButton" Text="Insert Customer"
                Clicked="InsertButton_Clicked" />
      </StackLayout>
    </ContentPage.Content>
  </ContentPage>
```

2. Откройте файл `CustomerDetailPage.xaml.cs` и измените его содержимое:

```
using Microsoft.Maui.Controls;
using System;
using System.Threading.Tasks;

namespace Northwind.Maui.Customers;

public partial class CustomerDetailPage : ContentPage
{
  private CustomersListViewModel customers;

  public CustomerDetailPage(CustomersListViewModel customers)
  {
    InitializeComponent();

    this.customers = customers;
    BindingContext = new CustomerDetailViewModel();
    Title = "Add Customer";
  }

  public CustomerDetailPage(CustomersListViewModel customers,
    CustomerDetailViewModel customer)
  {
    InitializeComponent();

    this.customers = customers;
    BindingContext = customer;
    InsertButton.IsVisible = false;
  }
```

```
  async void InsertButton_Clicked(object sender, EventArgs e)
  {
    customers.Add((CustomerDetailViewModel)BindingContext);
    await Navigation.PopAsync(animated: true);
  }
}
```

Обратите внимание на следующие моменты:

- конструктор по умолчанию устанавливает контекст привязки для нового экземпляра `customer`, а заголовок представления изменяется на `Add Customer`;
- конструктор с параметром `customer` устанавливает контекст привязки для этого экземпляра и скрывает кнопку Insert (Вставить), поскольку из-за двусторонней привязки данных она не нужна при редактировании существующего клиента;
- при нажатии кнопки Insert (Вставить) новый клиент добавляется к модели представления клиентов и навигация асинхронно возвращается к предыдущему виду.

Настройка главной страницы для мобильного приложения

Наконец, необходимо отредактировать код мобильного приложения, чтобы использовать список клиентов, обернутый в страницу навигации, в качестве главной страницы вместо удаленной, которая была создана с помощью шаблона проекта.

1. Откройте файл `App.xaml.cs`.
2. В конструкторе `App` измените оператор, создающий `MainPage`, чтобы вместо него создать экземпляр `CustomersListPage`, обернутый в экземпляр `NavigationPage`, как показано ниже:

```
public App()
{
  InitializeComponent();

  MainPage = new NavigationPage(new CustomersListPage());
}
```

Тестирование мобильного приложения в среде iOS

Протестируем мобильное приложение с помощью эмулятора устройства Android.

1. В программе Visual Studio справа от кнопки Run (Запуск) на панели инструментов установите целевой параметр Framework на значение net6.0-android и выберите эмулятор Android.

2. Начните с отладки проекта. Проект будет собран, а через несколько мгновений появится эмулятор устройства Android с запущенным приложением .NET MAUI (рис. 19.7).

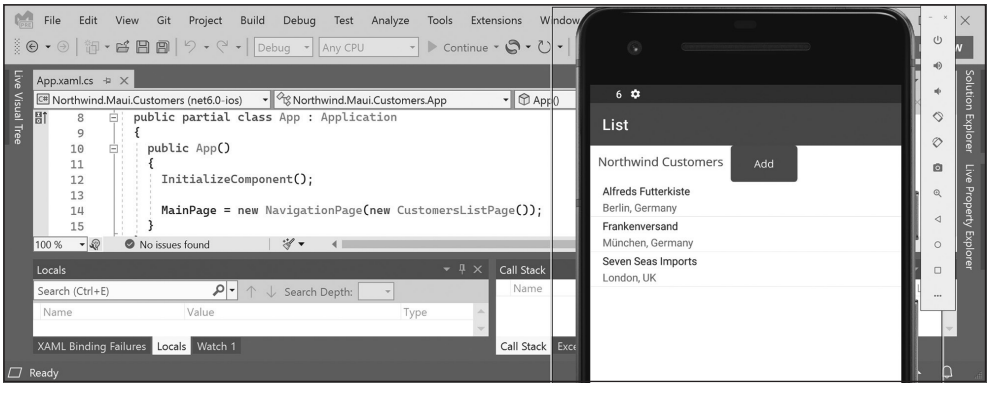

Рис. 19.7. Эмулятор устройства Android, на котором запущено приложение Northwind Customers .NET MAUI

3. Щелкните кнопкой мыши на Seven Seas Imports и измените Company Name (Название компании) на Seven Oceans Imports (рис. 19.8).

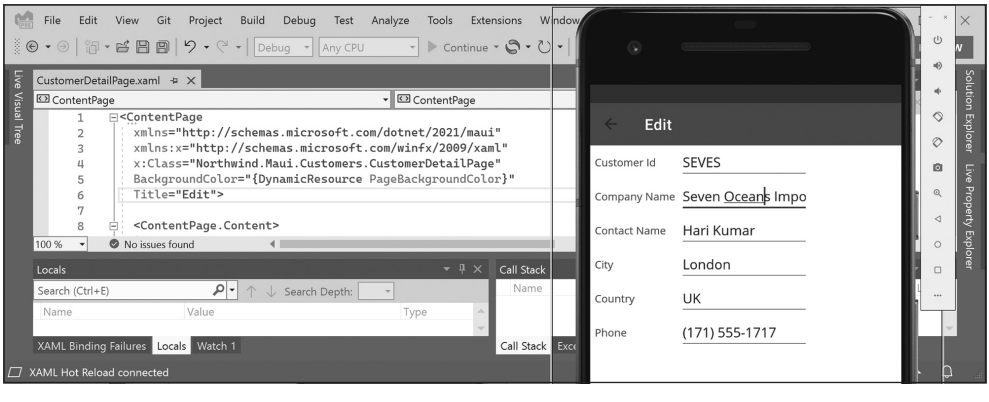

Рис. 19.8. Редактирование названия компании на странице сведений о клиенте

4. Нажмите кнопку возврата, чтобы вернуться к списку клиентов, и обратите внимание, что название компании было обновлено из-за двусторонней привязки данных.
5. Щелкните на ссылке Add (Добавить) и заполните поля ввода для добавления записи о новом клиенте.

По умолчанию в эмуляторе устройств Android при наборе текста на физической клавиатуре отображается виртуальная. Чтобы скрыть ее, щелкните на значке клавиатуры справа от квадратной программной кнопки Android, а затем установите флажок Show virtual keyboard (Показывать виртуальную клавиатуру).

6. На странице сведений о клиенте нажмите кнопку Insert Customer (Вставить клиента) и после возвращения к списку клиентов обратите внимание, что новый клиент был добавлен в нижнюю часть списка. (На момент написания этой книги при использовании .NET MAUI Preview 9 существует ошибка, из-за которой представление списка не обновляется должным образом. Нажмите, удерживайте и перетащите вниз представление списка, а затем отпустите, чтобы обновить его.)
7. Нажмите и удерживайте нажатие на одном из клиентов, чтобы открыть кнопки действий Phone (Позвонить) и Delete (Удалить), как показано на рис. 19.9.

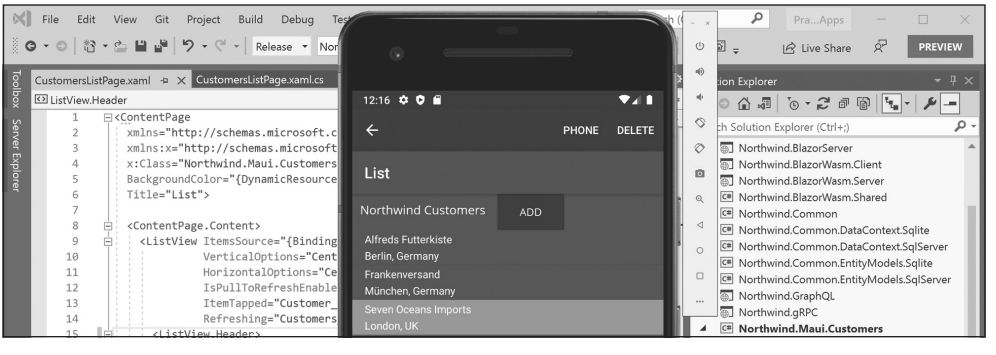

Рис. 19.9. Дополнительные команды для выбранного клиента

8. Щелкните на кнопке Phone (Позвонить) и обратите внимание на появившееся уведомление с подтверждением телефонного вызова клиента (кнопки Yes (Да) и No (Нет)).
9. Нажмите кнопку No (Нет).
10. Нажмите и удерживайте нажатие на одном из клиентов, чтобы открыть кнопки действий Phone (Позвонить) и Delete (Удалить). Щелкните кнопкой мыши на ссылке Delete (Удалить) и обратите внимание, что сведения о клиенте были удалены.
11. Нажмите, удерживайте и перетащите список вниз, а затем отпустите и обратите внимание на эффект анимации для обновления списка. Помните, что данная функция симулируется, поэтому список не изменяется.
12. Закройте эмулятор устройства Android.

Теперь мы заставим приложение вызвать сервис `Northwind.WebApi` для получения списка клиентов.

Взаимодействие мобильных приложений с веб-сервисами

Функция платформ Apple *App Transport Security (ATS)* заставляет разработчиков использовать передовой опыт, включая безопасные соединения между приложением и веб-сервисом. Она включена по умолчанию, и при небезопасном подключении мобильные приложения будут выдавать ошибку.

Вызвать веб-сервис, который защищен самоподписанным сертификатом, таким как наш сервис `Northwind.WebApi`, возможно, однако не слишком легко. Для простоты мы разрешим небезопасные подключения к веб-сервису и отключим проверки безопасности в мобильном приложении.

Разрешение небезопасных запросов в веб-сервисе

В первую очередь разрешим веб-сервису обрабатывать небезопасные соединения по новому URL.

1. В проекте `Northwind.WebApi` в файле `Program.cs` в разделе, который настраивает HTTP-конвейер, закомментируйте перенаправление HTTPS:

```
// закомментировано для проекта приложения .NET MAUI
// app.UseHttpsRedirection();
```

2. В файле `Program.cs` в методе `UseUrls` добавьте небезопасный URL, как показано ниже (выделено жирным шрифтом):

```
var builder = WebApplication.CreateBuilder(args);

builder.WebHost.UseUrls(
    "https://localhost:5002"
    , "http://localhost:5008" // для клиента .NET MAUI
);
```

3. Запустите проект веб-сервиса `Northwind.WebApi` без отладки.
4. Запустите браузер Google Chrome и убедитесь, что веб-сервис возвращает клиентов в формате JSON, перейдя по адресу http://localhost:5008/api/customers/.
5. Закройте браузер Google Chrome, но оставьте веб-сервис работать.

Разрешение небезопасных подключений в приложении для iOS

Настроим проект `Northwind.Maui.Customers`, чтобы отключить функцию ATS и разрешить незащищенные HTTP-запросы к веб-сервису.

1. В проекте `Northwind.Maui.Customers` в папке Platforms/iOS откройте файл `Info.plist`, щелкнув правой кнопкой мыши, и откройте его с помощью XML (Text) Editor.
2. В конце словаря добавьте новый ключ NSAppTransportSecurity, который является словарем, и добавьте в него ключ NSAllowsArbitraryLoads со значением `true`, как показано ниже:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
          "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>LSRequiresIPhoneOS</key>
  <true/>
  ...
  <key>NSAppTransportSecurity</key>
  <dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
  </dict>
</dict>
</plist>
```

3. Сохраните и закройте файл `Info.plist`.

Разрешение небезопасных подключений в приложении для Android

Аналогично для Apple и ATS в Android 9 (уровень API 28) поддержка открытого текста (то есть без HTTPS) по умолчанию отключена.

Настроим проект, чтобы включить функцию открытого текста и разрешить незащищенные HTTP-запросы к веб-сервису.

1. В папке Platforms/Android, находящейся в папке Properties, откройте файл `MainApplication.cs`.
2. В атрибуте `Application` включите открытый текст, как показано ниже:

```
namespace Northwind.Maui.Customers
{
  [Application(UsesCleartextTraffic = true)]
  public class MainApplication : MauiApplication
```

Получение данных о клиентах с помощью сервиса

Теперь мы можем изменить страницу со списком клиентов, чтобы получать ее список из веб-сервиса вместо того, чтобы использовать образцы данных.

1. В проекте `Northwind.Maui.Customers` откройте файл `CustomersListPage.xaml.cs`.
2. Импортируйте следующие дополнительные пространства имен:

```
using System.Collections.Generic; // IEnumerable<T>
using System.Linq; // OrderBy
using System.NET.Http; // HttpClient
using System.NET.Http.Headers; // MediaTypeWithQualityHeaderValue
using System.NET.Http.Json; // ReadFromJsonAsync<T>
```

3. Измените код конструктора `CustomersListPage` так, чтобы загружать список клиентов через прокси сервиса и вызывать метод `AddSampleData` только при возникновении исключения:

```
public CustomersListPage()
{
  InitializeComponent();
  CustomersListViewModel viewModel = new();

  try
  {
    HttpClient client = new()
    {
      BaseAddress = new Uri("http://localhost:5008/")
    };

    client.DefaultRequestHeaders.Accept.Add(
      new MediaTypeWithQualityHeaderValue("application/json"));

    HttpResponseMessage response = client
      .GetAsync("api/customers").Result;

    response.EnsureSuccessStatusCode();

    IEnumerable<CustomerDetailViewModel> customersFromService =
      response.Content.ReadFromJsonAsync
      <IEnumerable<CustomerDetailViewModel>>().Result;

    foreach (CustomerDetailViewModel c in customersFromService
      .OrderBy(customer => customer.CompanyName))
    {
      viewModel.Add(c);
    }
  }
```

```
    catch (Exception ex)
    {
      DisplayAlert(title: "Exception",
        message: $"App will use sample data due to: {ex.Message}",
        cancel: "OK");

      viewModel.AddSampleData();
    }

    BindingContext = viewModel;
  }
```

4. Выберите команду меню Build ▸ Clean Northwind.Maui.Customers (Сборка ▸ Очистить Northwind.Maui.Customers), поскольку изменения в файле `Info.plist`, такие как разрешение небезопасных соединений, иногда требуют чистой сборки.
5. Выберите команду меню Build ▸ Build Northwind.Maui.Customers (Сборка ▸ Собрать Northwind.Maui.Customers).
6. Запустите проект `Northwind.Maui.Customers` на эмуляторе Android и обратите внимание, что сведения о 91 клиенте загружены с помощью веб-сервиса.
7. Закройте эмулятор устройства Android.

Практические задания

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

Упражнение 19.1. Проверочные вопросы

Ответьте на следующие вопросы.

1. Каковы четыре категории компонентов пользовательского интерфейса .NET MAUI и что они собой представляют?
2. Каковы четыре типа ячеек?
3. Каким образом вы можете разрешить пользователю выполнять действия с ячейками в `ListView`?
4. В каком случае элемент `Entry` используется вместо элемента `Editor`?
5. В чем заключается эффект установки `IsDestructive` в значение `true` для пункта меню в контекстных действиях ячейки?
6. Когда выполняется вызов методов `PushAsync` и `PopAsync` в приложении .NET MAUI?

7. В чем разница между `Margin` и `Padding` для такого элемента, как `Button`?
8. Как обработчики событий прикрепляются к объекту с помощью XAML?
9. Что делают стили XAML?
10. Где можно определить ресурсы?

Упражнение 19.2. Дополнительные ресурсы

Воспользуйтесь ссылками на странице https://github.com/markjprice/cs10dotnet6/blob/main/book-links.md#chapter-19---building-mobile-and-desktop-apps-using-net-maui, чтобы получить дополнительную информацию по темам, приведенным в данной главе.

Резюме

В этой главе вы узнали, как создать кросс-платформенное мобильное и настольное приложение с помощью .NET MAUI, которое использует данные из веб-сервиса.

В следующей главе вы научитесь защищать данные и файлы с помощью хеширования, шифрования цифровой подписи, аутентификации и авторизации.

20 Защита данных и приложений

Эта глава посвящена тому, как с помощью шифрования защищать данные от просмотра злоумышленниками, а также тому, как применять хеширование и цифровые подписи для защиты данных от вмешательства или повреждения.

В .NET Core 2.1 компания Microsoft представила криптографические API, основанные на `Span<T>`, для хеширования, генерации случайных чисел, генерации и обработки асимметричной подписи и шифрования *RSA* (Rivest — Shamir — Adleman, Ривест — Шамир — Адлеман).

Криптографические операции реализуются операционной системой, поэтому при исправлении уязвимости системы безопасности в операционной системе приложения .NET немедленно получают преимущество. Однако это означает, что эти .NET-приложения могут использовать только те функции, которые поддерживает ОС. О том, какие функции и какой операционной системой поддерживаются, вы можете прочитать на сайте https://docs.microsoft.com/en-us/dotnet/standard/security/cross-platform-cryptography.

В этой главе:

- терминология безопасности;
- шифрование и дешифрование данных;
- хеширование данных;
- подписывание данных;
- генерация случайных чисел;
- аутентификация и авторизация пользователей.

Внимание! Код в этой главе соответствует примитивным сценариям безопасности и приводится только для базовых образовательных целей. Вы не должны использовать ее код для разработки настоящих библиотек и приложений. Работайте только с профессионально написанными библиотеками, которые были созданы с использованием принципов безопасности и усилены для реального применения в соответствии с актуальными рекомендациями по обеспечению безопасности.

Терминология безопасности

Существует множество способов защиты данных; несколько самых популярных из них перечислены ниже. В этой главе вы ознакомитесь с их подробным описанием и практическими реализациями.

- *Шифрование и дешифрование* — двусторонний процесс преобразования читаемого текста в шифрованный и обратно.
- *Хеши* — односторонний процесс генерации хеш-значения для безопасного хранения паролей или обнаружения вредоносных изменений либо повреждений данных. Простые хеши не дожны использоваться для паролей. Вы должны применять PBKDF2, bcrypt или scrypt, поскольку они гарантируют, что не может быть двух входных данных, которые генерируют одинаковый хеш.
- *Цифровые подписи* — метод проверки источника поступивших данных, то есть что данные были получены из заявленного источника путем верификации цифровой подписи, которая была применена к некоторым данным, с помощью открытого ключа.
- *Аутентификация* — метод идентификации пользователя путем проверки его учетных данных.
- *Авторизация* — метод выдачи допуска на выполнение неких действий или работы с определенными данными путем проверки ролей или групп, к которым принадлежат пользователи.

Если безопасность имеет для вас значение (а так и должно быть!), то следует нанять опытного эксперта по безопасности, а не полагаться на советы из Интернета. Очень легко совершить простые ошибки и оставить свои приложения и данные уязвимыми до тех пор, пока не окажется слишком поздно!

Ключи и их размеры

В алгоритмах защиты часто используются *ключи*. Они представлены байтовыми массивами различного размера. Ключи используются для таких целей, как:

- шифрование и дешифрование: AES, 3DES, RC2, Rijndael, RSA;
- подписание и проверка: RSA, ECDSA, DSA;
- аутентификация и проверка подлинности сообщений: HMAC;
- согласование ключей: Diffie-Hellman, Elliptical Curve Diffie-Hellman.

Выберите больший размер ключа для более надежной защиты. Это чрезмерное упрощение, поскольку некоторые реализации RSA поддерживают до 16 384-битных ключей, на создание которых может потребоваться несколько дней, и в большинстве сценариев это будет лишним; 2048-битного ключа должно быть достаточно до 2030 года, после чего вам следует перейти на 3192-битные ключи.

Ключи для шифрования и дешифрования могут быть *симметричными* (также известны как *общие* или *секретные*, поскольку один и тот же ключ используется для шифрования и дешифрования и поэтому должен храниться в безопасности) и *асимметричными* (пара из открытого и закрытого ключей, в которой для шифрования используется открытый, а для дешифрования — только закрытый).

Алгоритмы шифрования с помощью симметричных ключей работают быстро и могут шифровать большие объемы потоковых данных. Алгоритмы шифрования с помощью асимметричных ключей функционируют медленно и могут шифровать только небольшие массивы байтов. Чаще всего асимметричные ключи используются для создания и проверки подписи.

В своих проектах применяйте оба способа, используя симметричное шифрование для защиты самих данных и асимметричное — для распространения симметричного ключа. По такому принципу работал криптографический протокол *Secure Sockets Layer (SSL)* 2.0 в Интернете в 1995 году. Сегодня то, что все еще часто называют SSL, на самом деле является механизмом *безопасности транспортного уровня* (Transport Layer Security, TLS), который использует согласование ключей, а не зашифрованные RSA сеансовые ключи.

Ключи в шифровании представлены массивами байтов разного размера.

Векторы инициализации и размеры блоков

Вполне вероятно, что при шифровании больших объемов данных повторяются некоторые из их фрагментов (последовательностей символов). Например, в английском тексте часто применяется последовательность символов `the`, которая каждый раз шифруется как `hQ2`. Умный хакер воспользовался бы этим и упростил бы себе работу по взлому шифра:

```
When the wind blew hard the umbrella broke.
5:s4&hQ2aj#D f9d1d£8fh"&hQ2s0)an DF8SFd#][1
```

Мы можем избежать повторения последовательностей, разделив данные на *блоки*. После шифрования блока из него генерируется значение массива байтов, которое

передается в следующий блок в целях настройки алгоритма. Следующий блок шифруется таким образом, что выходные данные отличаются даже для того же входа, что и в предыдущем блоке. Зашифровать первый блок можно при наличии массива байтов для выполнения задачи. Это так называемый *вектор инициализации* (initialization vector, IV).

Вектор инициализации должен:

- генерироваться случайным образом вместе с каждым зашифрованным сообщением;
- передаваться вместе с зашифрованным сообщением;
- сам по себе не быть секретом.

Соли

Соль представляет собой случайный массив байтов, который используется как дополнительный ввод для односторонней хеш-функции. Если вы не применяете соль при генерации хешей, то при условии, что многие из ваших пользователей регистрируются, указывая `123456` в качестве пароля (по данным на 2016 год, примерно 8 % пользователей так и делают!), все они имеют одно и то же хеш-значение и их учетная запись будет уязвима для внешнего доступа через подбор пароля по словарю.

Когда пользователь регистрируется, соль должна генерироваться случайным образом и конкатенироваться с указанным пользователем паролем до того, как будет хеширована. Результат этой операции (но не исходный пароль) сохраняется с солью в базе данных.

Когда пользователь авторизуется в системе и вводит пароль, вы просматриваете соль, объединяете ее с введенным паролем, восстанавливаете хеш и затем сравниваете значение с хешем, хранящимся в базе данных. Если значения совпадают, то пароль введен верно.

Даже «соления» паролей недостаточно для действительно безопасного хранения. Вы должны поработать гораздо больше, например сделать PBKDF2, bcrypt или scrypt, но такая работа выходит за рамки этой книги.

Генерация ключей и векторов инициализации

Ключи и векторы инициализации представляют собой массивы байтов. Обеим сторонам, которые хотят обмениваться зашифрованными данными, необходимы значения ключей и векторов инициализации, однако надежный обмен байтовыми массивами может быть затруднен.

Вы можете надежно генерировать ключи и векторы инициализации, используя стандарт *формирования ключа на основе пароля* (password-based key derivation function, PBKDF2). Прекрасно подойдет класс `Rfc2898DeriveBytes`, который принимает пароль, соль, счетчик итераций и алгоритм хеширования (по умолчанию используется SHA-1, который больше не рекомендуется). Затем он генерирует ключи и векторы инициализации, вызывая метод `GetBytes`. Количество итераций — это то, сколько раз пароль хешируется в процессе работы. Чем больше итераций, тем сложнее взломать пароль.

Хотя класс `Rfc2898DeriveBytes` можно использовать для генерации IV и ключа, IV должен генерироваться каждый раз случайным образом и передаваться вместе с зашифрованным сообщением в виде открытого текста, поскольку оно не обязательно должно быть секретным.

Размер соли должен составлять 8 байт или больше, а количество итераций должно быть таким, чтобы для генерации ключа и IV для алгоритма шифрования на целевой машине требовалось около 100 мс. Это значение будет увеличиваться с течением времени по мере совершенствования процессоров. В приведенном ниже примере кода мы используем 150 000, но это значение уже будет слишком низким для некоторых компьютеров к тому времени, когда вы будете это читать.

Шифрование и дешифрование данных

На платформе .NET доступно несколько алгоритмов шифрования.

В устаревшей .NET Framework некоторые алгоритмы реализуются операционной системой, и их имена имеют суффикс `CryptoServiceProvider` или `Cng`. Есть алгоритмы, реализованные в .NET BCL, и их имена имеют суффикс `Managed`.

В современном .NET все алгоритмы реализуются операционной системой. Если алгоритмы ОС сертифицированы федеральными стандартами обработки информации *(Federal Information Processing Standards, FIPS)*, то .NET будет использовать алгоритмы, сертифицированные FIPS.

Как правило, для получения экземпляра алгоритма вы всегда будете использовать абстрактный класс, такой как `Aes` и его метод `Create`, поэтому вам не нужно знать, применяете ли вы `CryptoServiceProvider` или `Managed`.

Одни алгоритмы используют симметричные ключи, а другие — асимметричные. Главный асимметричный алгоритм шифрования — RSA. Рон Ривест, Ади Шамир и Леонард Адлеман описали этот алгоритм в 1977 году. Аналогичный алгоритм был разработан в 1973 году Клиффордом Коксом, английским математиком, работавшим в Центре правительственной связи Британского разведывательного

управления, но был рассекречен только в 1997 году, поэтому Ривест, Шамир и Адлеман получили признание и увековечили свои имена в аббревиатуре RSA.

Алгоритмы симметричного шифрования используют `CryptoStream` для эффективного шифрования или дешифрования большого количества байтов. Асимметричные алгоритмы могут обрабатывать только небольшое количество байтов, хранящихся в байтовом массиве вместо потока.

Ниже перечислены наиболее распространенные алгоритмы симметричного шифрования. Они наследуются от абстрактного класса `SymmetricAlgorithm`:

- AES;
- DESCryptoServiceProvider;
- TripleDES;
- RC2CryptoServiceProvider;
- RijndaelManaged.

Если вам нужно написать код для расшифровки неких данных, отправленных внешней системой, то вам придется прибегнуть к тому алгоритму, который внешняя система использовала для шифрования данных. Если вам нужно отправить зашифрованные данные в систему, которая может дешифровать только с помощью специального алгоритма, то вы снова не сможете выбрать алгоритм.

Если же ваш код выполняет и шифрование, и дешифрование, то можете выбрать алгоритм, который наилучшим образом соответствует вашим требованиям к надежности, производительности и т. д.

Выберите симметричный алгоритм блочного шифрования (Advanced Encryption Standard, AES), основанный на алгоритме Rijndael, для симметричного шифрования. Выберите RSA (алгоритм шифрования с открытым ключом) для асимметричного шифрования. Не путайте RSA с DSA. Алгоритм цифровой подписи (Digital Signature Algorithm, DSA) не может зашифровать данные. Он может генерировать только хеши и подписи.

Симметричное шифрование с помощью алгоритма AES

Чтобы упростить повторное использование обеспечивающего безопасность кода в нескольких проектах, мы создадим статический класс `Protector` в своей библиотеке классов и затем будем ссылаться на него в консольном приложении.

Приступим!

1. Откройте редактор кода и создайте рабочую область/решение `Chapter20`.
2. Создайте проект консольного приложения с такими настройками:

   1) шаблон проекта: Console Application/`console`;
   2) файл и папка рабочей области/решения: `Chapter20`;
   3) файл и папка проекта: `EncryptionApp`.
3. Создайте библиотеку классов `CryptographyLib` в рабочей области/решении `Chapter20`:
   1) в программе Visual Studio настройте стартовый проект для решения в соответствии с текущим выбором;
   2) в программе Visual Studio Code выберите `EncryptionApp` в качестве активного проекта OmniSharp.
4. В проекте `CryptographyLib` переименуйте файл `Class1.cs` в `Protector.cs`.
5. В проект `EncryptionApp` добавьте проектную ссылку на библиотеку `CryptographyLib`, как показано ниже:

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

6. Соберите проект `EncryptionApp` и убедитесь в отсутствии ошибок компиляции.
7. Откройте файл `Protector.cs` и измените его содержимое так, чтобы определить статический класс `Protector` с полями для хранения массива байтов соли и большого количества итераций, а также методами `Encrypt` и `Decrypt`:

```
using System.Diagnostics;
using System.Security.Cryptography;
using System.Security.Principal;
using System.Text;
using System.Xml.Linq;

using static System.Console;
using static System.Convert;

namespace Packt.Shared
{
  public static class Protector
  {
    // размер соли должен быть не менее 8 байт, мы будем использовать 16 байт
    private static readonly byte[] salt =
      Encoding.Unicode.GetBytes("7BANANAS");

    // количество итераций должно быть достаточно высоким, чтобы
    // на генерацию ключа и IV на целевой машине уходило не менее 100 мс.
    // 150 000 итераций занимают 139 мс на моем процессоре Intel
    // Core i7-1165G7 11-го поколения с тактовой частотой 2,80 ГГерц.
    private static readonly int iterations = 150_000;
```

```
public static string Encrypt(
  string plainText, string password)
{
  byte[] encryptedBytes;
  byte[] plainBytes = Encoding.Unicode.GetBytes(plainText);

  using (Aes aes = Aes.Create()) // фабричный метод абстрактных классов
  {
    // пишем, сколько времени требуется для генерации ключа и IV
    Stopwatch timer = Stopwatch.StartNew();

    using (Rfc2898DeriveBytes pbkdf2 = new(
      password, salt, iterations))
    {
      aes.Key = pbkdf2.GetBytes(32); // устанавливаем 256-битный ключ
      aes.IV = pbkdf2.GetBytes(16); // устанавливаем 128-битный IV
    }

    timer.Stop();

    WriteLine("{0:N0} milliseconds to generate Key and IV using {1:N0}
    iterations.",
      arg0: timer.ElapsedMilliseconds,
      arg1: iterations);

    using (MemoryStream ms = new())
    {
      using (ICryptoTransform transformer = aes.CreateEncryptor())
      {
        using (CryptoStream cs = new(
          ms, transformer, CryptoStreamMode.Write))
        {
          cs.Write(plainBytes, 0, plainBytes.Length);
        }
      }
      encryptedBytes = ms.ToArray();
    }
  }

  return ToBase64String(encryptedBytes);
}

public static string Decrypt(
  string cipherText, string password)
{
  byte[] plainBytes;
  byte[] cryptoBytes = FromBase64String(cipherText);

  using (Aes aes = Aes.Create())
  {
    using (Rfc2898DeriveBytes pbkdf2 = new(
```

```
          password, salt, iterations))
        {
          aes.Key = pbkdf2.GetBytes(32);
          aes.IV = pbkdf2.GetBytes(16);
        }

        using (MemoryStream ms = new())
        {
          using (ICryptoTransform transformer = aes.CreateDecryptor())
          {
            using (CryptoStream cs = new(
              ms, transformer, CryptoStreamMode.Write))
            {
              cs.Write(cryptoBytes, 0, cryptoBytes.Length);
            }
          }
          plainBytes = ms.ToArray();
        }
      }

      return Encoding.Unicode.GetString(plainBytes);
    }
  }
}
```

Обратите внимание на следующие моменты:

- хотя размер соли и количество итераций могут быть жестко закодированными (но их предпочтительнее хранить в самом сообщении), пароль *должен* передаваться в качестве параметра во время выполнения при вызове методов `Encrypt` и `Decrypt`;
- в примере используется временный тип `MemoryStream` для хранения результатов шифрования и дешифрования, а затем вызывается метод `ToArray` для преобразования потока в массив байтов;
- в примере зашифрованные массивы байтов переводятся в кодировку Base64 и из нее, чтобы упростить их чтение для человека.

Никогда не кодируйте жестко пароль в исходном коде, поскольку даже после компиляции пароль в сборке можно прочитать с помощью инструментов дизассемблера.

8. В проекте `EncryptionApp` откройте файл `Program.cs`, а затем импортируйте пространство имен для класса `Protector`, пространство имен для класса `CryptographicException` и статически импортируйте класс `Console`:

```
using System.Security.Cryptography; // CryptographicException
using Packt.Shared; // Protector

using static System.Console;
```

9. В файле `Program.cs` добавьте следующие операторы, которые позволяют запросить у пользователя сообщение и пароль, а затем выполняют шифрование и дешифрование:

```
Write("Enter a message that you want to encrypt: ");
string? message = ReadLine();

Write("Enter a password: ");
string? password = ReadLine();

if ((password is null) || (message is null))
{
  WriteLine("Message or password cannot be null.");
  return;
}

string cipherText = Protector.Encrypt(message, password);

WriteLine($"Encrypted text: {cipherText}");

Write("Enter the password: ");
string? password2Decrypt = ReadLine();

if (password2Decrypt is null)
{
  WriteLine("Password to decrypt cannot be null.");
  return;
}

try
{
  string clearText = Protector.Decrypt(cipherText, password2Decrypt);
  WriteLine($"Decrypted text: {clearText}");
}
catch (CryptographicException ex)
{
  WriteLine("{0}\nMore details: {1}",
    arg0: "You entered the wrong password!",
    arg1: ex.Message);
}
catch (Exception ex)
{
  WriteLine("Non-cryptographic exception: {0}, {1}",
    arg0: ex.GetType().Name,
    arg1: ex.Message);
}
```

10. Запустите код, попробуйте ввести сообщение и пароль для шифрования, введите тот же пароль для расшифровки и проанализируйте результат:

```
Enter a message that you want to encrypt: Hello Bob
Enter a password: secret
139 milliseconds to generate Key and IV using 150,000 iterations.
Encrypted text: eWt8sgL7aSt5DC9g74ONEPO7mjd55lXB/MmCZpUsFE0=
Enter the password: secret
Decrypted text: Hello Bob
```

Если ваш вывод показывает количество миллисекунд меньше 100, то увеличивайте количество итераций до тех пор, пока количество миллисекунд не станет 100 или больше. Обратите внимание, что другое количество итераций повлияет на хешированное значение, поэтому оно будет выглядеть иначе, чем показано выше.

11. Перезапустите код и вновь введите сообщение и пароль для шифрования, только на этот раз указав в пароле для расшифровки намеренную ошибку. Проанализируйте результат:

```
Enter a message that you want to encrypt: Hello Bob
Enter a password: secret
134 milliseconds to generate Key and IV using 150,000 iterations.
Encrypted text: eWt8sgL7aSt5DC9g74ONEPO7mjd55lXB/MmCZpUsFE0=
Enter the password: 123456
You entered the wrong password!
More details: Padding is invalid and cannot be removed.
```

Для поддержки будущих обновлений шифрования записывайте информацию о том, что вы выбрали, например AES-256, режим CBC с подстановкой PKCS#7, PBKDF2 и его алгоритм хеширования и количество итераций. Это известно как криптографическая гибкость.

Хеширование данных

На платформе .NET доступно несколько алгоритмов хеширования. Одни не требуют использования каких-либо ключей, вторым необходимы симметричные ключи, а третьим — асимметричные.

При выборе алгоритма хеширования следует учитывать два важных фактора:

- *устойчивость к коллизиям* — как часто два разных ввода могут иметь один и тот же хеш;
- *устойчивость к нахождению прообраза* — в отношении хеша, насколько сложно обнаружить другой ввод, который имеет идентичный хеш.

В табл. 20.1 представлены некоторые универсальные алгоритмы хеширования без ключа.

Таблица 20.1. Типичные алгоритмы хеширования без ключа

Алгоритм	Размер хеша	Описание
MD5	16 байт	Используется чаще всего, поскольку быстр в работе, но неустойчив к коллизиям
SHA1	20 байт	Применение алгоритма SHA1 в Интернете не рекомендуется с 2011 года
SHA256 SHA384 SHA512	32 байта 48 байт 64 байта	Алгоритмы из семейства «Безопасный алгоритм хеширования 2-го поколения» (SHA2, Secure Hashing Algorithm) с разными размерами хешей

Старайтесь не использовать алгоритмы MD5 и SHA1, поскольку у них выявлены существенные недостатки. Выбирайте алгоритм с бóльшим размером хеша, чтобы уменьшить вероятность повторения хешей. Первая известная коллизия алгоритма MD5 произошла в 2010 году, а первая известная коллизия алгоритма SHA1 — в 2017-м. Более подробно можно прочитать на сайте https://arstechnica.co.uk/information-technology/2017/02/at-deaths-door-for-years-widely-used-sha1-function-is-now-dead/.

Хеширование с помощью алгоритма SHA256

Добавим класс для представления пользователя, хранящегося в памяти, файле или базе данных. Мы будем использовать словарь для хранения нескольких пользователей в памяти.

1. В проекте библиотеки классов `CryptographyLib` добавьте файл класса `User.cs` и задайте ему три свойства для хранения имени пользователя, случайного значения соли, а также его соленого и хешированного пароля:

```
namespace Packt.Shared;

public class User
{
  public string Name { get; set; }
  public string Salt { get; set; }
  public string SaltedHashedPassword { get; set; }

  public User(string name, string salt,
    string saltedHashedPassword)
  {
```

```
    Name = name;
    Salt = salt;
    SaltedHashedPassword = saltedHashedPassword;
  }
}
```

2. В класс `Protector` добавьте следующие операторы, чтобы объявить словарь для хранения в памяти информации о нескольких пользователях. Применяются два метода: для регистрации нового пользователя и для проверки введенного пароля при последующей авторизации:

```
private static Dictionary<string, User> Users = new();

public static User Register(
  string username, string password)
{
  // генерируем случайную соль
  RandomNumberGenerator rng = RandomNumberGenerator.Create();
  byte[] saltBytes = new byte[16];
  rng.GetBytes(saltBytes);
  string saltText = ToBase64String(saltBytes);

  // генерируем соленый и хешированный пароль
  string saltedhashedPassword = SaltAndHashPassword(password, saltText);

  User user = new(username, saltText, saltedhashedPassword);

  Users.Add(user.Name, user);

  return user;
}

// проверяем пароль пользователя, который хранится
// в приватном статическом словаре Users
public static bool CheckPassword(string username, string password)
{
  if (!Users.ContainsKey(username))
  {
    return false;
  }

  User u = Users[username];

  return CheckPassword(password,
    u.Salt, u.SaltedHashedPassword);
}

// проверяем пароль пользователя, используя соль и хешированный пароль
public static bool CheckPassword(string password,
  string salt, string hashedPassword)
{
```

```
    // повторно генерируем соленый и хешированный пароль
    string saltedhashedPassword = SaltAndHashPassword(
      password, salt);

    return (saltedhashedPassword == hashedPassword);
  }

  private static string SaltAndHashPassword(string password, string salt)
  {
    using (SHA256 sha = SHA256.Create())
    {
      string saltedPassword = password + salt;
      return ToBase64String(sha.ComputeHash(
        Encoding.Unicode.GetBytes(saltedPassword)));
    }
  }
```

3. Откройте редактор кода и создайте консольное приложение `HashingApp` в рабочей области/решении `Chapter20`.
4. В программе Visual Studio Code выберите `HashingApp` в качестве активного проекта OmniSharp.
5. В проекте `HashingApp` добавьте ссылку на проект `CryptographyLib`.
6. Соберите проект `HashingApp` и убедитесь в отсутствии ошибок компиляции.
7. В файле `Program.cs` импортируйте пространство имен `Packt.Shared`.
8. В файле `Program.cs` добавьте операторы для регистрации пользователя и запроса регистрации другого пользователя, а также запроса авторизации под одним из логинов и проверки пароля:

```
WriteLine("Registering Alice with Pa$$w0rd:");
User alice = Protector.Register("Alice", "Pa$$w0rd");

WriteLine($"  Name: {alice.Name}");
WriteLine($"  Salt: {alice.Salt}");
WriteLine("  Password (salted and hashed): {0}",
  arg0: alice.SaltedHashedPassword);
WriteLine();

Write("Enter a new user to register: ");
string? username = ReadLine();

Write($"Enter a password for {username}: ");
string? password = ReadLine();

if ((username is null) || (password is null))
{
  WriteLine("Username or password cannot be null.");
  return;
}
```

```
WriteLine("Registering a new user:");
User newUser = Protector.Register(username, password);
WriteLine($" Name: {newUser.Name}");
WriteLine($" Salt: {newUser.Salt}");
WriteLine(" Password (salted and hashed): {0}",
  arg0: newUser.SaltedHashedPassword);
WriteLine();

bool correctPassword = false;

while (!correctPassword)
{
  Write("Enter a username to log in: ");
  string? loginUsername = ReadLine();

  Write("Enter a password to log in: ");
  string? loginPassword = ReadLine();

  if ((loginUsername is null) || (loginPassword is null))
  {
    WriteLine("Login username or password cannot be null.");
    return;
  }

  correctPassword = Protector.CheckPassword(
    loginUsername, loginPassword);

  if (correctPassword)
  {
    WriteLine($"Correct! {loginUsername} has been logged in.");
  }
  else
  {
    WriteLine("Invalid username or password. Try again.");
  }
}
```

9. Запустите код, зарегистрируйте нового пользователя с тем же паролем, что и у `Alice` (Алисы), и проанализируйте результат:

```
Registering Alice with Pa$$w0rd:
  Name: Alice
  Salt: I1I1dzIjkd7EYDf/6jaf4w==
  Password (salted and hashed): pIoadjE4W/XaRFkqS3br3UuAuPv/3LVQ8kzj6mvcz+s=

Enter a new user to register: Bob
Enter a password for Bob: Pa$$w0rd
Registering a new user:
  Name: Bob
  Salt: 1X7ym/UjxTiuEWBC/vIHpw==
  Password (salted and hashed):
```

```
DoBFtDhKeN0aaaLVdErtrZ3mpZSvpWDQ9TXDosTq0sQ=

Enter a username to log in: Alice
Enter a password to log in: secret
Invalid username or password. Try again.
Enter a username to log in: Bob
Enter a password to log in: secret
Invalid username or password. Try again.
Enter a username to log in: Bob
Enter a password to log in: Pa$$w0rd
Correct! Bob has been logged in.
```

Даже если оба пользователя зарегистрируются с одним и тем же паролем, соли будут генерироваться случайным образом, поэтому соленые и хешированные пароли этих пользователей будут различаться.

Подписывание данных

В качестве доказательства, что полученные данные на самом деле присланы доверенным отправителем, используются цифровые подписи. Вы подписываете не сами данные, а только их хеш, поскольку все алгоритмы подписи сначала хешируют данные в качестве шага реализации. Они также позволяют сократить этот шаг и предоставить уже хешированные данные.

Для генерации и подписывания хеша мы воспользуемся сочетанием алгоритмов RSA и SHA256.

Мы могли бы применить алгоритм DSA как для хеширования, так и для подписывания. DSA генерирует подпись быстрее, чем RSA, но медленнее проверяет ее. Поскольку подпись генерируется один раз, но проверяется многократно, лучше проводить быстрее проверку, а не генерацию.

Сегодня DSA используется редко. Улучшенным эквивалентом является эллиптическая кривая DSA (Elliptic Curve DSA). Хотя ECDSA работает медленнее, чем RSA, она генерирует более короткую подпись с тем же уровнем безопасности.

Подписывание с помощью алгоритмов SHA256 и RSA

Рассмотрим подписывание данных и проверку подписи с помощью алгоритма шифрования с открытым ключом.

1. В классе `Protector` добавьте следующие операторы, чтобы объявить поле для хранения открытого ключа в виде значения `string`, а также два метода для генерации и проверки подписи:

```
public static string? PublicKey;

public static string GenerateSignature(string data)
{
  byte[] dataBytes = Encoding.Unicode.GetBytes(data);
  SHA256 sha = SHA256.Create();
  byte[] hashedData = sha.ComputeHash(dataBytes);
  RSA rsa = RSA.Create();

  PublicKey = rsa.ToXmlString(false); // exclude private key

  return ToBase64String(rsa.SignHash(hashedData,
    HashAlgorithmName.SHA256, RSASignaturePadding.Pkcs1));
}

public static bool ValidateSignature(
  string data, string signature)
{
  if (PublicKey is null) return false;
  byte[] dataBytes = Encoding.Unicode.GetBytes(data);
  SHA256 sha = SHA256.Create();
  byte[] hashedData = sha.ComputeHash(dataBytes);
  byte[] signatureBytes = FromBase64String(signature);
  RSA rsa = RSA.Create();
  rsa.FromXmlString(PublicKey);
  return rsa.VerifyHash(hashedData, signatureBytes,
    HashAlgorithmName.SHA256, RSASignaturePadding.Pkcs1);
}
```

Обратите внимание на следующие моменты:

- перед использованием кода проверки подписи должен быть доступен только открытый ключ из пары «открытый — закрытый ключ», чтобы при вызове метода `ToXmlString` мы могли передать значение `false`. Закрытый ключ требуется для подписи данных и должен храниться в секрете, поскольку любой, кто имеет доступ к этому ключу, может подписывать данные от вашего имени!
- алгоритм хеширования, применяемый для генерации хеша из данных с помощью вызова метода `SignHash`, должен соответствовать алгоритму хеширования, используемому при вызове метода `VerifyHash`. В вышеприведенном примере мы задействовали алгоритм `SHA256`.

Теперь мы можем протестировать подписывание каких-нибудь данных и проверку подписи.

2. Откройте редактор кода и создайте консольное приложение `SigningApp` в рабочей области/решении `Chapter20`.
3. В программе Visual Studio Code выберите `SigningApp` в качестве активного проекта OmniSharp.
4. В проекте `SigningApp` добавьте ссылку на проект `CryptographyLib`.

5. Соберите проект `SigningApp` и убедитесь в отсутствии ошибок компиляции.
6. В файле `Program.cs` импортируйте пространство имен `Packt.Shared`.
7. Добавьте следующие операторы, чтобы предложить пользователю ввести некий текст, подписать его, проверить подпись, а затем изменить ее и снова проверить, чтобы намеренно вызвать несоответствие:

```
Write("Enter some text to sign: ");
string? data = ReadLine();

string signature = Protector.GenerateSignature(data);

WriteLine($"Signature: {signature}");
WriteLine("Public key used to check signature:");
WriteLine(Protector.PublicKey);

if (Protector.ValidateSignature(data, signature))
{
  WriteLine("Correct! Signature is valid.");
}
else
{
  WriteLine("Invalid signature.");
}

// имитируем поддельную подпись, заменив первый символ на X или Y
string fakeSignature = signature.Replace(signature[0], 'X');
if (fakeSignature == signature)
{
  fakeSignature = signature.Replace(signature[0], 'Y');
}

if (Protector.ValidateSignature(data, fakeSignature))
{
  WriteLine("Correct! Signature is valid.");
}
else
{
  WriteLine($"Invalid signature: {fakeSignature}");
}
```

8. Запустите код и введите любой текст, как показано в следующем выводе (отредактированном по длине):

```
Enter some text to sign: The cat sat on the mat.
Signature: BXSTdM...4Wrg==
Public key used to check signature:
<RSAKeyValue><Modulus>nHtwl3...mw3w==</Modulus><Exponent>AQAB</Exponent></
RSAKeyValue>
Correct! Signature is valid.
Invalid signature: XXSTdM...4Wrg==
```

Генерация случайных чисел

Иногда необходимо сгенерировать случайные числа, возможно при создании игры, имитирующей бросок игральных костей, либо для дальнейшего использования с криптографией при шифровании или подписывании. В .NET есть несколько классов, генерирующих случайные числа.

Генерация случайных чисел для игр и подобных приложений

В сценариях, не требующих истинно случайных чисел, например в играх, вы можете создать экземпляр класса `Random`, как показано в примере ниже:

```
Random r = new();
```

В конструкторе класса `Random` содержится параметр для указания начального значения, используемого для инициализации генератора псевдослучайных чисел:

```
Random r = new(Seed: 46378);
```

Как вы помните из главы 2, имена параметров необходимо задавать с помощью так называемого верблюжьего регистра. Разработчик, определивший конструктор класса `Random`, нарушил это соглашение! Имя параметра следовало задать как `seed`, а не `Seed`.

Общие начальные значения работают как секретные ключи, так что если вы воспользуетесь одинаковым алгоритмом генерации случайных чисел с одинаковым начальным значением в двух приложениях, то они могут генерировать одинаковые последовательности «случайных» чисел. Иногда это необходимо, например при синхронизации GPS-приемника со спутником. Но обычно следует держать начальное значение в секрете.

Создав объект `Random`, вы можете вызывать его методы для генерации случайных чисел, как показано в следующих примерах кода:

```
// minValue – включающая нижняя граница, то есть 1 – возможное значение
// maxValue – исключающая верхняя граница, то есть 7 – невозможное значение
int dieRoll = r.Next(minValue: 1, maxValue: 7); // возвращает от 1 до 6

double randomReal = r.NextDouble(); // возвращает от 0,0 до менее 1,0

byte[] arrayOfBytes = new byte[256];
r.NextBytes(arrayOfBytes); // 256 случайных байт в массиве
```

Метод `Next` принимает два параметра: `minValue` и `maxValue`. Параметр `maxValue` — не максимальное значение, возвращаемое методом! Это *эксклюзивная* верхняя

граница, то есть число, на единицу превышающее максимальное значение. Аналогичным образом, значение, возвращаемое методом `NextDouble`, больше или равно `0.0` и меньше `1.0`.

Генерация случайных чисел для криптографии

Класс `Random` генерирует криптографически слабые *псевдослучайные* числа. Для криптографии его недостаточно! Если случайные числа — неслучайные, то они будут предсказуемыми и взломщик может сломать вашу защиту.

Для криптографически сильных псевдослучайных чисел необходимо использовать тип, наследующий от `RandomNumberGenerator`, например `RNGCryptoServiceProvider`.

Создадим метод для генерации истинно случайного байтового массива, который можно использовать в таких алгоритмах, как шифрование, для получения ключей и значений вектора инициализации.

1. В классе `Protector` добавьте следующие операторы, чтобы определить метод, генерирующий случайный ключ или вектор инициализации для использования в шифровании:

```
public static byte[] GetRandomKeyOrIV(int size)
{
  RandomNumberGenerator r = RandomNumberGenerator.Create();
  byte[] data = new byte[size];
  r.GetBytes(data);

  // data представляет собой массив, который теперь заполнен
  // криптографически стойкими случайными байтами
  return data;
}
```

   Теперь мы можем протестировать случайные байты, сгенерированные для истинно случайного ключа шифрования или вектора инициализации.

2. Откройте редактор кода и создайте консольное приложение `RandomizingApp` в рабочей области/решении `Chapter20`.
3. В программе Visual Studio Code выберите `RandomizingApp` в качестве активного проекта для OmniSharp.
4. В проекте `RandomizingApp` добавьте ссылку на проект `CryptographyLib`.
5. Соберите проект `RandomizingApp` и убедитесь в отсутствии ошибок компиляции.
6. В файле `Program.cs` импортируйте пространство имен `Packt.Shared`.
7. Добавьте следующие операторы, чтобы предложить пользователю ввести размер байтового массива, а затем сгенерируйте случайные байтовые значения и запишите их в консоль:

```
Write("How big do you want the key (in bytes): ");
string? size = ReadLine();

byte[] key = Protector.GetRandomKeyOrIV(int.Parse(size));

WriteLine($"Key as byte array:");
for (int b = 0; b < key.Length; b++)
{
  Write($"{key[b]:x2} ");
  if (((b + 1) % 16) == 0) WriteLine();
}
WriteLine();
```

8. Запустите код, введите размерность ключа, например, 256 и проанализируйте случайно сгенерированный ключ:

```
How big do you want the key (in bytes): 256
Key as byte array:
f1 57 3f 44 80 e7 93 dc 8e 55 04 6c 76 6f 51 b9
e8 84 59 e5 8d eb 08 d5 e6 59 65 20 b1 56 fa 68
...
```

Аутентификация и авторизация пользователей

Аутентификация — это процесс проверки подлинности личности пользователя путем проверки его учетных данных через некоторый удостоверяющий центр. Учетные данные могут включать в себя имя пользователя и пароль, или отпечаток пальца, или скан лица. Проведя аутентификацию, удостоверяющий центр может выдавать утверждения о пользователе, например, каков его адрес электронной почты и к каким группам или ролям он принадлежит.

Авторизация — это процесс определения принадлежности группе или роли, проводимый перед выдачей доступа к таким ресурсам, как функции приложения или его данные. Хотя авторизация может основываться на индивидуальной идентификации, рекомендуется проводить ее на основе принадлежности роли или группе (что можно указать в утверждениях), даже если в группе или в роли имеется только один пользователь, поскольку это позволяет в дальнейшем менять информацию о принадлежности пользователя, не прибегая к повторному назначению индивидуальных прав доступа.

Например, вместо того, чтобы назначать права доступа к Букингемскому дворцу *Елизавете Александре Марии Виндзор* (пользователю), вы можете назначить права доступа *Монарху Соединенного Королевства Великобритании и Северной Ирландии и других королевств и территорий* (роль), а затем добавить Елизавету в качестве единственного участника этой роли. Тогда в какой-то момент в будущем вам не нужно будет изменять права доступа для роли *Монарх*; вы просто удалите Елизавету и добавите следующего человека в порядок престолонаследования. И конечно же, вы реализуете этот порядок в виде очереди.

Механизмы аутентификации и авторизации

Существует несколько механизмов аутентификации и авторизации. Все они реализуют пару интерфейсов `IIdentity` и `IPrincipal` в пространстве имен `System.Security.Principal`.

Идентификация пользователя

Интерфейс `IIdentity` представляет пользователя, поэтому имеет свойства `Name` и `IsAuthenticated`, которые указывают, является ли пользователь анонимным или успешно прошедшим аутентификацию по своим учетным данным:

```
namespace System.Security.Principal
{
  public interface IIdentity
  {
    string? AuthenticationType { get; }
    bool IsAuthenticated { get; }
    string? Name { get; }
  }
}
```

Распространенным классом, реализующим этот интерфейс, является `GenericIdentity`, который наследуется от класса `ClaimsIdentity`:

```
namespace System.Security.Principal
{
  public class GenericIdentity : ClaimsIdentity
  {
    public GenericIdentity(string name);
    public GenericIdentity(string name, string type);
    protected GenericIdentity(GenericIdentity identity);
    public override string AuthenticationType { get; }
    public override IEnumerable<Claim> Claims { get; }
    public override bool IsAuthenticated { get; }
    public override string Name { get; }
    public override ClaimsIdentity Clone();
  }
}
```

Объекты `Claim` имеют свойство `Type`, указывающее, к чему относится утверждение: к имени пользователя, его принадлежности к роли или группе, к дате рождения и т. д.:

```
namespace System.Security.Claims
{
  public class Claim
  {
    // различные конструкторы
```

```
    public string Type { get; }
    public ClaimsIdentity? Subject { get; }
    public IDictionary<string, string> Properties { get; }
    public string OriginalIssuer { get; }
    public string Issuer { get; }
    public string ValueType { get; }
    public string Value { get; }
    protected virtual byte[]? CustomSerializationData { get; }
    public virtual Claim Clone();
    public virtual Claim Clone(ClaimsIdentity? identity);
    public override string ToString();
    public virtual void WriteTo(BinaryWriter writer);
    protected virtual void WriteTo(BinaryWriter writer, byte[]? userData);
  }

  public static class ClaimTypes
  {
    public const string Actor = "http://schemas.xmlsoap.org/ws/2009/09/
    identity/claims/actor";
    public const string NameIdentifier = "http://schemas.xmlsoap.org/
    ws/2005/05/identity/claims/nameidentifier";
    public const string Name = "http://schemas.xmlsoap.org/ws/2005/05/
    identity/claims/name";
    public const string PostalCode = "http://schemas.xmlsoap.org/ws/2005/05/
    identity/claims/postalcode";

    // ... многие другие строковые константы

    public const string MobilePhone = "http://schemas.xmlsoap.org/ws/2005/05/
    identity/claims/mobilephone";
    public const string Role = "http://schemas.microsoft.com/ws/2008/06/
    identity/claims/role";
    public const string Webpage = "http://schemas.xmlsoap.org/ws/2005/05/
    identity/claims/webpage";
  }
}
```

Принадлежность пользователя

Интерфейс `IPrincipal` служит для связи пользователя с ролями и группами, членом которых он является, поэтому `IPrincipal` можно использовать для целей авторизации:

```
namespace System.Security.Principal
{
  public interface IPrincipal
  {
    IIdentity? Identity { get; }
    bool IsInRole(string role);
  }
}
```

Текущий поток, выполняющий ваш код, содержит свойство `CurrentPrincipal`, которому может быть присвоен в качестве значения любой объект, реализующий интерфейс `IPrincipal`. В дальнейшем система будет обращаться к `CurrentPrincipal` за разрешением на выполнение защищенного действия.

Наиболее распространенным классом, реализующим этот интерфейс, является `GenericPrincipal`, который наследуется от `ClaimsPrincipal`:

```
namespace System.Security.Principal
{
  public class GenericPrincipal : ClaimsPrincipal
  {
    public GenericPrincipal(IIdentity identity, string[]? roles);
    public override IIdentity Identity { get; }
    public override bool IsInRole([NotNullWhen(true)] string? role);
  }
}
```

Реализация аутентификации и авторизации

Реализуем пример пользовательского механизма аутентификации и авторизации и рассмотрим эти процессы более подробно.

1. В проекте `CryptographyLib` добавьте свойство в класс `User` для хранения массива ролей:

   ```
   public string[]? Roles { get; set; }
   ```

2. В файле `User.cs` добавьте параметр для установки `Roles` в конструкторе.
3. Измените метод `Register` в классе `Protector` так, чтобы разрешить передачу массива ролей в качестве необязательного параметра, как показано в коде ниже (выделено жирным шрифтом):

   ```
   public static User Register(
     string username, string password,
     string[]? roles = null)
   ```

4. В методе `Register` добавьте параметр, чтобы установить массив ролей в новом объекте `User`:

   ```
   User user = new(username, saltText,
     saltedhashedPassword, roles);
   ```

5. В проекте `CryptographyLib` добавьте операторы в класс `Protector`, чтобы определить метод `LogIn` для входа в систему пользователя, и если имя пользователя и пароль действительны, то создайте общие идентификатор и приниципал

и назначьте их текущему потоку, указав, что тип аутентификации был пользовательским и назывался `PacktAuth`:

```
public static void LogIn(string username, string password)
{
  if (CheckPassword(username, password))
  {
    GenericIdentity gi = new(
      name: username, type: "PacktAuth");

    GenericPrincipal gp = new(
      identity: gi, roles: Users[username].Roles);

    // установка принципала в текущем потоке, чтобы он по умолчанию
    // использовался для авторизации
    Thread.CurrentPrincipal = gp;
  }
}
```

6. Откройте редактор кода и создайте консольное приложение `SecureApp` в рабочей области/решении `Chapter20`.
7. В программе Visual Studio Code выберите `SecureApp` в качестве активного проекта OmniSharp.
8. В проекте `SecureApp` добавьте ссылку на проект `CryptographyLib`.
9. Соберите проект `SecureApp` и убедитесь в отсутствии ошибок компиляции.
10. В файле `Program.cs` импортируйте необходимые пространства имен для работы с аутентификацией и авторизацией:

```
using Packt.Shared; // Protector
using System.Security; // SecurityException
using System.Security.Principal; // IPrincipal
using System.Security.Claims; // ClaimsPrincipal, Claim

using static System.Console;
```

11. Добавьте операторы для регистрации трех пользователей `Alice` (Алиса), `Bob` (Боб) и `Eve` (Ева), имеющих различные роли. Затем попросите пользователя войти в систему и выведите сведения о них:

```
Protector.Register("Alice", "Pa$$w0rd",
  roles: new[] { "Admins" });

Protector.Register("Bob", "Pa$$w0rd",
  roles: new[] { "Sales", "TeamLeads" });

// Ева не имеет никаких ролей
Protector.Register("Eve", "Pa$$w0rd");
```

```
// предлагаем пользователю ввести имя пользователя и пароль
// для входа под одним из этих трех пользователей

Write($"Enter your user name: ");
string? username = ReadLine();

Write($"Enter your password: ");
string? password = ReadLine();

if ((username == null) || (password == null))
{
  WriteLine("Username or password is null. Cannot login.");
  return;
}

Protector.LogIn(username, password);

if (Thread.CurrentPrincipal == null)
{
  WriteLine("Log in failed.");
  return;
}

IPrincipal p = Thread.CurrentPrincipal;

WriteLine(
  $"IsAuthenticated: {p.Identity?.IsAuthenticated}");
WriteLine(
  $"AuthenticationType: {p.Identity?.AuthenticationType}");
WriteLine($"Name: {p.Identity?.Name}");
WriteLine($"IsInRole(\"Admins\"): {p.IsInRole("Admins")}");
WriteLine($"IsInRole(\"Sales\"): {p.IsInRole("Sales")}");

if (p is ClaimsPrincipal)
{
  WriteLine(
    $"{p.Identity?.Name} has the following claims:");

  IEnumerable<Claim>? claims = (p as ClaimsPrincipal)?.Claims;

  if (claims is not null)
  {
    foreach (Claim claim in claims)
    {
      WriteLine($"{claim.Type}: {claim.Value}");
    }
  }
}
```

12. Запустите код, войдите в систему под именем `Alice` (Алиса) с паролем `Pa$$w0rd` и проанализируйте результат:

```
Enter your user name: Alice
Enter your password: Pa$$w0rd
IsAuthenticated: True
AuthenticationType: PacktAuth
Name: Alice
IsInRole("Admins"): True
IsInRole("Sales"): False
Alice has the following claims:
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name: Alice
http://schemas.microsoft.com/ws/2008/06/identity/claims/role: Admins
```

13. Запустите код, войдите в систему под именем `Alice` (Алиса) с паролем `secret` и проанализируйте результат:

```
Enter your user name: Alice
Enter your password: secret
Log in failed.
```

14. Запустите код, войдите в систему под именем `Bob` (Боб) с паролем `Pa$$word` и проанализируйте результат:

```
Enter your user name: Bob
Enter your password: Pa$$w0rd
IsAuthenticated: True
AuthenticationType: PacktAuth
Name: Bob
IsInRole("Admins"): False
IsInRole("Sales"): True
Bob has the following claims:
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name: Bob
http://schemas.microsoft.com/ws/2008/06/identity/claims/role: Sales
http://schemas.microsoft.com/ws/2008/06/identity/claims/role: TeamLeads
```

Способы защиты приложения

Рассмотрим, как с помощью авторизации можно предотвратить доступ некоторых пользователей к тем или иным функциям приложения.

1. В конце файла `Program.cs` добавьте метод, защищенный проверкой разрешения внутри метода, и сгенерируйте соответствующие исключения, если пользователь анонимный или не член роли `Admins`:

```
static void SecureFeature()
{
  if (Thread.CurrentPrincipal == null)
  {
    throw new SecurityException(
```

```
        "A user must be logged in to access this feature.");
    }

    if (!Thread.CurrentPrincipal.IsInRole("Admins"))
    {
      throw new SecurityException(
        "User must be a member of Admins to access this feature.");
    }

    WriteLine("You have access to this secure feature.");
  }
```

2. Над методом `SecureFeature` добавьте операторы для вызова метода `SecureFeature` в операторе `try`:

```
try
{
  SecureFeature();
}
catch (Exception ex)
{
  WriteLine($"{ex.GetType()}: {ex.Message}");
}
```

3. Запустите код, войдите в систему под именем `Alice` (Алиса) с паролем `Pa$$word` и проанализируйте результат:

```
You have access to this secure feature.
```

4. Запустите код, войдите в систему под именем `Bob` (Боб) с паролем `Pa$$word` и проанализируйте результат:

```
System.Security.SecurityException: User must be a member of Admins to
access this feature.
```

Аутентификация и авторизация в реальном мире

Хотя очень полезно увидеть примеры того, как могут работать аутентификация и авторизация, в реальном мире вам не следует создавать собственные системы безопасности, поскольку слишком велика вероятность того, что вы можете внести в них ошибки.

Вместо этого вам следует обратиться к коммерческим или открытым реализациям. Обычно они реализуют такие стандарты, как OAuth 2.0 и OpenID Connect. Популярным вариантом с открытым исходным кодом является *IdentityServer4*, но он будет поддерживаться только до ноября 2022 года. Полукоммерческий вариант — Duende IdentityServer.

Официальная позиция Microsoft заключается в следующем: «У Microsoft уже есть команда и продукт в этой области, Azure Active Directory, который позволяет бесплатно использовать 500 000 объектов». Более подробно об этом вы можете прочитать по следующей ссылке: https://devblogs.microsoft.com/aspnet/asp-net-core-6-and-authentication-servers/.

Практические задания

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

Упражнение 20.1. Проверочные вопросы

Ответьте на следующие вопросы.

1. Какой из алгоритмов шифрования, доступных на платформе .NET, лучше всего подойдет для симметричного шифрования?
2. Какой из алгоритмов шифрования, доступных на платформе .NET, лучше всего подойдет для асимметричного шифрования?
3. Что такое радужная атака?
4. При использовании алгоритмов шифрования лучше применять блоки большого или малого размера?
5. Что такое криптографическое хеширование данных?
6. Что такое криптографическая подпись?
7. В чем разница между симметричным и асимметричным шифрованием?
8. Что такое RSA?
9. Почему пароли должны быть засолены перед сохранением?
10. SHA-1 — это алгоритм хеширования, разработанный Национальным агентством безопасности США. Почему его не следует использовать?

Упражнение 20.2. Защита данных с помощью шифрования и хеширования

В рабочей области/решении `Chapter10` создайте консольное приложение `Exercise02`, предназначенное для защиты конфиденциальных данных, таких как номер кредитной карты или пароль, хранящиеся в XML-файле, как показано в следующем примере:

```
<?xml version="1.0" encoding="utf-8" ?>
<customers>
  <customer>
    <name>Bob Smith</name>
    <creditcard>1234-5678-9012-3456</creditcard>
    <password>Pa$$w0rd</password>
  </customer>
  ...
</customers>
```

Обратите внимание: номер банковской карты и пароль клиента хранятся в открытом виде. Номер карты должен быть зашифрован, чтобы ее можно было расшифровать и использовать позже, а пароль следует засолить и хешировать.

Вы не должны хранить номера кредитных карт в своих приложениях. Это лишь пример секретных данных, которые вы, возможно, захотите защитить. Если вам приходится хранить номера кредитных карт, то вы должны приложить гораздо больше усилий, чтобы соответствовать требованиям индустрии платежных карт (Payment Card Industry, PCI).

Упражнение 20.3. Дешифрование данных

В рабочей области/решении `Chapter20` создайте консольное приложение `Exercise03`, открывающее XML-файл, защитой которого вы занимались в предыдущем упражнении, и расшифровывающее номер банковской карты.

Упражнение 20.4. Дополнительные ресурсы

Воспользуйтесь ссылками на странице https://github.com/markjprice/cs10dotnet6/blob/main/book-links.md#chapter-20---protecting-your-data-and-applications, чтобы получить дополнительную информацию по темам, приведенным в данной главе.

Резюме

В этой главе вы узнали, как зашифровать и дешифровывать данные с помощью симметричного шифрования, генерировать соленый хеш, подписывать данные и проверять цифровые подписи, генерировать истинно случайные числа и использовать аутентификацию и авторизацию для защиты ваших приложений.

Приложение
Ответы на проверочные вопросы

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

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

1. Среда разработки Visual Studio 2022 превосходит Visual Studio Code?

   Ответ. Нет. Каждая из них оптимизирована для разных задач. Visual Studio 2022 для Windows — большая, тяжеловесная и может создавать приложения с графическими пользовательскими интерфейсами, например Windows Forms, WPF, UWP и приложения .NET MAUI, но доступна только на Windows. Visual Studio 2022 — это интерактивная среда разработки (Interactive Development Environment, IDE), а не редактор кода. Visual Studio Code меньше, она легковесная, ориентирована на код, поддерживает больше языков и является кроссплатформенной. Ожидается, что в ноябре 2022 года, с выходом .NET 7, Visual Studio Code получит расширение, которое облегчает создание пользовательских интерфейсов для приложений .NET MAUI с помощью паттерна проектирования MVU.

2. Платформа .NET 6 лучше .NET Framework?

   Ответ. Для современной разработки — да, но это зависит от того, что вам нужно. .NET 6 — это современная, кросс-платформенная, ориентированная на производительность версия устаревшего, зрелого .NET Framework. Ее чаще улучшают. .NET Framework лучше поддерживает устаревшие приложения; однако текущая версия 4.8 будет последним крупным выпуском. Она никогда не будет поддерживать некоторые языковые возможности C# 8 и более поздних версий.

3. Что такое .NET Standard и почему он все еще важен?

   Ответ. .NET Standard определяет API, также известный как контракт, который может реализовать платформа .NET. Последние версии .NET Framework, Xamarin и современной .NET реализуют .NET Standard 2.0, чтобы обеспечить

единый стандартный API, который разработчики могут повторно использовать любое количество раз. .NET Core 3.0 и более поздние версии, включая .NET 6, реализуют .NET Standard 2.1, в котором есть некоторые новые возможности, не поддерживаемые .NET Framework. Если вы хотите создать новую библиотеку классов, поддерживающую все платформы .NET, то она должна быть совместима с .NET Standard 2.0.

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

   Ответ. Платформа .NET Core поддерживает разные языки, поскольку для каждого из них есть компилятор, преобразующий исходный код в код на промежуточном языке (IL). Затем этот IL-код с помощью общеязыковой исполняющей среды (CLR) преобразуется в машинные инструкции для процессора CPU во время выполнения.

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

   Ответ. Метод `Main`. Рекомендуется использовать массив `string` для аргументов командной строки и тип возвращаемого значения `int`, хоть это и не обязательно. Метод может быть объявлен так, как показано в коде ниже:

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

6. Что такое программа верхнего уровня и как получить доступ к любым аргументам командной строки?

   Ответ. Программа верхнего уровня — это проект, в котором нет необходимости явно определять класс `Program` с точкой входа в метод `Main` с параметром `args` для доступа к любым аргументам командной строки. Они определяются для вас неявно, так что вы можете вводить операторы без шаблонного кода.

7. Что вы вводите в подсказке, чтобы создать и выполнить исходный код на языке C#?

   Ответ. В папке с файлом `.csproj` вы вводите `dotnet run`.

8. Каковы преимущества использования расширения .NET Interactive Notebooks для написания кода на языке C#?

   Ответ. К числу преимуществ можно отнести смешивание языков в одном документе и смешивание кода с разметкой для обогащенного текста.

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

   Ответ. На сайте Microsoft Docs. В частности, документация по ключевым словам C# приведена на этой странице: https://docs.microsoft.com/ru-ru/dotnet/articles/csharp/language-reference/keywords/.

10. Где найти решения общих проблем программирования?

    Ответ. На сайте https://stackoverflow.com/.

Глава 2. Говорим на языке C#

Упражнение 2.1. Проверочные вопросы

1. Какой оператор можно ввести в файл C#, чтобы узнать версию компилятора и языка?

   Ответ. Оператор `#error version`.

2. Каковы два типа комментариев в C#?

   Ответ. Однострочный комментарий с префиксом `//` и многострочный комментарий, начинающийся с `/*` и заканчивающийся `*/`.

3. В чем разница между дословной и интерполированной строкой?

   Ответ. Дословная строка начинается с символа `@`, и каждый символ (кроме `"`) интерпретируется как таковой; например, обратная косая черта `\` — это обратная косая черта `\`. Интерполированная строка начинается с символа `$` и может включать выражения, заключенные в фигурные скобки, например `{выражение}`.

4. Почему следует быть осторожными при использовании значений `float` и `double`?

   Ответ. Потому что их точность не гарантируется, особенно при выполнении сравнения на равенство.

5. Как определить, сколько байтов памяти использует такой тип, как `double`?

   Ответ. С помощью оператора `sizeof()`, например, `sizeof(double)`.

6. Когда следует использовать ключевое слово `var`?

   Ответ. Ключевое слово `var` нужно использовать только для объявления локальных переменных, когда вы не можете указать известный тип. Этим словом легко злоупотребить на начальном этапе написания кода, поскольку оно очень удобное, но его использование может усложнить последующее сопровождение кода.

7. Каков новейший способ создания экземпляра класса, такого как `XmlDocument`?

   Ответ. Этот способ заключается в использовании выражения `new` с целевым типом:

   ```
   XmlDocument doc = new();
   ```

8. Почему следует быть осторожными при использовании типа `dynamic`?

   Ответ. Потому что тип объекта, хранящегося в нем, не проверяется до времени выполнения, что может привести к возникновению исключений времени выполнения, если вы попытаетесь использовать член, который не существует в типе.

9. Как выровнять строку формата по правому краю?

   Ответ. После индекса или выражения добавьте запятую и целочисленное значение, чтобы указать ширину столбца, в пределах которого нужно выровнять

значение. Положительные целые числа означают выравнивание вправо, а отрицательные целые числа — выравнивание влево.

10. Какой символ разделяет аргументы в консольном приложении?

    Ответ. Символ пробела.

Упражнение 2.2. Проверочные вопросы о числовых типах

Какой тип следует выбрать для каждого указанного ниже числа?

1. Телефонный номер.

   Ответ. Тип `string`.

2. Рост.

   Ответ. Тип `float` или `double`.

3. Возраст.

   Ответ. Тип `int` для лучшей производительности на большинстве процессоров или `byte` (0–255) для наименьшего размера.

4. Размер оклада.

   Ответ. Тип `decimal`.

5. Международный стандартный книжный номер.

   Ответ. Тип `string`.

6. Цена книги.

   Ответ. Тип `decimal`.

7. Вес книги.

   Ответ. Тип `float` или `double`.

8. Размер населения страны.

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

9. Количество звезд во Вселенной.

   Ответ. Тип `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 50 000 сотрудников на предприятие).

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

Глава 3. Управление потоком исполнения, преобразование типов и обработка исключений

Упражнение 3.1. Проверочные вопросы

Ответьте на следующие вопросы.

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

   Ответ. При делении целого или дробного числа вызывается исключение `DivideByZeroException`.

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) или, в случае деления на `0`, либо `PositiveInfinity`, либо `NegativeInfinity`.

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

   Ответ. Она 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 операторе `x = y++;` текущее значение `y` будет присвоено `x`, а затем инкрементировано. В операторе `x = ++y;` значение `y` сначала будет инкрементировано, а затем результат будет присвоен `x`.

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

   Ответ. Оператор `break` завершит весь цикл и продолжит выполнять код после цикла, оператор `continue` завершит текущую итерацию цикла и продолжит выполнение в начале блока цикла для следующей итерации, а оператор `return` завершит текущий вызов метода и продолжит выполнение после вызова метода.

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

   Ответ. Три части оператора `for` — это выражения инициализатора, условия и инкремента. Выражение условия должно быть логическим выражением, которое возвращает значение `true` или `false`, а две другие части необязательны.

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

   Ответ. С помощью операции `=` можно присвоить значения переменным, а `==` — операция проверки на равенство, возвращающая значение `true` или `false`.

8. Будет ли скомпилирован следующий оператор?

```
for ( ; true; ) ;
```

Ответ. Да. Для оператора `for` требуется только выражение условия, возвращающее как `true`, так и `false`. Выражения инициализатора и инкремента необязательны и могут быть опущены. В данном сдучае оператор `for` никогда не прекратит свое выполнение. Это пример бесконечного цикла.

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

   Ответ. Возвращаемое по умолчанию значение.

10. Какой интерфейс должен реализовать объект, чтобы его можно было перечислить с помощью оператора `foreach`?

    Ответ. Интерфейс `IEnumerable`. Он должен иметь правильные методы с правильными подписями, даже если объект фактически не реализует интерфейс.

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 после достижения значения `255` из-за переполнения будет вызываться исключение:

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

Упражнение 3.5. Проверка знания операций

Каковы будут значения переменных `x` и `y` после выполнения следующих операторов?

1. Операции инкремента и сложения:

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

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

2. Операции бинарного сдвига:

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

Ответ. x равен 12; y равен 5.

3. Побитовые операции:

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

Ответ. x равен 8; y равен 15.

Глава 4. Разработка, отладка и тестирование функций

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

Ответ. Оно указывает на то, что метод не имеет возвращаемого значения.

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

Ответ. Императивный стиль программирования означает написание последовательности операторов, которые исполняющая среда выполняет шаг за шагом, как рецепт. Написанный код сообщает среде выполнения, как именно выполнять задачу. Например, сначала необходимо выполнить шаг 1, затем шаг 2. Это парадигма программирования, в которой используются переменные, которые означают, что состояние программы может измениться в любой момент, в том числе вне текущей функции. Императивное программирование вызывает побочные эффекты, изменяя значение 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. Однако есть наиболее важный момент: во избежание побочных эффектов языки функционального программирования по умолчанию делают все состояния неизменяемыми.

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

Ответ. Клавиша F5 сохраняет, компилирует, запускает и присоединяет отладчик; сочетание клавиш Ctrl или Cmd+F5 сохраняет, компилирует и запускает отладчик; сочетание клавиш Shift+F5 останавливает отладчик и работающее приложение, а сочетание клавиш Ctrl или Cmd+Shift+F5 перезапускает приложение с подключенным отладчиком.

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

   Ответ. В любые настроенные прослушиватели трассировки. По умолчанию это терминал или командная строка, но также можно указать 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. Чем различаются классы `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 помощью xUnit каким атрибутом вы должны дополнять методы тестирования?

   Ответ. Атрибутом `[Fact]` или `[Theory]`.

9. Какая команда `dotnet` выполняет тесты xUnit?

   Ответ. Команда `dotnet test`.

10. Какой оператор следует использовать для повторного создания перехваченного исключения `ex` без потери трассировки стека?

    Ответ. Используйте оператор `throw;`. Не используйте `throw ex;`, так как это приведет к потере информации о трассировке стека.

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

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

   Ответ. Шесть модификаторов доступа и их действие представлены ниже:

   - `private` — доступ ограничен содержащим классом;
   - `internal` — доступ ограничен текущей сборкой;
   - `protected` — доступ ограничен содержащим классом или типами, которые являются производными от содержащего класса;
   - `internal protected` — доступ ограничен содержащим классом, производным классом или текущей сборкой;

- `private protected` — доступ ограничен содержащим или производным классом, находящимися в текущей сборке;
- `public` — неограниченный доступ.

2. Чем различаются ключевые слова `static`, `const` и `readonly` при их применении к члену типа?

   Ответ. Разница описана ниже:

   - `static` делает член общим для всех экземпляров, и доступ к нему должен осуществляться через тип, а не через экземпляр типа;
   - `const` устанавливает поле как фиксированное литеральное значение, которое никогда не должно меняться, так как в процессе компиляции сборки, которые используют это поле, копируют литеральное значение во время компиляции;
   - `readonly` создает поле, которое может быть назначено только для использования конструктора или инициализатора поля во время выполнения.

3. Для чего используется конструктор?

   Ответ. Для выделения памяти и инициализации значений полей.

4. Зачем применять атрибут `[Flags]` к типу `enum`, когда нужно 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 вызовом `ToString` в качестве сохраненного целочисленного значения вместо одного или нескольких списков текстовых значений, разделенных запятыми.

5. В чем польза ключевого слова `partial`?

   Ответ. Оно позволяет разделить определение типа на несколько файлов.

6. Что вы знаете о кортежах?

   Ответ. Кортеж — 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 `record`?

   Ответ. Оно определяет структуру данных, которая по умолчанию является неизменной, что обеспечивает более функциональный стиль программирования. Как и класс, `record` может иметь свойства и методы, но значения свойств можно установить только во время инициализации.

8. Что такое перегрузка?

   Ответ. Перегрузка — это когда вы определяете несколько методов с одинаковым именем 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. Нужно присвоить 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 `-=`, позволяющие безопасно объединить несколько делегатов без замены каких-либо существующих обработчиков событий.

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

   Ответ. Производный класс (подкласс) — это класс, унаследованный от базового класса (суперкласса). Чтобы получить доступ к классу, от которого наследуется подкласс, внутри производного класса необходимо использовать ключевое слово `base`.

4. В чем разница между операциями `is` и `as`?

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

5. С помощью какого ключевого слова можно предотвратить наследование класса и переопределение метода?

   Ответ. Ключевое слово `sealed`.

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

   Ответ. Ключевое слово `abstract`.

7. Какое ключевое слово служит для переопределения члена?

   Ответ. Ключевое слово `virtual`.

8. Чем деструктор отличается от метода деконструктора?

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

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

   Ответ. Описания этих сигнатур приведены ниже:

   - конструктор без параметров;
   - конструктор с параметром `string`, обычно называемым `message`;
   - конструктор с параметром `string`, обычно называемым `message`, и параметром `Exception`, 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 другого типа. Вы определяете, какой тип хотите расширить, добавляя перед первым параметром этого типа в методе ключевое слово `this`.

Глава 7. Упаковка и распространение 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`?

   Ответ. Нужно добавить элемент `<ProjectReference>`, который задает для атрибута `Include` путь к файлу эталонного проекта внутри элемента `<ItemGroup>`:

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

3. В чем преимущество такого инструмента, как ILSpy?

   Ответ. Он позволяет научиться писать код на C# для платформы .NET, наблюдая за тем, как пишутся другие пакеты. Конечно, не используйте их интеллектуальную собственность. Но особенно полезно ознакомиться с тем, как разработчики Microsoft реализовали ключевые компоненты библиотек базовых классов. Декомпиляция также может быть полезна при вызове сторонней библиотеки, но, чтобы правильно вызывать ее, сначала лучше разобраться, как это делать.

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

   Ответ. Тип `System.Single`.

5. Какой инструмент следует использовать перед переносом приложения с .NET Framework на .NET 6 и позволяет выполнить бóльшую часть работы по переносу?

   Ответ. Перед переносом приложения с .NET Framework на .NET 6 следует использовать .NET Portability Analyzer. Вы можете использовать .NET Upgrade Assistant для выполнения большей части работы по переносу.

6. В чем разница между платформенно-зависимым и автономным развертыванием современных приложений .NET?

   Ответ. Современные приложения .NET, зависящие от платформы, требуют установки .NET, чтобы операционная система могла их выполнить. Автономные приложения .NET включают в себя все необходимое для самостоятельного выполнения.

7. Что такое RID?

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

8. Чем различаются команды `dotnet pack` и `dotnet publish`?

   Ответ. С помощью команды `dotnet pack` создается пакет NuGet, который затем может быть загружен в канал NuGet, такой как Microsoft. Команда `dotnet publish` позволяет поместить приложение и его зависимости в папку для развертывания в хост-системе.

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

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

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

    Ответ. Да, при условии, что они вызывают только API в .NET Standard 2.0.

Глава 8. Работа с распространенными типами .NET

1. Какое максимальное количество символов можно сохранить в переменной `string`?

   Ответ. Максимальный размер переменной `string` составляет 2 Гбайт или примерно 1 миллиард символов, поскольку каждый символ занимает 2 байта из-за внутреннего использования кодировки Unicode (UTF-16) для символов в строке.

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

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

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

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

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

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

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

   Ответ. Класс `SortedList<T>` использует меньше памяти, чем `SortedDictionary<T>`, а тот, в свою очередь, быстрее выполняет операции добавления и удаления несортированных данных. Если список заполняется уже отсортированными данными, то `SortedList<T>` работает быстрее, чем `SortedDictionary<T>`.

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

   Ответ. `cy-GB`.

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

   Ответ

   - Локализация влияет на пользовательский интерфейс приложения. Определяется нейтральными (только языковыми) или специфичными (языковыми и региональными) настройками. Вы предоставляете текст и другие значения на нескольких языках. К примеру, метка текстового поля с именем может быть отображена как *First name* на английском языке и *Prénom* на французском.
   - Глобализация влияет на данные вашего приложения. 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 конкретными, чтобы форматирование десятичных значений денежных единиц было правильным, например канадские доллары вместо французских евро.
   - Интернационализация — это сочетание локализации и глобализации.

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

   Ответ. Конец ввода.

9. Как в регулярных выражениях представить цифровые символы?

   Ответ. Нужно использовать `\d` или `[0-9]`.

10. Почему *нельзя* использовать официальный стандарт для адресов электронной почты при создании регулярного выражения, призванного проверять адрес электронной почты пользователя?

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

Глава 9. Работа с файлами, потоками и сериализация

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

   Ответ. Класс `File` содержит методы `static`, 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. Чем различаются методы 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 используются 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`, чтобы иметь совместную функциональность.
   - 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, 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 помощью одного байта. Другим символам может потребоваться два и более байта.

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 коллекцию экземпляров `Order`.

7. Какой формат сериализации лучше всего минимизирует затраты памяти?

   Ответ. В объектной нотации JavaScript (JSON) хорошо сочетаются требования к пространству и практические факторы, такие как удобочитаемость. Однако буферы протокола лучше всего подходят для минимизации требований к пространству.

8. Какой формат сериализации позволяет обеспечить кросс-платформенную совместимость?

   Ответ. Расширяемый язык разметки (XML) все еще пригоден, если вам нужна максимальная совместимость, особенно с устаревшими системами, хотя JSON более эффективен, особенно если вам необходимо интегрироваться с веб-системами или буферами протокола, чтобы обеспечить лучшую производительность и использовать минимальную полосу пропускания.

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

   Ответ. Использовать подобное значение для представления пути неправильно, так как предполагается, что символ «обратный слеш» служит в качестве разделителя папок во всех операционных системах. Вместо этого следует использовать метод `Path.Combine` и передавать отдельные значения `string` или массив `string` для каждой папки, как показано ниже:

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

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

    Ответ. На сайте https://www.nuget.org/.

Глава 10. Работа с данными с помощью Entity Framework Core

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

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

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

   Ответ. Предполагается, что свойство `ID`, или `Id`, или `ClassNameID`, 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. Когда соглашения не могут обеспечить 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. Такие технологии, как ASP.NET Core MVC и Blazor, могут считывать их, чтобы предоставлять пользователям предупреждения об автоматической проверке.

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

   Ответ. Вы можете выбрать Fluent API, а не атрибуты аннотации, если требуется, чтобы классы элементов были свободными от постороннего кода, который не нужен во всех сценариях. Например, при создании библиотеки классов .NET Standard 2.0 для классов сущностей вам может потребоваться использовать только атрибуты проверки, чтобы эти метаданные могли считываться Entity Framework Core и такими технологиями, как проверка привязки модели ASP.NET Core и настольные и мобильные приложения .NET MAUI. Однако вы можете использовать Fluent API для определения специальных возможностей Entity Framework Core, таких как сопоставление с именем другой таблицы или столбца.

6. Что означает уровень изоляции 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 количества затронутых объектов.

8. В чем разница между жадной и явной загрузками?

   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 БД и должны быть явно загружены непосредственно перед тем, как понадобятся.

9. Как определить сущностный класс EF Core, чтобы он соответствовал 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. Какие преимущества дает объявление свойств навигации сущностей как `virtual`?

    Ответ. Это позволяет включить ленивую загрузку.

Глава 11. 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 данных того типа данных, с которыми хотите работать, за исключением провайдеров LINQ to Objects и LINQ to XML, которые встроены в .NET.

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

   Ответ. Метод `Select` позволяет осуществлять проекцию (выбор) свойств.

3. С помощью какого метода расширения LINQ вы бы выполнили фильтрацию последовательности?

   Ответ. Метод `Where` позволяет выполнить фильтрацию, предоставляя делегат (или лямбда-выражение), который возвращает логическое значение, показывающее, нужно ли включать значение в результаты.

4. Каковы 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. 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` и объединит их в одну последовательность.

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

   Ответ. Интерфейс `IEnumerable<T>` указывает поставщик LINQ, который будет выполнять запрос локально, например LINQ to Objects. Эти поставщики не имеют ограничений, но могут быть менее эффективными.

   Интерфейс `IQueryable<T>` указывает поставщик 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 запросы LINQ в операторы SQL. Эти поставщики иногда имеют ограничения, например отсутствие поддержки определенных выражений, и могут создавать исключения. Вы можете превратить из поставщика `IQueryable<T>` в поставщик `IEnumerable<T>`, вызвав метод `AsEnumerable`.

7. Что представляет последний параметр типа `T` в делегатах-дженериках `Func`, таких как `Func<T1, T2, T>`?

   Ответ. 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. 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. Это просто синтаксический сахар. Он облегчает чтение кода разработчиками, но не добавляет никакой дополнительной функциональности, кроме ключевого слова `let`.

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

    Ответ. Создайте класс `static`, содержащий метод `static` с параметром `IEnumerable<T>` и префиксом `this`, как показано ниже:

```
namespace System.Linq
{
  public static class MyLinqExtensionMethods
```

```
  {
    public static IEnumerable<T> MyChainableExtensionMethod<T>(
      this IEnumerable<T> sequence)
    {
      // возвращает значение IEnumerable<T>
    }

    public static int? MyAggregateExtensionMethod<T>(
      this IEnumerable<T> sequence)
    {
      // возвращает некое целочисленное значение
    }
  }
}
```

Глава 12. 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`.

2. Насколько точен класс `Stopwatch`?

   Ответ. С точностью до наносекунды (миллиардной доли секунды), но все равно может быть погрешность.

3. По соглашению, какой суффикс следует применять к методу, если он возвращает `Task` или `Task<T>`?

   Ответ. Суффикс `Async`. Например, при использовании синхронного метода `Open` используйте `OpenAsync` для эквивалента, возвращающего `Task` или `Task<T>`.

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

   Ответ. Ключевое слово `async`.

5. Как создать дочернюю задачу?

   Ответ. Необходимо вызвать метод `Task.Factory.StartNew` с параметром `TaskCreationOptions.AttachToParent`.

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

   Ответ. Оно не позволяет указать время ожидания, что может привести к взаимоблокировке. Используйте метод `Monitor.Enter` и передайте аргумент `TimeSpan` в качестве тайм-аута, а затем явно вызовите метод `Monitor.Exit` для снятия блокировки в конце работы.

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

   Ответ. Когда 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, выходя за границы процесса. Класс `Monitor` работает только с ресурсами внутри текущего процесса.

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

   Ответ. Это улучшает масштабируемость, но не производительность конкретного запроса, поскольку требуются дополнительные усилия по обработке взаимодействия между потоками.

10. Вы можете отменить задачу? Если да, то каким образом?

    Ответ. Да, вы можете отменить задачу. Информация о том, как это сделать, представлена на странице https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/async/cancel-an-async-task-or-a-list-of-tasks.

Глава 13. Реальные приложения на C# и .NET

1. .NET 6 является кросс-платформенной. Приложения Windows Forms и WPF могут работать на .NET 6. Могут ли эти приложения работать на macOS и Linux?

   Ответ. Нет. Хотя приложения Windows Forms и WPF могут работать на .NET 6, они также должны вызывать Win32 API, поэтому ограничены возможностями Windows. Когда вы загружаете .NET SDK для Windows, он включает дополнительную рабочую нагрузку, чтобы обеспечить поддержку приложений WPF и Windows Forms.

2. Как приложение Windows Forms определяет свой пользовательский интерфейс и почему это может быть проблемой?

   Ответ. Приложение Windows Forms определяет свой пользовательский интерфейс с помощью кода C#. Поэтому важно использовать визуальный конструктор Windows Forms с панелью инструментов и поддержкой перетаскивания. Непосредственно редактировать сгенерированный код напрямую довольно сложно.

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

   Ответ. Приложение WPF или UWP может определять свой пользовательский интерфейс с помощью XAML, который разработчикам понять и написать проще, чем код C#, даже без визуального конструктора. XAML также используется в .NET MAUI.

Глава 14. Разработка сайтов с помощью ASP.NET Core Razor Pages

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

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

2. Какие шесть кодов состояния и их описания могут быть возвращены в HTTP-ответе?

   Ответ. `200 OK`, `201 Created`, `301 Moved Permanently`, `400 Bad Request`, `404 Not Found` (отсутствует ресурс) и `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`.

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

   Ответ. Чтобы отделить добавление и настройку сервисов зависимости от настройки промежуточного программного обеспечения 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. Шаблоны проектов по умолчанию для ASP.NET Core 6 не используют класс `Startup` и вместо этого помещают весь код конфигурации в файл `Program.cs`.

4. Что такое HSTS и для чего он предназначен?

   Ответ. HTTP Strict Transport Security (HSTS) — дополнительный механизм улучшения безопасности. Если сайт указывает его и браузер его поддерживает, то он принудительно передает все данные по HTTPS и не позволяет посетителю использовать ненадежные или недействительные сертификаты.

5. Как включить статические HTML-страницы для сайта?

   Ответ. Необходимо добавить операторы в метод `Configure` класса `Startup`, чтобы использовать файлы по умолчанию, а затем использовать статические файлы (этот порядок важен!):

   ```
   app.UseDefaultFiles(); // index.html, default.html и т. п.
   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 определить общие макеты для Razor Pages?

   Ответ. 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 отделить разметку от выделенного кода на странице Razor?

   Ответ. Создайте два файла: `MyPage.cshtml`, который содержит разметку, и `MyPage.cshtml.cs`, который содержит класс, наследуемый от `PageModel`. В файле `MyPage.cshtml` установите модель для использования класса, как показано ниже:

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

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

   Ответ. Необходимо выполнить следующие действия:

   - в файле проекта указать ссылку на сборку, определяющую класс контекста данных;
   - в файле Program.cs или в классе `Startup` импортировать пространства имен для `Microsoft.EntityFrameworkCore` и класс контекста данных;
   - в методе `ConfigureServices` или в разделе `Program.cs`, в котором настраиваются сервисы, добавить оператор, который настраивает контекст данных со строкой подключения к базе данных для использования с указанным поставщиком базы данных, таким как SQLite или SQL Server:

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

   - в классе модели Razor Page или в разделе `@functions` объявить скрытое поле для хранения контекста данных, а затем установить его в конструкторе:

     ```
     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. Все, что связано со страницей Razor, можно скомпилировать в библиотеку классов, используя следующую команду:

    ```
    dotnet new razorclasslib -s
    ```

Глава 15. Разработка сайтов с помощью 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 выполнении 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 начале всех представлений.

2. Как называются три сегмента, определенные по умолчанию в ASP. NET Core MVC, что они представляют и какие из них необязательны?

   Ответ

   - `{controller}` — например, `/shippers`, представляет класс контроллера для создания экземпляра, например `HomeController`. Необязателен, так как может использовать значение по умолчанию `Home`.
   - `{action}` — например, `/privacy`, представляет метод действия для выполнения, например, `Privacy`. Необязателен, так как может использовать значение по умолчанию `Index`.
   - `{id}` — например, `/5` представляет параметр в методе действия, например `int id`. Необязателен, так как к нему добавляется символ `?`.

3. Что делает связыватель модели по умолчанию и какие типы данных он 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, допускающие значение `null`;
   - сложные типы, такие как `Person`, `Product`;
   - типы коллекции, такие как `ICollection<T>` или `IList<T>`.

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

   Ответ. Вызовите метод `RenderBody`:

   ```
   @RenderBody()
   ```

5. Как в общем файле макета, таком как `_Layout.cshtml`, вывести раздел, для которого содержимое может быть предоставлено текущим представлением, и как это представление выдает содержимое для данного раздела?

   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, как показано ниже:

   ```
   @section Scripts
   {
   ```

```
<script>
  alert('Hello, Mr. Page!');
</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 и метода действия и специальной папки Shared:

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

   Эти три пути можно обобщить следующим образом:

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

7. Как выдать браузеру пользователя инструкцию кэшировать ответ в течение 24 ч?

   Ответ. Дополните класс контроллера или метод действия атрибутом `[ResponseCache]` и установите для параметра `Duration` значение `86400` с, а для параметра `Location` — `ResponseCacheLocation.Client`.

8. Почему вы можете включить 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 проверяет, не дополнен ли класс (или класс, из которого он происходит) атрибутом `[Controller]`.

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

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

Глава 16. Разработка и использование веб-сервисов

1. От какого класса вы должны наследовать, чтобы создать класс контроллера для сервиса ASP.NET Core Web API?

   Ответ. Вы должны наследовать от класса `ControllerBase`. Не наследуйте от `Controller`, как в MVC, поскольку этот класс включает такие методы, как `View`, использующие файлы Razor для визуализации HTML, которые не нужны для веб-сервиса.

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

   Ответ. Вам следует также 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. 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. Дополните метод действия атрибутом `[ProducesResponseType]`, как показано 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. Какие методы можно вызывать для возврата ответов с разными кодами состояния?

   Ответ. Это методы:

   - `Ok` — возвращает код состояния `200` и объект, переданный этому методу в теле кода;
   - `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. Каковы четыре способа тестирования веб-сервиса?

   Ответ. Описание способов представлено ниже:

   - использование браузера для проверки простых HTTP-запросов `GET`;
   - установка расширения REST Client для Visual Studio Code;
   - установка пакета Swagger NuGet в свой проект веб-сервиса, включение Swagger и применение пользовательского интерфейса тестирования Swagger;
   - установка инструмента Postman по следующей ссылке: https://www.postman.com.

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

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

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

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

9. Как вы можете разрешить клиентам определять работоспособность вашего веб-сервиса в ASP.NET Core 2.2 и более поздних версиях?

   Ответ. Вы можете установить API проверки работоспособности, включая проверки работоспособности базы данных для контекстов данных Entity Framework Core. Чтобы сообщить клиенту подробную информацию, проверку работоспособности можно расширить.

10. Какие преимущества обеспечивает маршрутизация конечных точек?

    Ответ. Маршрутизация конечных точек обеспечивает улучшенную производительность при маршрутизации и выборе метода действия, а также сервис генерации ссылок.

Глава 17. Создание пользовательских интерфейсов с помощью Blazor

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

   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 может упростить реализацию функциональности. Пользовательский интерфейс обновляется с помощью SignalR. Это значит, между браузером и сервером необходимо постоянное соединение, что, в свою очередь, ограничивает масштабируемость.
   - Blazor WebAssembly выполняет код на стороне клиента. Это значит, код имеет доступ только к ресурсам в браузере, что может усложнить реализацию, поскольку всякий раз, когда требуются новые данные, должен выполняться обратный вызов на сервер.

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 помощью C# и .NET вместо JavaScript. Необходим ли Blazor какой-либо JavaScript?

   Ответ. Да, компонентам Blazor требуется минимальный 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 рабочий файл сервиса JavaScript `service-worker.js`. JavaScript также необходим для вызова браузера и других API на стороне клиента.

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. Он интегрируется с системой маршрутизации Blazor, поэтому может автоматически применять текущий стиль для визуального указания, когда текущий маршрут соответствует компоненту `<NavLink>`.

6. Каким образом значение передается 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. Он позволяет получать автоматические сообщения 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. Каковы два ключевых различия в классе `Program` между сервером Blazor и проектом Blazor WebAssembly?

    Ответ. Различия выглядят следующим образом: 1) использование `WebAssemblyHostBuilder` вместо `Host.CreateDefaultBuilder` и 2) регистрация `HttpClient` с базовым адресом среды хоста.

Глава 18. Создание и использование специализированных сервисов

1. У вас есть приложение, которое взаимодействует с сервисом, созданным с помощью устаревшего сервиса Windows Communication Foundation. Каковы два возможных варианта переноса сервиса и клиента на современную .NET?

   Ответ. Первый вариант: использование проекта Core WCF с открытым исходным кодом. Второй вариант: повторная реализация сервиса и клиента с помощью gRPC.

2. Какой транспортный протокол использует сервис OData?

   Ответ. Транспортный протокол HyperText Transport Protocol (HTTP).

3. Почему веб-сервис OData является более гибким, чем традиционный веб-сервис ASP.NET Core Web API?

   Ответ. OData использует строки запросов для своих запросов, что позволяет клиенту контролировать то, что возвращается, и сводит к минимуму циклы полного обхода. Традиционный Web API определяет все методы и то, что возвращается.

4. Что нужно сделать с методом действия в контроллере OData, чтобы включить строки запроса для настройки того, что он возвращает?

   Ответ. Нужно дополнить метод действия в контроллере OData атрибутом `[EnableQuery]`.

5. Какой транспортный протокол использует сервис GraphQL?

   Ответ. HTTP или другие протоколы, например WebSocket.

6. Как определяются контракты в gRPC?

   Ответ. С помощью файлов `.proto`.

7. Благодаря каким трем преимуществам gRPC хорошо подходит для реализации сервисов?

   Ответ. Эти преимущества описаны ниже:

   1) его двоичная сериализация Protobuf, которая минимизирует использование сети;
   2) его требование HTTP/2, которое обеспечивает значительные преимущества в производительности;
   3) его поддержка почти всеми языками и платформами.

8. Какие средства передачи данных использует SignalR и какой из них применяется по умолчанию?

   Ответ. SignalR предпочитает использовать WebSockets в качестве своего средства передачи данных, затем он будет возвращаться к событиям на стороне сервера, и, наконец, будет использовать длинный опрос, если ни один из них не поддерживается клиентом и сервером.

9. В чем разница между моделями внутрипроцессного и изолированного хостингов для Azure Functions?

   Ответ. Модель внутрипроцессного хостинга требует, чтобы ваша функция Azure загружалась вместе с другим кодом и была нацелена на заранее определенную версию LTS-выпуска, например .NET Core 3.1 или .NET 6. Модель изолированного хостинга позволяет вашей функции Azure загружаться в собственный процесс и использовать любую версию .NET, которую вы выберете.

10. Что рекомендуется делать при разработке сигнатур методов RPC?

    Ответ. Важно определять один параметр с помощью сложного типа. Это позволяет в будущем добавлять к типу дополнительные свойства, не нарушая контракт между клиентом и сервисов.

Глава 19. Разработка мобильных и настольных приложений с помощью .NET MAUI

1. Каковы четыре категории компонентов пользовательского интерфейса .NET MAUI и что они собой представляют?

   Ответ. Эти категории перечислены ниже:

   - *страницы* представляют экраны мобильных приложений;
   - *макеты* представляют структуру комбинации других компонентов пользовательского интерфейса;
   - *представления* определяют отдельный компонент пользовательского интерфейса;
   - *ячейки* представляют отдельный элемент в виде списка или таблицы.

2. Каковы четыре типа ячеек?

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

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

   Ответ. Вы можете установить некоторые контекстные действия, представляющие собой пункты меню, которые вызывают событие, как показано ниже:

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

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

   Ответ. Элемент `Entry` используется для одной строки текста, а `Editor` — для нескольких строк.

5. 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.

6. Когда выполняется вызов методов `PushAsync` и `PopAsync` в приложении .NET MAUI?

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

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

Чтобы перейти к следующему экрану, добавьте следующую страницу в объекте `Navigation`:

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

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

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

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

   Ответ. Разница заключается в том, что `Margin` находится вне `Border`, а `Padding` — внутри `Border`.

8. Как обработчики событий прикрепляются к объекту с помощью XAML?

   Ответ. С помощью присвоения атрибуту имени события имена метода в классе кода программной части:

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

9. Что делают стили XAML?

   Ответ. Позволяют установить одно или несколько свойств.

10. Где можно определить ресурсы?

    Ответ. Вы можете определить ресурсы в любом элементе в зависимости от того, где хотите использовать их:

    - для совместного использования ресурсов во всем приложении определите ресурсы в элементе `<Application. Resources>`;
    - для совместного использования ресурсов только в пределах страницы определите ресурсы в ее элементе `<Page.Resources>`;
    - для совместного использования ресурсов только в одном элементе, таком как кнопка, определите ресурсы в его элементе `<Button.Resources>`.

Глава 20. Защита данных и приложений

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

   Ответ. Алгоритм AES.

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

   Ответ. 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.

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

   Ответ. Малого.

5. Что такое криптографическое хеширование данных?

   Ответ. Криптографический хеш — это вывод фиксированного размера, который получается в результате ввода произвольного размера, обрабатываемого хеш-функцией. Хеш-функции — односторонние, то есть единственный способ воссоздать исходные данные — это перебрать все возможные входные данные и сравнить результаты.

6. Что такое криптографическая подпись?

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

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. Чтобы замедлить радужные словарные атаки.

10. SHA-1 — это алгоритм хеширования, разработанный Национальным агентством безопасности США. Почему его не следует использовать?

    Ответ. Алгоритм SHA-1 больше не является безопасным. Все современные браузеры перестали принимать SSL-сертификаты алгоритма SHA-1.