1 Знакомство с приложениями и сервисами на .NET

В первой главе мы настроим среду разработки, установив программы Visual Studio 2022 и Visual Studio Code, а также разберемся с необходимыми инструментами для разработки приложений и сервисов. В конце главы вы узнаете, куда обращаться за помощью.

В репозитории GitHub, который я создал для этой книги, размещены полные решения для всех описываемых задач. Он доступен по ссылке https://github.com/markjprice/apps-services-net8/.

Решать задачи из книги можно как в браузерной версии Visual Studio Code, так и в обычном редакторе кода, а также использовать эти инструменты одновременно. Удобно сравнивать свой код с моим решением и при необходимости копировать/вставлять нужные части.

Упоминая в этой книге слово *«современная»* в контексте платформы .NET, я имею в виду версию 8 и ее предшественниц, таких как .NET 6, произошедших из .NET Core. Слово *«устаревшая»* относится к платформам .NET Framework, Mono, Xamarin и .NET Standard. Современная версия .NET консолидирует перечисленные унаследованные платформы и стандарты.

Знакомство с этой книгой и ее содержимым

Эта книга рассчитана на следующие аудитории:

- читателей моей книги *C# 12 and .NET 8 – Modern Cross-Platform Development Fundamentals*, желающих продолжить обучение;
- разработчиков с базовыми навыками и знаниями языка C# и платформы .NET, желающих получить опыт разработки реальных приложений и сервисов.

Сопутствующие книги для продолжения обучения

Эта книга — вторая из трилогии, посвященной платформе .NET 8.

1. В первой книге даются основы языка C# и библиотеки .NET и ASP.NET Core для разработки веб-приложений. Она предназначена для последовательного чтения, поскольку навыки и знания, полученные в предыдущих главах, дополняются и необходимы для понимания материала последующих глав.
2. Во второй книге (вы держите ее в руках) рассматриваются более узкие темы, такие как интернационализация, а также популярные пакеты сторонних разработчиков, в том числе Serilog и NodaTime. Вы научитесь разрабатывать нативные сервисы с AOT-компиляцией на основе минимальных API ASP.NET Core и узнаете, как повысить производительность, масштабируемость и надежность посредством кэширования данных, очередей и фоновых сервисов. С технологиями GraphQL, gRPC, SignalR и Azure Functions вы реализуете дополнительные сервисы. Наконец, вы узнаете, как с помощью Blazor и .NET MAUI создавать графические пользовательские интерфейсы для сайтов, классических и мобильных приложений.
3. В третьей книге[1] описываются важные инструменты и навыки, которыми должен обладать каждый профессиональный разработчик .NET. Сюда входят паттерны проектирования и архитектуры решений, отладка, анализ памяти, все важные виды тестирования — модульное, производительности, системное, веб- и мобильное, а также инструменты хостинга и развертывания, такие как Docker и Azure Pipelines. Наконец, мы рассмотрим, как подготовиться к собеседованию на замещение вакансии разработчика .NET.

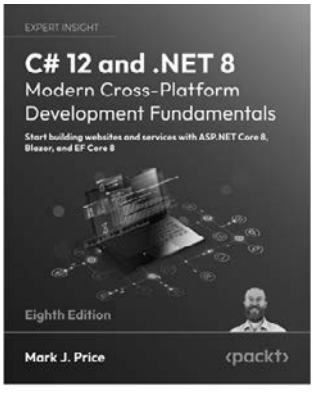
Краткое описание трилогии .NET 8 и ее наиболее важных тем показано на рис. 1.1.

Что вы узнаете из этой книги

За исключением первой главы, книгу можно разделить на четыре характерные части.

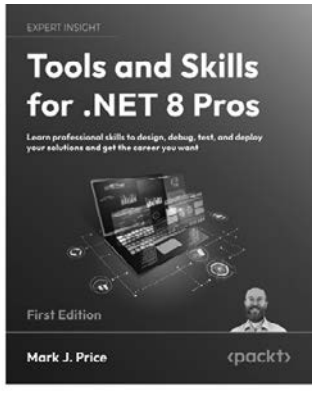
1. **Управление данными:** хранение и управление локальными и облачными данными с помощью SQL Server и Azure Cosmos DB. В последующих главах используются БД SQL Server и модели сущностей, созданием которых вы займетесь в конце главы 3. В главе, посвященной Cosmos DB, используется библиотека SQL API, а глава о Gremlin, интерфейсе, обеспечивающем способ конфигурации, выполнения и взаимодействия с обходами графа, доступна в качестве дополнения в Интернете.

[1] *Прайс М.* .NET 8: лучшие практики и паттерны проектирования. — СПб.: Питер, 2025.



- **Язык C#**, в том числе новые возможности версии C# 12, объектно-ориентированное программирование (ООП), отладка и модульное тестирование.
- **Библиотеки .NET**, в том числе касающиеся работы с числами, текстом, регулярными выражениями, коллекциями, файловым вводом-выводом и данными с помощью EF Core и SQLite.
- **Разработка сайтов и веб-сервисов** с помощью ASP.NET Core и Blazor

- **Дополнительные библиотеки:** интернационализация, многозадачность и пакеты сторонних разработчиков.
- **Дополнительная информация об обработке данных:** SQL Server и Cosmos DB.
- **Дополнительная информация о сервисах:** кэширование, очереди, минимальные API, GraphQL, gRPC, SignalR и Azure Functions.
- **Дополнительная информация о пользовательских интерфейсах**: ASP.NET Core и .NET MAUI



- **Инструменты:** интегрированные среды разработки, отладка, анализ памяти и ИИ-ассистенты.
- **Тестирование:** модульное, интеграционное, производительности, системное и веб-, в том числе тесты внедрения зависимостей и инверсий контроля.
- **Проектирование:** паттерны и архитектуры.
- **Развертывание:** непрерывная интеграция/непрерывное развертывание, хостинг в Azure, конвейеры развертывания и автоматизация.
- **Карьера:** подготовка к собеседованию

Рис. 1.1. Трилогия для изучения C# 12 и .NET 8

2. **Специализированные библиотеки:** обработка дат и времени, интернационализация, повышение производительности с помощью потоков и путем делегирования задач, сторонние библиотеки для работы с изображениями, правила проверки данных и т. д. Эти главы сродни книге кулинарных рецептов. Если вас не интересует какая-то тема, вы можете ее пропустить, читать эти главы можно в любом порядке.
3. **Сервисные технологии:** создание и защита сервисов, построенных на минимальных программных интерфейсах ASP.NET Core, GraphQL, gRPC, SignalR и Azure Functions. Вы узнаете об очередях, кэшировании и планировании событий в рамках повышения масштабируемости и надежности сервисов.

   Технологии пользовательских интерфейсов: создание пользовательских интерфейсов средствами ASP.NET Core, Blazor и .NET MAUI.

Уникальная философия обучения

Большинство людей лучше всего усваивают сложные темы путем подражания и повторения, а не читая подробное объяснение теории. Поэтому я не буду перегружать вас доскональными объяснениями каждого шага. Суть в том, чтобы мотивировать вас писать код и анализировать результаты его выполнения.

С ходу вам не понадобятся все тонкости и нюансы. Это дело времени, когда вы будете разрабатывать собственные приложения и выходить за рамки того, о чем прочли в книгах.

Исправление моих ошибок

Говоря словами Сэмюэла Джонсона, автора толкового словаря английского языка 1755 года, я совершил *«ряд дичайших промахов и забавных нелепостей, от которых не застрахована ни одна работа подобного масштаба»*. Я беру на себя всю ответственность за них и надеюсь, что вы оцените мою попытку идти против ветра, написав эту книгу о стремительно развивающихся технологиях, таких как C# и .NET, и приложениях и сервисах, создаваемых с их помощью.

Если у вас возникли какие-либо затруднения с этой книгой, пожалуйста, свяжитесь со мной, прежде чем писать отрицательный отзыв на сайте книжного магазина. Авторы не могут отвечать на такие отзывы, поэтому я не могу связаться с вами с целью решить проблему. Я хочу, чтобы вы извлекли максимальную пользу от моей книги, поэтому прислушиваюсь к вашим отзывам и учитываю их в следующем издании. Пожалуйста, напишите мне (адрес моей электронной почты опубликован в репозитории GitHub к этой книге), пообщайтесь со мной в специальном канале Discord (https://discord.com/invite/ETZTPmrQFX) или поднимите вопрос на форуме по адресу https://github.com/markjprice/apps-services-net8/issues.

Готовые решения на сайте GitHub

Универсальный код решений в репозитории GitHub для этой книги доступен по адресу https://github.com/markjprice/apps-services-net8/tree/main/code.

Соглашения об именовании проектов и нумерации портов

Выполняя задания по программированию, описанные в книге, вы создадите десятки проектов. Многие из них будут представлять собой сайты и сервисы, требующие номера портов для размещения на домене `localhost`.

В крупных и сложных решениях бывает трудно ориентироваться в кодовой базе. Поэтому для структурирования проектов весьма желательно упрощать поиск

компонентов. Рекомендуется использовать всеобъемлющие названия решений, передающие суть приложения/проекта.

В 1990-х годах компания Microsoft зарегистрировала торговый знак *Northwind* в качестве названия вымышленной компании для использования в БД и примерах кода. Сначала оно применялось в примерах для программы Access, а затем стало использоваться и в SQL Server. Мы создадим несколько проектов для этой вымышленной компании, поэтому будем использовать префикс Northwind в названиях всех проектов.

Дельный совет: есть множество способов структурирования и присвоения названий проектам и решениям, например выстраивание иерархии папок, а также соглашения об именовании. Если вы работаете в команде, согласуйте эти правила.

Чтобы любой разработчик мог с ходу определить предназначение проектов в решении, полезно учитывать соглашение об именовании проектов. Обычно упоминается тип проекта, например библиотека классов, консольное приложение, сайт и т. д. (табл. 1.1).

Таблица 1.1. Примеры названий для распространенных типов проектов

Название	Описание
`Northwind.Common`	Проект библиотеки классов для типов общего назначения, таких как интерфейсы, перечисления, классы, записи и структуры, используемых в ряде проектов
`Northwind.Common.EntityModels`	Проект библиотеки классов для моделей сущностей EF Core общего назначения. Модели сущностей часто используются как на стороне сервера, так и на стороне клиента, поэтому лучше всего отделить зависимости от конкретных провайдеров баз данных
`Northwind.Common.DataContext`	Проект библиотеки классов в контексте базы данных EF Core с зависимостями от конкретных провайдеров баз данных
`Northwind.Mvc`	Проект ASP.NET Core для сложного сайта на основе паттерна MVC. Проще проводить модульное тестирование
`Northwind.WebApi.Service`	Проект ASP.NET Core для сервиса HTTP API. Хороший выбор для интеграции с сайтами, поскольку для взаимодействия с сервисом может использоваться любая библиотека JavaScript или Blazor

Название	Описание
Northwind.WebApi.Client.Console	Клиент для веб-сервиса. Последняя часть названия указывает на то, что это консольное приложение
Northwind.gRPC.Service	Проект ASP.NET Core для gRPC-сервиса
Northwind.gRPC.Client.Mvc	Клиент для gRPC-сервиса. Последняя часть названия указывает на то, что это проект сайта ASP.NET Core MVC
Northwind.BlazorWasm.Client	Проект клиентской части ASP.NET Core Blazor WebAssembly
Northwind.BlazorWasm.Server	Проект серверной части ASP.NET Core Blazor WebAssembly
Northwind.BlazorWasm.Shared	Библиотека классов, совместно используемая клиентскими и серверными проектами Blazor

Чтобы запускать любой из этих проектов одновременно, важно не задавать одинаковые номера портов для разных проектов. Я использовал следующее соглашение:

```
https://localhost:5[номер_главы]1/
http://localhost:5[номер_главы]2/
```

Например, для зашифрованного подключения к сайту, созданному в главе 14, я использовал порт 5141:

```
https://localhost:5141/
```

Настройка информирования об ошибках

По умолчанию предупреждения компилятора возникают в случае потенциальных проблем с кодом при первой сборке проекта, но не препятствуют компиляции и скрыты при повторной сборке. Предупреждения возникают не на пустом месте, так что игнорировать их в любом случае не стоит.

Добросовестные разработчики своевременно реагируют на предупреждения, решая возникающие проблемы, для чего настраивают .NET-проекты следующим образом:

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

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <TreatWarningsAsErrors>true</TreatWarningsAsErrors>
  </PropertyGroup>
```

Я задействовал строгую обработку предупреждений (как ошибок) в (почти) всех решениях в репозитории GitHub.

Исключение касается gRPC-проектов ввиду сочетания ряда факторов. В .NET 7 и более поздних версиях компилятор выбрасывает предупреждение в процессе компиляции исходных файлов с типами, набранными строчными буквами. Такая ситуация возникает, если, к примеру, вы определите класс `person`:

```
public class person
{
}
```

Данное предупреждение компилятора было введено для предотвращения конфликтов добавляемых имен в будущих версиях C#.

Увы, инструменты Google для генерации исходных файлов C# из `.proto`-файлов создают имена классов, содержащие только строчные буквы:

```
#region Designer generated code

using pb = global::Google.Protobuf;
```

При строгой обработке предупреждений (они обрабатываются как ошибки) компилятор откажется компилировать исходный код:

```
Error CS8981 The type name 'pb' only contains lower-cased ascii characters.
Such names may become reserved for the language. Northwind.Grpc.Service C:\
apps-services-net8\Chapter14\Northwind.Grpc.Service\obj\Debug\net8.0\Protos\
Greet.cs
```

Дельный совет: в проектах .NET (за исключением gRPC, пока Google не обновит инструменты генерации кода) всегда задействуйте строгую обработку предупреждений.

Если после активации этой функции будет возникать слишком много ошибок, вы можете отключить определенные предупреждения, используя элемент `<WarningsNotAsErrors>` со списком кодов предупреждений, разделенных запятыми:

```
<WarningsNotAsErrors>0219,CS8981</WarningsNotAsErrors>
```

В копилку знаний: подробнее о настройке информирования об ошибках можно узнать по ссылке https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/compiler-options/errors-warnings#warningsaserrors-and-warningsnotaserrors.

Технологии разработки приложений и сервисов

В компании Microsoft платформы для создания приложений и сервисов называются *моделями приложений* (app models) или *рабочими нагрузками* (workloads).

Технология .NET

.NET, .NET Core, .NET Framework и Xamarin — это родственные и частично дублирующиеся платформы для разработки приложений и сервисов. Если вы не знакомы с историей .NET, рекомендую изучить концепции развития .NET по ссылке https://github.com/markjprice/cs12dotnet8/blob/main/docs/ch01-dotnet-history.md.

Создание сайтов и приложений с помощью ASP.NET Core

Сайты состоят из множества веб-страниц, загружаемых статически из файловой системы или генерируемых динамически с помощью серверной технологии типа ASP.NET Core. Браузер отправляет `GET`-запросы с *URL*-адресами (*Unique Resource Locator* — универсальный локатор ресурса) — уникальными идентификаторами каждой страницы — и управляет данными на сервере с помощью запросов `POST`, `PUT` и `DELETE`.

На многих сайтах браузер рассматривается как уровень представления, а почти вся обработка выполняется на стороне сервера. На стороне клиента могут выполняться JavaScript-сценарии для реализации некоторых функций представления, таких как веб-карусели, и для проверки данных.

Платформа ASP.NET Core предоставляет множество технологий для создания сайтов.

- *ASP.NET Core Razor Pages* позволяет динамически генерировать HTML-код простых сайтов.
- *ASP.NET Core MVC* — это реализация паттерна проектирования *MVC* (*Model — View — Controller*, «Модель — представление — контроллер»), востребованного при разработке сложных сайтов. О том, как с его помощью создавать пользовательские интерфейсы (user interface, UI), вы узнаете из главы 14.
- *Библиотеки классов Razor* позволяют упаковывать многократно используемый функционал проектов ASP.NET Core, в том числе компоненты пользовательского интерфейса.

- *Платформа Blazor* позволяет создавать UI-компоненты с помощью C# и .NET, а затем запускать их в браузере или встроенном веб-компоненте вместо JavaScript-фреймворков пользовательского интерфейса, таких как Angular, React и Vue.

Платформа Blazor предназначена не только для создания сайтов. В сочетании с .NET MAUI она также может использоваться для создания гибридных мобильных и классических приложений.

Предполагается, что вы уже знакомы с основами разработки на ASP.NET Core, поэтому, хотя в книге и рассматриваются основы, мы быстро перейдем к более продвинутым темам.

Создание веб- и других сервисов

Официальных определений не существует, но иногда сервисы описываются в зависимости от их сложности.

- *Сервис* — вся функциональность, необходимая для работы клиентского приложения, реализована в одном монолитном сервисе.
- *Микросервис* — небольшой автономный компонент, который предоставляет определенный набор функциональных возможностей. Руководящий принцип при определении границ функциональности микросервиса заключается в том, что каждый из них должен взаимодействовать с собственными данными. Только он должен читать и записывать эти данные. Если в ваше хранилище данных обращается несколько сервисов, то их нельзя назвать микросервисами.
- *Наносервис* — отдельная функция, предоставляемая в качестве услуги. В отличие от непрерывно работающих сервисов и микросервисов наносервисы часто неактивны, пока к ним не обратятся. Такая схема работы применяется для экономии ресурсов и затрат. По этой причине их также называют бессерверными сервисами.

Хотя в последнее десятилетие благодаря теоретическим преимуществам использовать микро- и наносервисы и стало модно, монолитные сервисы вновь набирают популярность, поскольку разработчики обнаружили, что в реальности микросервисы далеко не всегда так привлекательны, как в теории.

Вы научитесь создавать веб- и минимальные API ASP.NET Core — веб-сервисы с протоколом HTTP для базовых коммуникаций и принципами проектирования REST-архитектур, разработанными Роем Филдингом. Вы также узнаете, как созда-

вать сервисы с использованием веб- и прочих технологий, расширяющих базовые веб-API, в том числе следующих.

- *gRPC* — разработка высокоэффективных и производительных микросервисов с поддержкой практически любых платформ.
- *SignalR* — реализация между компонентами коммуникаций в реальном времени.
- *GraphQL* — управление со стороны клиента извлечением данных из нескольких источников. Хотя GraphQL и поддерживает протокол HTTP, это опциональная функция, и данный язык запросов не следует принципам веб-дизайна, описанным Роем Филдингом в своей диссертации о REST API.
- *Azure Functions* — развертывание бессерверных наносервисов в облаке.
- *OData* — удобное превращение EF Core и других моделей данных в веб-сервис.

Windows Communication Foundation

В 2006 году компания Microsoft выпустила версию .NET Framework 3.0 с несколькими новыми крупными фреймворками, в числе которых *Windows Communication Foundation (WCF)*. В нем бизнес-логика сервиса абстрагировалась от инфраструктуры коммуникационных технологий, чтобы в будущем можно было легко перейти на альтернативный вариант или даже использовать несколько механизмов для взаимодействия с сервисом.

Платформа WCF в значительной степени использует XML-конфигурацию для декларативного определения конечных точек, в том числе их адрес (address), привязку (binding) и контракт (contract). Это так называемая концепция ABC, трех ключевых элементов, конечных точек WCF. Освоив ее, вы оцените мощь и одновременную гибкость технологии WCF.

Компания Microsoft не стала официально портировать WCF на современную платформу .NET, тем не менее существует открытый проект *CoreWCF*, созданный сообществом разработчиков и управляемый организацией .NET Foundation. Если вам необходимо перенести сервис из .NET Framework на современную версию .NET или создать клиент для WCF-сервиса, воспользуйтесь проектом CoreWCF. Имейте в виду, что это решение никогда не будет считаться полноценным портом, поскольку некоторые компоненты технологии WCF специфичны для Windows.

Такие платформы, как WCF, позволяют создавать распределенные приложения. Клиентское приложение способно выполнять *удаленные вызовы процедур* (remote procedure calls, RPC) к серверному приложению. Вместо порта WCF следует

воспользоваться альтернативной технологией RPC, например gRPC, речь о которой и пойдет в этой книге.

В копилку знаний: дополнительная информация о CoreWCF приведена в репозитории GitHub по адресу https://github.com/CoreWCF/CoreWCF. А по ссылке https://devblogs.microsoft.com/dotnet/wcf-client-60-has-been-released/ вы сможете прочитать анонс о внедрении клиентской функциональности для вызова сервисов WCF/CoreWCF с помощью System.ServiceModel 6.0.

Общие принципы сервисов

Один из важнейших принципов архитектуры сервисов — оптимизация вызовов методов. Другими словами, старайтесь скомпоновать все данные, необходимые для выполнения операции, в одном вызове, а не в нескольких. Накладные расходы из-за удаленных вызовов — один из наибольших недостатков сервисов. Именно поэтому дробление на отдельные сервисы может негативно сказаться на архитектуре решения.

Советы по выбору сервиса

У каждой технологии сервисов есть свои преимущества и недостатки, касающиеся функционала (табл. 1.2).

Таблица 1.2. Преимущества и недостатки распространенных технологий сервисов

Функционал	Веб-API	OData	GraphQL	gRPC	SignalR
Клиенты могут запрашивать только необходимые данные	Нет	Да	Да	Нет	Нет
Минимальная версия HTTP	1.1	1.1	1.1	2.0	1.1
Поддержка браузеров	Да	Да	Да	Нет	Да
Формат данных	XML, JSON	XML, JSON	GraphQL (через JSON)	Бинарный	Различные
Сервисная документация	Swagger	Swagger	Нет	Нет	Нет
Генерация кода	Сторонняя	Сторонняя	Сторонняя	Google	Microsoft
Кэширование	Легко	Легко	Сложно	Сложно	Сложно