

OKIEM EKSPERTA

Zostań ekspertem .NET 8

Dobre praktyki, wzorce projektowe,
debugowanie i testowanie aplikacji



Mark J. Price



Helion 

<packt>

Tytuł oryginału: Tools and Skills for .NET 8: Get the career you want with good practices and patterns to design, debug, and test your solutions

Tłumaczenie: Wojciech Moch

ISBN: 978-83-289-2233-4

Copyright © Packt Publishing 2024. First published in the English language under the title 'Tools and Skills for .NET 8 – (9781837635207)'

Polish edition copyright © 2025 by Helion S.A.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

helion.pl/user/opinie/zoseks

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 230 98 63

e-mail: helion@helion.pl

WWW: helion.pl (księgarnia internetowa, katalog książek)

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści |

O autorze	21
O recenzentach	22
Wstęp	23
ROZDZIAŁ 1	
Wprowadzenie do narzędzi i umiejętności dla .NET	28
Wprowadzenie do tej książki i jej zawartości	29
Dodatkowe książki jako podstawa do nauki	29
Odbiorcy tej książki	30
Narzędzia	31
Umiejętności	32
Testowanie	32
Projektowanie systemów i rozwój kariery	33
Konfigurowanie środowiska programistycznego	34
Wybór odpowiedniego narzędzia i typu aplikacji do nauki	34
Jakiego sprzętu i oprogramowania używałem?	36
Wdrażanie wieloplatformowe	37
Pobieranie i instalowanie Visual Studio	37
Pobieranie i instalowanie Visual Studio Code	40
Pobieranie i instalowanie Ridera	43
Inne narzędzia firmy JetBrains	44
Narzędzia AI w Chrome	45
Wykorzystywanie repozytorium GitHuba dla tej książki	45
Pobieranie kodu rozwiązań z repozytorium GitHuba	45
Gdzie szukać pomocy?	46
Dokumentacja Microsoft Learn	46
Uzyskiwanie pomocy do narzędzia dotnet	46
Modele językowe, takie jak ChatGPT	47
Uzyskiwanie pomocy na Discordzie i innych forach	50

Konfigurowanie bazy danych i projektów na potrzeby tej książki	52
Używanie przykładowej relacyjnej bazy danych	53
Konfigurowanie SQL Server i bazy danych Northwind	54
Tworzenie biblioteki klas dla modeli encji przy użyciu serwera SQL Server	56
Tworzenie biblioteki klas dla kontekstu danych używanego w SQL Server	59
Tworzenie projektu testowego w celu sprawdzenia integracji bibliotek klas	62
Uruchamianie testów	63
Używanie .NET 9 w pracach z tą książką	64
Ćwiczenie i dalsza nauka	66
Ćwiczenie 1.1. Materiały dostępne wyłącznie online	66
Ćwiczenie 1.2. Ćwiczenia praktyczne	66
Ćwiczenie 1.3. Sprawdź swoją wiedzę	67
Ćwiczenie 1.4. Eksploracja tematów	67
Podsumowanie	67

ROZDZIAŁ 2

Pełne wykorzystanie edytora kodu	68
Wprowadzenie do powszechnie używanych narzędzi i funkcji	68
Funkcje refaktoryzacji	69
Wycinki kodu	70
Konfiguracja edytora	70
Asystenci AI	72
Narzędzia w Visual Studio 2022	72
Funkcje refaktoryzacji	73
Wycinki kodu	80
Konfiguracja edytora	86
Asystenci AI — GitHub Copilot	89
Nawigowanie w Visual Studio	91
Funkcje usprawniające edycję	91
Narzędzia w Visual Studio Code	95
Funkcje refaktoryzacji	96
Fragmenty kodu	96
Dekompilowanie zestawów .NET	98
Tworzenie aplikacji konsoli do zdekompilowania	98
Dekompilacja za pomocą rozszerzenia ILSpy dla Visual Studio	100
Przeglądanie kodu źródłowego za pomocą Visual Studio	103
Nie, nie można zapobiec dekompilacji	104
Obniżanie poziomu kodu C#	106

Własne szablony projektów i elementów	111
Tworzenie projektu dla szablonu	112
Testowanie szablonu projektu	117
Ćwiczenia i dalsza nauka	118
Ćwiczenie 2.1. Materiały dostępne wyłącznie online	118
Ćwiczenie 2.2. Ćwiczenia praktyczne	118
Ćwiczenie 2.3. Sprawdź swoją wiedzę	119
Podsumowanie	120

ROZDZIAŁ 3

Zarządzanie kodem źródłowym za pomocą Gita

Wprowadzenie do zarządzania kodem źródłowym	122
Funkcje zarządzania kodem źródłowym	122
Rodzaje systemów SCM	123
Popularne systemy SCM	123
Wprowadzenie do Gita	124
Funkcje Gita	125
Dlaczego nauka Gita jest trudna?	126
Role w zespole korzystającym z Gita	126
Pobieranie najnowszej wersji Gita	126
Integracja Gita z Visual Studio	126
Konfiguracja tożsamości w Gicie	127
Wymuszanie podpisów SSH w Gicie	128
Konfiguracja domyślnej gałęzi w Gicie	130
Uzyskiwanie pomocy dla poleceń Gita	130
Praca z Gitem	130
Rozpoczynanie pracy z repozytorium Gita	131
Tworzenie i dodawanie plików do repozytorium Gita — teoria	132
Śledzenie zmian w Gicie	133
Tworzenie repozytorium Gita — praktyka	134
Tworzenie nowego projektu	134
Umieszczanie plików w repozytorium	138
Cofanie commitów w Gicie	139
Czyszczenie commitu	140
Przechowalnia	141
Ignorowanie plików	143
Przeglądanie repozytoriów Gita	145
Wyświetlanie różnic w plikach	146
Wyświetlanie historii commitów	148
Filtrowanie historii commitów	152
Zarządzanie zdalnymi repozytoriami	152

Rozgałęzianie i scalanie	155
Przykład rozgałęziania i scalania	156
Usuwanie gałęzi i wyświetlanie ich listy	161
Podsumowanie najczęściej używanych poleceń Gita	161
Ćwiczenia i dalsza nauka	162
Ćwiczenie 3.1. Materiały dostępne wyłącznie online	163
Ćwiczenie 3.2. Ćwiczenia praktyczne	163
Ćwiczenie 3.3. Sprawdź swoją wiedzę	164
Ćwiczenie 3.4. Dalsza nauka	164
Podsumowanie	164

ROZDZIAŁ 4

Debugowanie i rozwiązywanie problemów z pamięcią 165

Strategie debugowania	165
Wprowadzenie do strategii debugowania	166
Zrozumieć problem	167
Jak rozpocząć debugowanie?	169
Kiedy przestać debugować?	170
Interaktywne debugowanie w Visual Studio	170
Tworzenie kodu do debugowania	170
Ustalanie punktu przerwania i rozpoczęcie debugowania	173
Pasek narzędzi debugowania	175
Okna debugowania	177
Kontrolowanie danych wyświetlanych w panelach debugowania	178
Debugowanie projektów testowych	180
Prośenie usługi GitHub Copilot Chat o pomoc w debugowaniu	182
Pamięć stosu i sterty	184
Jak typy referencyjne i typy wartości są przechowywane w pamięci	184
Czym jest niebezpieczny kod?	187
Wskaźniki	187
Czym jest boxing?	191
Mechanizm oczyszczania pamięci	192
Zarządzanie zasobami za pomocą interfejsu IDisposable	193
Narzędzia do analizy pamięci	195
Popularne narzędzia i umiejętności analizy pamięci	195
Narzędzia Visual Studio	196
Korzystanie z narzędzia do analizy pamięci w Visual Studio	197
Ćwiczenia i dalsza nauka	200
Ćwiczenie 4.1. Materiały dostępne wyłącznie online	200
Ćwiczenie 4.2. Ćwiczenia praktyczne	201

Ćwiczenie 4.3. Sprawdź swoją wiedzę	201
Ćwiczenie 4.4. Dalsza nauka	201
Podsumowanie	201

ROZDZIAŁ 5

Protokołowanie, śledzenie i metryki obserwowalności 203

Protokołowanie i śledzenie w .NET	203
Interfejs ILogger w .NET	205
Jak używać interfejsu ILogger?	207
Tworzenie usługi sieciowej do protokołowania	209
Testowanie podstawowych funkcji usługi sieciowej	214
Monitorowanie za pomocą metryk w .NET	215
Pojęcia dotyczące metryk i alertów	215
Implementowanie metryk	217
Wyświetlanie metryk	221
OpenTelemetry	222
Obsługiwane pakiety	223
Instrumentacja projektu ASP.NET Core	224
Przeglądanie danych telemetry	226
Ćwiczenia i dalsza nauka	230
Ćwiczenie 5.1. Materiały dostępne wyłącznie online	230
Ćwiczenie 5.2. Ćwiczenia praktyczne	230
Ćwiczenie 5.3. Sprawdź swoją wiedzę	230
Ćwiczenie 5.4. Dalsza nauka	231
Podsumowanie	231

ROZDZIAŁ 6

Dokumentowanie kodu, API i serwisów 232

Wprowadzenie do dokumentacji	232
Korzyści z dokumentacji	233
Kiedy nie dokumentować?	234
Dokumentowanie kodu źródłowego	235
Kiedy należy dokumentować kod źródłowy?	235
Dobre praktyki komentowania kodu źródłowego	237
Dokumentowanie publicznych API w bibliotekach klas	237
Dokumentowanie za pomocą komentarzy XML-a	239
Generowanie dokumentacji przy użyciu DocFX	246
Dodawanie własnych treści do dokumentacji	253
Język znaczników Markdown	254
Dokumentowanie usług	257
Kluczowe kwestie przy dokumentowaniu usług	257
Narzędzia do dokumentowania usług	258

Poznanie specyfikacji OpenAPI (OAS)	259
Wykorzystanie OpenAPI do dokumentowania usługi Minimal API	260
Dokumentowanie wizualne za pomocą diagramów Mermaid	264
Rysowanie diagramów Mermaid	266
Schematy blokowe	267
Diagramy klas	268
Zapisywanie diagramów Mermaid w formacie SVG	270
Ćwiczenia i dalsza nauka	271
Ćwiczenie 6.1. Materiały dostępne wyłącznie online	271
Ćwiczenie 6.2. Ćwiczenia praktyczne	272
Ćwiczenie 6.3. Sprawdź swoją wiedzę	272
Ćwiczenie 6.4. Dalsza nauka	273
Podsumowanie	273

ROZDZIAŁ 7

Obserwowanie kodu i dynamiczne wpływanie na jego wykonanie 274

Praca z refleksją i atrybutami	275
Metadane w zestawach .NET	275
Dynamiczne ładowanie zestawów i wykonywanie metod	283
Ostrzeżenie dotyczące refleksji i natywnego AOT	289
Ulepszenia refleksji w .NET 9	290
Więcej możliwości z refleksją	290
Praca z drzewami wyrażeń	291
Elementy drzewa wyrażeń	293
Wykonywanie najprostszego drzewa wyrażeń	293
Tworzenie generatorów kodu źródłowego	294
Implementacja najprostszego generatora źródeł	294
Praktyka i dalsza nauka	299
Ćwiczenie 7.1. Materiał dostępny wyłącznie online	299
Ćwiczenie 7.2. Ćwiczenia praktyczne	299
Ćwiczenie 7.3. Sprawdź swoją wiedzę	299
Ćwiczenie 7.4. Dalsza nauka	300
Podsumowanie	300

ROZDZIAŁ 8

Ochrona danych i aplikacji za pomocą kryptografii 301

Terminologia związana z ochroną danych	302
Techniki ochrony danych	302
Klucze i ich rozmiary	303
Wektory inicjujące i rozmiary bloków	304
Sól	305
Generowanie kluczy i wektorów IV	305

Szyfrowanie i rozszyfrowywanie danych	306
Szyfrowanie symetryczne z użyciem AES	307
Haszowanie danych	313
Haszowanie z użyciem algorytmu SHA-256	314
Podpisywanie danych	318
Podpisywanie z wykorzystaniem SHA-256 i RSA	319
Generowanie liczb losowych na potrzeby kryptografii	321
Uwierzytelnianie i autoryzacja użytkowników	323
Mechanizmy uwierzytelniania i autoryzacji	324
Implementowanie uwierzytelniania i autoryzacji	326
Ochrona funkcji udostępnianych przez aplikację	329
Uwierzytelnianie i autoryzacja w rzeczywistych zastosowaniach	330
Co nowego w .NET 9?	330
Metoda <code>CryptographicOperations.HashData()</code>	330
Algorytm KMAC	331
Ćwiczenia i dalsza nauka	331
Ćwiczenie 8.1. Materiały dostępne wyłącznie online	331
Ćwiczenie 8.2. Ćwiczenia praktyczne	332
Ćwiczenie 8.3. Sprawdź swoją wiedzę	332
Ćwiczenie 8.4. Dalsza nauka	333
Podsumowanie	333

ROZDZIAŁ 9

Tworzenie chatu używającego modelu LLM	334
Wprowadzenie do modeli LLM	335
Jak działają modele LLM?	335
Uzyskiwanie dostępu do modelu LLM	337
Używanie Semantic Kernel z modelem OpenAI	340
Czym jest Semantic Kernel?	341
Czym są funkcje?	347
Dodawanie funkcji	348
Dodanie pamięci sesji i włączanie wielu funkcji	353
Strumieniowanie wyników	355
Dodawanie protokołowania i zwiększanie niezawodności	356
Korzystanie z lokalnych modeli LLM	358
Hugging Face	358
Ollama	359
LM Studio	365
Ćwiczenia i dalsza nauka	367
Ćwiczenie 9.1. Materiały dostępne wyłącznie online	367
Ćwiczenie 9.2. Ćwiczenia praktyczne	368

Ćwiczenie 9.3. Sprawdź swoją wiedzę	369
Ćwiczenie 9.4. Dalsza lektura	369
Podsumowanie	369

ROZDZIAŁ 10

Wstrzykiwanie zależności, kontenery i czas życia serwisów 370

Wprowadzenie do wstrzykiwania zależności	371
Dlaczego warto korzystać z DI?	372
Mechanizmy wstrzykiwania zależności w .NET	372
Przykłady we współczesnym .NET	374
Czas życia rejestrowanych zależności	376
Rejestrowanie wielu implementacji	377
Kiedy rzucane są wyjątki?	378
Najlepsze praktyki wstrzykiwania zależności	379
Implementacja generycznego hosta w .NET	379
Najważniejsze funkcje generycznego hosta .NET	379
Budowanie generycznego hosta .NET	380
Poznanie usług i zdarzeń hosta	384
Metody rejestrowania usług	388
Grafy zależności i rozwiązywanie usług	389
Usuwanie usług	390
Wstrzykiwanie zależności w ASP.NET Core	390
Rejestrowanie usług dla funkcji za pomocą metod rozszerzających	391
Gdy nie można użyć wstrzykiwania przez konstruktor	391
Rozwiązywanie usług podczas uruchamiania aplikacji	393
Wstrzykiwanie zależności w widokach	393
Wstrzykiwanie do metod akcji i minimalnego API	394
Ćwiczenia i dalsza nauka	394
Ćwiczenie 10.1. Materiały dostępne wyłącznie online	394
Ćwiczenie 10.2. Ćwiczenia praktyczne	394
Ćwiczenie 10.3. Sprawdź swoją wiedzę	395
Ćwiczenie 10.4. Dalsza nauka	395
Podsumowanie	395

ROZDZIAŁ 11

Testowanie i mockowanie 396

Wprowadzenie do wszystkich rodzajów testów	397
Testy jednostkowe	397
Testy integracyjne, end-to-end i bezpieczeństwa	398
Testy wydajności, obciążenia i wytrzymałości	398
Testy funkcjonalne i testy użyteczności	399

Terminologia testowania	399
Cechy wszystkich dobrych testów	399
Wyniki testów	401
Dublery, mocki i zatyczki	401
Przyjęcie odpowiedniego nastawienia do testowania	402
Zalety i wady TDD	403
Główne zasady TDD	403
Zalety TDD	403
Wady TDD	404
Dobre praktyki TDD	404
Testy jednostkowe	405
Jak bardzo izolowane powinny być testy jednostkowe?	405
Nazewnictwo używane w testach jednostkowych	406
Tworzenie testów jednostkowych za pomocą xUnit	406
Często używane atrybuty xUnit	408
Tworzenie klasy do przetestowania	409
Pisanie prostych testów jednostkowych	410
Metody testujące z parametrami	413
Pozytywne i negatywne wyniki testów	417
Sygnały ostrzegawcze w testach jednostkowych	418
Wyświetlanie wyników podczas wykonywania testów	418
Przygotowanie i czyszczenie środowiska testowego	419
Kontrola przygotowania testów	422
Mockowanie w testach	425
Biblioteki do mockowania	427
Wykorzystanie NSubstitute do tworzenia dubli	428
Przykład mockowania z użyciem NSubstitute	429
Tworzenie płynnych asercji w testach jednostkowych	432
Tworzenie asercji dla ciągów znaków	432
Tworzenie asercji dla kolekcji i tablic	433
Tworzenie asercji dla dat i godzin	434
Generowanie fałszywych danych z biblioteką Bogus	435
Projekt testujący fałszywe dane	437
Pisanie metody z użyciem fałszywych danych	439
Ćwiczenia i dalsza nauka	440
Ćwiczenie 11.1. Materiały dostępne wyłącznie online	441
Ćwiczenie 11.2. Ćwiczenia praktyczne	441
Ćwiczenie 11.3. Sprawdź swoją wiedzę	442
Ćwiczenie 11.4. Dalsza nauka	442
Podsumowanie	443

ROZDZIAŁ 12

Testy integracyjne i testy bezpieczeństwa	444
Podstawy testów integracyjnych	445
Jakie systemy zewnętrzne testować?	446
Współdzielenie środowiska w testach integracyjnych	446
Analiza przykładowego testu integracyjnego	447
Testy integracyjne z wykorzystaniem magazynów danych	449
Deweloperskie instancje bazy danych i migracje baz danych	449
Cykl życia danych	451
Testowanie usług za pomocą tuneli deweloperskich	453
Instalowanie interfejsu wiersza poleceń tunelu deweloperskiego	454
Używanie wiersza poleceń tunelu deweloperskiego i serwisu echo	454
Używanie tunelu deweloperskiego w projekcie ASP.NET Core	456
Wprowadzanie do testów bezpieczeństwa	460
Open Web Application Security Project	462
OWASP Top 10	462
Modelowanie zagrożeń	467
Ćwiczenia i dalsza nauka	468
Ćwiczenie 12.1. Materiały dostępne wyłącznie online	469
Ćwiczenie 12.2. Ćwiczenia praktyczne	469
Ćwiczenie 12.3. Sprawdź swoją wiedzę	469
Ćwiczenie 12.4. Dalsza nauka	470
Podsumowanie	470

ROZDZIAŁ 13

Mierzenie wydajności i testy obciążeniowe	471
Mierzenie wydajności	471
Znaczenie pomiaru bazowego	472
Notacja dużego O	473
Metryki statystyczne	475
Używanie biblioteki BenchmarkDotNet do mierzenia wydajności	476
Unikanie błędów podczas testów wydajności	482
Rozpoznawanie kiepskich postów na temat wydajności	485
Testy obciążeniowe i wytrzymałościowe	486
Apache JMeter	489
Bombardier — szybkie wieloplatformowe narzędzie do testów wydajności żądań HTTP	489
Używanie Bombardiera	490
Pobieranie Bombardiera	491
Porównywanie serwisu sieciowego kompilowanego za pomocą AOT i tradycyjnie	491
Interpretacja wyników Bombardiera	498

NBomber — framework do testów obciążeniowych	500
Scenariusze w NBomberze	500
Symulacje obciążenia	501
Typy NBombera	501
Przykład projektu NBombera	501
Ćwiczenia i dalsza nauka	505
Ćwiczenie 13.1. Materiały dostępne wyłącznie online	505
Ćwiczenie 13.2. Ćwiczenia praktyczne	506
Ćwiczenie 13.3. Sprawdź swoją wiedzę	506
Ćwiczenie 13.4. Dalsza nauka	506
Podsumowanie	507

ROZDZIAŁ 14

Testy funkcjonalne i end-to-end	508
Testy funkcjonalne i end-to-end	508
Przykład 1. Testowanie usługi Web API	509
Przykład 2. Testowanie strony internetowej ASP.NET Core	509
Przykład 3. Testowanie aplikacji w czasie rzeczywistym SignalR	510
Testowanie webowych interfejsów użytkownika przy użyciu Playwrighta	511
Korzyści dla programistów .NET	512
Alternatywy dla Playwrighta	513
Typy testów w Playwrightcie	514
Metody automatyzacji strony w Playwrightcie	514
Metody lokalizacji elementów w Playwrightcie	515
Metody automatyzacji lokalizatorów w Playwrightcie	516
Testowanie typowych scenariuszy za pomocą aplikacji eShopOnWeb	517
Interakcja z interfejsem użytkownika	524
Wybieranie elementów z list rozwijanych i klikanie elementów	524
Przesyłanie formularzy, uwierzytelnianie i walidacja	526
Testowanie responsywnego projektu	526
Aplikacje SPA i dynamiczna zawartość	528
Generowanie testów z Playwright Inspector	529
Testowanie serwisów przy użyciu xUnit	533
Tworzenie serwisu internetowego na potrzeby testów	534
Tworzenie projektu testowego	535
Ćwiczenia i dalsza nauka	536
Ćwiczenie 14.1. Materiały dostępne wyłącznie online	536
Ćwiczenie 14.2. Ćwiczenia praktyczne	536
Ćwiczenie 14.3. Sprawdź swoją wiedzę	537
Ćwiczenie 14.4. Dalsza nauka	537
Podsumowanie	537

ROZDZIAŁ 15**Konteneryzacja przy użyciu Dockera 538**

Wprowadzenie do konteneryzacji	538
Jak działają kontenery i jakie są ich zalety?	540
Docker, Kubernetes i .NET Aspire	541
Rejestry kontenerów	543
Pojęcia związane z Dockerem	545
Narzędzia i technologie Dockera	546
Polecenia interfejsu wiersza poleceń Dockera	546
Tworzenie obrazów przy użyciu plików Dockerfile	548
Konfigurowanie portów i uruchamianie kontenera	551
Tryb interaktywny	552
Zmienne środowiskowe	553
Często używane obrazy kontenerów Dockera	554
Obrazy kontenerów .NET	554
System CVE i odchudzone Ubuntu	555
Zarządzanie kontenerami za pomocą Dockera	556
Instalowanie Dockera i używanie gotowych obrazów	556
Hierarchia obrazów Dockera i warstwy obrazów	559
Konteneryzacja własnych projektów .NET	563
Konteneryzacja projektu aplikacji konsoli	563
Publikowanie aplikacji do kontenera Dockera	565
Konteneryzacja projektu aplikacji ASP.NET Core	570
Praca z kontenerami testowymi	573
Jak działa Testcontainers dla .NET?	573
Przykład użycia	574
Ćwiczenia i dalsza nauka	575
Ćwiczenie 15.1. Materiały dostępne wyłącznie online	575
Ćwiczenie 15.2. Ćwiczenia praktyczne	575
Ćwiczenie 15.3. Sprawdź swoją wiedzę	575
Ćwiczenie 15.4. Dalsza nauka	576
Podsumowanie	576

ROZDZIAŁ 16**Oprogramowanie chmurowe z .NET Aspire 577**

Wprowadzenie do .NET Aspire	578
Co mówi zespół Aspire?	579
Aspire w edytorach kodu i w interfejsie wiersza poleceń	580
Uruchamianie rozwiązania Aspire	580
Typy projektów Aspire	581
Typy zasobów Aspire	582

Model aplikacji Aspire i orkiestracja	582
Szablony projektów Aspire	584
Szablon aplikacji startowej Aspire	585
Tworzenie aplikacji startowej Aspire	586
Przeglądanie rozwiązania startowego Aspire	589
Dokładniejsze spojrzenie na Aspire	592
Pulpit nawigacyjny dla deweloperów	592
Projekt AppHost — orkiestracja zasobów	593
Projekt ServiceDefaults — centralizacja konfiguracji	596
Inne projekty funkcyjne w rozwiązaniu	597
Konfigurowanie Redisa	599
Komponenty Aspire	600
Protokołowanie, śledzenie i metryki obserwowalności	602
Docker kontra Podman	602
Oczekiwanie na gotowość kontenerów	603
A co z Dapr, Orleans i Project Tye?	603
Aspire w nowych i istniejących rozwiązaniach	606
Tworzenie nowego rozwiązania Aspire	606
Aspire i PostgreSQL	611
Używanie woluminów danych i konfigurowanie stabilnego hasła	612
Dodanie Aspire do istniejącego rozwiązania	612
Aplikacja referencyjna eShop	614
Wdrażanie aplikacji za pomocą Aspire	619
Ćwiczenia i dalsza nauka	620
Ćwiczenie 16.1. Materiały dostępne wyłącznie online	621
Ćwiczenie 16.2. Ćwiczenia praktyczne	621
Ćwiczenie 16.3. Sprawdź swoją wiedzę	622
Ćwiczenie 16.4. Dalsza nauka	622
Podsumowanie	622

ROZDZIAŁ 17

Wzorce i zasady projektowe	623
Zasady SOLID	624
Zasada pojedynczej odpowiedzialności (Single Responsibility Principle — SRP)	624
Zasada otwarte — zamknięte (Open/Closed Principle — OCP)	627
Zasada podstawień Liskov (Liskov Substitution Principle — LSP)	629
Zasada segregacji interfejsów (Interface Segregation Principle — ISP)	633
Zasada odwracania zależności (Dependency Inversion Principle — DIP)	637

Wzorce projektowe	640
Wzorce kreacyjne	645
Strukturalne wzorce projektowe	647
Behawioralne wzorce projektowe	650
Zasady projektowe	653
Zasada DRY	653
Zasada KISS	653
YAGNI	654
Prawo Demeter	655
Kompozycja jest lepsza od dziedziczenia	656
Zasada najmniejszego zaskoczenia	658
Algorytmy i struktury danych	660
Algorytmy sortowania	661
Algorytmy wyszukiwania	661
Algorytmy struktury danych	662
Algorytmy haszujące	662
Algorytmy rekurencyjne	662
Gdzie dowiesz się więcej o algorytmach i strukturach danych?	663
Ćwiczenie i dalsza nauka	663
Ćwiczenie 17.1. Materiały dostępne wyłącznie online	663
Ćwiczenie 17.2. Ćwiczenia praktyczne	663
Ćwiczenie 17.3. Sprawdź swoją wiedzę	664
Ćwiczenie 17.4. Dalsza nauka	664
Podsumowanie	664

ROZDZIAŁ 18

Podstawy architektury rozwiązań i oprogramowania	665
Wprowadzenie do architektury oprogramowania i architektury rozwiązań	665
Architektura oprogramowania	666
Architektura rozwiązań	667
Konceptje architektury oprogramowania	667
Style architektury oprogramowania	669
Konceptje architektury rozwiązania	675
Wnioski	676
Czysta architektura według Wujka Boba	677
Konceptje czystej architektury	678
Dobre praktyki w czystej architekturze w .NET	680
Rysowanie diagramów za pomocą Mermaid	681
Mermaid w architekturze oprogramowania i rozwiązań	681
Rodzaje diagramów Mermaid	682
Schematy blokowe w Mermaid	683

Ćwiczenie i dalsza nauka	691
Ćwiczenie 18.1. Materiał dostępny wyłącznie online	692
Ćwiczenie 18.2. Ćwiczenia praktyczne	692
Ćwiczenie 18.3. Sprawdź swoją wiedzę	692
Ćwiczenie 18.4. Dalsza nauka	693
Podsumowanie	694

ROZDZIAŁ 19

Kariera, praca zespołowa i rozmowy kwalifikacyjne 695

Praca w zespole programistycznym	695
Praca inżyniera oprogramowania	696
Ścieżka kariery	697
Role osób w zespole programistycznym, z którymi będziesz współpracować	698
Proces onboardingu	701
Jak poprosić o szkolenie i rozwój?	701
Programowanie w parach	704
Poszukiwanie pracy	707
Przed złożeniem podania	707
Przygotowanie do rozmowy kwalifikacyjnej	710
Przykładowe pytania na rozmowie kwalifikacyjnej	730
1. Narzędzia wiersza poleceń w .NET	731
2. Podstawy Gita	732
3. Entity Framework Core	734
4. Interfejsy i klasy abstrakcyjne	736
5. Właściwości i indeksery	736
6. Typy generyczne	736
7. Delegaty i zdarzenia	736
8. LINQ	736
9. Programowanie asynchroniczne z użyciem słów kluczowych async i await	737
10. Zarządzanie pamięcią i jej oczyszczanie	737
11. Różnice między nowoczesnym .NET a .NET Framework	737
12. Możliwości wieloplatformowe	737
13. .NET Standard	737
14. Wstrzykiwanie zależności w .NET	737
15. Oprogramowanie pośredniczące w ASP.NET Core	738
16. Konfiguracja i wzorzec Opcje	738
17. Hosting i serwer Kestrel	738
18. Typy danych	738
19. Globalizacja i lokalizacja	738

20. Struktury sterujące	738
21. Obsługa wyjątków	738
22. Strategie tworzenia gałęzi w Gicie	739
23. Przeglądy kodu i programowanie w parach	739
24. Metodyki agile i scrum	739
25. Standardy dokumentacji	739
26. Umiejętność rozwiązywania problemów	739
27. Narzędzia do zarządzania projektami	739
28. Techniki szacowania	739
29. Współpraca w zespole	740
30. Przywództwo i mentoring	740
31. Wzorzec MVC	740
32. Składnia Razora	740
33. Web API	740
34. Najlepsze praktyki dotyczące usług RESTful	740
35. SignalR do pracy w czasie rzeczywistym	740
36. Zarządzanie stanem	741
37. Uwierzytelnianie i autoryzacja	741
38. Blazor WebAssembly	741
39. Korzyści wynikające z używania mikroserwisów	741
40. Wyzwania związane z architekturą mikroserwisów	741
41. Kontenery Dockera i .NET	741
42. Wzorce komunikacji w mikroserwisach	741
43. Odporność na błędy i obsługa błędów przejściowych	742
44. Śledzenie rozproszone	742
45. Kontrola stanu zdrowia i monitorowanie aplikacji	742
46. AutoMapper, metody rozszerzające i operator jawny	742
47. Podstawy ADO.NET	742
48. Optymalizacja wydajności Entity Framework Core	742
49. Frameworki do testów jednostkowych, np. xUnit	743
50. Frameworki do mockowania, np. NSubstitute	743
51. Strategie testowania integracyjnego	743
52. Testowanie wydajności	743
53. Testowanie bezpieczeństwa	743
54. Automatyczne testowanie interfejsu użytkownika	743
55. Zasady SOLID	743
56. Wzorzec projektowy Singleton	744
57. Wzorzec projektowy Fabryka	744
58. Wykrywanie wycieków pamięci	744
59. Metodyki programistyczne	744
60. Notacja wielkiego O	744
Uczysz się, kiedy popełniasz błędy	744

Ćwiczenia i dalsza nauka	745
Ćwiczenie 19.1. Materiały dostępne wyłącznie online	745
Ćwiczenie 19.2. Ćwiczenia praktyczne	745
Ćwiczenie 19.3. Sprawdź swoją wiedzę	745
Ćwiczenie 19.4. Dalsza nauka	746
Podsumowanie	746
ROZDZIAŁ 20	
Epilog	747
Kolejne kroki na ścieżce nauki .NET	747
Książki uzupełniające, które pomogą kontynuować naukę	747
Dziewiąte wydanie C# 12 i .NET 8 już wkrótce dla .NET 9	748
Planowana trylogia .NET 10	749
Książki do dalszej nauki	750
Powodzenia!	750
Skorowidz	751

Tworzenie chatu używającego modelu LLM

Integracja **wielkiego modelu językowego** (ang. *Large Language Model* — LLM), takiego jak GPT-4o firmy OpenAI, z projektem .NET może znacząco zwiększyć jego możliwości, oferując rozbudowane funkcje przetwarzania języka naturalnego, generowania treści itd.

Ten rozdział został poświęcony budowie usługi chatu wykorzystującej model LLM, która dostosuje standardowy model do własnych potrzeb poprzez uzupełnienie go o niestandardowe informacje, w tym biografię autora tej książki oraz dane z bazy Northwind. Dzięki temu użytkownik będzie mógł zadawać pytania dotyczące autora oraz bazy danych fikcyjnej korporacji.

Wskazówka

W momencie pisania tego rozdziału usługi API chmurowych modeli LLM są płatne. W tym rozdziale skorzystamy z jednej z najtańszych opcji: GPT-3.5 Turbo firmy OpenAI. Jak podaje ona na swoich stronach: „GPT-3.5 Turbo to nasz szybki i tani model do prostszych zadań. Model *gpt-3.5-turbo-0125* jest flagowym modelem tej rodziny — obsługuje kontekst o wielkości 16K i jest zoptymalizowany na potrzeby prowadzenia dialogów”. Koszt danych wejściowych wynosi 0,50 USD za 1 mln tokenów, a koszt danych wyjściowych to 1,50 USD za 1 mln tokenów. Wykonanie zadań z tego rozdziału kosztowało mnie mniej niż 0,03 USD.

Można też użyć modelu GPT-4o, ponieważ: „GPT-4o to nasz najbardziej zaawansowany model multimodalny, który jest szybszy i tańszy niż GPT-4 Turbo, a także ma większe możliwości przetwarzania obrazów. Model obsługuje kontekst o wielkości 128K i jest wyposażony w bazę wiedzy aktualną na październik 2023 r.”. Koszt danych wejściowych wynosi 5,00 USD za 1 mln tokenów, a koszt danych wyjściowych to 15,00 USD za 1 mln tokenów.

W tym rozdziale zajmiemy się następującymi zagadnieniami:

- wprowadzenie do modeli LLM,
- korzystanie z funkcji Semantic Kernel z modelem OpenAI,
- uruchamianie lokalnych modeli LLM.

Wprowadzenie do modeli LLM

Pojęcia **wielki model językowy (LLM)** i **generatywny, wstępnie wytrenowany transformator** (ang. *Generative Pre-trained Transformer* — GPT) są często używane zamiennie, ale istnieją między nimi subtelne różnice:

- LLM to bardziej ogólne określenie, odnoszące się do każdego modelu językowego z dużą liczbą parametrów (liczoną w miliardach), który został wytrenowany na ogromnych ilościach danych tekstowych i potrafi wykonywać różnorodne zadania językowe, takie jak tłumaczenia, streszczanie, odpowiadanie na pytania i inne.
- GPT to seria modeli opracowanych przez firmę OpenAI. Można to porównać do różnicy między iPhone’em a smartfonami — iPhone to rodzaj smartfona. Podobnie GPT jest rodzajem LLM. „Transformator” w nazwie opisuje architekturę, na której są zbudowane wszystkie modele rodziny GPT. Jest to przełomowa struktura modelu wprowadzona w 2017 r., która szczególnie dobrze radzi sobie ze sekwencjami danych takimi jak zdania w tekście. Chociaż wszystkie modele GPT bazują na architekturze transformatora, nie oznacza to, że robią to wszystkie modele LLM.

Zatem każdy model GPT jest modelem LLM, ale nie każdy model LLM jest modelem GPT.

Jak działają modele LLM?

Modele LLM działają na zasadach uczenia maszynowego, w szczególności wykorzystując techniki głębokiego uczenia. Modele LLM są trenowane na ogromnych zbiorach danych tekstowych pochodzących z książek, stron internetowych takich jak Stack Overflow i Reddit, artykułów oraz innych treści tekstowych. Dzięki temu dane te zapewniają modelowi szeroką znajomość języka, kontekstu i wiedzy faktograficznej aż do momentu zakończenia treningu.

Uwaga

Więcej informacji o partnerstwie OpenAI ze Stack Overflow znajdziesz pod tym adresem: <https://stackoverflow.co/company/press/archive/openai-partnership>. Szczegóły dotyczące partnerstwa OpenAI z Redditem dostępne są tutaj: <https://openai.com/index/openai-and-reddit-partnership>.

Modele LLM wykorzystują typ architektury sieci neuronowych nazywanej **transformatorem**. Architektura ta składa się z wielu warstw mechanizmów uwagi i sieci typu *feed-forward*. Do jej kluczowych elementów należą:

- **Mechanizmy uwagi.** Pozwalają modelowi ważyć znaczenie różnych słów w zdaniu, pomagając skupić się na istotnych częściach wejścia przy generowaniu odpowiedzi.
- **Sieci typu *feed-forward*.** Przetwarzają wejścia z mechanizmów uwagi i generują wyniki na każdym poziomie przetwarzania.

Uwaga

Obecna „ograniczona inteligencja” modeli LLM wynika z ich jednokierunkowej natury (ang. *feed-forward-only*). Eksperymenty z systemami wykorzystującymi sprzężenie zwrotne mogą doprowadzić do powstania systemów o wyższym poziomie rozpoznawania informacji, umożliwiając autorefleksję i ciągłe uczenie się.

Wejściowy tekst jest dzielony na mniejsze jednostki zwane tokenami. Tokeny mogą być słowami, częściami słów lub nawet pojedynczymi znakami. Tokenizacja pomaga modelowi efektywnie obsługiwać duże zbiory słów. Na przykład zdanie „Witaj, świecie!” może zostać podzielone na takie tokeny: ["Witaj", " ", "świecie", "!"].

Proces trenowania składa się z następujących kroków:

- **Przetwarzanie w przód** (ang. *forward pass*). Tekst wejściowy przechodzi przez model, który generuje najbardziej prawdopodobne wyjście (np. przewiduje następne słowo).
- **Obliczanie straty**. Przewidywanie modelu jest porównywane z rzeczywistym kolejnym słowem z danych treningowych i obliczana jest wartość straty. Strata mierzy różnicę między przewidywaniem a rzeczywistym słowem.
- **Przetwarzanie wstecz i optymalizacja**. Wykorzystując techniki takie jak wsteczna propagacja błędów i gradientowy spadek, model dostosowuje swoje wewnętrzne parametry, aby zminimalizować stratę. Proces ten powtarza się wiele razy na zbiorze danych treningowych.

Po początkowym treningu modele są często dostrajane na specyficznych zbiorach danych, aby poprawić wydajność w określonych dziedzinach lub zadaniach. Dostrajanie (ang. *fine-tuning*) polega na dalszym trenowaniu modelu na mniejszym, bardziej ukierunkowanym zbiorze danych. Na przykład wydawnictwo mogłoby dostroić własny model LLM za pomocą treści wszystkich swoich opublikowanych książek i wykorzystać go do obsługi chatbotów w swoich kanałach kontaktu z klientami. Chatbot mógłby wtedy odpowiadać na pytania dotyczące książek wydawnictwa i pomagać czytelnikom w razie problemów z konkretnymi stronami lub rozdziałami.

Podczas wnioskowania (czyli użytkowania) przeszkolony model generuje tekst na podstawie otrzymanego wejścia. W tym celu musi wykorzystać:

- **Rozumienie kontekstu**. Model wykorzystuje swoją wytrenowaną wiedzę, aby zrozumieć kontekst wejściowego tekstu.
- **Przewidywanie**. Model przewiduje najbardziej prawdopodobny następny token lub sekwencję tokenów, generując spójny tekst odpowiedni dla danego wejścia.

Modele LLM potrafią radzić sobie z niejednoznacznością i utrzymywać kontekst w dłuższych fragmentach tekstu dzięki zastosowaniu mechanizmów uwagi. Potrafią przypominać sobie istotne informacje z wcześniejszych części rozmowy, co pozwala im generować spójne i kontekstowo trafne odpowiedzi. Używane w modelach okna kontekstowe są nieustannie rozwijane.

Oprócz ogromnych możliwości modele LLM mają też pewne ograniczenia:

- **Stronniczość.** Modele mogą dziedziczyć uprzedzenia obecne w danych treningowych, w efekcie czego generują stronnicze lub niesprawiedliwe odpowiedzi.
- **Nieściśłości faktograficzne.** Modele generują tekst podobny do ludzkich wypowiedzi, mogą również tworzyć niepoprawne lub nonsensowne informacje. Jest to szczególnie niebezpieczne, ponieważ tworzą realistycznie brzmiące porady, np. prawne. Modele LLM należy zatem traktować jak młodszych członków zespołu, których praca zawsze wymaga weryfikacji.
- **Zależność od danych.** Ich wiedza jest ograniczona do danych, na których zostały wytrenowane, a te mogą być nieaktualne w czasie, gdy używasz modelu.

Modele LLM znajdują zastosowanie w różnych dziedzinach, np.:

- **Chatboty i wirtualni asystenci.** Zapewniają użytkownikom poziom interakcji przypominający rozmowę z prawdziwym człowiekiem, np. ChatGPT-4 czy Microsoft Copilot.
- **Tworzenie treści.** Udzielają pomocy w pisaniu artykułów, raportów, a nawet tekstów kreatywnych, np. Microsoft Designer: <https://designer.microsoft.com/>.
- **Tłumaczenia.** Pozwalają na oferowanie usług tłumaczenia tekstów.
- **Pomoc przy programowaniu.** Udzielają pomocy programistom, sugerując kierunek tworzenia kodu i wspomagając przy debugowaniu, np. GitHub Copilot i JetBrains AI Assistant.

Skoro już wiemy, jak działają LLM i jakie mają zalety, dowiedzmy się, jak można uzyskać do nich dostęp.

Uzyskiwanie dostępu do modelu LLM

Na potrzeby naszego małego chatu będziemy potrzebować dostępu do modelu LLM. Najprostszym sposobem uzyskania dostępu do chmurowego modelu LLM przystosowanego do integracji z projektami .NET jest skorzystanie z usług OpenAI lub Azure OpenAI. W momencie pisania tego tekstu aplikowanie o model Azure OpenAI wymaga posiadania adresu e-mail w domenie firmowej. Konta Gmail lub MSN nie są akceptowane. Dla tego w tym rozdziale będziemy korzystać głównie z modeli firmy OpenAI.

Uwaga

Jeżeli masz adres e-mail w domenie firmowej i wolisz korzystać z płatnej usługi Azure, to pod tym adresem dowiesz się, jak przygotować dla siebie model Azure OpenAI: <https://learn.microsoft.com/azure/ai-services/openai/how-to/create-resource>.

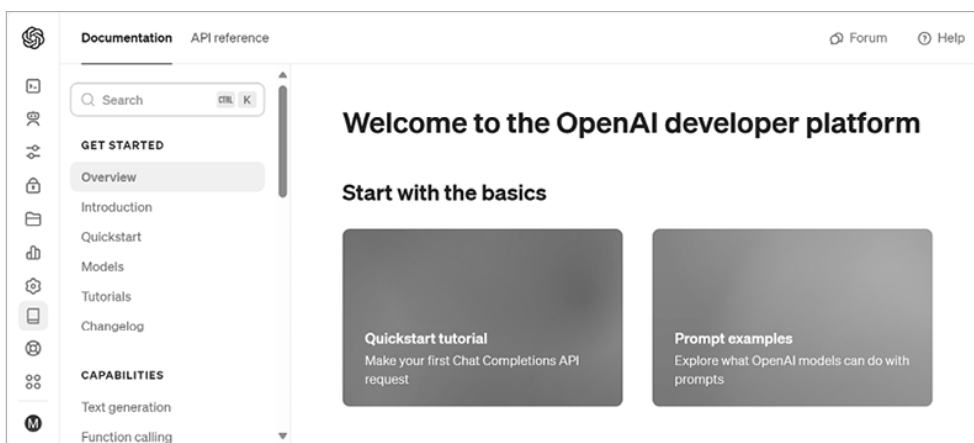
Aby skorzystać z chmurowego modelu LLM, będą nam potrzebne dwie informacje:

- **Klucz API.** Udostępniany jest w portalu usługi LLM i powinien być przechowywany w bezpiecznym miejscu. Każdy, kto posiada Twój klucz API, może korzystać z zasobów Twojego konta.

- **Identyfikator modelu chatu.** W naszym przypadku użyjemy modelu *gpt-3.5-turbo*, który jest dobrą kombinacją niskich kosztów i rozsądnego poziomu inteligencji. Więcej o tym modelu można przeczytać pod adresem <https://platform.openai.com/docs/models/gpt-3-5-turbo>.

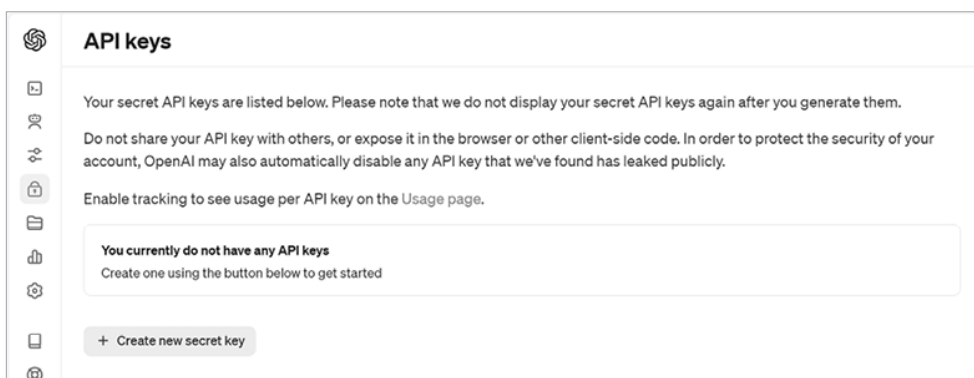
No to zaczynamy:

1. Otwórz przeglądarkę i przejdź na stronę rejestracji firmy OpenAI pod adresem <https://platform.openai.com/signup>.
2. Po zarejestrowaniu się i zalogowaniu zobaczysz stronę główną platformy deweloperskiej OpenAI, taką jak na rysunku 9.1.



Rysunek 9.1. Strona główna platformy deweloperskiej OpenAI

3. W menu nawigacyjnym po lewej stronie przejdź do sekcji *API keys* lub przejdź bezpośrednio pod adres <https://platform.openai.com/api-keys>. Kliknij tutaj przycisk *+Create new secret key*, tak jak na rysunku 9.2.



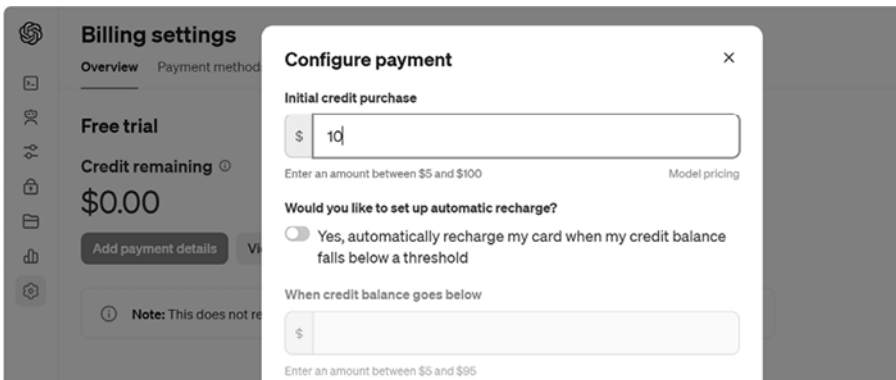
Rysunek 9.2. Strona kluczy dostępu do API OpenAI

4. Wprowadź nazwę, np. *tools-skills-net8*, a następnie kliknij przycisk *Create secret key*.
5. Gdy pojawi się okno dialogowe *Save your key*, skopiuj klucz do schowka i zapisz go w bezpiecznym i dostępnym miejscu.

Dobra praktyka

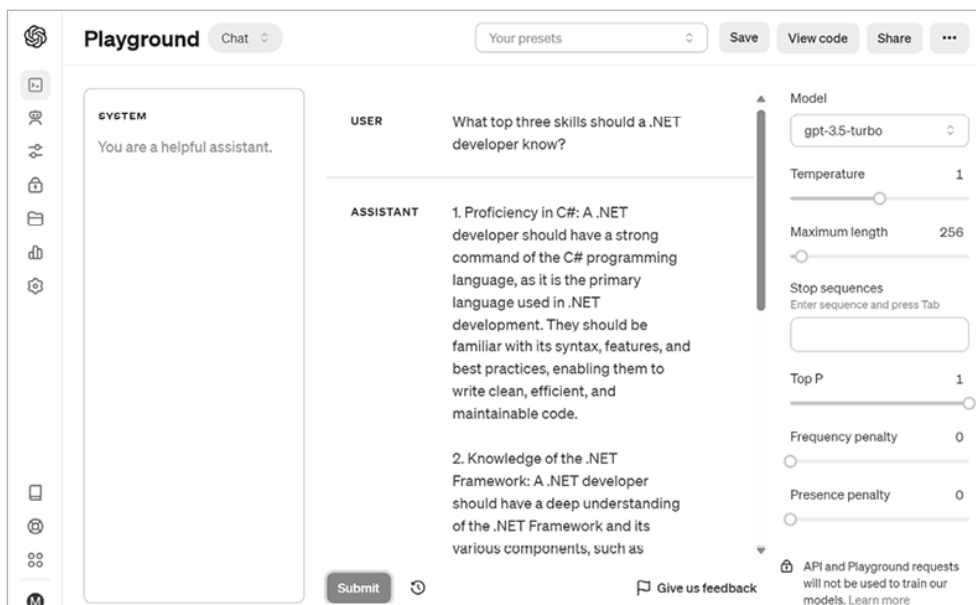
Zapisz swój klucz w bezpiecznym miejscu, ponieważ nie ma możliwości ponownego wyświetlenia go w koncie OpenAI. Jeżeli zgubisz klucz, trzeba będzie wygenerować nowy.

6. W menu nawigacyjnym po lewej stronie przejdź do pozycji *Settings/Billing* i kliknij przycisk *Payment methods*, a następnie dodaj nową metodę płatności, np. kartę Visa, lub przejdź pod adres <https://platform.openai.com/account/billing/overview>.
7. Na stronie konfigurowania rozliczeń w zakładce *Overview* kliknij przycisk *Add to credit balance* i dodaj środki. Zazwyczaj wydasz jedynie kilka centów, więc zalecam dodanie minimalnej kwoty. Ja dodałem 10 USD, ale możesz dodać minimalną wartość, tak jak na rysunku 9.3.



Rysunek 9.3. Wpłacanie pieniędzy na konto OpenAI

8. W menu nawigacyjnym po lewej stronie przejdź do sekcji *Playground* i upewnij się, że na liście rozwijanej obok tytułu strony wybrana jest wartość *Chat*, a z listy modeli wybrana jest wartość *gpt-3.5-turbo*. Możesz także przejść pod adres <https://platform.openai.com/playground?model=gpt-3.5-turbo>. Oczywiście można też dokładniej wybrać numer wersji modelu, np. *gpt-3.5-turbo-0125*, ale bardziej ogólna nazwa oznacza, że zawsze będzie używana najnowsza wersja.
9. Aby przetestować dostępność swoich środków, wpisz następujące pytanie użytkownika: **Jakie trzy najważniejsze umiejętności powinien znać programista .NET?**, kliknij przycisk *Submit*, a następnie przyjrzyj się wynikowi, który pokazano na rysunku 9.4.



Rysunek 9.4. Testowanie dostępnych środków w piaskownicy

Ostrzeżenie!

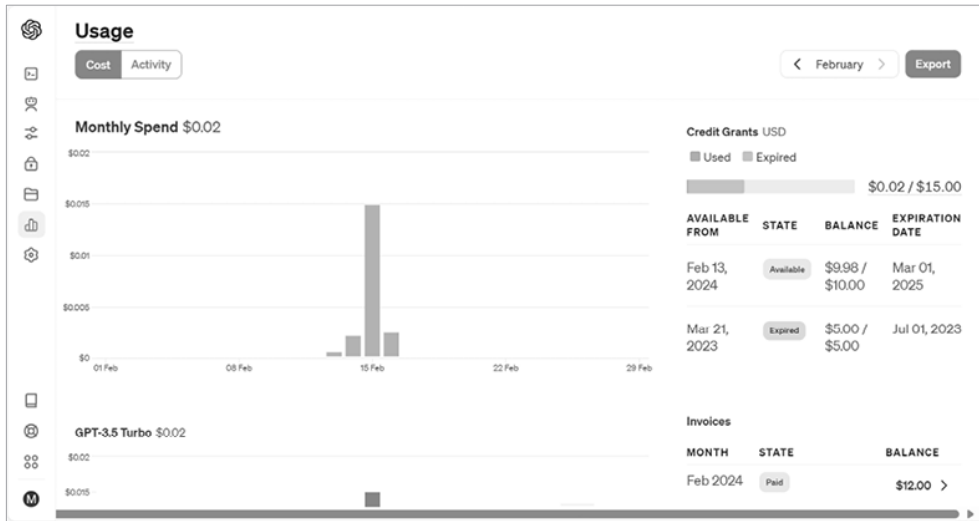
Jeżeli pojawi się błąd mówiący o braku środków, to odczekaj kilka minut, aby system zorientował się, że przed chwilą wpłacono jakąś kwotę. Jednym z powodów, dla których używamy teraz usługi Playground, jest sprawdzenie, czy wpłacone środki rzeczywiście zostały przyjęte przez systemy OpenAI.

10. W menu nawigacyjnym po lewej stronie wybierz pozycję *Usage* lub przejdź pod adres <https://platform.openai.com/usage>, gdzie można się dowiedzieć, że używane tutaj wywołania API powinny kosztować tylko kilka centów, tak jak na rysunku 9.5.
11. Zamknij przeglądarkę.

Teraz możemy już przystąpić do integrowania modelu LLM z naszym projektem .NET.

Używanie Semantic Kernel z modelem OpenAI

Eksperci AI tworzący wielkie modele językowe (LLM) zazwyczaj używają systemu operacyjnego Linux i języka Python. Samouczki i dokumentacja często zakładają, że deweloperzy również będą korzystać z tych platform. Użycie innego języka programowania lub



Rysunek 9.5. Wykresy obrazujące wykorzystanie API

frameworka, który nie został zaprojektowany dla platformy .NET, może wprowadzać dodatkową złożoność i zmniejszać łatwość utrzymania projektu. Poza tym rozwiązywanie problemów wynikających z różnic między językami lub frameworkami to dodatkowe kłopoty w pracy.

Czym jest Semantic Kernel?

Semantic Kernel został zaprojektowany z myślą o ekosystemie .NET, zapewniając integrację z językiem C# i innymi językami .NET. Stanowi to ułatwienie w pracy dla programistów .NET, ponieważ otrzymują oni biblioteki lub pakiety zgodne z idiomami języka C#.

Semantic Kernel może korzystać z asynchronicznego modelu programowania w .NET do efektywniejszej realizacji zadań zależnych od wejścia-wyjścia (I/O), co jest niezwykle ważne w przypadku integracji usług wymagających komunikacji sieciowej, takich jak API modeli LLM. Wynikiem są tutaj nieblokujące operacje wejścia-wyjścia, co jest istotne dla zachowania wysokiej responsywności aplikacji.

Korzystanie z Semantic Kernel umożliwia lepszą integrację modelu w projekcie .NET. Pozwala on na precyzyjne dostosowanie sposobu, w jaki model LLM wchodzi w interakcję z aplikacją. Można np. dostosować potok żądań, sposób przetwarzania odpowiedzi lub integrację z innymi usługami .NET. Ta elastyczność umożliwia również skuteczniejsze skalowanie aplikacji, dostosowując użycie modelu LLM w miarę rozwoju projektu.

Biorąc pod uwagę popularność platformy .NET, integracja realizowana za pomocą takiego rozwiązania jak Semantic Kernel może zapewnić lepsze wsparcie ze strony społeczności i udostępnić zasoby dostosowane do potrzeb programistów .NET. Do dyspozycji mamy rozbudowaną dokumentację, fora oraz narzędzia firm trzecich, które mogą być nieocenioną pomocą podczas rozwiązywania problemów i usprawniania aplikacji.

Teraz możemy już przystąpić do budowania projektu .NET korzystającego z API OpenAI z wykorzystaniem przygotowanego wcześniej klucza.

Przed wszystkim musimy bezpiecznie przechowywać konfigurację naszej aplikacji. Użyjemy do tego funkcji obsługi konfiguracji .NET oraz kombinacji pliku ustawień JSON (będzie przechowywał nazwy modeli itd.) i zmiennych środowiskowych (tu zapiszemy nasz klucz API).

No to do roboty:

1. Użyj swojego edytora kodu, aby do rozwiązania *Rozdział09* dodać nową aplikację konsoli o nazwie *ChatApp*.
2. Zmodyfikuj plik *ChatApp.csproj*, aby statycznie i globalnie zaimportować klasę `System.Console`, dodać odwołanie do pakietów umożliwiających współpracę z Semantic Kernel i obsługę konfiguracji przy użyciu plików JSON i zmiennych środowiskowych, włączyć traktowanie ostrzeżeń jako błędów oraz dołączyć plik *appsettings.json* z wdrożonym zestawem, zgodnie z wyróżnieniami w poniższym kodzie:

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

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

  <ItemGroup>
    <!--Pakiety tworzenia konfiguracji z pliku appsettings.json i zmiennych środowiskowych.-->
    <PackageReference Version="8.0.1"
      Include="Microsoft.Extensions.Configuration.Binder" />
    <PackageReference Version="8.0.0"
      Include="Microsoft.Extensions.Configuration.Json" />
    <PackageReference Version="8.0.0"
      Include="Microsoft.Extensions.Configuration.EnvironmentVariables" />

    <!--Pakiety umożliwiające pracę z Semantic Kernel.-->
    <PackageReference Include="Microsoft.SemanticKernel" Version="1.13.0" />
  </ItemGroup>

  <ItemGroup>
    <Content Include="appsettings.json">
      <CopyToOutputDirectory>Always</CopyToOutputDirectory>
    </Content>
  </ItemGroup>

  <ItemGroup>
    <Using Include="System.Console" Static="true" />
  </ItemGroup>

</Project>
```

Uwaga

Najnowsze wersje pakietów Semantic Kernel znajdziesz pod tym adresem: <https://www.nuget.org/packages/Microsoft.SemanticKernel>.

3. Skompiluj projekt *ChatApp*, aby pobrać niezbędne pakiety.
4. W folderze projektu *ChatApp* dodaj nowy plik JSON o nazwie *appsettings.json* i wpisz do niego poniższy fragment:

```
{
  "Settings": {
    "ModelId": "gpt-3.5-turbo-0125",
    "OpenAISecretKey": "<twój-tajny-klucz>"
  }
}
```

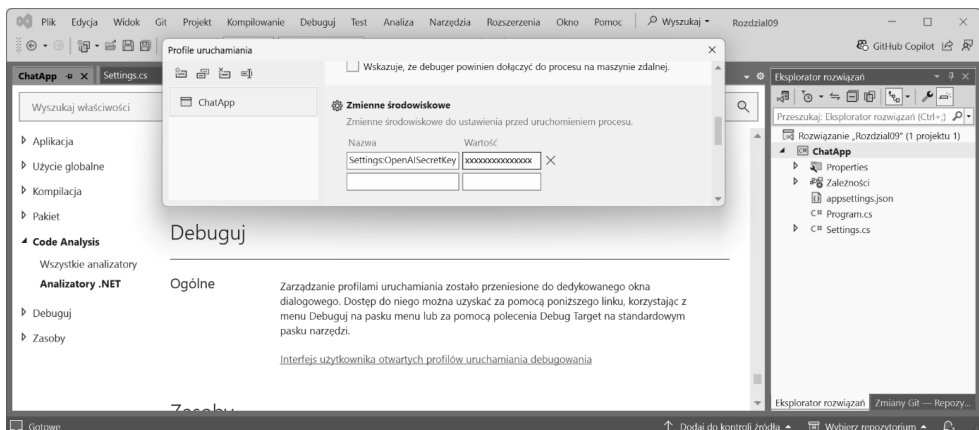
5. W folderze projektu *ChatApp* dodaj nowy plik klasy o nazwie *Settings.cs*, taki jak w poniższym kodzie:

```
public sealed class Settings
{
    public required string ModelId { get; set; }
    public required string OpenAISecretKey { get; set; }
}
```

Uwaga

Klasa *Settings* znajduje się w domyślnej przestrzeni nazw, podobnie jak automatycznie generowana klasa *Program*.

6. Jeżeli korzystasz z Visual Studio, wybierz z menu pozycję *Projekt/ChatApp właściwości*, a następnie w sekcji *Debuguj* kliknij łącze *Interfejs użytkownika otwartych profilów debugowania* i w okienku dialogowym wprowadź zmienną środowiskową, tak jak na rysunku 9.6.



Rysunek 9.6. Definiowanie zmiennej środowiskowej w profilu uruchamiania projektu

7. Zamknij okno ustawień uruchamiania i upewnij się, że utworzony został folder *Properties* zawierający plik o nazwie *launchSettings.json* z zawartością definiującą zmienną środowiskową, taką jak w poniższym kodzie:

```
{
  "profiles": {
    "ChatApp": {
      "commandName": "Project",
      "environmentVariables": {
        "Settings:OpenAISecretKey": "<twój-tajny-klucz>"
      }
    }
  }
}
```

Uwaga

Można też zdefiniować zmienną środowiskową w wierszu poleceń lub w terminalu. Jeżeli nie masz doświadczenia w tworzeniu zmiennych środowiskowych, skorzystaj z tego linku: <https://github.com/markjprice/cs12dotnet8/blob/main/docs/ch09-environment-variables.md>. Możesz także zdefiniować tajne dane użytkownika dla lokalnego środowiska deweloperskiego, a w środowisku produkcyjnym użyć usługi Azure Key Vault (lub podobnej).

8. Utwórz plik *.gitignore* w folderze projektu i dopisz do niego poniższą treść, aby upewnić się, że plik *launchSettings.json* nie będzie zapisywany w repozytorium Gita.

```
# Nie przechowuj poświadczeń w repozytorium.
launchSettings.json
**/Properties/launchSettings.json
```

Dobra praktyka

Takie pliki jak *launchSettings.json*, które mogą zawierać wrażliwe informacje, np. klucz dostępu, nigdy nie powinny być umieszczane w publicznym repozytorium Gita.

9. W folderze projektu dodaj nowy plik klasy o nazwie *Program.GetSettings.cs*.
10. W pliku *Program.GetSettings.cs* usuń istniejące instrukcje i dopisz kod definiujący metodę ładującą plik konfiguracyjny i zmienne środowiskowe, zapisując odczytane dane w obiekcie klasy *Settings*, a następnie zwracając ten obiekt, tak jak w poniższym kodzie:

```
// ConfigurationBuilder itd.
using Microsoft.Extensions.Configuration;

partial class Program
{
    private static Settings? GetSettings()
    {
```

```

const string settingsFile = "appsettings.json";
const string settingsSectionKey = nameof(Settings);

// Tworzenie obiektu konfiguracji na bazie pliku JSON
// i zmiennych środowiskowych.
 IConfigurationRoot config = new ConfigurationBuilder()
    .AddJsonFile(settingsFile)
    .AddEnvironmentVariables()
    .Build();

// Pobieranie ustawień z konfiguracji na podstawie klucza
// i silnie typowanej klasy.
 Settings? settings = config.GetRequiredSection(
    settingsSectionKey).Get<Settings>();

if (settings is null)
{
    WriteLine($"W pliku {settingsFile} nie znaleziono sekcji
        ↳{settingsSectionKey}.");
    return null;
}
else
{
    return settings;
}
}
}

```

11. W folderze projektu dodaj nowy plik klasy o nazwie *Program.GetKernel.cs*.

12. W pliku *Program.GetKernel.cs* usuń wszystkie istniejące instrukcje, wpisz instrukcje tworzące obiekt typu *KernelBuilder*, dodaj funkcję interakcji z chatem OpenAI przy użyciu skonfigurowanego modelu i tajnego klucza, a następnie zwróć utworzony obiekt, tak jak w poniższym kodzie:

```

using Microsoft.SemanticKernel; // Kernel.

partial class Program
{
    private static Kernel GetKernel(Settings settings)
    {
        IKernelBuilder kernelBuilder = Kernel.CreateBuilder();

        // Konfiguruj chat OpenAI, podając nazwę modelu i swój tajny klucz.
        kernelBuilder.AddOpenAIChatCompletion(
            settings.ModelId,
            settings.OpenAISecretKey);

        Kernel kernel = kernelBuilder.Build();

        return kernel;
    }
}

```

13. W pliku *Program.cs* usuń wszystkie istniejące instrukcje i wpisz instrukcje pobierające ustawienia, używające ich do pobrania jądra semantycznego, a następnie umożliwiające chatowanie w pętli, którą można zakończyć naciśnięciem klawisza *X*, tak jak w poniższym kodzie:

```
using Microsoft.SemanticKernel; // Kernel.

Settings? settings = GetSettings();
if (settings is null)
{
    WriteLine("Nie znaleziono ustawień lub ustawienia są nieprawidłowe.
    ↳ Aplikacja zostanie zamknięta.");
    return; // Zamknij aplikację.
}

Kernel kernel = GetKernel(settings);

ConsoleKey key = ConsoleKey.A;

while (key is not ConsoleKey.X)
{
    Write("Wpisz swoje pytanie: ");
    string question = ReadLine(!);
    WriteLine(await kernel.InvokePromptAsync(question));
    WriteLine();
    WriteLine("Naciśnij klawisz X, aby zakończyć, albo inny klawisz,
    ↳ aby zadać kolejne pytanie.");
    key = ReadKey(intercept: true).Key;
}
}
```

14. Uruchom projekt *ChatApp* bez debugowania.
15. Wpisz takie pytanie: „Jakie trzy najważniejsze umiejętności powinien mieć programista .NET?”. Przyjrzyj się odpowiedzi (oczywiście może być inna, ponieważ istnieje w tym pewien element losowości), którą pokazano w poniższym przykładzie:

```
Wpisz swoje pytanie: Jakie trzy najważniejsze umiejętności powinien mieć
programista .NET?
1. Znajomość języków programowania .NET (C# lub VB.NET). Biegłość w C#,
w tym nowoczesne funkcje jak LINQ, async/await oraz pisanie czystego
i czytelnego kodu.
2. Znajomość ekosystemu .NET i technologii pokrewnych. Zrozumienie .NET
Framework, .NET Core/.NET 6+, ASP.NET Core (MVC, Web API, Blazor) oraz
pracy z bazami danych (Entity Framework Core).
3. Umiejętność projektowania i stosowania wzorców architektonicznych.
Znajomość wzorców projektowych (np. Dependency Injection), API (REST,
GraphQL) i architektur, takich jak mikroserwisy czy DDD.

Naciśnij klawisz X, aby zakończyć, albo inny klawisz, aby zadać kolejne
pytanie.
```


16. Naciśnij dowolny klawisz, a następnie wprowadź kolejne pytanie: „Proszę, napisz krótką biografię Marka Price’a”. Przyjrzyj się odpowiedzi, którą pokazano na poniższym wydruku:

```
Mark Price to były amerykański koszykarz, który zyskał sławę jako jeden z najlepszych rozgrywających NBA w latach 80. i 90. Urodzony 15 lutego 1964 roku w Bartlesville, Oklahoma, był czterokrotnym uczestnikiem Meczu Gwiazd NBA. Price spędził większość swojej kariery w Cleveland Cavaliers, gdzie był znany z precyzyjnego rzutu, szczególnie z rzutów wolnych, osiągając jedne z najwyższych procentów skuteczności w historii ligi. Po zakończeniu kariery zawodniczej zajął się trenowaniem, zarówno na poziomie uniwersyteckim, jak i w NBA. Mark Price jest uważany za jednego z najlepszych strzelców i najinteligentniejszych graczy swoich czasów.
```

Uwaga

Najsłynniejszym Markiem Price'em w internecie jest koszykarz.

17. Naciśnij klawisz X, aby zakończyć.

Praktyczne aplikacje chatbotów AI zazwyczaj muszą pracować z danymi na temat naszej organizacji, a te zazwyczaj zapisane są w relacyjnej bazie danych lub innym magazynie danych.

Przyjrzyjmy się teraz funkcjom, które mogą dostarczyć naszemu chatbotowi uzupełniające informacje, aby mógł lepiej spełniać przypisane mu zadania.

Czym są funkcje?

W kontekście modeli OpenAI funkcje odnoszą się do pewnych cech lub operacji, które te modele mogą wykonywać na żądanie. Takie funkcje rozszerzają możliwości modeli językowych poza samo generowanie tekstu, umożliwiając im realizację bardziej ustrukturyzowanych zadań.

Funkcje w modelach OpenAI to z góry zdefiniowane operacje lub zadania, które model może wykonać na żądanie. Funkcje pozwalają modelowi wykonywać wyspecjalizowane działania na podstawie otrzymanych danych wejściowych, co zwiększa jego użyteczność i elastyczność w wybranych zastosowaniach.

Funkcje są bardzo zróżnicowane w zależności od zastosowania. Oto kilka typowych przykładów:

- **Manipulacje na tekstach.** Chodzi o takie działania jak streszczanie tekstu, tłumaczenie między językami czy wyodrębnianie określonych informacji z podanego tekstu.
- **Analiza danych.** Wykonywanie obliczeń, generowanie wykresów, analizowanie zbiorów danych czy wyszukiwanie wzorców w podanych danych.
- **Pomoc w programowaniu.** Funkcje pomagające w pisaniu, debugowaniu i rozumieniu kodu w różnych językach programowania.

- **Interakcja z interfejsami API.** Funkcje zaprojektowane do interakcji z zewnętrznymi systemami, bazami danych lub API w celu pobierania lub wysyłania informacji. Jest to typ funkcji, którego używamy w celu rozbudowania modelu OpenAI o niestandardowe informacje.

Funkcje zazwyczaj działają zgodnie z ustaloną strukturą procesu:

- **Wywołanie.** Funkcja jest wywoływana lub uruchamiana na podstawie określonych słów kluczowych lub instrukcji zawartych w danych wejściowych użytkownika. Może to być wywołanie jawne (bezpośrednie wywołanie funkcji) lub niejawne (model rozpoznaje zadanie, które odpowiada danej funkcji).
- **Przetwarzanie.** Po wywołaniu funkcja przetwarza dane wejściowe zgodnie z określoną logiką lub algorytmem. Może to obejmować analizowanie tekstu, wykonywanie obliczeń, zapytania do baz danych lub interakcje z API.
- **Generowanie wyniku.** Po przetworzeniu otrzymanych danych funkcja generuje wynik, który jest następnie zwracany do modelu, aby ten mógł go przetworzyć przed zaprezentowaniem użytkownikowi. Wynik ten może mieć formę odpowiedzi tekstowej, struktury danych lub mieć zupełnie inny format.

Do korzyści wynikających z używania funkcji można zaliczyć:

- Upraszczenie złożonych zadań przez generowanie szybkich i precyzyjnych odpowiedzi na konkretne zapytania.
- Umożliwienie modelowi wykonywania wyspecjalizowanych zadań, które wymagają czegoś więcej niż tylko ogólne rozumienie języka.
- Integrację z zewnętrznymi systemami, bazami danych i API, dzięki czemu model jest bardziej wszechstronnym narzędziem, przydatnym w szerokim zakresie zastosowań.

Funkcje stanowią potężne rozszerzenie możliwości modeli OpenAI, umożliwiając im realizację różnorodnych zadań i efektywną interakcję z zewnętrznymi systemami.

Teraz dodajmy do naszej aplikacji funkcję, aby dostarczyć chatbotowi więcej informacji o mnie i o bazie danych Northwind.

Dodawanie funkcji

Istnieją dwa podstawowe sposoby użycia funkcji w połączeniu z modelem OpenAI. Po pierwsze, można zdefiniować własną funkcję i bezpośrednio ją wywołać. Po drugie, można skonfigurować wiele funkcji i umożliwić interfejsowi API podjęcie decyzji, kiedy ich używać, co model robi, wykorzystując opisy tekstowe i zapytania użytkownika. Przeanalizujemy tutaj oba rozwiązania: najpierw pojedynczą funkcję do zadawania pytań o autora, a następnie dowolną funkcję zarejestrowaną we wtyczce.

Do roboty!

1. W pliku *ChatApp.csproj* dodaj odwołanie do projektu biblioteki klas kontekstu danych Northwind, którą przygotowaliśmy w rozdziale 1. Możesz do tego wykorzystać poniższy kod:

```
<ItemGroup>
  <ProjectReference
    Include="..\..\Rozdzial01\Northwind.DataContext\Northwind.DataContext.
    ↪csproj" />
</ItemGroup>
```

2. Skompiluj projekt *ChatApp*, aby skompilować też dołączany projekt i skopiować utworzony plik zestawu do lokalnego folderu *bin*. Możesz też użyć polecenia `dotnet build`.
3. Utwórz nowy plik klasy o nazwie *Program.ChatFunctions.cs*, usuń z niego istniejące instrukcje, a następnie zdefiniuj statyczne metody w częściowej klasie *Program*, które będą pozwalały na uzyskanie informacji o autorze tej książki oraz o produktach w wybranych kategoriach bazy danych *Northwind*. Możesz zastosować podany niżej kod:

```
using Microsoft.EntityFrameworkCore; // Include.
using Northwind.EntityModels; // NorthwindContext.
using System.Text.Json; // JsonSerializer.
```

```
partial class Program
{
    private static string GetAuthorBiography()
    {
        return ""
```

Mark J Price to były certyfikowany trener Microsoftu (MCT) i obecny specjalista do spraw programowania w C# oraz architekt rozwiązań dla Azure, z ponad 20-letnim doświadczeniem w edukacji i programowaniu. Od 1993 roku Mark zdał ponad 80 egzaminów programistycznych Microsoftu i specjalizuje się w przygotowywaniu innych do tych egzaminów. Jego uczniowie to zarówno profesjonaliści z ogromnym doświadczeniem, jak i 16-letni praktykanci bez żadnego przygotowania. Mark skutecznie prowadzi wszystkich, łącząc umiejętności edukacyjne z praktycznym doświadczeniem w konsultingu i tworzeniu systemów dla przedsiębiorstw na całym świecie. W latach 2001-2003 Mark pracował dla Microsoftu, pisząc oficjalne materiały szkoleniowe w Redmond, w USA. Zespół Marka stworzył pierwsze kursy szkoleniowe języka C# jeszcze w fazie jego wczesnej wersji alfa. Pracując dla Microsoftu, prowadził również zajęcia typu "train-the-trainer", aby przygotować innych trenerów MCT do pracy z C# i .NET. W latach 2016-2022 Mark tworzył i prowadził kursy szkoleniowe dla Digital Experience Platform firmy Optimizely, najlepszego systemu CMS budowanego w .NET, przeznaczonego dla platform marketingu cyfrowego i e-commerce. W 2010 roku Mark zdobył dyplom studiów podyplomowych w dziedzinie edukacji (PGCE). Uczył matematyki na poziomie GCSE w dwóch szkołach średnich w Londynie. Mark posiada tytuł licencjata z wyróżnieniem (BSC Hons) w dziedzinie informatyki, uzyskany na Uniwersytecie w Bristolu w Wielkiej Brytanii.

Mark J Price jest autorem następujących książek:

- C# 6 and .NET Core 1.0 - Modern Cross-Platform Development, 2016
- C# 7 and .NET Core - Modern Cross-Platform Development, 2017
- C# 7.1 and .NET Core 2.0 - Modern Cross-Platform Development, 2017
- C# 8.0 and .NET Core 3.0 - Modern Cross-Platform Development, 2019

```

- C# 9 and .NET 5 - Modern Cross-Platform Development, 2020
- C# 10 and .NET 6 - Modern Cross-Platform Development, 2021
- C# 11 and .NET 7 - Modern Cross-Platform Development Fundamentals, 2022
- Apps and Services with .NET 7, 2022
- C# 12 and .NET 8 - Modern Cross-Platform Development Fundamentals, 2023
- Apps and Services with .NET 8, 2023
""";
}
// Zapisanie tych informacji w pamięci podręcznej jest szybsze niż
// odtwarzanie ich za każdym razem od nowa.
private static JsonSerializerOptions jsonSerializerOptions =
    new() { WriteIndented = true };

private static string GetProductsInCategory(string categoryName)
{
    using NorthwindContext db = new();

    var products = db.Products
        .Include(p => p.Category)
        .Where(p => p.Category!.CategoryName == categoryName)
        .Select(p => new
        {
            p.ProductId,
            p.ProductName,
            p.Category!.CategoryName,
            p.UnitPrice,
            p.UnitsInStock,
            p.UnitsOnOrder,
            p.Discontinued
        })
        .ToArray();

    // Zamiana tablicy produktów na ciąg znaków JSON.
    string json = JsonSerializer.Serialize(
        products, jsonSerializerOptions);

    return json;
}
}

```

Uwaga

Wybrano taki podzbiór pól produktów, aby ograniczyć dane przesyłane do modelu i uniknąć złożoności związanej z tabelami pokrewnymi, a mimo to umożliwić odpowiadanie na pytania dotyczące produktów i kategorii. Pomiąłem tu powiązane tabele (takie jak zamówienia i dostawcy), aby zmniejszyć ilość serializowanych danych. Oznacza to, że model nie będzie umiał odpowiedzieć na takie pytania jak „Kto jest dostawcą produktu Chai?” albo „Ilu klientów zamawia produkt Chang?”.

4. W pliku *Program.GetKernel.cs* w metodzie `GetKernel` przed zwróceniem obiektu `kernel` dodaj instrukcje definiujące dwie funkcje na bazie naszych metod statycznych, tak jak w poniższym kodzie:

```
// Utwórz funkcję promptu w ramach wtyczki i dodaj ją do kernela.
kernel.ImportPluginFromFunctions(pluginName:"AuthorInformation", [
    kernel.CreateFunctionFromMethod(
        method: GetAuthorBiography,
        functionName: nameof(GetAuthorBiography),
        description: "Pobiera biografię autora.")
]);

kernel.ImportPluginFromFunctions("NorthwindProducts", [
    kernel.CreateFunctionFromMethod(
        method: GetProductsInCategory,
        functionName: nameof(GetProductsInCategory),
        description: "Pobiera z bazy danych Northwind produkty danej
↳kategorii.")
]);
```

5. W pliku *Program.cs* po uzyskaniu obiektu `kernel` zdefiniuj funkcję na podstawie naszej metody, następnie umieść w komentarzu wywołanie metody `InvokePromptAsync` i zapisz otrzymane od użytkownika pytanie jako argument wywołania metody `InvokeAsync`, tak jak w poniższym kodzie:

```
Kernel kernel = GetKernel(settings);

// Zmienna $question zostanie zdefiniowana jako argument.
KernelFunction function = kernel.CreateFunctionFromPrompt("""
    Biografia autora: {{ authorInformation.getAuthorBiography }}.
    {{ $question }}
    """);

KernelArguments arguments = new();

ConsoleKey key = ConsoleKey.A;

while (key is not ConsoleKey.X)
{
    Write("Wpisz swoje pytanie: ");
    string question = ReadLine(!);

    // WriteLine(await kernel.InvokePromptAsync(question));
    arguments["question"] = question;
    // Wywołaj pojedynczą funkcję.
    WriteLine(await function.InvokeAsync(kernel, arguments));

    WriteLine();
    WriteLine("Naciśnij klawisz X, aby zakończyć, albo inny klawisz,
↳żeby zadać kolejne pytanie.");
    key = ReadKey(intercept: true).Key;
}
```

- Uruchom projekt *ChatApp* bez debugowania.
- Wprowadź taką instrukcję: Napisz, proszę, krótką biografię Marka Price'a i przyjrzyj się otrzymanej odpowiedzi, która powinna wyglądać podobnie do poniższej:

Mark J. Price - ekspert w dziedzinie programowania z ponad 20-letnim doświadczeniem, był Microsoft Certified Trainer (MCT) i obecny specjalista Microsoft w zakresie programowania w C# oraz projektowania rozwiązań w Azure.

Mark zdobył tytuł licencjata z wyróżnieniem z informatyki na Uniwersytecie w Bristolu. W latach 2001-2003 pracował w Redmond, gdzie pisał oficjalne materiały szkoleniowe dla Microsoftu, w tym pierwsze kursy dotyczące języka C#. W swojej karierze zdał ponad 80 egzaminów Microsoft i pomógł setkom osób - od początkujących po doświadczonych specjalistów - osiągnąć podobny sukces.

Mark tworzył także kursy dla platformy Optimizely, a wcześniej nauczał matematyki w londyńskich szkołach średnich, po zdobyciu dyplomu PGCE. Łączy pasję do edukacji z praktycznym doświadczeniem w projektowaniu systemów dla firm na całym świecie.

- Wprowadź taką instrukcję: Napisz biografię Marka Price'a nie dłuższą niż jeden akapit i przyjrzyj się otrzymanej odpowiedzi, która powinna wyglądać podobnie do poniższej:

Mark J. Price to doświadczony programista, edukator i były Microsoft Certified Trainer (MCT) z ponad 20-letnim doświadczeniem. Specjalizuje się w programowaniu w C# oraz projektowaniu rozwiązań w Microsoft Azure. Współtworzył pierwsze kursy C# w Microsoft, pracując w Redmond, a także prowadził szkolenia dla trenerów. Jest autorem materiałów edukacyjnych i wykładowcą, który pomaga zarówno początkującym, jak i ekspertom zdobywać certyfikaty Microsoft. Absolwent Uniwersytetu w Bristolu, łączy wiedzę akademicką z praktyką, tworząc rozwiązania i kursy dla firm oraz ucząc matematyki w szkołach średnich w Londynie.

- Wprowadź taką instrukcję: Kiedy autor napisał książkę o C# 10? i przyjrzyj się otrzymanej odpowiedzi, która powinna wyglądać podobnie do poniższej:

Książka "C# 10 and .NET 6 - Modern Cross-Platform Development" autorstwa Marka J. Price'a została wydana w listopadzie 2021 roku. Jeśli potrzebujesz dokładnej daty lub szczegółów, mogę to sprawdzić.

- Wprowadź taką instrukcję: Ile książek napisał Mark? i przyjrzyj się otrzymanej odpowiedzi, która powinna wyglądać podobnie do poniższej:

Mark J Price napisał łącznie dziesięć książek.

- Wprowadź taką instrukcję: Gdzie autor nauczył się języka C#? i przyjrzyj się otrzymanej odpowiedzi, która powinna wyglądać podobnie do poniższej:

Mark J. Price nauczył się języka C# w jego początkowej fazie rozwoju, gdy pracował w siedzibie Microsoft w Redmond, USA, w latach 2001-2003. W tym czasie był członkiem zespołu odpowiedzialnego za tworzenie pierwszych oficjalnych kursów szkoleniowych dla języka C#, jeszcze zanim został on oficjalnie wydany. Dzięki temu miał bezpośredni dostęp

do zespołów projektowych Microsoftu, co pozwoliło mu zdobyć dogłębną wiedzę o języku i platformie .NET od samego początku ich istnienia.

12. Naciśnij klawisz X, aby zakończyć aplikację.

Spróbujmy teraz włączyć automatyczne wykorzystywanie wielu funkcji i dodajmy do kernela pamięć konwersacji, aby umożliwić naszemu chatbotowi korzystanie z poprzednich pytań i odpowiedzi w trakcie prowadzonej konwersacji.

Dodanie pamięci sesji i włączanie wielu funkcji

Obecnie nasz chatbot nie pamięta wcześniejszych pytań, co oznacza, że nie możemy prowadzić z nim interaktywnej rozmowy. Wyobraź sobie następującą interakcję:

```
Wpisz swoje pytanie: Opowiedz dowcip o programistach.
Jak programista zmienia żarówkę? Wcale, to problem sprzętowy!

Wprowadź swoje pytanie: Dlaczego to jest zabawne?
Umiejętność Marka dostrzegania zabawnych stron rzeczy dodaje uroku jego
profilowi.
```

Przy braku kontekstu następujące po sobie pytania mogą powodować generowanie dziwnych odpowiedzi przez model LLM. Zobaczmy, jak można zapewnić chatbotowi pamięć kontekstu:

1. Na początku pliku *Program.cs* zaimportuj przestrzeń nazw umożliwiającą pracę z historią chatu, tak jak w poniższym kodzie:
2. W pliku *Program.cs* przed pętlą `while` dodaj instrukcje zapamiętywania poprzednich wiadomości z chatu i tak skonfiguruj model OpenAI, aby według potrzeby automatycznie wywoływał dowolne zarejestrowane funkcje. Następnie w pętli `while` umieść w komentarzu instrukcję wywołującą pojedynczą funkcję i dodaj instrukcje zapisujące wiadomości użytkownika i odpowiedzi modelu, zgodnie z wyróżnieniami w poniższym kodzie:

```
IChatCompletionService completion =
    kernel.GetRequiredService<IChatCompletionService>();

ChatHistory history = new(systemMessage: "Jesteś asystentem AI z wiedzą,
↳umiejętnościami i doświadczeniem Marka J. Price'a.");

OpenAIPromptExecutionSettings options = new()
    { ToolCallBehavior = ToolCallBehavior.AutoInvokeKernelFunctions };

while (key is not ConsoleKey.X)
{
    Write("Wpisz swoje pytanie: ");
    string question = ReadLine(!);

    // WriteLine(await kernel.InvokePromptAsync(question));
```

```

//arguments["question"] = question;
// Wywołaj jedną funkcję.
// WriteLine(await function.InvokeAsync(kernel, arguments));

history.AddUserMessage(question);
ChatMessageContent answer = await
    completion.GetChatMessageContentAsync(history);
history.AddAssistantMessage(answer.Content!);
WriteLine(answer.Content);

WriteLine();
WriteLine("Naciśnij klawisz X, aby zakończyć, albo inny klawisz,
↳ żeby zadać kolejne pytanie.");
key = ReadKey(intercept: true).Key;
}

```

3. Uruchom projekt *ChatApp* bez debugowania.
4. Wprowadź taką instrukcję: Opowiedz dowcip o programistach. Następnie przyjrzyj się odpowiedzi, która powinna być podobna do poniższej:

```
Dlaczego programiści mylą Halloween z Bożym Narodzeniem?
Bo 31 OCT == 25 DEC!
```

5. Wprowadź teraz pytanie: Dlaczego to jest zabawne? Ponownie przyjrzyj się odpowiedzi, podobnej do poniższej:

```
W systemie szesnastkowym (hexadecimal), liczba 31 (hex) w systemie
dziesiętnym to 49. Ale w systemie ósemkowym (octal), 31 (octal) odpowiada
liczbie 25 w dziesiętnym. Więc w pewnym sensie 31 OCT równa się 25 DEC,
jeśli patrzysz z perspektywy systemów liczbowych.
Dowcip jest grą na słowach i logice, którą docenią osoby techniczne,
znające systemy numeryczne.
```

6. Wprowadź kolejne pytanie: Jakie owoce morza są najdroższe? Przyjrzyj się odpowiedzi, takiej jak pokazana na poniższym wydruku:

```
Aby znaleźć najdroższy owoc morza, mogę pobrać informacje o produktach
z kategorii owoców morza dostępnych w bazie danych, a następnie
zidentyfikować produkt z najwyższą ceną. Pozwól, że zdobędę dla Ciebie
potrzebne dane. Najdroższym produktem z owoców morza są „Carnarvon Tigers”
z ceną jednostkową 62,5 $. Jest to najwyżej wyceniony produkt w bazie
danych. Jeśli masz więcej pytań lub potrzebujesz dalszych informacji,
daj znać!
```

7. Wprowadź następujące pytanie: Jakie produkty znajdują się w kategorii owoców morza? Ponownie przyjrzyj się odpowiedzi, która powinna wyglądać tak jak poniższa:

```
Oto produkty w kategorii owoce morza:
1. Ikura
2. Konbu
3. Carnarvon Tigers
4. Nord-Ost Matjeshering
5. Inlagd Sill
6. Gravad lax
```



```

7. Boston Crab Meat
8. Jack's New England Clam Chowder
9. Rogede sild
10. Spegesild
11. Escargots de Bourgogne
12. Röd Kaviar

```

8. Wprowadź jeszcze jedno pytanie: Proszę, podaj też ceny. Tym razem odpowiedź powinna zostać uzupełniona, tak jak na poniższym wydruku:

```

Oto produkty w kategorii owoce morza wraz z ich cenami:
1. Ikura - $31
2. Konbu - $6
3. Carnarvon Tigers - $62.5
4. Nord-Ost Matjeshering - $25.89
5. Inlagd Sill - $19
6. Gravad lax - $26
7. Boston Crab Meat - $18.4
8. Jack's New England Clam Chowder - $9.65
9. Rogede sild - $9.5
10. Spegesild - $12
11. Escargots de Bourgogne - $13.25
12. Röd Kaviar - $15

```

9. Naciśnij klawisz X, aby zakończyć aplikację.

Czasami trzeba chwilę poczekać na otrzymanie odpowiedzi z usługi OpenAI. Możemy jednak włączyć asynchroniczne przesyłanie wyników, by zapewnić użytkownikowi lepsze doświadczenia.

Strumieniowanie wyników

Zaimplementujmy teraz strumieniowanie wyników:

1. W pliku *Program.cs* przed pętlą `while` dodaj instrukcję definiującą obiekt `StringBuilder`, a następnie w pętli `while` dodaj instrukcje implementujące asynchroniczne strumieniowanie wyników, zgodnie z wyróżnieniami w poniższym kodzie:

```

// Podstawa implementacji asynchronicznego strumieniowania odpowiedzi.
StringBuilder builder = new();

while (key is not ConsoleKey.X)
{
    Write("Wpisz swoje pytanie: ");
    string question = ReadLine(!);

    // WriteLine(await kernel.InvokePromptAsync(question));
    // arguments["question"] = question;
    // Wywołaj jedną funkcję.
    // WriteLine(await function.InvokeAsync(kernel, arguments));

    history.AddUserMessage(question);
}

```

```

// ChatMessageContent answer = await
// completion.GetChatMessageContentAsync(history);

builder.Clear();
await foreach (StreamingChatMessageContent message
in completion.GetStreamingChatMessageContentsAsync(history))
{
    Write(message.Content);
    builder.Append(message.Content);
}
history.AddAssistantMessage(builder.ToString());

WriteLine();
WriteLine("Naciśnij klawisz X, aby zakończyć, albo inny klawisz, żeby
zadać kolejne pytanie.");
key = ReadKey(intercept: true).Key;
}

```

2. Uruchom bez debugowania projekt *ChatApp*.
3. Zwróć uwagę na to, że odpowiedzi z usługi OpenAI są teraz strumieniowane asynchronicznie.

Dodawanie protokołowania i zwiększanie niezawodności

Teraz poprawmy chatbota, dodając możliwość protokołowania i zwiększając jego niezawodność:

1. W pliku projektu *ChatApp.csproj* dodaj odwołania do pakietów pozwalających na włączenie protokołowania, tak jak w poniższym kodzie:

```

<!-- Dodaj protokołowanie w konsoli i odporność HTTP. -->
<PackageReference Version="8.0.0" Include="Microsoft.Extensions.Http" />
<PackageReference Version="8.2.0"
Include="Microsoft.Extensions.Http.Resilience" />
<PackageReference Version="8.0.0"
Include="Microsoft.Extensions.Logging.Console" />

```

2. Na początku pliku *Program.GetKernel.cs* zaimportuj przestrzenie nazw związane z protokołowaniem, tak jak w poniższym kodzie:

```

using Microsoft.Extensions.DependencyInjection; // AddLogging.
using Microsoft.Extensions.Logging; // LogLevel.

```

3. W pliku *Program.GetKernel.cs* przed instrukcją tworzącą obiekt kernel dopisz instrukcje włączające protokołowanie i funkcje odporności, tak jak w poniższym kodzie:

```

// Dodaj protokołowanie i odporność.
kernelBuilder.Services.AddLogging(c =>
    c.AddConsole().SetMinimumLevel(LogLevel.Trace));

```

```
kernelBuilder.Services.ConfigureHttpClientDefaults(c =>
    c.AddStandardResilienceHandler());
```

```
Kernel kernel = kernelBuilder.Build();
```

4. Uruchom bez debugowania projekt *ChatApp* i zauważ, że od razu wyświetlane są dodatkowe informacje, tak jak na poniższym wydruku:

```
trce: Program[0]
    Created KernelFunction 'GetAuthorBiography' for 'GetAuthorBiography'
trce: Microsoft.SemanticKernel.KernelPromptTemplate[0]
    Extracting blocks from template: Author biography:
    ↳ {{ authorInformation.getAuthorBiography }}.
    {{ $question }}
trce: Program[0]
    Created KernelFunction 'GetProductsInCategory'
    ↳ for 'GetProductsInCategory'
Wpisz swoje pytanie:
```

5. Wprowadź następujące zapytanie: Proszę, wygeneruj kod źródłowy HTML dla tabeli produktów z kategorii condiments. Uwzględnij przy tym cenę i liczbę sztuk w magazynie.
6. Przyjrzyj się odpowiedzi, która powinna być podobna do tej z poniższego wydruku:

```
Pobrałem produkty w kategorii Przyprawy wraz z ich cenami i informacjami
o stanie magazynowym. Pozwól, że wygeneruję kod źródłowy HTML dla tabeli
wyświetlającej te informacje.
Oto kod źródłowy HTML dla tabeli wyświetlającej produkty w kategorii
Przyprawy wraz z ich cenami i liczbą sztuk w magazynie:
```

```
```html
<table>
 <thead>
 <tr>
 <th>Nazwa produktu</th>
 <th>Cena jednostkowa</th>
 <th>Sztuki w magazynie</th>
 </tr>
 </thead>
 <tbody>
 <tr>
 <td>Syrop anyżowy</td>
 <td>$10.00</td>
 <td>13</td>
 </tr>
 ...
 <tr>
 <td>Oryginalna Frankfurter grüne Soße</td>
 <td>$13.00</td>
 <td>32</td>
 </tr>
 </tbody>
```

```
</table>

```

Ta tabela zawiera nazwę produktu, cenę jednostkową oraz liczbę sztuk w magazynie dla każdego produktu w kategorii Condiments. Możesz dowolnie modyfikować kod, jeśli zajdzie taka potrzeba.

### 7. Naciśnij klawisz X, aby zakończyć aplikację.

Teraz zobaczymy, jak użyć lokalnego modelu LLM zamiast modeli działających w chmurze.

## Korzystanie z lokalnych modeli LLM

W tym podrozdziale sprawdzimy, jak pobierać i uruchamiać wielkie modele językowe (LLM) na lokalnym komputerze.

### Hugging Face

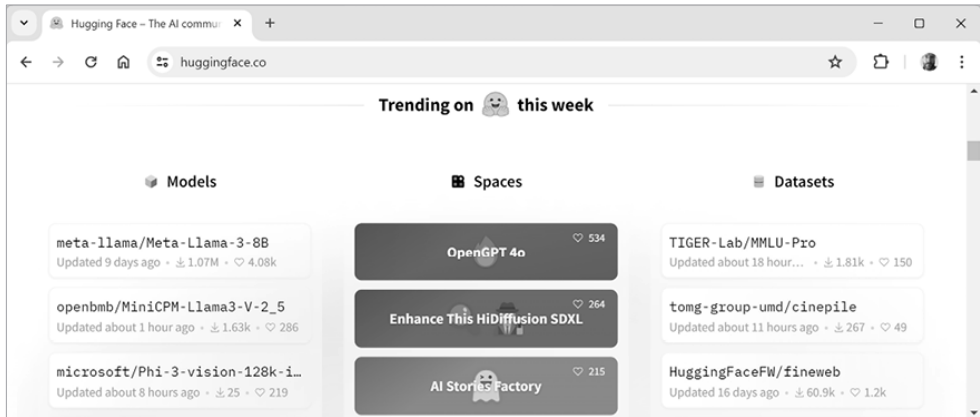
Hugging Face to popularna platforma open source, która umożliwia tworzenie i udostępnianie nowoczesnych modeli w dziedzinie przetwarzania języka naturalnego (NLP). Jest znana z opracowywania narzędzi, bibliotek i zasobów, które demokratyzują dostęp do zaawansowanych modeli uczenia maszynowego. Projekt Hugging Face w kilku kluczowych obszarach przyczynia się do rozwoju społeczności LLM:

- **Biblioteka Transformers.** Oferuje szeroki zakres wstępnie wytrenowanych modeli do różnych zadań NLP, takich jak generowanie tekstu, tłumaczenie, streszczanie i odpowiadanie na pytania.
- **Biblioteka Datasets.** Zapewnia ogromny zbiór gotowych do użycia zestawów danych do projektów uczenia maszynowego, obejmujących szeroki zakres zadań i dziedzin.
- **Model Hub.** To platforma, na której użytkownicy mogą przesyłać i udostępniać własne modele społeczności, przeglądać i pobierać modele udostępnione przez innych oraz korzystać z narzędzi do oceny i testowania wydajności modeli.
- **Biblioteka Tokenizers.** Udostępnia wydajne i elastyczne metody tokenizacji, które są wykorzystywane do przygotowywania danych tekstowych dla modeli bazujących na transformatorach.

#### Uwaga

Jedną z nowych funkcji, które zostaną wprowadzone w .NET 9 w listopadzie 2024 r., jest standardowy typ przeznaczony do pracy w bibliotekach tokenizacji. Więcej informacji na ten temat znajdziesz pod tym adresem: <https://github.com/dotnet/core/blob/main/release-notes/9.0/preview/preview4/libraries.md#tokenizer-library-enhancements>.

Możesz sprawdzić, co jest obecnie popularne na stronie głównej Hugging Face, używając adresu <https://huggingface.co/>. Widać tam, że model Llama 3 firmy Meta z 8 mld parametrów został pobrany ponad milion razy, co widać też na rysunku 9.7.



Rysunek 9.7. Strona główna Hugging Face

### Uwaga

Model Llama 3 firmy Meta jest popularny z kilku powodów. Rozmiar 8 mld parametrów stanowi zrównoważoną wielkość modelu, co pozwala mu uchwycić subtelności w danych, na których został wytrenowany. Meta prawdopodobnie zastosowała zaawansowane metody szkoleniowe i zróżnicowane zestawy danych, aby zapewnić dobrą wydajność modelu w szerokim zakresie zadań. Dotychczasowa popularność powoduje dalsze zwiększanie popularności, ponieważ zachęca to do tworzenia silnego systemu wsparcia, w tym dokumentacji, forów i pomocy społeczności.

Teraz przyjrzymy się narzędziom, które mogą maksymalnie ułatwić pobieranie i używanie modeli Hugging Face na lokalnym komputerze.

## Ollama

**Ollama** to program narzędziowy, którego zadaniem jest zarządzanie modelami LLM na urządzeniach lokalnych. Pozwala użytkownikom uruchamiać zaawansowane modele językowe, takie jak **LLaMA** (ang. *Large Language Model Meta AI*), bez konieczności polegania na rozwiązaniach działających w chmurze.

Dzięki lokalnemu uruchamianiu modeli Ollama sprawia, że wrażliwe dane pozostają na urządzeniu użytkownika, co zwiększa prywatność i bezpieczeństwo w porównaniu z usługami chmurowymi. Użytkownicy zachowują pełną własność i kontrolę nad swoimi danymi, co jest szczególnie ważne w przypadku informacji wrażliwych lub chronionych.

Deweloperzy i badacze mogą korzystać z Ollama do eksperymentowania i dostosowywania modeli LLM bez konieczności odwoływania się do zasobów chmurowych. Na dłuższą

metę uruchamianie modeli lokalnie może być bardziej opłacalne, ponieważ eliminuje potrzebę ciągłego subskrybowania usług. Z drugiej strony, instalacje lokalne potrzebują ciągłej konserwacji i aktualizacji, co wymaga dodatkowych nakładów w porównaniu do zarządzanych usług chmurowych.

Aby uruchomić LLM lokalnie, użytkownicy potrzebują wydajnego sprzętu, zwykle obejmującego zaawansowane procesory (CPU), znaczną ilość pamięci RAM oraz **karty graficzne** (GPU) do obsługi intensywnych obliczeń. Lokalne uruchamianie modeli zmniejsza jednak opóźnienia w generowaniu odpowiedzi, co jest istotne w aplikacjach wymagających natychmiastowych reakcji. Modele LLM są bardzo zasobożerne, a to oznacza, że nie każdy będzie dysponował sprzętem odpowiednim do ich efektywnego uruchamiania.

Projekt Ollama ma na celu uproszczenie procesu instalacji i konfiguracji modelu, udostępniając narzędzia i dokumentację, które pomagają użytkownikom szybko rozpocząć pracę.

Spróbujmy teraz pobrać i zainstalować pakiet Ollama:

1. Otwórz swoją przeglądarkę i przejdź na stronę <https://ollama.com/>.
2. Na stronie głównej kliknij przycisk *Download*.
3. Wybierz swój system operacyjny i postępuj zgodnie z instrukcjami instalacji.

## Modele Ollama

Modele zwykle występują w rozmaitych wariantach, różniących się od siebie rozmiarem i możliwościami. Na przykład:

- `llama` i `llama:70b` różnią się rozmiarem.

### Uwaga

Mniejsze modele, takie jak `llama`, mogą być wystarczające do wykonywania mniej złożonych zadań, jak podstawowe generowanie tekstu, proste odpowiadanie na pytania i podsumowywanie. W przypadku aplikacji wymagających odpowiedzi w czasie rzeczywistym, takich jak chatboty, mniejsze modele są zazwyczaj lepsze ze względu na mniejsze opóźnienia w procesie wnioskowania. Bardziej złożone zadania, takie jak dogłębne rozumienie, szczegółowe rozumowanie i generowanie dłuższych, spójnych tekstów, mogą wymagać użycia większych modeli, takich jak `llama:70b`. Jednak większe modele wymagają znacznie większej mocy obliczeniowej i pamięci zarówno podczas treningu, jak i przy wykonywaniu zadań. Warto również pamiętać, że w wybranych dziedzinach mniejszy, ale dobrze dostrojony model może przewyższyć większy, ogólny model.

- `mistral:instruct` — model wykonujący przekazane mu polecenia.
- `mistral:text` — model bazowy bez wykonanego dostrajania, najlepiej nadający się do prostego uzupełniania tekstu.

Aby szybko pobrać i uruchomić model Ollama w trybie interaktywnym, użyj następującego polecenia:

```
ollama run <model>
```

Najczęściej używane modele obsługiwane przez Ollama zostały zebrane w tabeli 9.1.

**Tabela 9.1. Popularne modele Ollama**

Model	Liczba parametrów	Wielkość	Opis
llama:70b	70 mld	40 GB	Najbardziej zaawansowany otwarty LLM dostępny do tej pory. Modele Llama 3 zostały dostrojone do prowadzenia dialogu i w tym zastosowaniu sprawdzają się lepiej niż jakiegokolwiek inne otwarte modele
llama3	8 mld	4,7 GB	Mniejsza wersja modelu z rodziny Llama 3
code11ama	7 mld	3,8 GB	Zajmuje się generowaniem i analizowaniem kodu na podstawie podpowiedzi tekstowych
llama2-uncensored	7 mld	3,8 GB	Nieocenzurowany model Llama 2, przygotowany przez George'a Sunga i Jarrada Hope'a. Wykorzystali oni proces zdefiniowany przez Erica Hartforda w tym artykule: <a href="https://erichartford.com/uncensored-models">https://erichartford.com/uncensored-models</a>
phi3	3,8 mld	2,3 GB	Phi-3 to rodzina lekkich, małych (3 mld parametrów) i średnich (14 mld parametrów) modeli przygotowana przez Microsoft. Na konferencji Microsoft Build 2024 zaprezentowano wersję phi3-silica, która została zoptymalizowana do pracy z procesorami NPU (Neural Processing Unit) o wydajności większej niż 40 TOPS
mistral	7 mld	4,1 GB	Model o otwartej licencji Apache, dostępny w wersji wykonującej podane instrukcje i w wersji przygotowanej do uzupełniania tekstu
gemma:7b	7 mld	4,8 GB	Gemma to rodzina lekkich, nowoczesnych modeli przygotowanych przez Google DeepMind
gemma:2b	2 mld	1,4 GB	Najmniejszy model z rodziny Gemma odpowiedni dla urządzeń mobilnych

### Uwaga

„Nieocenzurowany” w kontekście LLM oznacza, że model działa bez z góry narzuconych ograniczeń dotyczących generowanych treści, co pozwala na szerszy zakres wyników, ale zwiększa również ryzyko generowania szkodliwych lub nieodpowiednich treści. Z tego powodu modele nieocenzurowane mają większą wartość dla zadań związanych z badaniami i rozwojem, mogą pomagać komikom przygotowywać nowe dowcipy, ale też wymagają dokładniejszej kontroli, ze względu na problemy etyczne, jakie mogą się pojawiać w praktycznych zastosowaniach.

W naszym przypadku skorzystamy z modelu Llama 3, który jest wystarczająco wszechstronny i ma całkiem dużą szansę poprawnego działania na Twoim komputerze. Szczegóły licencji tego modelu znajdziesz pod adresem <https://llama.meta.com/llama3/license/>.

## Interfejs wiersza poleceń Ollama

Interfejs wiersza poleceń Ollama udostępnia wiele różnych poleceń do lokalnego uruchamiania modeli LLM i zarządzania nimi. Co prawda poszczególne polecenia i opcje mogą się różnić w zależności od wersji i implementacji Ollama, ale poniżej przedstawiam ogólny przegląd najczęściej używanych poleceń.

Przyjrzyjmy się teraz możliwościom poleceń Ollama w połączeniu z modelem Llama 3:

1. W wierszu poleceń lub terminalu wpisz poniższe polecenie, aby sprawdzić wersję narzędzia:

```
ollama --version
```

2. Przyjrzyj się odpowiedzi, która powinna wyglądać mniej więcej tak:

```
ollama version is 0.1.38
```

3. Aby pobrać określony model, taki jak Llama 3, wpisz polecenie:

```
ollama pull llama3
```

4. Przyjrzyj się odpowiedzi, która powinna wyglądać tak:

```
pulling manifest
pulling 6a0746a1ec1a... 100%
██████████ 4.7 GB
pulling 4fa551d4f938... 100%
██████████ 12 KB
pulling 8ab4849b038c... 100%
██████████ 254 B
pulling 577073ffcc6c... 100%
██████████ 110 B
pulling 3f8eb4da87fa... 100%
██████████ 485 B
verifying sha256 digest
writing manifest
removing any unused layers
success
```

### Uwaga

Możesz też usunąć ten model za pomocą polecenia `ollama rm llama3`.

5. Aby wyświetlić listę dostępnych lokalnych modeli, wpisz polecenie:

```
ollama list
```

6. Przyjrzyj się odpowiedzi, która powinna wyglądać tak:

NAME	ID	SIZE	MODIFIED
llama3:latest	365c0bd3c000	4.7 GB	21 minutes ago



7. Aby uruchomić określony model (lub pobrać go, jeśli jeszcze nie został pobrany), wpisz poniższe polecenie w wierszu poleceń lub w terminalu:  

```
ollama run llama3
```
8. W efekcie powinien się pojawić nowy wiersz poleceń, taki jak poniżej:  

```
>>> Send a message (? for help)
```
9. Wprowadź swoje pytanie, np. *Czym jest .NET?*, i przyjrzyj się odpowiedzi.
10. Aby zakończyć, wpisz polecenie */bye*.

### Uwaga

Wiersz poleceń Ollama oferuje również polecenia do kopiowania i tworzenia nowych modeli. Szczegółową dokumentację znajdziesz pod adresem <https://github.com/ollama/ollama#cli-reference>.

Projekt Ollama udostępnia biblioteki klienckie wyłącznie dla Pythona i JavaScriptu, ale programiści .NET przygotowali już dedykowane pakiety, którym przyjrzymy się w kolejnym punkcie.

## Pakiet OllamaSharp dla .NET

OllamaSharp to projekt udostępniania API Ollama na platformie .NET, umożliwiający łatwą interakcję aplikacji .NET z modelami Ollama. Obsługuje wszystkie punkty końcowe API Ollama, w tym dostęp do chatów, osadzania, wyświetlania dostępnych modeli, pobierania i tworzenia nowych modeli.

### Więcej informacji

Dodatkowe informacje znajdziesz w repozytorium OllamaSharp na GitHubie: <https://github.com/awaescher/OllamaSharp>.

Sprawdźmy, jak to działa:

1. W swoim edytorze kodu utwórz nowy projekt aplikacji konsoli o nazwie *OllamaApp* i dodaj go do rozwiązania *Rozdzial09*.
2. W pliku projektu *OllamaApp.csproj* dodaj odwołania do pakietów *Spectre.Console* i *Ollama*, a także statycznie i globalnie zaimportuj klasę *System.Console*, zgodnie z poniższym kodem:

```
<ItemGroup>
 <PackageReference Include="Spectre.Console" Version="0.49.1" />
 <PackageReference Include="OllamaSharp" Version="1.1.9" />
</ItemGroup>

<ItemGroup>
 <Using Include="System.Console" Static="true" />
</ItemGroup>
```

**Uwaga**

Numery najnowszych wersji pakietów możesz sprawdzić pod adresami <https://www.nuget.org/packages/Spectre.Console> i <https://www.nuget.org/packages/OllamaSharp>.

3. W pliku *Program.cs* usuń istniejące instrukcje i dopisz kod:

- Importujący przestrzenie nazw potrzebne do pracy z Ollama i Spectre Console.
- Tworzący klienta Ollama wysyłającego żądania do lokalnego hosta na domyślnym porcie 11434.
- Wyświetlający tabelę Spectre Console z modelami dostępnymi w lokalnej instalacji Ollama.
- Nakazujący klientowi Ollama używanie najnowszego modelu Llama 3.
- Pobierający zapytanie od użytkownika, wysyłający je do klienta Ollama i wyświetlający odpowiedź.
- Mierzący czas pracy modelu za pomocą stopera.

```
using OllamaSharp; // OllamaApiClient.
using OllamaSharp.Models; // Model.
using Spectre.Console; // Table.
using System.Diagnostics; // Stopwatch.

// Domyślny port API Ollama to 11434.
string port = "11434";
Uri uri = new($"http://localhost:{port}");

OllamaApiClient ollama = new(uri);

Table table = new();
table.AddColumn("Nazwa");
table.AddColumn("Wielkość");

// Pobierz listę modeli.
IEnumerable<Model> models = await ollama.ListLocalModels();

foreach (Model model in models)
{
 table.AddRow(model.Name, model.Size.ToString("N0"));
}

AnsiConsole.Write(table);

string modelName = "llama3:latest";

WriteLine();
WriteLine($"Wybrany model: {modelName}");
ollama.SelectedModel = modelName;
```

```

Write("Wpisz swój prompt: ");
string? prompt = ReadLine();
if (string.IsNullOrEmpty(prompt))
{
 WriteLine("Musisz podać swój prompt. Zamykam aplikację.");
 return;
}

Stopwatch timer = Stopwatch.StartNew();

ConversationContext context = new([]);
context = await ollama.StreamCompletion(
 prompt, context, stream => Write(stream.Response));

timer.Stop();

WriteLine();
WriteLine();
WriteLine($"Czas wykonania: {timer.ElapsedMilliseconds:N0} ms");

```

4. Uruchom bez debugowania projekt *OllamaApp*.
5. Wprowadź zapytanie, np. Czym jest .NET?, i przejrzyj otrzymaną odpowiedź:

```

Wpisz swój prompt: Czym jest .NET?
.NET (pierwotnie znany jako Common Language Runtime, CLR) to framework
programistyczny stworzony przez firmę Microsoft. Jest to platforma,
która umożliwia tworzenie aplikacji komputerowych w różnych językach
programowania, takich jak C#, F#, Visual Basic .NET, itp.
...
Czas wykonania: 170 501 ms

```

Przyjrzyjmy się teraz innym narzędziom do eksperymentowania z lokalnymi LLM.

## LM Studio

Używając LM Studio, możesz:

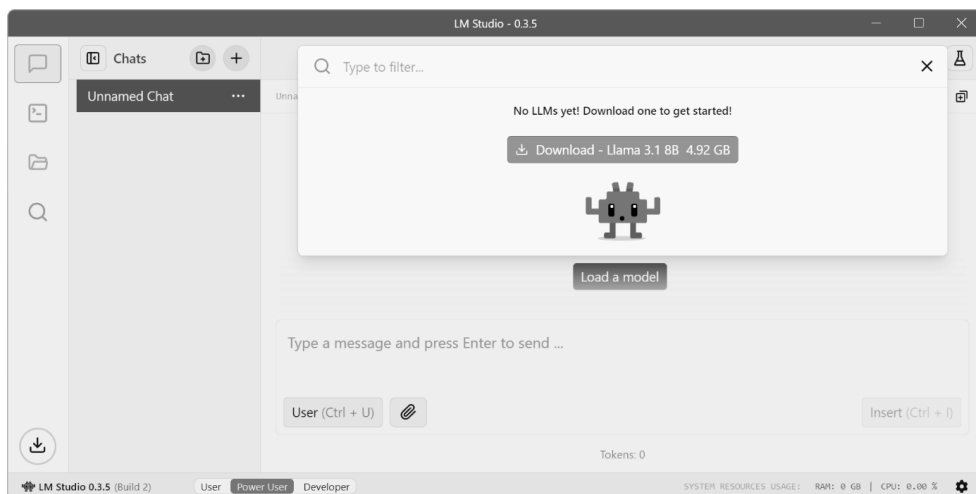
- Uruchamiać modele LLM na swoim laptopie bez dostępu do internetu.
- Korzystać z tych modeli za pomocą interfejsu chatu w aplikacji lub lokalnego serwera zgodnego z OpenAI.
- Pobierać kompatybilne pliki modeli z repozytoriów Hugging Face.
- Odkrywać nowe i interesujące modele LLM na stronie głównej aplikacji.

### Uwaga

Minimalne wymagania to komputer Mac z procesorem M1 lub komputer z systemem Windows i procesorem obsługującym zestaw instrukcji AVX2. Wersja dla systemu Linux jest obecnie w fazie beta.

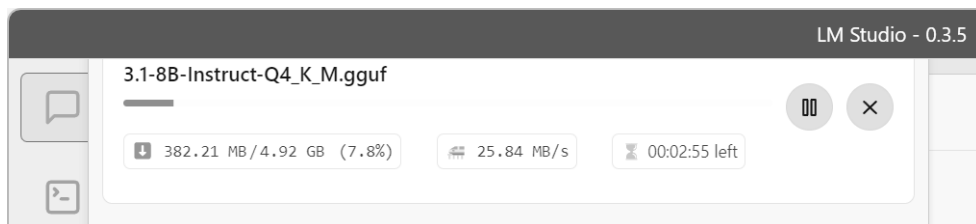
Aby zacząć używać LM Studio:

1. W przeglądarce otwórz stronę <https://lmstudio.ai/>.
2. Aby pobrać LM Studio, kliknij przycisk właściwy dla Twojego systemu operacyjnego.
3. Uruchom pobraną aplikację.
4. Na stronie głównej aplikacji zobaczysz model Llama 3 o 8 mld parametrów (**8B**) przygotowany przez firmę Meta AI, ponieważ jest to jeden z najpopularniejszych modeli. (Jeżeli aktualnie nie jest najpopularniejszy, skorzystaj z funkcji wyszukiwania, aby znaleźć ten lub inny model). Kliknij przycisk *Download*, jak na rysunku 9.8.



**Rysunek 9.8. Strona główna LM Studio z modelem Llama 3 8B firmy Meta AI**

5. Poczekaj na pobranie modelu. Postęp pobierania można monitorować, rozwijając obszar statusu, jak na rysunku 9.9.



**Rysunek 9.9. Pobieranie modelu Llama 3 8B w LM Studio**

6. Przejdź do sekcji *AI Chat* i wybierz pobrany model.
7. W polu *USER* wpisz zapytanie, np. **Czym jest .NET?**, i przyjrzyj się odpowiedzi, która powinna być dość wyczerpująca.

8. W polu *USER* wpisz inne zapytanie, np. **Kto mówi "Dlaczego Gamora?" i dlaczego?**
9. Zwróć uwagę, że odpowiedź może zawierać halucynacje, jak w poniższym przykładzie:

To pytanie odnosi się do popularnego filmu Marvela, "Guardians of the Galaxy". W jednej ze scen, główny bohater Peter Quill (Star-Lord) otrzymuje zlecenie zniszczenia planety Xandar i pyta "Dlaczego Gamora?", co oznacza, że chce wiedzieć dlaczego Gamora, strzelająca do niego z pistoletu. Głębsze znaczenie tej kwestii polega na tym, że Peter Quill, jako syn Zarekana, jest pozbawiony swojej tożsamości i rodziny, a Gamora, która go strzeliła w dzieciństwie, jest dla niego symbolem cierpienia i utraconego początku.

10. Jeżeli zadasz to samo pytanie ChatGPT-4, otrzymasz znacznie lepszą odpowiedź:

Cytat "Dlaczego Gamora?" pochodzi z filmu "Avengers: Wojna bez granic" (Avengers: Infinity War) i wypowiada go Drax, grany przez Dave'a Bautistę. Scena, w której pada ten tekst, jest humorystycznym momentem podczas konfrontacji między Strażnikami Galaktyki a Iron Manem, Doktorem Strange'em i Spider-Manem.

Jeżeli porównasz lokalny model z modelem w chmurze, to zauważysz, że modele lokalne są znacznie wolniejsze i mniej precyzyjne, choć zależy to też od możliwości Twojego sprzętu. Jednak wraz z rozwojem możliwości sztucznej inteligencji, przy wykorzystaniu urządzeń takich jak **Copilot+ PC** (np. Surface Pro 11 i Surface Laptop 7 z chipami Snapdragon X Elite lub Plus), ta sytuacja powinna ulec poprawie. Planuję przetestować, jak szybko te nowe urządzenia działają z lokalnymi modelami LLM, ale mój nowy laptop dotrze do mnie dopiero 18 czerwca 2024 r., po zakończeniu ostatnich rozdziałów tej książki.

### Więcej informacji

Dodatkowe informacje na temat urządzeń Copilot+ PC znajdziesz pod tym adresem: <https://blogs.microsoft.com/blog/2024/05/20/introducing-copilot-pcs/>.

## Ćwiczenia i dalsza nauka

Sprawdź swoją wiedzę, odpowiadając na kilka pytań dotyczących tematów poruszonych w tym rozdziale. Przeprowadź praktyczne ćwiczenia i poszerz swoją wiedzę poprzez zgłębianie tych zagadnień.

### Ćwiczenie 9.1. Materiały dostępne wyłącznie online

Wprowadzenie do Semantic Kernel: <https://devblogs.microsoft.com/semantic-kernel/hello-world/>.

Artykuł *Semantic Kernel Embeddings and Memories*: <https://devblogs.microsoft.com/semantic-kernel/semantic-kernel-embeddings-and-memories-explore-github-repos-with-chat-ui/>.

Artykuł na temat poprawiania jednoznaczności i inteligencji aplikacji używających AI za pomocą Redisa: <https://devblogs.microsoft.com/semantic-kernel/making-ai-powered-net-apps-more-consistent-and-intelligent-with-redis/>.

Wpis na blogu *Building AI-powered Microsoft Copilot with SignalR and other open-source tools*: <https://devblogs.microsoft.com/dotnet/building-ai-powered-bing-chat-with-signalr-and-other-open-source-tools/>.

Wpis na blogu *Build Intelligent Applications using ChatGPT & Azure Cosmos DB*: <https://devblogs.microsoft.com/cosmosdb/chatgpt-azure-cosmos-db/>.

Korzystanie z modeli Hugging Face z API Semantic Kernel to potężny sposób na budowanie precyzyjnych i wydajnych aplikacji przetwarzania języka naturalnego (NLP). Wykorzystując mocne strony obu narzędzi, programiści mogą tworzyć znacznie bardziej efektywne aplikacje NLP. Więcej na ten temat możesz przeczytać w tym artykule: <https://devblogs.microsoft.com/semantic-kernel/how-to-use-hugging-face-models-with-semantic-kernel/>.

## Ćwiczenie 9.2. Ćwiczenia praktyczne

Oto kilka pomysłów na inne usługi wykorzystujące modele językowe, które możesz spróbować przygotować:

1. Asystent naukowy przeszkolony na wielu artykułach, udzielający odpowiedzi z dokładnymi cytatami.
2. Chatbot programistyczny, który czyta e-booki programistyczne w formacie PDF z folderu i umożliwia prowadzenie interaktywnej rozmowy na ich temat. Jeżeli napotkasz problem na konkretnej stronie, możesz powiedzieć chatbotowi, o który fragment chodzi, pokazać komunikat o błędzie i otrzymać sensowną sugestię, jak rozwiązać ten problem.
3. Chatbot pomagający programiście .NET przygotować się do rozmowy kwalifikacyjnej.
4. Chatbot projektowy do wprowadzania nowych członków zespołu, przeszkolony na podstawie specyfikacji projektu, dokumentacji i bazy kodu, który może zaprezentować projekt nowym programistom i odpowiadać na pytania dotyczące tego projektu. Dzięki temu lider projektu nie musi odpowiadać na takie pytania na Slacku.
5. Asystent twórcy artykułów na bloga, który sugeruje tematy wpisów na bloga z poradami dotyczącymi .NET. Wybierasz jeden temat, a on pomaga Ci go napisać i następnie publikuje go na platformach LinkedIn, X, Threads i Mastodon.

## Ćwiczenie 9.3. Sprawdź swoją wiedzę

Odpowiedz na poniższe pytania. Jeśli napotkasz problemy, spróbuj wyszukać odpowiedzi w internecie. Jeśli nadal będziesz miał trudności, odpowiedzi znajdują się w dodatku.

1. Czym jest Semantic Kernel?
2. Co robi metoda `CreateFunctionFromMethod`?
3. Domyślnie każdy prompt wysyłany do modelu LLM za pomocą biblioteki Semantic Kernel jest niezależny od pozostałych promptów. Jak dodać pamięć sesji do chatu, aby model zapamiętywał wcześniejsze prompty?
4. Domyślnie musisz czekać, aż cała odpowiedź zostanie zwrócona przez model LLM. Jak włączyć strumieniowanie odpowiedzi?
5. Czym jest Hugging Face?

## Ćwiczenie 9.4. Dalsza lektura

Użyj linków z poniższej strony, aby dowiedzieć się więcej na tematy poruszane w tym rozdziale: <https://github.com/markprice/tools-skills-net8/blob/main/docs/book-links.md#chapter-9---building-a-custom-llm-based-chat-service>.

## Podsumowanie

W tym rozdziale dowiedzieliśmy się:

- Jak działają modele LLM i jak uzyskać do nich dostęp.
- Jak używać biblioteki Semantic Kernel.
- Jak rozbudowywać modele LLM za pomocą funkcji.
- Jak dodawać pamięć sesji i strumieniować wyniki.
- Czym jest Hugging Face i jak używać modeli lokalnie.

W następnym rozdziale poznamy kontenery wstrzykiwania zależności (ang. *dependency injection* — DI), które automatyzują proces zarządzania zależnościami i czasem życia usług.





# Skorowidz |

- .NET 9, 64, 748
- .NET Aspire, 543, 577, 578
  - aplikacja referencyjna eShop, 614
  - centralizacja konfiguracji, 596
  - hostowanie kontenerów, 602
  - komponenty, 600, 601
  - konfigurowanie
    - hasła, 612
    - Redisa, 599
  - metody, 583, 584, 611
  - metryki obserwowalności, 602
  - model aplikacji, 582
  - orkiestracja zasobów, 582, 593
  - projekty funkcyjne, 597
  - protokołowanie, 602
  - przeglądanie rozwiązania startowego, 589
  - pulpit nawigacyjny, 592
  - rejestrwanie serwera PostgreSQL, 611
  - szablon aplikacji startowej, 585
  - szablony projektów, 584
  - śledzenie, 602
  - tworzenie nowego rozwiązania, 606
  - typy projektów, 581
  - typy zasobów, 582
  - uruchamianie rozwiązania, 580
  - używanie woluminów danych, 612
  - wdrażanie aplikacji, 619
  - zalety
    - Podmana, 603
    - Dockera, 602

## A

- administrator bazy danych, 700
- AES, 307
- alerty, 204, 216
- algorytm
  - AES, 307
  - KMAC, 331
  - RSA, 318
  - SHA-256, 314, 318
- algorytmy
  - haszujące, 314, 662
  - podpisujące, 303
  - rekurencyjne, 662
  - sortowania, 661
  - struktury danych, 662
  - szyfrujące, 303
  - uwierzytelniające, 303
  - uzgadniania kluczy, 303
  - wyszukiwania, 661
  - złożoność, 474
- analitik
  - biznesowy, 699
  - jakości, 699
- analizowanie pamięci
  - narzędzia, 195
  - umiejętności, 195
  - w Visual Studio, 196, 197
- AOT, Ahead-Of-Time, 289
- Apache JMeter
  - mierzenie wydajności, 489
- API, Application Programming Interfaces, 232
- architekt, 701
- architektura czysta
  - dobre praktyki, 680
  - koncepcje, 678
- architektura oprogramowania, 665– 677
  - koncepcje, 667
  - Mermaid, 681
  - style, 669–673
- architektura rozwiązań, 665, 667, 677
  - koncepcje, 675, 676
  - Mermaid, 681
- asercje
  - dla ciągów znaków, 432
  - dla dat i godzin, 434
  - dla kolekcji i tablic, 433
- ASP.NET Core
  - dodanie telemetrii, 224
  - dodawanie metryk, 218
  - konteneryzacja projektu aplikacji, 570
  - testowanie strony internetowej, 509

ASP.NET Core  
 używanie tunelu deweloperskiego, 456  
 wstrzykiwanie zależności, 390  
 wzorzec  
 Adapter, 650  
 Budowniczy, 647  
 Metoda szablonowa, 652  
 asystent AI, 89, 182  
 atrybut  
 ClassData, 414, 415  
 InlineData, 413  
 MemberData, 413, 417  
 MethodData, 416  
 atrybuty, 275  
 tworzenie, 280  
 xUnit, 408  
 autoryzacja, 303, 323  
 implementowanie, 326  
 testowanie, 329  
 Azure Security and Compliance Blueprints,  
 468

## B

baza danych Northwind, 53  
 tworzenie, 54, 55  
 bazy danych  
 instancje, 449  
 migracje, 449  
 BenchmarkDotNet  
 mierzenie wydajności, 476  
 bezpieczeństwo  
 funkcje bezpieczeństwa .NET, 461  
 modelowanie zagrożeń, 462, 467  
 OWASP Top 10, 461, 462  
 testowanie aplikacji, 460  
 wymagania zgodności, 462  
 biblioteka  
 BenchmarkDotNet, 476  
 Bogus, 435  
 FakeItEasy, 427  
 klas  
 dla kontekstu danych, 59  
 dla modeli encji, 56  
 sprawdzenie integracji, 62  
 Moq, 427  
 NSubstitute, 427  
 Testcontainers, 573  
 xUnit, 405  
 Bogus  
 generatory danych, 436  
 generowanie fałszywych danych, 435

Bombardier  
 interpretacja wyników, 498  
 opcje wiersza poleceń, 492  
 pobieranie, 491  
 testowanie wydajności, 489  
 używanie, 490  
 boxing, 191

## C

chat, 334  
 dodanie  
 funkcji, 348  
 pamięci sesji, 353  
 protokołowania, 356  
 strumieniowanie wyników, 355  
 użycie  
 modelu LLM, 337  
 Semantic Kernel, 340  
 zwiększanie niezawodności, 356  
 ChatGPT, 47  
 inżynieria promptów, 49  
 Code, 34, 35  
 debugowanie, 170  
 okna, 177  
 pasek narzędzi, 175  
 dekompilacja, 98  
 funkcje refaktoryzacji, 96  
 instalowanie, 40  
 rozszerzenia, 41  
 skróty klawiaturowe, 42  
 wycinki kodu, 96  
 COM, Component Object Model, 190  
 CQRS, Command Query Responsibility  
 Segregation, 674  
 CV, curriculum vitae, 709, 710  
 CVE, Common Vulnerabilities and Exposures,  
 555  
 cykl życia  
 danych, 451  
 oprogramowania, 669  
 czasy życia zależności, 376

## D

Dapr, 603  
 wzorzec architektury sidecar, 604  
 DDD, Domain-Driven Design, 669  
 deasemblacja, 105  
 debugowanie  
 atrybuty kontrolujące, 178  
 narzędzia, 105

- okna, 177
  - pasek narzędzi, 176
  - projektów testowych, 180
  - punkt przerwania, 173
  - rezygnacja, 170
  - rozpoczynanie, 169
  - strategie, 165, 166
  - ustawianie punktu przerwania, 173
  - w Visual Studio, 170
  - wsparcie GitHub Copilot Chat, 182
  - dekompilacja, 98
    - narzędzia, 105
    - w Visual Studio, 100
    - zapobieganie, 104
  - DI, dependency injection, 370
  - diagramy
    - klas, 268, 682
    - komponentów, 683
    - Mermaid, 265, 682
    - relacji encji, 683
    - sekwencji, 682, 688–691
    - stanów, 682
  - DIP, Dependency Inversion Principle, 637
  - Discord, 50
  - DocFX, 246, 250
  - Docker, 538, 541
    - hierarchia obrazów, 559
    - instalowanie, 556
    - konfigurowanie portów, 551
    - narzędzia, 546, 547
    - obrazy kontenerów, 554
    - pojęcia, 545, 546
    - polecenia, 547, 548
    - polecenia wiersza poleceń, 546
    - publikowanie aplikacji do kontenera, 565
    - technologie, 546, 547
    - tryb interaktywny, 552
    - uruchamianie kontenera, 551
    - używanie gotowych obrazów, 556
    - warstwy obrazów, 559
    - zarządzanie kontenerami, 556
    - zmiennne środowiskowe, 553
  - dokumentacja, 232
    - Microsoft Learn, 46
  - dokumentowanie
    - kodu źródłowego, 235
    - publicznych API, 237
      - dodawanie własnych treści, 253
      - przy użyciu DocFX, 246
      - za pomocą komentarzy XML-a, 239
    - usług, 257
    - narzędzia, 258
    - wykorzystanie OpenAPI, 260
    - wizualne, 264
      - diagramy klas, 268
      - diagramy Mermaid, 266
      - schematy blokowe, 267
  - DRY, Don't Repeat Yourself, 653
  - drzewa wyrażeń, 274, 291
    - elementy, 293
    - wykonywanie, 293
  - drzewo zależności, 390
  - duple testowe, 428
  - dubler, 400, 401
- ## E
- edytor kodu
    - JetBrains Rider, 34
    - Visual Studio 2022, 34
    - Visual Studio Code, 34
  - edytory kodu
    - asystenci AI, 72
    - funkcje refaktoryzacji, 69
    - konfiguracja, 70
    - wycinki kodu, 70
- ## F
- FakeItEasy, 428
  - fałszywe dane, 435, 437, 439
  - fałszywki, fakes, 402, 426
  - FluentAssertions, 432
  - format SVG, 270
  - fragmenty kodu
    - dystribuowanie, 86
    - elementy XML-a, 83
    - importowanie, 82
    - schemat, 82
    - tworzenie, 82
    - w Visual Studio, 80
    - w Visual Studio Code, 96
  - framework
    - NBomber, 500
    - OpenTelemetry, 223
    - orleans, 604
    - Playwright, 511
  - funkcje
    - bezpieczeństwa .NET, 461
    - generycznego hosta, 379
    - refaktoryzacji, 69, 73
      - w Visual Studio, 73
      - w Visual Studio Code, 96
    - w modelach OpenAI, 347

**G**

GC, garbage collector, 192  
 generatory  
   danych, 436  
   kodu, 274, 294  
 generowanie  
   dokumentacji, 246  
   fałszywych danych, 435  
   kluczy, 305  
   liczb losowych, 321  
   testów, 529  
   wektorów IV, 305  
 generyczny host .NET, 379  
   budowanie, 380  
   funkcje, 379  
   implementacja w .NET, 379  
   metoda CreateDefaultBuilder, 380  
   poznawanie usług i zdarzeń, 384  
 Git, 124, 732  
   cofanie commitów, 139  
   czyszczenie commitu, 140  
   dodawanie plików, 132  
   filtrowanie historii commitów, 152  
   funkcje, 125  
   ignorowanie plików, 143  
   integracja z Visual Studio, 126  
   konfiguracja domyślnej gałęzi, 130  
   konfiguracja tożsamości, 127  
   podpisy SSH, 128  
   polecenia, 140, 162  
   polecenia przechowalni, 142  
   pomoc dla poleceń, 130  
   przechowalnia, 141  
   role w zespole, 126  
   rozgałęzianie, 155  
   rozpoczynanie pracy, 131  
   scalanie, 155  
   stany plików, 133  
   śledzenie zmian, 133  
   tworzenie repozytorium, 134  
   umieszczanie plików, 138  
   usuwanie gałęzi, 161  
   wyświetlanie  
     historii commitów, 148  
     listy gałęzi, 161  
     różnic w plikach, 146  
 GitHub, 28, 45  
   Codespaces, 35  
   Copilot, 89  
   Copilot Chat, 182  
 GPT, Generative Pre-trained Transformer,  
 335  
 grafy zależności, 389

**H**

haszowanie, 302, 313, 314  
 Hugging Face, 358

**I**

IDE, 34  
 identyfikatory środowiska  
   uruchomieniowego, RID, 99  
 identyfikowanie użytkownika, 324  
 implementacja generycznego hosta, 379  
 instrukcje  
   LINQ, 77  
   pliku Dockerfile, 549  
 IntelliSense, 245  
 interfejs  
   IDisposable, 193  
   ILogger, 205  
   używanie, 207  
   IMemoryCache, 377  
   ITestOutputHelper, 418  
   wiersza poleceń, 454  
 inżynier DevOps, 700  
 IoC, Inversion of Control, 371  
 ISP, Interface Segregation Principle, 633

**J**

JetBrains Rider, *Patrz Rider*  
 język  
   Markdown, 254  
   pośredni IL, 102

**K**

kierownik projektu, 698  
 KISS, Keep It Simple, Stupid, 653  
 klasa  
   Faker<T>, 435  
   Random, 322  
 klucze, 303  
 kod niebezpieczny, unsafe code, 187  
 komentarze XML-a, 239  
   znaczniki, 241  
 komentowanie kodu źródłowego, 237  
 kompilator  
   AOT, 289, 491, 492  
   Just-In-Time, JIT, 484  
 kontener DI, 390

kontenery  
działanie, 540  
rejestry, 543  
testowe, 573  
zalety, 540  
konteneryzacja, 538  
projektu aplikacji ASP.NET Core, 570  
projektu aplikacji konsoli, 563  
Kubernetes, 542

## L

liczby pseudolosowe, 322  
lider techniczny, 701  
LLM, Large Language Model, 334, 335  
LM Studio, 365  
LoD, Law of Demeter, 655  
LSP, Liskov Substitution Principle, 629

## M

magazyn danych, 449  
makiety, dummies, 426  
Markdown, 254  
bloki kodu, 256  
formatowanie tekstu, 255  
kolorowanie składni, 256  
linki i obrazy, 255  
nagłówki, 254  
składnia języka, 255  
tabele, 256  
tworzenie list, 255  
Mermaid, 232, 264  
rodzaje diagramów, 265, 682  
rysowanie diagramów, 266, 681  
składnia diagramów sekwencji, 688  
składnia schematów blokowych, 684  
zapisywanie diagramów, 270  
metoda  
CreateDefaultBuilder, 380  
CryptographicOperations.HashData(), 330  
metody  
Aspire, 583, 584, 611  
klasy Faker<T>, 435  
NBombera, 502  
rejestrowania usług, 388  
rozszerzające, 391  
testujące, 413  
metodyki cyklu życia oprogramowania, 669  
Agile, 670  
Extreme Programming, XP, 670  
Kanban, 670

Lean, 670  
Rapid Application Development, 670  
Scrum, 670  
Waterfall, 670  
metryka pokrycia kodu, 400  
metryki, 203, 215, 472  
Bombardiera, 499  
implementowanie, 217  
obserwowalności, 602  
statystyczne, 475  
wyświetlanie, 221  
Microsoft Threat Modeling Tool, 467  
mierzenie wydajności, *Patrz* wydajność  
aplikacji  
migawka, 199  
mocki, 400, 401, 425  
mockowanie  
biblioteki, 427  
przy użyciu NSubstitute, 429  
zalety, 426  
modele LLM  
działanie, 335  
OpenAI, 340  
funkcje, 347  
uzyskiwanie dostępu, 337  
zarządzanie, 361  
modelowanie zagrożeń, 462, 467  
monitorowanie, 204  
za pomocą metryk, 215  
Moq, 427  
MUT, Method Under Test, 400, 405

## N

narzędzia, 31  
AI w Chrome, 45  
do analizy pamięci, 195  
do dekompilacji, 105  
do dokumentowania usług, 258  
do obserwowalności, 204  
do testowania bezpieczeństwa, 461  
Dockera, 547  
JetBrains, 44  
OWASP, 468  
podstawowe, 47  
w Visual Studio 2022, 72  
w Visual Studio Code, 95  
w Visual Studio Enterprise, 38  
wiersza poleceń, 731  
narzędzie  
Bombardier, 489  
curl, 454  
DAST, 461

## narzędzie

- DocFX, 246
- Docker Compose, 603
- Dotfuscator, 105
- dotnet, 46
- dotnet-ef, 56
- Homebrew, 454
- IL Viewer, 98, 106
- ILSpy, 100
- Microsoft Threat Modeling Tool, 467
- Playwright Inspector, 529
- Puppeteer, 513
- SAST, 461
- Security Development Lifecycle, 467
- Selenium, 513
- Sharplab.io, 106
- winget, 454
- Wizualizator tekstu, 175

## NBomber, 500

- metody, 502
- scenariusze, 500
- symulacje obciążenia, 501
- typy, 502

## notacja

- dużego O, 473
- omega, 475
- theta, 475

## NSubstitute, 427

- metody rozszerzające, 429
- mockowanie, 429
- tworzenie dubli, 428

**O**

## obciążenia

- rodzaje, 487
- symulacje, 501
- testowanie, 500

## obiekt zastępczy, substitute, 428

## obniżanie poziomu kodu, 106

## obrazy kontenerów

- .NET, 554
- Dockera, 548, 554

## ochrona danych, 301

## OCP, Open/Closed Principle, 627

## oczyszczanie pamięci, GC, 192

## odwrócenie sterowania, IoC, 371

## Ollama, 359

- interfejs wiersza poleceń, 362
- modele, 360, 361
- pakiet OllamaSharp, 363

## OpenAPI, 259

- metody dokumentowania usługi, 260

## OpenTelemetry, 222

- instrumentacja projektów, 224
- obsługiwane pakiety, 223
- Protocol, 223
- przeglądanie danych, 226

## Orleans, 604

## OWASP, Open Web Application Security

- Project, 462
- Top 10, 461, 462

**P**

## pakiety systemu protokołowania, 207

## Playwright, 511

- alternatywy dla .NET, 513

## Inspector

- generowanie testów, 529

## metody

- automatyzacji lokalizatorów, 516
- automatyzacji stron, 514
- lokalizacji elementów, 515
- testowanie eShopOnWeb, 517
- formularze, 526
- JavaScript, 528
- klikanie elementów, 524
- listy rozwijane, 524
- responsywność, 526
- uwierzytelnianie, 526
- walidacja, 526

## testowanie zautomatyzowane, 517

## typy testów, 514

## zalety, 512

## pliki

- .editorconfig, 71
- .pdb, 105
- Dockerfile, 548
- instrukcje, 549

## podpisywanie, 303

- danych, 318

## PoLA, Principle of Least Astonishment, 658

## polecenia

- Dockera, 547, 548
- Gita, 140, 149, 162

## pomoc, 46

- ChatGPT, 47
- dla dotnet, 46
- dokumentacja techniczna, 46
- na Discord, 50

## PostgreSQL, 611

## praca inżyniera oprogramowania

- najważniejsze umiejętności, 696
- podanie o pracę, 709

- poszukiwanie, 707
- proces onboardingu, 701, 702
- programowanie w parach, 704
- role w zespole, 698
- rozmowa kwalifikacyjna, 710–716
  - metoda STAR, 721
  - przygotowania, 718, 719
  - pytania behawioralne, 719
  - pytania techniczne, 730–744
  - trudne pytania, 726
  - typowe pytania, 724
- stanowiska, 697
- szkolenie i rozwój, 701
- ścieżka kariery, 697
- umiejętności, 707
- umiejętności miękkie, 697
- wsparcie modelu językowego, 705
- wspólne zadania, 696
- programista frontendowy, 700
- Project Tye, 605
- projektant doświadczeń użytkownika, 699
- projektowanie
  - domenowe, DDD, 669
  - systemów, 33
- protokołowanie, 203, 356, 602
  - centralne, 204
  - pakiety, 207
  - strukturalne, 204
  - tworzenie usługi sieciowej, 209
- protokół
  - OTLP, 223
  - SignalR, 510
- przygotowanie testu, 400

## R

- Redis
  - konfigurowanie, 599
- refaktoryzacja, 69
  - funkcje, 73
- refleksja, 274, 275
  - ostrzeżenie, 289
  - w .NET 9, 290
- regresja, 400
- rejestrwanie usług
  - dla funkcji, 391
  - metody, 388
  - z różnymi kluczami, 377
  - za pomocą metod rozszerzających, 391
- rejestry kontenerów, 543
- relacyjna baza danych, 53
- repozytoria zdalne, 152

- Rider, 34, 36
  - debugowanie, 170
  - instalowanie, 43
  - obniżanie poziomu kodu, 106
  - wtyczka IL Viewer, 106
- rozszerzenia Code, 41
- rozszyfrowywanie, 306
- rozwiązywanie usług, 389, 393
- rysowanie diagramów, 681

## S

- schemat blokowy, 267, 682–687
- SCM, Source Code Management, 121
- Security Development Lifecycle, 467
- Semantic Kernel, 341
- SignalR, 510
- skrótów klawiaturowe, 92, 93
- SOLID
  - zasada
    - odwracania zależności, DIP, 637
    - otwarte – zamknięte, OCP, 627
    - podstawień Liskov, LSP, 629
    - pojedynczej odpowiedzialności, SRP, 624
    - segregacji interfejsów, ISP, 633
- sól, 305
- SQL Server, 56
  - konfigurowanie, 54
- SRP, Single Responsibility Principle, 624
- sterta, heap, 184
  - alokacja typów, 186
- stos, stack, 184
  - alokacja typów, 186
  - technologiczny .NET Aspire, 578
- struktury danych, 660
- strumieniowanie, 355
- SUT, System Under Test, 400
- system
  - CVE, 555
  - kontroli wersji, VCS, 121
    - Git, 123
    - Mercurial, 123
    - rozproszony, 123
    - scentralizowany, 123
    - SVN, 123
- szablony
  - elementów, 111
  - projektów, 111
    - testowanie, 117
    - tworzenie, 112

szpiegi, spies, 426  
 szyfrowanie, 302, 306  
 symetryczne, 307

## Ś

śledzenie żądań, 203, 204, 602

## T

TDD, Test-Driven Development, 403

- dobrze praktyki, 404
- główne zasady, 403
- wady, 404
- zalety, 403

telemetria, 222

- testowanie, 226

test

- bezpieczeństwa, 398, 460
- end-to-end, 398, 508
  - scenariusz, 509, 510
- funkcjonalny, 399, 508
  - scenariusz, 509, 510
- integracyjny, 398, 445
  - analiza, 447
  - współdzielenie środowiska, 446
  - wykorzystanie magazynów danych, 449

jednostkowy, 397

- atrybuty xUnit, 408
- izolowanie, 405
- nazewnictwo, 406
- pakiety, 398
- sygnały ostrzegawcze, 418
- tworzenie, 406, 410
- tworzenie asercji, 432
- używanie xUnit, 406

obciążeniowy, 398, 486, 500

użyteczności, 399

wydajności, 398, 489

- izolowanie, 482
- unikanie błędów, 482
- wytrzymałości, 398, 488

Testcontainers, 573

- działanie biblioteki, 573

- zastosowanie, 574

tester, 699, 717

testowanie, 32, 397

- aplikacji eShopOnWeb, 517
- aplikacji w czasie rzeczywistym, 510
- fałszywych danych, 437
- funkcji usługi sieciowej, 214

metod za pomocą

- ClassData, 414, 415

- InlineData, 413

- MemberData, 413, 417

- MethodData, 416

- Playwrighta, 514

metody rozszerzające, 429, 432

piramida testów, 446

przygotowanie środowiska testowego,  
 419, 422

serwisów, 533

strategia TDD, 403

strony internetowej ASP.NET Core, 509

systemów zewnętrznych, 446

szablonu projektu, 117

usług, 453

usługi Web API, 509

webowych interfejsów użytkownika, 511

testy

- cechy, 399

- generowanie, 529

- mockowanie, 425

- scenariusze wyników, 402

- uruchamianie, 63

- wyniki, 401

- wyświetlanie, 418

tunele deweloperskie, 453

- interfejs wiersza poleceń, 454

- w ASP.NET Core, 456

- w Visual Studio, 458, 459

tworzenie

- aplikacji startowej Aspire, 586

- asercji, 432

- bazy danych, 54, 55

- biblioteki klas, 56, 59

- chatu, 334

- dubli, 428

- fragmentu kodu, 82

- generatorów kodu, 294

- generycznego hosta, 380

- niestandardowych atrybutów, 280

- obrazów kontenerów, 548

- projektu testowego, 62, 409

- szablonów projektów, 111

- testów jednostkowych, 406, 410

typy

- referencyjne, 184

- wartości, 184



## U

Ubuntu, 555  
umiejętności, 32, 48  
uruchamianie testów, 63  
usługa chatu, 334  
usługi  
  czas życia  
    scoped, 376  
    singleton, 377  
    transient, 376  
  metody rejestrowania, 388  
  rozwiązywanie, 393, 389  
  testowanie, 453  
  usuwanie, 390  
uwierzytelnianie, 303, 323  
  implementacja, 326

## V

VCS, Version Control System, 121  
Visual Studio, 34  
  asystent AI, 89  
  debugowanie interaktywne, 170  
  dekompilacja, 98  
    rozszerzenie ILSpy, 100  
    zaciemnianie kodu, 105  
  diagnostyka pamięci, 196  
  funkcje refaktoryzacji, 73  
  funkcje usprawniające edycję, 91  
  instalowanie, 37  
  konfiguracja  
    edytora, 86  
    tunelu deweloperskiego, 459  
  menu Gita, 127  
  narzędzia diagnostyczne, 198  
  nawigowanie, 91  
  obsługa Aspire, 580  
  okna debugowania, 177  
  pasek narzędzi debugowania, 175  
  przeglądanie kodu źródłowego, 103  
  skrótów klawiszowe, 38  
  szablony projektów i elementów, 111  
  tworzenie  
    migawki, 199  
    tunelu deweloperskiego, 458  
  wycinki kodu, 80  
Visual Studio Code, *Patrz* Code  
Visual Studio Enterprise, 38  
  narzędzia, 38

## W

wdrażanie aplikacji, 37  
  za pomocą Aspire, 619  
wektor inicjujący, initialization vector, 304  
wielki model językowy, LLM, 334  
wiersz poleceń  
  Aspire, 580  
  Bombardiera, 492  
  Dockera, 546  
  pomoc dla dotnet, 46  
  rozszerzenia Code, 42  
  tunelu deweloperskiego, 454  
  w .NET, 731  
wskaźniki, 187  
  zastosowania, 190  
wstrzykiwanie zależności, DI, 370  
  do metod akcji, 394  
  do minimalnego API, 394  
  najlepsze praktyki, 379  
  przez konstruktor, 373, 374, 391  
  przez metodę, 373, 376  
  przez właściwość, 373, 375  
  stosowanie, 372  
  w .NET, 372  
  w ASP.NET Core, 390  
  w widokach, 393  
wycinki kodu, 70  
wydajność aplikacji  
  pomiar, 471  
    bazowy, 472  
    unikanie błędów, 482, 485  
  testowanie, 489  
  używanie  
    Apache JMeter, 489  
    BenchmarkDotNet, 476  
    Bombardiera, 489  
  złożoność  
    czasowa, 474  
    pamięciowa, 474  
wyjątki, 378  
wykres Gantta, 683  
wymagania zgodności, 462  
wzorce projektowe, 640  
  behawioralne, 641–644, 650  
  kreacyjne, 640–642, 645  
  strukturalne, 641–643, 647  
  wstrzykiwanie zależności, 371

wzorzec

- Adapter, 648
- Budowniczy, 645
- CQRS, 674
- Metoda szablonowa, 650
- Wrapper, 648

## X

xUnit, 405

- atrybuty, 408
- testowanie serwisów, 533
- testy jednostkowe, 406
- wyświetlanie wyników, 418

## Y

YAGNI, You Ain't Gonna Need It, 654

## Z

zarządzanie

- kodem źródłowym, SCM, 121
- funkcje, 122
- rodzaje systemów, 123
- zdalnymi repozytoriami, 152

zasady projektowe, 653

- DRY, 653
- KISS, 653
- kompozycja jest lepsza od dziedziczenia, 656
- prawo Demeter, LoD, 655
- SOLID, 624
- YAGNI, 654
- zasada najmniejszego zaskoczenia, PoLA, 658

zatycki, stubs, 402, 426

zestawy .NET, 275

- dekompilowanie, 98
- dynamiczne ładowanie, 283
- metadane, 275
- odczytywanie metadanych, 277
- wersjonowanie, 276

zintegrowane środowisko programistyczne, IDE, 34

złożoność algorytmów

- czasowa, 474
- pamięciowa, 474

zmienne środowiskowe, 553

# PROGRAM PARTNERSKI

— GRUPY HELION —



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

GRUPA  
**Helion** 

# Od juniora do eksperta. Wszystko, co musisz wiedzieć o .NET!

Większość książek programistycznych obejmuje jedynie wybrane tematy, takie jak bezpieczeństwo, testowanie aplikacji czy wdrażanie w chmurze. Inne koncentrują się na architekturze aplikacji, wzorcach projektowych albo przygotowaniu się do rozmów kwalifikacyjnych. Trudno jednak znaleźć przystępny przewodnik, który pozwoliłby krok po kroku przyswoić wszystkie umiejętności zawodowego programisty .NET.

Dzięki temu podręcznikowi odblokujesz swój potencjał i otworzysz drogę do kariery. Nauczysz się zarządzania kodem źródłowym przy użyciu Gita i skutecznego nawigowania po projektach. Odkryjesz zaawansowane techniki debugowania i dokumentowania kodu, co poprawi jego czytelność i ułatwi utrzymanie projektów. Zgłębisz też tajniki kryptografii, by zapewnić poufność i spójność danych na każdym etapie cyklu życia aplikacji. W książce nie zabrakło wnikliwego spojrzenia na nowoczesne zagadnienia, takie jak budowanie inteligentnych aplikacji korzystających z modeli LLM, wstrzykiwanie zależności, testowanie czy konteneryzacja za pomocą Dockera. Dzięki wskazówkom dotyczącym najlepszych praktyk architektury oprogramowania zaczniesz tworzyć solidne, skalowalne i łatwe w utrzymaniu aplikacje!

## W książce:

- zaawansowane techniki debugowania
- ochrona danych i aplikacji przy użyciu kryptografii
- zastosowanie modeli LLM i programowanie chmurowe z .NET Aspire
- wstrzykiwanie zależności
- optymalizacja wydajności poprzez benchmarking i testowanie
- przygotowanie się do rozmowy kwalifikacyjnej

**Mark J. Price** specjalizuje się w programowaniu w języku C#. Pracuje w Microsoftzie, tworzy rozwiązania dla Microsoft Azure. Zdał ponad 80 egzaminów Microsoftu. Zajmuje się też dydaktyką: przeszkolił wielu przyszłych programistów, od 16-letnich początkujących aż po 70-latków na emeryturze.

	<b>KOD KORZYŚCI</b> Sięgnij po więcej! ▶	
 <a href="https://helion.pl">helion.pl</a>	ISBN 978-83-289-2233-4	
 <b>HELION S.A.</b> ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	 9 788328 922334	
<b>Cena: 139,00 zł</b>		

**<packt>**