Politechnika Częstochowska

Wydział Informatyki i Sztucznej Inteligencji

PRACA DYPLOMOWA MAGISTERSKA

Architektura mikroserwisów i rozwiązania chmurowe w ekosystemie .NET

Microservices Architecture and Cloud Solutions in the .NET Ecosystem

Mariusz Trzeciak

Nr albumu: 133583

Kierunek: Informatyka

Forma studiów: stacjonarne

Poziom studiów: II

Promotor pracy:

dr inż. Bartosz Kowalczyk

Praca przyjęta dnia:

Podpis promotora:

Częstochowa, 2025

Niniejszym chciałbym serdecznie podziękować
. . .
za bezcenne wsparcie udzielone mi
w trakcie trwania studiów.

Spis treści

Wstęp

W ostatnim czasie obserwuje się intensywny rozwój [Proszę uzupełnić].

Niestety [Proszę wpisać wadę lub niedogodność istniejących rozwiązań].

W związku z tym, ciekawym wydaje się zaprojektowanie i zrealizowanie [Proszę uzupełnić].

Cel pracy

Celem niniejszej pracy jest zaprojektowanie i zrealizowanie [Proszę uzupełnić].

Zakres pracy

Zakres pracy obejmuje:

- zebranie wiadomości z zakresu [Proszę uzupełnić],
- porównanie [Proszę uzupełnić],
- opracowanie założeń projektowych dotyczących [Proszę uzupełnić],
- implementację [Proszę uzupełnić],
- przetestowanie [Proszę uzupełnić].

Rozdział 1

Wprowadzenie do architektury mikroserwisów w ekosystemie .NET

1.1 Definicja i charakterystyka architektury mikroserwisowej

Ten podrozdział stanowił będzie fundamentalne wprowadzenie teoretyczne niezbędne dla zrozumienia całej pracy. Architektura mikroserwisowa to podejście do projektowania aplikacji jako zestawu małych, niezależnych usług, z których każda działa w swoim własnym procesie i komunikuje się przez lekkie mechanizmy. Definicja ta jest kluczowa dla czytelnika, ponieważ ustala terminologię i koncepcyjne podstawy dla dalszych rozdziałów.

1.2 Porównanie z architekturą monolityczną

Porównanie z architekturą monolityczną jest niezbędne dla ukazania kontekstu i motywacji wyboru mikroserwisów. Ten podrozdział uzasadniał się koniecznością pokazania różnic w podejściu do projektowania, wdrażania i skalowania aplikacji. Tradycyjna architektura monolityczna używa pojedynczego modelu danych dla operacji odczytu i zapisu, co może prowadzić do problemów z wydajnością i skalowalością. Mikroserwisy natomiast pozwalają na niezależne skalowanie komponentów i izolację awarii. To porównanie pomaga czytelnikowi zrozumieć, dlaczego organizacje wybierają mikroserwisy i jakie są minusy tej decyzji.

1.3 Zalety i wyzwania mikroserwisów

Przedstawienie zalet i wyzwań jest kluczowe dla obiektywnej oceny architektury mikroserwisowej. Zalety obejmują zwiększoną skalowalność, elastyczność technolo-

giczną oraz odporność na awarie. Jednak mikroserwisy wprowadzają również znaczne wyzwania, takie jak złożoność zarządzania danymi rozproszonymi, konieczność implementacji mechanizmów komunikacji międzyserwisowej oraz trudności w testowaniu. Ten podrozdział jest niezbędny dla zrównoważonego przedstawienia tematu i przygotowania czytelnika na dalsze szczegóły techniczne.

1.4 Ekosystem .NET w kontekście mikroserwisów

Uzasadnienie tego podrozdziału wynika z tematu pracy skupiającego się na ekosystemie .NET. Platforma .NET Core oferuje zaawansowane narzędzia do budowy systemów mikroserwisowych, włączając integrację z nowoczesnymi technologiami chmurowymi. ASP.NET Core zapewnia natywne wsparcie dla gRPC, REST API oraz zaawansowanych wzorców architektonicznych. Ten podrozdział ustala technologiczne podstawy dla praktycznej części pracy i pokazuje, dlaczego .NET jest odpowiednim wyborem dla implementacji mikroserwisów.

Rozdział 2

Mechanizmy komunikacji w architekturze mikroserwisowej

2.1 Komunikacja synchroniczna

2.1.1 REST API w ASP.NET Core

REST pozostaje najpopularniejszym stylem architektonicznym dla komunikacji między mikroserwisami. Uzasadnienie tego podrozdziału wynika z konieczności przedstawienia tradycyjnego podejścia, które będzie później porównywane z gRPC w części badawczej pracy. REST wykorzystuje protokół HTTP/1.1 i jest intuicyjny dla programistów webowych. ASP.NET Core oferuje bogate wsparcie dla tworzenia REST API z funkcjami takimi jak automatyczna walidacja modeli, routing oraz integracja z OpenAPI.

2.1.2 gRPC - architektura i implementacja

Ten podrozdział jest kluczowy dla aspektu badawczego pracy. gRPC wykorzystuje HTTP/2 i Protocol Buffers, co zapewnia 5-10 razy większą przepustowość niż REST. Framework ten oferuje unikalne funkcje jak np. silne typowanie interfejsów. Szczegółowe omówienie architektury gRPC jest niezbędne dla zrozumienia wyników testów wydajnościowych, które będą przeprowadzone w części empirycznej pracy.

2.1.3 GraphQL jako alternatywa

GraphQL zasługuje na uwagę jako alternatywne podejście do projektowania API, które może wpływać na wydajność komunikacji. Ten podrozdział uzasadnia się potrzebą przedstawienia kompletnego spektrum opcji komunikacji synchronicznej.

2.2 Komunikacja asynchroniczna

2.2.1 Message Brokers (RabbitMQ, Azure Service Bus)

Komunikacja asynchroniczna jest fundamentalna dla osiągnięcia luźnego sprzężenia w architekturze mikroserwisowej. Message brokers eliminują bezpośrednie zależności między serwisami i umożliwiają implementację wzorców event-driven. RabbitMQ i Azure Service Bus są wiodącymi rozwiązaniami w ekosystemie .NET, oferującymi różne mechanizmy dostarczania wiadomości i gwarancje spójności.

2.2.2 Event-driven Architecture

Event-driven architecture stanowi kluczowy wzorzec dla mikroserwisów, umożliwiając reaktywne przetwarzanie zdarzeń. Ten podrozdział uzasadnia się koniecznością pokazania, jak zdarzenia mogą zastąpić bezpośrednie wywołania API i zwiększyć elastyczność systemu. Wzorzec ten jest szczególnie istotny w kontekście Event Sourcing, który będzie omówiony w dalszej części pracy.

2.3 Komunikacja w czasie rzeczywistym

2.3.1 WebSockets w aplikacjach internetowych

WebSockets są kluczowe dla komunikacji w czasie rzeczywistym, oferując pełnodupleksową komunikację między klientem a serwerem. WebSockets eliminują ograniczenia tradycyjnego modelu request-response HTTP i umożliwiają natychmiastowe przekazywanie aktualizacji.

2.3.2 Server-Sent Events (SSE)

SSE stanowią alternatywę dla WebSockets w scenariuszach jednostronnej komunikacji serwer-klient. Uzasadnienie tego podrozdziału wynika z potrzeby przedstawienia pełnego spektrum opcji komunikacji w czasie rzeczywistym.

2.3.3 gRPC Streaming

gRPC oferuje unikalne możliwości streamingu, które łączą wydajność binarnej komunikacji z elastycznością streaming. Ten podrozdział uzasadnia się potrzebą pokazania, jak gRPC może konkurować z WebSockets w aplikacjach wymagających komunikacji w czasie rzeczywistym.

Rozdział 3

gRPC-Web i komunikacja frontend-backend

3.1 Ograniczenia natywnego gRPC w przeglądarkach

Przeglądarki nie oferują wystarczającej kontroli nad żądaniami HTTP/2, aby wspierać pełną specyfikację gRPC. Brak dostępu do surowych ramek HTTP/2 oraz trailing headers uniemożliwia natywną implementację gRPC w JavaScript. Ten podrozdział jest kluczowy dla zrozumienia, dlaczego potrzebne są dodatkowe rozwiązania.

3.2 Architektura gRPC-Web

3.2.1 Rola proxy w komunikacji gRPC-Web

gRPC-Web wymaga warstwy translacyjnej między przeglądarką a backendem gRPC. Ten podrozdział uzasadnia się koniecznością wyjaśnienia, jak proxy tłumaczy uproszczone żądania gRPC-Web na natywny protokół gRPC.

3.2.2 Envoy Proxy i nginx jako reverse proxy

Envoy i nginx to najpopularniejsze rozwiązania proxy dla gRPC-Web.

3.3 Implementacja gRPC-Web w aplikacjach frontendowych

3.3.1 Integracja z frameworkami JavaScript

Brak natywnego wsparcia dla gRPC-Web w popularnych frameworkach frontendowych stanowi praktyczne wyzwanie. Ten podrozdział uzasadnia się potrzebą poka-

zania, jak integrować gRPC-Web z React, Angular czy Vue.js oraz jakie są związane z tym trudności i ograniczenia.

3.3.2 Pakiety npm i konfiguracja

Szczegółowe omówienie wymaganych pakietów npm (@types/google-protobuf, grpc-web) jest uzasadnione przez praktyczny aspekt implementacji. Ten podrozdział pomoże czytelnikom zrozumieć konkretne kroki niezbędne do wdrożenia gRPC-Web w projektach frontendowych.

3.4 GraphQL Gateway dla gRPC

Integracja GraphQL z gRPC przez grpc-graphql-gateway oferuje alternatywne podejście do ekspozycji usług gRPC.

Rozdział 4

Wzorce projektowe i architektoniczne

4.1 CQRS (Command Query Responsibility Segregation)

CQRS jest fundamentalnym wzorcem dla mikroserwisów, pozwalającym na separację operacji odczytu i zapisu. Uzasadnienie tego podrozdziału wynika z faktu, że CQRS umożliwia niezależną optymalizację wydajności dla różnych typów operacji, co jest kluczowe w środowisku mikroserwisowym.

4.2 Event Sourcing

Event Sourcing jest ściśle powiązany z CQRS i oferuje unikalne korzyści jak 100% audit logging oraz możliwość odtworzenia stanu aplikacji. Ten podrozdział uzasadnia się przez konieczność pokazania, jak Event Sourcing rozwiązuje problem atomowego aktualizowania bazy danych i wysyłania wiadomości. Wzorzec ten jest szczególnie istotny w kontekście event-driven architecture w mikroserwisach.

4.3 Database-per-Service

Wzorzec Database-per-Service jest fundamentalny dla osiągnięcia niezależności mikroserwisów. Każdy serwis zarządza własną bazą danych, co zapewnia izolację, skalowalność i elastyczność technologiczną. Ten podrozdział uzasadnia się przez konieczność pokazania, jak ten wzorzec wpływa na zarządzanie danymi rozproszonymi oraz jakie wprowadza wyzwania związane z transakcjami i ich spójnością.

4.4 API Gateway

4.4.1 Porównanie Ocelot i Envoy

API Gateway jest kluczowym komponentem architektury mikroserwisowej, centralizującym routing, uwierzytelnianie i monitoring. Porównanie Ocelot (natywnego dla .NET) z Envoy uzasadnia się przez praktyczne potrzeby wyboru odpowiedniego rozwiązania. Envoy oferuje lepszą wydajność (100k RPS vs 10k RPS dla Ocelot) ale wymaga dodatkowej konfiguracji.

4.4.2 Circuit Breaker Pattern

Circuit Breaker jest kluczowy dla zapewnienia odporności systemu mikroserwisowego na awarie. Ten podrozdział uzasadnia się przez konieczność pokazania, jak wzorzec ten zapobiega kaskadowym awariom.

Rozdział 5

Orkiestracja kontenerów i rozwiązania chmurowe

5.1 Kubernetes w ekosystemie .NET

5.1.1 Azure Kubernetes Service (AKS)

AKS jest wiodącą platformą dla deploynetu mikroserwisów .NET w chmurze Azure. Ten podrozdział uzasadnia się przez praktyczne potrzeby wdrażania i zarządzania mikroserwisami w środowisku produkcyjnym. AKS oferuje zarządzaną płaszczyznę kontrolną Kubernetes oraz integrację z innymi usługami Azure.

5.1.2 Wdrażanie z Helm Charts

Helm Charts stanowią standard dla wdrażania aplikacji w Kubernetes. Ten podrozdział uzasadnia się przez konieczność pokazania, jak automatyzować deployment mikroserwisów oraz zarządzać ich konfiguracją w różnych środowiskach. Helm umożliwia wersjonowanie i przywracanie poprzednich wersji wdrożeń, co jest kluczowe dla CI/CD.

5.2 Service Mesh

5.2.1 Linkerd i Istio

Service Mesh rozwiązuje problemy komunikacji, bezpieczeństwa i observability w architekturze mikroserwisowej. Porównanie Linkerd i Istio uzasadnia się przez znaczące różnice w wydajności - Linkerd dodaje 40-400% mniej latencji niż Istio. Ten podrozdział jest istotny dla zrozumienia, jak Service Mesh wpływa na wydajność komunikacji między mikroserwisami.

5.2.2 Distributed Tracing

Distributed Tracing jest niezbędny dla debugowania i monitorowania systemów mikroserwisowych.

5.3 CI/CD Pipeline dla mikroserwisów

CI/CD dla mikroserwisów wprowadza unikalne wyzwania związane z niezależnym deploynmentem serwisów. Azure DevOps oferuje dedykowane narzędzia dla tego celu.

Rozdział 6

Metodologia badań i projekt eksperymentu

Uzasadnienie części empirycznej (Rozdziały 6-8). Te rozdziały realizują główny cel badawczy pracy poprzez praktyczne testy wydajnościowe w różnych scenariuszach obciążenia i rozmiarów danych.

6.1 Cele i hipotezy badawcze

6.2 Projekt aplikacji testowej

6.2.1 Architektura systemu testowego

6.2.2 Implementacja mikroserwisów w .NET

6.3 Scenariusze testowe

6.3.1 Testy obciążeniowe

6.3.2 Testy wydajnościowe

6.4 Narzędzia i środowisko testowe

Rozdział 7

Porównawcza analiza wydajności gRPC vs REST

7.1 Przeprowadzenie testów wydajnościowych

7.1.1 Różne rozmiary payload'ów

7.1.2 Różne poziomy obciążenia

7.1.3 Modele komunikacji (synchroniczna, strumieniowa)

7.2 Analiza wyników

7.2.1 Przepustowość (transakcje na sekundę)

7.2.2 Opóźnienia i czas odpowiedzi

7.2.3 Wykorzystanie zasobów systemowych

7.3 Porównanie z WebSockets i gRPC-Web

Rozdział 8

Studium przypadku: aplikacja eShop

8.1 Architektura aplikacji referencyjnej

8.2 Implementacja wybranych wzorców

8.3 Testowanie i walidacja rozwiązań

Rozdział 9

Podsumowanie

W ramach niniejszej pracy:

- zebrano wiadomości z zakresu [Proszę wpisać tematykę],
- opisano istniejące rozwiązania z zakresu [Proszę wpisać],
- zaprojektowano autorski system informatyczny [Proszę wpisać pełną nazwę systemu],
- zaimplementowano autorski system informatyczny [Proszę wpisać pełną nazwę systemu],
- przetestowano [Proszę wpisać, co przetestowano w pracy].

Zaprojektowany i zrealizowany w pracy autorski [Proszę wpisać pełną nazwę systemu] charakteryzuje się tym, że:

- ... [Proszę wpisać zaletę systemu],
- ... [Proszę wpisać zaletę systemu],
- .. [Proszę wpisać zaletę systemu].

Zaprojektowany i zrealizowany w pracy autorski [Proszę wpisać pełną nazwę systemu] można wykorzystać m.in. do:

- ... [Proszę wpisać potencjalną możliwość zastosowania],
- ... [Proszę wpisać potencjalną możliwość zastosowania],
- ... [Proszę wpisać potencjalną możliwość zastosowania].

Zaprojektowany i zrealizowany w pracy autorski [Proszę wpisać pełną nazwę systemu] można w przyszłości rozbudować o następujące funkcje:

- ... [Proszę wpisać potencjalną możliwość rozbudowy],
- ... [Proszę wpisać potencjalną możliwość rozbudowy],
- ... [Proszę wpisać potencjalną możliwość rozbudowy].

Podsumowując można stwierdzić, że [Proszę wpisać pełną nazwę systemu], ze względu na [Proszę raz jeszcze wymienić podstawową zaletę rozwiązania], może być niezwykle użyteczny w [Proszę krótko wpisać obszar zastosowań].

Bibliografia

[1] Zasady pisania prac dyplomowych, https://wiisi.pcz.pl/student-wiisi/vademecum-studenta/praca-dyplomowa, stan na dzień: 26.10.2024

[2] Strona internetowa Wydziału Informatyki i Sztucznej Inteligencji, https://wiisi.pcz.pl, stan na dzień: 26.10.2024

Dodatek. Zawartość dołączonej płyty

Do niniejszej pracy dołączono płytę z następującą zawartością:

- Dokument pracy w formatach tex i pdf.
- Kod źródłowy zaprojektowanego i zrealizowanego w ramach pracy systemu.

Spis rysunków

Spis tabel

Spis listingów

Streszczenie

Praca pt. *„Tytuł pracy”* nawiązuje do bardzo aktualnych zagadnień współczesnej informatyki, a mianowicie do [Proszę wpisać ogólną tematykę pracy]. W ostatnim czasie obserwuje się intensywny rozwój [Proszę wpisać ogólne wprowadzenie do tematu]. Jednocześnie [Proszę wpisać wady lub niedogodności takiego rozwiązania]. W rozważanej pracy podjęto zatem próbę rozwiązania tego problemu. W szczególności zebrano wiadomości z zakresu [Proszę wpisać tematykę], porównano [Proszę wpisać, jakie rozwiązania działające porównano], opracowano [Proszę wpisać, co opracowano w pracy], zrealizowano [Proszę wpisać, co zrealizowano w pracy], przetestowano [Proszę wpisać, co przetestowano w pracy]. Zaprojektowany i zaimplementowany system charakteryzuje się następującymi cechami: [Proszę uzupełnić], [Proszę uzupełnić], [Proszę uzupełnić]. Dzięki temu zaprojektowany i zaimplementowany system może być wykorzystany w praktyce np. do: [Proszę wpisać potencjalną możliwość zastosowania], [Proszę wpisać potencjalną możliwość zastosowania], [Proszę wpisać potencjalną możliwość zastosowania].

Do realizacji praktycznej części pracy wykorzystano najnowsze narzędzia i technologie informatyczne, w tym m.in. [Proszę wpisać nazwę technologii], [Proszę wpisać nazwę technologii], [Proszę wpisać nazwę technologii].

Zaprojektowany i zaimplementowany system można w przyszłości przykładowo rozbudować o następujące funkcje: [Proszę wpisać], [Proszę wpisać] i [Proszę wpisać].

Summary

Proszę umieścić tłumaczenie polskiej wersji streszczenia.

Słowa kluczowe

informatyka;
zaprojektowanie autorskiego systemu informatycznego;
implementacja autorskiego systemu informatycznego;
przetestowanie autorskiego systemu informatycznego;
[Proszę wpisać słowo dotyczące tematyki pracy];
[Proszę wpisać słowo dotyczące kluczowej funkcjonalności systemu];
[Proszę wpisać słowo dotyczące użytej technologii];
[Proszę wpisać słowo dotyczące użytej technologii];
[Proszę wpisać słowo dotyczące użytej technologii]

Częstochowa, dd.mm.20rr

Imię i nazwisko: Mariusz Trzeciak
Nr albumu: 133583
Kierunek: Informatyka
Wydział: Informatyki i Sztucznej Inteligencji

Oświadczenie autora pracy dyplomowej*

Oświadczam pod rygorem odpowiedzialności karnej, że złożona przeze mnie praca dyplomowa pt. *„Architektura mikroserwisów i rozwiązania chmurowe w ekosystemie .NET”* jest moim samodzielnym opracowaniem i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Jednocześnie oświadczam, że moja praca (w całości ani we fragmentach) nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w Politechnice Częstochowskiej.

Wyrażam/~~nie wyrażam~~** zgodę/~~zgody~~ na nieodpłatne wykorzystanie przez Politechnikę Częstochowską całości lub fragmentów wyżej wymienionej pracy w publikacjach Politechniki Częstochowskiej.

.....
podpis studenta

*W przypadku zbiorowej pracy dyplomowej, dołącza się oświadczenia każdego ze współautorów pracy dyplomowej.
*Niepotrzebne skreślić.