POLISH-JAPANESE ACADEMY
OF INFORMATION TECHNOLOGY

Warsaw, February 26, 2025

Abstract. Rapid development and adoption of microservices architecture in last few years changed the world of Informtics. Microservices architecture has many advantages, such as flexibility, fast and comfortable deployment, and easy maintenance. However, this architecture also causes new challenges, such as code duplication and complicated code management. In my master's thesis, I've ceried on deep and detailed analysis of all available approaches to manage shared code in system of microservices. In my work, I want to compare all available approaches of code share code in system of microservices using literature sources and prepared code laboratory, also to define cases in which it is better to choose one or another approach. In my practical part, I want to present my own approach to sharing code that will combine the advantages of all existing solutions and provide the best performance and code comfort for developer in code management. At the end of my work, I will place comparison of my solution with existing solutions using prepared performance tests.

Keywords: Microservices Architecture · Code Sharing · Performance.

Streszczenie Szybki rozwój i porwszechne przyjęcie architektóry mikroserwisowej w ostatnich latach zminił świat informatyki. Architerkura mikroserwisowa ma wile zalet, takich jak elastyczność, latłe wdrożenia i prostota w utrzymaniu. Natomast powoduje nowe wyzwanie, takie jak, na przykład, problem duplikacji kodu i zarządzanie kodem. W mojej pracy magisterskiej przeprowadziłem dokładną analizę metod i podejść wspówdzielenia kodu w systemach mikroserwisów. W mojej prace chcę porównać za pomocą źródeł literaturowych oraz preprowadonych badań dostępne metody wspówdzielenia kodu, zdofiniować przypadki w których warto wybrać takie lub inne podejście. Oraz w czesci praktycznej chcę zaproponować łasne wonoczesne podejście do wsówdzielenia kodu które połączy zalety istniejących rozwiązań i zapełni wydajność najbardziej oraz komfort w zarządzaniu kodem. Na końcu pracy znjaduje się poruwnanie mojego zowiązania z istniejącymi za pomocą przygotowanych testów wydajnośowych.

Keywords: Architektura mikrousług · Współdzielenie kodu · Wydajność.

Spis treści

1 Wstęp

1.1 Zakres pracy

W pracy magisterskiej przeprowadzono dokładną analizę metod i podejść do współdzielenia kodu w systemach mikroserwisów, zakres tej pracy zawiera porównanie między architekturą monolitową a mikroserwisową, identyfikację typów obiektów wykorzystywanych w kodzie wwspółczesnych aplikacji, analizę dostępnych metod i podejść do wspówdzielenia kodu IDL, SDK oraz biblioteki, praktyczna implementacja aplikacji wspierającej nowatorskie podejście do wpówdzielenia kodu ServerLess.

1.2 Motywacja

Główną motywacją do napisania pracy było rosnacę w współczesnych czasach zainteresowanie mikroserwisowym podejściem do architektury, które ma wiele zalet, takich jak elastyczność, łatwość wdrożenia i utrzymania na produkcji, możliwość łatwej i szybkiej naprawy problemów na produkcji ale natomiast ma róenież powoduje nowe wyzwania, takie jak problem duplikacji kodu i zapotrzebowanie na współdzielenie kodu między mikroserwisami w systemie.

W mojej pracy chcę przeanalizować jakie metody współdzielenia kodu są dostępne na rynku, jakie są ich zalety i wady, jakie są najlepsze praktyki w spówdzieleniu kodu w systemach mikroserwisów. Porównać bardziej traqdycyjne podejścia do wspówdzielenia kodu, takie jak IDL, SDK, biblioteki z nowatorskim podejściem do wspówdzielenia kodu takich jak ServerLess, chcę stożyć oprogramanie które połączy główne zalety pradycyjnych modod wsówdzieniena kodu z zaletami bardziej nowczesnych podejść.

1.3 Zawartość pracy

Wpaca jesu ustrukturyzowana w kilka rozdziałów, które wprowadzają czytelnika najpierw czszegóły teoretyaczne a później w praktyczne aspekty wspówdzielenia kodu w systemach mikroserwisów. Króki opis rozdziałów znajduje się poniżej:

1. Wstęp - W tym rozdziale analizuję wady i zalety achitertrury mikroserwisowej, monolitwej, porównuję dlaczego z architektury monolitowej powstawa architektura mikroserwisowa, analizuje przybaki w których lepiej uzyc arcitektóry monolotowej a w któray architakturę mikroserwisową. Opisuję dlaczego powstało zapotrzebowanie na współdzielenie kodu. Wyjaśniam logikę leżącą u podstaw eksploracji wielu metod współdzielenia kodu. Uzasadnim informację przykladami z literatury.
2. Architektura mikroserwisowa a Monolitowa - W tym rozdziale porównuję architekturę mikroserwisową z monolitową, przedstawiam różnie w utrumaniu i skalowalności, za pomozą źródeł literaturowych prozentuję.
3. Analiza dziedziny problemowej - W tym rozdziale pracy kategoryzuję typy obiektów któte mogą potrzebować współdzielenia kodu w systemie mikroserwisów, analizuję możliwe wyswania zwiazane z współdzieleniem poszczególnych typów obiektów, przedstawiam uzasadnienia z literatury.
4. Analiza metod współdzielenia kodu - W tym rozdziale kompleksowa analizę istniejących metod współdzielenia kodu w systemach mikroserwisów za pomocą źródeł literaturowych krytrrium oceniaia.
5. Opis części praktycznej - Zawira opis przygtowanej w ramach pracy aplikacji która wpomaga nowoczesne rozwiąznie do współdzielenia kodu w systemach mikroserwisów - Server Less rozszeżając możliwosci tego rozwiązania.

2 Architektura mikro serwisowa a monolitowa

Rozwój architektury mikro serwisowej w ostatnich latach przyniósł wiele korzyści, takich jak łatwość skalowania, niezależność wdrożenia i elastyczność. Jednak, wraz z tą elastycznością, pojawiają się również nowe wyzwania, związane między innymi z odpowiednim współdzieleniem obiektów pomiędzy mikro serwisami.

2.1 Architektura mikroserwisowa

Podział aplikacji na mniejsze, niezależne serwisy umożliwia elastyczne skalowanie poszczególnych komponentów, co przekłada się na lepszą wydajność i dostępność systemu. Ponadto, rozbudowa i utrzymanie aplikacji w oparciu o mikro serwisy jest znacznie prostsza, ponieważ każdy serwis może być rozwijany niezależnie. Taka modularna struktura pozwala na szybsze wprowadzanie nowych funkcjonalności oraz łatwiejsze naprawianie błędów. Istotną korzyścią jest również możliwość korzystania z różnych technologii w poszczególnych serwisach, co daje większą elastyczność i umożliwia wykorzystanie najlepszych narzędzi dla każdej części aplikacji. W rezultacie, architektura mikro serwisowa umożliwia bardziej efektywne zarządzanie projektem, lepszą skalowalność zespołu oraz izolację błędów, co przyczynia się do wyższej jakości i niezawodności systemu.

> "Microservices are small, autonomous services that work together. Let's break that definition down a bit and consider the characteristics that make microservices different." [4, p. 2]

Charakterystyka architektury mikrousług: Modułowość - Usługi są modułowe, co umożliwia łatwiejszy rozwój, konserwację i skalowalność, ponieważ każda usługa kontroluje określoną funkcję lub cechę. Solidność - Mikrousługi zwiększają ogólną solidność systemu, ponieważ błędy w jednej usłudze nie wpływają na całą aplikację, zapewniając tolerancję błędów i niezawodność systemu. Interoperacyjność - Różne mikrousługi komunikują się za pośrednictwem dobrze zdefiniowanych interfejsów API, zapewniając bezproblemową integrację i interakcję usług. Równoległy rozwój - Oddzielne zespoły programistyczne mogą pracować nad różnymi mikrousługami jednocześnie,

co przyspiesza rozwój i funkcje. Elastyczność w zakresie elastycznych technologii - Różne mikrousługi można budować przy użyciu różnych technologii, co pozwala na wykorzystanie najskuteczniejszych narzędzi dla każdej przydzielonej funkcji/cechy. [7]

2.2 Architektura monolitowa

Architektura monolityczna, która opiera się na jednym spójnym kodzie źródłowym, oferuje pewne korzyści w kontekście prostoty zarządzania i łatwości wdrożenia. Wszystkie komponenty aplikacji są ze sobą ściśle powiązane, co ułatwia debugowanie i śledzenie błędów. Ponadto, brak konieczności konfigurowania i zarządzania infrastrukturą dla wielu usług upraszcza proces wdrażania. W przypadku mniejszych projektów o prostszych wymaganiach, architektura monolityczna może być bardziej efektywna i wydajna, eliminując niepotrzebną złożoność komunikacji między usługami. Jednak warto pamiętać, że architektura monolityczna może napotkać trudności w skalowaniu i utrzymaniu w przypadku większych, bardziej złożonych systemów.

Charakterystyka architektury monolitycznej: Pojedyncza jednostka wdrożenia - Cała aplikacja jest wdrażana jako pojedyncza, niepodzielna jednostka. Wszelkie uaktualnienia, ulepszenia lub modyfikacje wymagają wdrożenia całej aplikacji, w tym wszystkich jej komponentów. Scentralizowany przepływ kontroli - Centralny moduł lub funkcja podstawowa nadzoruje przepływ kontroli w aplikacji, koordynując sekwencyjny postęp wykonywania od jednego komponentu do następnego. Ścisłe sprzężenie - Komponenty i moduły w aplikacji są silnie powiązane i zależne od siebie. Współdzielona pamięć - Wszystkie komponenty oprogramowania mają bezpośredni dostęp do zasobów pamięci, co sprzyja ścisłej integracji. Jednak taka konfiguracja może również powodować konflikty zasobów i trudności ze skalowaniem aplikacji. [7]

2.3 Porównanie

W spówczesnym świecie programowania, zawsze wybieramy między podejściem monolitycznym i mikroserwisowym, to wiąże się z kompromisem między prostotą, głuwną cechą podejścia monolotycznego

a elastycznością w przypadku podejścia mirkoserwisowego. Podejście monolitowe, które polega na przechowywaniu wszyskich komponentów suytemu w jednym miejscu, oferuje łatwośc w rozroju i zarządzaniu kodem ale natomiast może stać wąskim gardłem kiedy aplikacja urośnie. W przeciwienstwie do architektury monolitowej, architektura mikroserwisowa dekomponuje aplikacje w małe, niezależnie zarządzane i drażane mikrousługi, umożliwiające granularną skalowalność i niezależne aktualizacje, zmniejszając w ten sposób ryzyko tego, że zmiana w jednym module spowoduje awarię innych. Natomiast takie podejście dodaje skomplikowanoście projektom poprzez zapotrzebowanie na obługę komunikacji międze mikroserwisami, versjonowaniem, współdzielem kodu.

> "I should call out that microservices are no free lunch or silver bullet, and make for a bad choice as a golden hammer. They have all the associated complexities of distributed systems." [4, p. 11]

Zalety architektury monolitowej: Czas napisaznie aplikacji - W przypadku małych i średnich aplikacji budowanie aplikacji z architekturą monolityczną jest łatwiejsze i szybsze. Zespół programistów może pracować efektywniej z ujednoliconą bazą kodu bez konfigurowania i zarządzania komunikacją między usługami. Łatwe wdrożenie - Wdrożenie architektury monolitycznej obejmuje wdrożenie pojedynczej jednostki, co jest mniej złożone i wymaga mniejszej liczby katalogów konfiguracyjnych niż systemy rozproszone. Uproszczone testowanie i debugowani - O wiele watwiejszym jest testowanie aplikacji monolitycznej. Ze względu na ścisłą integrację wszystkich komponentów, testy jednostkowe i integracyjne można przeprowadzać w ramach pojedynczej bazy kodu, co upraszcza proces testowania. Skalowalność - Architektura monolityczna, w przypadku małych i średnich aplikacji, zapewnia odpowiednią skalowalność poprzez replikację całej aplikacji. [7]

Zalety architektury moikroserwisowej: Skalowalność - Mikrousługi umożliwiają niezależne skalowanie różnych usług w oparciu o ich zapotrzebowanie, optymalizując wykorzystanie zasobów i zapewniając wydajną wydajność nawet podczas szczytów ruchu. Utrzymanie kodu - Mikrousługi umożliwiają równoległy rozwój przez różne zespoły, co prowadzi do szybszego rozwoju funkcji i szybszego wdraża-

nia. Elastyczność w trakcie wyboru technologii - Każda mikrousługa może być rozwijana przy użyciu najbardziej odpowiedniego stosu technologicznego dla jej konkretnej funkcji, co promuje elastyczność i adaptowalność do różnych technologii w ramach tej samej aplikacji. Łatwe debugowanie - Lokalizowanie i rozwiązywanie błędów w poszczególnych usługach jest proste. Bezpieczeństwo - Mikrousługi ułatwiają separację danych. Każda usługa ma swoją bazę danych, co utrudnia hakerom próbę naruszenia aplikacji. [7]

Wybór między architerkturą monolotową a mikroserwisową zależy od wilu faktorów, takich jak zormiar projektu, zapotrzebowanie na ciągły rozwój, wymagania dotycącze skalowalnosci oraz kompetencje i doświadczenia zespołu. Architektura monolitowa lepiej pasuje dla małych i mniej skomplikowanych projektów natomiast mikroserwisowa oferuje benefity p rzypadku wykorzystania w większych, bardziej skomplikowanych systemach.

3 Analiza dziedziny problemowej

Celem niniejszej pracy magisterskiej jest zgłębienie tematu współdzielenia obiektów w systemach mikroserwisów oraz zrozumienie wyzwań i możliwości związanych z tym zagadnieniem. Praca ma na celu analizę różnych strategii, narzędzi i metodyk, które mogą pomóc w efektywnym i bezpiecznym współdzieleniu danych i obiektów w skali mikroserwisowej architektury.

3.1 Typy obiektów w aplikacjach mikroserwisowych

Postanowiłem rozpocząć analizę dziedziny problemowej od tego że za pomocą źródeł literaturowych zidentyfikuję typy obiektów oraz dokonać analizy tego, które z nich mogą wymagać współdzielenia w systemach mikroserwisów.

1. DTO (Data Transfer Object) - Takie obiekty są specjalnie przeznaczone do obsługi przesyłania danych między elementami systemu mikroserwisów, są lekkie i nie zawierają logiki biznesowej, dlatego współdzielenie takich obiektów jest jak najbardziej zalecane ponieważ zapewniają spójność danych po obu stronach komunikacji oraz zmniejsza ilość błędów w przypadku rozwoju lub utrzymania aplikacji.
2. Model (lub Encja) - Najczęściej są logicznie powiązane z tabelami w bazie danych, ze względu na to że we współczesnych czasach za dobrą praktykę jest uważane podejście w którym jeden mikroserwis jest powiązany z maksymalnie jedną bazą danych, współdzielenie tego typu obiektów nie jest zalecane.
3. Wyjątek (Exception) - Zalecanym jest współdzielenie informacji o wyjątkach w systemie mikroserwisów tym samym tworząc jednolitą obsługę błędów w systemie. Logika związana z obsługą błędów, w przypadku podobności w kilku serwisach też jest dobrym kandydatem do współdzielenia. Taka standaryzacja w systemie mikroserwisów sprawia że łatwiej później znaleźć źródło błędu i naprawić go.
4. Walidatory - W przypadku współdzielenia walidatorów w systemie mikroserwisów można zapewnić integralność danych w całym systemie oraz zmniejszyć duplikację kodu, również standaryzacja

walidacji obiektów w systemie zmniejsza ilość ewentualnych problemów w trakcie ewolucji i rozwoju mikroserwisów w przyszłości, współdzielenie takich obiektów jest dobrą praktyką i jest zalecane.

5. Serwisy - W przypadku logiki która powtarza się w dwóch i więcej serwisach zalecanym jest współdzielenie takiego serwisu za pomocą najlepiej w tym przypadku pasującej metody, zmniejsza to duplikację kodu natomiast zwiększa czytelność i spójność kodu w systemie mikroserwisów, zmniejsza ilość ewentualnych problemów w trakcie ewolucji i rozwoju kodu.

Zawsze należy pamiętać w trakcie developmentu aplikacji, że współdzielenie kodu ma wiele zalet, natomiast nadmierne współdzielenie kodu może spowodować nadmierne powiązania między mikroserwisami i utrudnić niezależną ewolucję poszczególnych serwisów oraz zniweczyć zalety infrastruktury mikroserwisowej. Również nadmierne współdzielenie kodu utrudnia testowanie aplikacji co może spowodować zmniejszenie jakości kodu. Również nadmierne współdzielenie kodu może spowodować błędy kaskadowe, czyli błędy które pojawiają się w kilku miejscach w systemie jednocześnie spowodowane współdzielonym kawałkiem kodu, utrudnia utrzymanie aplikacji również są powodem na to, żeby podchodzić do współdzielenia kodu ostrożnie podejmować przemyślaną decyzję w każdym konkretnym przypadku.

3.2 Kryteria porównania i analizy metod współdzielenia kodu

Po przeprowadzeniu dogłębnej analizy tematu i między innymi źródeł literaturowych, takich jak [4], [6], [5], [2] mogę zdefiniować kryteria porównania metod współdzielenia kodu w systemach mikroserwisów jako następujące:

1. Komunikacja - Metoda współdzielenia obiektów powinna uwzględniać efektywną komunikację między serwisami. Ważne jest, aby obiekty były dostępne w sposób, który minimalizuje opóźnienia i obciążenie sieci. Dobre rozwiązania mogą obejmować użycie asynchronicznej komunikacji, takiej jak kolejki wiadomości, czy też bezpośrednie zapytania między serwisami.
2. Izolacja - Metoda współdzielenia obiektów powinna zapewniać odpowiednią izolację między serwisami. Każdy serwis powinien mieć kontrolę nad swoimi własnymi obiektami i nie powinien być zależny od innych serwisów. To pozwala na większą elastyczność i umożliwia niezależny rozwój i skalowanie serwisów.
3. Bezpieczeństwo - W przypadku współdzielenia obiektów, istotne jest zapewnienie odpowiedniego poziomu bezpieczeństwa. Dostęp do obiektów powinien być kontrolowany i zabezpieczony w sposób, który zapobiega nieautoryzowanemu dostępowi. Mechanizmy uwierzytelniania i autoryzacji powinny być odpowiednio wdrożone, aby zapewnić bezpieczne współdzielenie obiektów.
4. Skalowalność - Metoda współdzielenia obiektów powinna być skalowalna. System powinien być w stanie efektywnie obsługiwać rosnącą liczbę żądań i zapewniać odpowiednią wydajność. Współdzielenie obiektów powinno być projektowane w taki sposób, aby możliwe było łatwe skalowanie poszczególnych serwisów bez wpływu na cały system.
5. Wersjonowanie - Ważne jest również odpowiednie zarządzanie wersjami obiektów, szczególnie w środowisku mikroserwisów, gdzie różne serwisy mogą używać różnych wersji obiektów. Metoda współdzielenia obiektów powinna uwzględniać zarządzanie wersjami i umożliwiać aktualizacje w sposób kontrolowany i bezpieczny.

4 Analiza metod współdzielenia kodu

We współczesnych czasach współdzielenie kodu jest niezbędne w skomplikowanych systemach mikroserwisów, w tym rozdziale pracy porównuję istniejące podejścia do współdzielenia kodu oraz porównam je za pomocą zdefiniowanych w poprzednim rozdziale kryteriów. Głównym celem jest zdefiniowanie najlepszych praktyk współdzielenia kodu, zdefiniować, które podejście najbardziej dopasowane do współdzielenia konkretnych typów obiektów, zdefiniowanych powyżej w tej pracy oraz porównanie wydajności i możliwości skalowania w przypadku każdego z podejść za pomocą przygotowanych programistycznych testów wydajnościowych.

4.1 Metody współdzielenia kodu z opisem

Za pomocą źródeł literatury oraz własnego doświadczenia zdefiniowałem dostępne na dzień dzisiejszy podejścia:

1. Interface definition languages - do IDL należą wiele popularnych technologii, Protocol buffers, Avro IDL, Open API. [8]
2. Libraries - najbardziej oczywiste podejście, które daje możliwość współdzielenia wszystkich rodzajów obiektów w aplikacji.
3. Client Libraries -
4. Wyniesienie kodu do osobnego REST serwisu - podejście polega na przechwytywaniu i udostępnieniu wspólnej logiki poprzez utworzenie osobnego mikroserwisu.
5. Serverless - nowoczesne podejście do pisania kodu i przetwarzania informacji wykorzystywane w chmurach obliczeniowych. Pozwala na uruchamianie kawałków kodu, metod i funkcji niezależnie, bez warstwy zarządzania aplikacją, a także na uruchamianie kodu bez konieczności zarządzania podstawową infrastrukturą. [1]

Pryncypia działania poszczególnych metod współdzielenia kodu:
Interface definition languages – są powszechnie używane w oprogramowaniu zdalnych wywołań procedur. W takich przypadkach maszyny po obu stronach łącza mogą używać różnych systemów operacyjnych i języków programowania. IDL oferują most pomiędzy dwoma różnymi systemami. Natomiast również mogą być użyte dla generacji obiektów lub kodów w systemach mikroserwisów. Każdy

system IDL posiada określony przez twórców język IDL oraz interpretator języka IDL. Interpretator języka IDL przygotowany i dostarczony przez twórców potrafi na podstawie udostępnionych reguł wygenerować kod używając przygotowane pliki IDL. Używając języka IDL możemy przygotować obiekty lub kod zapisany za pomocą języka IDL, a później udostępnić przygotowane pliki IDL nieograniczonej ilości mikroserwisów i na podstawie udostępnionych plików wygenerować w każdym z mikroserwisów kod lub obiekty, które zostaną użyte przez specyficzną logikę konkretnego serwisu dla osiągnięcia konkretnego celu biznesowego. Tworzenie kodu na podstawie plików IDL jest łatwo zautomatyzowane za pomocą narzędzi do budowania aplikacji, takich jak Gradle czy Maven, dlatego możemy używać IDL jako metodę współdzielenia kodu. W trakcie pracy zamierzam sprawdzić skuteczność tej metody, problemy związane z jej użyciem oraz określić przypadki, w których dobrze się nadaje, jak również przypadki, w których lepiej jej nie stosować.

Libraries – biblioteki to w odpowiedni sposób przygotowany kod, który my za pomocą odpowiednich narzędzi możemy łatwo importować i używać jako część innego programu. Program importujący bibliotekę może używać kodu biblioteki tak, jakby to był własny kod programu. Istnieje wiele narzędzi, które wspomagają łatwe i szybkie importowanie i zarządzanie bibliotekami kodu, takie jak Maven czy Gradle. W współczesnych systemach mikroserwisów współdzielenie kodu za pomocą bibliotek kodu odbywa się za pomocą serwisów do przechowywania artefaktów i plików binarnych, takich jak Nexus. [3, 5] Na pierwszym etapie narzędzie do budowania projektu przygotowuje bibliotekę i wysyła ją na serwer. Dalej kod może być przechowywany nieograniczoną ilość czasu na serwerze. Aplikacje, które mają dostęp do serwera, mogą pobrać bibliotekę z kodem i przechowywać w lokalnym systemie plików, używając jako część kodu źródłowego. Biblioteka może być udostępniona nieograniczonej liczbie projektów. Organizacja może ograniczać dostęp do serwera.

SDK – mechanizm działania podobny do bibliotek, różni się jedynie podejściem. W przypadku kodu SDK na serwerze udostępniającym dependencje przechowywane są jedynie obiekty, które muszą być użyte do komunikacji z innym programem lub kodem. Logika biznesowa nie została udostępniona w takim przypadku i zostaje ukryta oraz nie może zostać zmieniona. W przypadku udostępnienia

SDK możemy łatwo zapewnić bezpieczeństwo kodu (użytkownicy wciągający dependencje nie widzą logiki biznesowej) oraz chronimy się przed przypadkowymi oraz niepożądanymi zmianami kodu. Pozwala to zaoszczędzić na testach manualnych oraz automatycznych kodu. Przykład, w którym możemy użyć współdzielenia kodu SDK to – mamy mikroserwis, który udostępnia API. API przyjmuje obiekty JSON, które mogą być opisane za pomocą obiektów Java. Obiekty, które pozostałe mikroserwisy w systemie mikroserwisów mogą użyć do wysłania żądań do określonego wcześniej mikroserwisu udostępniającego API, możemy udostępnić dla naszego systemu mikroserwisów jako bibliotekę. Każdy serwis, który chce wysyłać żądania do serwisu REST-owego, może pobrać bibliotekę w postaci dependencji za pomocą narzędzia do budowania i użyć przygotowane obiekty do komunikacji. Natomiast kod serwisu nie zostanie udostępniony. W ramach prac nad serwisem korzystającym z API REST-owego nie musimy testować kodu API, bo on nie został udostępniony i dlatego nie mógł ulec zmianie w trakcie pisania kodu serwisu.

REST API - Representational State Transfer Application Programming Interface jeden z najpopularniejszych podejść do komunikacji między mikroserwisami i dla współdzielenia kodu w systemach mikroserwisów. Przykładami współdzielenia kodu za pomocą REST mogą być mikroserwis do uwierzytelniania użytkowników, który może być wykorzystany przez każdy serwis w systemie mikroserwisów, tym samym logika związana z uwierzytelnieniem użytkowników jest współdzielona między elementami systemu. REST API pozwala na kompletne odseparowanie współdzielonego kawałka logiki od implementacji aplikacji, tym samym redukując problemy związane z zarządzaniem wersjami, które występują w przypadku bibliotek i SDK. Również współdzielenie kodu za pomocą REST API nie powoduje zależności i sztywnych powiązań między mikroserwisami, co sprawia, że system jest bardziej elastyczny, natomiast trudniej w takim przypadku utrzymać spójność API kontraktów w systemie. Współdzielenie kodu za pośrednictwem interfejsu API REST obejmuje udostępnianie danych lub funkcji za pomocą standardowych metod HTTP (GET, POST, PUT, DELETE) i wymianę informacji w formatach takich jak JSON lub XML. Do zalet podejścia można odnieść dobrą skalowalność, nowe mikroserwisy mogą być łatwo dodawane i usuwane bez wpływu na cały system, również REST API pozwala na

lepszy i bardziej zrozumiały podział odpowiedzialności w systemie, co prowadzi do zmniejszenia duplikacji kodu. Do wad tego podejścia możemy odnieść trudności w utrzymaniu, w przeciwieństwie do bibliotek i SDK zmiany w API nie są automatycznie wykrywane przez klientów, również różni klienci mogą równocześnie używać różnych wersji API, co wymaga mechanizmów kontroli wersji.

Serverless - Nowoczesne podejście, które pozwala na uruchamianie kodu bez bezpośredniego zarządzania infrastrukturą i sprzętem. W przypadku tego podejścia chmura obliczeniowa zarządza przydzieleniem odpowiednich zasobów, zarządza skalowaniem i utrzymaniem serwera, dając możliwość programiście skupić się na napisaniu kodu. W kontekście współdzielenia kodu, serverless pozwala na stworzenie małych, niezależnych funkcji, które mogą być łatwo wykorzystane do współdzielenia kodu w systemach mikroserwisów. Takie podejście dobrze pasuje do współdzielenia małych, często powtarzających się kawałków kodu. Technologia serverless również pozwala usprawnić współpracę między zespołami, zmniejsza duplikację kodu i upraszcza konserwację. Funkcje współdzielone za pomocą serverless mogą być udostępniane za pomocą technologii REST, co wiąże się z wszystkimi zaletami i wadami tego podejścia, albo za pomocą SDK udostępnionego przez dostawcę chmury.

Spis rysunków

Spis tablic

Literatura

1. IBM. What is serverless computing?, 2024. Accessed: 2025-02-26.
2. Martin Kleppmann. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, 2017.
3. Mohamed Labouardy. *Pipeline as Code: Continuous Delivery with Jenkins, Kubernetes, and Terraform*. Manning Publications, 2021.
4. Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2015.
5. Michael T. Nygard. *Release It!: Design and Deploy Production-Ready Software*. Pragmatic Bookshelf, 2008.
6. Chris Richardson. *Microservices Patterns: With Examples in Java*. Manning Publications, 2018.

7. Harish Sharma. Monolithic vs microservices architecture: Advantages, disadvantages, and differences, February 2023.
8. Wikipedia contributors. Interface description language, 2025. Accessed: 2025-02-26.

5 Appendices

5.1 Sample Code

5.2 Sample Data