

Wydział Informatyki

Temat:

współdzielenie obiektów w mikroserwisach

**STRESZCZENIE** Praca analizuje możliwości współdzielenia kodu w systemach opartych na architekturze mikroserwisowej. Omówione są dostępne na rynku metodyki oraz rozwiązania wspierające współdzielenie kodu między mikroserwisami. Praca zawiera porównanie tych metod i rozwiązań pod względem wydajności, komunikacji, izolacji, bezpieczeństwa, skalowalności oraz wersjonowania, oparte na przygotowanym projekcie.

Praca omawia możliwe kombinacje metod i narzędzi w celu osiągnięcia najwyższego poziomu wydajności, komunikacji, izolacji, bezpieczeństwa, skalowalności oraz wersjonowania w systemie mikroserwisów. Dodatkowo określa możliwe przypadki użycia architektury mikroserwisowej oraz najlepsze kombinacje metod i narzędzi dla każdego przypadku.

W ramach pracy magisterskiej przedstawiony jest system aplikacji webowych oparty na architekturze mikroserwisowej, który prezentuje możliwe podejścia oraz narzędzia wspomagające współdzielenie kodu w systemie mikroserwisów. Na podstawie tego systemu przeprowadzone zostały badania rozwiązań i narzędzi pod kątem wydajności, komunikacji, izolacji, bezpieczeństwa, skalowalności oraz wersjonowania. Przygotowane zostały także kombinacje metod i narzędzi dla osiągnięcia najwyższego poziomu w tych aspektach.

**ABSTRACT**

The following thesis explores the possibilities of code sharing in systems based on a microservices architecture. It presents available market methodologies and solutions that support code sharing among microservices in a system. The thesis provides a comparison of methods and solutions in terms of performance, communication, isolation, security, scalability, and versioning based on a prepared project.

The work proposes possible combinations of methods and tools to achieve the highest level of performance, communication, isolation, security, scalability, and versioning in a microservices system. It also identifies potential use cases for microservices architecture and the best combination of methods and tools for each case.

As part of the master's thesis, the foundation of web application systems based on microservices architecture is presented, showcasing possible approaches and tools that support code sharing in a microservices system. Based on the system, studies were conducted on solutions and tools in terms of performance, communication, isolation, security, scalability, versioning, and combinations of methods and tools were prepared to achieve the highest level of performance, communication, isolation, security, scalability, and versioning.

**SPIS TREŚCI**

1.WSTEP

* 1. ZAKRES PRACY
  2. MOTYWACJA
  3. ZAWARTOŚĆ PRACY

2. ARCHITEKTURA MIKRO SERWISOWA A MONOLITOWA

2.1 ARCHITEKTURA MIKRO SERWISOWA

2.2 ARCHITEKTURA MONOLITOWA  
 2.3 PORÓWNANIE

3. ANALIZA DZIEDZINY PROBLEMOWEJ

3.1 TYPY OBIEKTÓW W MIKRO SERWISACH

3.2 ZAPOTRZEBOWANIE NA WSZPOWDZIELENIE POSZCZEGULNYCH OBIEKTÓW  
 3.3 KRYTERIUM METOD WSPÓŁDZIELENIA

4 ANALIZA METOD WSPÓWDZIELENIA KODU

4.1 METODY WSPÓWDZIELENIA KODU Z OPISEM

4.2 ANALIZA METOD WSPÓWDZIENIA KODU DLA RÓZNYCH TYPÓW OBIEKTÓW  
 4.3 OCENA METOD WSPÓWDZIENIA KODU

5 KOMBINACJĘ METOD WSPÓWDZIELENIA KODU  
 5.1 ANALIZA MOŻLIWYCH PRZYPADKÓW

5.2 PRZYGOTOWANIE KOMBINACJI METOD DLA POSZCEGULNUCH PRZYPADKÓW

6. PODSUMOWANIE

WYKAZ RYSUNKÓW

WYKAZ LISTINGÓW

WYKAZ TBEL

WYKAZ LITERATURY

Wstęp

1 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.

Page2 /building-microservices-designing-fine-grained-systems definicja mikroserwisu

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.

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.

Celem niniejszej pracy magisterskiej jest zgłębienie tematu współdzielenia obiektów w systemach mikro serwisów 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 mikro serwisowej architektury.

Rodzaje obiektów które mogą być współdzielone w systemie mikro serwisów.

Typy obiektów w mikro serwisach

DTO (Data Transfer Object): DTO to obiekt, który służy do przesyłania danych między warstwami aplikacji lub między klientem a serwerem. DTO zawiera pola, które reprezentują dane, które są wymieniane między stronami. (Strona 265 [Robert Daigneau](https://www.google.pl/search?sca_esv=7db0ce9ab356b080&hl=pl&udm=36&udm=36&q=inauthor:%22Robert+Daigneau%22&sa=X&ved=2ahUKEwj26bKTmfKJAxVdFxAIHUVEBYsQ9Ah6BAgNEAc)

[Service Design Patterns: Fundamental Design Solutions for somain services)](https://books.google.pl/books?id=tK3_vB304bEC&pg=PA318&dq=DTO+(Data+Transfer+Object)+definition&hl=pl&newbks=1&newbks_redir=1&sa=X&ved=2ahUKEwj26bKTmfKJAxVdFxAIHUVEBYsQ6AF6BAgNEAI)

Model: Model to obiekt, który reprezentuje dane w kontekście biznesowym lub domenowym. Modele są używane do przetwarzania danych w warstwie logiki biznesowej i są często mapowane na struktury danych w bazie danych.

Exeption: Obiekty wyjątków są używane do obsługi błędów i wyjątkowych sytuacji w API. Mogą zawierać informacje o błędzie, takie jak kod błędu, wiadomość, szczegóły itp.

Entity: Entity to obiekt, który reprezentuje strukturę danych w bazie danych. Entity są często mapowane na tabele w bazie danych i są używane do przechowywania i pobierania danych.

Walidatory sprawdzają poprawność danych i mogą generować komunikaty o błędach, jeśli dane nie spełniają określonych kryteriów.

Serwisy: Obiekty które zawierają metody które wykonują logikę biznesową.

Jak można ocenić metody współdzielenia mikro serwisów:

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.

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.

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.

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.

Wersjonowanie: Ważne jest również odpowiednie zarządzanie wersjami obiektów, szczególnie w środowisku mikro serwisów, gdzie różne serwisy mogą używać różnych wersji obiektów. Metoda współdzielenia obiektów powinna uwzględniać zarządzanie wersjami i umożliwiać aktualizacje w sposób kontrolowany i bezpieczny.

Metody współdzielenia obiektów w mikro serwisach

1Interface definitione languages – do IDL należą wiele popularnych technologii, Protocol buffers, Avro IDL, Open Api wiki(https://en.wikipedia.org/wiki/Interface\_description\_language)

2 Libraries – definition Programming Techniques page 5 najbardziej oczywiste podejście, daje możliwość współdzielenia wszystkich rodzajów obiektów w aplikacji.   
 2 Client Libraries (SDK) – definition Client-Library migration Guide page 1 dobry artykuł na temat Client libraries w Spring (https://www.baeldung.com/java-microservices-share-dto).

Każde z powyżej opisanych podejść posiada wadę która polega na tym, że przy wprowadzeniu zmian nie wiemy których dokładnie serwisów w naszym systemie dotyczy zmiana. Na przykład posiadamy bibliotekę, w tej bibliotece mamy 2 klasy numer 1 i 2, biblioteka jest używana przez serwis A i B, serwis A używa klasę numer 1 a serwis B używa klasę numer 2. Gdy robimy nową wersje biblioteki wprowadzając zmianę w klasie 1, musimy zmienić wersje biblioteki, ponownie zbudować, przetestować aplikację A, natomiast dla aplikacji B to nie jest konieczne, może działać na produkcyjnym środowisku bez zmian. Oczywiście dobrą praktyką będzie podnieść wersję również dla aplikacji B, ale nie jest to sprawa pilna i można to zrobić później kiedy nawal pac jest mniejszy. Zawsze jest wygodnym znać priorytetowość zadań. Spróbuję znaleźć dostępne na rynku rozwiązanie które zapewni możliwość śledzenia których dokładnie serwisów dotyczy wprowadzona zmiana, albo w przypadku gdy opisanej powyżej funkcyjności nie posiada żadne dostępne rozwiązanie spróbuję zmodyfikować dostępne rozwiązanie tak żeby zapełniało taką możliwość.

**Pryncypia działania:**   
Interface definitione 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 mikro serwisów. 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 mikro serwisów i na podstawie udostępnionych plików wygenerować w każdym z mikro serwisów kod lub obiekty które zostaną użyte przez specyficzną logikę konkretnego serwisu dla osiągniecia konkretnego celu biznesowego. Tworzenie kodu na podstawie plików IDL jest ławo automatyzowane za pomocą narządź 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 cześć innego programy. Program importujący bibliotekę może używać kod biblioteki tak, gdyby to był własny kod programu. Istnieje wiele narządź które wspomagają łatwe i szybkie importowanie i zarządzanie bibliotekami kodu, takie jak Maven czy Gradle. W współczesnych systemach mokro serwisów współdzielenie kodu za pomocą bibliotek kodu odbywa się za pomocą serwerów do przechowywania artefaktów i plików binarnych, takich jak Nexus. (9.2.1 strona 9-5 [Pipeline as Code: Continuous Delivery with Jenkins](https://books.google.pl/books?id=Lt9EEAAAQBAJ&pg=SA9-PA5&dq=nexus+repository&hl=pl&newbks=1&newbks_redir=1&sa=X&ved=2ahUKEwiB9oGlm_KJAxWWExAIHcAZDjoQ6AF6BAgMEAI) [[Mohamed Labouardy](https://books.google.pl/books?id=Lt9EEAAAQBAJ&pg=SA9-PA5&dq=nexus+repository&hl=pl&newbks=1&newbks_redir=1&sa=X&ved=2ahUKEwiB9oGlm_KJAxWWExAIHcAZDjoQ6AF6BAgMEAI)](https://www.google.pl/search?sca_esv=7db0ce9ab356b080&hl=pl&udm=36&udm=36&q=inauthor:%22Mohamed+Labouardy%22&sa=X&ved=2ahUKEwiB9oGlm_KJAxWWExAIHcAZDjoQ9Ah6BAgMEAc)[)](https://books.google.pl/books?id=Lt9EEAAAQBAJ&pg=SA9-PA5&dq=nexus+repository&hl=pl&newbks=1&newbks_redir=1&sa=X&ved=2ahUKEwiB9oGlm_KJAxWWExAIHcAZDjoQ6AF6BAgMEAI)

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 maja dostęp do serwera mogą pobrać bibliotekę z kodem i przechowywać w lokalnym systemie plików używając jako część kodu źródłowego. Biblioteka może być udostępniona nieograniczone 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 przychowywane sąjedynie obiekty które muszą być użyrte do komuitacji z innym progrmem lub kodem, logika biznesowa nie zostaje udostęniona w takim przypadku i zostaje ukryta oraz nie może zostać zmieniona, w przypadku udostępnienia SDK możemy łatwo zapewnić bezpieczeństwo kody(użytkownicy wciągające 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 współdzielenie kodu SDK to – mamy mikro serwis który udostępnia api, api przyjmyje obiekty json które mogą być opisane za pomocą obiektów Java, obiekty które pozostałe mikro serwisy w systemie mikro serwisów mogą użyć do wysłania żądań do określonego wcześniej mikro serwisu udostępniającego api możemy udostępnic dla naszego systemu mikro serwisów jako bibliotekę. Każdy serwis który chcę wysyłać żądania do serwisu restowego może pobrać bibliotekę w postaci dependencji za pomocą narzedzia dla budowania i użyć przygotowane obiekty do komunikacji. Natomiast kod serwisu nie zostanie udostępniony. W ramach prac nad serwisem korzystającym z api restowego nie musimy testować kod api bo on nie został udostępniony i dlatego ni mógł ulec zmianie w trakcie pisania kodu serwisu.

Opis tego, co i w jakich przypadkach możemy współdzielić

Współdzielenie DTO – obiekty DTO możemy współdzielić trzech metod opisanych powyżej, możemy generować obiekty DTO za pomocą IDL, możemy współdzielić obiekty za pomocą bibliotek, możemy również współdzielić obiekty za pomocą SDK.

Współdzielenie obiektów reprezentujących dane z bazy danych - Model. Współdzielenie takich obiektów nie jest zalecane z powodu tego, że powszechnie przyjętą dobrą praktyką jest używanie przez jeden mikro serwis jednej bazy danych z powodu trudności rozwiązania problemów w kodzie i w załażeniu źródła błędu później na etapie utrzymania aplikacji. Teoretycznie jest możliwym współdzielenie za pomocą bibliotek oraz SDK. Ref [Microservices Design Patterns by](https://books.google.pl/books?id=4iIrEQAAQBAJ&pg=PA255&dq=Database+per+Service&hl=pl&newbks=1&newbks_redir=1&sa=X&ved=2ahUKEwjfu8LgtfKJAxUTEBAIHcQIO-wQ6AF6BAgKEAI) [[Aditya Pratap Bhuyan](https://books.google.pl/books?id=4iIrEQAAQBAJ&pg=PA255&dq=Database+per+Service&hl=pl&newbks=1&newbks_redir=1&sa=X&ved=2ahUKEwjfu8LgtfKJAxUTEBAIHcQIO-wQ6AF6BAgKEAI)](https://www.google.pl/search?sca_esv=ead86818698e8678&hl=pl&gl=pl&udm=36&udm=36&q=inauthor:%22Aditya+Pratap+Bhuyan%22&sa=X&ved=2ahUKEwjfu8LgtfKJAxUTEBAIHcQIO-wQ9Ah6BAgKEAc) [strona 255](https://books.google.pl/books?id=4iIrEQAAQBAJ&pg=PA255&dq=Database+per+Service&hl=pl&newbks=1&newbks_redir=1&sa=X&ved=2ahUKEwjfu8LgtfKJAxUTEBAIHcQIO-wQ6AF6BAgKEAI)

Współdzielenie walidatorów, walidatory mogą mieć różną implementację w różnych językach, w większości popularnych językach są zaimplementowane w sposób podobny do serwisów. Walidatory możemy współdzielić za pomocą SDK oraz bibliotek.

Współdzielenie serwisów, serwisy zawierają główna logikę biznesową programu, dlatego zawierają największą ilość kodu aplikacji, dlatego musimy uważać na współdzielenie logiki między mikro serwisami żeby ułatwić naprawienie błędów oraz zwiększyć czytelność oraz zrozumiałość kodu. Narzędzie które możemy używać dla współdzielenia logiki w serwisach to biblioteki oraz SDK.

Analiza metod współdzielenia obiektów

1Interface definitione languages (IDL)

Ocena pod kątem komunikacji – zapewnia potrzebny poziom komunikacji, obiekty są generowane na podstawie defition language na etapie uruchamiania aplikacji, dalej w trakcie wykonania logiki programu takie obiekty mogą być użyte przez program bez żadnych obciążeń wydajnościowych. Obiekty wygenerowane z plików IDL mogą być wykorzystane dla komunikacji asynchronicznej takich jak brokery widomości jak również dla wysłania bezpośrednich zapytań między serwisami.

Metody zarządzania plikami idl w systemach :

1 SVN systemy takie jak, na przykład GIT

Korzyści:

Zapewnia, że ​​wszystkie usługi korzystają z najnowszej wersji IDL.

Łatwiejsze egzekwowanie wersji i wstecznej kompatybilności.

Ułatwia korzystanie z jednego źródła prawdy.

Wyzwania:

Może stać się wąskim gardłem, jeśli wiele zespołów jest zależnych od częstych aktualizacji.

Wymaga silnego zarządzania, aby uniknąć zmian powodujących przerwy.

2 Dystrybucja za pomocą bibliotek – Kompilowanie plików IDL do współdzielonych bibliotek, i publikacja w współdzielonym repozytorium pakietów, na przykład nexus.

Korzyści:

Łatwa integracja z procesami CI/CD.

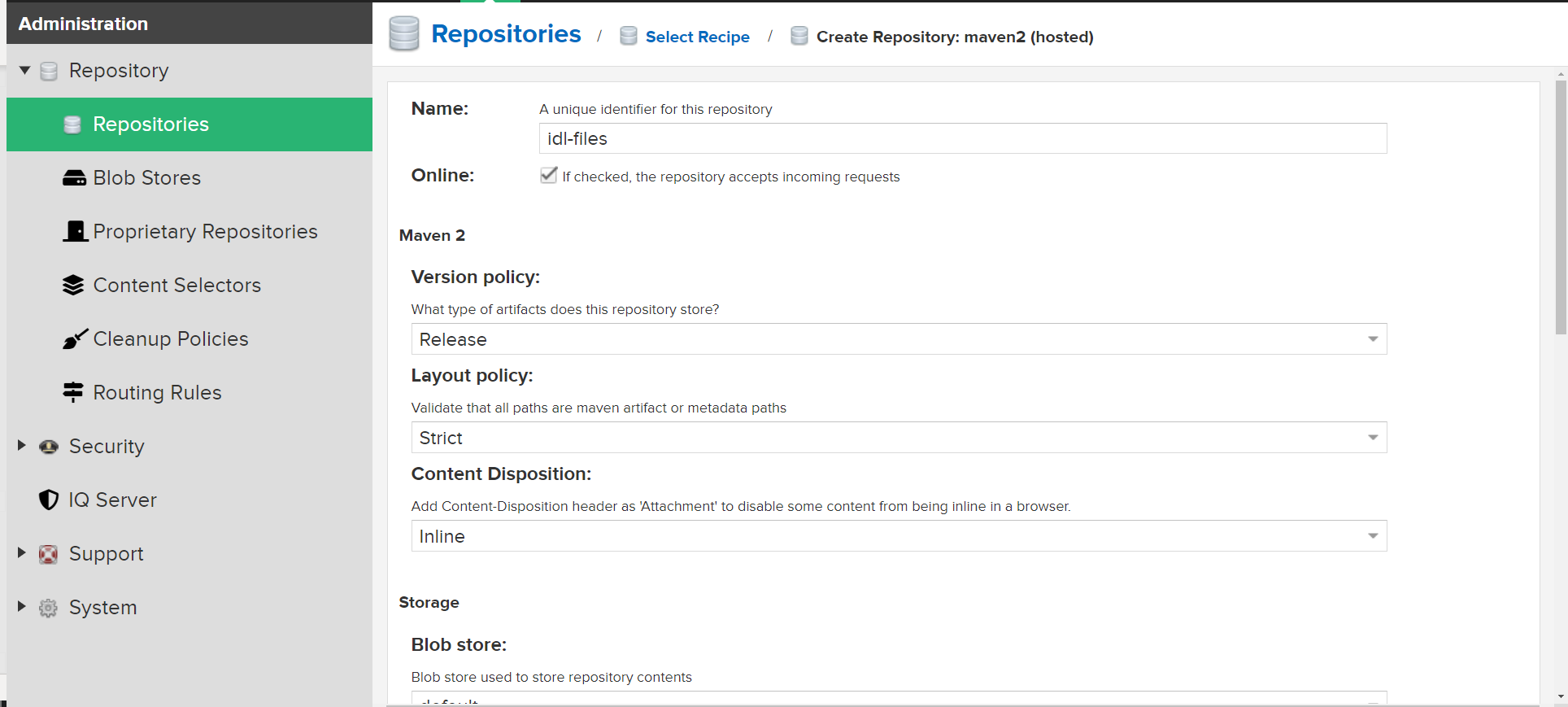
Zespoły mogą korzystać z wersjonowanych artefaktów bez bezpośredniej interakcji z surowymi plikami IDL.

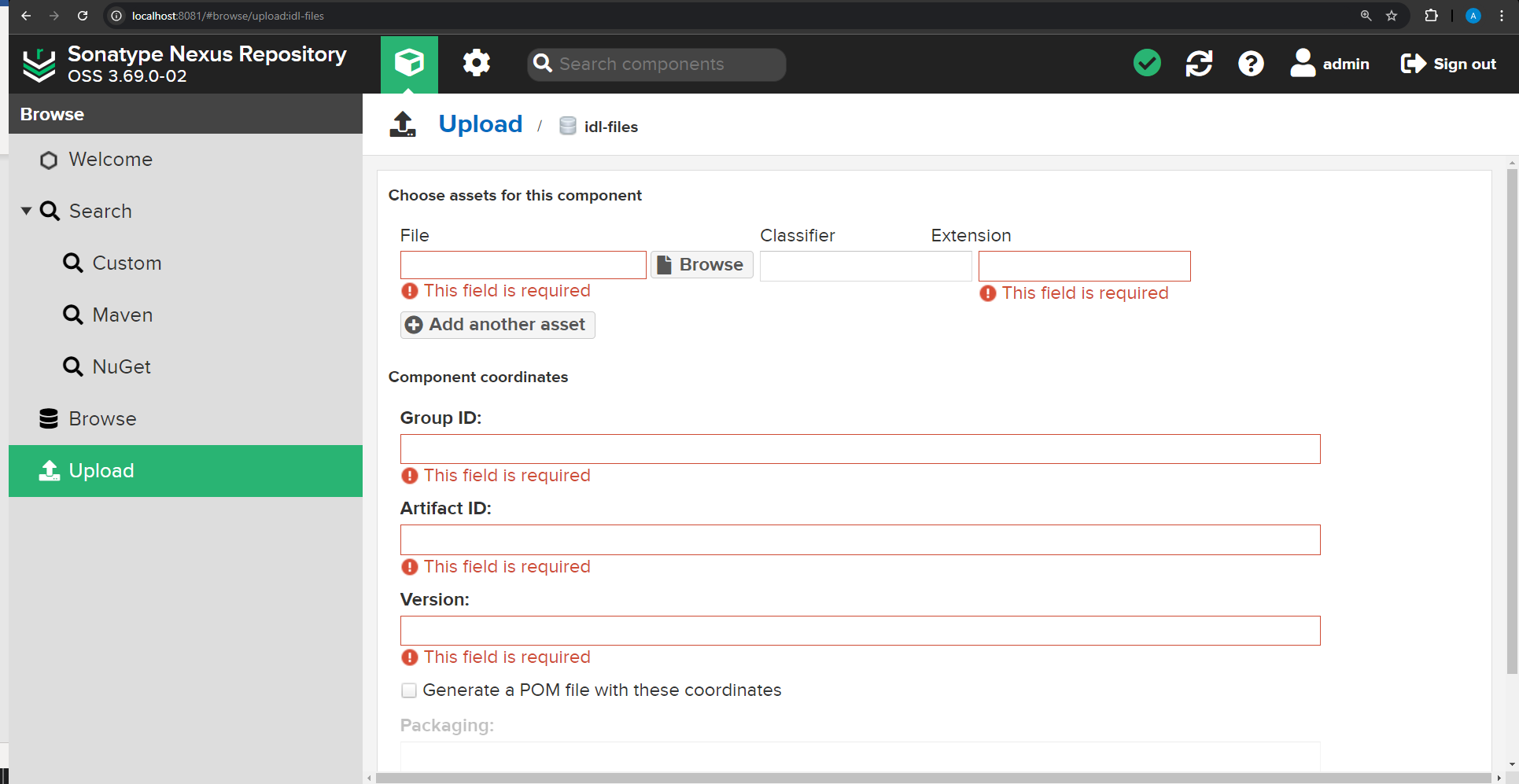
Obsługuje powiązania specyficzne dla języka (np. generowanie klas Java z buforów protokołu).

Wyzwania:

Wymaga automatyzacji do tworzenia i dystrybucji artefaktów.

Wiele języków może wymagać różnych bibliotek, co zwiększa złożoność.





[http://nexus-server:8081/repository/idl-repository/<artifact-group>/<artifact-id>/<version>/<file-name>.idl](http://nexus-server:8081/repository/idl-repository/%3cartifact-group%3e/%3cartifact-id%3e/%3cversion%3e/%3cfile-name%3e.idl)

Podejście Monorepo – Przechowywanie wszystkich mikrserwisów i powiązane z nimi pliki IDL w jednym repozytorium, co zapewni, że wszystkie zespoły będą pracować na tej samej bazie kodu.

Korzyści:

Gwarantuje, że pliki IDL i implementacje usług są zawsze zsynchronizowane.

Uproszcza zarządzanie zależnościami, ponieważ cały kod znajduje się w jednym miejscu.

Wyzwania:

Problemy ze skalowalnością w miarę wzrostu repozytorium.

Wymaga solidnych narzędzi do zarządzania dużymi bazami kodu.

Repozytoria z automatyczną synchronizacją

Korzyści:

Łączy zalety decentralizacji (np. autonomię zespołu) z potrzebą spójności.

Umożliwia dostosowanie do konkretnych usług.

Wyzwania:

Złożoność w zarządzaniu synchronizacją i konfliktami.

Może prowadzić do dryfu wersji, jeśli nie jest starannie zarządzany.

**IDL as a Service (IDLaS) – Przygotowanie usługi, która obsługuje pliki IDL na żądanie. Ta usługa może wersjonować, weryfikować i dostarczać niezbędne pliki IDL na żądanie.**

Korzyści:

Dostęp na żądanie do najnowszych plików IDL.

Możliwość dynamicznego wymuszania wersjonowania i zgodności.

Wyzwania:

Wymaga zbudowania i utrzymania usługi.

Może powodować opóźnienia w dostępie do plików IDL.

Podstawowa implementacja współdzielenia kodu za pomocą idl, przykład przygotowany w ramach pracy znajduje się w kodzie źródłowym dołączonym do pracy.

Przygotowałem projekt który zademonstruje współdzielenie kodu za pomocą idl na przykładzie Języka IDL – „Protobuf” stworzonego przez Google.

Ocena pod katem izolacji – ocena pod kątem izolacji zależy od implementacji, Potrzebne badanie

Ocena pod kątem bezpieczeństwa – Potrzebne badanie

Ocena pod kątem skalowalności – Potrzebne badanie

Ocena pod kątem wersjonowania – Potrzebne badanie

2 Biblioteki (Libraries)

Ocena pod kątem komunikacji – zapewnia potrzebny poziom komunikacji, obiekty są dodawane do projektu przez narzędzie do budowania projektów na etapie uruchamiania aplikacji, dalej w trakcie wykonania logiki programu takie obiekty mogą być użyte przez program bez żadnych obciążeń wydajnościowych. Obiekty wciągnięte z biblioteki mogą być użyte dla komunikacji asynchronicznej takich jak brokery widomości jak również dla wysłania bezpośrednich zapytań między serwisami.

Ocena pod katem izolacji – Potrzebne badanie

Ocena pod kątem bezpieczeństwa – Potrzebne badanie

Ocena pod kątem skalowalności – Potrzebne badanie

Ocena pod kątem wersjonowania – Potrzebne badanie

3 Biblioteki klienckie (SDK)

Ocena pod kątem komunikacji – zapewnia potrzebny poziom komunikacji, mechnizm współdzielenia jest podobny do biblioteki, tak samo jak w przypadku biblioteki obiekty są dodawane do projektu przez narzędzie do budowania projektów na etapie uruchamiania aplikacji, dalej w trakcie wykonania logiki programu takie obiekty mogą być użyte przez program bez żadnych obciążeń wydajnościowych. Obiekty wciągnięte z biblioteki mogą być użyte dla komunikacji asynchronicznej takich jak brokery widomości jak również dla wysłania bezpośrednich zapytań między serwisami.

Ocena pod katem izolacji – Potrzebne badanie

Ocena pod kątem bezpieczeństwa – Potrzebne badanie

Ocena pod kątem skalowalności – Potrzebne badanie

Ocena pod kątem wersjonowania – Potrzebne badanie

Wnioski z analizy metod spółdzielnia obiektów w miro serwisach, pod kątem komunikacji każde rozwiązanie zapewnia odpowiedni poziom komunikacji między serwisami systemu.