



**POLSKO-JAPOŃSKA AKADEMIA
TECHNIK KOMPUTEROWYCH**

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Natalia Janasiewicz

Nr albumu s22847

**Koncepcja systemu monitorowania wielu środowisk
Salesforce w modelu tylko do odczytu o minimalnych
uprawnieniach**

Praca magisterska napisana pod
kierunkiem:

Dr inż. Mariusz Trzaska, prof.
PJATK

Warszawa, lipiec, 2026

Oświadczam, że zapoznałam się z treścią Zarządzenia Rektora Polsko-Japońskiej Akademii Technik Komputerowych z dnia 1 kwietnia 2026 r. oraz z „Wytycznymi dotyczącymi wykorzystywania sztucznej inteligencji w procesie dydaktycznym PJATK”, stanowiącymi załącznik do ww. zarządzenia.

Oświadczam, że ewentualne wykorzystanie narzędzi sztucznej inteligencji (AI) w niniejszej pracy miało charakter wyłącznie pomocniczy i nie stanowiło zastępstwa dla mojej samodzielnej pracy.

Oświadczam, że treść pracy stanowi moje własne, samodzielne opracowanie oraz ponoszę pełną odpowiedzialność za jej zawartość, niezależnie od wykorzystania narzędzi AI.

Oświadczam, że nie wykorzystywałam narzędzi AI do generowania treści stanowiących zasadniczą część pracy, w szczególności analiz, wniosków, interpretacji ani części podlegających ocenie jako przejaw twórczości własnej.

W przypadku korzystania z narzędzi AI sporządziłam Raport użycia AI zgodnie z obowiązującymi Wytycznymi i dołączyłam go do pracy.

Streszczenie

Praca dotyczy problemu monitorowania wielu środowisk platformy Salesforce w sytuacji, w której nadawanie bezpośredniego dostępu do każdego z nich jest niepożądane ze względów bezpieczeństwa oraz kosztów licencyjnych. Problem ten nasila się w modelu dostawcy usług zarządzanych (*Managed Service Provider*), zarządzającego środowiskami należącymi do wielu klientów.

W pracy opracowano koncepcję systemu monitorowania działającego jako zewnętrzny obserwator w trybie wyłącznie odczytowym (*read-only*), o minimalnym zakresie uprawnień i bez trwałego przechowywania danych. Koncepcja obejmuje architekturę zewnętrznego pośrednika, model uwierzytelniania oparty na standardzie JWT OAuth 2.0, agregację danych z wielu środowisk oraz mechanizm wtyczek umożliwiający rozszerzanie zakresu monitorowania bez modyfikacji istniejących komponentów.

Realizowalność koncepcji potwierdzono prototypem złożonym z warstwy serwerowej (Node.js, Express) oraz klienckiej (Vue), zintegrowanym z platformą poprzez REST API oraz Tooling API. Weryfikacja empiryczna na rzeczywistych środowiskach potwierdziła cztery warunki przyjęte w tezie pracy. Wykazano również ograniczenie wynikające z konstrukcji samej platformy: udostępnienie informacji o zainstalowanych pakietach oraz historii wdrożeń wymusza nadanie uprawnień administracyjnych, czego praktycznym rozwiązaniem pozostaje mechanizm przełączników funkcjonalności.

Słowa kluczowe: Managed Service Provider, minimalne uprawnienia, model read-only, monitorowanie środowisk, Salesforce

Podziękowania

Dziękuję promotorowi, dr. hab. Mariuszowi Trzasce, za opiekę nad niniejszą pracą.

Spis treści

WSTĘP	7
1.1. KONTEKST PROBLEMU	7
1.2. PROBLEM BADAWCZY	7
1.3. CEL PRACY	7
1.4. ZAKRES PRACY	8
1.5. REZULTATY PRACY	8
1.6. ORGANIZACJA PRACY	9
2. SALESFORCE I PROBLEM MONITOROWANIA WIELU ŚRODOWISK	10
2.1. PLATFORMA SALESFORCE I ORGANIZACJE	10
2.2. TYPY ŚRODOWISK SALESFORCE	11
2.2.1 Środowisko produkcyjne	11
2.2.2 Sandbox	11
2.2.3 Scratch org	11
2.2.4 Wykorzystanie środowisk w cyklu rozwoju aplikacji	12
2.3. PARAMETRY ŚRODOWISKA ISTOTNE Z PUNKTU WIDZENIA MONITOROWANIA	12
2.3.1 Pakiety	12
2.3.2 Limity systemowe i licencje	13
2.3.3 Metadane i konfiguracja środowiska	13
2.4. INTERFEJSY DOSTĘPU DO INFORMACJI O ŚRODOWISKU	14
2.5. PROBLEMY MONITOROWANIA WIELU ŚRODOWISK SALESFORCE	15
2.5.1 Brak centralizacji informacji	15
2.5.2 Ograniczenia dostępu użytkowników	15
2.5.3 Koszty i ograniczenia licencyjne	15
2.5.4 Bezpieczeństwo środowisk produkcyjnych	15
2.5.5 Brak agregacji danych	15
2.5.6 Model MSP i zarządzanie wieloma orgami	15
2.6. ISTNIEJĄCE PODEJŚCIA DO MONITOROWANIA	16
2.6.1 Monitorowanie manualne i raporty wbudowane	16
2.6.2 Dodatki ze sklepu AppExchange	17
2.6.3 Komercyjne narzędzia typu DevOps	17
2.6.4 Rozwiązania o otwartym kodzie źródłowym	18
2.6.5 Centralizacja danych w dedykowanym orgu	19
2.7. OGRANICZENIA ISTNIEJĄCYCH PODEJŚĆ	19
2.7.1 Brak agregacji wielu środowisk w rozwiązaniach natywnych	19
2.7.2 Koszty licencyjne narzędzi komercyjnych	19
2.7.3 Wymagania infrastrukturalne rozwiązań o otwartym kodzie źródłowym	20
2.7.4 Skupienie na innym celu monitorowania	20
2.7.5 Ograniczenia centralizacji w dedykowanym orgu	20
2.7.6 Wnioski z analizy ograniczeń	20
2.8. WNIOSKI	20
3. KONCEPCJA SYSTEMU MONITOROWANIA ŚRODOWISK SALESFORCE	22
3.1. ZAŁOŻENIA PROJEKTOWE	22
3.1.1 Założenia nadrzędne	22
3.1.2 Założenia drugorzędne	22
3.2. MODEL OBSERWACYJNY TYLKO DO ODCZYTU	23
3.2.1 Charakterystyka modelu	23

3.2.2	<i>Operacje dopuszczone w modelu</i>	23
3.2.3	<i>Konsekwencje przyjętego modelu</i>	24
3.3.	ARCHITEKTURA ZEWNĘTRZNEGO POŚREDNIKA BEZ WARSTWY TRWAŁEGO PRZECHOWYWANIA DANYCH	24
3.3.1	<i>Zewnętrzny pośrednik</i>	24
3.3.2	<i>Brak warstwy trwałego przechowywania danych</i>	24
3.3.3	<i>Konsekwencje przyjętej architektury</i>	25
3.4.	MODEL UWIERZYTELNIANIA I MINIMALIZACJA UPRAWNIENI	25
3.4.1	<i>Wybór mechanizmu uwierzytelniania</i>	25
3.4.2	<i>Użytkownik techniczny i zasada najmniejszych uprawnień</i>	26
3.4.3	<i>Buforowanie tokenów dostępowych</i>	26
3.5.	MECHANIZM WTYCZEK I PRZEŁĄCZNIKÓW FUNKcjONALNOŚCI	26
3.5.1	<i>Motywacja</i>	26
3.5.2	<i>Pojęcie wtyczki</i>	27
3.5.3	<i>Przełączniki funkcjonalności</i>	27
3.5.4	<i>Konsekwencje przyjętego mechanizmu</i>	27
3.6.	AGREGACJA DANYCH Z WIELU ŚRODOWISK	28
3.6.1	<i>Charakterystyka agregacji</i>	28
3.6.2	<i>Zakresy agregacji</i>	28
3.6.3	<i>Konsekwencje wybranego modelu</i>	28
3.7.	PORÓWNANIE KONCEPCJI Z ISTNIEJĄCYMI PODEJŚCIAMI	29
3.7.1	<i>Zestawienie spełnienia założeń</i>	29
3.7.2	<i>Pozycjonowanie koncepcji</i>	29
4.	NARZĘDZIA ZASTOSOWANE W PRACY	30
4.1.	ARCHITEKTURA KLIENT-SERWER I OGÓLNY STOS TECHNOLOGICZNY	30
4.2.	TECHNOLOGIE WARSTWY SERWEROWEJ	31
4.3.	INTEGRACJA Z PLATFORMĄ SALESFORCE	32
4.4.	TECHNOLOGIE WARSTWY KLIENCKIEJ	33
4.5.	KOMUNIKACJA MIĘDZY WARSTWAMI I NARZĘDZIA WSPIERAJĄCE	33
5.	IMPLEMENTACJA PROTOTYPU	35
5.1.	STRUKTURA PROJEKTU	35
5.2.	KONFIGURACJA ŚRODOWISK I JEJ WALIDACJA	37
5.3.	UWIERZYTELNIANIE I KOMUNIKACJA Z PLATFORMĄ SALESFORCE	38
5.4.	MECHANIZM WTYCZEK I PRZEŁĄCZNIKÓW FUNKcjONALNOŚCI	40
5.5.	MODUŁY MONITORUJĄCE I AGREGACJA DANYCH	43
5.6.	WARSTWA KLIENCKA	44
5.7.	PREZENTACJA DANYCH W INTERFEJSIE	45
6.	WERYFIKACJA EMPIRYCZNA	49
6.1.	METODYKA WERYFIKACJI	49
6.2.	KONFIGURACJA ŚRODOWISKA TESTOWEGO	49
6.3.	WERYFIKACJA MODELU ODCZYTOWEGO I MINIMALNYCH UPRAWNIENI	52
6.4.	WERYFIKACJA AGREGACJI WIELU ŚRODOWISK	55
6.5.	WERYFIKACJA PRZEŁĄCZNIKÓW I BRAKU TRWAŁEGO PRZECHOWYWANIA DANYCH	56
6.6.	WNIOSKI	58
7.	PODSUMOWANIE	60
7.1.	PODSUMOWANIE PRACY	60
7.2.	WNIOSKI	60
7.3.	KIERUNKI DAŁSZEGO ROZWOJU	60
8.	BIBLIOGRAFIA	62
9.	DODATKI	64

SŁOWNIK UŻYTEJ TERMINOLOGII I SKRÓTÓW	64
SPIS RYSUNKÓW	65
SPIS TABEL.....	66
SPIS LISTINGÓW	66

Wstęp

Niniejszy rozdział wprowadza w tematykę pracy. Przedstawiono w nim kontekst problemu monitorowania wielu środowisk Salesforce, sformułowano problem badawczy oraz cel pracy, a także określono jej zakres i spodziewane rezultaty. Ostatni podrozdział omawia organizację dalszej części tekstu.

1.1. Kontekst problemu

Organizacje korzystające z platformy Salesforce zazwyczaj nie ograniczają się do jednego środowiska. Standardem stało się utrzymywanie środowiska produkcyjnego oraz kilku instancji pomocniczych, takich jak sandboxy deweloperskie, przedprodukcyjne i testowe. Każde z nich posiada własną konfigurację, zestaw zainstalowanych pakietów, przydzielone licencje użytkowników oraz indywidualne limity systemowe.

W przypadku pojedynczego środowiska dostęp do informacji o jego stanie nie stanowi istotnego problemu. Trudność pojawia się, gdy organizacja zarządza coraz większą liczbą orgów. Uzyskanie aktualnych informacji o konfiguracji każdego z nich wymaga bezpośredniego dostępu, co wiąże się z:

- koniecznością tworzenia kont użytkowników w każdym środowisku,
- przyznawaniem licencji dostępowych,
- kontrolowaniem ryzyka operacyjnego związanego z szerokimi uprawnieniami.

Problem nasila się w modelu Dostawcy Usług Zarządzanych (*Managed Service Provider - MSP*), w którym jedna organizacja zarządza wieloma środowiskami należącymi do różnych klientów. W takiej sytuacji przyznawanie dostępu do środowisk produkcyjnych osobom odpowiedzialnym wyłącznie za monitorowanie ich stanu lub tworzenie rozwiązań generuje dodatkowe ryzyko bezpieczeństwa oraz nieuzasadnione koszty licencyjne.

1.2. Problem badawczy

Problem badawczy niniejszej pracy dotyczy sposobu pozyskiwania informacji diagnostycznych o środowiskach Salesforce w sytuacjach, w których nadawanie bezpośredniego dostępu do każdego z nich jest niepożądane ze względów bezpieczeństwa lub kosztów licencyjnych. Problem obejmuje monitorowanie takich informacji jak zainstalowane pakiety, limity systemowe oraz pozostałe parametry konfiguracyjne, bez konieczności logowania się do każdego środowiska osobno i przez wielu użytkowników.

1.3. Cel pracy

Celem pracy jest opracowanie koncepcji systemu monitorowania wielu środowisk Salesforce, działającego w modelu zewnętrznego obserwatora w trybie tylko do odczytu, który pozwala na dostęp do informacji diagnostycznych bez nadawania bezpośredniego dostępu do każdego środowiska. Koncepcja obejmuje model architektury, sposób uwierzytelniania o minimalnym zakresie uprawnień oraz mechanizm wtyczek umożliwiający rozszerzanie zakresu monitorowania.

Przyjęto założenie, że możliwe jest opracowanie lekkiego systemu monitorowania środowisk Salesforce, który:

- działa zewnątrz wobec monitorowanych orgów w trybie tylko do odczytu, bez instalowania komponentów po stronie monitorowanych środowisk;
- agreguje aktualny stan wielu orgów w pojedynczym interfejsie użytkownika;
- umożliwia rozszerzanie funkcjonalności poprzez mechanizm wtyczek, niezależnie od pozostałych modułów;
- nie wymaga warstwy trwałego przechowywania danych ani infrastruktury ciągła integracja / ciągłe dostarczanie (*continuous integration / continuous delivery*, CI/CD) po stronie systemu monitorującego.

Taka konfiguracja nie jest realizowana w komplecie przez żadne z rozwiązań analizowanych w rozdziale drugim. Weryfikację praktycznej realizowalności przyjętych założeń stanowi prototyp implementujący wybrane moduły monitorujące.

1.4. Zakres Pracy

Zakres pracy obejmuje:

- analizę istniejących metod monitorowania środowisk Salesforce,
- opracowanie modelu architektury zewnętrznego systemu obserwacyjnego,
- zaprojektowanie mechanizmu rozszerzeń w postaci wtyczek funkcjonalnych,
- implementację prototypu w celu weryfikacji koncepcji,
- przeprowadzenie oceny rozwiązania w środowiskach testowych.

Praca nie obejmuje wdrożenia produkcyjnego systemu, pełnej dokumentacji wdrożeniowej dla zespołów *Managed Service Provider* ani implementacji wszystkich możliwych modułów monitorujących - prototyp realizuje wybrane wtyczki demonstrujące wykonalność koncepcji.

1.5. Rezultaty pracy

Głównym rezultatem pracy jest koncepcja systemu monitorowania wielu środowisk Salesforce oparta na modelu zewnętrznego obserwatora działającego w trybie odczytu. Koncepcja określa architekturę rozwiązania, model uwierzytelniania o minimalnym zakresie uprawnień oraz mechanizm wtyczek pozwalający na rozszerzanie zakresu monitorowania bez modyfikacji istniejących komponentów.

Realizowalność koncepcji potwierdza prototyp, który udostępnia w pojedynczym interfejsie informacje pochodzące z wielu środowisk równocześnie, umożliwiając przeglądanie limitów systemowych, licencji użytkowników, zainstalowanych pakietów oraz historii wdrożeń, a także porównywanie tych parametrów pomiędzy wybranymi środowiskami. W odróżnieniu od dostępnych narzędzi monitorujących rozwiązanie nie wymaga instalowania komponentów w monitorowanych środowiskach, nie zajmuje licencji użytkowników ani nie wymaga konfiguracji infrastruktury ciągła integracja / ciągłe dostarczanie.

1.6. Organizacja pracy

Praca składa się z siedmiu rozdziałów. Rozdział pierwszy stanowi wstęp i wprowadza w tematykę pracy, formułując problem badawczy oraz cel pracy. Rozdział drugi zawiera analizę platformy Salesforce w kontekście monitorowania wielu środowisk: omówiono w nim typy środowisk, parametry istotne z punktu widzenia monitorowania oraz dostępne interfejsy API, a także przedstawiono istniejące podejścia do monitorowania wraz z ich ograniczeniami. Rozdział trzeci przedstawia koncepcję proponowanego systemu, w tym założenia projektowe, model obserwacyjny w trybie tylko do odczytu, architekturę zewnętrznego pośrednika bez warstwy trwałego przechowywania danych, model uwierzytelniania oraz mechanizm wtyczek pozwalający na rozszerzanie funkcjonalności. Rozdział czwarty omawia narzędzia i technologie wykorzystane do realizacji prototypu, a rozdział piąty opisuje samą implementację, w tym strukturę projektu oraz najważniejsze moduły funkcjonalne. Rozdział szósty przedstawia empiryczną weryfikację prototypu na rzeczywistych środowiskach Salesforce, sprawdzającą cztery warunki przyjęte w tezie pracy. Rozdział siódmy podsumowuje pracę, odnosi jej wyniki do tezy oraz wskazuje kierunki dalszego rozwoju.

2. Salesforce i problem monitorowania wielu środowisk

Rozdział omawia platformę Salesforce w kontekście monitorowania wielu środowisk. W kolejnych podrozdziałach przedstawiono podstawowe pojęcia platformy, typy dostępnych środowisk, parametry istotne z punktu widzenia monitorowania, dostępne interfejsy programistyczne oraz problemy związane z monitorowaniem wielu orgów. W końcowych podrozdziałach omówiono istniejące podejścia do monitorowania oraz ich ograniczenia, stanowiące motywację do opracowania koncepcji przedstawionej w rozdziale trzecim.

2.1. Platforma Salesforce i organizacje

Platforma Salesforce jest jednym z głównych rozwiązań w obszarze systemów zarządzania relacjami z klientami (*Customer Relationship Management*, CRM), wspierających zarządzanie relacjami z klientami oraz integrację procesów biznesowych w ramach pojedynczego rozwiązania chmurowego. Salesforce, wprowadzony na rynek w 1999 roku, dostarczany jest w modelu oprogramowanie jako usługa (*Software as a Service*, SaaS), w którym aplikacja udostępniana jest przez dostawcę poprzez przeglądarkę internetową, oraz w modelu platforma jako usługa (*Platform as a Service*, PaaS) umożliwiającym budowę własnych aplikacji na infrastrukturze dostawcy [1][2]. Dzięki temu organizacje uzyskują dostęp zarówno do gotowych aplikacji biznesowych w obszarach sprzedaży, marketingu i obsługi klienta, jak i do narzędzi pozwalających na implementację własnych rozwiązań, w tym pakietów dostępnych w sklepie AppExchange [3].

Każda implementacja platformy funkcjonuje w ramach odrębnego środowiska, określanego jako **org** (organizacja).

Definicja 1

Organizacja Salesforce (*org*) stanowi odrębne środowisko platformy Salesforce, zawierające własną konfigurację, użytkowników, dane biznesowe, metadane oraz zainstalowane komponenty [4]. □

Poszczególne orgi nie współdzielą konfiguracji ani danych. W zależności od potrzeb oraz typu środowiska możliwe jest tworzenie kopii konfiguracji pomiędzy orgami. W praktyce organizacje korzystające z platformy utrzymują więcej niż jedno środowisko - standardem jest istnienie środowiska produkcyjnego oraz kilku środowisk pomocniczych wykorzystywanych do prac deweloperskich oraz testowania zmian [5].

Konfiguracja organizacji obejmuje między innymi ustawienia systemowe, użytkowników wraz z ich licencjami, metadane oraz zainstalowane pakiety. Licencje użytkowników określają dostępne funkcjonalności oraz ich maksymalną liczbę, natomiast limity systemowe definiują maksymalne wykorzystanie zasobów platformy, takich jak liczba wywołań API czy dostępna przestrzeń danych. Wartości limitów oraz licencji różnią się pomiędzy orgami w zależności od typu środowiska oraz posiadanych dodatków [6][7][8].

Istotnym elementem konfiguracji są również metadane. Opisują one strukturę aplikacji, w tym obiekty oraz ich pola - nie stanowią danych biznesowych, lecz definiują sposób działania aplikacji. Komponenty metadanych mogą być grupowane w pakiety, które umożliwiają przenoszenie funkcjonalności pomiędzy środowiskami Salesforce [8].

Ze względu na odrębność orgów oraz różnice w konfiguracji każde środowisko wymaga niezależnego zarządzania i monitorowania. Szczególne znaczenie ma kontrolowanie limitów systemowych, zainstalowanych pakietów oraz licencji. W przypadku większej liczby środowisk uzyskanie tych informacji wiąże się z koniecznością dostępu do każdego orga osobno, co prowadzi do potrzeby opracowania rozwiązania umożliwiającego centralne monitorowanie wielu instancji platformy.

2.2. Typy środowisk Salesforce

Platforma Salesforce udostępnia różne rodzaje środowisk, pełniących odmienne role w procesie rozwoju i utrzymania aplikacji. W zależności od potrzeb organizacje korzystają ze środowiska produkcyjnego, środowisk pomocniczych typu *sandbox* oraz krótkotrwałych środowisk *scratch org*. Każdy z tych typów posiada inne przeznaczenie, konfigurację oraz sposób wykorzystania. Kolejne podpodrozdziały omawiają charakterystykę poszczególnych środowisk oraz typowy model ich wykorzystania w cyklu wytwarzania oprogramowania.

2.2.1 Środowisko produkcyjne

Środowisko produkcyjne (*Production Org*) jest główną instancją wykorzystywaną przez użytkowników końcowych organizacji. Zawiera rzeczywiste dane biznesowe, aktywną konfigurację systemu oraz zainstalowane aplikacje wykorzystywane w codziennej pracy. Wprowadzanie zmian bezpośrednio na środowisku produkcyjnym wiąże się z ryzykiem - niesprawdzone modyfikacje mogą wpłynąć negatywnie na działanie systemu, a w skrajnych przypadkach doprowadzić do awarii. Z tego powodu większość organizacji wprowadza zmiany w pierwszej kolejności na środowiskach pomocniczych, a dopiero po przeprowadzeniu testów wdraża je na produkcji.

Środowisko produkcyjne może również pełnić rolę środowiska typu DevHub. Jest to funkcjonalność platformy Salesforce DX, umożliwiająca zarządzanie procesem tworzenia aplikacji oraz pakietami. Po jej włączeniu możliwe staje się tworzenie środowisk typu *scratch org* oraz definiowanie pakietów [10].

2.2.2 Sandbox

Środowisko typu sandbox stanowi odizolowaną kopię produkcji, wykorzystywaną do budowania funkcjonalności, testowania zmian oraz walidacji nowych konfiguracji. Dzięki izolacji umożliwia wprowadzanie modyfikacji bez ryzyka wpływu na dane produkcyjne. Środowiska typu sandbox różnią się między sobą stopniem podobieństwa do produkcji, w zależności od wybranego wariantu [8]:

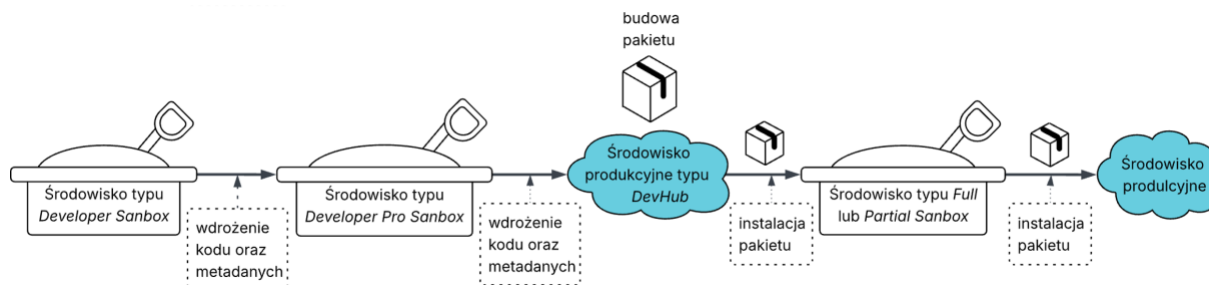
- **Full Copy Sandbox** - wariant najbliższy środowisku produkcyjnemu. Zawiera pełną kopię konfiguracji oraz wszystkich danych produkcyjnych. Wykorzystywany jest do testów integracyjnych, akceptacyjnych oraz wydajnościowych.
- **Partial Copy Sandbox** - zawiera pełną konfigurację środowiska produkcyjnego, lecz jedynie wybraną część danych. Stosowany jest do testów funkcjonalnych oraz integracyjnych wymagających danych testowych. Wraz z *Full Copy* może pełnić funkcję środowiska przedprodukcyjnego w cyklu rozwoju aplikacji.
- **Developer Pro Sandbox** - zawiera konfigurację środowiska produkcyjnego, lecz tylko część danych. Wykorzystywany jest zazwyczaj do testów zespołowych oraz walidacji większych zmian funkcjonalnych.
- **Developer Sandbox** - najprostszy i najczęściej używany wariant. Stosowany jest głównie do pracy deweloperskiej. Zawiera konfigurację środowiska produkcyjnego bez danych biznesowych. Każdy deweloper lub zespół może posiadać własny *Developer Sandbox* w celu implementacji i testowania nowych funkcjonalności.

2.2.3 Scratch org

Środowisko typu *scratch org* jest krótkotrwałym środowiskiem tworzonym przy użyciu DevHuba. Wykorzystywane jest głównie w procesie ciągłej integracji (*continuous integration*) oraz testów automatycznych. Tworzone jest na podstawie predefiniowanej konfiguracji i może zostać usunięte natychmiast po zakończeniu testów. Ze względu na tymczasowy charakter środowiska tego typu nie są wykorzystywane do długoterminowego monitorowania [11].

2.2.4 Wykorzystanie środowisk w cyklu rozwoju aplikacji

Organizacje stosują różne konfiguracje połączeń pomiędzy środowiskami. Popularny model cyklu wytwarzania oprogramowania przedstawiono na Rysunek 1.



Rysunek 1. Cykl wytwarzania oprogramowania z wykorzystaniem środowisk Salesforce.
Źródło: opracowanie własne.

Rysunek 1 przedstawia model procesu wdrażania zmian w Salesforce. Indywidualni deweloperzy pracują początkowo na środowiskach niższego rzędu typu *Developer Sandbox*. Następnie przygotowane funkcjonalności oraz kolejne zmiany, które nie są jeszcze częścią pakietu, są przenoszone na środowiska zespołowe typu *Developer Pro Sandbox* poprzez wdrożenie kodu i metadanych.

Po zakończeniu prac deweloperskich oraz weryfikacji zmian tworzony jest pakiet przy użyciu DevHuba. Pakiet ten jest następnie instalowany na środowisku przedprodukcyjnym typu *Full Sandbox* lub *Partial Sandbox*, gdzie przeprowadzane są testy końcowe. Ostatni etap stanowi instalacja pakietu na środowisku produkcyjnym.

2.3. Parametry środowiska istotne z punktu widzenia monitorowania

Jak omówiono w poprzednim podrozdziale, środowiska Salesforce różnią się konfiguracją, zainstalowanymi pakietami, dostępnymi limitami oraz strukturą metadanych. W przypadku organizacji utrzymujących wiele środowisk istotne staje się monitorowanie kluczowych parametrów, dzięki któremu możliwa jest ocena stanu systemu, identyfikacja różnic pomiędzy środowiskami oraz wdrażanie scenariuszy reaktywnych w przypadku wykrycia nieprawidłowości. Do najważniejszych parametrów należą zainstalowane pakiety wraz z ich wersjami, limity systemowe, licencje użytkowników oraz konfiguracja metadanych.

2.3.1 Pakiety

Pakiety (*packages*) stanowią kontenery zawierające powiązane ze sobą komponenty, aplikacje oraz metadane. Wykorzystywane są do dystrybucji funkcjonalności pomiędzy orgami Salesforce - pakiet nie jest sam w sobie funkcjonalnością, lecz sposobem jej dostarczenia. Platforma udostępnia kilka rodzajów pakietów [12][13][14]:

- *Managed Packages* - pakiety zarządzane przez producenta, w których komponenty są chronione przed modyfikacją po stronie organizacji instalującej. Stosowane są głównie do dystrybucji rozwiązań komercyjnych dostępnych w sklepie AppExchange.
- *Unmanaged Packages* - pakiety udostępniające komponenty w sposób otwarty, umożliwiające ich modyfikację po instalacji w docelowym orgu. Wykorzystywane są zwykle do dystrybucji szablonów lub przykładowych implementacji.

- *Unlocked Packages* - pakiety stosowane najczęściej do dystrybucji rozwiązań wewnętrznych, łączące zalety zarządzania wersjami pakietów *managed* z elastycznością modyfikacji typową dla pakietów *unmanaged*.

Każdy pakiet posiada zestaw wersji, odpowiadających kolejnym iteracjom jego rozwoju. Monitorowanie wersji pakietów zainstalowanych na poszczególnych środowiskach umożliwia identyfikację różnic pomiędzy orgami oraz kontrolę procesu wdrażania funkcjonalności.

2.3.2 Limity systemowe i licencje

Platforma Salesforce definiuje wiele typów limitów systemowych, określających maksymalne wykorzystanie zasobów na danym orgu. Wartości limitów różnią się w zależności od typu środowiska, posiadanych funkcjonalności i licencji oraz zainstalowanych dodatków. Do najważniejszych kategorii limitów należą:

- limity zarządcze (*governor limits*) - limity dotyczące wykonania kodu w języku Apex,
- limity wywołań API,
- limity przestrzeni danych,
- limity liczby obiektów i pól,
- limity automatyzacji i procesów.

Governor limits definiują maksymalną liczbę operacji wykonywanych w pojedynczej transakcji. Obejmują między innymi liczbę zapytań SOQL, operacji DML oraz wykorzystanie pamięci [6]. Limity API dzielą się na limity czasu, jednoczesnych żądań oraz całkowitej liczby żądań w przyjętym okresie. Z perspektywy monitorowania szczególne znaczenie ma kontrolowanie tych ostatnich, ponieważ ich przekroczenie blokuje komunikację z platformą za pośrednictwem interfejsów programistycznych.

Istotnym parametrem są również licencje użytkowników. Określają one dostępne funkcjonalności oraz maksymalną liczbę użytkowników w środowisku. W przypadku organizacji wykorzystujących *DevHub* dodatkowo występują limity związane z liczbą wytworzonych pakietów oraz środowisk typu *scratch org*.

2.3.3 Metadane i konfiguracja środowiska

Metadane opisują strukturę aplikacji oraz konfigurację systemu. Obejmują między innymi definicje obiektów, pól, automatyzacji, uprawnień oraz komponentów interfejsu użytkownika. W przeciwieństwie do danych biznesowych metadane definiują sposób działania aplikacji, nie zaś jej zawartość. Do typowych komponentów metadanych zalicza się:

- obiekty niestandardowe (*Custom Objects*),
- pola niestandardowe (*Custom Fields*),
- przepływy (*Flows*),
- klasy Apex (*Apex Classes*),
- zestawy uprawnień (*Permission Sets*),
- profile użytkowników (*Profiles*),
- reguły walidacji (*Validation Rules*).

Obiekty niestandardowe rozszerzają standardowy model danych platformy. Każdy obiekt zawiera zestaw pól, relacji oraz reguł walidacji. Monitorowanie warstwy metadanych umożliwia identyfikację różnic strukturalnych pomiędzy orgami, takich jak różna liczba obiektów niestandardowych czy odmienne zestawy reguł walidacji.

2.4. Interfejsy dostępu do informacji o środowisku

Platforma Salesforce udostępnia szereg interfejsów programistycznych (API), umożliwiających dostęp do danych, metadanych oraz konfiguracji środowiska bez korzystania z interfejsu użytkownika. Dzięki temu możliwe jest tworzenie narzędzi zewnętrznych, w tym rozwiązań monitorujących stan orgów. Spośród dostępnych interfejsów do najczęściej wykorzystywanych w zastosowaniach diagnostycznych należą REST API, Tooling API oraz Metadata API, a także język zapytań SOQL wykorzystywany do pobierania danych z obiektów platformy.

REST API stanowi usługę internetową opartą na architekturze *REST*, umożliwiającą wykonywanie operacji na danych oraz wybranych elementach konfiguracji. Za pomocą metod protokołu HTTP możliwe jest tworzenie, odczyt, aktualizacja oraz usuwanie rekordów, a także pobieranie informacji o limitach systemowych za pośrednictwem dedykowanych zasobów [15].

Rozszerzeniem podstawowego interfejsu jest **Bulk API**, pozwalające na asynchroniczne operacje na dużych zbiorach danych. Wykorzystywane jest głównie do importu i eksportu dużych wolumenów rekordów, a w kontekście monitorowania jego znaczenie pozostaje ograniczone.

Interfejs **Pub/Sub API** umożliwia dostęp do informacji o zdarzeniach w czasie rzeczywistym na zasadzie modelu subskrypcyjnego. Pozwala na ograniczenie liczby wykonywanych zapytań poprzez reagowanie na zmiany danych zamiast okresowego ich pobierania.

Tooling API udostępnia informacje o metadanych związanych z procesem rozwoju aplikacji. Umożliwia odczyt danych dotyczących klas Apex, triggerów, testów oraz innych komponentów deweloperskich, a także udostępnia obiekty administracyjne, takie jak `DeployRequest` czy `SandboxInfo`, niedostępne za pośrednictwem standardowego REST API [16].

Metadata API umożliwia dostęp do konfiguracji środowiska na poziomie metadanych. Pozwala na pobieranie definicji obiektów, pól oraz pozostałych komponentów konfiguracyjnych. Wykorzystywane jest między innymi w procesie wdrażania zmian pomiędzy środowiskami [8].

Język zapytań **SOQL** (*Salesforce Object Query Language*) służy do pobierania danych z obiektów platformy Salesforce. Składnia SOQL przypomina język SQL, umożliwiając filtrowanie oraz łączenie danych z powiązanych obiektów. W kontekście monitorowania SOQL wykorzystywany jest do odczytu informacji z obiektów takich jak `InstalledSubscriberPackage`, `UserLicense` czy `Organization`.

Zestawienie omówionych interfejsów wraz z ich zastosowaniem i zakresem dostępnych danych przedstawiono w Tabeli 1.

Tabela 1. Interfejsy programistyczne platformy Salesforce.

Interfejs	Zastosowanie	Zakres danych
REST API	Operacje na danych i podstawowej konfiguracji	Rekordy, limity, użytkownicy
Bulk API	Operacje na dużych zbiorach danych	Import i eksport danych
Pub/Sub API	Dane w czasie rzeczywistym	Zdarzenia i zmiany danych
Tooling API	Informacje deweloperskie i administracyjne	Klasy Apex, testy, komponenty, wdrożenia, sandboxy
Metadata API	Konfiguracja środowiska	Obiekty, pola, automatyzacje
SOQL	Zapytania do danych	Rekordy obiektów

2.5. Problemy monitorowania wielu środowisk Salesforce

Jak omówiono w poprzednich podrozdziałach, organizacje korzystające z platformy Salesforce utrzymują wiele środowisk pełniących różne role w procesie rozwoju aplikacji, przy czym każde z nich posiada własną konfigurację, własny zestaw zainstalowanych pakietów oraz indywidualne limity systemowe. Taka architektura, mimo iż uzasadniona z punktu widzenia bezpieczeństwa i organizacji pracy, prowadzi do szeregu trudności w obszarze monitorowania stanu środowisk. Trudności te mają charakter zarówno techniczny, jak i organizacyjny.

2.5.1 Brak centralizacji informacji

Informacje o zainstalowanych pakietach, limitach systemowych oraz konfiguracji muszą być pobierane osobno dla każdego środowiska. Oznacza to konieczność monitorowania każdego orga osobno, bez względu na liczbę utrzymywanych instancji. Dodatkowo znaczenie poszczególnych parametrów może różnić się w zależności od typu środowiska - na przykład limity systemowe na środowisku produkcyjnym mają większą wagę niż na środowisku deweloperskim, gdzie ich okresowe przekroczenie nie wpływa na pracę użytkowników końcowych.

2.5.2 Ograniczenia dostępu użytkowników

Uzyskanie informacji o stanie środowiska, zarówno za pośrednictwem interfejsu użytkownika, jak i interfejsów programistycznych, wymaga aktywnego konta użytkownika oraz odpowiednich uprawnień. W praktyce oznacza to konieczność utrzymywania konta na każdym monitorowanym orgu. Sytuacja ta dotyczy szczególnie środowisk produkcyjnych - przykładowo deweloperzy potrzebują dostępu do informacji o aktualnej wersji pakietu zainstalowanej na produkcji, lecz nie powinni otrzymywać pełnego dostępu do danych biznesowych.

2.5.3 Koszty i ograniczenia licencyjne

Każdy aktywny użytkownik platformy Salesforce wymaga przypisanej licencji. Wraz ze wzrostem liczby użytkowników rośnie liczba wymaganych licencji oraz związany z tym koszt. Licencje są zasobem ograniczonym, a zwiększenie ich liczby wiąże się z istotnymi nakładami finansowymi, w związku z czym muszą być zarządzane w sposób efektywny oraz stale monitorowane, aby nie generowały kosztów nieuzasadnionych z punktu widzenia rzeczywistych potrzeb operacyjnych.

2.5.4 Bezpieczeństwo środowisk produkcyjnych

Środowiska produkcyjne zawierają rzeczywiste dane biznesowe, w tym dane wrażliwe klientów. Dostęp do takich środowisk powinien być nadawany wyłącznie użytkownikom posiadającym uzasadnioną potrzebę biznesową. Deweloperzy wymagają dostępu w celu odczytu wersji pakietów oraz aktualnych limitów, jednak ze względu na zasadę rozdziału obowiązków nie powinni dysponować uprawnieniami do danych klientów. Standardowy model dostępu do orga nie umożliwia jednoczesnego spełnienia obu warunków bez konfigurowania dedykowanych profili lub zestawów uprawnień, co znacząco zwiększa złożoność administracyjną.

2.5.5 Brak agregacji danych

Standardowe narzędzia platformy umożliwiają analizę danych wyłącznie w obrębie pojedynczego orga. Brakuje wbudowanego mechanizmu pozwalającego na agregację danych z wielu środowisk w jednym widoku. W rezultacie porównanie konfiguracji lub stanu różnych orgów wymaga ręcznego zbierania danych z każdej instancji osobno, co wydłuża czas analizy oraz zwiększa ryzyko błędów wynikających z niespójności pomiędzy źródłami.

2.5.6 Model MSP i zarządzanie wieloma orgami

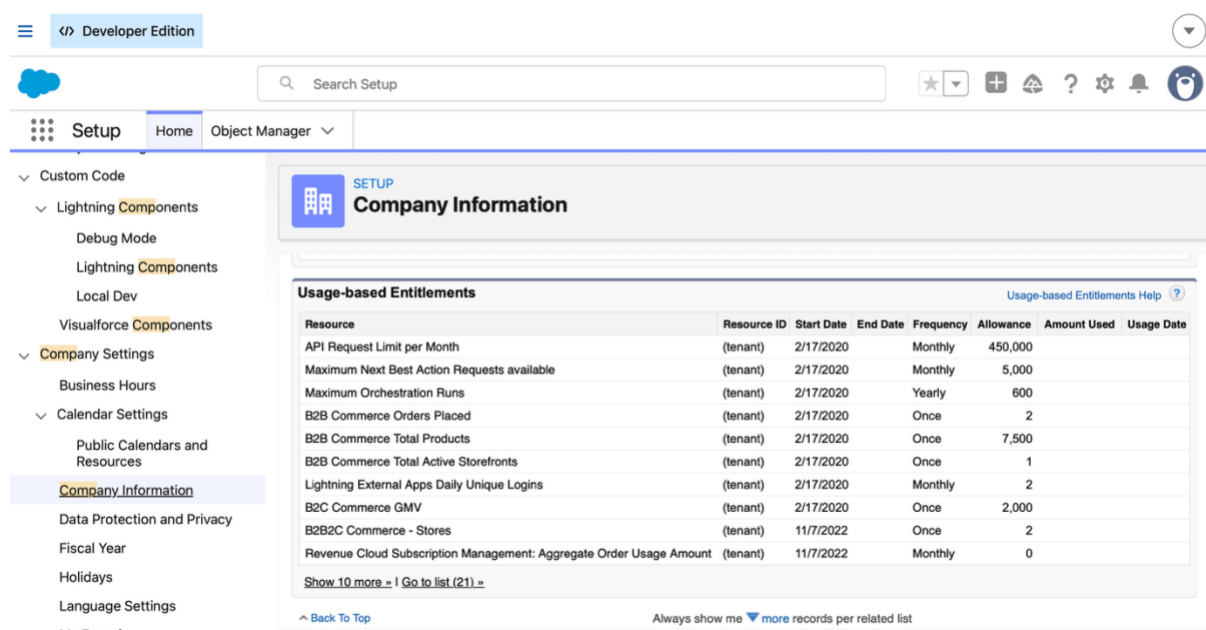
Opisane problemy ulegają zwielokrotnieniu w modelu *Managed Service Provider*. W przypadku organizacji zarządzającej środowiskami należącymi do wielu klientów liczba orgów znacząco rośnie, a dostęp do nich jest dodatkowo ograniczony ze względów bezpieczeństwa oraz wymogów kontraktowych. Monitorowanie wielu środowisk różnych klientów przy jednoczesnym zachowaniu zasady minimalnych uprawnień stanowi w tym modelu istotny problem zarówno techniczny, jak i organizacyjny.

2.6. Istniejące podejścia do monitorowania

Podrozdział omawia istniejące sposoby monitorowania środowisk Salesforce. Przedstawiono kolejno podejścia natywne udostępniane przez platformę, dodatki dostępne w sklepie AppExchange, zewnętrzne narzędzia komercyjne typu DevOps oraz rozwiązania o otwartym kodzie źródłowym (*open-source*). Każde z podejść scharakteryzowano pod kątem zakresu monitorowania oraz wymaganego sposobu integracji ze środowiskami Salesforce.

2.6.1 Monitorowanie manualne i raporty wbudowane

Najprostsze podejście do monitorowania polega na ręcznym sprawdzaniu konfiguracji przez administratora lub wyznaczonego użytkownika. W takim modelu osoba odpowiedzialna loguje się do orga i analizuje wybrane informacje, takie jak zainstalowane pakiety, przydzielone licencje czy aktualne wykorzystanie limitów (Rysunek 2). Podejście to jest reaktywne - informacje o problemach trafiają do administratora dopiero w momencie zalogowania.



The screenshot shows the Salesforce Setup interface. The left sidebar contains a navigation menu with categories like 'Custom Code', 'Lightning Components', 'Visualforce Components', 'Company Settings', and 'Calendar Settings'. The 'Company Information' link is highlighted. The main content area displays the 'Usage-based Entitlements' table, which lists various resources and their usage limits.

Resource	Resource ID	Start Date	End Date	Frequency	Allowance	Amount Used	Usage Date
API Request Limit per Month	(tenant)	2/17/2020		Monthly	450,000		
Maximum Next Best Action Requests available	(tenant)	2/17/2020		Monthly	5,000		
Maximum Orchestration Runs	(tenant)	2/17/2020		Yearly	600		
B2B Commerce Orders Placed	(tenant)	2/17/2020		Once	2		
B2B Commerce Total Products	(tenant)	2/17/2020		Once	7,500		
B2B Commerce Total Active Storefronts	(tenant)	2/17/2020		Once	1		
Lightning External Apps Daily Unique Logins	(tenant)	2/17/2020		Monthly	2		
B2C Commerce GMV	(tenant)	2/17/2020		Once	2,000		
B2B2C Commerce - Stores	(tenant)	11/7/2022		Once	2		
Revenue Cloud Subscription Management: Aggregate Order Usage Amount	(tenant)	11/7/2022		Monthly	0		

Rysunek 2. Ręczne sprawdzenie wykorzystania limitów w interfejsie platformy Salesforce.

Źródło: opracowanie własne

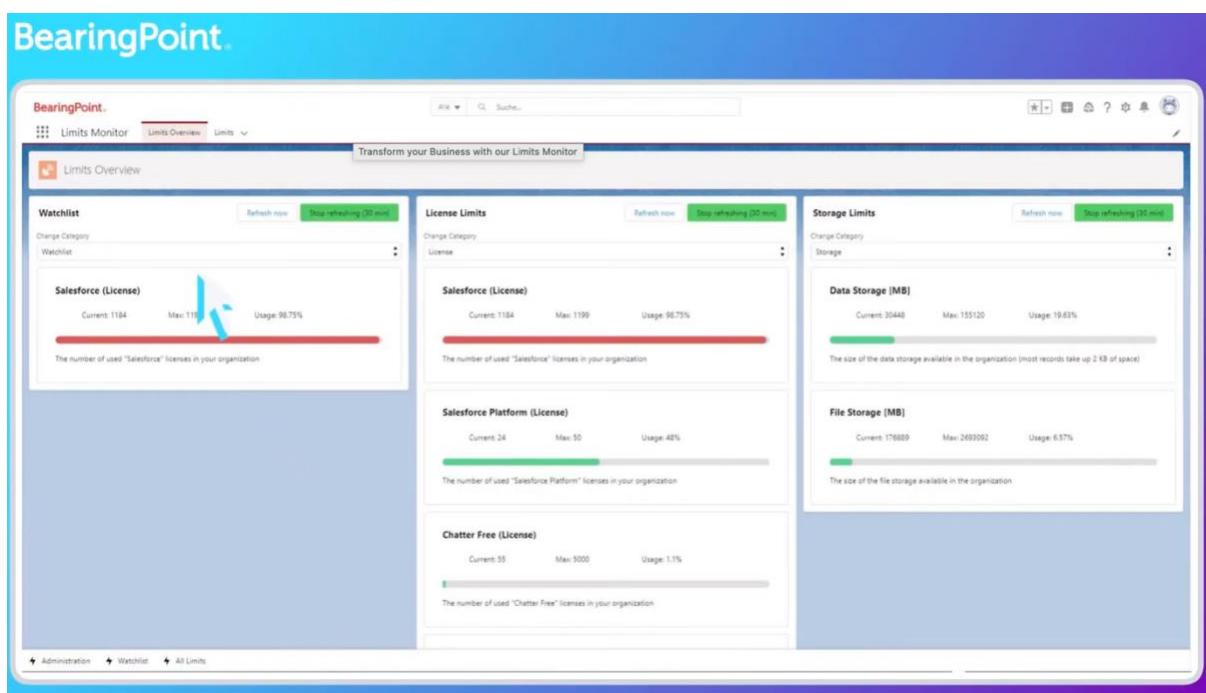
Alternatywę dla działań manualnych stanowią raporty i pulpity menedżerskie tworzone w obrębie orga. Standardowy model raportowania w platformie Salesforce ogranicza się jednak do danych zapisanych w obiektach. Limity systemowe i większość parametrów istotnych z punktu widzenia monitorowania nie są przechowywane jako rekordy obiektów, w związku z czym przygotowanie pulpitu

wymaga implementacji dodatkowych mechanizmów zapisujących te wartości - najczęściej okresowych zadań Apex zapisujących migawki (*snapshots*) do obiektów niestandardowych [17].

2.6.2 Dodatki ze sklepu AppExchange

W sklepie AppExchange dostępne są pakiety dedykowane monitorowaniu wybranych aspektów środowiska. Do reprezentatywnych przykładów należą BearingPoint Limits Monitor [18] (Rysunek 3) oraz OrgLimit Monitor [19]. Aplikacje tego typu udostępniają pulpity z bieżącym wykorzystaniem limitów, możliwość definiowania progów alarmowych oraz powiadomienia mailowe.

Wspólną cechą tych rozwiązań jest instalacja w postaci pakietu po stronie monitorowanego orga. Oznacza to, że każde monitorowane środowisko wymaga osobnej instalacji oraz konfiguracji, a wszystkie aplikacje tego typu działają w obrębie pojedynczego orga - nie udostępniają widoku agregującego dane z wielu środowisk.



Rysunek 3. Pulpit monitorowania limitów w aplikacji BearingPoint Limits Monitor ze sklepu AppExchange. Źródło: [18].

2.6.3 Komercyjne narzędzia typu DevOps

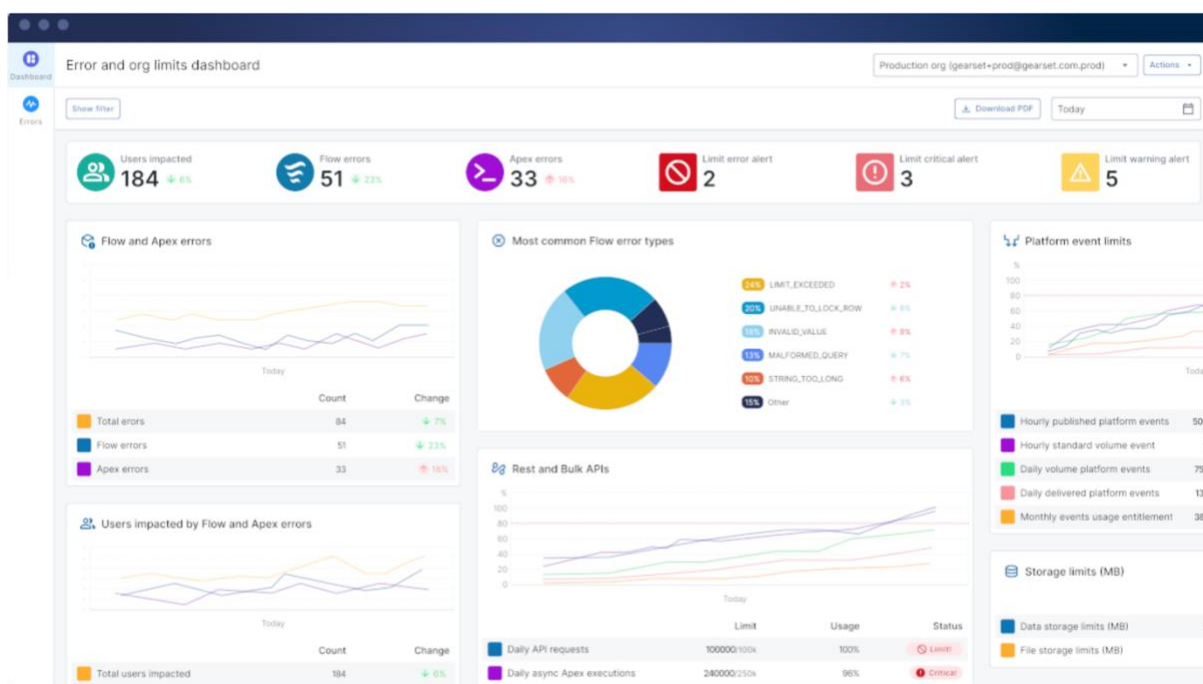
Na rynku dostępne są komercyjne platformy DevOps wspierające zarządzanie środowiskami Salesforce, do których należą Gearset [21], Copado [22] oraz Flosum [23]. Narzędzia te koncentrują się przede wszystkim na procesie wdrażania zmian, wersjonowaniu metadanych oraz porównywaniu konfiguracji środowisk, lecz część z nich obejmuje również funkcjonalności monitorujące.

Gearset udostępnia moduł obserwowalności (*observability*) obejmujący monitorowanie błędów Flow i Apex, wykorzystania limitów API, przestrzeni danych oraz limitów (Rysunek 4) wiadomości e-mail. Komunikacja z orgami realizowana jest za pośrednictwem REST API, bez konieczności instalowania komponentów po stronie monitorowanych środowisk. Standardowa konfiguracja umożliwia monitorowanie maksymalnie dwóch orgów - rozszerzenie tej liczby wiąże się z dodatkowymi opłatami licencyjnymi [21].

Copado oraz **Flosum** realizują podobną funkcjonalność w ramach szerszych platform DevOps. Ich głównym przeznaczeniem pozostaje jednak zarządzanie procesem wdrożeń, a monitorowanie stanowi w nich element uzupełniający, nie zaś funkcję podstawową.

Aplikacja **Metazoa Snapshot** [24] reprezentuje odmienne podejście - pracuje jako aplikacja instalowana lokalnie, wykonująca okresowe migawki metadanych każdego z monitorowanych orgów oraz umożliwiającą ich porównywanie. Narzędzie skupia się na audycie zmian konfiguracyjnych w czasie, nie zaś na monitorowaniu bieżącego stanu operacyjnego środowiska.

Każde z wymienionych rozwiązań komercyjnych wiąże się z modelem licencjonowania zależnym od liczby użytkowników lub monitorowanych orgów, co w przypadku organizacji utrzymujących większą liczbę środowisk generuje znaczące koszty.

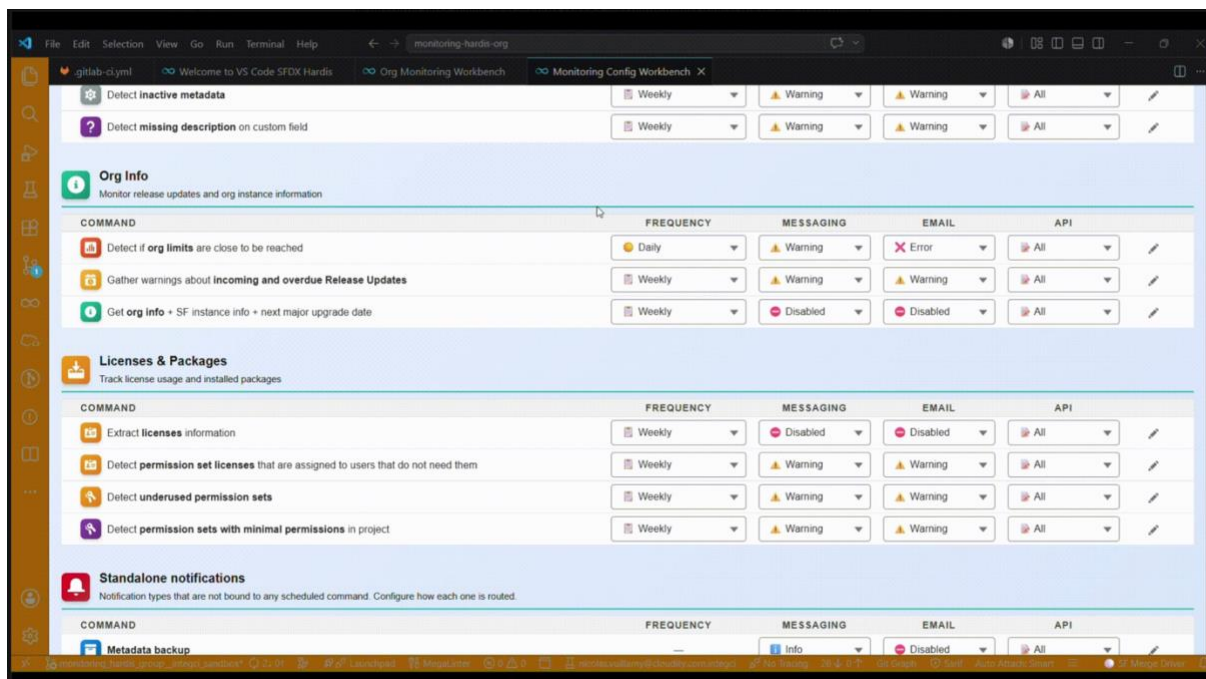


Rysunek 4. Pulpit modułu "Error and org limits dashboard" platformy Gearset. Źródło: [21].

2.6.4 Rozwiązania o otwartym kodzie źródłowym

Najbardziej rozwiniętym rozwiązaniem *open-source* w obszarze monitorowania środowisk Salesforce jest **sfdx-hardis** [25] (Rysunek 5). Projekt udostępnia zestaw komend uruchamianych w ramach potoku ciągłej integracji (GitHub Actions, GitLab CI, Azure Pipelines), wykonujących między innymi okresową kopię metadanych, wykrywanie zmian konfiguracyjnych, raportowanie nieużywanych licencji oraz audyt podejrzanych operacji administracyjnych. Wyniki kolejnych przebiegów zapisywane są w repozytorium Git, co umożliwia porównywanie konfiguracji w czasie. Powiadomienia o zmianach wysyłane są na kanały Slack lub Microsoft Teams, a dane mogą być wizualizowane w zewnętrznym systemie Grafana [25].

Sfdx-hardis wykorzystuje uwierzytelnianie JWT OAuth do każdego monitorowanego orga oraz nie wymaga instalowania komponentów po stronie monitorowanych środowisk, co czyni je najbliższym koncepcyjnie z dotychczas wymienionych narzędzi. Jednocześnie wymaga repozytorium Git, pipeline'u ciągłej integracji oraz dedykowanej konfiguracji dla każdego orga, w tym osobnej gałęzi Git dla każdego monitorowanego środowiska oraz certyfikatów SSL generowanych przy użyciu zewnętrznego narzędzia OpenSSL [25].



Rysunek 5. "Monitoring config workbench" z widokiem na "Org Info" oraz "Licenses & Packages" w narzędziu sfdx-hardis. Źródło: [25].

2.6.5 Centralizacja danych w dedykowanym orgu

Odmienne podejście polega na utworzeniu dedykowanego orga Salesforce pełniącego funkcję centralnego punktu monitorowania. W takim modelu zewnętrzne środowisko łączy się z monitorowanymi orgami przy użyciu mechanizmów *Salesforce Connect* lub okresowych zadań Apex, pobierając wybrane informacje i prezentując je w pulpitych utworzonych przy użyciu wbudowanych narzędzi analitycznych. Rozwiązanie tego typu wymaga utrzymywania dodatkowej licencji oraz konfigurowania połączeń pomiędzy orgami, których liczba na pojedynczym orgu jest ograniczona [26].

2.7. Ograniczenia istniejących podejść

Analiza istniejących podejść omówionych w poprzednim podrozdziale pozwala wskazać szereg ograniczeń, które uzasadniają potrzebę opracowania alternatywnego rozwiązania. Ograniczenia te dotyczą zarówno mechanizmów udostępnianych przez platformę Salesforce, jak i narzędzi zewnętrznych - różnią się jednak charakterem i źródłem.

2.7.1 Brak agregacji wielu środowisk w rozwiązaniach natywnych

Mechanizmy natywne platformy - raporty, pulpity oraz pakiety instalowane w orgu - operują wyłącznie w obrębie pojedynczego środowiska. Próba uzyskania widoku porównawczego wymaga ręcznego logowania się do każdego orga i samodzielnego zestawiania danych. Pakiety dostępne w sklepie AppExchange, mimo że udostępniają pulpity dedykowane monitorowaniu, dziedziczą to ograniczenie - każde monitorowane środowisko wymaga niezależnej instalacji, konfiguracji oraz aktywnego użytkownika z odpowiednią licencją.

2.7.2 Koszty licencyjne narzędzi komercyjnych

Komercyjne platformy DevOps, takie jak Gearset, Copado czy Flosum, wiążą się z opłatami zależnymi od liczby użytkowników, liczby monitorowanych orgów lub zakresu funkcjonalności. W przypadku Gearset moduł *observability* w standardowej konfiguracji obejmuje monitorowanie maksymalnie dwóch orgów, a rozszerzenie tej liczby wymaga indywidualnych ustaleń z dostawcą [21]. Dla organizacji utrzymujących większą liczbę środowisk, w szczególności działających w modelu *Managed Service Provider*, model licencyjny tego typu stanowi istotną barierę kosztową.

2.7.3 Wymagania infrastrukturalne rozwiązań o otwartym kodzie źródłowym

Rozwiązania *open-source*, w szczególności *sfdx-hardis*, eliminują koszt licencyjny, lecz wprowadzają wymagania infrastrukturalne. Konfiguracja monitoringu wymaga repozytorium Git, działającego pipeline'u ciągłej integracji, znajomości narzędzi linii komend *Salesforce DX* oraz utrzymywania osobnej gałęzi konfiguracyjnej dla każdego monitorowanego orga [25]. Próg wejścia tego typu narzędzi pozostaje wysoki - z dokumentacji projektu wynika konieczność wyznaczenia dedykowanej osoby pełniącej rolę kierownika wydań (*release manager*), posiadającej kompetencje w obszarze Git i *Salesforce DX* [25].

2.7.4 Skupienie na innym celu monitorowania

Większość omówionych narzędzi koncentruje się na monitorowaniu zmian zachodzących w środowiskach - porównaniu konfiguracji w czasie, wykrywaniu różnic metadanych, raportowaniu nowych wdrożeń. Podejście to dobrze odpowiada na pytanie „co zmieniło się od poprzedniego pomiaru”, lecz mniej skutecznie na pytanie „w jakim stanie znajduje się środowisko w chwili obecnej”. Operacyjne monitorowanie bieżącego stanu - wykorzystania limitów, alokacji licencji, aktualnych wersji zainstalowanych pakietów - wymaga w istniejących rozwiązaniach komercyjnych dopłat za dodatkowe moduły, a w rozwiązaniach z otwartym kodem źródłowym dodatkowej konfiguracji i integracji z systemem wizualizacji typu Grafana.

2.7.5 Ograniczenia centralizacji w dedykowanym orgu

Podejście polegające na centralizacji danych w dedykowanym orgu Salesforce wprowadza wszystkie ograniczenia samej platformy: limit liczby zewnętrznych połączeń, koszt utrzymywania dodatkowej organizacji oraz konieczność zapewnienia licencji dla użytkowników operujących na środowisku monitorującym. Co istotne, rozwiązanie tego typu nie eliminuje konieczności konfigurowania uwierzytelnienia osobno dla każdego orga ani problemu związanego z licencjonowaniem dostępu.

2.7.6 Wnioski z analizy ograniczeń

Każde z analizowanych podejść spełnia część założeń sformułowanych w założeniach pracy (podrozdział 1.3), żadne jednak nie spełnia ich łącznie. Rozwiązania natywne i pakiety AppExchange nie umożliwiają agregacji wielu środowisk. Komercyjne narzędzia DevOps eliminują problem agregacji, lecz wprowadzają koszt licencyjny zależny od skali. *Sfdx-hardis* eliminuje koszt licencyjny i problem agregacji, lecz wymaga rozbudowanej infrastruktury ciągłej integracji oraz warstwy trwałego przechowywania danych w postaci repozytorium Git. Centralizacja w dedykowanym orgu rozwiązuje problem widoku zbiorczego, ale wprowadza ograniczenia samej platformy.

Wskazane ograniczenia stanowią motywację do opracowania koncepcji systemu omówionej w rozdziale trzecim.

2.8. Wnioski

Przedstawiona analiza prowadzi do wniosku, że każda organizacja Salesforce stanowi odrębne środowisko, posiadające własną konfigurację, metadane, licencje użytkowników oraz limity

systemowe. W praktyce organizacje utrzymują wiele takich środowisk, pełniących różne role w cyklu rozwoju i utrzymania aplikacji, co prowadzi do potrzeby ich systematycznego monitorowania.

Platforma Salesforce udostępnia zestaw interfejsów programistycznych, w tym REST API, Tooling API oraz Metadata API, pozwalających na odczyt informacji o środowisku bez korzystania z interfejsu użytkownika. Interfejsy te stanowią techniczną podstawę dla budowy rozwiązań zewnętrznych.

Analiza istniejących podejść do monitorowania wskazuje, że żadne z dostępnych rozwiązań nie realizuje w komplecie założeń sformułowanych w tezie pracy. Rozwiązania natywne i pakiety dystrybuowane przez sklep AppExchange nie obsługują agregacji wielu środowisk. Komercyjne narzędzia DevOps wiążą się z modelem licencyjnym ograniczającym skalę monitorowania. Rozwiązanie `sfdx-hardis`, najbliższe koncepcyjnie pod względem architektury, wymaga rozbudowanej infrastruktury ciągłej integracji oraz warstwy trwałego przechowywania danych w postaci repozytorium Git. Centralizacja danych w dedykowanym orgu wprowadza dodatkowe ograniczenia samej platformy.

W rezultacie istnieje przestrzeń projektowa dla lekkiego, bezstanowego systemu monitorowania wielu środowisk, działającego w trybie tylko do odczytu, niewymagającego instalacji komponentów po stronie monitorowanych orgów ani infrastruktury ciągłej integracji po stronie systemu monitorującego. Koncepcja takiego systemu, wraz z jego założeniami projektowymi oraz architekturą, stanowi przedmiot kolejnego rozdziału.

3. Koncepcja systemu monitorowania środowisk Salesforce

Rozdział przedstawia koncepcję systemu monitorowania środowisk Salesforce stanowiącą zasadniczy rezultat pracy. W kolejnych podrozdziałach omówiono założenia projektowe, model obserwacyjny tylko do odczytu, architekturę zewnętrznego pośrednika bez warstwy trwałego przechowywania danych, model uwierzytelniania i minimalizacji uprawnień, mechanizm wtyczek oraz sposób agregacji danych pochodzących z wielu środowisk. Rozdział zamyka porównanie zaproponowanej koncepcji z istniejącymi podejściami omówionymi w rozdziale drugim.

3.1. Założenia projektowe

Założenia projektowe systemu wynikają z analizy ograniczeń istniejących podejść (podrozdział 2.7) oraz celu pracy (podrozdział 1.3). Każde z nich adresuje konkretną wadę dostępnych narzędzi, którą zidentyfikowano w trakcie analizy stanu sztuki, oraz wpisuje się w jeden z warunków sformułowanych w celu. Trzy założenia uznano za nadrzędne, ponieważ definiują charakter systemu i wpływają na pozostałe decyzje projektowe - opisano je szerzej w dalszej części podrozdziału. Pozostałe założenia uznano za drugorzędne, mimo iż są również istotne; przedstawiono je w formie listy, a ich szczegółowe omówienie znajduje się w kolejnych podrozdziałach.

3.1.1 Założenia nadrzędne

Prostota wdrożenia. System ma być możliwy do uruchomienia bez wielogodzinnej konfiguracji infrastruktury towarzyszącej. Jak omówiono w podrozdziale 2.7.3, rozwiązania o otwartym kodzie źródłowym typu `sfdx-hardis` wymagają repozytorium Git, działającego *pipeline'u* ciągłej integracji znajomości narzędzi linii komend Salesforce DX oraz dedykowanej osoby pełniącej rolę kierownika wydań. Tak wysoki próg wejścia stanowi istotną barierę dla zespołów, których głównym celem jest jedynie monitorowanie stanu środowisk, nie zaś pełna automatyzacja procesu wdrażania zmian. Proponowane rozwiązanie sprowadza konfigurację do dwóch czynności po stronie administratora: utworzenia plików konfiguracyjnych ze zdefiniowanymi środowiskami oraz wygenerowania kluczy uwierzytelniających dla użytkownika technicznego.

Niski koszt utrzymania. System nie wykorzystuje komercyjnego modelu licencyjnego ani zależnego od liczby monitorowanych orgów. Komercyjne narzędzia DevOps omówione w podrozdziale 2.7.2 wprowadzają opłaty proporcjonalne do skali - co w przypadku organizacji utrzymujących większą liczbę środowisk, w szczególności działających w modelu *Managed Service Provider*, stanowi istotną barierę kosztową. Proponowane rozwiązanie wykorzystuje pojedynczego użytkownika technicznego po stronie każdego monitorowanego orga, a po stronie systemu monitorującego nie wymaga dodatkowych licencji ani opłat zewnętrznych.

Tryb tylko do odczytu. System operuje wyłącznie w trybie odczytu - nie modyfikuje danych ani konfiguracji monitorowanych orgów. Każda operacja wykonywana przez system to wywołanie REST API lub Tooling API w metodzie GET, względnie zapytanie SOQL. Brak operacji zapisujących eliminuje ryzyko niezamierzonego wpływu systemu monitorującego na pracę monitorowanego środowiska. Założenie to wpływa również na model uwierzytelniania omówiony w podrozdziale 3.4 - pozwala na wykorzystanie użytkownika technicznego o ograniczonych uprawnieniach, bez konieczności nadawania uprawnień modyfikacji.

3.1.2 Założenia drugorzędne

- **Modularność.** System udostępnia mechanizm wtyczek pozwalający na rozszerzanie zakresu monitorowania o kolejne parametry bez modyfikacji istniejących komponentów. Założenie szczegółowo omówiono w podrozdziale 3.5.
- **Agregacja widoku.** Dane pochodzące z wielu środowisk prezentowane są w pojedynczym interfejsie, umożliwiającym ich porównanie oraz analizę różnic. Założenie omówiono w podrozdziale 3.6.
- **Bezpieczeństwo komunikacji.** Komunikacja z monitorowanymi orgami realizowana jest przy użyciu mechanizmu uwierzytelniania JWT OAuth 2.0, eliminującego konieczność przechowywania haseł użytkownika technicznego. Klucze prywatne pozostają wyłącznie po stronie systemu monitorującego. Założenie omówiono w podrozdziale 3.4.
- **Brak warstwy trwałego przechowywania danych.** System nie utrzymuje lokalnej bazy danych - informacje pobierane są na bieżąco z monitorowanych środowisk. Założenie eliminuje problem synchronizacji oraz redukuje wymagania infrastrukturalne. Założenie omówiono w podrozdziale 3.3.

Łączne spełnienie powyższych założeń stanowi istotę proponowanego rozwiązania. Pojedyncze założenia są spełniane przez wybrane istniejące narzędzia - żadne z nich nie realizuje jednak wszystkich łącznie. Szczegółowe porównanie z istniejącymi podejściami zamieszczono w podrozdziale 3.7.

3.2. Model obserwacyjny tylko do odczytu

Centralnym założeniem koncepcji jest oparcie systemu na modelu obserwacyjnym - system jedynie odczytuje informacje o stanie monitorowanych środowisk, nie wprowadzając w nich żadnych zmian. Założenie to wpływa na sposób interakcji z platformą Salesforce, model uprawnień użytkownika technicznego oraz architekturę całego rozwiązania.

3.2.1 Charakterystyka modelu

W proponowanym modelu monitorowane środowisko Salesforce pełni rolę źródła obserwacji, a system monitorujący - rolę obserwatora. Wszystkie operacje wykonywane przez obserwatora ograniczone są do odczytu danych konfiguracyjnych oraz diagnostycznych za pośrednictwem interfejsów programistycznych. System monitorujący nie wykonuje operacji modyfikujących, nie tworzy obiektów ani rekordów w monitorowanym środowisku, nie instaluje komponentów oraz nie zmienia konfiguracji platformy.

Model obserwacyjny stoi w odróżnieniu od podejść opartych na komponentach instalowanych w monitorowanym środowisku, omówionych w podrozdziale 2.6.2. Pakiety dystrybuowane przez sklep AppExchange, mimo iż służą celom monitorującym, wymagają wprowadzenia zmian w konfiguracji każdego monitorowanego orga - instalacji pakietu, przydzielenia licencji oraz konfiguracji powiadomień. Model obserwacyjny eliminuje tę konieczność.

3.2.2 Operacje dopuszczone w modelu

System wykonuje wyłącznie operacje należące do trzech kategorii:

- żądania HTTP metodą GET kierowane do interfejsów REST API i Tooling API platformy Salesforce,
- zapytania SOQL odczytujące rekordy ze standardowych obiektów platformy, w tym Organization, UserLicense, InstalledSubscriberPackage, DeployRequest oraz SandboxInfo,
- wywołania zasobu /limits zwracającego bieżące wykorzystanie limitów systemowych.

Wszystkie wymienione operacje są jednoznacznie bezpieczne z punktu widzenia integralności monitorowanego środowiska - nie wpływają na dane biznesowe ani konfigurację. Brak operacji modyfikujących pozwala na deklaratywne ograniczenie zakresu działań systemu, co stanowi punkt wyjścia dla minimalizacji uprawnień użytkownika technicznego omówionej w podrozdziale 3.4.

3.2.3 Konsekwencje przyjętego modelu

Przyjęcie modelu obserwacyjnego pociąga za sobą konkretne konsekwencje projektowe. Pierwszą z nich jest brak konieczności obsługi transakcji oraz mechanizmów odporności na awarie po stronie systemu monitorującego - żadna z wykonywanych operacji nie wprowadza zmian wymagających odwołania. Drugą konsekwencją jest możliwość uruchamiania systemu w środowiskach o ograniczonych uprawnieniach administracyjnych, ponieważ system nie wymaga praw do modyfikacji danych. Trzecią konsekwencją, omówioną szczegółowo w podrozdziale 3.4, jest zgodność z zasadą najmniejszych uprawnień (*principle of least privilege*) - użytkownik techniczny po stronie monitorowanego orga otrzymuje wyłącznie uprawnienia odczytu, nie obejmujące modyfikacji danych ani konfiguracji.

Model obserwacyjny tylko do odczytu jest niezbędnym, lecz niewystarczającym warunkiem bezpiecznego monitorowania środowisk. Sam zapis programowy gwarantuje wyłącznie, że system nie wykonuje operacji modyfikujących z poziomu kodu - nie zastępuje jednak konfiguracji uprawnień po stronie monitorowanego orga. W ostatecznej konfiguracji bezpieczeństwa oba mechanizmy działają wspólnie: kod systemu nie wykonuje operacji modyfikujących, a uprawnienia użytkownika technicznego po stronie Salesforce uniemożliwiają takie operacje nawet w razie błędu implementacyjnego.

3.3. Architektura zewnętrznego pośrednika bez warstwy trwałego przechowywania danych

Drugim założeniem koncepcji jest umieszczenie logiki monitorowania w zewnętrznym pośredniku działającym poza monitorowanymi środowiskami, bez utrzymywania lokalnej bazy danych.

3.3.1 Zewnętrzny pośrednik

System składa się z dwóch warstw: warstwy klienckiej (interfejsu użytkownika) oraz warstwy serwerowej (pośrednika). Pośrednik realizuje funkcje pobierania danych z monitorowanych orgów, ich przetwarzania oraz udostępniania warstwie klienckiej za pośrednictwem własnego interfejsu REST. Warstwa kliencka nie komunikuje się bezpośrednio z platformą Salesforce - każde żądanie jest kierowane do warstwy serwerowej, która wykonuje odpowiednie wywołania API i zwraca wynik w ustalonym formacie.

Umieszczenie logiki monitorującej poza monitorowanymi środowiskami stanowi konsekwencję założenia z podrozdziału 3.2. Skoro system nie modyfikuje stanu monitorowanego orga, nie ma uzasadnienia, aby działał w jego obrębie. Wynika z tego również, że pojedynczy pośrednik może obsługiwać dowolną liczbę monitorowanych środowisk - liczba ta jest ograniczona wyłącznie przepustowością samego pośrednika oraz limitami API po stronie monitorowanego orga.

3.3.2 Brak warstwy trwałego przechowywania danych

System nie utrzymuje lokalnej bazy danych. Informacje o stanie środowisk pobierane są bezpośrednio z platformy Salesforce w momencie każdego żądania użytkownika, bez pośredniego zapisu do bazy danych lub repozytorium. Założenie to różnicuje proponowane rozwiązanie od podejść

omówionych w podrozdziale 2.6.4, w szczególności od `sfdx-hardis`, który zapisuje kolejne migawki metadanych do repozytorium Git w celu wykrywania zmian w czasie.

Wybór modelu bezstanowego oparto na obserwacji, że dane diagnostyczne pobierane z platformy Salesforce - wykorzystanie limitów, alokacja licencji, lista zainstalowanych pakietów - są same w sobie aktualnym stanem orga. Salesforce pełni rolę systemu źródłowego (*system of record*) dla tych informacji. Zapisywanie kopii lokalnej nie wnosi w tym przypadku wartości dodanej - przeciwnie, wprowadza ryzyko niespójności pomiędzy stanem zapisanym a stanem rzeczywistym.

3.3.3 Konsekwencje przyjętej architektury

Brak warstwy trwałego przechowywania danych eliminuje cztery klasy problemów typowych dla aplikacji bazodanowych: konieczność projektowania schematu bazy danych, problem migracji schematu przy zmianach modelu, problem synchronizacji stanu z systemem źródłowym oraz problem utrzymywania kopii zapasowej bazy danych. W zamian system staje się zależny od dostępności API monitorowanego orga w momencie wykonywania żądania użytkownika. Konsekwencją jest również brak możliwości analizy danych historycznych - system udostępnia jedynie obraz bieżący stanu środowiska.

Ograniczenie to jest świadome i pozostaje zgodne z celem pracy, którym jest monitorowanie bieżącego stanu środowisk (podrozdział 2.7.4), nie zaś monitorowanie zmian w czasie. Analiza danych historycznych stanowi odmienne zagadnienie projektowe, adresowane przez narzędzia typu `sfdx-hardis` czy Metazoa Snapshot. Możliwość rozszerzenia systemu o warstwę pamięci podręcznej (*cache*) lub składowanie historyczne omówiono w podrozdziale 6.5.

3.4. Model uwierzytelniania i minimalizacja uprawnień

Trzecim założeniem koncepcji jest model uwierzytelniania umożliwiający dostęp do wielu środowisk Salesforce bez przechowywania haseł użytkowników oraz z minimalnym zakresem uprawnień po stronie monitorowanych orgów.

3.4.1 Wybór mechanizmu uwierzytelniania

System wykorzystuje mechanizm JWT Bearer Token Flow w wariantcie komunikacji międzyserwerowej (*server-to-server*), udostępniany przez platformę Salesforce jako jeden z wariantów standardu OAuth 2.0 [27]. W modelu tym system monitorujący nie posługuje się hasłem użytkownika technicznego, lecz prywatnym kluczem kryptograficznym RS256, którym podpisuje token JWT. Token ten jest następnie wymieniany z platformą Salesforce na token dostępowy (*access token*). Klucz publiczny powiązany z certyfikatem rejestruje się po stronie aplikacji External Client App docelowego orga, a klucz prywatny pozostaje wyłącznie po stronie systemu [28]. External Client App stanowi wprowadzony przez Salesforce następce mechanizmu Connected App, zaprojektowany pod kątem bezpieczeństwa i dystrybucji w modelu pakietów drugiej generacji. Tworzenie nowych aplikacji typu Connected App zostało ograniczone począwszy od wydania Spring '26, a zalecanym rozwiązaniem dla nowych integracji jest External Client App.

Wybór tego mechanizmu motywowany jest jego cechami szczególnie istotnymi dla architektury proponowanego systemu:

- brak konieczności interaktywnego logowania użytkownika przy każdym żądaniu - istotne dla automatycznego pobierania danych z wielu orgów,
- brak konieczności przechowywania hasła użytkownika technicznego - eliminuje znaczącą klasę zagrożeń bezpieczeństwa,

- niezależność uwierzytelnienia od każdego monitorowanego orga - każdy org rejestrowany jest jako osobna aplikacja External Client App i może dysponować własnym kluczem prywatnym, co umożliwia rotację poświadczeń bez wpływu na pozostałe środowiska.

3.4.2 Użytkownik techniczny i zasada najmniejszych uprawnień

W każdym monitorowanym orgu wymagane jest utrzymywanie dedykowanego użytkownika technicznego, w imieniu którego system wykonuje zapytania. Wynika to z modelu autoryzacji platformy Salesforce - każde wywołanie API odbywa się w kontekście konkretnego użytkownika i podlega jego uprawnieniom.

Przyjęcie modelu tylko do odczytu (podrozdział 3.2) oraz znajomość konkretnego zestawu operacji wykonywanych przez system pozwala na zdefiniowanie minimalnego zestawu uprawnień użytkownika technicznego. Uprawnienia te obejmują:

- uprawnienie `API Enabled` - warunek dostępu do interfejsów programistycznych,
- uprawnienie `View Setup and Configuration` - dostęp do informacji o konfiguracji orga,
- uprawnienia `Read` dla wybranych obiektów standardowych dostępnych przez REST API, w tym `Organization` oraz `UserLicense`

Część danych monitorowanych przez system - informacje o zainstalowanych pakietach, historii wdrożeń oraz środowiskach typu *sandbox* - pozyskiwana jest z obiektów Tooling API (`InstalledSubscriberPackage`, `DeployRequest`, `SandboxInfo`). Obiekty te nie są udostępniane jako obiekty standardowe z odrębnym uprawnieniem odczytu na poziomie obiektu, w związku z czym nie wchodzi w skład minimalnego zestawu. Zakres dostępu wymagany dla danych pozyskiwanych tą drogą przeanalizowano empirycznie w podrozdziale 6.3, gdzie wykazano, że odczyt pakietów oraz wdrożeń wymaga uprawnień administracyjnych. Praktycznym sposobem zachowania minimalnego zestawu pozostaje wyłączenie odpowiednich modułów mechanizmem przełączników (podrozdział 3.5).

Uprawnienia nadawane są w postaci dedykowanego zestawu uprawnień (`Permission Set`) przypisanego do użytkownika technicznego. Użytkownik nie posiada uprawnień do modyfikacji danych, instalowania komponentów ani administrowania platformą. Zestaw uprawnień jest deklaracyjny, co oznacza, że audytor bezpieczeństwa po stronie organizacji może zweryfikować zakres uprawnień systemu monitorującego bez konieczności analizy kodu źródłowego.

3.4.3 Buforowanie tokenów dostępowych

Każde uzyskanie tokenu dostępowego OAuth stanowi operację kosztowną - wymaga generowania i podpisywania tokenu JWT, wykonania żądania HTTP do *endpointu* autoryzacyjnego oraz wymiany odpowiedzi. Tokeny dostępowe zachowują ważność przez określony czas (standardowo dwie godziny), w związku z czym ich powtórne uzyskiwanie przy każdym żądaniu byłoby zbędnym obciążeniem zarówno dla pośrednika, jak i dla samej platformy Salesforce. W proponowanym rozwiązaniu wprowadzono mechanizm buforowania tokenów po stronie pośrednika - token uzyskany dla danego orga jest przechowywany w pamięci aplikacji i wykorzystywany ponownie do momentu zbliżania się terminu jego wygaśnięcia, po czym zostaje odświeżony.

3.5. Mechanizm wtyczek i przełączników funkcjonalności

Założeniem charakteryzującym proponowany system jako rozwiązanie otwarte na rozszerzenia jest mechanizm wtyczek - pozwalający na dodawanie kolejnych modułów monitorujących bez modyfikacji istniejących komponentów systemu.

3.5.1 Motywacja

Liczba parametrów potencjalnie istotnych z punktu widzenia monitorowania środowiska Salesforce jest znacząca i zmienia się wraz z rozwojem samej platformy. Lista parametrów omówionych w podrozdziale 2.3 - pakiety, limity systemowe, licencje, metadane - stanowi jedynie wybór najczęściej monitorowanych elementów. Konkretnie potrzeby monitorowania różnią się między organizacjami: dla zespołu *Managed Service Provider* istotne mogą być inne parametry niż dla zespołu wewnętrznego pracującego nad pojedynczą aplikacją.

System zaprojektowany dla zamkniętego zbioru funkcjonalności wymaga modyfikacji kodu źródłowego każdorazowo, gdy pojawia się potrzeba monitorowania nowego parametru. Modyfikacja taka pociąga za sobą ponowną kompilację, testy regresji oraz wdrożenie nowej wersji systemu. Mechanizm wtyczek eliminuje tę konieczność - pozwala na dodanie nowego modułu monitorującego jako jednostki rozszerzenia, niezależnej od istniejących modułów.

3.5.2 Pojęcie wtyczki

W proponowanej koncepcji wtyczką nazywamy moduł realizujący monitorowanie wybranego parametru lub grupy powiązanych parametrów środowiska Salesforce. Wtyczka definiuje:

- nazwę identyfikującą moduł,
- usługę odpowiedzialną za komunikację z platformą Salesforce w zakresie danego parametru,
- *endpoint* lub grupę *endpointów* udostępniających dane warstwie klienckiej,
- opcjonalnie - komponenty interfejsu użytkownika prezentujące wyniki.

Wtyczki rejestrowane są w centralnym rejestrze modułów, który jest źródłem informacji o dostępnych modułach dla pozostałych części systemu. Rejestr ten jest deklaracyjny - dodanie nowego modułu wymaga utworzenia jego komponentów oraz wpisania go do rejestru, bez modyfikacji pozostałych modułów.

3.5.3 Przełączniki funkcjonalności

Niezależnie od rejestru modułów wprowadzono mechanizm przełączników funkcjonalności (*feature toggles*) pozwalający na dynamiczne wyłączanie wybranych modułów lub pojedynczych funkcji bez usuwania ich z kodu źródłowego. Mechanizm ten wprowadza dwa poziomy granularności:

- wyłączanie całych modułów (na przykład: wyłączenie modułu porównywania środowisk),
- wyłączanie wybranych funkcji w obrębie modułu (na przykład: wyłączenie obsługi historii wdrożeń w module szczegółów środowiska).

Konfiguracja przełączników odbywa się przez zmienne środowiskowe systemu, niezależnie od kodu aplikacji. Warstwa kliencka pobiera informacje o aktywnych modułach z dedykowanego *endpointu* udostępnianego przez pośrednika i dostosowuje interfejs użytkownika - ukrywając zakładki oraz elementy odpowiadające wyłączonym funkcjom.

3.5.4 Konsekwencje przyjętego mechanizmu

Mechanizm wtyczek pociąga za sobą trzy konkretne konsekwencje projektowe:

- jednolity protokół komunikacji pomiędzy modułami a warstwą kliencką - każda wtyczka udostępnia dane w tym samym formacie odpowiedzi, co pozwala na implementację uniwersalnych komponentów interfejsu (na przykład komponentu prezentującego dane porównawcze, niezależnie od tego, jaki parametr środowiska jest porównywany).
- możliwość wdrożenia systemu w konfiguracji zredukowanej - z wyłączonymi modułami, których organizacja nie potrzebuje, co upraszcza interfejs użytkownika i ogranicza zakres wykonywanych zapytań do platformy Salesforce.

- możliwość rozwijania systemu w trybie inkrementalnym, w którym kolejne moduły monitorujące są dodawane przez różne osoby lub w różnych iteracjach, bez konieczności koordynacji zmian w jednym wspólnym module.

Proponowany mechanizm ma charakter generyczny już na poziomie koncepcyjnym, ponieważ zakłada możliwość dostosowania rozwiązania do różnych potrzeb organizacji. Elastyczność ta może być realizowana poprzez system dodatków, wtyczek lub modułów, niezależnie od konkretnej implementacji technicznej omówionej w rozdziale piątym.

3.6. Agregacja danych z wielu środowisk

Założenie agregacji - przedstawiania danych z wielu monitorowanych środowisk w pojedynczym interfejsie - adresuje problem rozproszenia informacji omówiony w podrozdziale 2.5.1.

3.6.1 Charakterystyka agregacji

Agregacja w proponowanym systemie ma charakter agregacji na żądanie (*on-demand*). W odpowiedzi na żądanie warstwy klienckiej pośrednik wykonuje równoległe zapytania do każdego z wybranych monitorowanych orgów, zbiera odpowiedzi, normalizuje je do wspólnego formatu i zwraca pojedynczy wynik zbiorczy. Agregacja realizowana jest po stronie pośrednika, nie po stronie warstwy klienckiej - ta otrzymuje już dane zestawione, gotowe do prezentacji.

Wybór modelu agregacji na żądanie, w odróżnieniu od agregacji okresowej zapisywanej do bazy danych, jest spójny z założeniem braku warstwy trwałego przechowywania danych (podrozdział 3.3). System nie utrzymuje zapisanego stanu zbiorczego - przy każdym wywołaniu agregacja powstaje od nowa na podstawie aktualnych danych.

3.6.2 Zakresy agregacji

W proponowanym systemie wyróżniono dwa typy widoków agregujących:

- Widok zbiorczy listy środowisk - prezentuje podstawowe informacje o każdym ze skonfigurowanych orgów (typ, opis, status połączenia), umożliwiając szybkie zorientowanie się w stanie wszystkich monitorowanych środowisk. Zapytanie agregujące wykonuje równoległe pobranie metadanych z każdego orga.
- Widok porównawczy parametrów - prezentuje wartość wybranego parametru (limity systemowe, licencje, zainstalowane pakiety) z dwóch lub więcej środowisk w postaci macierzowej, umożliwiającej bezpośrednie porównanie. Zapytanie agregujące wykonuje równoległe pobranie danego parametru z każdego z porównywanych orgów.

Lista parametrów dostępnych w widoku porównawczym jest funkcją zarejestrowanych wtyczek (podrozdział 3.5). Dodanie nowej wtyczki, której dane mają sens porównawczy, automatycznie udostępnia ten parametr do porównań - bez modyfikacji modułu porównawczego.

3.6.3 Konsekwencje wybranego modelu

Agregacja na żądanie wprowadza zależność czasu odpowiedzi pośrednika od najwolniejszego z monitorowanych orgów. W skrajnym przypadku awarii pojedynczego orga lub przekroczenia limitu API po jego stronie, całe żądanie zbiorcze może zostać opóźnione. W proponowanej koncepcji każdy z orgów odpytywany jest niezależnie, a jego ewentualna niedostępność jest sygnalizowana w odpowiedzi zbiorczej, nie zaś przerywa całe zapytanie. Dane z orgów dostępnych są zwracane wraz z informacją o nieobecnych pozycjach.

Drugą konsekwencją wybranego modelu jest brak możliwości analizy trendów w czasie. Każde wywołanie agregacji zwraca stan bieżący, niezestawienie ze stanem historycznym. Ograniczenie to, zgodne z założeniem braku warstwy trwałego przechowywania danych, omówiono w podrozdziale 3.3 oraz uznano za świadomy wybór projektowy.

3.7. Porównanie koncepcji z istniejącymi podejściami

Niniejszy podrozdział zestawia proponowaną koncepcję z istniejącymi podejściami omówionymi w rozdziale drugim, w odniesieniu do założeń projektowych przedstawionych w podrozdziale 3.1.

3.7.1 Zestawienie spełnienia założeń

Tabela 2 przedstawia syntetyczne zestawienie spełnienia czterech głównych założeń celu pracy (podrozdział 1.3) przez wybrane reprezentatywne podejścia oraz przez proponowany system. Symbol „+” oznacza pełne spełnienie założenia, „±” spełnienie częściowe lub uzależnione od konfiguracji, „-” oznacza niespełnienie.

Tabela 2. Spełnienie założeń projektowych przez reprezentatywne podejścia do monitorowania środowisk Salesforce.

Podejście	Model zewnętrzny tylko do odczytu	Agregacja	Wtyczki	Bez bazy danych i CI/CD
Aplikacje AppExchange	–	–	–	+
Gearset Observability	+	± (limit 2 orgów)	–	+
Metazoa Snapshot	+	+	–	– (lokalna baza)
Sfdx-hardis	+	+	±	– (Git + CI/CD)
Centralizacja w orgu	–	+	–	–
Proponowany system	+	+	+	+

Z zestawienia wynika, że żadne z istniejących podejść nie spełnia łącznie wszystkich czterech założeń. Pojedyncze założenia są realizowane przez wybrane narzędzia - żadne z nich nie udostępnia jednak ich pełnego zestawu, stanowiącego istotę proponowanego rozwiązania.

3.7.2 Pozycjonowanie koncepcji

Proponowany system pozycjonowany jest jako lekkie narzędzie monitorowania bieżącego stanu wielu środowisk Salesforce, dedykowane zespołom poszukującym alternatywy dla komercyjnych narzędzi typu DevOps o rozbudowanej funkcjonalności, jak i dla rozwiązań o otwartym kodzie źródłowym wymagających zaawansowanej infrastruktury ciągłej integracji.

Koncepcja nie pretenduje do zastąpienia narzędzi takich jak `sfdx-hardis` czy Metazoa Snapshot w obszarach, w których są one mocniejsze - audycie zmian metadanych w czasie, integracji z pełnym cyklem CI/CD generowaniu raportów historycznych. Adresuje natomiast lukę pomiędzy pakietami instalowanymi w pojedynczym orgu a pełnymi platformami DevOps - przestrzeń projektową, w której nie zidentyfikowano dotąd rozwiązania spełniającego łącznie wszystkie cztery założenia tezy pracy.

4. Narzędzia zastosowane w pracy

Rozdział czwarty charakteryzuje narzędzia i technologie wykorzystane do realizacji prototypu systemu monitorowania. Opis koncentruje się na cechach narzędzi istotnych dla przyjętej koncepcji (rozdział 3) oraz na powiązaniach między warstwami systemu, pomijając szczegóły dostępne w dokumentacji poszczególnych technologii. Podrozdział 4.1 przedstawia architekturę klient-serwer oraz ogólny stos technologiczny, natomiast podrozdziały 4.2-4.5 omawiają kolejno warstwę serwerową, integrację z platformą Salesforce, warstwę kliencką oraz komunikację między warstwami wraz z narzędziami wspierającymi.

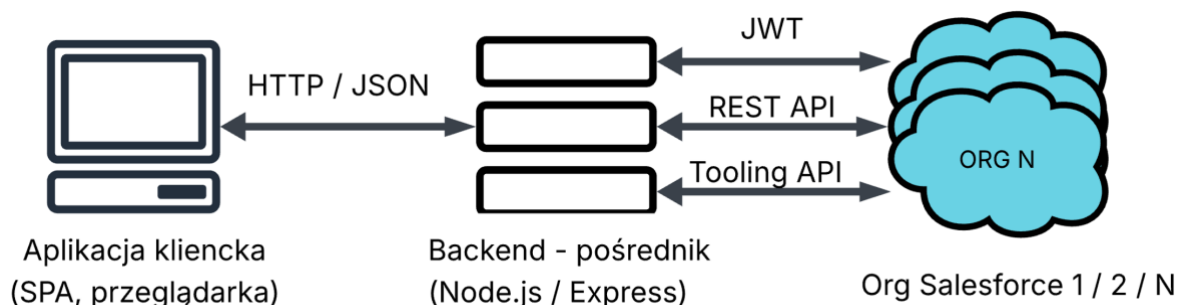
4.1. Architektura klient-serwer i ogólny stos technologiczny

Prototyp zrealizowano w architekturze klient-serwer, z podziałem na dwie odrębne aplikacje. Aplikacja kliencka działa w przeglądarce użytkownika jako aplikacja jednostronicowa (*Single Page Application*, SPA) i odpowiada wyłącznie za prezentację danych oraz nawigację. Aplikacja serwerowa pełni rolę zewnętrznego pośrednika opisanego w podrozdziale 3.3 - stanowi jedyny punkt styku z monitorowanymi orgami i udostępnia warstwie klienckiej ujednolicone interfejsy REST.

Podział na warstwę kliencką i serwerową wynika bezpośrednio z założeń przyjętych w rozdziale 3. Umieszczenie całości komunikacji z platformą Salesforce po stronie serwera podyktowane jest następującymi przesłankami:

- centralizacją uwierzytelniania - klucze prywatne RS256 oraz tokeny dostępowe pozostają wyłącznie po stronie serwera i nie są przekazywane do przeglądarki,
- ograniczeniem warstwy klienckiej do prezentacji - klient nie zna szczegółów integracji ani poświadczeń, co zmniejsza powierzchnię ataku,
- bezstanowością pośrednika - zgodnie z modelem bez warstwy trwałego przechowywania danych (podrozdział 3.3) serwer nie przechowuje danych monitorowanych środowisk poza krótkotrwałym buforem tokenów,
- niezależnością rozwoju warstw - rozdzielenie odpowiedzialności pozwala modyfikować interfejs użytkownika oraz logikę pośrednika niezależnie od siebie.

Rysunek 6 przedstawia ogólny przepływ danych w systemie. Aplikacja kliencka komunikuje się wyłącznie z aplikacją serwerową za pośrednictwem protokołu HTTP, z wymianą danych w formacie JSON. Aplikacja serwerowa, w odpowiedzi na żądania klienta, nawiązuje połączenia z wieloma orgami Salesforce, korzystając z interfejsów REST API oraz Tooling API. Monitorowane orgi pozostają wzajemnie niezależne, a ich liczba nie wpływa na architekturę systemu.



Rysunek 6. Architektura klient-serwer systemu monitorowania: aplikacja kliencka, serwerowy pośrednik oraz monitorowane środowiska Salesforce. Źródło: opracowanie własne.

Po obu stronach systemu zastosowano język TypeScript, co zapewnia spójną definicję typów danych przesyłanych między warstwami. Warstwę serwerową oparto na środowisku Node.js z biblioteką Express (podrozdział 4.2). Integrację z platformą Salesforce zrealizowano przy użyciu interfejsów REST API i Tooling API oraz mechanizmu uwierzytelniania JWT w standardzie OAuth 2.0 (podrozdział 4.3). Warstwę kliencką zbudowano z wykorzystaniem szkieletu Vue 3 wraz z biblioteką zarządzania stanem Pinia i mechanizmem trasowania Vue Router (podrozdział 4.4). Komunikację między warstwami oraz narzędzia wspierające proces wytwarzania omówiono w podrozdziale 4.5.

Tabela 3 zestawia technologie przypisane do poszczególnych warstw systemu wraz z ich rolą.

Tabela 3. Stos technologiczny prototypu w podziale na warstwy systemu.

Warstwa	Technologia	Rola w systemie
Kliencka	Vue 3, Pinia, Vue Router, TypeScript	interfejs użytkownika, prezentacja zagregowanych danych i nawigacja między widokami
Serwerowa	Node.js, Express, TypeScript	logika pośrednika, obsługa żądań REST i orkiestracja zapytań do orgów
Integracja z Salesforce	REST API, Tooling API, JWT OAuth 2.0	uwierzytelnianie oraz odczyt bieżącego stanu monitorowanych środowisk
Komunikacja i narzędzia wspierające	HTTP/JSON, axios, OpenAPI/Swagger, ESLint, Prettier	wymiana danych między warstwami, dokumentacja interfejsu API i kontrola jakości kodu

4.2. Technologie warstwy serwerowej

Wardstwę serwerową oparto na środowisku uruchomieniowym Node.js, które pozwala wykonywać kod języka JavaScript, a po kompilacji również TypeScript, poza przeglądarką [29]. Node.js działa w modelu jednowątkowym z pętlą zdarzeń (*event loop*) i nieblokującą obsługą operacji wejścia-wyjścia. Cecha ta jest istotna dla roli pośrednika: w odpowiedzi na pojedyncze żądanie klienta serwer wykonuje wiele równoległych zapytań do różnych orgów Salesforce, a model nieblokujący pozwala obsługiwać je współbieżnie, bez przydzielania osobnego wątku każdemu żądaniu. Wybór środowiska wynika również ze wspólnego języka z warstwą kliencką - zastosowanie jednego

ekosystemu po obu stronach systemu upraszcza utrzymanie oraz umożliwia współdzielenie definicji typów danych.

Obsługę żądań HTTP zrealizowano przy użyciu biblioteki Express, minimalistycznego szkieletu aplikacji webowych dla środowiska Node.js [30]. Express opiera się na potoku komponentów pośredniczących (*middleware*) oraz deklaratywnym definiowaniu tras (*routing*). Lekkość tego rozwiązania odpowiada przyjętemu w podrozdziale 3.1 założeniu prostoty wdrożenia - szkielet nie narzuca rozbudowanej struktury ani dodatkowej infrastruktury. Komponenty pośredniczące wykorzystano do obsługi reguł współdzielenia zasobów między domenami (*Cross-Origin Resource Sharing*, CORS) oraz do przetwarzania treści żądań w formacie JSON.

Kod warstwy serwerowej napisano w języku TypeScript, będącym nadzbiorem języka JavaScript rozszerzonym o statyczne typowanie [31]. Typowanie kontrolowane na etapie kompilacji pozwala wykrywać znaczną część błędów przed uruchomieniem aplikacji oraz precyzyjnie definiować kontrakty danych przekazywanych między komponentami. Te same definicje typów opisują obiekty transferu danych (*Data Transfer Object*, DTO) wymieniane między warstwą serwerową a kliencką, co ogranicza ryzyko niezgodności struktur danych po obu stronach systemu.

Połączenie wymienionych narzędzi tworzy lekką warstwę serwerową, której organizację oraz podział na moduły funkcjonalne opisano w rozdziale 5.

4.3. Integracja z platformą Salesforce

Integrację z platformą Salesforce zrealizowano wyłącznie poprzez interfejsy programistyczne typu REST, bez bezpośredniego dostępu do warstwy danych monitorowanych środowisk. Dostęp autoryzowany jest standardem JSON Web Token Bearer Flow w ramach protokołu OAuth 2.0; szczegółowy opis mechanizmu uwierzytelniania, w tym roli aplikacji External Client App, klucza RS256 oraz dedykowanego zbioru uprawnień, zawarto w podrozdziale 3.4. Niniejszy podrozdział charakteryzuje wykorzystane interfejsy jako narzędzia odczytu stanu środowisk.

Podstawowym interfejsem jest REST API platformy Salesforce, udostępniające zasoby poprzez żądania HTTP zwracające dane w formacie JSON [32]. Interfejs jest wersjonowany, a poszczególne zasoby identyfikowane są ścieżkami URL. System korzysta z niego wyłącznie w trybie odczytu (metoda GET): pobiera bieżące wykorzystanie limitów organizacji za pośrednictwem dedykowanego zasobu oraz wykonuje zapytania w języku SOQL (*Salesforce Object Query Language*) zwracające rekordy obiektów konfiguracyjnych.

Część informacji opisujących stan środowiska nie jest dostępna przez standardowe REST API i wymaga zastosowania Tooling API. Jest to wyspecjalizowany interfejs przeznaczony pierwotnie dla narzędzi deweloperskich, udostępniający obiekty metadanych platformy poprzez analogiczny mechanizm zapytań SOQL. Równoległe użycie obu interfejsów wynika z zakresu danych, jaki każdy z nich obejmuje, a nie z preferencji projektowych - dane o wdrożeniach, środowiskach typu sandbox oraz zainstalowanych pakietach pozyskiwane są wyłącznie przez Tooling API.

Tabela 4 zestawia kategorie danych monitorowanych przez system z interfejsami oraz obiektami, z których są pozyskiwane. Wszystkie wymienione operacje mają charakter wyłącznie odczytowy, zgodnie z przyjętą zasadą nieingerencji w monitorowane środowiska.

Tabela 4. Kategorie monitorowanych danych i przypisane im interfejsy programistyczne platformy Salesforce.

Kategoria danych	Interfejs	Obiekt/ zasób
Wykorzystanie limitów organizacji	REST API	zasób /limits

Informacje o organizacji	REST API	SOQL: Organization
Licencje użytkowników	REST API	SOQL: UserLicense
Historia wdrożeń	Tooling API	SOQL: DeployRequest
Środowiska typu sandbox	Tooling API	SOQL: SandboxInfo
Zainstalowane pakiety	Tooling API	SOQL: InstalledSubscriberPackage

4.4. Technologie warstwy klienckiej

Warstwę kliencką zbudowano w oparciu o szkielet Vue 3 [33]. Vue to deklaratywny szkielet do tworzenia interfejsów użytkownika oparty na reaktywnych komponentach, w którym zmiana danych automatycznie odświeża powiązane z nimi fragmenty widoku. Wybór tego rozwiązania podyktowany jest niskim progiem wejścia oraz niewielkim narzutem szkieletu, co odpowiada skali prototypu, a także rozbudowaną i uporządkowaną dokumentacją ułatwiającą rozwój aplikacji.

Stan aplikacji klienckiej zarządzany jest biblioteką Pinia, stanowiącą zalecane w ekosystemie Vue rozwiązanie do centralnego przechowywania stanu [34]. Umożliwia ona współdzielenie danych, takich jak lista monitorowanych środowisk, pomiędzy wieloma widokami bez konieczności przekazywania ich przez kolejne poziomy komponentów.

Nawigację w obrębie aplikacji realizuje Vue Router, mechanizm trasowania (*routing*) po stronie klienta charakterystyczny dla aplikacji jednostronicowych [35]. Odzworowuje on ścieżki adresu URL na poszczególne widoki bez przeładowania strony, co zapewnia płynne przechodzenie między ekranami listy środowisk, szczegółów pojedynczego środowiska oraz porównania.

Tabela 5 zestawia biblioteki tworzące warstwę kliencką wraz z pełnioną przez nie rolą.

Tabela 5. Biblioteki warstwy klienckiej i ich rola w systemie.

Biblioteka	Rola w warstwie klienckiej
Vue 3	budowa interfejsu użytkownika z reaktywnych komponentów
Pinia	centralne zarządzanie stanem współdzielonym między widokami
Vue Router	Mechanizm trasowania po stronie klienta, odwzorowanie adresów URL na widoki

4.5. Komunikacja między warstwami i narzędzia wspierające

Komunikacja między warstwą kliencką a serwerową odbywa się za pośrednictwem protokołu HTTP, z danymi przesyłanymi w formacie JSON. Po stronie klienta żądania realizuje biblioteka axios [38], czyli klient HTTP oparty na obietnicach (*promises*), upraszczający wysyłanie żądań oraz obsługę odpowiedzi. Ujednolicony format wymiany danych wraz ze wspólnymi definicjami typów (podrozdział 4.2) zapewnia spójność struktur danych po obu stronach systemu.

Budowanie i uruchamianie warstwy klienckiej obsługują narzędzie Vite [36]. W trybie deweloperskim udostępnia ono lokalny serwer z natychmiastowym przeładowaniem zmienionych modułów, a na potrzeby wdrożenia tworzy zoptymalizowany pakiet produkcyjny. Krótki czas

uruchomienia oraz prosta konfiguracja skracają cykl wprowadzania i podglądu zmian podczas rozwoju interfejsu, co ma znaczenie przy iteracyjnym tworzeniu warstwy prezentacji.

Proces wytwarzania wspierają dodatkowo narzędzia pomocnicze. Statyczną analizę kodu oraz ujednolicenie jego formatowania zapewniają odpowiednio ESLint i Prettier. Parametry konfiguracyjne aplikacji odczytywane są ze zmiennych środowiskowych, co oddziela ustawienia wdrożenia od kodu źródłowego. Interfejs REST warstwy serwerowej udokumentowano w standardzie OpenAPI, udostępniając jego interaktywną postać (Swagger UI); pełna dokumentacja interfejsu jest możliwa dzięki wyłącznie odczytowemu charakterowi udostępnianych operacji.

Tabela 6 zestawia narzędzia wspierające proces wytwarzania wraz z pełnioną przez nie funkcją. Sposób wykorzystania opisanych w niniejszym rozdziale narzędzi w implementacji prototypu przedstawiono w rozdziale 5.

Tabela 6. Narzędzia wspierające proces wytwarzania prototypu.

Narzędzie	Funkcja
Vite	budowanie warstwy klienckiej i lokalny serwer deweloperski
axios	klient HTTP do komunikacji między warstwami
ESLint	statyczna analiza kodu
Prettier	ujednolicenie formatowania kodu
dotenv	konfiguracja poprzez zmienne środowiskowe
OpenAPI / Swagger UI	dokumentacja interfejsu REST

5. Implementacja prototypu

Rozdział piąty opisuje implementację prototypu systemu monitorowania, łącząc koncepcję przedstawioną w rozdziale 3 z narzędziami scharakteryzowanymi w rozdziale 4. Strukturę rozdziału dobrano tak, aby odzwierciedlała założenia koncepcyjne: kolejne podrozdziały omawiają organizację projektu, konfigurację monitorowanych środowisk wraz z jej walidacją, uwierzytelnianie i komunikację z platformą Salesforce, mechanizm wtyczek, moduły monitorujące oraz warstwę kliencką. W niewralgicznych miejscach implementację zilustrowano fragmentami kodu źródłowego.

5.1. Struktura projektu

Prototyp składa się z dwóch niezależnych aplikacji: serwerowej, umieszczonej w katalogu `backend` oraz klienckiej, umieszczonej w katalogu `frontend`. Każda z nich posiada własną konfigurację oraz zależności, co odpowiada architekturze klient-serwer opisanej w podrozdziale 4.1. Strukturę katalogów prototypu przedstawia Rysunek 7.

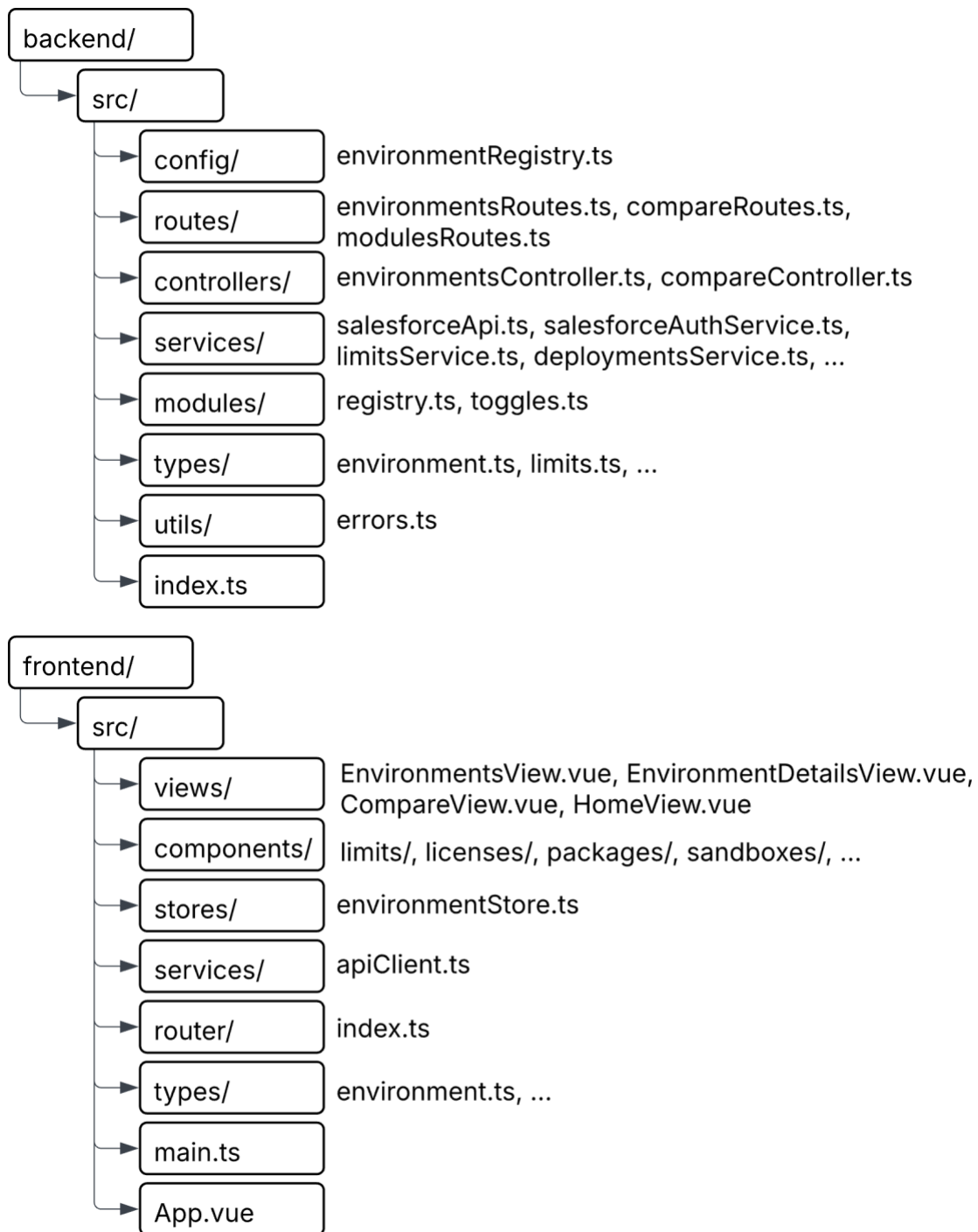
Kod warstwy serwerowej zorganizowano zgodnie z zasadą rozdzielania odpowiedzialności, dzieląc go na warstwy o ściśle określonych zadaniach:

- trasy (`routes`) - definicja punktów końcowych interfejsu REST oraz przyjmowanie żądań,
- kontrolery (`controllers`) - obsługa żądań i przygotowanie odpowiedzi,
- serwisy (`services`) - logika domenowa poszczególnych obszarów monitorowania,
- moduł komunikacji (`salesforceApi`) - realizacja wywołań HTTP do interfejsów platformy Salesforce.

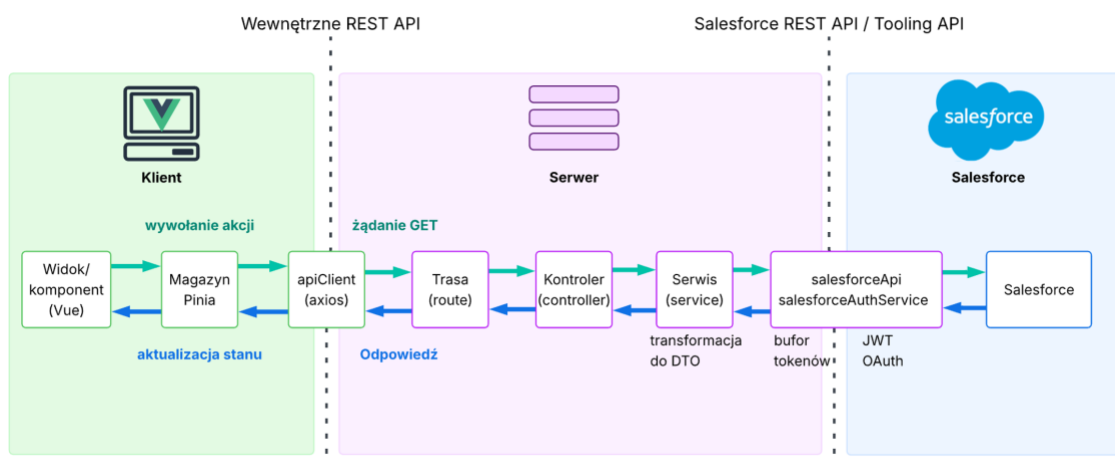
Warstwy uzupełniają katalogi pomocnicze: konfiguracja monitorowanych środowisk (`config`), mechanizm wtyczek (`modules`), definicje typów danych (`types`) oraz obsługa błędów (`utils`). Żądanie klienta przechodzi kolejno przez trasę, kontroler i serwis, a następnie, jeśli wymaga danych ze środowiska, przez moduł komunikacji wykonujący zapytanie do platformy Salesforce. Taki przepływ izoluje logikę domenową od szczegółów transportu i uwierzytelniania.

Warstwę kliencką podzielono analogicznie. Widoki (`views`) odpowiadają trasom aplikacji, komponenty (`components`) realizują prezentację poszczególnych kategorii danych, magazyn stanu (`store`) przechowuje dane współdzielone między widokami, a moduł klienta HTTP (`apiClient`) pośredniczy w komunikacji z warstwą serwerową. Definicje typów danych, obecne po obu stronach systemu, opisują strukturę obiektów wymienianych między aplikacjami, zapewniając spójność, o której mowa w podrozdziale 4.2.

Drzewo katalogów oddaje statyczny podział kodu, nie pokazuje jednak drogi pojedynczego żądania przez warstwy systemu. Przepływ ten przedstawia Rysunek 8.



Rysunek 7. Struktura katalogów prototypu (wybrane pliki). Źródło: opracowanie własne.



Rysunek 8. Cykl życia żądania w architekturze pośrednika: od warstwy klienckiej do platformy Salesforce i z powrotem. Źródło: opracowanie własne.

Żądanie jest inicjowane w warstwie klienckiej przez komponent Vue, który wywołuje odpowiednią akcję magazynu stanu Pinia. Następnie magazyn korzysta z modułu `apiClient`, który kieruje żądania do odpowiedniej trasy pośrednika. Trasa przekazuje żądanie do kontrolera, a ten do serwisu realizującego logikę danego obszaru monitorowania. Serwis odwołuje się do modułu `salesforceApi`, który - po pobraniu lub odświeżeniu tokenu dostępowego z bufora (podrozdział 5.3) - wykonuje żądanie GET do interfejsu REST API lub Tooling API monitorowanego orga. Odpowiedź platformy wraca tą samą drogą: serwis przekształca surowe rekordy do wewnętrznego modelu danych, kontroler buduje odpowiedź, a `apiClient` przekazuje ją do magazynu stanu i widoku.

5.2. Konfiguracja środowisk i jej walidacja

Monitorowane środowiska definiowane są deklaratywnie w pliku konfiguracyjnym (`environments.json`), niezależnie od kodu aplikacji. Każdy wpis opisuje pojedynczy org za pomocą zestawu pól: identyfikatora, nazwy, typu środowiska, nazwy użytkownika technicznego, identyfikatora aplikacji (`clientId`), ścieżki do klucza prywatnego, adresu usługi uwierzytelniania oraz pól opcjonalnych. Dodanie nowego środowiska do monitorowania sprowadza się do dopisania kolejnego wpisu i nie wymaga modyfikacji kodu, co odpowiada przyjętej zasadzie rozszerzalności.

Plik konfiguracyjny nie przechowuje poświadczeń wrażliwych. Zawiera jedynie ścieżkę do klucza prywatnego (`privateKeyPath`), który pozostaje poza repozytorium i nie opuszcza serwera, zgodnie z modelem uwierzytelniania opisanym w podrozdziale 3.4.

Poprawność konfiguracji weryfikowana jest podczas wczytywania pliku przy użyciu biblioteki walidacyjnej Zod [37]. Zawartość pliku jest parsowana, a następnie porównywana ze zdefiniowanym schematem określającym wymagane pola oraz ich typy. Niezgodność struktury lub brak pliku skutkują zgłoszeniem błędu konfiguracji (`ConfigurationError`) i przerwaniem uruchamiania aplikacji, co wyklucza działanie systemu na niekompletnych danych. Po pierwszym poprawnym odczycie definicje są buforowane w pamięci. Schemat walidacji przedstawia Listing 1.

Listing 1. Schemat walidacji definicji środowiska z użyciem biblioteki Zod.

```

const envDefinitionSchema = z.object({
  id: z.string().min(1),
  name: z.string().min(1),
  type: z.enum(["production", "sandbox"]),
  username: z.string().min(1),
  clientId: z.string().min(1),
  privateKeyPath: z.string().min(1),
  authBase: z.string().url(),
  parentId: z.string().optional(),
  description: z.string().optional(),
  apiVersion: z.string().optional()
});

const registrySchema = z.array(envDefinitionSchema);

```

5.3. Uwierzytelnianie i komunikacja z platformą Salesforce

Każde odwołanie do orga poprzedzone jest uzyskaniem tokenu dostępowego w przepływie JWT *Bearer Flow*, którego założenia opisano w podrozdziale 3.4. Dla danego środowiska system buduje podpisany token JWT, a następnie wymienia go na token dostępowy w punkcie uwierzytelniania platformy Salesforce.

Ładunek tokenu JWT zawiera identyfikator aplikacji jako wystawcę (*iss*), nazwę użytkownika technicznego jako podmiot (*sub*), adres usługi uwierzytelniania jako odbiorcę (*aud*) oraz krótki czas ważności. Token podpisywany jest biblioteką *jsonwebtoken* [39], algorytmem RS256 kluczem prywatnym wczytanym z dysku, po czym przesyłany do platformy z typem nadania *jwt-bearer*. Proces ten przedstawia Listing 2

Listing 2. Budowa i wymiana tokenu JWT na token dostępowy.

```

const payload = {
  iss: env.clientId, //id aplikacji (External Client App)
  sub: env.username, // użytkownik techniczny
  aud: env.authBase, // adres usługi uwierzytelniania
  exp: Math.floor(Date.now() / 1000) + 3 * 60
};

const assertion = jwt.sign(payload, privateKey, {
  algorithm: "RS256"
});

const response = await axios.post(
  `${env.authBase}/services/oauth2/token`,
  new URLSearchParams({
    grant_type: "urn:ietf:params:oauth:grant-type:jwt-bearer",
    assertion
  }),
  {headers: {
    „Content-Type”: "application/x-www-form-urlencoded"}}
);

```

Uzyskany token dostępowy oraz adres instancji są buforowane w pamięci, osobno dla każdego środowiska, do momentu poprzedzającego ich wygaśnięcie. Kolejne żądania w okresie ważności

korzystają z bufora, co eliminuje powtórne podpisywanie tokenu przy każdym wywołaniu i ogranicza liczbę odwołań do punktu uwierzytelniania, zgodnie z mechanizmem buforowania z podrozdziału 3.4.

Wszystkie odwołania do danych środowiska przechodzą przez jedną funkcję wykonującą żądanie metodą GET na wersjonowanej ścieżce interfejsu REST, z tokenem dostępowym przekazanym w nagłówku autoryzacji. Zapytania w języku SOQL kierowane do REST API oraz do Tooling API stanowią cienkie nakładki nad tą funkcją, różniące się jedynie ścieżką zasobu. Udostępnienie wyłącznie operacji odczytu, bez metod tworzących lub modyfikujących dane, stanowi techniczne odzwierciedlenie modelu obserwacyjnego z podrozdziału 3.2. Rozwiązanie to przedstawia Listing 3.

Listing 3. Wspólny mechanizm odczytu danych dla REST API i Tooling API.

```
export async function restGet<T>(  
  envId: string,  
  path: string,  
  options: RequestOptions = {}  
) : Promise<T> {  
  try {  
    const conn = await getSalesforceConnection(envId);  
    const url = `  
${conn.instanceUrl}/services/data/v${conn.apiVersion}${path}`;  
    const response = await axios.get<T>(url, {  
      headers: {  
        Authorization: `Bearer ${conn.accessToken}`  
      },  
      params: options.params  
    });  
    return response.data;  
  } catch (error: any) {  
    console.error(  
      "SF error:",  
      JSON.stringify(error.response?.data));  
    throw error;  
  }  
}  
  
export async function query<T>(  
  envId: string,  
  soql: string) {  
  return restGet<QueryResult<T>>(envId, "/query", {params: {q:  
soql}}});  
}  
  
export async function toolingQuery<T>(  
  envId: string,  
  soql: string  
) : Promise<QueryResult<T>> {  
  return restGet<QueryResult<T>>(  
    envId,  
    "/tooling/query",  
    {params: {q: soql}});  
}
```

5.4. Mechanizm wtyczek i przełączników funkcjonalności

Funkcjonalności warstwy serwerowej zorganizowano w moduły rejestrowane w centralnym wykazie. Każdy moduł opisany jest deskryptorem zawierającym nazwę, ścieżkę bazową, zestaw tras (router) oraz listę funkcji składowych. W prototypie zdefiniowano dwa moduły: moduł `environments`, udostępniający dane pojedynczych środowisk (limity, licencje, pakiety, historię wdrożeń oraz środowiska typu *sandbox*), oraz moduł `compare`, realizujący porównywanie środowisk. Poszczególne kategorie danych w obrębie modułu `environments` stanowią jego funkcje składowe (*features*) wymienione w deskrytorze modułu. Mechanizm wtyczek opisany w podrozdziale 3.5 realizowany jest zatem na dwóch poziomach - dodania nowego modułu oraz dodania nowej funkcji do modułu istniejącego - w obu przypadkach bez ingerencji w pozostały kod.

Podczas uruchamiania aplikacji wykaz jest przeglądany, a trasy danego modułu montowane są wyłącznie wtedy, gdy moduł pozostaje włączony. Włączanie i wyłączanie całych modułów oraz pojedynczych funkcji odbywa się poprzez zmienne środowiskowe (`DISABLED_MODULES`, `DISABLED_FEATURES`), bez modyfikacji kodu, co odpowiada przełącznikom funkcjonalności z podrozdziału 3.5. Rejestrację modułów przedstawia Listing 4.

Listing 4. Warunkowa rejestracja modułów na podstawie ich stanu włączenia.

```
export function registerModules(app: Express) {
  moduleDescriptors.forEach((descriptor) => {
    if (isModuleEnabled(descriptor.name)) {
      app.use(descriptor.basePath, descriptor.router);
    }
  });
}
```

Wykaz modułów wraz ze stanem ich włączenia udostępniany jest przez dedykowany punkt końcowy interfejsu. Pozwala to warstwie klienckiej dostosować prezentowany interfejs do funkcji aktualnie dostępnych w systemie, bez konieczności zakładania ich stałej obecności.

Strukturę deskryptora modułu, formalizującą opisane wyżej pola, przedstawia Listing 5.

Listing 5. Struktura deskryptora modułu oraz rejestracja przykładowego modułu monitorującego.

```
interface ModuleDescriptor {
  name: string;
  basePath: string;
  router: Router;
  description: string; // opis prezentowany w wykazie modułów
  features?: string[]; // funkcje składowe podlegające
przełącznikom
}
```

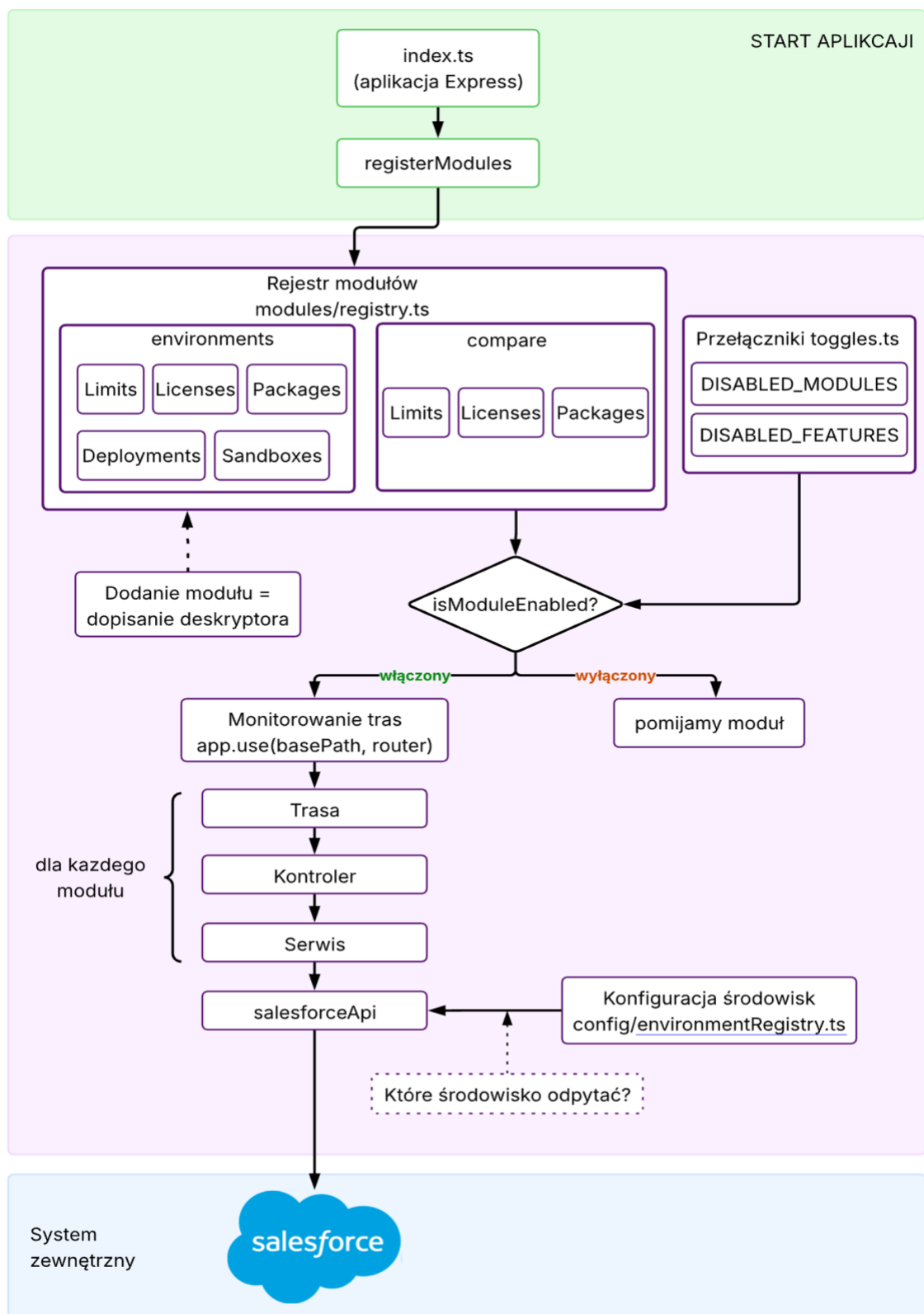
Dodanie nowego modułu monitorującego obejmuje trzy czynności:

- Implementację serwisu pobierającego dane z platformy Salesforce i przekształcającego je do wewnętrznego modelu danych, analogicznie do serwisów opisanych w podrozdziale 5.5.
- Zdefiniowanie tras udostępniających te dane warstwie klienckiej pod ścieżką bazową modułu.
- Dopisanie deskryptora do tablicy `moduleDescriptors`, z podaniem nazwy, ścieżki bazowej, routera oraz listy funkcji składowych.

Po dopisaniu deskryptora moduł jest automatycznie montowany przez funkcję `registerModules`, o ile pozostaje włączony oraz udostępniany w wykazie modułów pobieranym

przez warstwę kliencką (podrozdział 5.6). Dodanie nowej funkcji do istniejącego modułu przebiega analogicznie, lecz bez tworzenia nowego deskryptora. Wymaga ono implementacji serwisu pobierającego dane, zdefiniowania odpowiadającej mu trasy montowanej warunkowo na podstawie stanu funkcji (`isFeatureEnabled`) oraz dopisania nazwy funkcji do listy `features` w deskrypcji modułu. Tak dodana funkcja podlega następnie przełącznikom funkcjonalności na równi z pozostałymi. Pozostałe moduły nie wymagają żadnej modyfikacji. Tym samym implementacja realizuje wymóg generyczności prototypu, omówiony na poziomie koncepcji w podrozdziale 3.5: rozszerzenie zakresu monitorowania sprowadza się do dodania niezależnej jednostki rozszerzenia.

Statyczną organizację warstwy serwerowej wraz z rolą rejestru modułów przedstawia Rysunek 9.



Rysunek 9. Warstwy backendu i rola rejestru modułów w warunkowym montowaniu tras.
Źródło: opracowanie własne.

Podczas uruchamiania aplikacji moduł wejściowy odwołuje się do rejestru przechowującego deskryptory wszystkich modułów monitorujących. Dla każdego deskryptora rejestr sprawdza jego stan włączenia, wyznaczany przez przełączniki funkcjonalności (zmienne `DISABLED_MODULES` oraz `DISABLED_FEATURES`), i montuje trasy modułu wyłącznie wtedy, gdy moduł pozostaje aktywny. Każdy zamontowany moduł zachowuje jednolity podział na trasę, kontroler oraz serwis, a komunikację z platformą realizuje wspólny moduł `salesforceApi` korzystający z konfiguracji monitorowanych środowisk. Dodanie nowego obszaru monitorowania sprowadza się do dopisania kolejnego deskryptora, bez ingerencji w pozostałe warstwy.

5.5. Moduły monitorujące i agregacja danych

Każdy obszar monitorowania zaimplementowano jako odrębny serwis o jednorodnej budowie: wykonuje on pojedyncze zapytanie do właściwego interfejsu, a następnie przekształca surowe rekordy zwrócone przez platformę Salesforce do wewnętrznego, ujednoliconego modelu danych. Dzięki temu dalsze warstwy systemu nie zależą od formatu odpowiedzi platformy. Listing 6 przedstawia serwis pobierający historię wdrożeń przez Tooling API.

Listing 6. Serwis monitorujący wdrożenia z transformacją do modelu wewnętrznego.

```
export async function getDeploymentsForEnvironment(
  envId: string
) {
  const soql = `
    SELECT Id, Status, StartDate, CompletedDate, CreatedBy.Name
    FROM DeployRequest
    ORDER BY CreatedDate DESC
  `;
  const result = await toolingQuery<DeploymentRecord>(envId,
soql);
  return result.records.map((record) => {
    const start = record.StartDate ? new Date(record.StartDate) :
null;
    const completed = record.CompletedDate
      ? new Date(record.CompletedDate)
      : null;
    const duration =
      start && completed
        ? Math.max(completed.getTime() - start.getTime(), 0)
        : 0;
    return {
      id: record.Id,
      status: record.Status,
      startDate: record.StartDate ?? "",
      completedDate: record.CompletedDate ?? "",
      createdBy: record.CreatedBy?.Name ?? "Unknown",
      duration
    };
  });
}
```

Zestawianie danych pochodzących z wielu środowisk, opisane jako założenie w podrozdziale 3.6, realizuje odrębny serwis porównawczy. Dla wybranej kategorii danych oraz listy środowisk pobiera on dane ze wszystkich wskazanych orgów równolegle, a następnie scala je w wiersze indeksowane nazwą

zasobu, gdzie każdy wiersz gromadzi wartości ze wszystkich środowisk. Oprócz przepisania pól rekordu serwis wyznacza czas trwania wdrożenia jako różnicę między datą zakończenia a datą rozpoczęcia, udostępniany następnie w kolumnie czasu trwania w widoku historii wdrożeń (Rysunek 13). Pozwala to zestawić tę samą metrykę z wielu orgów obok siebie. Równoległe pobieranie wykorzystuje asynchroniczny model środowiska Node.js, o którym mowa w podrozdziale 4.2.

Przed pobraniem danych nazwa kategorii oraz identyfikatory środowisk są weryfikowane względem dostępnej konfiguracji. Nieobsługiwana kategoria lub nieznane środowisko skutkują zwróceniem błędu żądania, bez podejmowania prób odpytania platformy. Mechanizm scalania danych przedstawia Listing 7.

Listing 7. Równoległe pobranie i scalenie danych z wielu środowisk.

```
async function buildLimitRows(envIds: string[]) {
  const dataByEnv = await Promise.all(
    envIds.map(async (environmentId) => ({
      environmentId,
      limits: await getLimitsForEnvironment(environmentId)
    }))
  );

  const rowsMap = new Map<string, LimitComparisonRow>();
  dataByEnv.forEach(({ environmentId, limits }) => {
    limits.forEach((limit) => {
      const row =
        rowsMap.get(limit.name) ?? { name: limit.name,
environments: [] };
      row.environments.push({
        environmentId,
        max: limit.max,
        remaining: limit.remaining
      });
      rowsMap.set(limit.name, row);
    });
  });

  return Array.from(rowsMap.values());
}
```

Ujednolicenie modeli danych oraz wykonanie agregacji po stronie serwera odciążają warstwę kliencką, która otrzymuje zestawienia gotowe do prezentacji. Sposób ich wyświetlenia omówiono w podrozdziale 5.6.

5.6. Warstwa kliencka

Warstwa kliencka komunikuje się z serwerem poprzez pojedynczą, współdzieloną instancję klienta HTTP (*axios*), skonfigurowaną adresem bazowym interfejsu oraz limitem czasu odpowiedzi. Przechwytywacz odpowiedzi ujednolici komunikaty o błędach, dzięki czemu widoki otrzymują spójny ich format niezależnie od źródła problemu.

Stan aplikacji utrzymywany jest w magazynie *Pinia*, gromadzącym pobrane dane, takie jak lista środowisk, ich szczegóły, limity, licencje, pakiety, wdrożenia oraz wyniki porównań, wraz ze znacznikami trwającego ładowania. Dane pobrane raz są przechowywane w magazynie i

wykorzystywane ponownie, co ogranicza liczbę żądań kierowanych do serwera, a jednocześnie udostępnia je wielu widokom bez przekazywania przez kolejne komponenty.

Przed pobraniem danych magazyn pobiera z serwera wykaz dostępnych modułów i funkcji, opisany w podrozdziale 5.4, i pomija odwołania do obszarów wyłączonych. W efekcie interfejs prezentuje wyłącznie funkcje aktualnie udostępniane przez serwer, bez zakładania ich stałej obecności. Pobranie wykazu modułów oraz pominięcie obszarów wyłączonych przedstawia Listing 8

Listing 8. Pobranie wykazu modułów i pominięcie funkcji wyłączonych po stronie klienta.

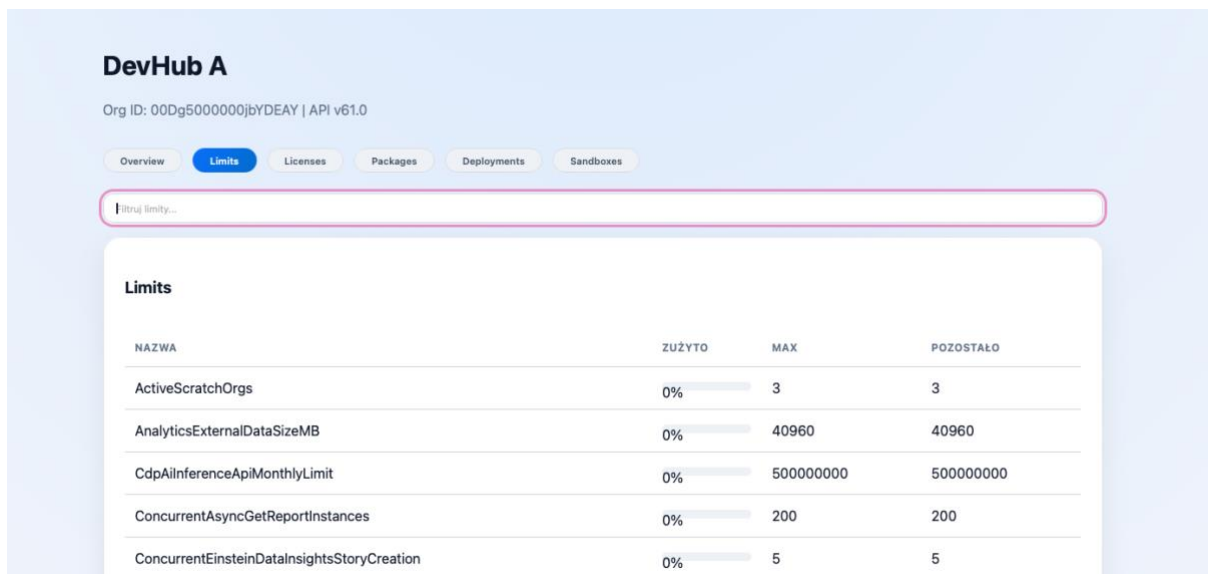
```
async fetchEnvironments(force = false) {
  await this.fetchModuleCapabilities();
  if (this.moduleCapabilities["environments"] === false) {
    this.environments = [];
    this.environmentsLoaded = true;
    return;
  }
  // ... pobranie listy środowisk z serwera
}
```

Nawigację realizuje Vue Router w trybie historii przeglądarki, odwzorowując adresy na poszczególne widoki: stronę główną, listę monitorowanych środowisk, szczegóły pojedynczego środowiska oraz widok porównania. Widoki ładowane są na żądanie, co zmniejsza początkowy rozmiar aplikacji. Prezentację poszczególnych kategorii danych realizują wyspecjalizowane komponenty, odpowiadające za zestawienia limitów, licencji, pakietów, wdrożeń, środowisk sandbox oraz porównań międzyśrodowiskowych.

5.7. Prezentacja danych w interfejsie

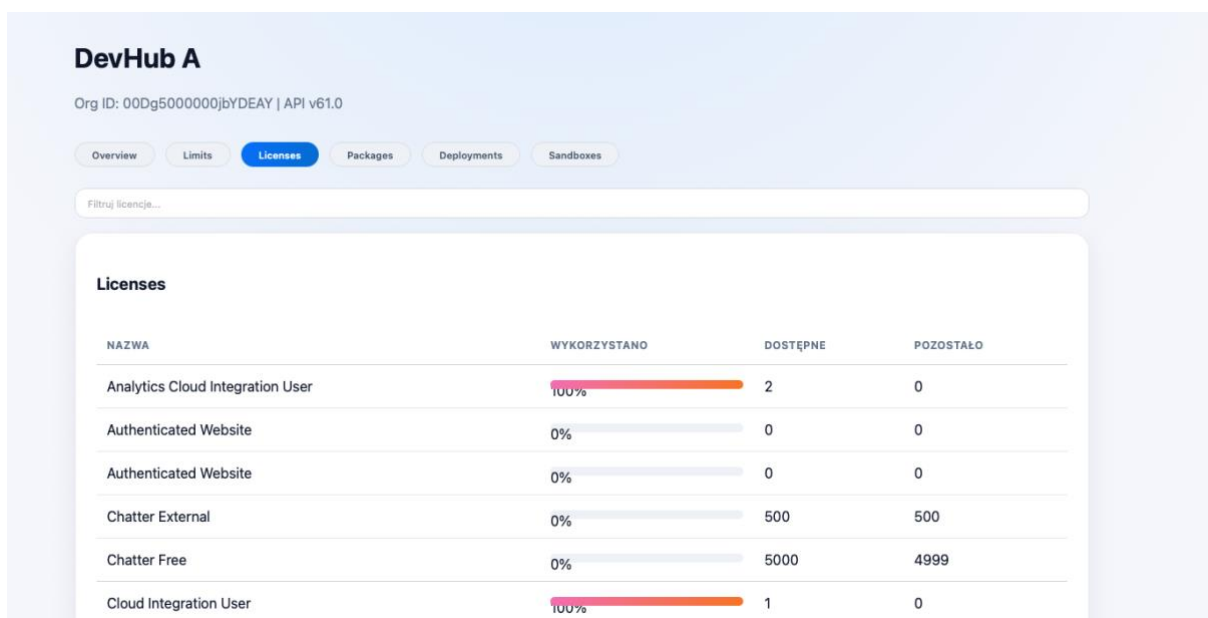
Warstwa kliencka prezentuje dane każdej kategorii w odrębnym widoku, zasilanym pojedynczym żądaniem do pośrednika i odświeżanym na żądanie użytkownika. Poszczególne widoki odpowiadają modułom monitorującym opisanym w podrozdziale 5.5 - limitom systemowym, licencjom użytkowników, zainstalowanym pakietom oraz historii wdrożeń. Kolejne rysunki przedstawiają te widoki dla pojedynczego środowiska.

Widok limitów systemowych zestawia bieżące wykorzystanie zasobów organizacji pobrane z zasobu `/limits` interfejsu REST API. Każdy wiersz odpowiada pojedynczemu limitowi i prezentuje jego nazwę, wartość maksymalną oraz wartość pozostałą, co pozwala ocenić stopień wykorzystania zasobu bez logowania do środowiska. Zestawienie obejmuje między innymi *governor limits*, limity wywołań API oraz limity przestrzeni danych. Widok limitów przedstawia Rysunek 10.



Rysunek 10. Widok wykorzystania limitów systemowych pojedynczego środowiska. Źródło: opracowanie własne.

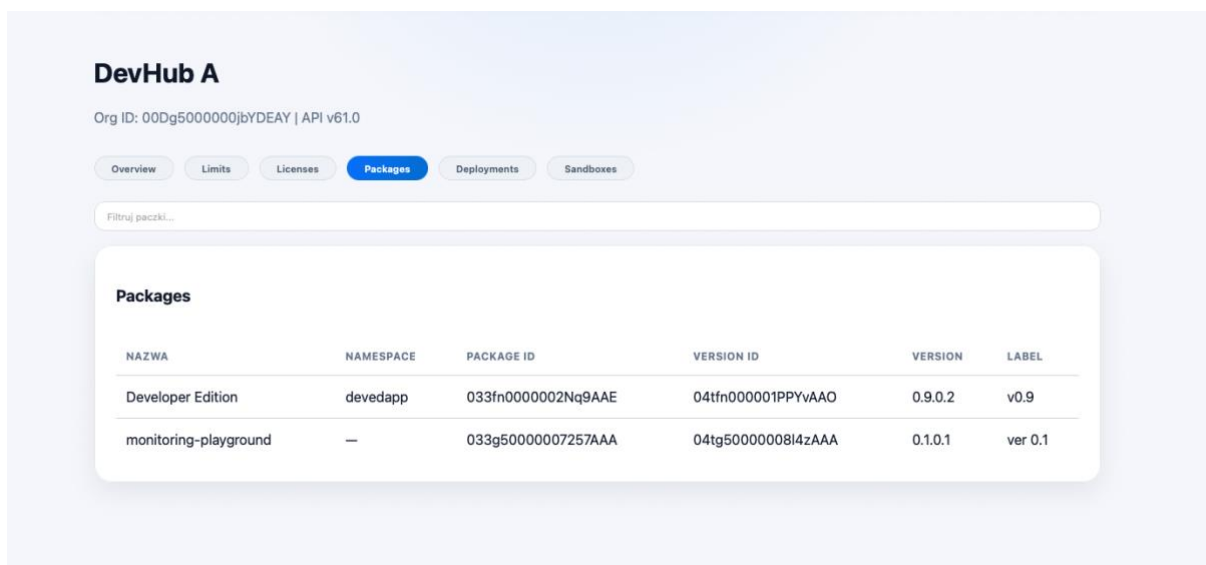
Widok licencji użytkowników prezentuje dane pobrane z obiektu `UserLicense`. Dla każdego typu licencji zestawiono liczbę pozycji przydzielonych i wykorzystanych oraz procent wykorzystania, co umożliwia kontrolę alokacji i wskazanie pozycji niewykorzystanych. Widok licencji przedstawia Rysunek 11.



Rysunek 11. Widok licencji użytkowników środowiska. Źródło: opracowanie własne.

Widok zainstalowanych pakietów prezentuje dane pobrane z obiektu `InstalledSubscriberPackage` za pośrednictwem Tooling API. Dla każdego pakietu

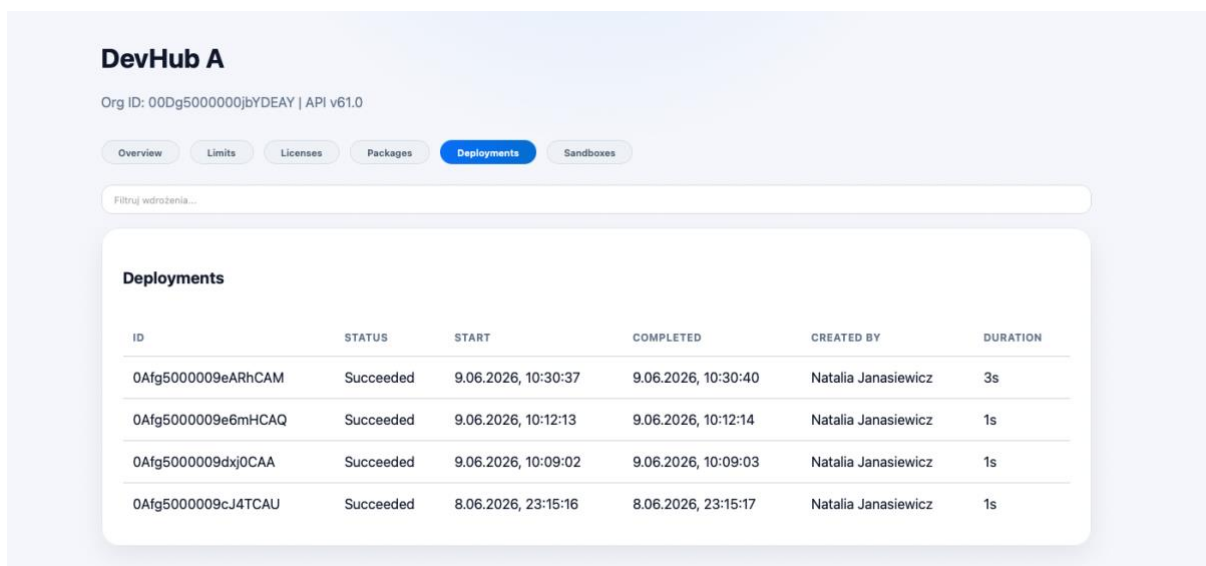
zestawiono jego nazwę, numer zainstalowanej wersji oraz jej id co pozwala porównać wersje pakietów pomiędzy środowiskami oraz kontrolować proces wdrażania funkcjonalności. Widok pakietów przedstawia Rysunek 12.



NAZWA	NAMESPACE	PACKAGE ID	VERSION ID	VERSION	LABEL
Developer Edition	devedapp	033fn0000002Nq9AAE	04tfn000001PPYvAAO	0.9.0.2	v0.9
monitoring-playground	—	033g50000007257AAA	04tg50000008l4zAAA	0.1.0.1	ver 0.1

Rysunek 12. Widok zainstalowanych pakietów środowiska. Źródło: opracowanie własne.

Widok historii wdrożeń prezentuje dane pobrane z obiektu `DeployRequest` za pośrednictwem Tooling API. Każdy wpis opisuje pojedyncze wdrożenie: jego status, daty rozpoczęcia i zakończenia oraz użytkownika inicjującego operację, co umożliwia śledzenie ostatnich zmian wprowadzonych w środowisku. Widok wdrożeń przedstawia Rysunek 13.



ID	STATUS	START	COMPLETED	CREATED BY	DURATION
0Afg5000009eARhCAM	Succeeded	9.06.2026, 10:30:37	9.06.2026, 10:30:40	Natalia Janasiewicz	3s
0Afg5000009e6mHCAQ	Succeeded	9.06.2026, 10:12:13	9.06.2026, 10:12:14	Natalia Janasiewicz	1s
0Afg5000009dxj0CAA	Succeeded	9.06.2026, 10:09:02	9.06.2026, 10:09:03	Natalia Janasiewicz	1s
0Afg5000009cJ4TCAU	Succeeded	8.06.2026, 23:15:16	8.06.2026, 23:15:17	Natalia Janasiewicz	1s

Rysunek 13. Widok historii wdrożeń środowiska. Źródło: opracowanie własne.

Zaimplementowany prototyp realizuje założenia koncepcyjne przyjęte w rozdziale 3. Jego empiryczną weryfikację, obejmującą między innymi sprawdzenie modelu minimalnych uprawnień, przedstawiono w rozdziale 6.

6. Weryfikacja empiryczna

Rozdział szósty przedstawia empiryczną weryfikację prototypu, której celem jest sprawdzenie, czy zrealizowany system spełnia cztery warunki przyjęte w tezie pracy: wyłącznie odczytowy charakter, działanie na minimalnych uprawnieniach, agregację danych z wielu środowisk oraz brak trwałego przechowywania danych. Weryfikację przeprowadzono na rzeczywistych środowiskach platformy Salesforce, a każdy z warunków powiązano z odrębnym testem. Kolejne podrozdziały opisują metodykę, konfigurację środowiska testowego oraz wyniki poszczególnych testów, zakończone podsumowaniem.

6.1. Metodyka weryfikacji

Weryfikację oparto na testach funkcjonalnych i konfiguracyjnych przeprowadzonych na dwóch niezależnych środowiskach Salesforce w wersji Developer Edition. Każdemu warunkowi tezy przypisano konkretny sposób sprawdzenia, dający obserwowalny wynik. Model odczytowy oraz minimalne uprawnienia zweryfikowano przez iteracyjne ustalanie najmniejszego zbioru uprawnień użytkownika technicznego, przy którym kolejne endpointy zwracają dane. Agregację sprawdzono przez zestawienie danych z dwóch środowisk w widoku porównania, a brak warstwy trwałego przechowywania danych przez ponowne uruchomienie procesu serwera. Dodatkowo zweryfikowano mechanizm przełączników jako element sterowania zakresem monitorowania. Powiązanie warunków z metodami weryfikacji przedstawia Tabela 7.

Tabela 7. Powiązanie warunków tezy ze sposobem ich weryfikacji.

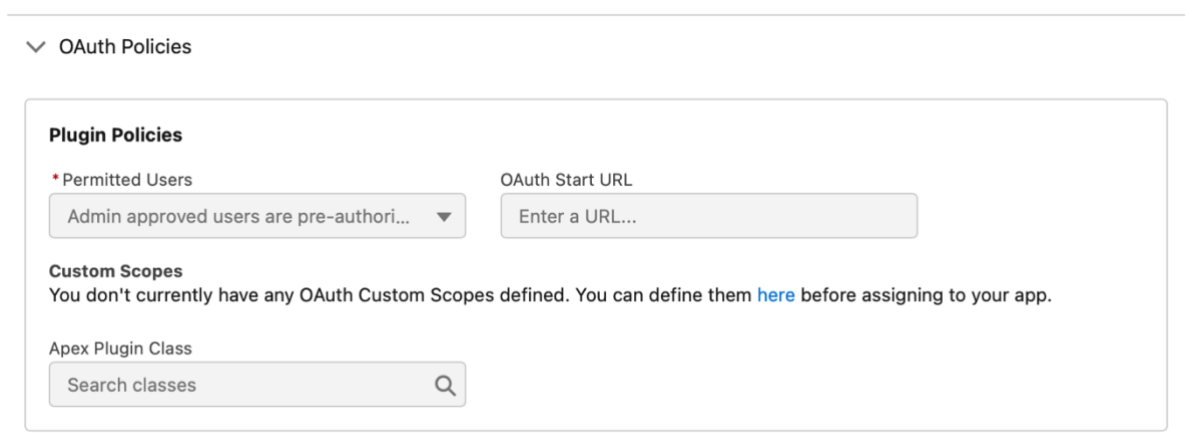
Weryfikowany warunek	Sposób weryfikacji	Podrozdział
Model wyłącznie odczytowy	analiza wykonywanych operacji HTTP oraz test przy podniesionych uprawnieniach	6.3
Minimalne uprawnienia	iteracyjne ustalanie minimalnego zbioru uprawnień użytkownika technicznego	6.3
Agregacja wielu środowisk	zestawienie danych z dwóch środowisk w widoku porównania	6.4
Brak trwałego przechowywania danych	wyłączenie i ponowne uruchomienie procesu serwera	6.5
Sterowanie zakresem (przełączniki)	wyłączanie modułów i funkcji przez zmienne środowiskowe	6.5

6.2. Konfiguracja środowiska testowego

Środowisko testowe stanowiły dwa sprawne orgi Developer Edition, oznaczone dalej jako A i B, wykorzystywane we wszystkich testach funkcjonalnych. W każdym z nich zainstalowano pakiet z platformy AppExchange, aby zapewnić dane dla modułu pakietów. Dodatkowo skonfigurowano trzecie środowisko, oznaczone jako C, z celowo błędną konfiguracją połączenia, służące wyłącznie do

zaprezentowania sposobu obsługi błędu w widoku środowisk. W obu sprawnych środowiskach skonfigurowano osobną aplikację External Client App z własnym identyfikatorem (`clientId`) oraz włączonym uwierzytelnianiem JWT Bearer Flow. Na potrzeby testów wykorzystano wspólny certyfikat, którego klucz prywatny umieszczono wyłącznie po stronie serwera prototypu. Konfiguracja monitorowanych środowisk dopuszcza przypisanie osobnego klucza prywatnego do każdego orga (pole `privateKeyPath` w definicji środowiska), co pozwala na niezależną rotację poświadczeń zgodnie z modelem opisanym w podrozdziale 3.4. W politykach aplikacji ustawiono wstępną autoryzację użytkowników przez administratora (Rysunek 14), a dostęp do aplikacji przypisano poprzez profile użytkowników (Rysunek 15). Wśród przypisanych profili znalazł się profil użytkownika technicznego, oparty na licencji Salesforce; pozostałe profile, w tym System Administrator oraz Salesforce API Only System Integration, dodano na potrzeby testów porównawczych opisanych w podrozdziale 6.3.

W zakresie uprawnień OAuth aplikacji wybrano dostęp do danych przez interfejsy programistyczne (Manage user data via APIs) oraz zakres umożliwiający wykonywanie żądań w dowolnym czasie (`refresh_token`, `offline_access`). Drugi z zakresów okazał się w zastosowanej konfiguracji niezbędny do poprawnego przeprowadzenia przepływu uwierzytelniania, mimo że przepływ JWT Bearer Flow nie posługuje się tokenem odświeżającym w sposób jawny. Pominięto natomiast zakres pełny oraz zakresy tożsamościowe. Wybór zakresów wyznacza jedynie górną granicę możliwości tokenu, natomiast faktyczny dostęp ogranicza zbiór uprawnień przypisany użytkownikowi. Wybrane zakresy przedstawia Rysunek 16.



Rysunek 14. Konfiguracja External Client App: polityki OAuth z wstępną autoryzacją użytkowników. Źródło: opracowanie własne.

Policies Settings Package Defaults

Configure policies to customize the external client app and plugins for this Salesforce organization.

▼ App Policies

* Start Page ⓘ

None ▼

Select Profiles

Available Profiles

- Analytics Cloud Integration User
- Analytics Cloud Security User
- Anypoint Integration
- Authenticated Website

Selected Profiles

- System Administrator
- Minimum Access - Salesforce
- Salesforce API Only System Integrations

Select Permission Sets

Available Permission Sets

- Constraint Rules Engine Licenseless permission set
- Developer Edition
- Experience Profile Manager
- SCRT2 Integration User

Selected Permission Sets

- MonitoringPermSet

Rysunek 15. Konfiguracja External Client App: profile przypisane do aplikacji. Źródło: opracowanie własne.

App Settings

[Consumer Key and Secret](#)

* Callback URL

<http://localhost:4000/auth/sf/callback>

* OAuth Scopes

Available OAuth Scopes

- Access the identity URL service (id, profile, email, address, phone)
- Manage user data via Web browsers (web)
- Full access (full)
- Access Connect REST API resources (chatter_api)
- Access Visualforce applications (visualforce)
- Perform requests at any time (refresh_token, offline_access)
- Access unique user identifiers (openid)

Selected OAuth Scopes

- Manage user data via APIs (api)

☐ Introspect all Tokens

☐ Configure ID token

Flow Enablement

☐ Enable Client Credentials Flow

☐ Enable Authorization Code and Credentials Flow

☐ Enable Device Flow

☒ Enable JWT Bearer Flow

Certificate Upload

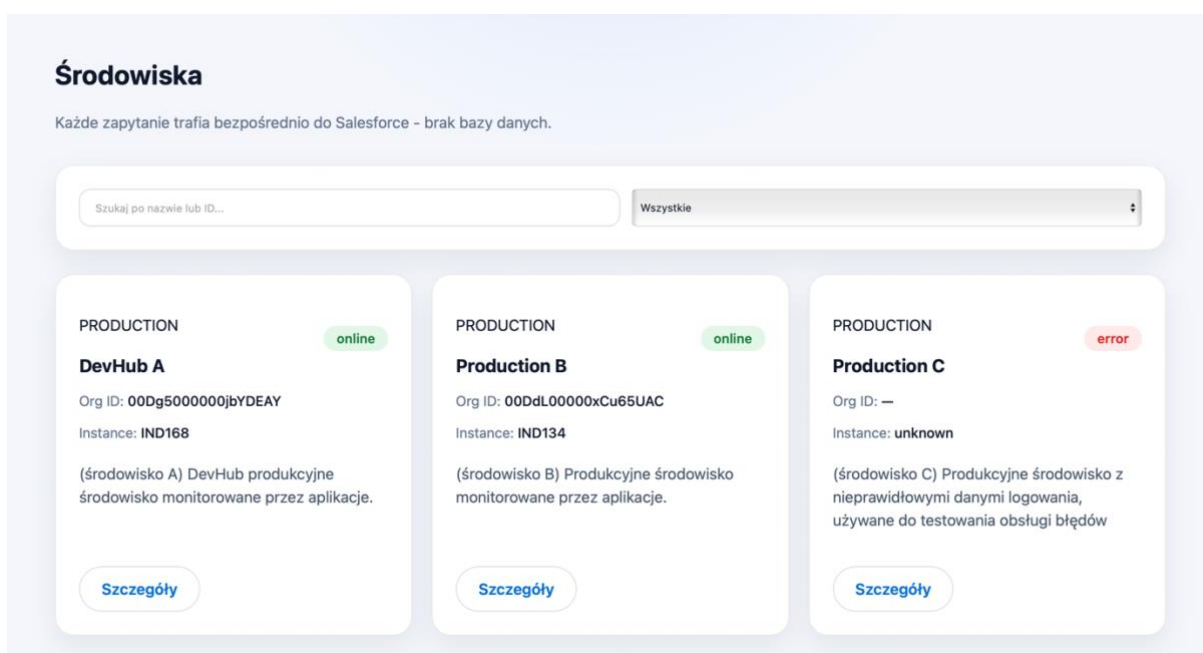
[Upload Files](#) Or drop files

CN=SalesforceMonitoringApp 27 Oct 2026 18:33:56 GMT

☐ Enable Token Exchange Flow

Rysunek 16. Konfiguracja External Client App: wybrane zakresy OAuth. Źródło: opracowanie własne.

Użytkownikowi technicznemu nadano pełną licencję Salesforce oraz profil o najmniejszym zakresie uprawnień, bez profilu administratora. Właściwe uprawnienia przydzielono poprzez dedykowany zbiór uprawnień, co pozwala precyzyjnie kontrolować zakres dostępu i stanowi punkt wyjścia do weryfikacji opisanej w podrozdziale 6.3. Konfiguracja monitorowanych środowisk w prototypie obejmowała wszystkie trzy środowiska. Środowiska A i B były odpytywane poprawnie, natomiast środowisko C celowo zwracało błąd połączenia, co pozwoliło zweryfikować, że niepowodzenie jednego środowiska jest prezentowane niezależnie i nie wpływa na pozostałe. Listę środowisk podłączonych do prototypu, wraz z prezentacją błędu dla środowiska C, przedstawia Rysunek 17.



Rysunek 17. Widok monitorowanych środowisk A, B i C, z prezentacją błędu połączenia dla środowiska C. Źródło: opracowanie własne.

6.3. Weryfikacja modelu odczytowego i minimalnych uprawnień

Weryfikację modelu odczytowego oparto na charakterze operacji wykonywanych przez system. Wszystkie odwołania do platformy realizowane są metodą GET poprzez wspólny mechanizm opisany w podrozdziale 5.3, a interfejs nie udostępnia operacji tworzących ani modyfikujących dane. Istotnym potwierdzeniem okazał się przebieg testu uprawnień: nawet po przyznaniu użytkownikowi technicznemu uprawnienia `Modify All Data`, co nastąpiło w sposób wymuszony i opisano dalej, system nie zmodyfikował żadnych danych, ponieważ nie wykonuje żądań innych niż odczytujące. Model odczytowy jest zatem gwarantowany po stronie aplikacji, niezależnie od zakresu uprawnień nadanych użytkownikowi.

Minimalny zbiór uprawnień ustalano iteracyjnie, rozpoczynając od zestawu zawierającego uprawnienia systemowe `API Enabled` oraz `View Setup and Configuration`, a także prawo odczytu obiektów `Organization` i `UserLicense`. Kluczowe uprawnienia systemowe tego zbioru, `API Enabled` oraz `View Setup and Configuration`, przedstawia Rysunek 18 oraz Rysunek

19. Wcześniejsza próba wykorzystania licencji Salesforce Integration zakończyła się niepowodzeniem, ponieważ przy tej licencji nawet odczyt obiektu Organization zwracał błąd INVALID_TYPE; z tego powodu użytkownikowi technicznemu przypisano pełną licencję Salesforce. Przy opisanym zbiorze uprawnień endpointy udostępniające informacje o organizacji, limity oraz licencje poprawnie zwracały dane.

Apex REST Services	☐	Allow access to Apex REST services
API Enabled	☒	Access any Salesforce.com API.
API Only User	☐	Access Salesforce.com only through a Salesforce.com API.

Rysunek 18. Uprawnienia systemowe minimalnego zbioru: API Enabled. Źródło: opracowanie własne.

View Security Center pages	☐	View the Security Health Check page and the Security Center Dashboard page.
View Setup and Configuration	☒	View the App Setup and Administrative Settings pages.
View Shield App Pages	☐	View the Shield App pages for Data Detect, Field Audit Trail, Platform Encryption and Event Monitoring.

Rysunek 19. Uprawnienia systemowe minimalnego zbioru: View Setup and Configuration. Źródło: opracowanie własne.

Odmienne zachowanie zaobserwowano dla danych pozyskiwanych przez Tooling API. Zapytania o zainstalowane pakiety oraz historię wdrożeń zwracały błąd HTTP 400 z kodem INVALID_TYPE i komunikatem o nieobsługiwany typie obiektu, co oznacza, że obiekty te pozostają niewidoczne dla użytkownika bez podniesionych uprawnień. Nadanie uprawnienia Download AppExchange Packages udostępniło te dane, lecz pociągnęło za sobą włączenie szeregu dalszych uprawnień. Platforma sygnalizuje tę zależność już na etapie konfiguracji - przy uprawnieniu Download AppExchange Packages wyświetlana jest informacja o uprawnieniach wymaganych do jego nadania (Rysunek 20). Po zatwierdzeniu zmiany faktycznie włączony zostaje pełny zbiór uprawnień zależnych, obejmujący między innymi Modify All Data, Customize Application oraz Manage Users (Rysunek 21). Rozróżnienie to wskazuje, że poszerzenie uprawnień nie jest skutkiem konfiguracji systemu monitorującego, lecz wynika z zależności zdefiniowanych po stronie platformy.

 SETUP Permission Sets		
Design and Deliver In-App Guidance	☐	Create, edit, and delete prompts and walkthroughs in Lightning Experience.
Digital Lending Admin User	☐	Allow user access Digital Lending admin specific features
Digital Lending Edit Read Only Party Related Records	☐	Grant access to edit Party related records when Read Only is selected on the corresponding Party record and Turn on read only digital lending party related records.
Digital Lending User	☐	Requires these permissions: Edit HTML Templates, Modify All Data, Customize Application, Manage Users, Manage Letterheads
Digital Lending Workbench	☐	Access digital lending specific features
Dispute Management User	☐	Access dispute management workbench features
Download AppExchange Packages	☒	Install or uninstall AppExchange packages as system administrators.
Download CRM Analytics Data	☐	Download data in tabular format from the CRM Analytics user interface. If the Subscribe to CRM Analytics Assets permission is granted, downloaded table data can be attached to subscription emails.

Rysunek 20. Informacja platformy o uprawnieniach wymaganych przez uprawnienie Download AppExchange Packages. Źródło: opracowanie własne.

☐ Allows access to create Tableau Next workspaces.

Save changes?
✕

Review permission changes, including dependent permissions that must be enabled or disabled as part of this update or previous changes.

Permission Changes and Dependencies

Status	App Permissions	System Permissions	Object Permissions
✓ Enabled	Convert Leads	- Manage Profiles and Permission Sets - Modify Metadata Through Metadata API Functions - Manage Users - Manage Login Access Policies - Create Dashboard Folders - Reset User Passwords and Unlock Users - Manage Translation - Manage Sharing - Manage Letterheads - Manage Internal Users - Edit HTML Templates - Manage Roles - Delete Topics - Create and Customize List Views - Modify All Data - Edit My Dashboards - Manage Password Policies - Edit My Reports - Create and Customize Dashboards - Edit Topics - View Event Log Files - Connect Organization to Environment Hub - Download AppExchange Packages - Freeze Users - Customize Application - Create and Customize Reports - Run Reports - View All Users - Monitor Login History	Read, Create, Edit, Delete, View All, Modify All on all standard and custom objects
		- View Reports in Public Folders - Manage Dashboards in Public Folders - Manage Public Documents - Edit Events - Manage Public Classic Email Templates - View All Data - Transfer Record - Assign Topics - Manage Reports in Public Folders	
		- Manage IP Addresses - Create Report Folders - Assign Permission Sets - Manage Custom Permissions - Edit Tasks - View Dashboards in Public Folders - View Real-Time Event Monitoring Data - Manage Public List Views - Create and Set Up Experiences - Create Topics - View Dashboards in Public Folders - View Real-Time Event Monitoring Data - Manage Public List Views - Create and Set Up Experiences - Create Topics - View Login Forensics Events - Manage MFA in User Interface	
✕ Disabled			

☐ Create, edit, and delete Data Cloud reports.

☐ Unsupported. Use the page layout editor to customize which actions show up in Salesforce and in the Salesforce mobile app.

Rysunek 21 . Uprawnienia faktycznie włączone po nadaniu uprawnienia Download AppExchange Packages. Źródło: opracowanie własne.

Wyniki weryfikacji poszczególnych endpointów przy minimalnym zbiorze uprawnień zestawiono w Tabeli 8.

Tabela 8. Dostępność endpointów przy minimalnym zbiorze uprawnień użytkownika technicznego.

Dane	Interfejs	Wynik przy minimalnym zbiorze uprawnień
Informacje o organizacji	REST API	dostępne
Limity organizacji	REST API	dostępne
Licencje użytkowników	REST API	dostępne
Środowiska sandbox	Tooling API	brak sandboxów w Developer Edition (zapytanie zwraca pusty wynik, nie błąd uprawnień)
Historia wdrożeń	Tooling API	niedostępne, wymaga podniesionych uprawnień
Zainstalowane pakiety	Tooling API	niedostępne, wymaga uprawnienia Download AppExchange Packages

Z weryfikacji wynika, że model minimalnych uprawnień jest spełniony dla większości monitorowanych danych, natomiast dwa endpointy oparte na Tooling API wymagają uprawnień o charakterze administracyjnym, co stanowi ograniczenie narzucone przez platformę, a nie przez projekt systemu. Ograniczenie to można obejść opisanym w podrozdziale 5.4 mechanizmem przełączników, wyłączając moduły pakietów oraz wdrożeń w środowiskach, w których konieczne jest utrzymanie ściśle minimalnych uprawnień. Weryfikację tego mechanizmu opisano w podrozdziale 6.5. Endpoint środowisk typu *sandbox* nie został zweryfikowany pod kątem wymaganych uprawnień, ponieważ środowiska Developer Edition nie pozwalają na tworzenie sandboxów, a obiekt `SandboxInfo` zwracał w nich pusty zbiór rekordów. Ustalenie zakresu uprawnień wymaganego dla tego obiektu na środowisku produkcyjnym pozostaje poza zakresem przeprowadzonej weryfikacji.

6.4. Weryfikacja agregacji wielu środowisk

Spełnienie warunku agregacji zweryfikowano przy użyciu widoku porównania, opisanego koncepcyjnie w podrozdziale 3.6 oraz zaimplementowanego w podrozdziale 5.5. Dla wybranej kategorii danych system pobrał odpowiadające jej wartości równolegle z obu środowisk testowych, a następnie zestawiał je w jednej tabeli, w której wartości z orgów A i B prezentowane są obok siebie.

Test potwierdził, że dane pochodzące z dwóch niezależnych środowisk trafiają do wspólnego zestawienia bez ich trwałego łączenia po stronie platformy. Wynik agregacji w widoku porównania przedstawia Rysunek 22.

Compare

Zaznacz środowiska i wybierz typ danych do porównania.

Wybierz typ

Paczki

☒

DevHub A

production

☒

Production B

production

☐

Production C

production

Porównanie

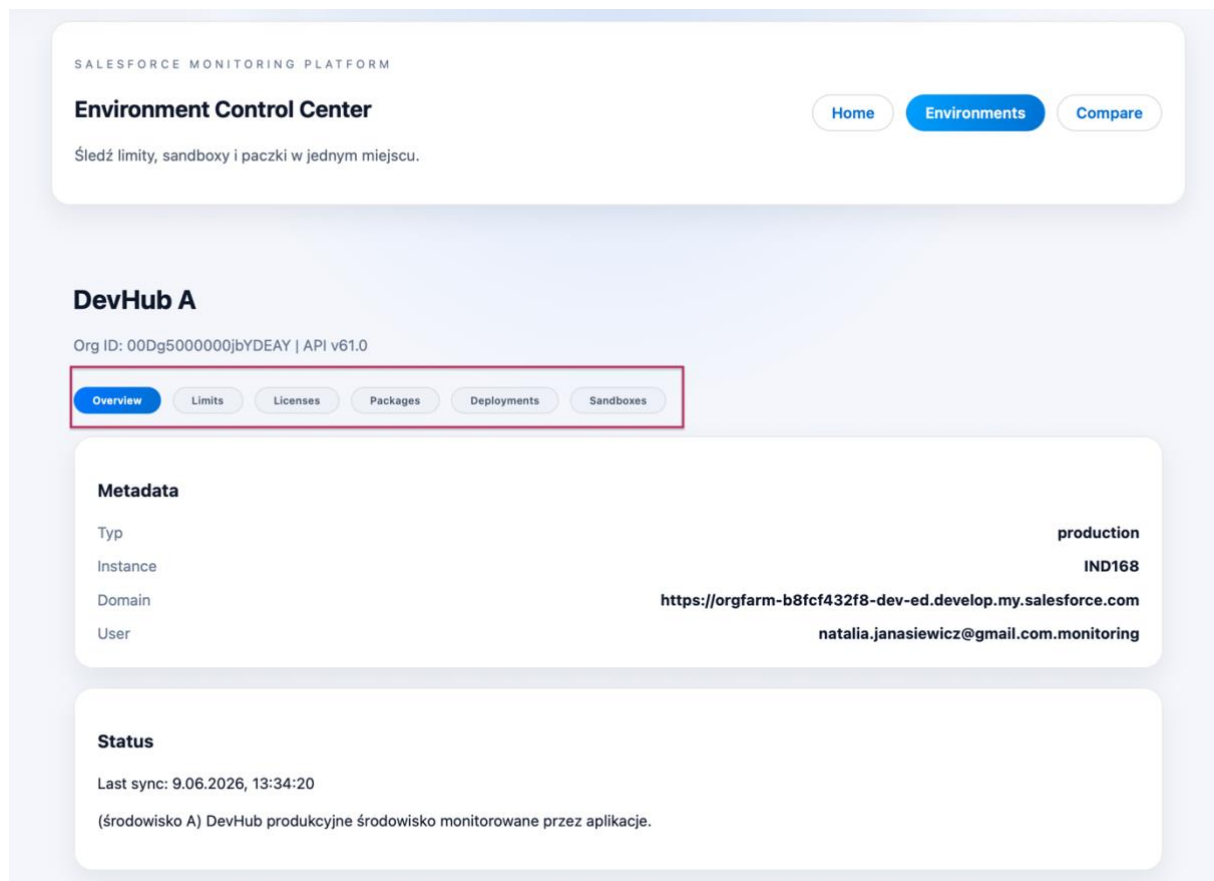
NAZWA	DEVHUB A	PRODUCTION B
Developer Edition	0.9.0.2	0.9.0.2
monitoring-playground	0.1.0.1	—

Rysunek 22. Zestawienie danych ze środowisk A i B w widoku porównania. Źródło: opracowanie własne.

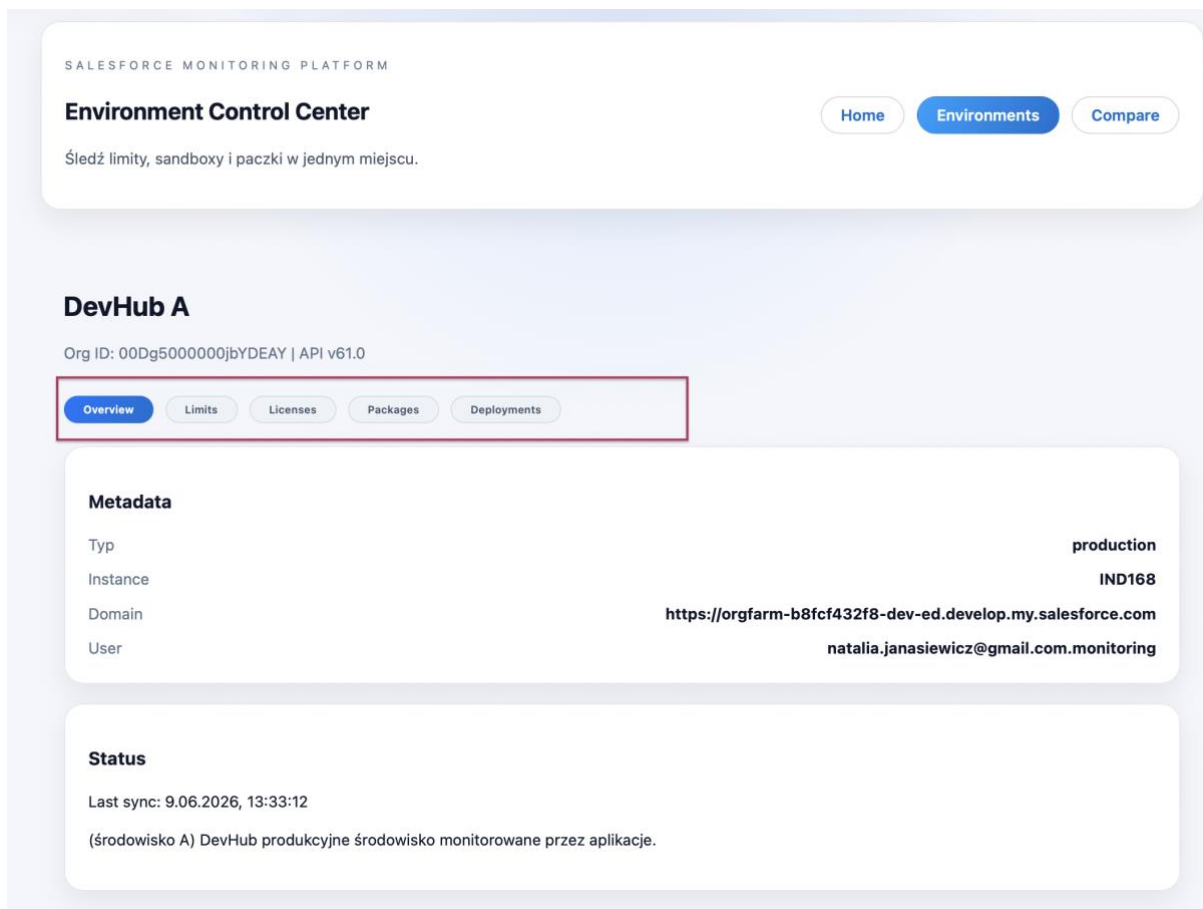
Sposób realizacji agregacji nie zależy od liczby monitorowanych środowisk, ponieważ dane pobierane są niezależnie dla każdego z nich, zgodnie z architekturą opisaną w podrozdziale 4.1. Dodanie kolejnych środowisk nie wymaga zmiany sposobu przetwarzania, a jedynie rozszerzenia konfiguracji. Warunek agregacji danych z wielu środowisk uznano za spełniony.

6.5. Weryfikacja przełączników i braku trwałego przechowywania danych

Mechanizm przełączników, opisany koncepcyjnie w podrozdziale 3.5 oraz zaimplementowany w podrozdziale 5.4, zweryfikowano przez wyłączanie pojedynczych funkcji oraz całych modułów za pomocą zmiennych środowiskowych. Jako przykład wyłączono funkcję obsługi środowisk typu *sandbox*. Funkcja ta jest szczególnie odpowiednia do takiej demonstracji, ponieważ w środowiskach Developer Edition nie można utworzyć środowisk *sandbox*, a więc odpowiadająca jej sekcja i tak nie zawierałaby danych. Po uruchomieniu serwera z wyłączoną funkcją środowisk typu *sandbox* została ona oznaczona jako niedostępna w wykazie funkcji, a interfejs, dzięki opisanemu w podrozdziale 5.6 dostosowaniu do tego wykazu, pominął jej pobieranie i ukrył odpowiadającą jej sekcję w widoku szczegółów środowiska. Stan interfejsu przed wyłączeniem funkcji przedstawia Rysunek 23, a po wyłączeniu Rysunek 24.



Rysunek 23. Interfejs aplikacji przed wyłączeniem funkcji środowisk typu *sandbox*. Źródło: opracowanie własne.



Rysunek 24. Interfejs aplikacji po wyłączeniu funkcji środowisk typu *sandbox*. Źródło: opracowanie własne.

Analogiczny wynik uzyskano przy wyłączeniu całego modułu. Uruchomienie serwera z wyłączonym modułem porównania spowodowało, że jego trasy nie zostały zarejestrowane po stronie serwera, a interfejs ukrył odpowiadającą mu zakładkę. Potwierdza to, że sterowanie zakresem monitorowania działa zarówno na poziomie pojedynczych funkcji, jak i całych modułów, bez modyfikacji kodu.

Brak trwałego przechowywania danych zweryfikowano przez wyłączenie i ponowne uruchomienie procesu serwera. Po ponownym starcie system działał poprawnie bez odtwarzania jakiegokolwiek bazy danych ani zapisanego stanu, pobierając dane na bieżąco z monitorowanych środowisk. Potwierdza to zgodność z modelem bez warstwy trwałego przechowywania danych, przyjętym w podrozdziale 3.3.

6.6. Wnioski

Przeprowadzona weryfikacja wykazała, że prototyp spełnia cztery warunki przyjęte w tezie pracy. Model odczytowy jest gwarantowany po stronie aplikacji, ponieważ system wykonuje wyłącznie operacje GET, niezależnie od zakresu uprawnień nadanych użytkownikowi. Agregacja danych z wielu środowisk działa poprawnie i nie zależy od ich liczby. Brak trwałego przechowywania danych potwierdzono testem ponownego uruchomienia. Spełnienie warunku minimalnych uprawnień ma natomiast charakter warunkowy, omówiony poniżej. Zestawienie wyników przedstawia Tabela 9.

Tabela 9. Podsumowanie weryfikacji warunków tezy.

Warunek tezy	Wynik weryfikacji
Model wyłącznie odczytowy	spełniony, gwarantowany przez wyłączenie użycie metody GET
Minimalne uprawnienia	spełniony dla większości danych, z ograniczeniem dla dwóch endpointów Tooling API
Agregacja wielu środowisk	spełniony, niezależny od liczby środowisk
Brak trwałego przechowywania danych	spełniony, potwierdzony testem ponownego uruchomienia

Najistotniejszym ustaleniem weryfikacji jest ograniczenie modelu minimalnych uprawnień wynikające z konstrukcji platformy. Udostępnienie danych o zainstalowanych pakietach oraz historii wdrożeń wymaga uprawnienia `Download AppExchange Packages`, którego nadanie pociąga za sobą włączenie uprawnień administracyjnych, w tym `Modify All Data` - zarówno zgodnie z informacją platformy o zależnościach, jak i z faktycznie zaobserwowanym zbiorem włączonych uprawnień (podrozdział 6.3). Oznacza to, że pełen zakres monitorowania nie jest możliwy przy ściśle minimalnych uprawnieniach. Praktycznym rozwiązaniem tego kompromisu jest mechanizm przełączników: w środowiskach wymagających najniższego poziomu uprawnień moduły pakietów oraz wdrożeń można wyłączyć, zachowując minimalny zbiór uprawnień dla pozostałych danych. Tym samym mechanizm sterowania zakresem, stanowiący jeden z elementów koncepcji, odpowiada na rzeczywiste ograniczenie wykryte podczas weryfikacji.

7. Podsumowanie

Rozdział siódmy podsumowuje wykonaną pracę, odnosi jej wyniki do tezy oraz wskazuje możliwe kierunki dalszego rozwoju.

7.1. Podsumowanie pracy

Punktem wyjścia pracy był problem monitorowania wielu środowisk platformy Salesforce przez dostawców usług zarządzanych, przy zachowaniu odczytowego charakteru dostępu oraz minimalnych uprawnień. W odpowiedzi na ten problem opracowano koncepcję systemu opartego na zewnętrznym pośredniku działającym wyłącznie w trybie odczytu, bez trwałego przechowywania danych, z uwierzytelnianiem w modelu minimalnych uprawnień oraz mechanizmem wtyczek i przełączników umożliwiającym sterowanie zakresem monitorowania. Koncepcję tę zweryfikowano, budując prototyp złożony z warstwy serwerowej w środowisku Node.js oraz warstwy klienckiej opartej na szkielecie Vue, zintegrowany z platformą poprzez interfejsy REST API i Tooling API.

Zgodnie z założeniami przyjętymi w rozdziale 0, zasadniczym rezultatem pracy jest koncepcja rozwiązania, natomiast prototyp pełni rolę jej empirycznej weryfikacji. Kolejne rozdziały przedstawiły stan wiedzy i narzędzi, koncepcję, zastosowane technologie, implementację oraz weryfikację, prowadząc od sformułowania problemu do potwierdzenia przyjętych założeń.

7.2. Wnioski

Przeprowadzona weryfikacja potwierdziła spełnienie czterech warunków przyjętych w tezie pracy. Odczytowy charakter systemu jest gwarantowany przez wyłączenie stosowania operacji pobierania danych, niezależnie od zakresu nadanych uprawnień. Agregacja danych z wielu środowisk działa poprawnie i nie zależy od ich liczby. Brak trwałego przechowywania danych potwierdzono testem ponownego uruchomienia. Warunek minimalnych uprawnień został spełniony dla większości monitorowanych danych, z udokumentowanym ograniczeniem dotyczącym dwóch kategorii pozyskiwanych przez Tooling API.

Wkład pracy ma dwa wymiary. Pierwszym jest koncepcja monitorowania wielu środowisk Salesforce w modelu odczytowym i minimalnych uprawnień, odróżniająca się od istniejących narzędzi, które koncentrują się na wdrożeniach i zarządzaniu wydaniem lub wymagają szerokiego dostępu do środowisk. Drugim jest ustalenie empiryczne dotyczące ograniczeń modelu uprawnień platformy: wykazano, że udostępnienie danych o pakietach i wdrożeniach wymusza nadanie uprawnień administracyjnych, oraz wskazano mechanizm przełączników jako praktyczne rozwiązanie tego kompromisu. Oba wyniki łącznie potwierdzają tezę pracy.

7.3. Kierunki dalszego rozwoju

Część możliwych rozszerzeń mieści się w przyjętym modelu odczytowym. Należą do nich objęcie monitorowaniem kolejnych kategorii danych i obiektów, rozszerzenie informacji o pakietach o wszystkie wersje dostępne w organizacji zarządzającej pakietami, a nie wyłącznie o pakiety aktualnie zainstalowane, oraz automatyczne powiadomienia o zbliżaniu się do limitów platformy. Rozszerzenia te nie naruszają odczytowego charakteru systemu.

Odrębną grupę stanowi gromadzenie danych historycznych, umożliwiające analizę trendów oraz prowadzenie rejestru zdarzeń. Kierunek ten pozostaje w napięciu z założeniem braku trwałego przechowywania danych i wymagałby wprowadzenia opcjonalnej warstwy przechowywania, przy zachowaniu odczytowego pośrednika dla danych bieżących.

Najdalej idące rozszerzenia wykraczają poza model odczytowy. Możliwość instalowania pakietów w środowiskach testowych w celu ich zrównania ze środowiskiem produkcyjnym lub tworzenia środowisk sandbox na podstawie produkcji wymaga operacji zapisu, a tym samym ponownego rozważenia gwarancji odczytowego charakteru oraz minimalnych uprawnień. Funkcje tego rodzaju powinny stanowić wyraźnie wydzielony, opcjonalny moduł, odseparowany od części monitorującej, aby nie naruszać podstawowych właściwości systemu. Dołączanie wszystkich wymienionych rozszerzeń ułatwia mechanizm wtyczek opisany w podrozdziale 5.4, pozwalający dodawać nowe moduły bez ingerencji w pozostały kod.

8. Bibliografia

- [1]. Salesforce. What is Salesforce? <https://www.salesforce.com/eu/products/what-is-salesforce> [dostęp: 9.06.2026]
- [2]. Salesforce. SaaS (Software as a Service). <https://www.salesforce.com/eu/saas/> [dostęp: 9.06.2026]
- [3]. Salesforce. AppExchange. <https://appexchange.salesforce.com> [dostęp: 9.06.2026]
- [4]. Geeksforgeeks Salesforce Environments. <https://www.geeksforgeeks.org/websites-apps/salesforce-environments/> [dostęp: 9.06.2026]
- [5]. Salesforce Developers. Choose the Right Salesforce Org for the Right Job. <https://developer.salesforce.com/blogs/2024/05/choose-the-right-salesforce-org-for-the-right-job> [dostęp: 9.06.2026]
- [6]. Salesforce Developers. Execution Governors and Limits. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_gov_limits.htm [dostęp: 9.06.2026]
- [7]. Salesforce Help. Licenses Overview. https://help.salesforce.com/s/articleView?id=platform.users_licenses_overview.htm&type=5 [dostęp: 9.06.2026]
- [8]. Salesforce Help. Sandbox Types and Templates. https://help.salesforce.com/s/articleView?id=platform.create_test_instance.htm&type=5 [dostęp: 9.06.2026]
- [9]. Salesforce Developers. Understanding Metadata API. https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/meta_intro.htm [dostęp: 9.06.2026]
- [10]. Salesforce Developers. Select and Enable a Dev Hub Org. https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_setup_enable_devhub.htm [dostęp: 9.06.2026]
- [11]. Salesforce Developers. Scratch Orgs. https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_scratch_orgs.htm [dostęp: 9.06.2026]
- [12]. Salesforce Developers. Managed Packages Types. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_manpkgs_dev.htm [dostęp: 9.06.2026]
- [13]. Salesforce Developers. Developing and Distributing Unmanaged Packages. <https://developer.salesforce.com/docs/atlas.en->

- [us.pkg1_dev.meta/pkg1_dev/package_developing_unmanaged.htm](https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/package_developing_unmanaged.htm) [dostęp: 9.06.2026]
- [14]. Salesforce Developers. Unlocked Packages. https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_unlocked_pkg_intro.htm [dostęp: 9.06.2026]
- [15]. Salesforce Developers. Limits. https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/resources_limits.htm [dostęp: 9.06.2026]
- [16]. Salesforce Developers. Introducing Tooling API. https://developer.salesforce.com/docs/atlas.en-us.api_tooling.meta/api_tooling/intro_api_tooling.htm [dostęp: 9.06.2026]
- [17]. Salesforce Ben. 17 Free Ways to Monitor Your Salesforce Org. <https://www.salesforceben.com/free-ways-to-monitor-your-salesforce-org/> [dostęp: 9.06.2026]
- [18]. BearingPoint. Salesforce Limits Monitor. <https://bearingpoint.services/limits-monitor/> [dostęp: 9.06.2026]
- [19]. OrgLimit Monitor - AppExchange Listing. <https://appexchange.salesforce.com/appxListingDetail?listingId=daa4f8c0-64a9-4454-bec5-b604c032a37c> [dostęp: 9.06.2026]
- [20]. Salesforce Labs. Limits Dashboard - AppExchange. <https://appexchange.salesforce.com> [dostęp: 9.06.2026]
- [21]. Gearset. <https://gearset.com/solutions/observability/> [dostęp: 9.06.2026]
- [22]. Copado. Salesforce DevOps & Cloud Solutions for Teams. <https://www.copado.com/solution-overview/salesforce> [dostęp: 9.06.2026]
- [23]. Flosum - AppExchange Listing. <https://appexchange.salesforce.com/appxListingDetail?listingId=a0N30000000psM0EAI> [dostęp: 9.06.2026]
- [24]. Metazoa. Salesforce Org Monitoring. <https://www.metazoa.com/use-cases-salesforce-org-monitoring/> [dostęp: 9.06.2026]
- [25]. Cloudity. sfdx-hardis Documentation - Monitoring home. <https://sfdx-hardis.cloudity.com/salesforce-monitoring-home/> [dostęp: 9.06.2026]
- [26]. Salesforce. CRM Analytics. <https://www.salesforce.com/eu/analytics/crm/> [dostęp: 9.06.2026]
- [27]. Salesforce Help. OAuth 2.0 JWT Bearer Flow for Server-to-Server Integration. https://help.salesforce.com/s/articleView?id=sf.remoteaccess_oauth_jwt_flow.htm [dostęp: 9.06.2026]

- [28]. Create an External Client App in Your Org
https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_auth_ea.htm [dostęp: 9.06.2026]
- [29]. Node.js. About Node.js. <https://nodejs.org/en/about> [dostęp: 9.06.2026]
- [30]. Express. Express - Node.js web application framework. <https://expressjs.com> [dostęp: 9.06.2026]
- [31]. Microsoft. TypeScript Documentation. <https://www.typescriptlang.org/docs/> [dostęp: 9.06.2026]
- [32]. Salesforce Developers. Introduction to REST API / About REST API.
https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/intro_what_is_rest_api.htm [dostęp: 9.06.2026]
- [33]. Vue.js. Introduction. <https://vuejs.org/guide/introduction.html> [dostęp: 9.06.2026]
- [34]. Pinia. Introduction. <https://pinia.vuejs.org/introduction.html> [dostęp: 9.06.2026]
- [35]. Vue Router. Getting Started. <https://router.vuejs.org/guide/> [dostęp: 9.06.2026]
- [36]. Vite. Getting Started. <https://vite.dev/guide/> [dostęp: 9.06.2026]
- [37]. Zod. Intro. <https://zod.dev/> [dostęp: 9.06.2026]
- [38]. Axios. Getting Started. <https://axios-http.com/docs/intro> [dostęp: 9.06.2026]
- [39]. Auth0. node-jsonwebtoken. <https://github.com/auth0/node-jsonwebtoken> [dostęp: 9.06.2026]

9. Dodatki

Słownik użytej terminologii i skrótów

- API (Application Programming Interface) - interfejs programistyczny umożliwiający komunikację między aplikacjami.
- CI/CD (Continuous Integration / Continuous Delivery) - ciągła integracja i ciągłe dostarczanie oprogramowania.
- CRM (Customer Relationship Management) - system zarządzania relacjami z klientami.
- DTO (Data Transfer Object) - obiekt służący do przekazywania danych między warstwami systemu.
- ECA (External Client App) - aplikacja Salesforce do integracji zewnętrznej, następca mechanizmu Connected App.
- JWT (JSON Web Token) - standard tokenu wykorzystywany w uwierzytelnianiu.
- MSP (Managed Service Provider) - dostawca usług zarządzanych, obsługujący środowiska wielu klientów.

- OAuth - standard autoryzacji dostępu do zasobów bez przekazywania hasła.
- org - organizacja Salesforce, odrębne środowisko platformy.
- PaaS (Platform as a Service) - model udostępniania platformy do budowy aplikacji.
- REST (Representational State Transfer) - styl architektury usług sieciowych.
- SaaS (Software as a Service) - model udostępniania oprogramowania jako usługi.
- sandbox - pomocnicze środowisko Salesforce będące kopią środowiska produkcyjnego.
- SOQL (Salesforce Object Query Language) - język zapytań do obiektów platformy Salesforce.
- SPA (Single Page Application) - aplikacja jednostronicowa.

Spis rysunków

Rysunek 1. Cykl wytwarzania oprogramowania z wykorzystaniem środowisk Salesforce. Źródło: opracowanie własne.	12
Rysunek 2. Ręczne sprawdzenie wykorzystania limitów w interfejsie platformy Salesforce. Źródło: opracowanie własne	16
Rysunek 3. Pulpit monitorowania limitów w aplikacji BearingPoint Limits Monitor ze sklepu AppExchange. Źródło: [18].	17
Rysunek 4. Pulpit modułu "Error and org limits dashboard" platformy Gearset. Źródło: [21].	18
Rysunek 5. "Monitoring config workbench" z widokiem na "Org Info" oraz "Licenses & Packages" w narzędziu sfdx-hardis. Źródło: [25].	19
Rysunek 6. Architektura klient-serwer systemu monitorowania: aplikacja kliencka, serwerowy pośrednik oraz monitorowane środowiska Salesforce. Źródło: opracowanie własne.	31
Rysunek 7. Struktura katalogów prototypu (wybrane pliki). Źródło: opracowanie własne.	36
Rysunek 8. Cykl życia żądania w architekturze pośrednika: od warstwy klienckiej do platformy Salesforce i z powrotem. Źródło: opracowanie własne.	37
Rysunek 9. Warstwy backendu i rola rejestru modułów w warunkowym montowaniu tras. Źródło: opracowanie własne.	42
Rysunek 10. Widok wykorzystania limitów systemowych pojedynczego środowiska. Źródło: opracowanie własne.	46
Rysunek 11. Widok licencji użytkowników środowiska. Źródło: opracowanie własne.	46
Rysunek 12. Widok zainstalowanych pakietów środowiska. Źródło: opracowanie własne. ...	47
Rysunek 13. Widok historii wdrożeń środowiska. Źródło: opracowanie własne.	47
Rysunek 14. Konfiguracja External Client App: polityki OAuth z wstępną autoryzacją użytkowników. Źródło: opracowanie własne.	50
Rysunek 15. Konfiguracja External Client App: profile przypisane do aplikacji. Źródło: opracowanie własne.	51
Rysunek 16. Konfiguracja External Client App: wybrane zakresy OAuth. Źródło: opracowanie własne.	51
Rysunek 17. Widok monitorowanych środowisk A, B i C, z prezentacją błędu połączenia dla środowiska C. Źródło: opracowanie własne.	52
Rysunek 18. Uprawnienia systemowe minimalnego zbioru: API Enabled. Źródło: opracowanie własne.	53
Rysunek 19. Uprawnienia systemowe minimalnego zbioru: View Setup and Configuration. Źródło: opracowanie własne.	53

Rysunek 20. Informacja platformy o uprawnieniach wymaganych przez uprawnienie Download AppExchange Packages. Źródło: opracowanie własne.....	53
Rysunek 21 . Uprawnienia faktycznie włączone po nadaniu uprawnienia Download AppExchange Packages. Źródło: opracowanie własne.	54
Rysunek 22. Zestawienie danych ze środowisk A i B w widoku porównania. Źródło: opracowanie własne.	56
Rysunek 23. Interfejs aplikacji przed wyłączeniem funkcji środowisk typu <i>sandbox</i> . Źródło: opracowanie własne.	57
Rysunek 24. Interfejs aplikacji po wyłączeniu funkcji środowisk typu <i>sandbox</i> . Źródło: opracowanie własne.	58

Spis tabel

Tabela 1. Interfejsy programistyczne platformy Salesforce.....	14
Tabela 2. Spełnienie założeń projektowych przez reprezentatywne podejścia do monitorowania środowisk Salesforce.....	29
Tabela 3. Stos technologiczny prototypu w podziale na warstwy systemu.	31
Tabela 4. Kategorie monitorowanych danych i przypisane im interfejsy programistyczne platformy Salesforce.	32
Tabela 5. Biblioteki warstwy klienckiej i ich rola w systemie.....	33
Tabela 6. Narzędzia wspierające proces wytwarzania prototypu.	34
Tabela 7. Powiązanie warunków tezy ze sposobem ich weryfikacji.	49
Tabela 8. Dostępność endpointów przy minimalnym zbiorze uprawnień użytkownika technicznego.....	55
Tabela 9. Podsumowanie weryfikacji warunków tezy.....	59

Spis listingów

Listing 1. Schemat walidacji definicji środowiska z użyciem biblioteki Zod.	37
Listing 2. Budowa i wymiana tokenu JWT na token dostępowy.....	38
Listing 3. Wspólny mechanizm odczytu danych dla REST API i Tooling API.	39
Listing 4. Warunkowa rejestracja modułów na podstawie ich stanu włączenia.	40
Listing 5. Struktura deskryptora modułu oraz rejestracja przykładowego modułu monitorującego.....	40
Listing 6. Serwis monitorujący wdrożenia z transformacją do modelu wewnętrznego.....	43
Listing 7. Równoległe pobranie i scalenie danych z wielu środowisk.....	44
Listing 8. Pobranie wykazu modułów i pominięcie funkcji wyłączonych po stronie klienta..	45