2019

Pubweb - dokumentacja

GRUPA

DUBIŃSKI JAN
GRĘBOWSKI ŁUKASZ
KALETA JOANNA
OGONOWSKI ALEKSANDER
ONISZCZUK MARIA
SZYMCZYK KORNEL
WALKOWIAK PAWEŁ

Spis treści

1. Wymagania

W poniższym fragmencie opisano w sposób spriorytetyzowany wymagania funkcjonalne i niefunkcjonalne systemu. Przy opisie priorytetów realizacji poszczególnych wymagań posłużono się opisaną dalej metodą MoSCoW.

1.1. Opis Metody MoSCoW

Metoda **MoSCoW** jest techniką priorytetyzacji wykorzystywaną w analizie biznesowej i przy tworzeniu oprogramowania w celu osiągnięcia wspólnego zrozumienia pomiędzy interesariuszami co do znaczenia jakie ma dla nich dostarczenie każdego z wymagań. Wyróżnia się następujące kategorie wymagań według metody MoSCoW:

- **M** – MUST (*musi być*): Opisuje wymaganie, które *musi być* spełnione w końcowym, finalnym rozwiązaniu
- **S** – SHOULD (*powinien być*): Reprezentuje pozycję o wysokim priorytecie, która *powinna być* zawarta w rozwiązaniu, jeżeli jest to możliwe
- **C** – COULD (*może być*): Opisuje wymaganie, które jest postrzegane jako pożądane, ale niekonieczne. Zostanie ono zawarte, jeżeli pozwolą na to czas i zasoby
- **W** – WON'T (*nie będzie*): Reprezentuje wymaganie, które – za zgodą interesariuszy – *nie będzie* implementowane w danym wydaniu, ale może być rozpatrzone w przyszłości

1.2. Wymagania funkcjonalne

MUST:

- system umożliwia utworzenie konta umożliwiającego korzystanie z usług systemu
- system umożliwia logowanie
- system umożliwia dodawanie recenzji pubów
- system umożliwia dodawanie nowych pubów
- użytkownik ma możliwość wyszukania w bazie pubów na podstawie nazw lub zadanych słów kluczowych, lub miasta

SHOULD:

- system umożliwia personalizacje profilu użytkownika
- system umożliwia komentowanie recenzji
- system umożliwia dodawanie zdjęć pubów
- system umożliwia dodanie pubu do listy do odwiedzenia
- system przedstawia rekomendacje pubów

COULD:

- system umożliwia dodawanie znajomych
- system zawiera system rekomendacji pubów
- system prowadzi dziennik aktywności użytkownika

WON'T:

- system umożliwia wyszukiwanie pubów w wybranej lokalizacja
- system wyświetla lokalizację pubów na mapie
- system umożliwia logowanie za pomocą innych platform

1.3. Wymagania niefunkcjonalne

MUST:

- system opiera się na architekturze rozproszonej w postaci mikroserwisów
- struktura systemu umożliwia dodawanie nowych funkcjonalności i dalszy rozwój
- system musi zapewnić skalowalność
- system jest zgodny z RODO [Patrz: Analiza zgodności z RODO]
- wszyscy użytkownicy mogą zgłaszać błędy na dedykowany adres email
- system spełnia wymagania bezpieczeństwa [Patrz: Kwestie bezpieczeństwa]
    - system implementuje protokół OAuth 2.0
- w systemie implementowane są następujące role różniące się poziomami uprawnień
    - niezalogowany użytkownik
    - użytkownik
    - administrator
- system powinien realizować strukturę zabezpieczeń w dostępie do jego zasobów w oparciu o System uprawnień nadawanych Rolom
- interfejs klienta otwiera się w przeglądarkach:
    - Chrome 68 +
    - Mozilla 61 +

SHOULD:

- Interfejs klienta jest dostosowany do urządzeń mobilnych
- System zapewnia najlepszy możliwy User Experience
- Językiem systemu jest język polski
- W aplikacja pola obowiązkowe są oznaczone w inny sposób niż pola nieobowiązkowe

COULD:

- w systemie implementowana jest oddzielna rola o poziomie uprawnień większym niż użytkownik dla użytkownika-właściciela pubu
- System cenzuruje słowa wulgarne

WON’T:

- System posiada angielską wersję językową
- System posiada dedykowaną aplikacje mobilną

2. Postanowienia ogólne

2.1. Ogólna architektura serwisów

Mikroserwisy pisane są w Javie i Pythonie. Interfejsy serwisów opisane są w poniższym dokumencie, dodatkowo dla serwisu autoryzacyjnego, użytkowników, zgód i znajomych wygenerowane zostały specyfikacje w formacie OPENAPI - *https://swagger.io/specification/* . Do uwierzytelnienia w serwisach wykorzystywane są tokeny JWT. Ich działanie opisuje dokument JWT.docx.

Wyróżniamy serwisy:

User service – serwis udostępniający operacje i informacje o użytkownikach. (Java)

Permission service – serwis udostępniający operacje i informacje o zgodach. (Java)

Friend service – serwis udostępniający informacje i operacje dotyczące relacji pomiędzy poszczególnymi użytkownikami. (Java)

Pub service – serwis udostępniający operacje i informacje o pubach. (Java)

Rating service – serwis udostępniający informacje i operacje dotyczące ocen i recenzji pubów tworzonych przez użytkowników.

3. Operacje

3.1. Serwis autoryzacyjny

Ścieżka względna	Metoda HTTP	Znaczenie	Typ przyjmowany	Typ zwracany	Wymagane uprawnienia	Informacje dodatkowe
/auth/check	GET	Zwraca informacje o tokenie JWT	{}, token w headerze Authorization	CheckAuthRespons e	Brak	
/auth/signin	POST	Przyjmuje login i hasło użytkownika w celu wygenerowania tokenu JWT	LoginRequest	jwtAuthenticationRe sponse	Brak	

3.2. Serwis użytkowników

Ścieżka względna	Metoda HTTP	Znaczenie	Typ przyjmowany	Typ zwracany	Wymagane uprawnienia	Informacje dodatkowe
/users	GET	Pobiera listę użytkowników i ich danych publicznych	{}	[GetUserResponse], tylko informacje widoczne w kontekście JWT z zapytania	Zalogowany	Stronnicowanie
/users/{id}	GET	Pobiera dane konkretnego użytkownika		GetUserResponse, tylko informacje widoczne w kontekście JWT z zapytania	Zalogowany	

/users	POST	Dodaje nowego użytkownika	SignUpRequest	GetUserResponse	Brak	
/users/{id}	DELETE	Usuwa dane użytkownika o podanym id	{}	{}	Kontekst własny	
/users/{id}/displaySettings	PUT	Aktualizuje dane wyświetlania użytkownika	UDisplaySettings	GetUserResponse	Kontekst własny	
/users/{id}/personalInformation	PUT	Aktualizuje dane osobiste użytkownika	UserPersonalInfoDto	GetUserResponse	Kontekst własny + zgoda	Zgoda o przetwarzaniu danych osobowych
/users/{id}/friends	GET	Pobiera dane o znajomościach użytkownika	{}	FriendshipInfo[]	Zalogowany	
/users/{id}/permission	GET	Pobiera dane o zgodach (aktywnych), które zaakceptował dany użytkownik	{}	AcceptedPermission[]	Kontekst własny	
/users/{id}/avatar	GET	Pobiera avatar/zdjęcie użytkownika	{}	String z awatarem w Base64	Zalogowany	
/users/{id}/avatar	POST/PUT	Dodaje avatar/zdjęcie użytkownika	AddAvatarReqeust	{}	Kontekst własny	
/users/{id}/avatar	DELETE	Usuwa avatar/zdjęcie użytkownika		{}	Kontekst własny	

3.3. Serwis znajomych

Ścieżka względna	Metoda HTTP	Znaczenie	Typ przyjmowany	Typ zwracany	Wymagane uprawnienia	Informacje Dodatkowe
/requests	GET	Pobiera informacje o wszystkich zaproszeniach do znajomych	{}	Lista zaproszeń do znajomych	Kontekst własny	
/requests/sent	GET	Pobiera informacje o zaproszeniach wysłanych	{}	Lista zaproszeń wysłanych	Kontekst własny	
/requests/received	GET	Pobiera informacje o zaproszeniach wychodzących	{}	Lista zaproszeń otrzymanych	Kontekst własny	
/requests/{id}/confirm	POST	Potwierdza dodanie do znajomych	{}	Informacje o potwierdzonym zaproszeniu	Kontekst własny	Tylko zaproszenia przychodzące do danego użytkownika
/requests/{id}/cancel	POST	Odrzuca zaproszenie do znajomych	{}	Informacje o odrzuconym zaproszeniu	Kontekst własny	Tylko zaproszenia przychodzące do danego użytkownika

3.4. Serwis zgód

Ścieżka względna	Metoda HTTP	Znaczenie	Typ przyjmowany	Typ zwracany	Wymagane uprawnienia	Informacje Dodatkowe
/permissions	GET	Pobiera wszystkie informacje o zgodach	{}	permissionInfo[]	Brak	
/permissions/{id}	GET	Pobiera dane jednej ze zgód	{}	PermissionInfo	Brak	
/permission/accepted	GET	Pobiera dane o zaakceptowanych zgodach	{}	AcceptedPermission[]	Kontekst własny	
/permission/{id}/accept	POST	Akceptuje zgodę o danym id	{}	AcceptedPermission[] – lista po akceptacji	Kontekst własny	
/permission/{id}/revoke	POST	Wycofuje zgodę o danym id	{}	AcceptedPermission[] – lista po wycofaniu akceptacji	Kontekst własny	

3.5. Serwis pubów

Ścieżka względna	Metoda HTTP	Znaczenie	Typ przyjmowany	informacje zwrotne	Błędy	Uprawnienia
/pubs	GET	lista pubow	{}	BasicIPubsInfo []		
/pubs/{id}	GET	info szczegolowe	{}	SpecifiicPubInfoWithTagList	IdNotFound	
/pubs	POST	dodaje pub	AddPubName	PubName		Zalogowany

/pubs/{id}/info	PATCH	dodaje opis pubu	AddPubInfo	PubInfo	IdNotFound	Zalogowany
/pubs/{id}/info	GET	wyswietla opis	{}	PubInfo	IdNotFound	
/pubs/{id}/city	PATCH	dodaje miasto	AddPubCity	PubCity	IdNotFound	Zalogowany
/pubs/{id}/city	GET	wyswietla miasto	{}	PubCity	IdNotFound	
/pubs/{id}	DELETE	usuwa pub	{}	DeletetedPubName	IdNotFound	Admin
/pubs/{id}/tag	GET	Wyświetla tagi	{}	TagInfoAndId []IdNotFound	IdNotFound	
/pubs/tag	GET	Puby z danym tagiem	TagName	BasicIPubsInfo []	TagNotFound	
/pubs/city	GET	Puby w mieście	CityName	BasicIPubsInfo []	CitydNotFound	
/pubs/{id}/photo	POST	Dodaje zdjecie	AddPhotoFile	AddedConfirmation	UploadError	Zalogowany
/pubs/{id}photo	GET	Pobiera zdjecia w formacie Base64	{}	Base64Photo []	IdNotFound	
/pubs/check	GET	Zwraca istnienie pubu	PubId	Boolinfo		

3.6. Serwis recenzji

Ścieżka względna	Metoda HTTP	Działanie	Parametry	Informacje zwrotne	Błędy	Uprawnienia
/reviews	GET	lista ID recenzji I ID pubów, których dotyczą	{}	IDReviewIDPub[]		
/reviews/{id}	GET	info szczegolowe	{}	SpecificReview	IdNotFound	
/ reviews/{id}/opinion	PUT	dodaje tekst recenzji	AddOpinion	Opinion	IdNotFound	
/ reviews/{id}/opinion	GET	wyswietla tekst recenzji	{}	Opinion	IdNotFound	
/ reviews/{id}/stars	PUT	dodaje ocenę	AddStars	Stars	IdNotFound	
/ reviews /{id}/city	GET	wyświetla ocenę	{}	Stars	IdNotFound	
/ reviews/{id}/user_id	PUT	dodaje ID użytkownika autora recenzji	AddUserId	UserId	IdNotFound	
/ reviews /{id}/user_id	GET	wyświetla ID użytkownika autora recenzji	{}	UserId	IdNotFound	
/ reviews/{id}/pub_id	PUT	dodaje ID pubu do recenzji	AddPubId	PubId	IdNotFound	

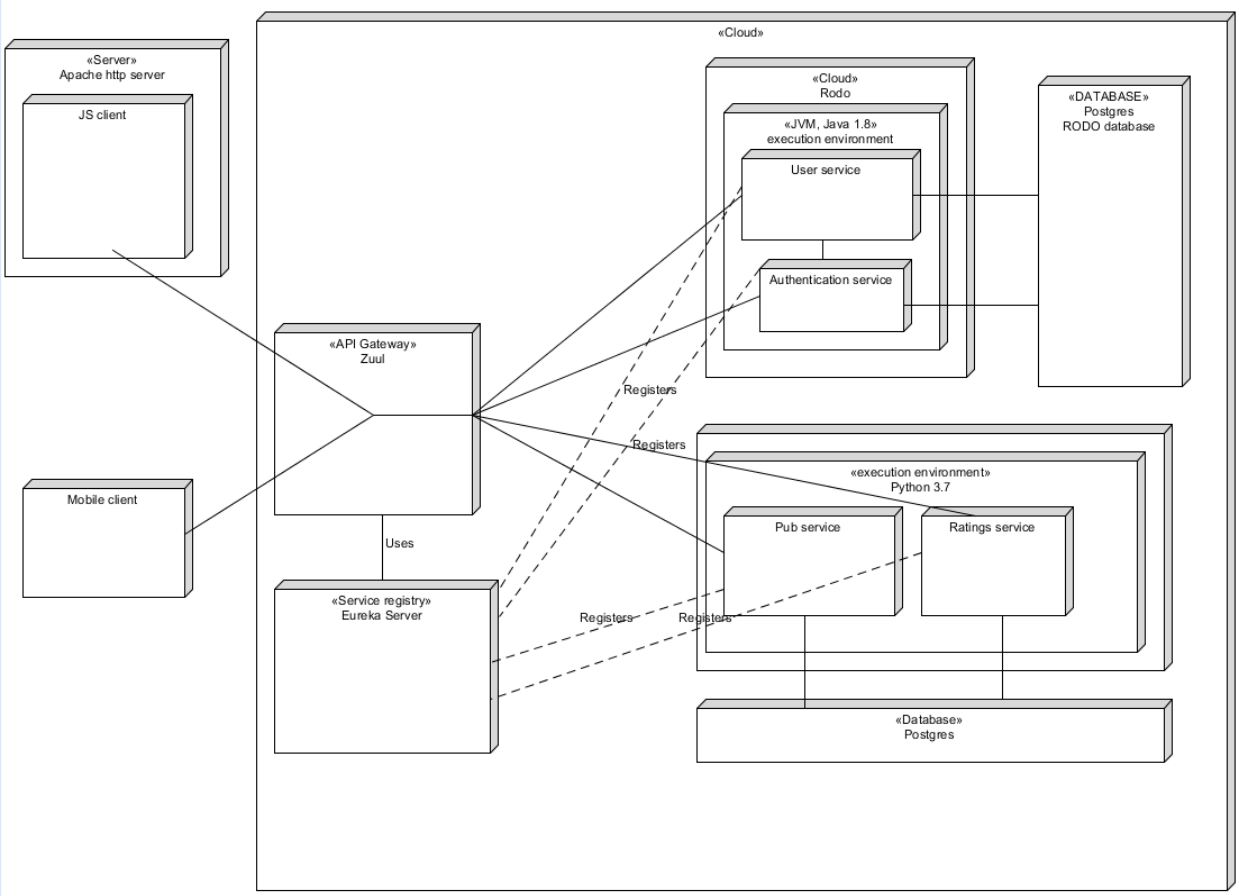
/ reviews /{id}/pub_id	GET	wyświetla ID recenzowanego pubu	{}	PubId	IdNotFound	
/reviews/{id}	DELETE	usuwa recenzje	{}	DeletedId	IdNotFound	

4. Ogólny opis rozwiązań architektonicznych

W celu zrealizowania celów stawianych systemowi zdecydowaliśmy się wprowadzić architekturę opartą na mikroserwisach. Dają one możliwość konteneryzacji poszczególnych części aplikacji, a w efekcie wdrożenie w środowiskach chmurowych jest nieskomplikowane. Wzorce architektoniczne opisane są przy diagramie fizycznym.

5. 4 Views

5.1. Widok fizyczny rozwiązania



Zuul – jest to proxy ze stajni Netflixa. Zapewnia jeden wspólny interfejs dla wszystkich serwisów (realizacja wzorca *API gateway* nazywanego dalej bramą API). Pozwala ukryć implemntację i rzeczywiste rozmieszczenie serwisów. Klient aplikacji zna jedynie adres bramy API. Daje to możliwość zarówno łatwiejszego dostęp do API jak i oddzielenie interfejsów oraz implementacji. Możliwe jest również zdefiniowanie ustawień

bezpieczeństwa (w omawianym przypadku niewykorzystywane, zarządzanie nimi odbywa się na poziomie aplikacji).

Eureka – jest to serwer *service discovery*. W przypadku mikroserwisów ręczne zmiany konfiguracji w momencie pojawienia się nowej instacji lub przeniesienia instniejącej instancji mikroserwisu byłyby żmudne. Eureka jest rejestrem serwisów, który wystawia API REST, dzięki czemu mikroserwis sam może zarejestrować swoją obecność pod wskazanym adresem (np. przy uruchomieniu).

Współpraca Zuul i Eureka:

Ze względu na wspólnego dostawcę Zuul i Eureka posiadają mechanizm integracji. Zuul odczytuje rejestr Eureki i trasuje zapytania do poszczególnych serwisów.

5.2. Widok logiczny, diagram klas

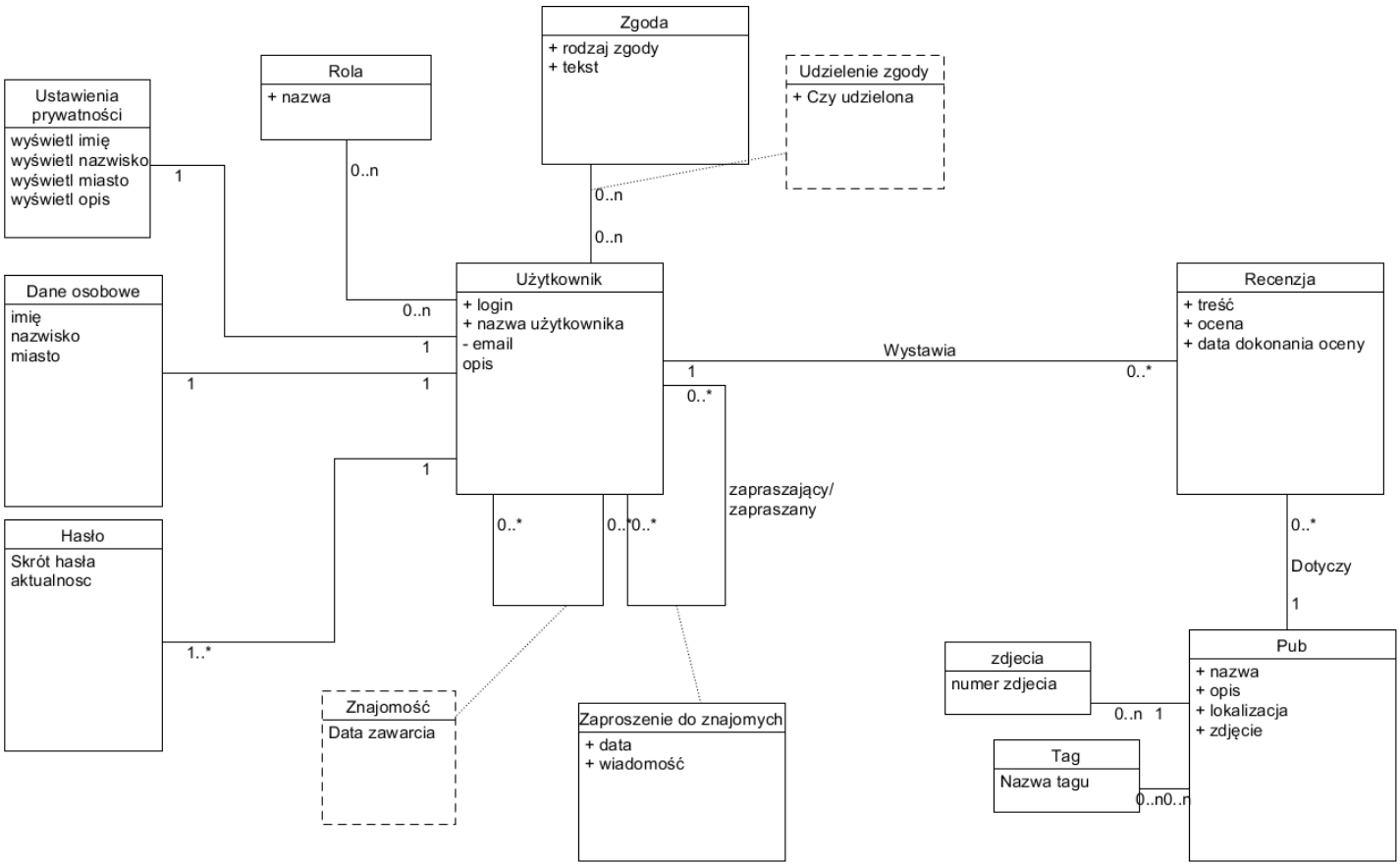
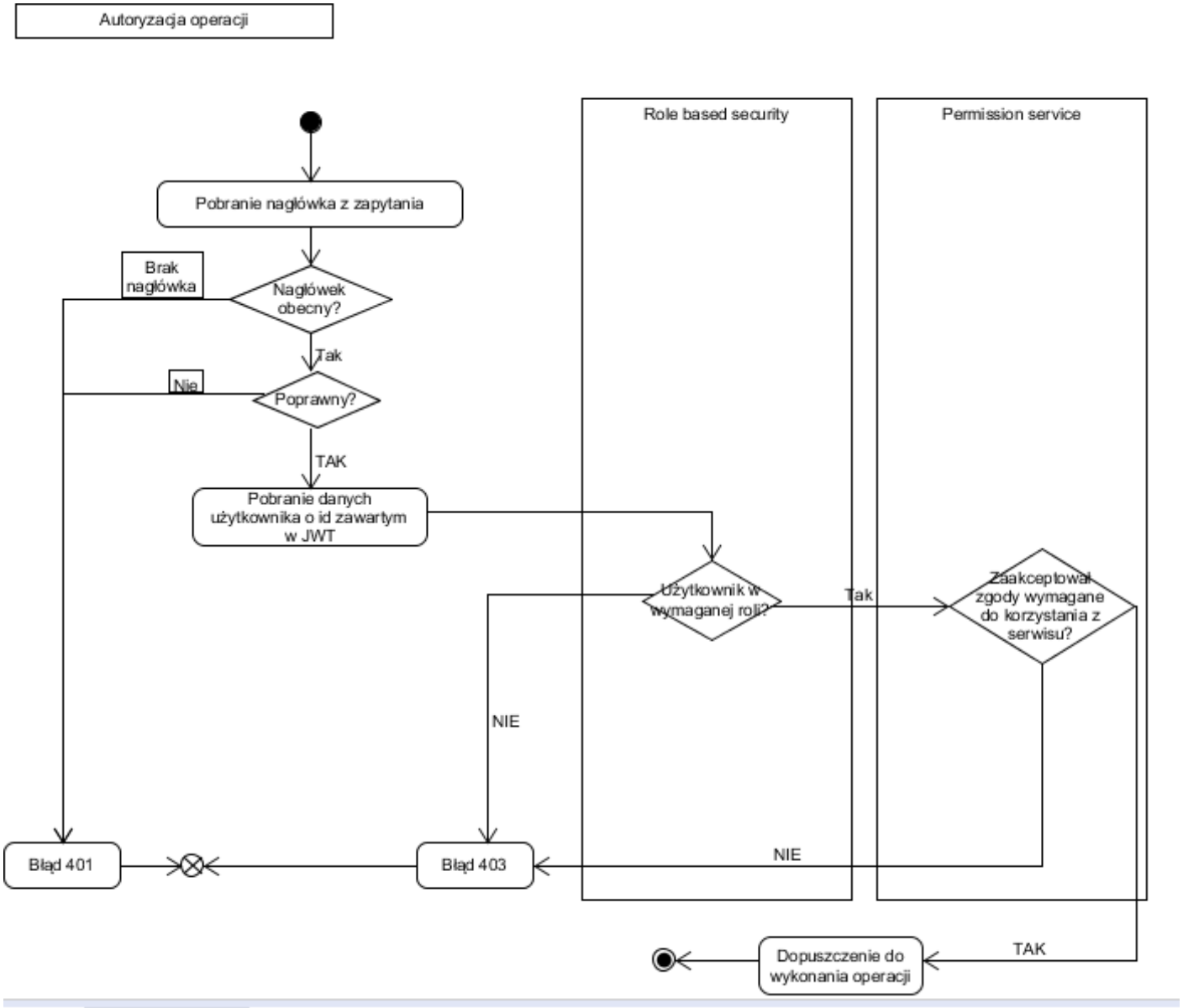


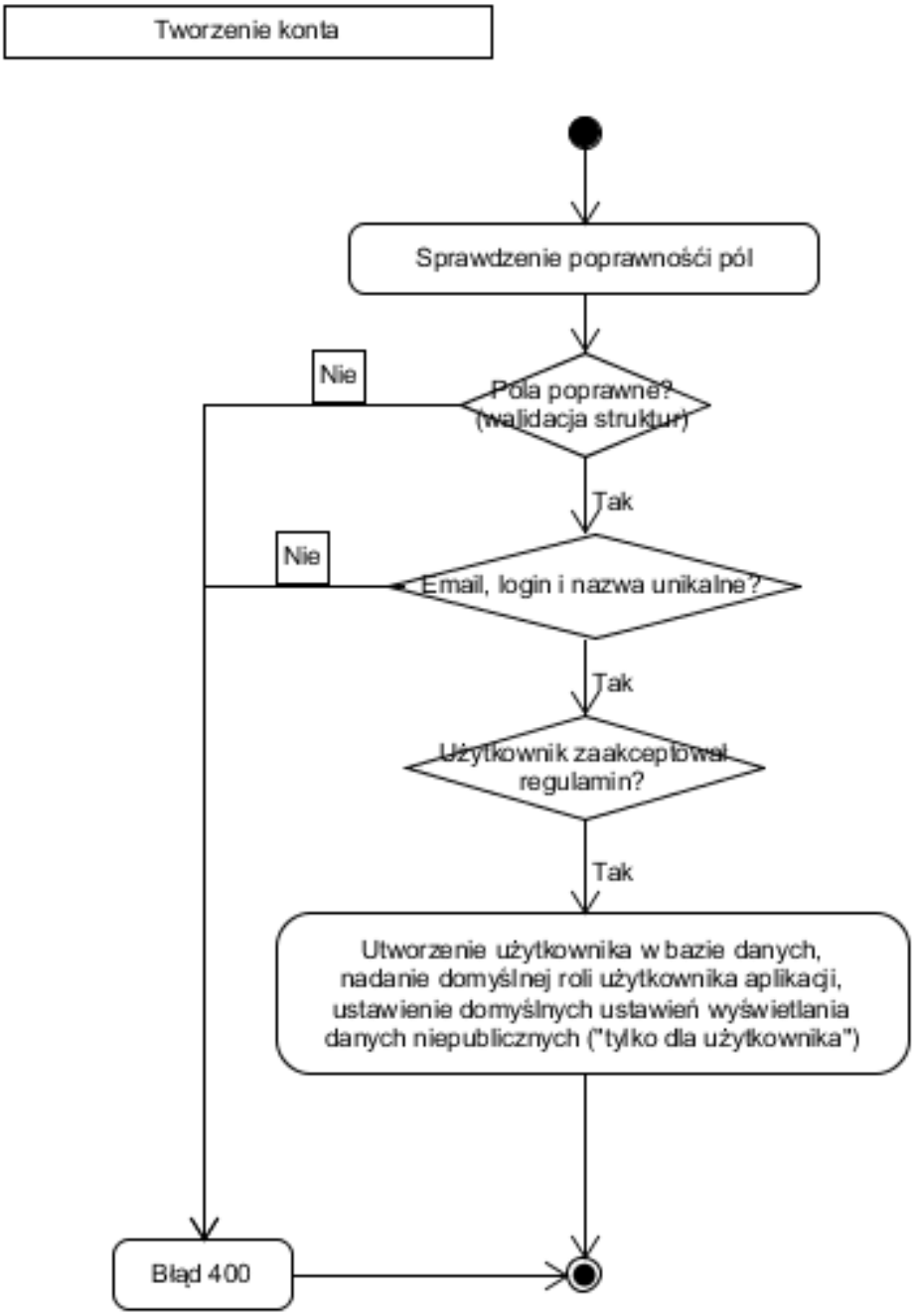
Diagram klas obrazuje logiczne zależności pomiędzy poszczególnymi komponentami systemu.

6. Procesy

6.1. Diagramy aktywności

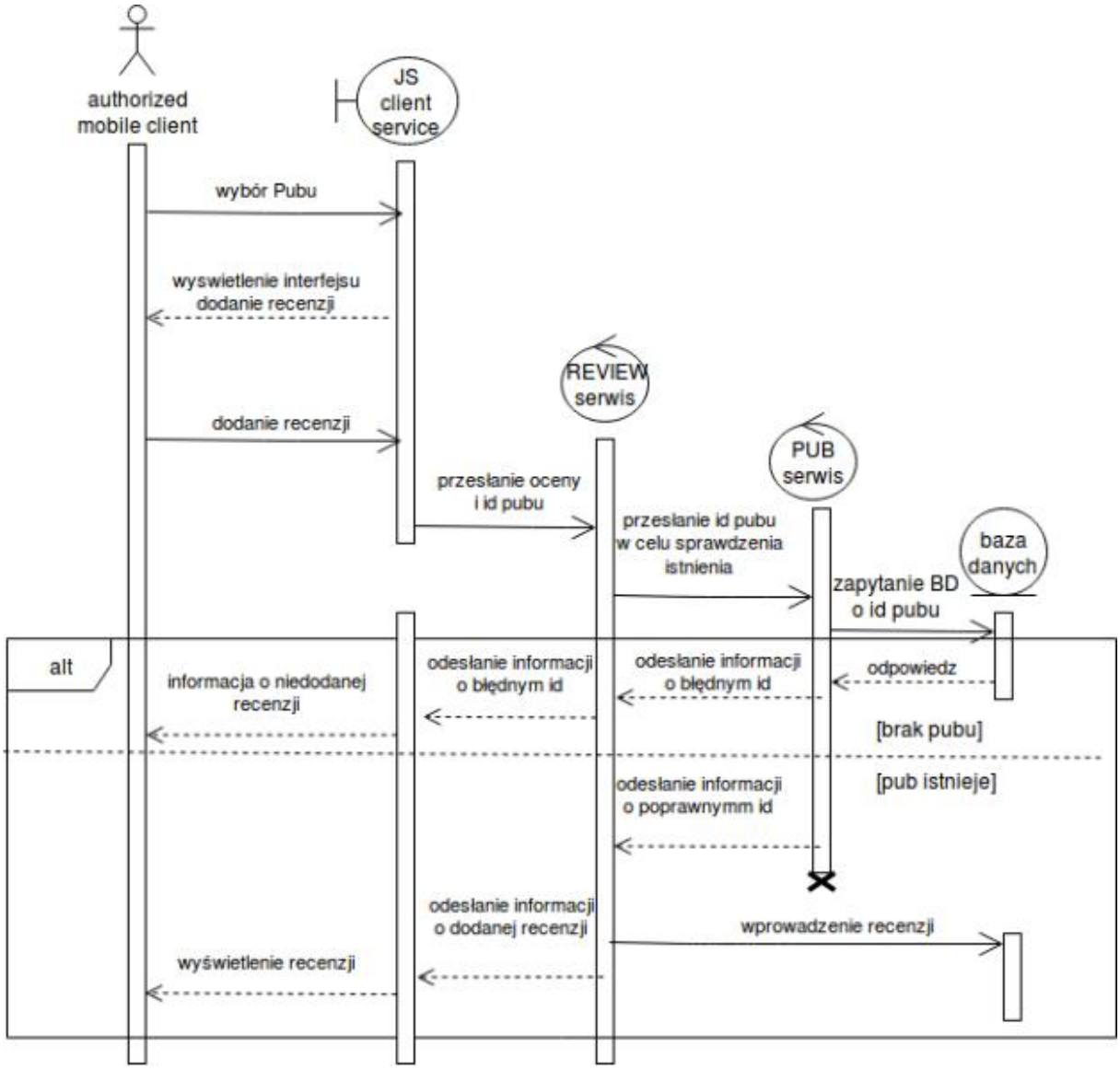


6.2. Autoryzacja operacji

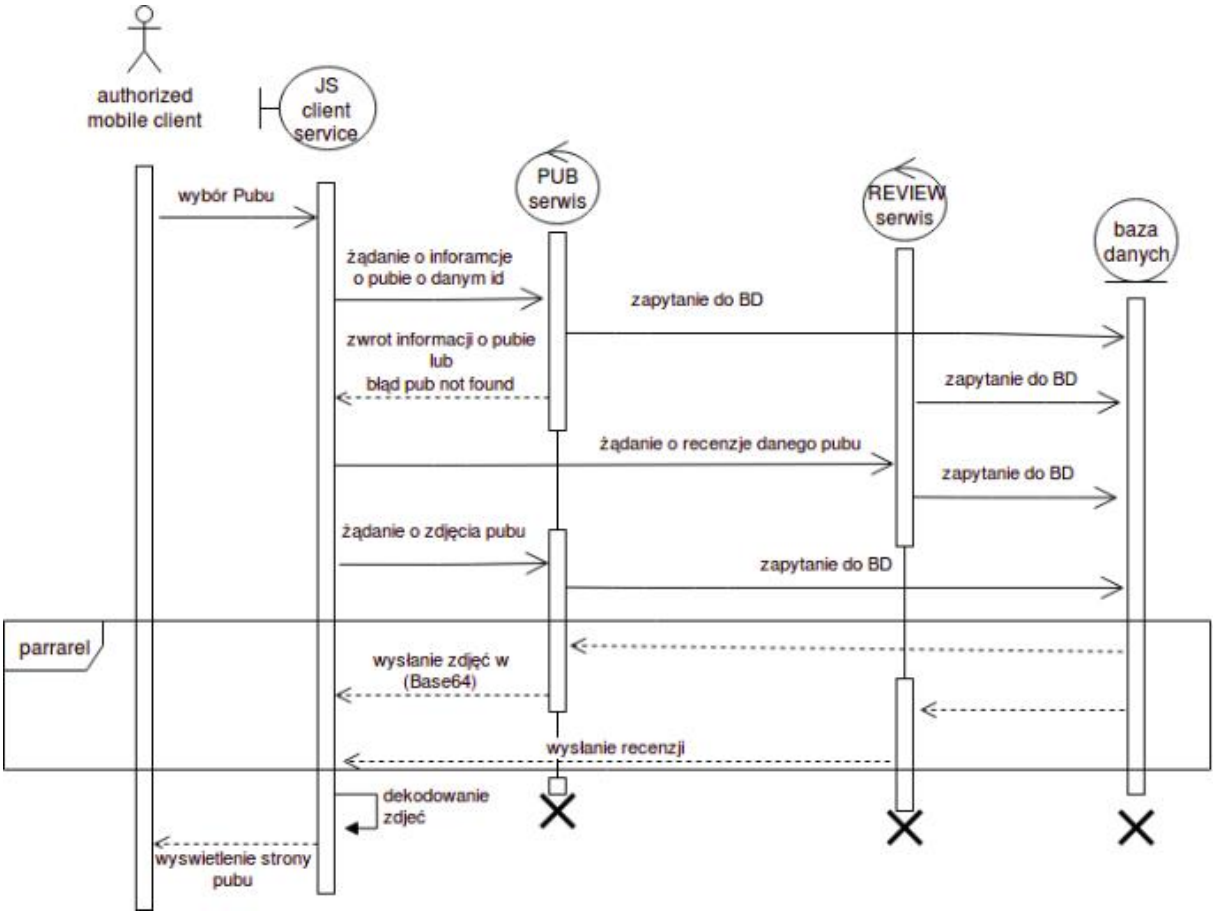


7. Diagram sekwencji

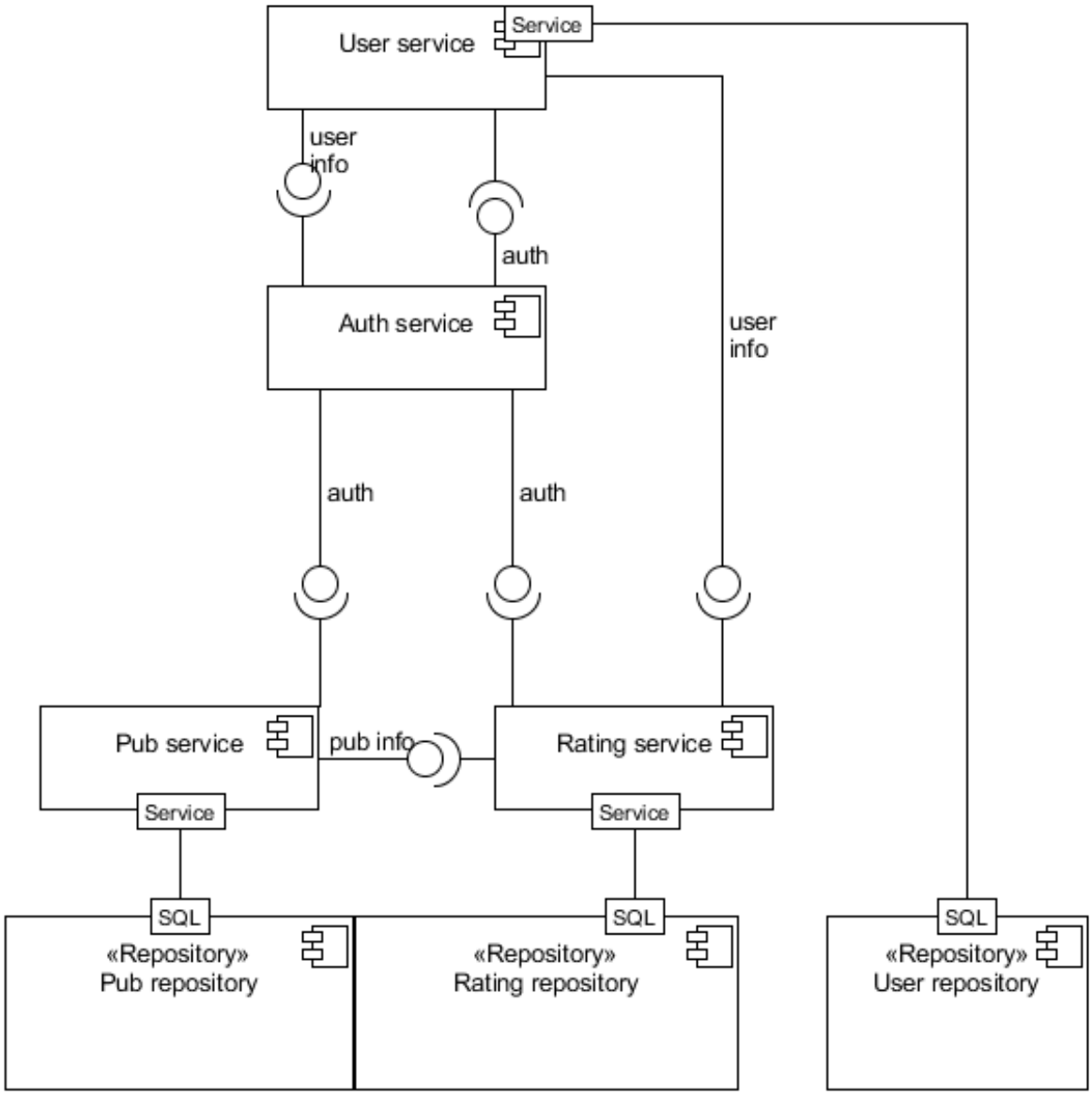
7.1. Dodanie recenzji pubu



7.2. Wyświetlanie strony pubu

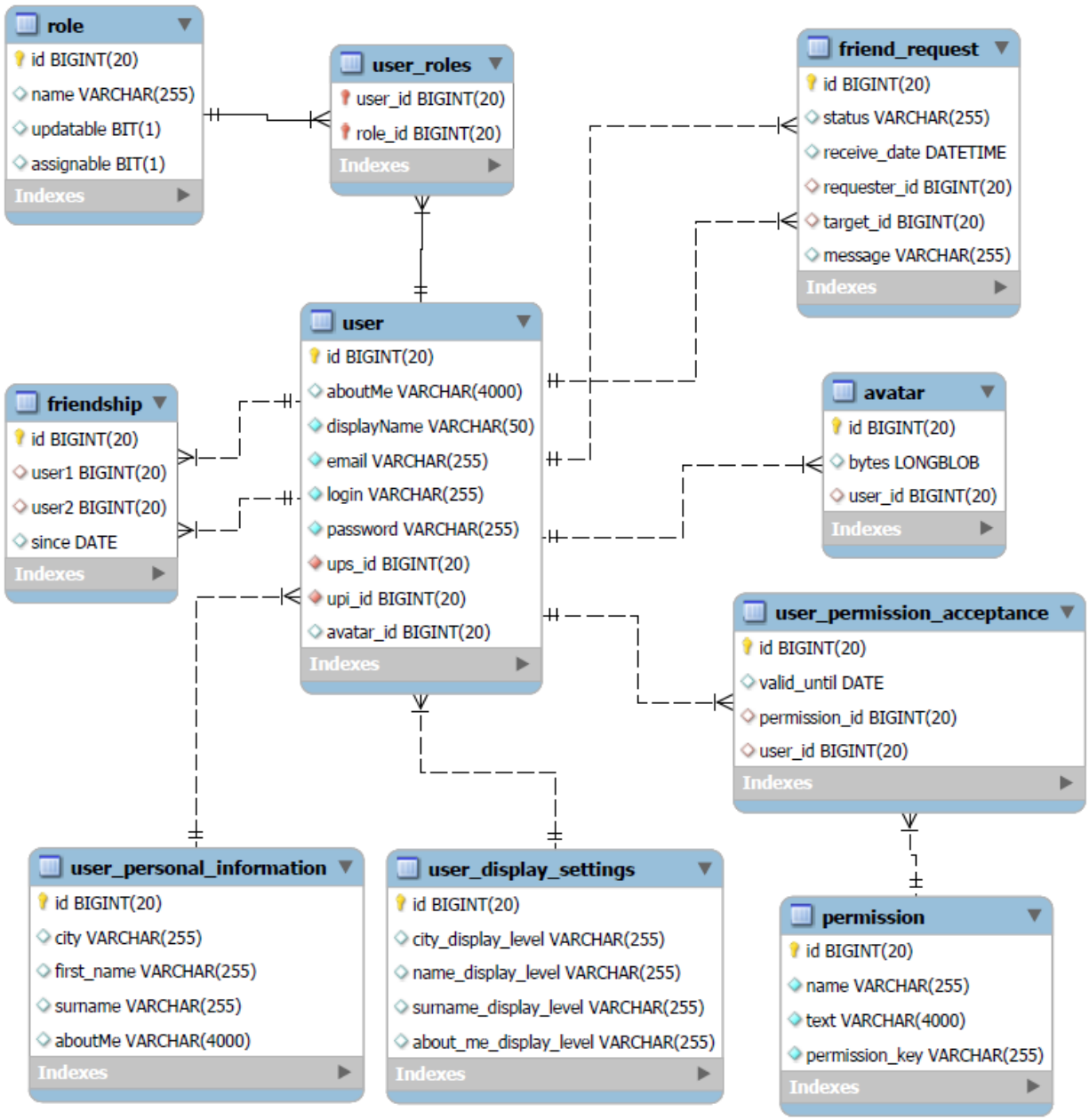


8. Diagram komponentów

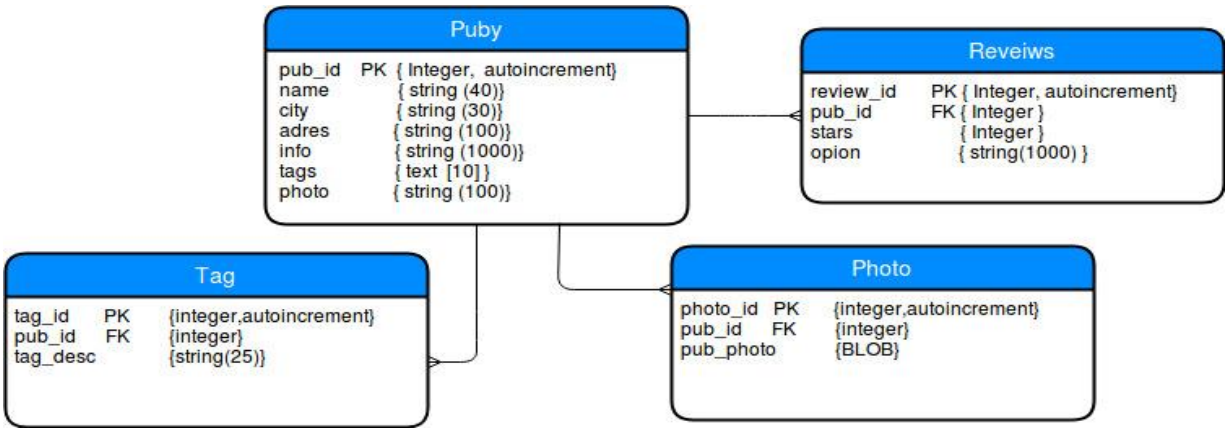


9. Diagramy związków encji

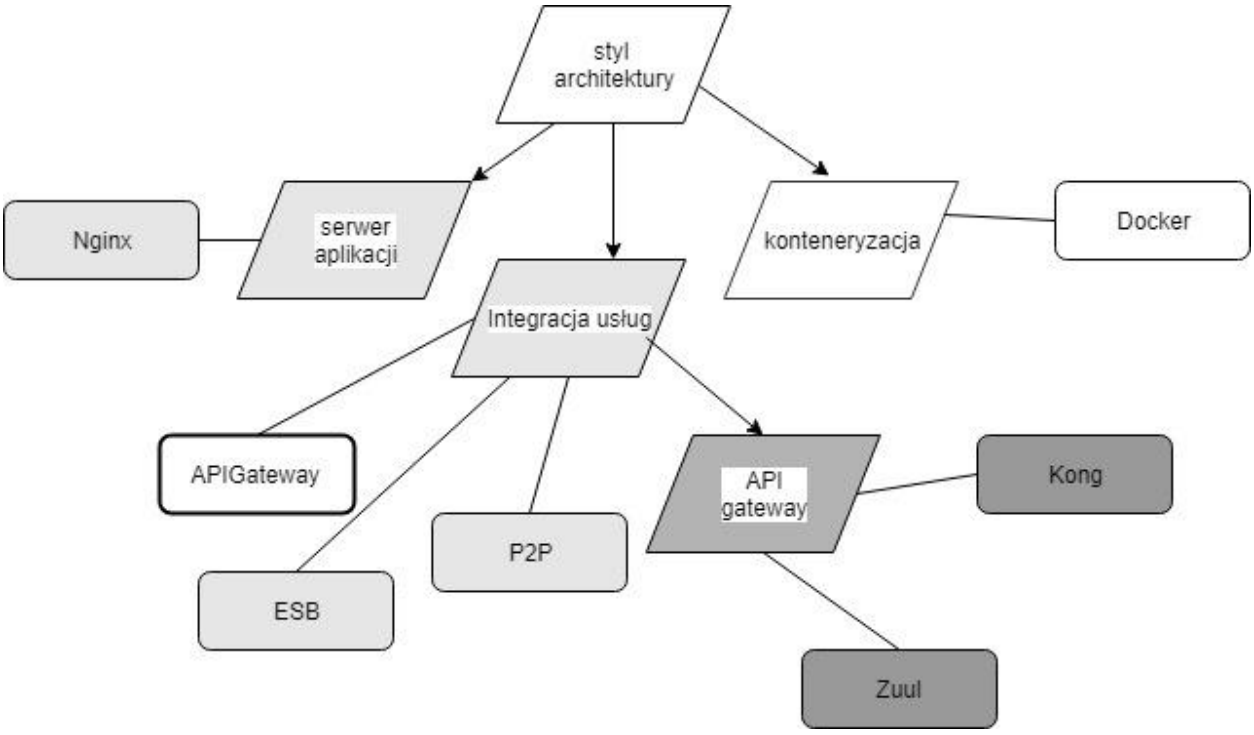
9.1. Diagram związków encji serwisów użytkowników i autoryzacji

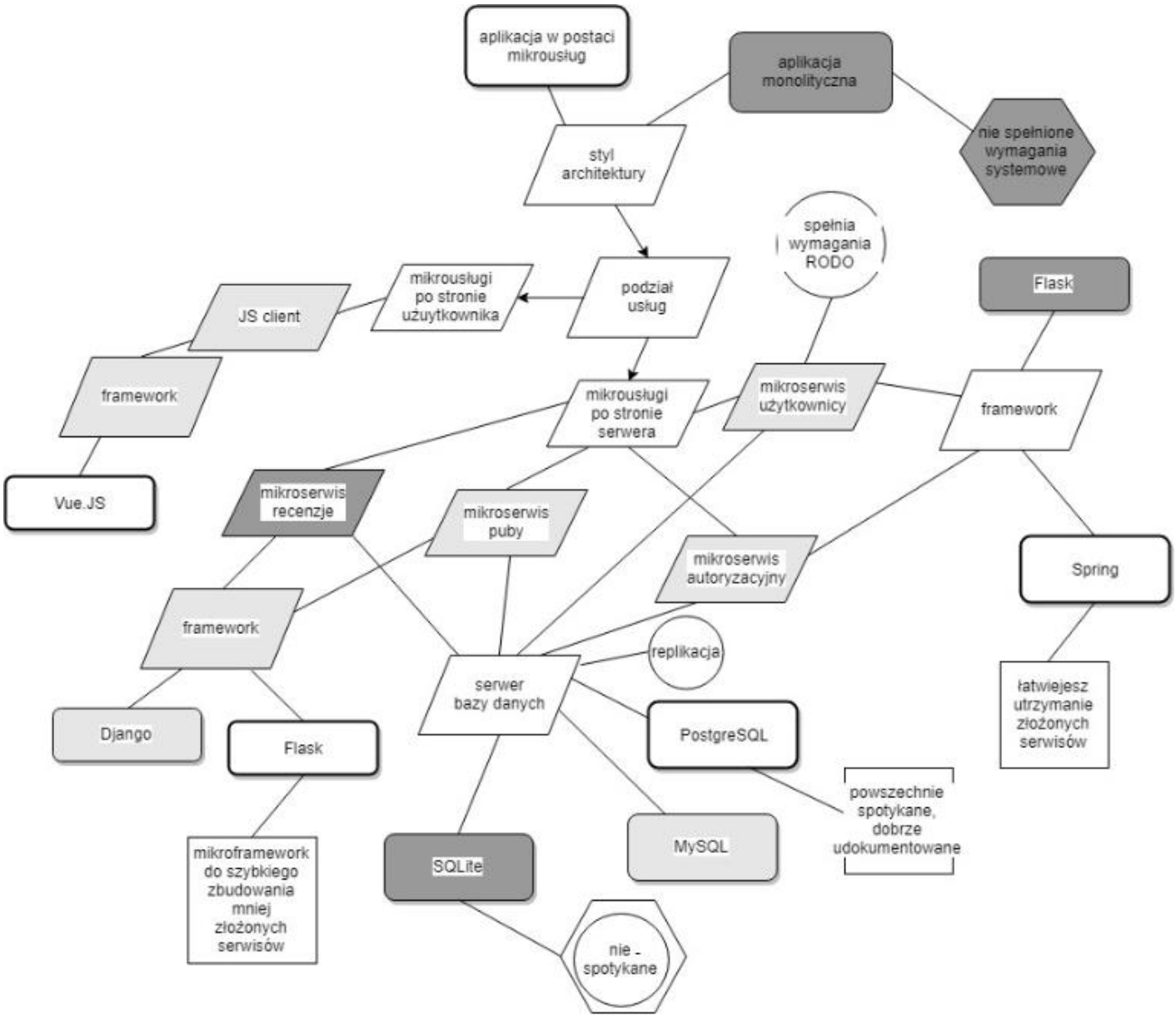


9.2. Diagram związków encji serwisów pubów i recenzji



10. Modelowanie decyzji architektonicznych w postaci MAD 2.0




aplikacja w postaci mikrousług
aplikacja monolityczna
nie spełnione wymagania systemowe
styl architektury
spełnia wymagania RODO
Flask
podział usług
mikrousługi po stronie użuytkownika
JS client
framework
mikrousługi po stronie serwera
mikroserwis użytkownicy
framework
Vue.JS
mikroserwis recenzje
mikroserwis puby
mikroserwis autoryzacyjny
Spring
framework
replikacja
serwer bazy danych
łatwiejesz utrzymanie złożonych serwisów
Django
Flask
PostgreSQL
powszechnie spotykane, dobrze udokumentowane
mikroframework do szybkiego zbudowania mniej złożonych serwisów
SQLite
MySQL
nie - spotykane