Cross application notification system

Wieloplatformowy system przesyłania i agregacji powiadomień

Projekt jest realizowany przez studentów Politechniki Wrocławskiej:

Filip Baszak

Mateusz Wójcik

Adam Maciak

Bartosz Gardziejewski

Arkadiusz Sokołowski

w kooperacji z opiekunami z firmy Nokia:

Dominik Markiewicz

Mateusz Sołtysik

Cel projektu:

Stworzenie systemu do agregacji i przetwarzania powiadomień z dowolnych zewnętrznych serwisów

Dzisiaj jesteśmy podłączeni do szerokiej gamy serwisów.

Problemy z którymi przeciętny użytkownik się spotyka:

- zbyt dużo informacji do przetworzenia pochodzących z wielu źródeł
- duży nakład czasu potrzebny na przełączanie się pomiędzy serwisami w celu uzyskania informacji

Rozwiązania zaproponowane przez nasz system:

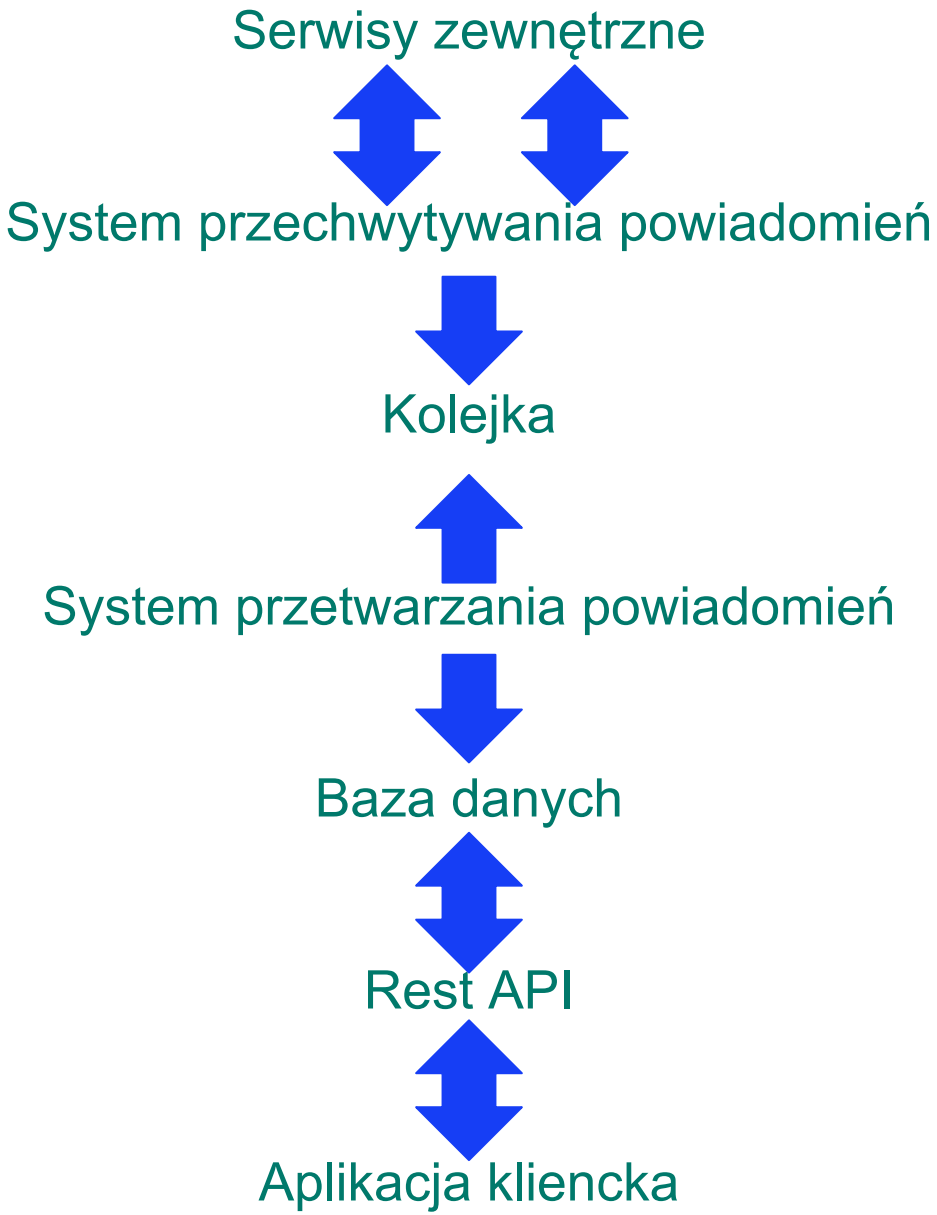
- jedno miejsce do odczytywania powiadomień ze wszystkich podłączonych serwisów
- agregacja powiadomień dla każdego źródła
- możliwość rozszerzania systemu o źródła niewspierane przez nas

Dla kogo jest nasz system:

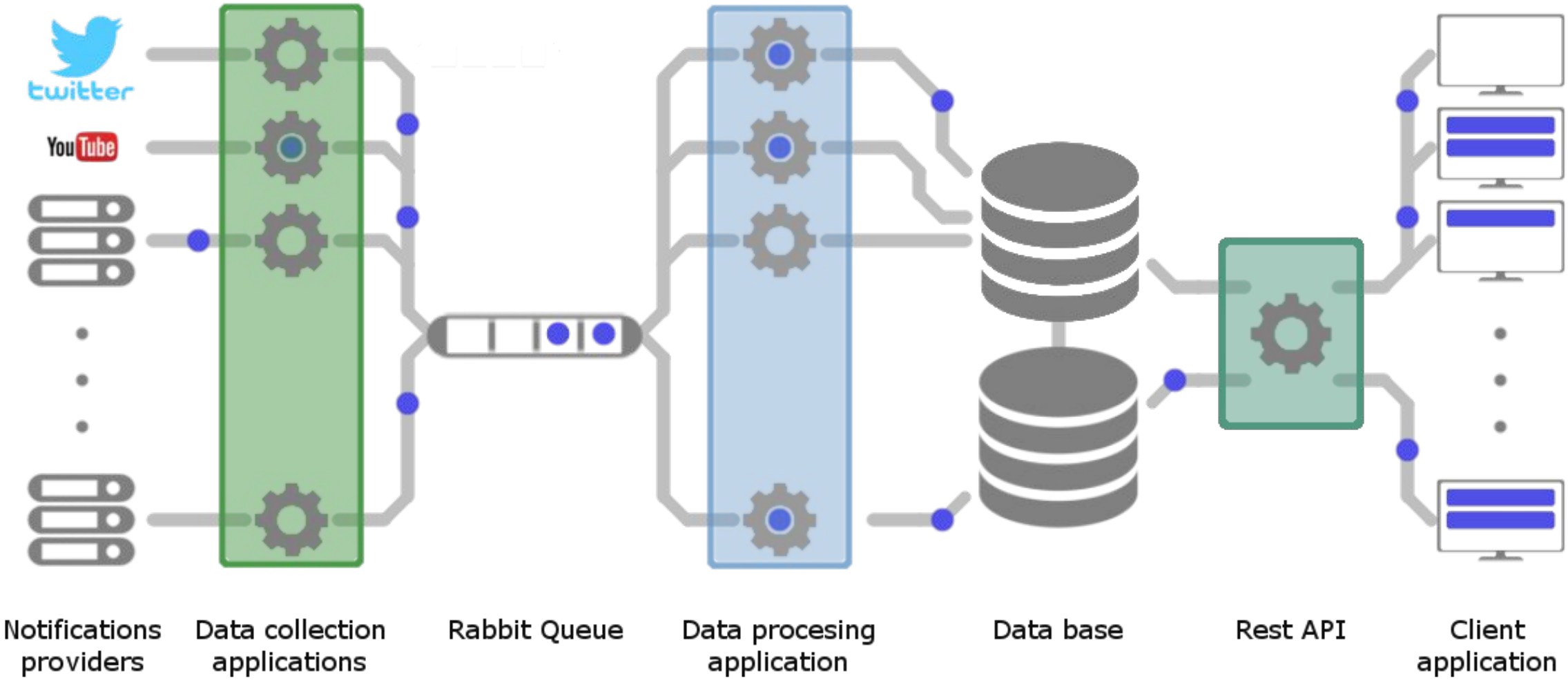
- Specjaliści z branży finansowej
- Osoby pracujące z Social Media
- Programiści
- Analitycy
- Osoby chcące mieć jedno miejsce do podglądu ze swoich serwisów - krótko mówiąc dla osób ceniących swój czas

Krótka powtórka

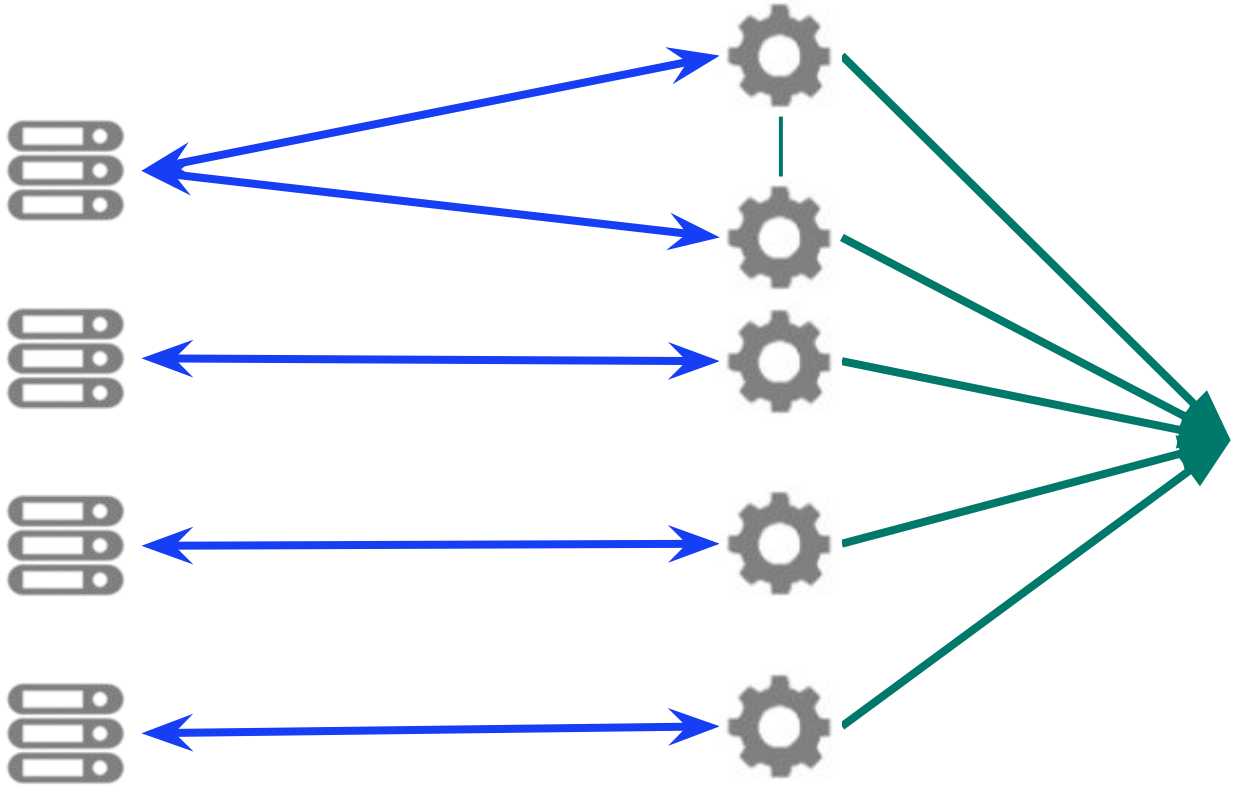
Idea:


Serwisy zewnętrzne
System przechwytywania powiadomień
Kolejka
System przetwarzania powiadomień
Baza danych
Rest API
Aplikacja kliencka

Schemat naszego systemu:



W celu dodania zewnętrznego źródła powiadomień podłączmy je do naszego systemu za pomocą mikroserwisu który pobiera powiadomienia i przesyła je na kolejkę:



Mikroserwisy ściągające powiadomienia formatują je w format nam odpowiadający.
Mikroserwisy są skalowalne.

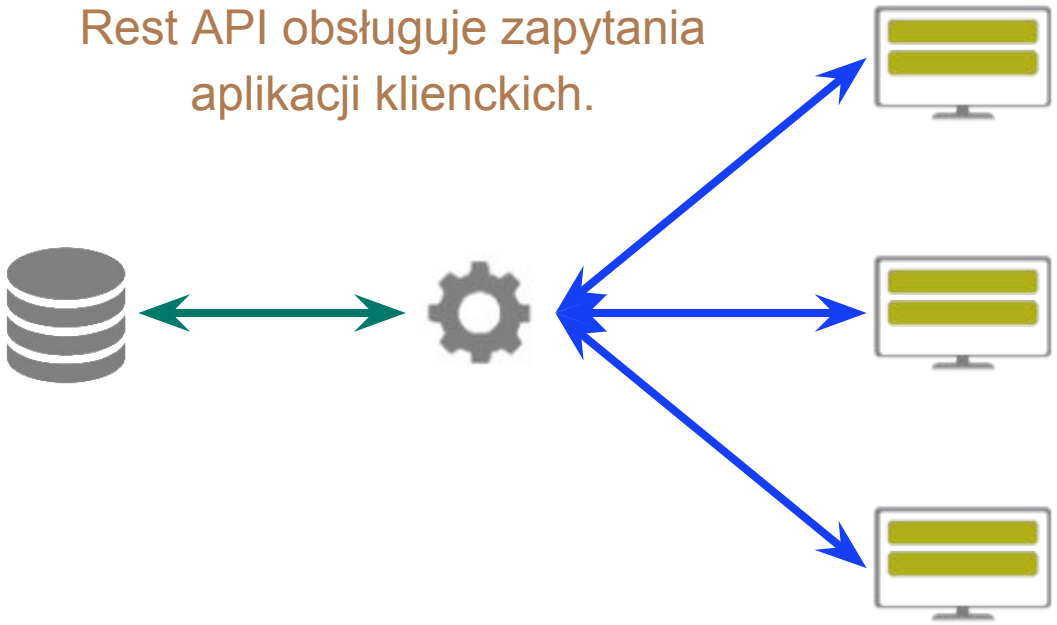
Następnie system pobierający powiadomienia z kolejki, przesyła je do bazy danych:

Notification JSON format:

```
{
        notificationID : ,
        userID : ,
        sourceID : ,
        flag : ,
        topic : ,
        message : ,
        timestamp : ,
        priority :
}
```

Rest Api odpowiada za komunikację pomiędzy aplikacjami klienckimi a bazą danych:

Rest API obsługuje zapytania aplikacji klienckich.



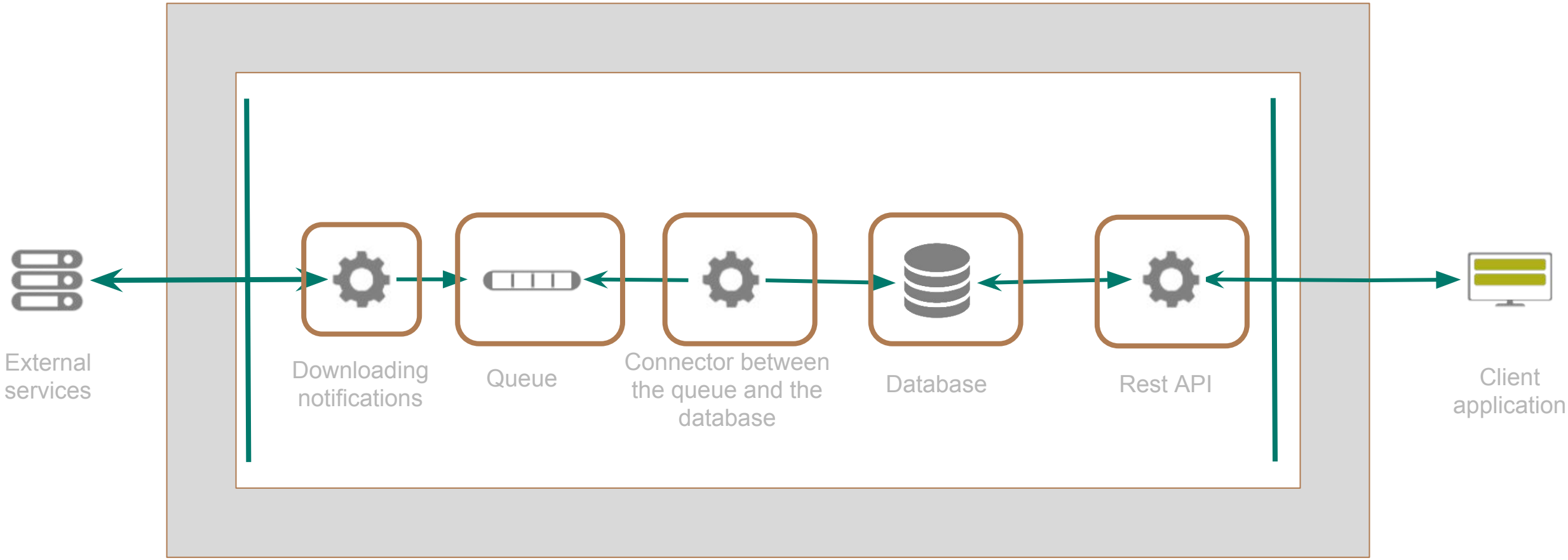
W ramach projektu powstaną:

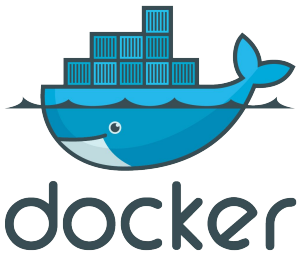
- **prosta aplikacja desktopowa(pokazana na ostatnim demo, dalej nierozwijana)**
- **plugin do przeglądarki Chrome napisany w Angular JS**
- **aplikacja mobilna**

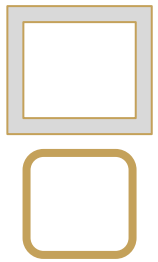
Jedną z głównych funkcjonalności naszego systemu jest agregacja powiadomień pochodzących z jednego serwisu.
Agregacja powstanie na poziomie aplikacji klienckiej oraz na poziomie przetwarzania powiadomień w systemie, który będzie analizował treść oraz czas pomiędzy powiadomieniami w celu ich zagregowania.

Wciąż pracujemy nad tą funkcjonalnością.

Jak to wszystko razem działa?







Server

Docker container

Milestones

Czyli co miało zostać zrealizowane

Rest API	✔
Plugin do przeglądarki	✔
Monitoring serwera	✔
Łącznik pomiędzy kolejką a bazy danych	✔
Monitoring kolejki	✔
Monitoring bazy danych	✔
Monitoring kontenerów dockera	✔
Testy obciążeniowe Rest API	✔
Pobieranie powiadomień z jednego zewnętrznego źródła(Twitter)	✔
Pobieranie powiadomień z drugiego zewnętrznego źródła(YouTube)	✖

Organizacja pracy

- przydział zadań : GitHub
- spotkania: ok. 6 - 12 godzin, podczas których pracujemy nad przydzielonymi zadaniami
- spotkania w firmie Nokia raz w tygodniu podczas których weryfikujemy postęp prac

Standardy sprawdzania kodu

Każdy fragment systemu który wdrażamy musi być na początku sprawdzony pod względem wyszukania błędów oraz zaakceptowany. Wygląda to następująco:

a) gdy kod jest pisany przez osobę odpowiedzialną za ściąganie powiadomień z innych serwisów:

- **sprawdza**: osoba odpowiedzialna za kolejkę i jej obsługę
- **akceptuje**: osoba odpowiedzialna za komunikację z bazą danych

b) gdy kod jest pisany przez osobę odpowiedzialną za kolejkę i jej obsługę:

- **sprawdza**: osoba odpowiedzialna za komunikację z bazą danych
- **akceptuje**: osoba odpowiedzialna za utrzymanie bazy danych

c) gdy kod jest pisany przez osobę odpowiedzialną za komunikację z bazą danych:

- **sprawdza**: osoba odpowiedzialna za utrzymanie bazy danych
- **akceptuje**: osoba odpowiedzialna za aplikację kliencką

d) gdy kod jest pisany przez osobę odpowiedzialną za utrzymanie bazy danych:

- **sprawdza**: osoba odpowiedzialna za aplikację kliencką
- **akceptuje**: osoba odpowiedzialna za konteneryzację na serwerze

e) gdy kod jest pisany przez osobę odpowiedzialną za aplikację kliencką:

- **sprawdza**: osoba odpowiedzialna za konteneryzację na serwerze
- **akceptuje**: osoba odpowiedzialna za ściąganie powiadomień

f) gdy kod jest pisany przez osobę odpowiedzialną za agregację na serwerze:

- **sprawdza**: osoba odpowiedzialna za ściąganie powiadomień
- **akceptuje**: osoba odpowiedzialna za kolejkę i jej obsługę

g) gdy kod jest pisany przez parę osób naraz:

- **sprawdza**: osoba oddelegowana przez lidera projektu
- **akceptuje**: lider projektu

Użyte technologie:

Notifications downloading system:
- Java

Notification processing system:
- Java

Rest API:
- Java Spring

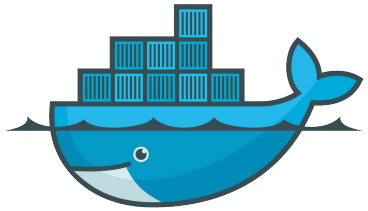
Queue service:
- RabbitMQ

Database:
- PostgreSQL
- JDBC Java
- Hibernate

Client application:
- Java
- AngularJS
- Android

Testing tools:
- JMeter

Monitoring:
- DataDog



Co zostało stworzone?

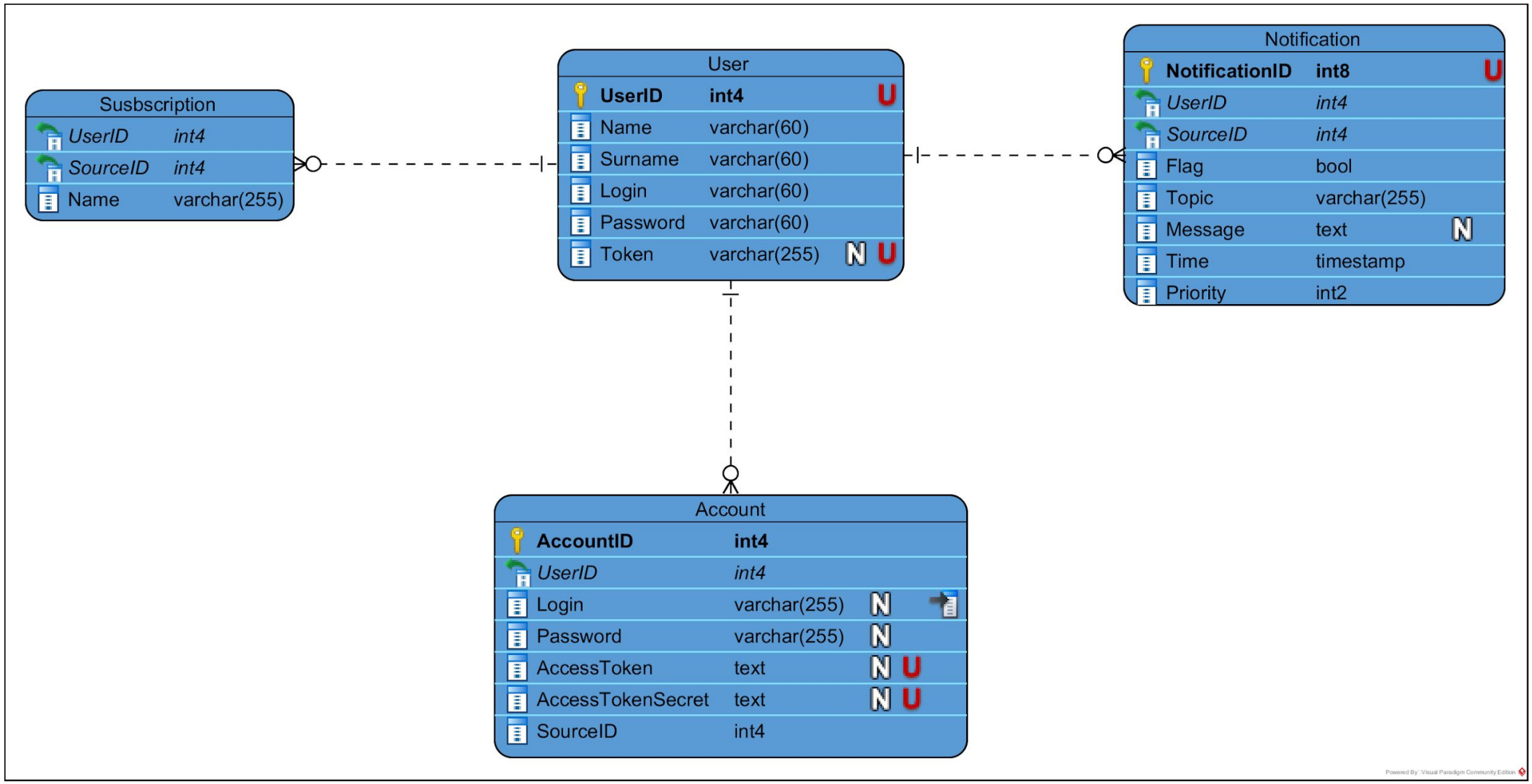
Live

Czas na pytania

Dziękujemy za uwagę

Filip Baszak, Mateusz Wójcik, Adam Maciak, Bartosz Gardziejewski, Arkadiusz Sokołowski
Wroclaw University of Technology, Department of Electronics, IT Faculty

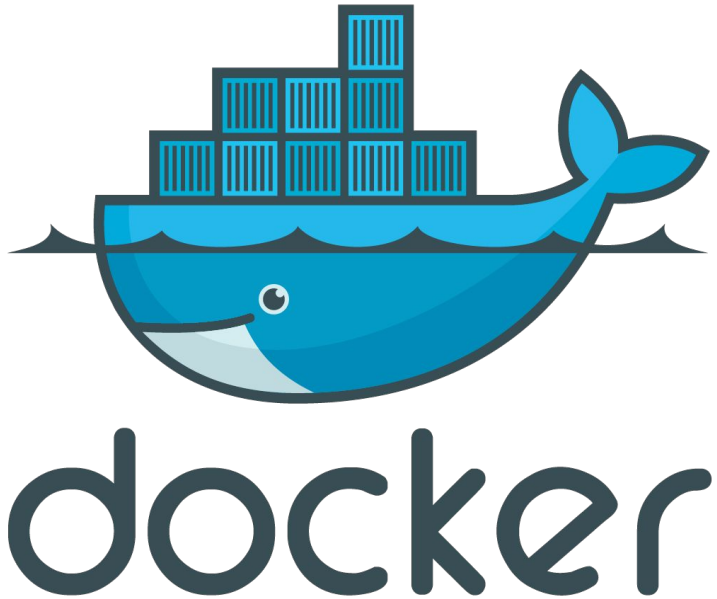
Database - Entity Relationship Diagram



Database - Views

View	Table	Column
User_subscriptions	User	Login
	Subscription	Name
Notification_name	Subscription	Name
	Notification	Topic
	Notification	Message
	Notification	Time
	Notification	Flag

Rabbit Queue



Do wdrożenia i uruchomienia naszej aplikacji rozproszonej wybraliśmy Dockera. Pozwoli on umieścić nasz program oraz jego zależności w lekkich, przenośnych, wirtualnych kontenerach które można uruchomić na prawie każdym serwerze z systemem Linux.