Architektura i narzędzia w systemach mikroserwisowych

Agenda

Welcome
History of Microservices
Problems of Monolith & SOA
Microservices Architecture
Problems Solved by Microservices
Designing Microservices Architecture
Deploying Microservices
Testing Microservices
Service Mesh
Logging And Monitoring
When NOT to use Microservices
Microservices and the Organization
Anti-Patterns and Common Mistakes
Breaking Monolith to Microservices
Case Study
Conclusion

Testing Microservices

- Testowanie ważne wewszystkich systemach i dla wszystkich typów arhiktektur
- Z mikroserwisami jeszcze bardziej
- Testowanie mikroserwisowej architektury posiada własne wezwania

Typy testów:

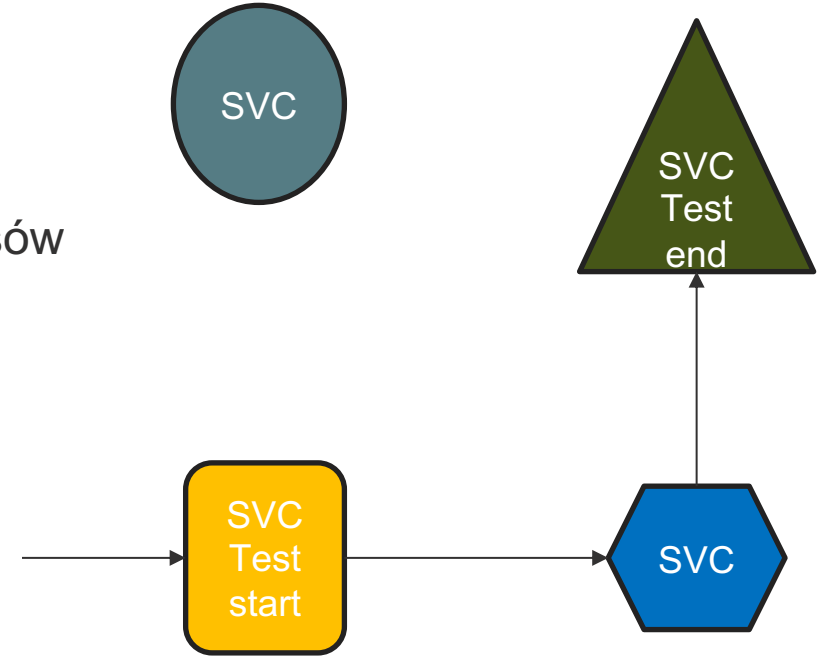
Unit

Integration

End-to-End

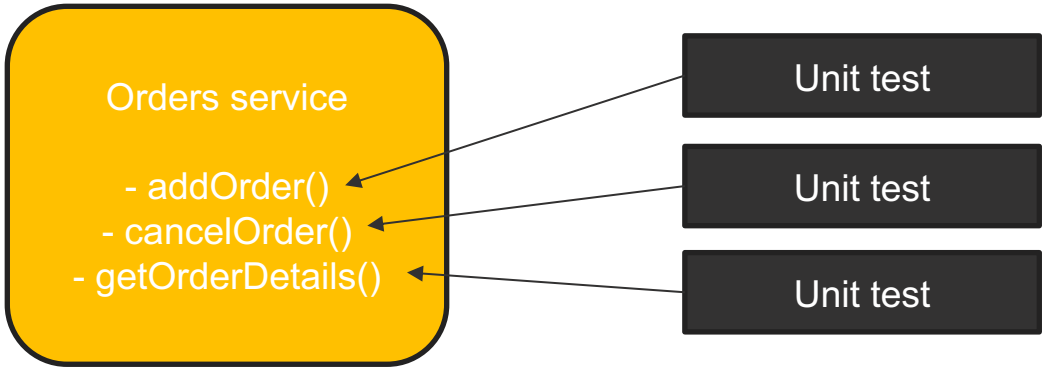
Challenges with Microservices Testing

- Ruchome częsci
- Testowanie nie jest triwialne:
  - ”cross” – testowanie serwisów
  - “nonfunctional” zależności



Unit tests

- Testowanie indywidualnych jednostek kodu (metody, interfacy)
- Zautomatyzowane
- Tworzone przez deweloperów



Integration Tests

- Testowana funkcjonalność
- Nie interesuję nas jak serwis jest napisany
- Probujemy pokryć całośc kodu w serwisie
- Testujemy również zewnętrzne zależności (bazy danych, inne serwisy)

End-to-End Tests

- Testujemy cały “flow”
- Sprawdzamy wszystkie serwisy
- Testy bardzo kruche
- Wymagają bezpośredniego dostępu do systemu (do db na przykład)
- Zwykłe pokrywają tylko głównei scenaria

Service Mesh

- Zarządza komunikacją pomiędzy serwisami
- Dostarcza dodatkowe serwisy
- Platform-agnostic

Jaki problemy rozwiązuje servicemesh?

- Microserwisy gadają pomiędzy sobą
- W komunikacji mogą być problemy:
  - Timeouty
  - Bezpieczeństwo
  - Retries
  - Monitoring

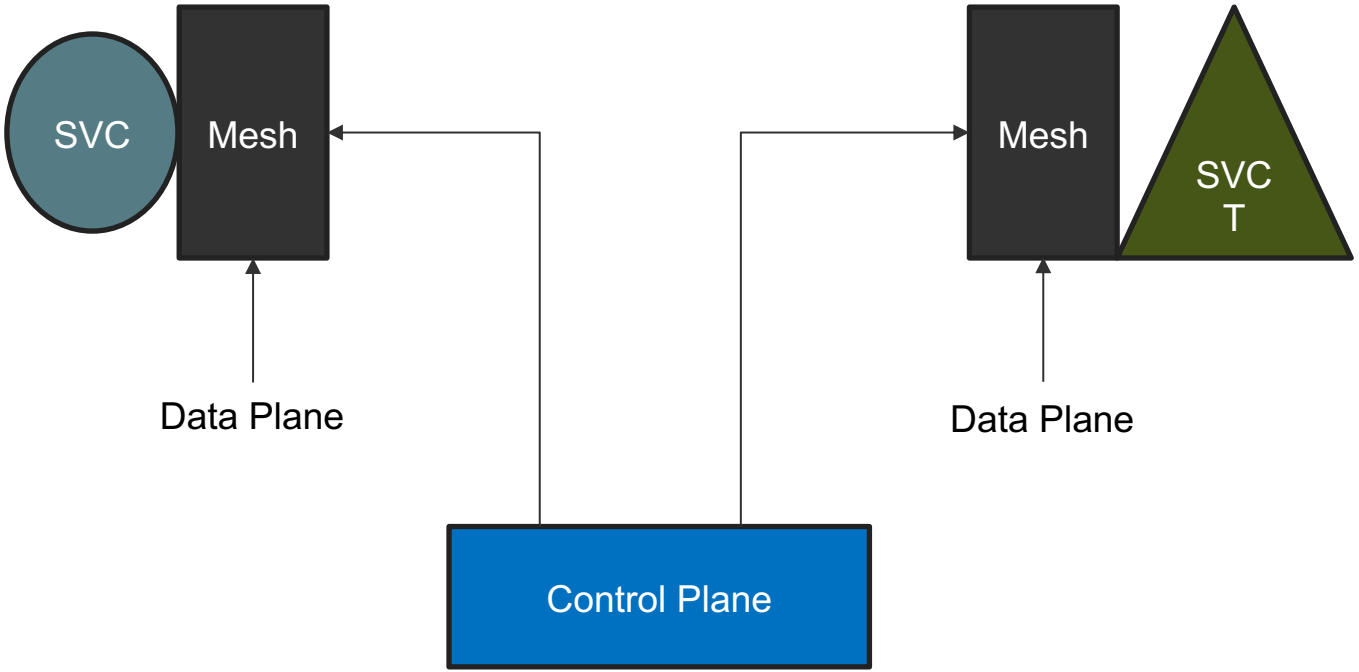
Service Mesh

- Zbiór komponentów, którzy znajdują się blisko usługi I zapewniają komunikacje
- Zapezpiecza całośc komunikacji
- Serwisy komunikują wylącznie przez service mesh

Zabezpiecza:

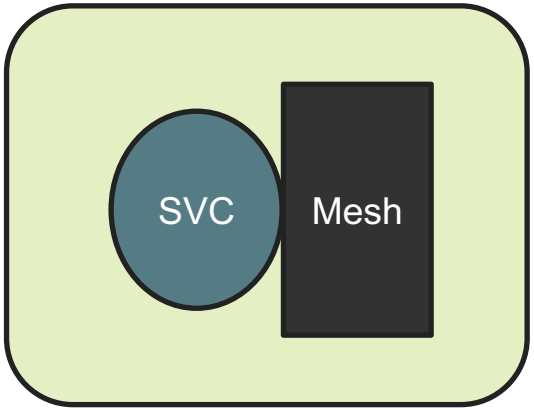
- Konwersjie protokołow
- Bezpieczeństwo
- Uwierzytelnianie
- Niezawodność

Arhitektura Service Mesh



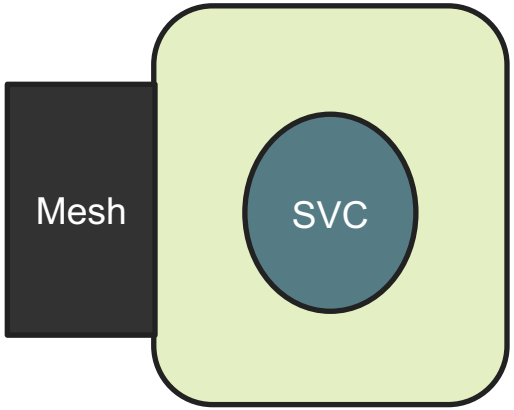
Rodzaje Service Mesh

In-process



Wydajny

Sidecar



Platform Agnostic

Code Agnostic

Demo: Anthos Service Mesh

- ASM = Istio

Logging and Monitoring

- Bardzo ważny
- Flow - przez wiele serwisów
- Ciężko otrzymać całościowy obraz
- Ciężko zrozumieć co sie dzieje z usługą

Logging VS Monitoring

Logging

Notuje aktywności systemowe

Zajmuje się audytem

Notuje błędy

Logging

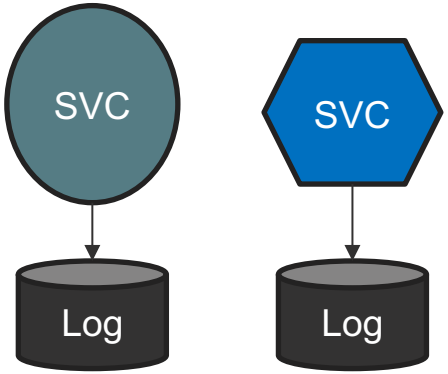
Kontroluje metryki systemowe

Alertuje w przypadku anomalii

Logging

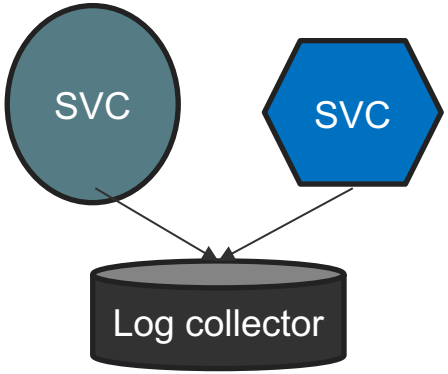
- Logować jaknajwięcej
- Logować cały end-to-end flow
- Pokazuje całościowy widok na system

Logging



Klasyczny system

Osobne logi
Różne formaty
Nie zbierane
Trudno ich analizować



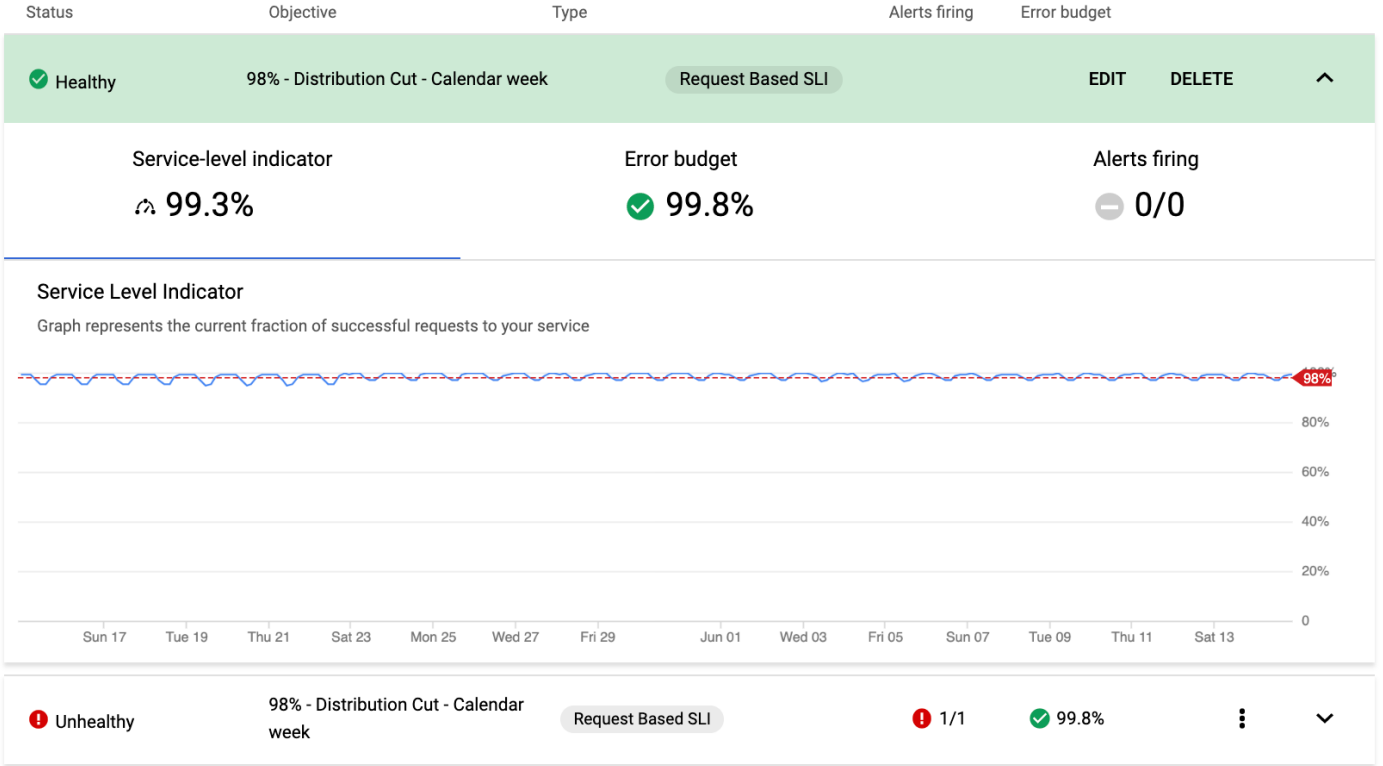
Microserwisowy system

Uniwersalne
Agregowane
Łatwo analizowane

Monitoring

- Monitoring kontroluje metryki, wyswietla anomalie
- Udostępnia prosty widok stanu systemów
- Alertuje, kiedy jest problem

Monitoring


Status
Objective
Type
Alerts firing
Error budget
Healthy
98% - Distribution Cut - Calendar week
Request Based SLI
EDIT
DELETE
Service-level indicator
99.3%
Error budget
99.8%
Alerts firing
0/0
Service Level Indicator
Graph represents the current fraction of successful requests to your service
98%
80%
60%
40%
20%
0
Sun 17
Tue 19
Thu 21
Sat 23
Mon 25
Wed 27
Fri 29
Jun 01
Wed 03
Fri 05
Sun 07
Tue 09
Thu 11
Sat 13
Unhealthy
98% - Distribution Cut - Calendar week
Request Based SLI
1/1
99.8%

Rodzaje monitoringu

Infrastrukturny

Monitoruje system.
Alertuje problemy z:
CPU
RAM
Dysk
Siec

Źródło: agent

Aplikacja

Monitoruje aplikacje
Kontruluje
Zapytania
Zamowienia
Zamowienia za godzine
Alertuje problemy z aplikacją

Źródło: logi aplikacji, wydarzeń

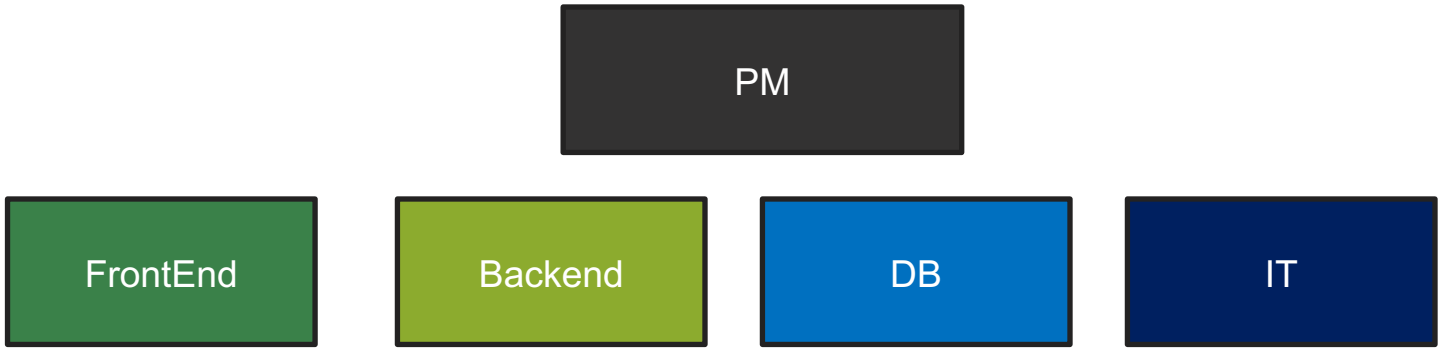
Kiedy nie używać microserwisów

- Małe systemy
- Przeplatanie logiki aplikacji oraz danych
- Systemy wrażliwe na wydajność
- POC systems
- Embedded systemy (“no-update” systemy)

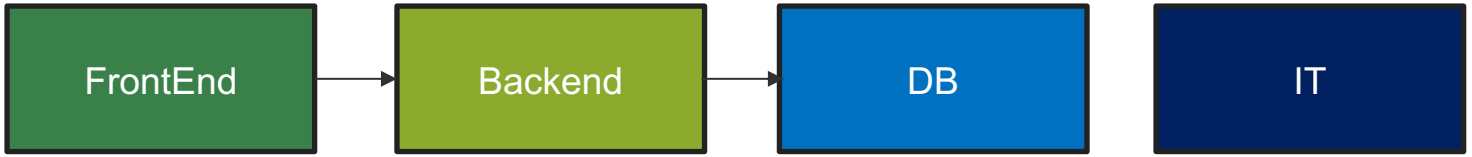
Microservices and Organization

Conway's Law:

"Organizations design systems that mirror their own communication structure"



Microservices and Organization



Projektowe podejscie

Serwisowe podejscie

Anti-Patterns and Common Mistakes

Pamiętamy:

Mikroserwisy wymagają dobrego projektowania

Mikroserwisy nie “Fire and Forget”

Łatwo zrobić błąd który rozwali cały projekt

Anti-Patterns and Common Mistakes

Brak dobrze zdefiniowanych serwisów

Brak dobrze zdefiniowanego API

Istnieja pomiedzy serwisowe zależności

Functionality	Path	Return Codes
Get next list to be processed	`GET /api/v1/lists/next?location=…`	`200 OK` `400 Bad Request`
Mark item as collected / unavailable	`PUT /api/v1/list/{listId}/item/{itemId}`	`200 OK` `404 Not Found`
Export list's payment data	`POST /api/v1/list/{listId}/export`	`200 Ok` `404 Not Found`

Migracja Monolita do Arkitektury mikroserwisowej

To pozwoli na:

Skrócenie cyklu developmentu oraz update

Modularyzacje systemu

Zmniejszenie kosztów

Modernyzacje systemu

Marketingowe zalety?

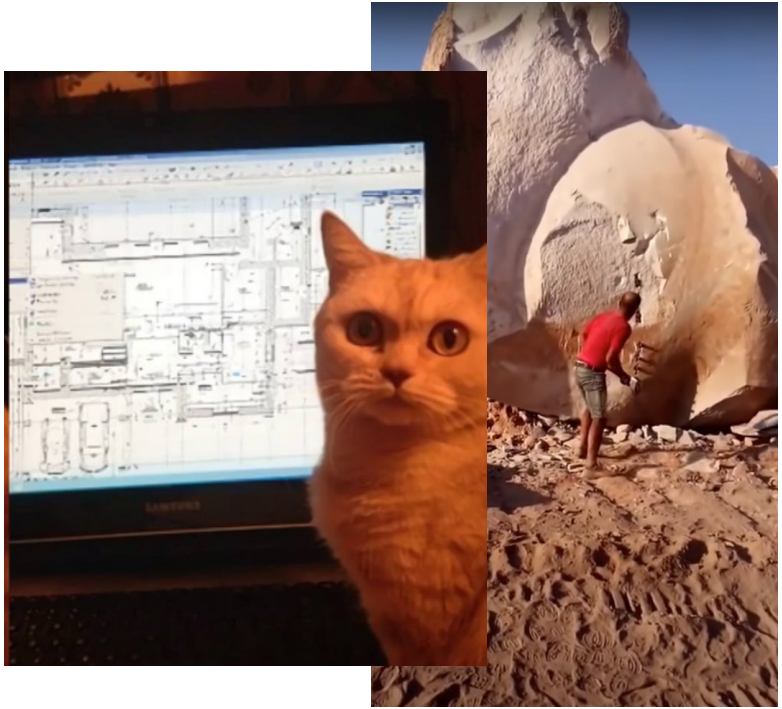
Strategji migracji

Rożdzielenie jest procesem delikatnym

Trzeba go planować mocno

3 głownych strategji:

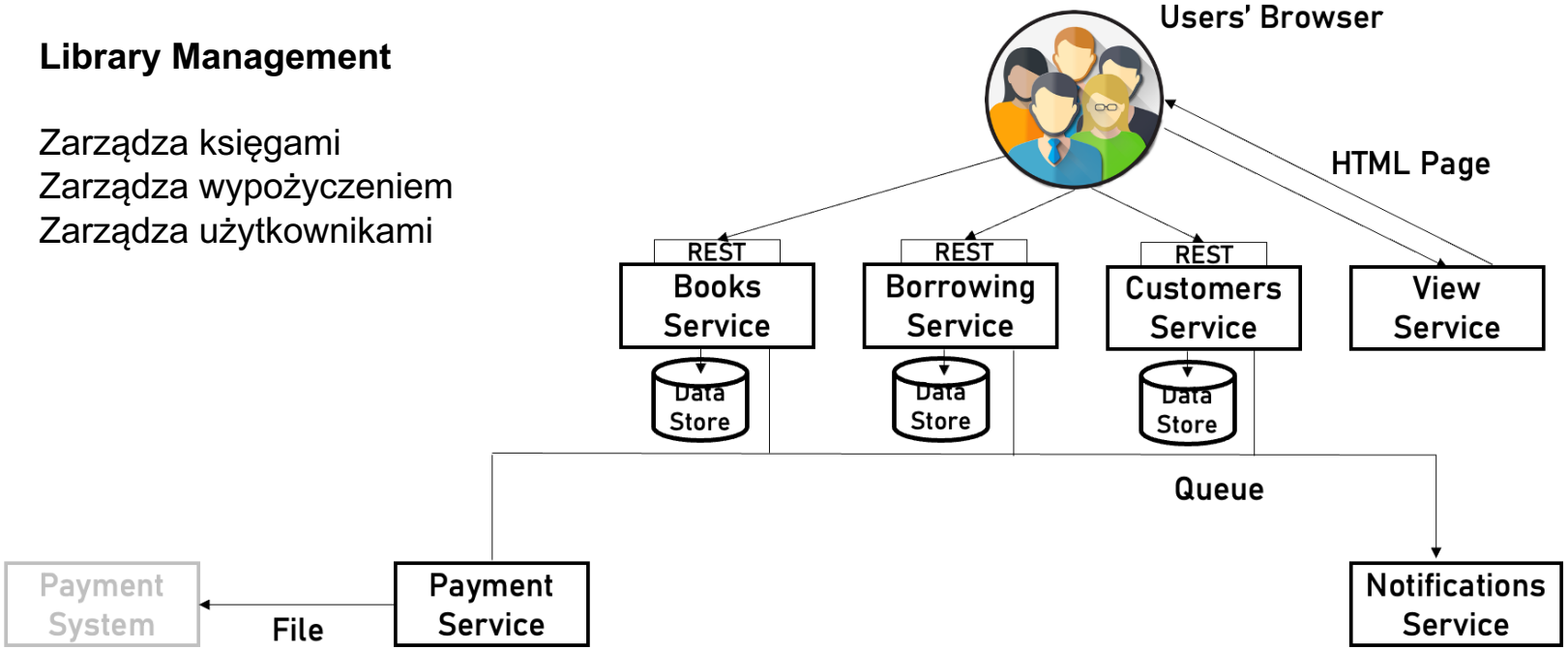
- Nowe module jako serwisy
- Wydzielenie istniejących modulow
- Pełne przepisywanie



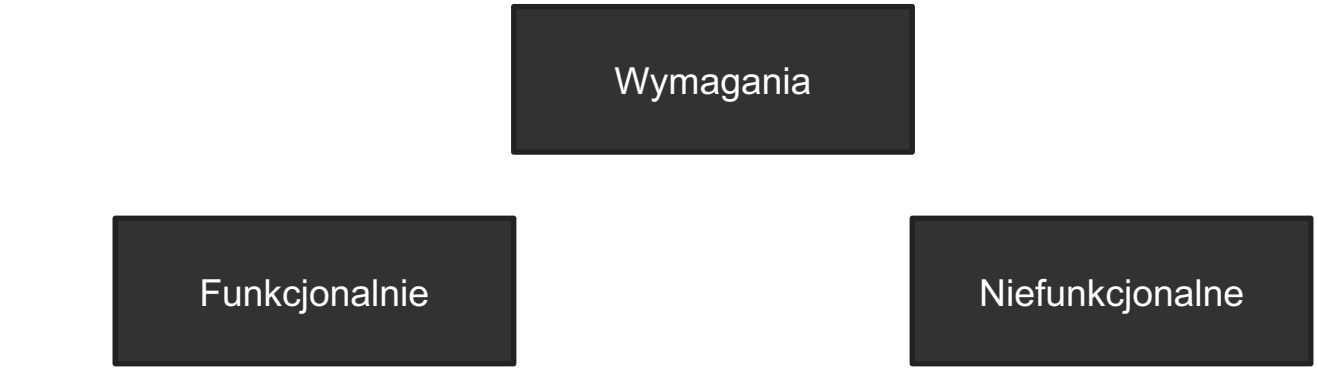
Case study

Library Management

Zarządza księgami
Zarządza wypożyczeniem
Zarządza użytkownikami



Case study



Co muśi robić system

Muśi być “web-based”
Zarządzać księgami
Zarządzać wypozyczeniem
Zarządzać użytkownikami
Pokazywać powiadomienia
Kasować miesięczną oplatę

Z czym muśi wpólpracować system

Case study

Książki

Wypożyczenie

Użytkownicy

Powiadomienia

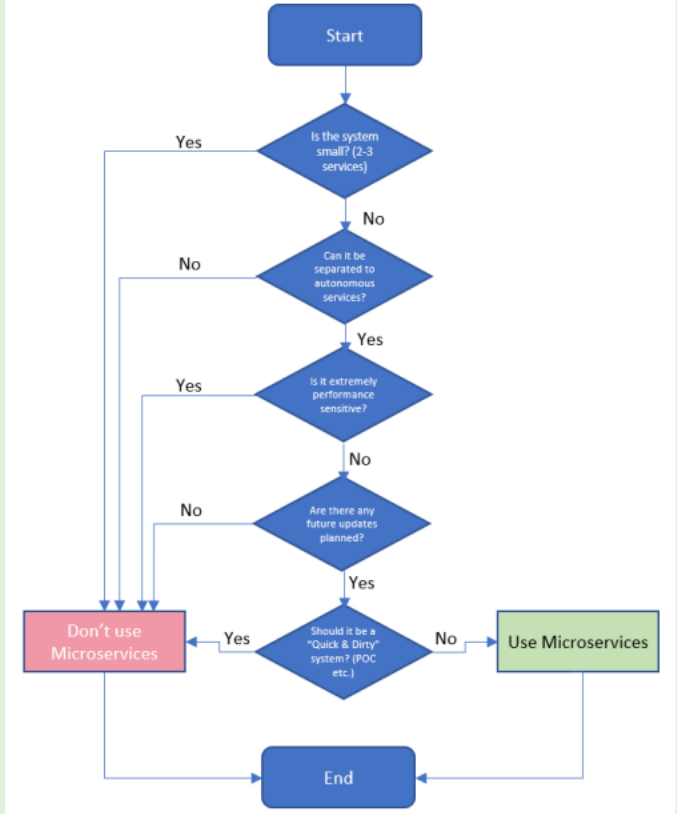
Płatności

Web-page

Bonus

Microservices

Flow Chart



Migrate VM to Service

- Practice

Przydatne linki

https://martinfowler.com/articles/microservices.html

https://encyclopedia2.thefreedictionary.com/WS*+protocols

Dzieki

Pytania, zadania domowe:

oleksii.tsyganov@gmail.com