infoShare ACADEMY

Architektura mikroserwisów

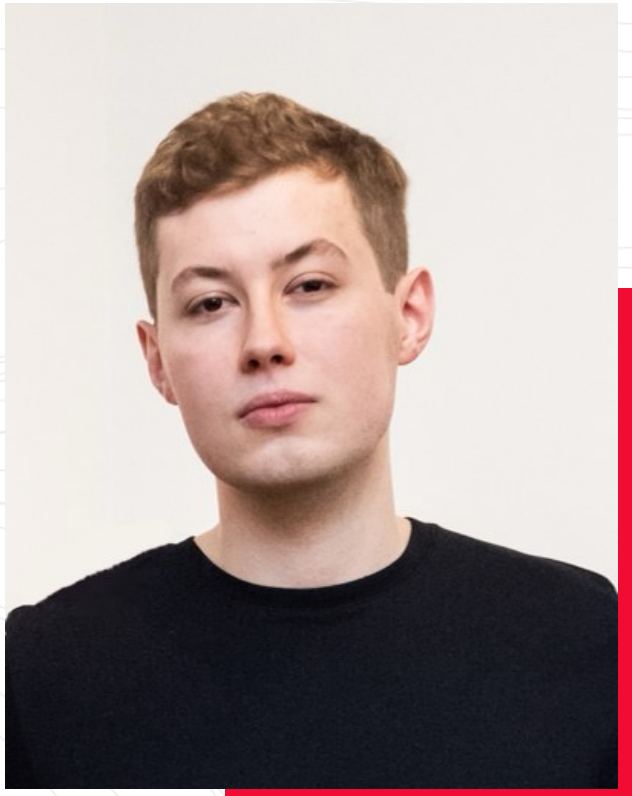
infoShareAcademy.com

HELLO

Dominik Młynarczyk

Software Engineer

almservices.tech | atarise.com



infoShare ACADEMY

Agenda szkolenia

1. Wprowadzenie do szkolenia – fundamenty architektury mikroserwisowej
2. Określanie granic odpowiedzialności mikroserwisów
3. Coupling i odporność systemu
4. Wprowadzenie do projektu praktycznego w Nest.js

info Share ACADEMY

Agenda szkolenia

5. Wprowadzenie do frameworka Nest.js
6. Budowa pierwszego mikroserwisu w oparciu o monorepo
7. Wprowadzenie do arch. heksagonalnej
8. Dockeryzacja
9. Budowa drugiego mikroserwisu – serwis uwierzytelnienia
10. Komunikacja TCP pomiędzy dwoma serwisami
11. Z monolitu do mikrousług
12. Podsumowanie

info Share ACADEMY

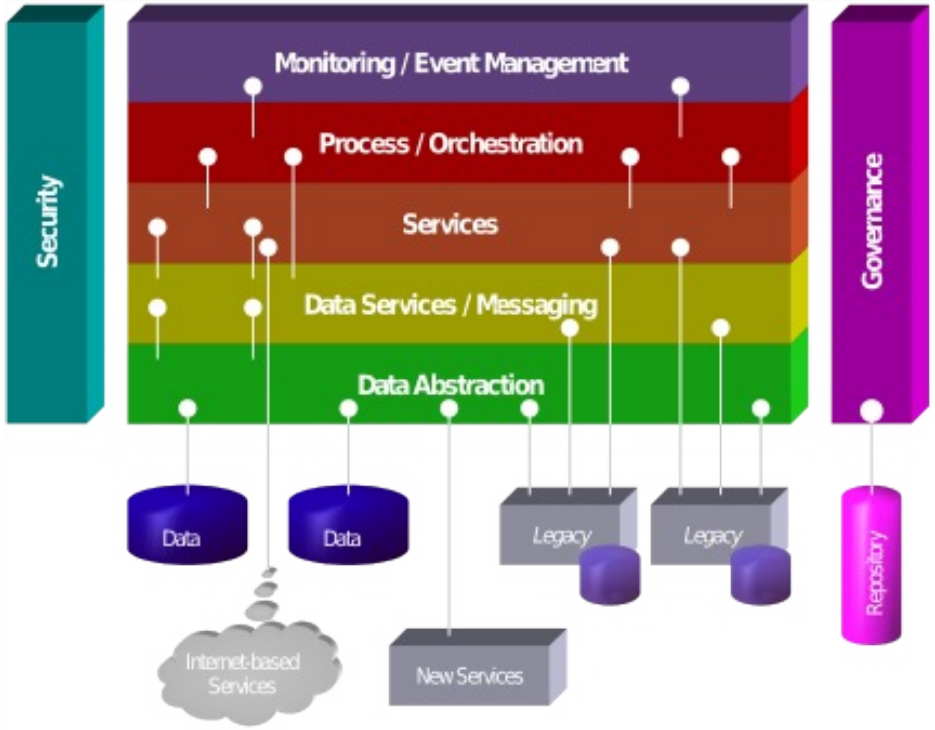
info Share ACADEMY

Wprowadzenie teoretyczne do architektury mikroserwisowej

infoShareAcademy.com

Historia mikroserwisów – SOA

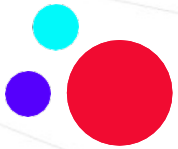
Architektura zorientowana na usługi (ang. *service-oriented architecture*, *SOA*) – koncepcja tworzenia systemów informatycznych, w której główny nacisk stawia się na definiowanie usług, które spełnią wymagania użytkownika. Pojęcie SOA obejmuje zestaw metod organizacyjnych i technicznych mający na celu powiązanie biznesowej strony organizacji z jej zasobami informatycznymi.



Źródło: https://pl.wikipedia.org/wiki/Architektura_zorientowana_na_us%C5%82ugi

DYSKUSJA

Jakie problemy realnie rozwiązują mikroserwisy?



Problemy jakie rozwiązują mikroserwisy

- Skalowalna wydajność systemu (horyntalne skalowanie)
- Skalowalny rozwój system / zarządzanie złożonością.
- Poprawnienie odporności systemu* (zależy od implementacji!)
- Niezależne i szybsze wdrożenie* (zależy od implementacji!)
- Większa elastyczność systemu
- Autonomia – luźny coupling vs kanoniczny model

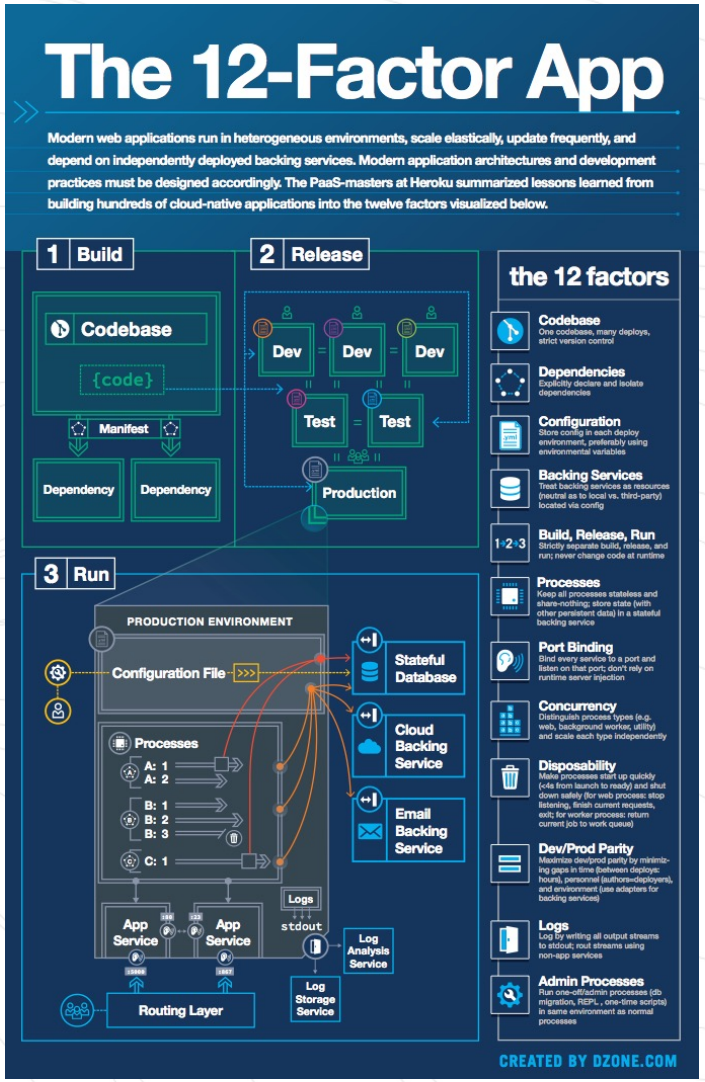
Twelve-Factor App

We współczesnym świecie oprogramowanie jest powszechnie wytwarzane w formie usługi, nazywane software-as-service (SaaS) lub aplikacjami internetowymi. Dwanaście aspektów aplikacji jest metodologią budowania aplikacji SaaS, które:

- Używają deklaratywnego formatu by zautomatyzować konfigurację aplikacji w celu zmniejszenia czasu i kosztów dołączenia nowych programistów do projektu;

- Mają czysty kontrakt z systemem operacyjnym, umożliwiając jak największą możliwość przenoszenia pomiędzy środowiskami, w których działają;

- Są dopasowane do wdrożenia na nowoczesne chmury obliczeniowe, zapobiegając potrzebie użycia serwerów i administracji systemu;



Źródło: https://dzone.com/whitepapers/the-12-factor-app-infographic

infoShare ACADEMY

Twelve-Factor App

I. Codebase

- Jedno źródło kodu śledzone systemem kontroli wersji, wiele wdrożeń

II. Zależności

- Jawnie zadeklaruj i wydziel zależności

III. Konfiguracja

- Przechowuj konfigurację w środowisku

IV. Usługi wspierające

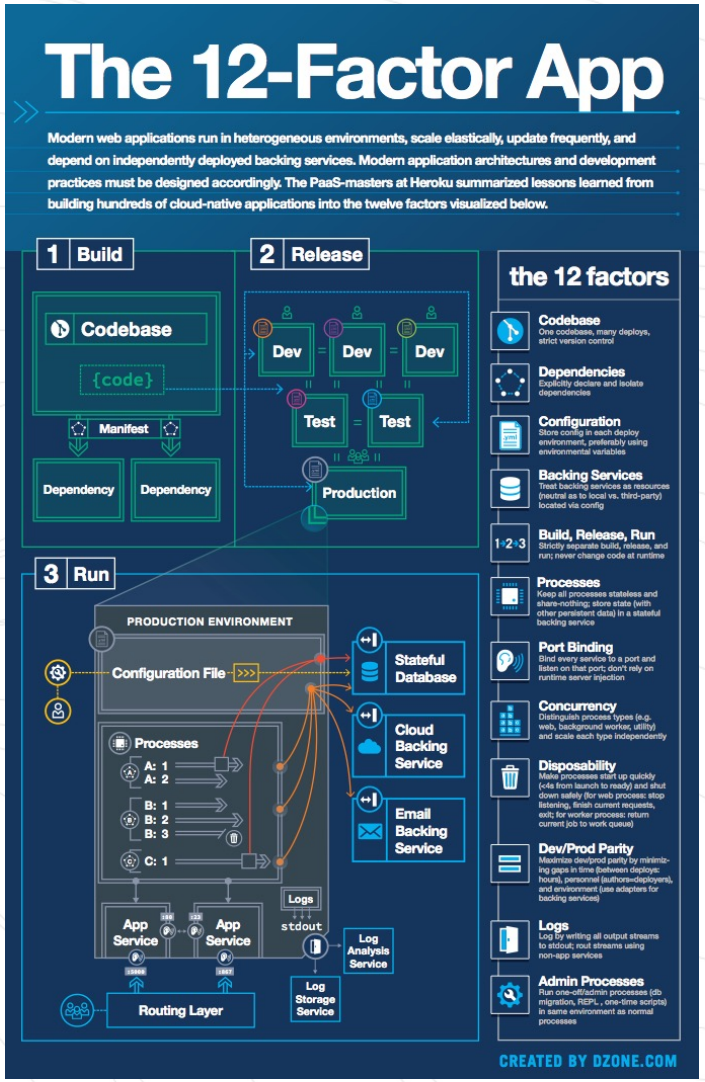
- Traktuj usługi wspierające jako przydzielone zasoby

V. Buduj, publikuj, uruchamiaj

- Oddzielaj etap budowania od uruchamiania

VI. Procesy

- Uruchamiaj aplikację jako jeden lub więcej bezstanowych procesów



infoShare ACADEMY

Twelve-Factor App

VII. Przydzielanie portów

- Udostępniaj usługi przez przydzielanie portów

VIII. Współbieżność

- Skaluj przez odpowiednio dobrane procesy

IX. Zbywalność

- Zwiększ elastyczność pozwalając na szybkie uruchamianie i zatrzymywanie aplikacji

X. Jednolitość środowisk

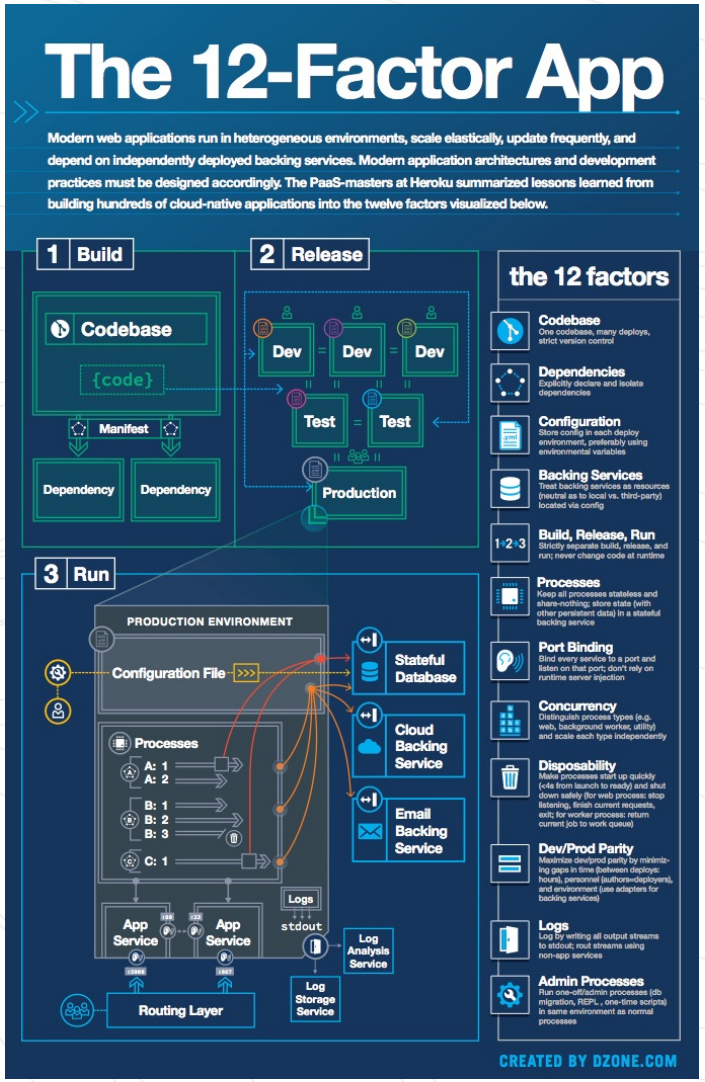
- Utrzymuj konfigurację środowisk jak najbardziej zbliżoną do siebie

XI. Logi

- Traktuj logi jako strumień zdarzeń

XII. Zarządzanie aplikacją

- Uruchamiaj zadania administracyjne jako jednorazowe procesy



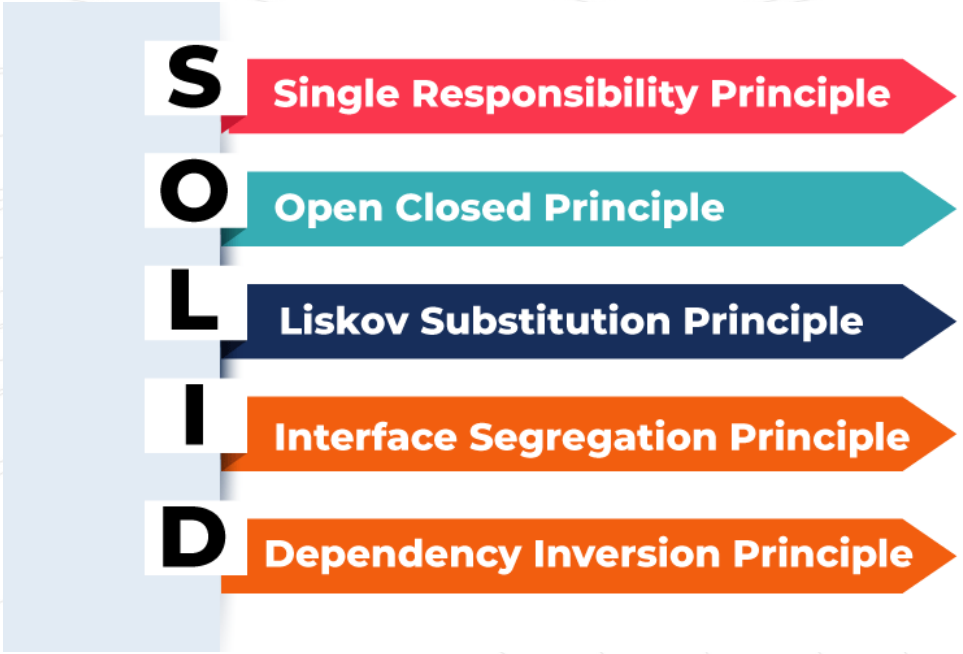
infoShare ACADEMY

DYSKUSJA

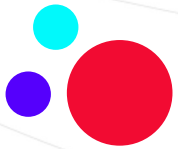
Jak „duży” powinien być mikroserwis?

Single Responsibility Principle

Popularnym sposobem na zdefiniowanie dobrego mikroserwisu jest nawiązanie do reguły pojedynczej odpowiedzialności (Single Responsibility Principle), dotyczącej projektowania klas w programowaniu obiektowym. Pojęcie odpowiedzialności jest określane jako powód do zmiany stanu. Jedna klasa powinna mieć nie więcej niż jeden powód do modyfikacji. Jeśli jest ich więcej – należy ją rozbić na mniejsze.



Źródło: https://www.geeksforgeeks.org/single-responsibility-in-solid-design-principle/



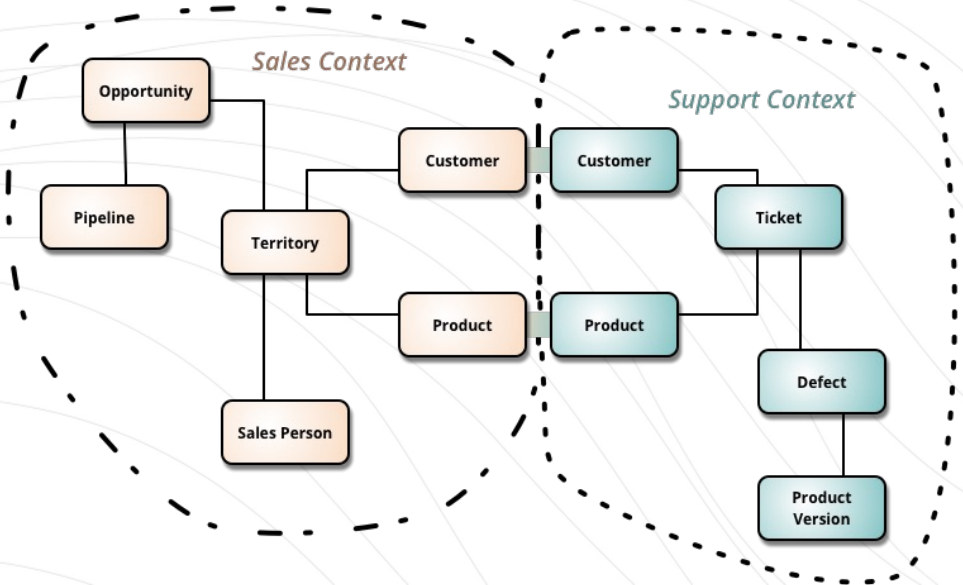
Pojedyncza domena

Ważną regułą dla architektury mikrousług jest to, że każda mikrousługa musi być właścicielem danych i logiki domeny. Tak samo jak pełna aplikacja jest właścicielem logiki i danych, dlatego każda mikrousługa musi posiadać swoją logikę i dane w ramach autonomicznego cyklu życia z niezależnym wdrożeniem na mikrousługę.

Bounding Context

Model koncepcyjny domeny będzie się różnić między podsystemami lub mikrousługami. Weź pod uwagę aplikacje dla przedsiębiorstw, w których aplikacje do zarządzania relacjami z klientami (CRM), podsystemy zakupów transakcyjnych i podsystemy obsługi klienta każdy z nich wywołuje unikatowe atrybuty i dane jednostki klienta, a każdy z nich korzysta z innego powiązanego kontekstu (BC).

Ta zasada jest podobna w projekcie opartym na domenie (DDD), gdzie każdy powiązany kontekst lub autonomiczny podsystem lub usługa musi być właścicielem modelu domeny (dane plus logika i zachowanie). Każdy kontekst ograniczony DDD jest skorelowany z jedną mikrousługą biznesową (jedną lub kilkoma usługami).

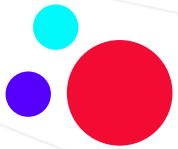


Źródło: https://martinfowler.com/bliki/BoundedContext.html
https://learn.microsoft.com/pl-pl/dotnet/architecture/microservices/architect-microservice-container-applications/data-sovereignty-per-microservice

Pojedynczy właściciel

Mikroserwis powinien mieć jednego właściciela, czyli jeden zespół, który odpowiada za ten mikroserwis.

Źródło: https://www.toppr.com/guides/legal-aptitude/jurisprudence/difference-between-possessions-and-ownership/



Przeznaczenie mikroserwisu

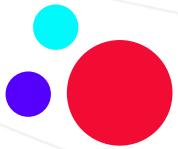
Przeznaczenie mikroserwisu moża skorelować z rodzajami mikroserwiów – tu przyjmijmy następujący podział:

1. Biznesowy
2. Agregacyjny
3. Techniczny
4. Adapter

infoShare ACADEMY

DYSKUSJA

Jak wyznaczyć granicę odpowiedzialności mikroserwisu?

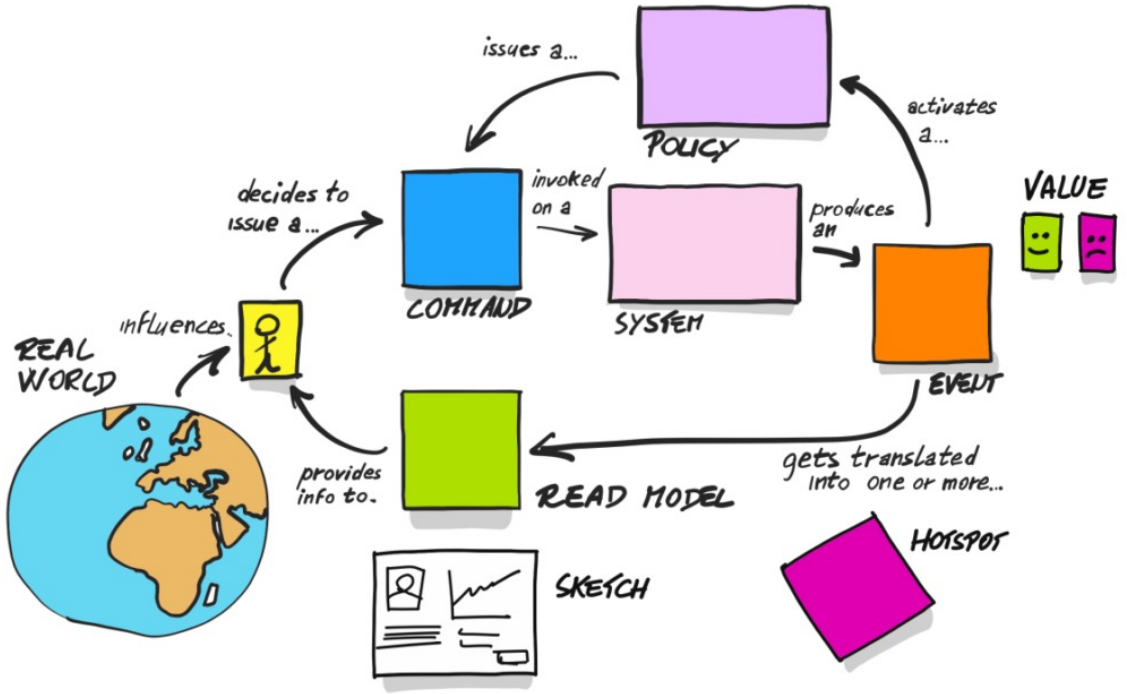


Granica odpowiedzialności jest... umowna

- Odpowiedzialność mikroserwisu nie jest wyrażona wzorem matematycznym, za pomocą którego wiemy na 100% jaka ilośc logiki biznesowej jest wystarczająca.
- Celem jest znalezienie się w punkcie o dużej elastyczności, separując od siebie newralgiczne elementy w systemie.
- Unikamy podziału na atomowe części – nadmierna granulacja – doprowadzi to do nadmiernego skomplikowania systemu.
- Nie bój się “podzielić” mikroserwisu później...

Event Storming

Event Storming jest metodą odkrywania i konkretyzowania informacji o domenie biznesowej, w ramach której wytwarzane jest oprogramowanie. Handel, Telekomunikacja, Finanse, Edukacja, Prawo, Medycyna – domen biznesowych, z którymi w codziennej pracy spotykają się programiści i eksperci techniczni można wymieniać jeszcze wiele. Co więcej, często zmiana projektu pociąga za sobą zmianę domeny biznesowej, w ramach której pracujemy.

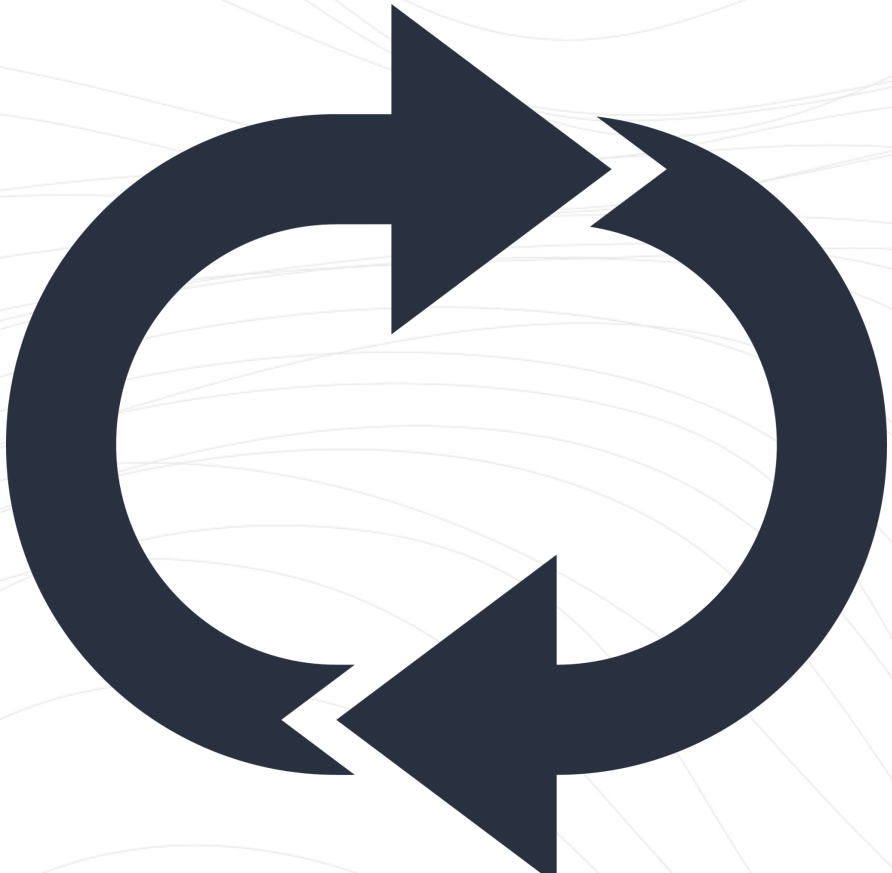


Źródło: https://devstyle.pl/2021/01/14/process-level-event-storming/
https://bulldogjob.pl/readme/event-storming-pierwszy-krok-do-ddd

infoShare ACADEMY

Zaakceptuj iterację tego procesu

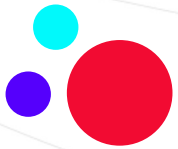
- Trzeba liczyć się z faktem, że prawdopodobnie podzielimy mikroserwisy początkowe źle, bo nie jesteśmy w stanie przewidzieć wszystkiego.
- Możemy mieć błędne założenia lub dane wejściowe, co zmusi nas do bycia elastycznym w trakcie projektowania systemu.



infoShare ACADEMY

DYSKUSJA

Czym jest coupling? Jakie rodzaje couplingu spotkaliśmy?

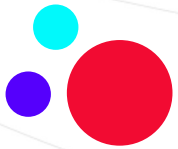


Rodzaje couplingu

- Temporal coupling
- Runtime coupling
- Data-layer coupling
- Shared / Library coupling

DYSKUSJA

Jak poprawić odporność systemu w architekturze mikroserwisowej?



Odporność systemu w architekturze mikroserwisowej

- Healtchecks
- Autoscaling
- Circuit Breaker
- Retry / Rate limiter

infoShare ACADEMY

Praktyczne wprowadzenie do Nest.js

infoShareAcademy.com

Omówienie architektury projektu

infoShareAcademy.com

infoShare ACADEMY

Q&A

infoShareAcademy.com

Koniec części teoretycznej.