TRANSITION
TECHNOLOGIES
MANAGED
SERVICES

GROUP
TRANSITION
TECHNOLOGIES

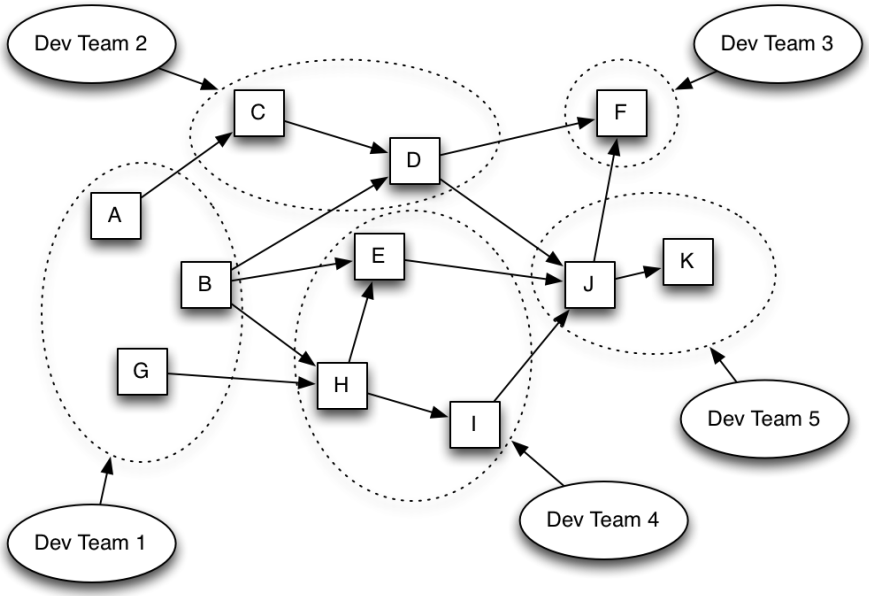
MIKROSERWISY

KRZYSZTOF ŁOPUCKI • 2018

TECHNOLOGIES THAT GUIDE INDUSTRY

Mikroserwisy

- Architektura tworzenia oprogramowania
- Niezależne i małe moduły **wykonujące swoje zadanie biznesowe**
- Komunikacja między innymi modułami
- Mikroserwisami mogą zajmować się niezależne od siebie zespoły



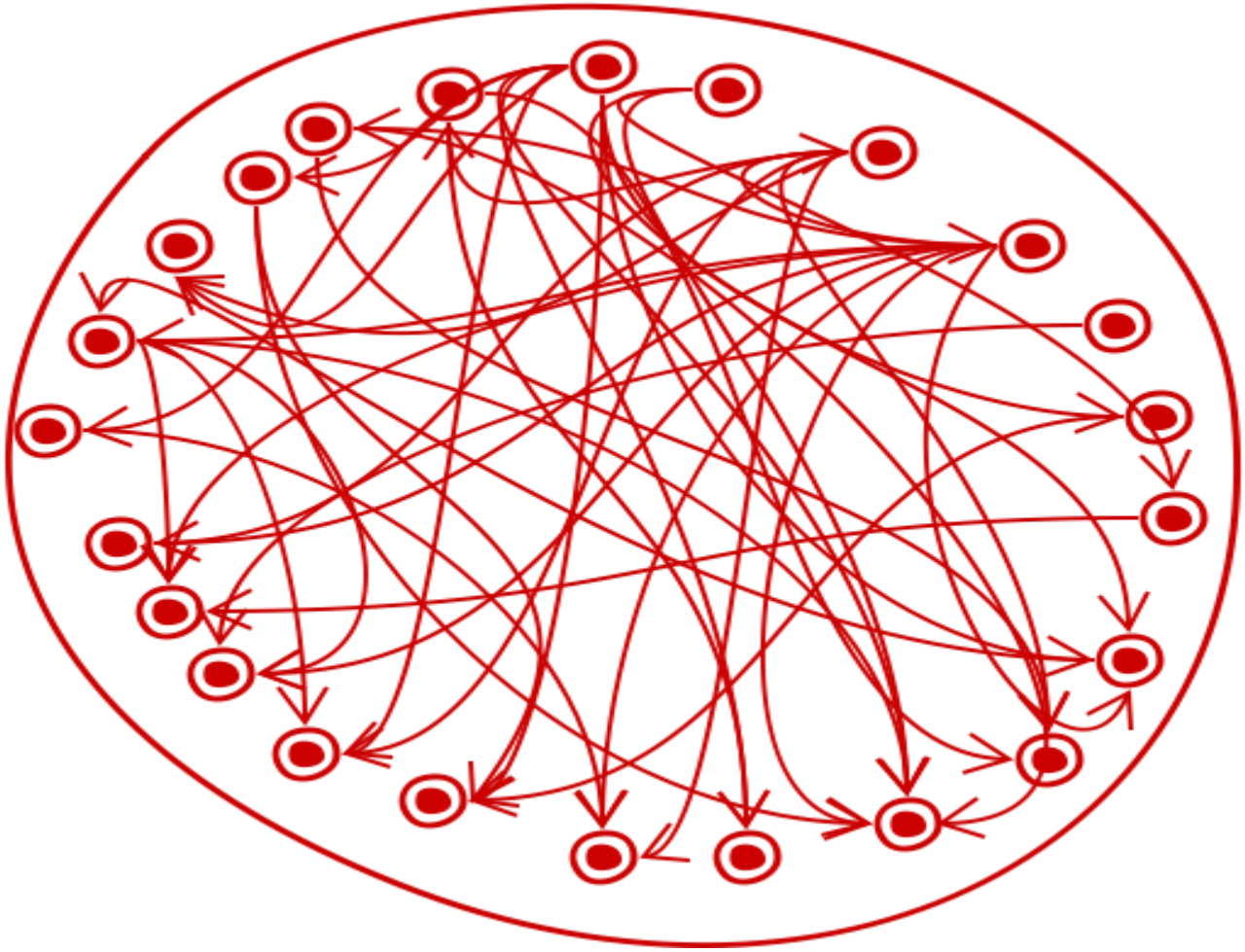
Zalety

- Poprawa modułowości projektu
- Łatwiejsze zrozumienie rozwiązania problemów architektonicznych i założeń biznesowych
- Równoległe rozwijanie aplikacja przez wiele zespołów
- Ciągłe udoskonalanie architektury (każdy moduł budujemy z innych nowszych technologii)
- Dajemy pracę DevOps'om

Wady

- Zwiększa stopień skomplikowania aplikacji
- Ujawnia problemy w zespole (komunikacja, organizacja)
- Problematyczna komunikacja między serwisami
- Trudniejsze testowanie

Jak to wygląda ?



Liczby

- Amazon – 150
- Uber – 500+
- Netflix – 2000

Jak zacząć

1. Tworzymy aplikacje
2. Tworzymy kolejne aplikacje w zależności od przeznaczenia
3. Tworzymy jeszcze więcej aplikacji

4. Łączymy aplikacje ze sobą
5. Ustawiamy dodatkowe moduły wspomagające komunikację i kontrolę nad środowiskiem (load balancery, narzędzia skalujące)
6. Testujemy rozwiązania

7. Cieszymy spokojem do póki któraś z aplikacji nie padnie

Mikroserwisy

Już to robiłeś !!!

Maven jest narzędziem przeznaczonym do zarządzania zależnościami. Odpowiada również za budowanie aplikacji i pośrednio jej uruchamiania.

Jest narzędziem konsolowym dzięki czemu łatwo zintegrować aplikację z innymi narzędziami (jira, bamboo, jenkins, git, artifactory)

mvn archetype:generate

Buduje nowy projekt jednomodułowy.

Po wpisaniu komendy zostaniemy zapytani o szereg informacji. Większość z nich zostawiamy jako default.

Ustawiamy groupId (np. pl.umcs) oraz artifactId (np. users).

Z konwencji możemy rozumieć że projekt jest pisany dla polskiej firmy UMCS i będzie udostępniał api do zarządzania użytkownikami.

Multimoduły

Utworzenie projektu złożonego z wielu modułów jest równie proste. Jedyna zmiana jaką trzeba wykonać to zmienić lub dodać wpis w poprzednio utworzonym pliku pom.

<packaging>pom</packaging>

W katalogu gdzie jest pom uruchamiamy polecenie z poprzedniego slajdu ponownie.

Kolejne moduły dodajemy przez ponowne uruchomienie polecenia z poprzedniego zlajdu w tym samym miejscu to nasz główny pom.

mvn clean

Usuwa katalogi **target** gdzie maven przechowuje zbudowaną aplikację.

mvn package

Buduje aplikację, a pliki zbudowane umieszczane są w katalogach target.

mvn install

Właściwie robi to samo co package plus dodatkowo instaluje zbudowane paczki w repozytorum lokalnym na naszych komputerach.

Lokalizacja lokalnego repozytorium:

~/.m2

Tylda oznacza katalog domowy użytkownika.

mvn test

Uruchamia testy.

Jeśli są odpowiednio skonfigurowane uruchomi testy jednostkowe lub integracyjne albo oba rodzaje testów jednocześnie.

Ćwiczenie 1

Stórz projekt wielomodułowy zbudowany z 3 modułów. Każdy z nich będzie wyświetlał inny napis po uruchomieniu. Każdy z modułów uruchamiamy oddzielnie (powinny mieć osobne klasy z metodami main).

Spring boot

W celu automatyzacji wielu procesów jak np. baza danych czy chociażby komunikacja rest lub soap, użyjemy Spring boot.

Wykorzystamy moduły z poprzedniego projektu.

Zależności maven

Dodajemy odpowiednie zależności i biblioteki konieczne do uruchomienia spring boota:

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.5.RELEASE</version>
</parent>
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
</dependency>
```

Klasa uruchamiająca kontener

- Adnotacja @SpringBootApplication nad klasą.
- Metoda main zaimplementowana jak niżej:

```
SpringApplication.run(MainBrain.class, args);
```

Klasa uruchamiająca kontener

- Adnotacja @SpringBootApplication nad klasą.
- Metoda main zaimplementowana jak niżej:

```
SpringApplication.run(MainBrain.class, args);
```

Kommand runner

- W klasie gdzie jest metoda main dodajemy runner:

```
@Bean
public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
        return args -> {
                System.out.println(„YEAH BABE!!!");
        } ;
}
```

Ćwiczenie 2

Trzy poprzednie moduły powinny być beanami springowymi.

Spring boot i rest

- Zależność w pom:

```
<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

- Odpowiedni kontroler

Ćwiczenie 3

Napisz 3 kontrolery w każdym z modułów.

Ćwiczenie 4

Korzystając z wiedzy zdobytej do tej pory utwórz zestaw mikro aplikacji będących mikroserwisami. Napiszemy prosty system księgowy.

Pierwszy z nich będzie odpowiadał za użytkowników i ich dane takie jak nip, pesel czy regon.

Drugi mikroserwis będzie miał informacje o fakturach wystawionych.

Trzeci mikroserwis przechowa informacje o kosztach użytkownika.

Czwarty mikroserwis podsumuje miesiąc użytkownika i wyliczy jego podatki.

Ćwiczenie 4

Paiętaj że mikroserwisy będą się ze sobą komunikowały.

Bardzo uprośćmy mechanizm wyliczania podatków i wyliczmy jeden podatek wg poniższego algorytmu.

- **Sumuj cały przychód użytkownika**
- **Odejmij wszystkie koszty użytkownika**
- **1/3 kwoty która zostanie to podatki !!!**

TRANSITION
TECHNOLOGIES
MANAGED
SERVICES

GROUP
TRANSITION
TECHNOLOGIES

DZIĘKUJĘ

Krzysztof Łopucki

Krzysztof.Lopucki@ttms.pl