

## Program szkolenia:

# Reactor i Spring WebFlux: programowanie reaktywne

### Informacje:

<b>Nazwa:</b>	<b>Reactor i Spring WebFlux: programowanie reaktywne</b>
<b>Kod:</b>	<b>Spring -</b>
<b>Kategoria:</b>	Spring Framework
<b>Odbiorcy:</b>	developerzy, architekci
<b>Czas trwania:</b>	3 dni
<b>Forma:</b>	10% wykład, 90% ćwiczenia

---

Szkolenie wprowadza w temat programowania reaktywnego na przykładzie biblioteki Reactor. Kładzie duży nacisk na zrozumienie fundamentalnych pojęć i mechanizmów oraz świadomie stosowanie w rzeczywistych projektach.

Szkolenie ma bardzo praktyczny charakter, uczestnicy najpierw pracują z serią małych ćwiczeń pod okiem instruktora. Na koniec wspólnie przygotowujemy większą reaktywną aplikację opartą o framework Spring. Może być to rozproszona baza danych, strumień zdarzeń, itp. - w zależności od potrzeb uczestników.

### Zalety szkolenia:

- niemal całe szkolenie to praktyczny warsztat i kodowanie na żywo
- idiomatyczne podejście do programowania reaktywnego, od kontrolera po bazę
- całkowity brak slajdów

## Szczegółowy program:

### 1. Podstawy

1.1. Czym jest programowanie reaktywne

1.2. CompletableFuture i pule wątków

1.3. Biblioteka Reactor

1.4. Jak stworzyć strumień

1.4.1. ``just()``, ``generate()``, ``create()``, ``fromCallable()``, ``fromStream()``

1.5. Leniwość, strumienie zimne i gorące

1.6. Podstawowe operatory

1.6.1. ``map()``, ``filter()``, ``filterWhen()``, ``flatMap()``, ``handle()``, ``take()``, ``skip()``

1.6.2. ``doOn*()`` operators

1.6.3. ``window()``, ``buffer()``, ``distinct()``

1.6.4. ``cast()``, ``ofType()``, ``index()``

1.6.5. ``timestamp()``, ``elapsed()``

1.6.6. ``zip()``, ``merge()``

1.7. Obsługa błędów

1.7.1. ``timeout()``, ``retry*()``, ``retryBackoff()``

1.7.2. ``onError*()``

1.8. Blokowanie i wielowątkowość

1.8.1. ``subscribeOn()``, ``publishOn()``

1.8.2. ``parallel()``

1.9. Testowanie jednostkowe

### 2. Zaawansowane

2.1. Nieblokujący kod

2.2. Zaawansowana obsługa błędów

2.3. `transform()` vs. `compose()`

2.4. Zaawansowane operatory

2.4.1. `groupBy()`, `window()`

2.4.2. `reduce()`, `scan()`

2.4.3. `expand*()`

2.5. Backpressure

2.6. `Processor`

2.7. Testowanie z wirtualnym czasem

2.8. Kompatybilność z RxJava

### 3. Zastosowania praktyczne

3.1. `ConnectableFlux`

3.2. Refaktoring istniejącej aplikacji

3.3. Spring Boot

3.4. Reaktywny dostęp do bazy danych

3.4.1. Reaktywne kontrolery

3.4.2. `WebFilter`

3.4.3. Globalna obsługa błędów

3.4.4. Web sockety

3.5. Strumieniowanie danych na wejściu i wyjściu

3.6. Debugowanie

3.6.1. `checkpoint()`, `onOperatorDebug()`, `doOn*()`