BO·TT·EGA
IT minds

Program szkolenia:

Czysta architektura i Domain-driven Design w Pythonie

Informacje:

Nazwa:	**Czysta architektura i Domain-driven Design w Pythonie**
Kod:	**ddd-clean**
Kategoria:	Architektura systemów i aplikacji
Odbiorcy:	developerzy, architekci
Czas trwania:	3 dni
Forma:	30% wykłady / 70% warsztaty

Wprowadzenie do DDD i budowania modularnych aplikacji w Pythonie z wykorzystaniem czystej architektury. Szkolenie oparte o studium przypadku.

Dzień 1: Wstęp do Domain-Driven Design i modularności - organizujemy projekt

Dzień 2: Wzorce taktyczne - czysta architektura + DDD - piszemy testowalny i łatwo utrzymywalny kod

Dzień 3: Zagadnienia przekrojowe, wstrzykiwanie zależności oraz CQRS - zajmujemy się tymi "przyziemnymi" problemami

Zalety szkolenia:

- Modularyzacja systemu
- Architektura systemowa i aplikacyjna
- Praktyczne zastosowanie DDD

BOTTEGA IT minds

Szczegółowy program:

1. Wprowadzenie do Domain-Driven Design

- 1.1. Dlaczego potrzebujemy modularności?
  - 1.1.1. Utrzymywalność oprogramowania
  - 1.1.2. Skalowanie wytwarzania oprogramowania
  - 1.1.3. Łatwiejsze rozbijanie projektu na mniejsze
  - 1.1.4. Pogodzenie różnych perspektyw
  - 1.1.5. Pogodzenie różnych interesariuszy
- 1.2. Jak DDD pomaga modularyzować
  - 1.2.1. Jeden model to z mało…
  - 1.2.2. …zamiast tego używamy kilku, wokół których organizujemy usługi / komponenty
  - 1.2.3. DDD, a organizacja zespołów

2. DDD i modularność w praktyce - szukamy Bounded Contextów

- 2.1. Poddomeny - problem space
- 2.2. Bounded Contexty - solution space
- 2.3. Heurystyki znajdowania Bounded Contextów
  - 2.3.1. Subdomeny
  - 2.3.2. Inni interesariusze
  - 2.3.3. Czy możemy to wyciągnąć i sprzedać osobno? Czy ktoś już to zrobił?
  - 2.3.4. Czy możemy znaleźć jedno źródło prawdy dla każdej informacji?
  - 2.3.5. Czy nie występują zjawiska data envy / feature envy?
  - 2.3.6. Czy używamy tych samych nazw na struktury w kodzie które działają inaczej (np. na modele)?
  - 2.3.7. Czy mylimy sposób działania biznesu z interfejsem użytkownika?
  - 2.3.8. Inne

BOTTEGA IT minds

2.4. Budujemy mapę kontekstów

3. Budujemy modularny monolit na podstawie Bounded Contextów

3.1. Dlaczego modularny monolit?

3.1.1. Łatwe wdrożenie

3.1.2. Tańsze eksperymenty i refaktoryzacja

3.2. Budowanie API dla komponentów

3.2.1. Fasada

3.2.2. Serwisy

3.2.3. Przypadki użycia

3.2.4. Komendy

3.2.5. Zdarzenia

3.3. Hermetyzacja komponentu

3.3.1. Poza API zawartość pozostaje prywatna

3.4. Pilnowanie granic poprzez weryfikację importów i testy architektury

3.4.1. pylint-forbidden-imports

3.4.2. imports-linter

3.5. Sposoby integracji

3.5.1. Wywoływanie API innego komponentu

3.5.2. Port / Adapter

3.5.3. Zdarzenia

3.5.3.1. Asynchronicznie

3.5.3.2. Synchronicznie

4. Czysta architektura - wzorce taktyczne

4.1. Przypadek użycia

BOTTEGA IT minds

4.1.1. Często punkt startowy

4.1.2. Logika orkiestracji

4.2. Encje

4.2.1. Logika biznesowa ważna w każdym kontekście

4.2.2. Trzymają dane i pilnują poprawności

4.3. Repozytoria

4.3.1. Zorientowane na persystencję

4.3.2. Repozytoria - kolekcje

4.4. Porty

4.4.1. Definicja “wtyczki” do logiki biznesowej

4.4.2. Implementowana jako klasa abstrakcyjna

4.5. Adaptery

4.5.1. Implementacja “wtyczki” do logiki biznesowej

5. DDD - wzorce taktyczne

5.1. Agregaty

5.1.1. Znajdowanie agregatów - heurystyki

5.1.2. Powody stosowania agregatów

5.1.3. Różnice między agregatem, a encją

5.2. Serwisy domenowe

5.2.1. Kiedy odpowiedzialność nie pasuje do innego obiektu?

5.3. Polityki

5.4. Zdarzenia domenowe

5.4.1. Implementacja z użyciem DTO

5.5. Inne wzorce taktyczne

5.5.1. Fabryka

BOTTEGA IT minds

5.5.2. Process Manager / Saga

6. Kiedy nie stosować czystej architektury lub wzorców taktycznych DDD?

6.1. Prosty problem

6.2. Prototypowanie

6.3. CRUD

6.4. Proxy nad API firmy trzeciej

7. Rozwiązanie problemu zarządzania zależnościami

7.1. Kontenery IoC i ich rola

7.1.1. Inicjalizacja kontenera podczas uruchamiania aplikacji

7.2. Przykłady dojrzałych narzędzi dostępnych w Pythonie

7.2.1. Injector

7.2.2. Dependency Injector

7.3. Zakresy (ang. scopes) w kontenerach IoC

7.3.1. Singleton

7.3.2. Threadlocal

7.3.3. Context vars

7.4. Dobre praktyki używania wstrzykiwania zależności

7.4.1. Kod aplikacyjny nie wie nic o kontenerze IoC

7.4.2. Nie używamy kontenera bezpośrednio (antywzorzec Service Locator)

7.4.3. Nie wstrzykujemy zależności po to, by przekazać je dalej

7.4.4. Nadal możemy łatwo nawigować po kodzie

8. Modularyzacja aplikacji z kontenerem IoC

8.1. Zarządzanie konfiguracją

8.2. Każdy komponent jako moduł kontenera IoC - składamy aplikację jak z klocków

8.3. Wykorzystanie w testach

BOTTEGA IT minds

9. CQRS

- 9.1. Uzasadnienie modelu do odczytu
- 9.2. Implementacja
  - 9.2.1. Stos do zapisu (ang. write stack)
    - 9.2.1.1. Command + Command Handler
    - 9.2.1.2. Wpływ na hermetyzację komponentu
  - 9.2.2. Stos do odczytu (ang. read stack)
    - 9.2.2.1. różnica tylko na poziomie kodu
    - 9.2.2.2. różnica na poziomie zapisu danych (kilka tabel/kolekcji/indeksów etc)
    - 9.2.2.3. różnica na poziomie bazy danych (kilka baz danych)