BOTTEGA IT minds

Program szkolenia:

Refaktoryzacja w Django

Informacje:

Nazwa:	**Refaktoryzacja w Django**
Kod:	**django-refactor**
Kategoria:	Python
Odbiorcy:	developerzy, architekci
Czas trwania:	3 dni
Forma:	20% wykłady, 80% warsztaty

Szkolenie z refaktoryzacji i porządkowania projektów napisanych w Django z typowymi dla tego frameworka bolączkami.

Dzień 1: Antywzorce i zapachy kodu w projektach z Django, rozbijanie projektu wertykalnie z użyciem DDD

Dzień 2: Wyciąganie odpowiedzialności z building blocks i wprowadzanie nowych wzorców

Dzień 3: Wyciąganie odpowiedzialności z building blocks i wprowadzanie nowych wzorców cd., Testowanie

Zalety szkolenia:

- Wzorce i antywzorce
- Wyjście z architekturą poza framework
- Domain-driven Design

BO·TT·EGA
IT minds

Szczegółowy program:

1. Po co refaktoryzujemy? Jak to robić?

- 1.1. Refaktoryzacja do czegoś
- 1.2. Refaktoryzacja w kierunku czegoś
- 1.3. Wybór miejsca refaktoryzacji

2. Powszechne antywzorce

- 2.1. Zbyt duże modele
  - 2.1.1. Kohezja i zbyt obszerne modele
  - 2.1.2. Brak realnego wertykalnego podziału (na przykład z DDD)
  - 2.1.3. Łączenie zbyt wielu odpowiedzialności które można by wydzielić np. z użyciem Gateway Pattern
  - 2.1.4. Wyjątek: model jako Process Manager
- 2.2. Przypadkowy podział na aplikacje
  - 2.2.1. Aplikacje importujące dużo kodu z siebie wzajemnie
  - 2.2.2. Feature i data envy
  - 2.2.3. Aplikacja “core”
- 2.3. Logika, którą trudno zrozumieć
  - 2.3.1. W widokach lub serializerach z DRF
  - 2.3.2. Poukrywana w obsłudze sygnałów
- 2.4. Niewydajne widoki
  - 2.4.1. Profilowanie z django-silk
  - 2.4.2. Delegowanie zadań w tło przy użyciu kolejek jak Celery
  - 2.4.3. Modele do odczytu (ang. read models)
- 2.5. Niska testowalność
  - 2.5.1. Nieczytelne testy

BOTTEGA IT minds

- 2.5.2. Wolne testy
- 2.5.3. Niestabilne testy
- 2.5.4. Niskie pokrycie kodu testami
- 2.5.5. Brak możliwości testowania bez używania bazy danych
- 2.6. Pohakowany admin
- 2.6.1. Do czego admin powstał, a do czego jest używany

3. Naprawianie podziału wertykalnego z Domain-Driven Design

- 3.1. Symptomy nietrafionej modularyzacji
- 3.2. Słabości pojedynczego modelu i patrzenia na wszystko w jeden sposób
- 3.3. Wprowadzanie granic między aplikacjami
- 3.3.1. Fasada
- 3.3.2. Para port / adapter
- 3.3.3. Zdarzenia
- 3.4. Walidowanie granic między aplikacjami

4. Building blocks w Django i odpowiedzialności

- 4.1. Modele
- 4.2. Object Managery
- 4.3. Signal Handlers
- 4.4. Serializer
- 4.5. Widok
- 4.6. Formularz

5. Rozszerzanie standardowego zestawu building blocks

- 5.1. Data Transfer Object
- 5.2. Przekazywanie danych bez słowników
- 5.3. Walidować czy nie?

BOTTEGA IT minds

- 5.4. Biblioteki
  - 5.4.1. Pydantic
  - 5.4.2. dataclasses
  - 5.4.3. attrs
- 5.5. Service layer
  - 5.5.1. Kiedy stosować?
  - 5.5.2. Jaką logikę w nim umieścić?
  - 5.5.3. Jakiej logiki nie umieszczać w serwisie?
- 5.6. Własne sygnały używane jako zdarzenia
- 5.7. Fasada
  - 5.7.1. Jako API pomiędzy aplikacjami
- 5.8. Value Object
- 5.9. Wstrzykiwanie zależności (ang. dependency injection)

6. Testowanie

- 6.1. Przegląd tego, co mamy w Django
- 6.2. Zrównoleglanie testów
- 6.3. Pisanie testowalnego kodu w Django