

SPIS TREŚCI

1. WPROWADZENIE	11
1.1. Era narzędzi	11
1.2. Dla kogo jest ta książka?	13
1.3. Co znajdziesz w książce?	13
2. PODSTAWY CZYSTEJ ARCHITEKTURY	15
2.1. Po co to wszystko?	15
2.2. System płytki kontra system głęboki	16
2.2.1. CRUD, czyli system płytki	16
2.2.2. System głęboki	17
2.3. Założenia czystej architektury	20
2.3.1. Niezależność od frameworków	20
2.3.2. Wysoka testowalność	20
2.3.3. Niezależność od API i interfejsu użytkownika	20
2.3.4. Niezależność od bazy danych	21
2.3.5. Niezależność od firm trzecich	21
2.3.6. Elastyczność	21
2.3.7. Rozszerzalność	21
2.4. Warstwy, czyli horyzontalna organizacja kodu	22
2.4.1. Świat zewnętrzny	22
2.4.2. Infrastruktura	23
2.4.3. Aplikacja	23
2.4.4. Domena	24
2.4.5. Zasada zależności	26
2.4.6. Granice	26
2.5. Podsumowanie	29
IMPLEMENTOWANIE CZYSTEJ ARCHITEKTURY W PYTHONIE	
3. WZORCOWA IMPLEMENTACJA	30
3.1. Oznajmienie	30
3.2. Przepływ sterowania w czystej architekturze	30

3.3. Wymagania biznesowe	31
3.4. Implementacja	33
3.4.1. Diagram sekwencji	33
3.4.2. Granica wejściowa (input boundary)	33
3.4.3. Granica wyjściowa (output boundary)	34
3.4.4. Prezenter (presenter)	34
3.4.5. Model widoku (view model)	36
3.4.6. Przypadek użycia (use case)	36
3.4.7. Interfejs dostępu do danych (data access interface)	38
3.4.8. Dostęp do danych (data access)	38
3.4.9. Encja oferty (bid)	38
3.4.10. Encja aukcji (auction)	39
3.5. Podsumowanie	40
4. MODYFIKACJE CZYSTEJ ARCHITEKTURY	41
4.1. Dylemat prezentera	41
4.2. Pozbywamy się granicy wejściowej	44
4.3. Alternatywne podejścia do projektowania przypadków użycia	45
4.3.1. Fasada	45
4.3.2. Mediator pomiędzy wejściowym DTO a przypadkiem użycia	46
4.4. Użycie modeli bazodanowych jako encji	48
4.5. Podsumowanie	55
5. WSTRZYKIWANIE ZALEŻNOŚCI	57
5.1. Czym są zależności?	57
5.2. Wszędobylskie abstrakcje i klasy	58
5.3. Abstrakcje w czystej architekturze	61
5.4. Odwrócenie sterowania a zależności	64
5.5. Kontener IoC kontra service locator	66
5.6. Wstrzykiwanie zależności kontra konfiguracja	68
5.7. Podsumowanie	68
6. CQRS	70
6.1. Wstęp	70

6.2. Co to ma wspólnego z czystą architekturą?	72
6.3. Osobny stos odczytu — dlaczego?	73
6.4. Osobny stos odczytu — jak?	74
6.4.1. Zapytanie jako DTO	75
6.4.2. Zapytania jako osobne klasy	76
6.4.3. Fasada modelu do odczytu	77
6.5. CQRS kontra REST API	78
6.6. CQRS kontra GraphQL	79
6.7. Podsumowanie	81
7. OSTRE GRANICE	82
7.1. Słowo o złożoności	82
7.2. Dwa światy	84
7.3. Granica pomiędzy warstwą aplikacji a światem zewnętrznym	85
7.4. Pisanie wejściowego DTO	85
7.5. Value objects	86
7.6. Podsumowanie	89
8. STUDIUM PRZYPADKU — PLATFORMA AUKCYJNA	90
8.1. Jak pracujemy?	90
8.2. Jak zacząć, czyli chodzący szkielet	91
8.3. Nasz chodzący szkielet	93
8.4. Przypadek użycia dla składania oferty na aukcji	94
8.4.1. Nazewnictwo	94
8.4.2. Argumenty	94
8.4.3. Wyjście	95
8.4.4. Testy	96
8.5. Encje aukcji i oferty	97
8.5.1. Nazewnictwo	97
8.5.2. Value objects jako identyfikatory	98
8.5.3. Implementacja	98
8.5.4. Testy jednostkowe	99
8.5.5. Implementacja — ciąg dalszy	100

8.6. Abstrakcyjne repozytorium	101
8.6.1. Nazewnictwo	102
8.6.2. Implementacja	102
8.7. Repozytorium	102
8.7.1. Nazewnictwo	102
8.7.2. Implementacja działająca w pamięci	103
8.7.3. Rozwijanie implementacji pod osłoną TDD	103
8.8. Kończymy przypadek użycia — składanie oferty	105
8.8.1. Wstrzykiwanie zależności	105
8.8.2. Sprawiamy, że pierwszy sensowny test przechodzi	105
8.8.3. Refaktoryzacja	106
8.9. Organizacja kodu	107
8.9.1. Jak można ułożyć kod w Pythonie?	107
8.9.2. Organizujemy kod projektu	108
8.9.3. Organizujemy kod warstwy infrastruktury	110
8.9.4. Łączymy wszystko razem w komponencie main	112
8.9.5. Wystawiamy API	116
IMPLEMENTOWANIE CZYSTEJ ARCHITEKTURY W PYTHONIE	
8.10. Finalizujemy aukcję w kolejnym przypadku użycia	118
8.10.1. Zarys przypadku użycia i wejściowe DTO	119
8.10.2. Rozszerzamy encję, by spełnić nowe wymagania	120
8.10.3. Skoro encje nie powinny mieć żadnych zależności, to czy mogą pytać o czas?	122
8.10.4. Wprowadzamy port dla płatności	123
8.10.5. Implementujemy adapter	125
8.10.6. Obsługa błędów kontra zasada zależności	126
8.10.7. A co, gdybyśmy chcieli dodać zapamiętywanie karty płatniczej?	127
8.10.8. Jak żyć, gdy adapter rośnie?	129
8.10.9. Bramka płatności ma już SDK. Nie możemy go po prostu użyć?	131

- 8.11. Przypadek użycia — rozpoczynanie nowej aukcji 131
 - 8.11.1. Skąd się biorą nowe aukcje? 131
 - 8.11.2. Encja aukcji i jej opis w jednym obiekcie — za i przeciw 132
 - 8.11.3. Wprowadzamy deskryptor 134
 - 8.11.4. Repozytorium z interfejsem kolekcji 134
 - 8.11.5. Które repozytorium wybrać? 137
- 8.12. Operacje odczytu danych 138
 - 8.12.1. Podejście z przypadkami użycia 138
 - 8.12.2. CQRS na ratunek 140
 - 8.12.3. Zapytania jako klasa 140
 - 8.12.4. Model do odczytu 142
 - 8.12.5. Podsumowanie operacji odczytujących dane 144
- 8.13. Odwracamy kontrolę za pomocą zdarzeń 144
 - 8.13.1. Przykład — wysyłka e-maili 144
 - 8.13.2. Techniki odwracania kontroli 145
 - 8.13.3. Implementacja zdarzeń 146
 - 8.13.4. Skąd wziąć szynę zdarzeń? 147
 - 8.13.5. Jak wy dostać zdarzenia z encji? 148
 - 8.13.6. Encja gromadzi zdarzenia,
które potem publikuje repozytorium 148
 - 8.13.7. Encja zwraca zdarzenia z metod, które zmieniają jej stan 149
 - 8.13.8. Testowanie encji, które zwracają zdarzenia 151
 - 8.13.9. Subskrybowanie się na zdarzenia 152
 - 8.13.10. Zdarzenia kontra transakcje kontra efekty uboczne 153
 - 8.13.11. Niezawodne publikowanie zdarzeń — outbox pattern 155
 - 8.13.12. Wprowadzamy jednostkę pracy 156
 - 8.13.13. Czas życia jednostki pracy 158
 - 8.13.14. Relacja pomiędzy jednostką pracy a szyną zdarzeń 158
- 8.14. Radzimy sobie z innymi przekrojowymi zagadnieniami 160
 - 8.14.1. Konfiguracja 161
 - 8.14.2. Walidacja 162

8.14.3. Synchronizacja	163
8.15. Podsumowanie	166
9. MODULARNOŚĆ	167
9.1. Ciężar sukcesu — rozrost i ciągłe zmiany	167
9.2. Komponenty i kohezja	167
9.3. Organizacja kodu według komponentu	168
9.4. Komponenty i swoboda architektoniczna	170
9.5. Komponenty kontra mikroserwisy	170
9.6. Komponenty a użytkownik	172
9.7. Komponenty a bounded context	173
9.8. Komponenty — implementacja	173
9.9. Zależności między komponentami	175
9.9.1. Oddzielne drogi	175
9.9.2. Bezpośrednia zależność — oba komponenty implementują czystą architekturę	176
9.9.3. Niebezpośrednia zależność — oba komponenty implementują czystą architekturę	177
9.9.4. Zależność, gdy jeden z komponentów nie implementuje czystej architektury	178
9.9.5. Odmiany integracji za pomocą zdarzeń	178
9.9.6. Zależności między komponentami — podsumowanie	179
9.10. Studium przypadku — platforma aukcyjna	179
9.10.1. Odkrywamy komponenty	179
9.10.2. Komponenty platformy aukcyjnej	180
9.10.3. Co komponent wystawia na zewnątrz?	181
9.10.4. Tam, gdzie wszystko składa się w całość — komponent main	185
9.10.5. Korzystamy z komponentu main do uruchomienia aplikacji	186
9.10.6. Jedna architektura dla wszystkich komponentów — czy to możliwe?	187
9.10.7. Zależności pomiędzy komponentami	189
9.10.8. Integrowanie komponentów za pomocą zdarzeń	191

- 9.10.9. Wewnętrzna obsługa zdarzeń w tym samym komponencie 197
- 9.10.10. Integracja różnych komponentów za pomocą zdarzeń — prosty przypadek 198
- 9.10.11. Integracja różnych komponentów za pomocą zdarzeń — złożony przypadek 201
- 9.10.12. Inne ciekawe zastosowania menadżera procesu 206
- 9.10.13. Menadżer procesu kontra wyścigi 207
- 9.11. Podsumowanie 210

IMPLEMENTOWANIE CZYSTEJ ARCHITEKTURY W PYTHONIE

10. TESTOWANIE 212

- 10.1. Strategia testowania i odmiany funkcji 212
 - 10.1.1. Piramida testów — mit czy jedyna słuszna droga? 213
 - 10.1.2. Rodzaje testów 215
 - 10.1.3. Jak przetestować przeglądarkę do bazy danych? 218
 - 10.1.4. Jak przetestować proxy? 220
 - 10.1.5. Jak przetestować system głęboki? 223
- 10.2. Odkrywamy testowanie jednostkowe na nowo 224
 - 10.2.1. Ile musi wiedzieć test? 224
- 10.3. Testowanie stanu kontra testowanie interakcji 228
 - 10.3.1. Rodzaje weryfikacji 228
 - 10.3.2. Niebezpieczeństwa związane z inspekcją stanu 229
 - 10.3.3. Niebezpieczeństwa związane ze sprawdzaniem interakcji 231
 - 10.3.4. Stuby kontra mocki 232
 - 10.3.5. Rodzaje obiektów dublerów 233
- 10.4. Testujemy cały komponent jednostkowo 234
 - 10.4.1. Ustawiamy komponent w pożądanym stanie 235
 - 10.4.2. Wywołujemy akcję na komponencie 237
 - 10.4.3. Weryfikujemy rezultat akcji na poziomie komponentu 237
 - 10.4.4. Radzimy sobie z zależnościami w postaci portów i repozytoriów 240
- 10.5. Podsumowanie 241

11. ZAKOŃCZENIE	243
11.1. Co dalej?	243
12. SUPLEMENT A: MIGRACJA Z PROJEKTU ODZIEDZICZONEGO	245
12.1. Czy powinno się to robić?	245
12.2. Jak to zrobić?	245
12.3. „Nie mogę przestać dostarczać nowych funkcji!”	247
13. SUPLEMENT B: WPROWADZENIE DO EVENT SOURCING	248
13.1. Co to jest event sourcing?	248
13.2. Agregat z event sourcing kontra agregat z domain-driven design	250
13.3. Prosty przykład agregatu	251
13.3.1. Zamówienie jako encja	251
13.3.2. Istotne zmiany zamówienia w formie zdarzeń	252
13.3.3. Uwaga na te zdarzenia!	253
13.3.4. Zamówienie jako agregat	254
13.3.5. Testowanie agregatów	257
13.4. Persystencja	257
13.4.1. Nowe zdarzenia są dołączane na koniec strumienia zdarzeń	257
13.4.2. Pobieranie strumienia zdarzeń	259
13.4.3. Dopisywanie nowych zdarzeń do strumienia	260
13.4.4. Wybór bazy danych — podsumowanie wymagań	261
13.4.5. Przykładowa implementacja z użyciem PostgreSQL	261
13.4.6. Użycie event store	266
13.4.7. Co robić, gdy wykryjemy wyścig?	266
13.4.8. Użycie repozytorium do ukrycia event store	268
13.4.9. Migawki stanu agregatu	268
13.5. Projekcje	272
13.6. Event sourcing w aplikacji składającej się z komponentów	276
13.6.1. Event sourcing to szczegół implementacyjny komponentu	276
13.6.2. Stosuj zdarzenia domenowe na potrzeby integracji	277
13.7. Podsumowanie	277
BIBLIOGRAFIA	279