

Spis treści

przedmowa xxiii

podziękowania xxvi

o książce xxviii

o autorze xxxi

o ilustracji na okładce xxxii

1. Ucieczka z monolitycznego piekła 1

1.1. Powolny marsz w kierunku monolitycznego piekła 2

1.1.1. Architektura aplikacji FTGO 3

1.1.2. Zalety architektury monolitycznej 4

1.1.3. Życie w monolitycznym piekle 4

- Kompleksowość przeraża deweloperów 4
- Rozwój jest powolny 5
- Ścieżka od zatwierdzenia do wdrożenia jest długa i mozolna 5
- Skalowanie jest trudne 6
- Dostarczanie niezawodnego monolitu jest wyzwaniem 6
- Blokada w coraz bardziej przestarzałym stosie technologicznym 7

1.2. Dlaczego ta książka jest dla ciebie odpowiednia 7

1.3. Czego się nauczysz z tej książki 8

1.4. Architektura mikroserwisowa na ratunek 8

1.4.1. Sześcian skal i mikroserwisy 9

- Skalowanie w osi X równowagę obciążenia dotyczące żądań między wiele instancji 9 ■ Skalowanie w osi Z trasuje żądania na podstawie atrybutu żądania 10 ■ Skalowanie w osi Y funkcjonalnie rozkłada aplikację na usługi 11
- 1.4.2. Mikroserwisy jako forma modułowości 11
- 1.4.3. Każda usługa ma własną bazę danych 12
- 1.4.4. Architektura mikroserwisowa FTGO 12
- 1.4.5. Porównanie architektury mikroserwisowej i SOA 13
- 1.5. Zalety i wady architektury mikroserwisowej 14
 - 1.5.1. Zalety architektury mikroserwisowej 14
 - Umożliwianie ciągłego dostarczania i wdrażania dużych, złożonych aplikacji 15 ■ Każda usługa jest mała i łatwa w utrzymaniu 15 ■ Usługi są niezależnie skalowalne 16 ■ Lepsza izolacja błędów 16 ■ Łatwe eksperymentowanie i wdrażanie nowych technologii 17
 - 1.5.2. Wady architektury mikroserwisowej 17
 - Znalezienie odpowiednich usług jest trudne 17 ■ Systemy rozproszone są złożone 17 ■ Wdrażanie funkcjonalności obejmujących wiele usług wymaga starannej koordynacji 18 ■ Decyzja o przyjęciu jest trudna 18
- 1.6. Język wzorców architektury mikroserwisowej 19
 - 1.6.1. Architektura mikroserwisowa nie jest złotym środkiem 19
 - 1.6.2. Wzorce i języki wzorców 20
 - Sily: kwestie, które musimy wziąć pod uwagę, kiedy rozwiązujemy problem 21
 - Wynikowy kontekst: konsekwencje zastosowania wzorca 21 ■ Wzorce powiązane: pięć różnych rodzajów relacji 22
 - 1.6.3. Omówienie języka wzorców architektury mikroserwisowej 23
 - Wzorce do dekompozycji aplikacji na usługi 23 ■ Wzorce komunikacyjne 24 ■ Wzorce spójności danych do wdrażania zarządzania transakcjami 25 ■ Wzorce odpytywania danych w architekturze mikroserwisowej 26 ■ Wzorce wdrażania usług 26 ■ Wzorce obserwowalności zapewniają wgląd w zachowanie aplikacji 27 ■ Wzorce automatycznego testowania usług 28 ■ Wzorce do obsługi potrzeb przekrojowych 28 ■ Wzorce bezpieczeństwa 28
- 1.7. Poza mikroserwisami: proces i organizacja 29
 - 1.7.1. Tworzenie oprogramowania i organizacja dostarczania 29
 - 1.7.2. Proces tworzenia i dostarczania oprogramowania 30
 - 1.7.3. Ludzka strona wprowadzania mikroserwisów 31

2. Strategie dekompozycyjne 33

2.1. Czym dokładnie jest architektura mikroserwisowa? 34

2.1.1. Czym jest architektura oprogramowania i dlaczego ma ona znaczenie? 34

Definicja architektury oprogramowania 35 ■ Model perspektyw „4+1” w architekturze oprogramowania 35 ■ Dlaczego architektura jest ważna? 37

2.1.2. Przegląd stylów architektonicznych 37

Styl: architektura warstwowa 37 ■ Styl: architektura heksagonalna 38

2.1.3. Architektura mikroserwisowa jest stylem architektonicznym 40

Co to jest usługa? 41 ■ Czym są luźne powiązania? 42 ■ Rola współdzielonych bibliotek 43 ■ Rozmiar usługi jest w zasadzie nieważny 43

2.2. Definiowanie architektury mikroserwisowej aplikacji 44

2.2.1. Identyfikowanie operacji systemu 45

Tworzenie wysokopoziomowego modelu domeny 46 ■ Określanie operacji systemu 48

2.2.2. Definiowanie usług z zastosowaniem wzorca dekompozycji według zdolności biznesowych 50

Zdolności biznesowe określają, co robi organizacja 51 ■ Identyfikowanie zdolności biznesowych 51 ■ Od zdolności biznesowych do usług 52

2.2.3. Definiowanie usług z zastosowaniem dekompozycji według wzorca subdomeny 54

2.2.4. Wytyczne dotyczące dekompozycji 55

Zasada pojedynczej odpowiedzialności 56 ■ Reguła wspólnego domknięcia 56

2.2.5. Przeszkody w dekompozycji aplikacji na usługi 57

Opóźnienia sieciowe 57 ■ Synchroniczna komunikacja międzyprocesowa ogranicza dostępność 57 ■ Utrzymanie spójności danych w usługach 57 ■ Uzyskanie spójnego widoku danych 58 ■ Dekompozycję utrudniają god classes 58

2.2.6. Tworzenie API usług 60

Przypisywanie operacji systemu do usług 61 ■ Określanie API wymaganego do realizowania współpracy między usługami 62

3. Komunikacja międzyprocesowa w architekturze mikroserwisowej 65

3.1. Przegląd komunikacji międzyprocesowej w architekturze mikroserwisowej 66

3.1.1. Style interakcji 67

3.1.2. Definiowanie API w architekturze mikroserwisowej 68

- 3.1.3. Ewolucja API 69
 - Użycie semantycznego wersjonowania 70
 - Wprowadzanie drobnych, kompatybilnych wstecznie zmian 70
 - Wprowadzanie istotnych zmian 70
- 3.1.4. Formaty komunikatów 71
 - Tekstowe formaty komunikatów 71
 - Binarne formaty komunikatów 72
- 3.2. Komunikacja z użyciem wzorca synchronicznego zdalnego wywołania procedury 72
 - 3.2.1. Korzystanie z REST 73
 - Model dojrzałości REST 74
 - Specyfikowanie REST API 74
 - Wyzwania związane z pobieraniem wielu zasobów w jednym żądaniu 75
 - Wyzwania związane z mapowaniem operacji na czasowniki HTTP 75
 - Zalety i wady REST 75
 - 3.2.2. Korzystanie z gRPC 76
 - 3.2.3. Postępowanie w przypadku częściowej awarii z użyciem wzorca bezpiecznika 78
 - Tworzenie solidnych proxy RPI 79
 - Reagowanie na niedostępne usługi 79
 - 3.2.4. Użycie wykrywania usług 80
 - Wykrywanie usług 81
 - Stosowanie wzorców wykrywania usług na poziomie aplikacji 81
 - Stosowanie wzorców wykrywania usług na poziomie platformy 83
- 3.3. Komunikacja z użyciem wzorca asynchronicznego przesyłania komunikatów 85
 - 3.3.1. Spojrzenie na przesyłanie komunikatów 86
 - O komunikatach 86
 - O kanałach komunikatów 86
 - 3.3.2. Realizacja stylów interakcji za pomocą przesyłania komunikatów 87
 - Realizacja żądanie/odpowiedź i asynchroniczne żądanie/odpowiedź 87
 - Realizacja powiadomień jednokierunkowych 88
 - Realizacja publikuj/subskrybuj 89
 - Realizacja publikowanie/asynchroniczna odpowiedź 89
 - 3.3.3. Tworzenie specyfikacji dla API usługi opartej na przesyłaniu komunikatów 89
 - Dokumentowanie operacji asynchronicznych 89
 - Dokumentowanie publikowanych zdarzeń 90
 - 3.3.4. Użycie brokera komunikatów 90
 - Przesyłanie komunikatów bez brokera 91
 - Przesyłanie komunikatów oparte na brokerze 92
 - Realizacja kanałów komunikatów za pomocą brokera 93
 - Korzyści i wady przesyłania komunikatów przez brokerów 93
 - 3.3.5. Konkurujący odbiorcy i sortowanie komunikatów 94

- 3.3.6. Obsługa zduplikowanych komunikatów 95
 - Tworzenie idempotentnej obsługi komunikatów 96 ■ Śledzenie komunikatów i odrzucanie duplikatów 96
- 3.3.7. Komunikaty transakcyjne 97
 - Użycie tabeli bazy danych jako kolejki komunikatów 97 ■ Publikowanie zdarzeń z użyciem wzorca publikatora z przeszukiwaniem 98 ■ Publikowanie zdarzeń z zastosowaniem wzorca śledzenia dziennika transakcji 99
- 3.3.8. Biblioteki i frameworki do przesyłania komunikatów 100
 - Proste przesyłanie komunikatów 101 ■ Publikowanie zdarzeń domenowych 102 ■ Przesyłanie komunikatów oparte na poleceniu/odpowiedzi 102
- 3.4. Użycie asynchronicznego przesyłania komunikatów w celu poprawy dostępności 103
 - 3.4.1. Komunikacja synchroniczna zmniejsza dostępność 103
 - 3.4.2. Eliminowanie interakcji synchronicznych 105
 - Użycie asynchronicznych stylów interakcji 105 ■ Replikacja danych 105
 - Zakończenie przetwarzania po zwróceniu odpowiedzi 106
- 4. Zarządzanie transakcjami z użyciem sag 110**
 - 4.1. Zarządzanie transakcjami w architekturze mikroserwisowej 111
 - 4.1.1. Potrzeba transakcji rozproszonych w architekturze mikroserwisowej 112
 - 4.1.2. Problem z rozproszonymi transakcjami 112
 - 4.1.3. Użycie wzorca sagi do zarządzania spójnością danych 114
 - Przykładowa saga: Create Order Saga 114 ■ Do wycofania zmian sagi wykorzystują transakcje kompensujące 115
 - 4.2. Koordynowanie sag 117
 - 4.2.1. Sagi oparte na choreografii 118
 - Realizacja Create Order Saga z wykorzystaniem choreografii 118
 - Niezawodna komunikacja oparta na zdarzeniach 120 ■ Zalety i wady sag opartych na choreografii 121
 - 4.2.2. Sagi oparte na orkiestracji 121
 - Realizacja Create Order Saga z wykorzystaniem orkiestracji 122
 - Modelowanie orkiestratorów sagi jako maszyn stanów 123 ■ Orkiestracja sagi i komunikaty transakcyjne 124 ■ Zalety i wady sag opartych na orkiestracji 125
 - 4.3. Radzenie sobie z brakiem izolacji 125
 - 4.3.1. Przegląd anomalii 126
 - Utracone aktualizacje 127 ■ Brudne odczyty 127

4.3.2.	Przeciwdziałania związane z brakiem izolacji	127
	Struktura sagi	128
	▪ Blokada semantyczna	129
	▪ Aktualizacje przemienne	130
	▪ Podejście pesymistyczne	130
	▪ Ponowne odczytanie wartości	131
	▪ Rejestr kroków	131
	▪ Według korzyści	131
4.4.	Projekt usługi Order Service i sagi Create Order Saga	132
4.4.1.	Klasa OrderService	133
4.4.2.	Implementacja Create Order Saga	134
	Orkiestrator CreateOrderSaga	136
	▪ Klasa CreateOrderSagaState	137
	▪ Klasa KitchenServiceProxy	138
	▪ Framework Eventuate Tram Saga	139
4.4.3.	Klasa OrderCommandHandlers	142
4.4.4.	Klasa OrderServiceConfiguration	143
5.	Projektowanie logiki biznesowej w architekturze mikroserwisowej	146
5.1.	Wzorce organizacji logiki biznesowej	147
5.1.1.	Projektowanie logiki biznesowej z użyciem wzorca skryptu transakcji	149
5.1.2.	Projektowanie logiki biznesowej z użyciem wzorca modelu domeny	150
5.1.3.	Domain-Driven Design	151
5.2.	Projektowanie modelu domeny z użyciem wzorca agregatu DDD	152
5.2.1.	Problem z rozmytymi granicami	153
5.2.2.	Agregaty mają wyraźne granice	154
	Agregaty są granicami spójności	154
	▪ Kluczową jest identyfikacja agregatów	155
5.2.3.	Zasady agregatów	155
	Zasada 1: Odniesienie tylko do korzenia agregatu	155
	▪ Zasada 2: Odniesienia między agregatami muszą korzystać z kluczy głównych	156
	▪ Zasada 3: Jedna transakcja tworzy lub aktualizuje jeden agregat	156
5.2.4.	Ziarnistość agregatu	157
5.2.5.	Projektowanie logiki biznesowej z agregatami	158
5.3.	Publikowanie zdarzeń domenowych	159
5.3.1.	Dlaczego publikować zdarzenia związane ze zmianą stanu?	160
5.3.2.	Czym jest zdarzenie domenowe?	160
5.3.3.	Wzbogacanie zdarzeń	161
5.3.4.	Identyfikowanie zdarzeń domenowych	162

- 5.3.5. Generowanie i publikowanie zdarzeń domenowych 163
 - Generowanie zdarzeń domenowych 163 ■ Jak niezawodnie publikować zdarzenia domenowe? 165
- 5.3.6. Pobieranie zdarzeń domenowych 167
- 5.4. Logika biznesowa Kitchen Service 167
 - 5.4.1. Agregat Ticket 169
 - Struktura klasy Ticket 169 ■ Zachowanie agregatu Ticket 170 ■ Usługa KitchenService 171 ■ Klasa KitchenServiceCommandHandler 171
- 5.5. Logika biznesowa Order Service 172
 - 5.5.1. Agregat Order 174
 - Struktura agregatu Order 174 ■ Maszyna stanów agregatu Order 175
 - Metody agregatu Order 176
 - 5.5.2. Klasa OrderService 179
- 6. Rozwijanie logiki biznesowej za pomocą pozyskiwania zdarzeń 182**
 - 6.1. Wykorzystywanie wzorca pozyskiwania zdarzeń do tworzenia logiki biznesowej 183
 - 6.1.1. Problemy tradycyjnego utrwalania 184
 - Niedopasowanie między modelem obiektowym a relacyjnym 184 ■ Brak zagregowanej historii 185 ■ Wdrożenie dzienników zdarzeń jest uciążliwe i podatne na błędy 185 ■ Publikowanie zdarzeń jest powiązane z logiką biznesową 185
 - 6.1.2. Pozyskiwanie zdarzeń 185
 - Pozyskiwanie zdarzeń utrwała agregaty, korzystając ze zdarzeń 185
 - Zdarzenia reprezentują zmiany stanu 187 ■ Metody agregatów dotyczą zdarzeń 188 ■ Agregat Order wykorzystujący pozyskiwanie zdarzeń 190
 - 6.1.3. Obsługa jednoczesnych aktualizacji z użyciem optymistycznego blokowania 192
 - 6.1.4. Pozyskiwanie zdarzeń i publikowanie zdarzeń 193
 - Użycie przeszukiwania do publikowania zdarzeń 193 ■ Wykorzystanie śledzenia dziennika transakcji do niezawodnego publikowania zdarzeń 194
 - 6.1.5. Użycie migawek do poprawy wydajności 194
 - 6.1.6. Idempotentne przetwarzanie komunikatów 195
 - Idempotentne przetwarzanie komunikatów z magazynem zdarzeń używającym RDBMS 196 ■ Idempotentne przetwarzanie komunikatów z magazynem zdarzeń używającym NoSQL 196

- 6.1.7. Rozwój zdarzeń domenowych 197
 - Ewolucja schematu zdarzeń 197 ■ Zarządzanie zmianami schematu przez rzutowanie w górę 198
- 6.1.8. Zalety pozyskiwania zdarzeń 198
 - Niezawodne publikowanie zdarzeń domenowych 199 ■ Zachowywanie historii agregatów 199 ■ Zwykle pomaga uniknąć problemu niedopasowania między modelem relacyjnym a obiektowym 199 ■ Oferuje programistom maszynę czasu 199
- 6.1.9. Wady pozyskiwania zdarzeń 199
 - Inny model programowania powodujący wzrost krzywej uczenia się 200
 - Złożoność aplikacji opartej na przesyłaniu komunikatów 200 ■ Ewolucja zdarzeń może być trudna 200 ■ Usuwanie danych jest trudne 200
 - Wykonywanie zapytań do magazynu zdarzeń jest trudne 201
- 6.2. Implementowanie magazynu zdarzeń 201
 - 6.2.1. Jak działa magazyn zdarzeń Eventuate Local 202
 - Schemat bazy danych zdarzeń Eventuate Local 202 ■ Pobieranie zdarzeń przez subskrybowanie brokera zdarzeń Eventuate Local 204 ■ Przekaznik zdarzeń Eventuate Local propaguje zdarzenia z bazy danych do brokera komunikatów 204
 - 6.2.2. Środowisko klienta Eventuate dla Javy 205
 - Definiowanie agregatów za pomocą klasy ReflectiveMutableCommandProcessingAggregate 206 ■ Definiowanie poleceń agregatu 206 ■ Definiowanie zdarzeń domenowych 206
 - Tworzenie, wyszukiwanie i aktualizacja agregatów za pomocą klasy AggregateRepository 207 ■ Subskrybowanie zdarzeń domenowych 208
- 6.3. Łączne używanie sag i pozyskiwania zdarzeń 208
 - 6.3.1. Realizacja sag opartych na choreografii z wykorzystaniem pozyskiwania zdarzeń 209
 - 6.3.2. Tworzenie sagi opartej na orkiestracji 210
 - Tworzenie orkiestratora sagi podczas korzystania z magazynu zdarzeń opartego na RDMBS 210 ■ Tworzenie orkiestratora sagi podczas korzystania z magazynu zdarzeń opartego na NoSQL 211
 - 6.3.3. Tworzenie uczestnika sagi opartej na pozyskiwaniu zdarzeń 212
 - Idempotentna obsługa komunikatów poleceń 213 ■ Atomowe wysyłanie komunikatów zwrotnych 213 ■ Przykład uczestnika sagi opartej na pozyskiwaniu zdarzeń 213
 - 6.3.4. Projektowanie orkiestratorów sagi z użyciem pozyskiwania zdarzeń 216
 - Utrzymywanie orkiestratora sagi z użyciem pozyskiwania zdarzeń 216
 - Niezawodne wysyłanie komunikatów poleceń 216 ■ Dokładnie jednokrotne przetwarzanie odpowiedzi 217

- 7. Implementowanie zapytań w architekturze mikroserwisowej 219
 - 7.1. Tworzenie zapytań z użyciem wzorca kompozycji API 220
 - 7.1.1. Operacja zapytania findOrder() 220
 - 7.1.2. Wzorzec kompozycji API 221
 - 7.1.3. Implementowanie operacji zapytania findOrder() z wykorzystaniem wzorca kompozycji API 223
 - 7.1.4. Problemy projektowe związane z wzorcem kompozycji API 224
 - Kto powinien odgrywać rolę API Composer? 224
 - API Composer powinien stosować model programowania reaktywnego 226
 - 7.1.5. Wady i zalety wzorca kompozycji API 226
 - Zwiększone koszty ogólne 226
 - Ryzyko zmniejszonej dostępności 226
 - Brak spójności danych transakcyjnych 227
 - 7.2. Korzystanie z wzorca CQRS 227
 - 7.2.1. Motywacja do korzystania z CQRS 228
 - Realizacja operacji zapytania findOrderHistory() 228
 - Wymagające zapytanie dotyczące pojedynczej usługi findAvailableRestaurants() 230
 - Konieczność oddzielenia obowiązków 230
 - 7.2.2. Przegląd CQRS 231
 - CQRS oddziela polecenia od zapytań 231
 - CQRS i usługi tylko dla zapytań 232
 - 7.2.3. Zalety CQRS 234
 - Umożliwia wydajną implementację zapytań w architekturze mikroserwisowej 234
 - Umożliwia wydajną implementację zróżnicowanych zapytań 234
 - Zapewnia wykonywanie zapytań w aplikacji opartej na pozyskiwaniu zdarzeń 234
 - Poprawia separację obowiązków 234
 - 7.2.4. Wady CQRS 235
 - Bardziej złożona architektura 235
 - Konieczność radzenia sobie z opóźnieniem replikacji 235
 - 7.3. Projektowanie widoków CQRS 236
 - 7.3.1. Wybór magazynu danych widoku 236
 - Bazy danych SQL kontra NoSQL 237
 - Wspieranie operacji aktualizacji 238
 - 7.3.2. Projekt modułu dostępu do danych 238
 - Obsługa jednoczesnych aktualizacji 239
 - Idempotentne procedury obsługi zdarzeń 239
 - Umożliwienie aplikacji klienckiej korzystania z ostatecznie spójnego widoku 240
 - 7.3.3. Dodawanie i aktualizacja widoków CQRS 241
 - Budowanie widoków CQRS z użyciem zarchiwizowanych zdarzeń 241
 - Stopniowe budowanie widoków CQRS 241

- 7.4. Implementowanie widoku CQRS za pomocą AWS DynamoDB 241
 - 7.4.1. Moduł OrderHistoryEventHandlers 243
 - 7.4.2. Modelowanie danych i projektowanie zapytań za pomocą DynamoDB 243
 - Projektowanie tabeli ftgo-order-history 244
 - Zdefiniowanie indeksu dla zapytania findOrderHistory 245
 - Realizacja zapytania findOrderHistory 245
 - Stronicowanie wyników zapytania 246
 - Aktualizowanie zamówień 247
 - Wykrywanie zduplikowanych zdarzeń 247
 - 7.4.3. Klasa OrderHistoryDaoDynamoDb 248
 - Metoda addOrder() 248
 - Metoda notePickedUp() 249
 - Metoda idempotentUpdate() 249
 - Metoda findOrderHistory() 250
- 8. Wzorce zewnętrznych API 253
 - 8.1. Problemy z projektowaniem zewnętrznego API 254
 - 8.1.1. Problemy z projektowaniem API dla mobilnego klienta FTGO 255
 - Gorszy komfort użytkownika spowodowany klientem wykonującym wiele zapytań 257
 - Brak enkapsulacji wymaga od programistów frontendu, aby zmieniali swój kod krok w krok ze zmianami w backendzie 257
 - Usługi mogą wykorzystywać mechanizmy IPC nieprzyjazne dla klientów 257
 - 8.1.2. Problemy z projektowaniem API dla innych typów klientów 258
 - Problemy z projektowaniem API dla aplikacji webowych 258
 - Problemy z projektowaniem API dla aplikacji JavaScript działających w przeglądarce 258
 - Projektowanie API dla aplikacji zewnętrznych 258
 - 8.2. Wzorzec bramy API 259
 - 8.2.1. Wzorzec bramy API 259
 - Przekierowywanie żądań 260
 - Kompozycja API 260
 - Translacja protokołów 261
 - Brama API oferuje każdemu klientowi specyficzną API 262
 - Implementowanie funkcji brzegowych 262
 - Architektura bramy API 263
 - Model własności bramy API 264
 - Korzystanie z wzorca Backends for Frontends 264
 - 8.2.2. Zalety i wady bramy API 266
 - Zalety bramy API 266
 - Wady bramy API 267
 - 8.2.3. Netflix jako przykład bramy API 267
 - 8.2.4. Problemy z projektowaniem bramy API 267
 - Wydajność i skalowalność 268
 - Użycie abstrakcji programowania reaktywnego 269
 - Postępowanie w przypadku częściowej awarii 270
 - Bycie dobrym obywatelem w architekturze aplikacji 270

8.3. Implementowanie bramy API 271

8.3.1. Skorzystanie z gotowego produktu/usługi bramy API 271

Brama API AWS 271 ■ AWS Application Load Balancer 272 ■ Użycie produktu typu brama API 272

8.3.2. Opracowanie własnej bramy API 272

Użycie Netflix Zuul 273 ■ O Spring Cloud Gateway 273
■ Klasa OrderConfiguration 275 ■ Klasa OrderHandlers 276 ■ Klasa OrderService 278 ■ Klasa ApiGatewayApplication 279

8.3.3. Realizacja bramy API za pomocą GraphQL 279

Definiowanie schematu GraphQL 281 ■ Wykonywanie zapytań GraphQL 284 ■ Podłączanie schematu do danych 285 ■ Optymalizacja ładowania za pomocą grupowania i buforowania 287 ■ Integracja serwera Apollo GraphQL z Expressem 288 ■ Tworzenie klienta GraphQL 290

9. Testowanie mikroserwisów: część 1 292

9.1. Strategie testowania w architekturze mikroserwisowej 294

9.1.1. Testowanie 295

Pisanie testów automatycznych 295 ■ Testowanie z użyciem obiektów typu mock i stub 296 ■ Różne rodzaje testów 297 ■ Wykorzystanie kwadrantu testowego do kategoryzacji testów 298 ■ Wykorzystanie piramidy testów jako przewodnika w wysiłkach związanych z testowaniem 299

9.1.2. Wyzwania związane z testowaniem mikroserwisów 300

Test kontraktowy ukierunkowany na klienta 302 ■ Testowanie usług z wykorzystaniem Spring Cloud Contract 304 ■ Konsumenckie testy kontraktowe API przesyłania komunikatów 306

9.1.3. Strumień wdrażania 306

9.2. Pisanie testów jednostkowych dla usługi 307

9.2.1. Projektowanie testów jednostkowych dla encji 310

9.2.2. Pisanie testów jednostkowych dla obiektów wartości 310

9.2.3. Opracowywanie testów jednostkowych dla sag 311

9.2.4. Pisanie testów jednostkowych dla usług domenowych 313

9.2.5. Opracowywanie testów jednostkowych dla kontrolerów 314

9.2.6. Pisanie testów jednostkowych dla procedur obsługi zdarzeń i komunikatów 316

10. Testowanie mikroserwisów: część 2 319

10.1. Pisanie testów integracyjnych 320

10.1.1. Testy integracyjne trwałości 322

- 10.1.2. Testy integracyjne interakcji żądanie/odpowiedź wykorzystującej REST 323
 - Przykładowy kontrakt dla REST API 325 ■ Integracyjny test kontraktowy ukierunkowany na klienta dla Order Service 325 ■ Test integracyjny po stronie konsumenta dla OrderServiceProxy z API Gateway 326
- 10.1.3. Testy integracyjne interakcji publikuj/subskrybuj 328
 - Kontrakt dla publikowania zdarzenia OrderCreated 329 ■ Test kontraktowy ukierunkowany na konsumenta dla Order Service 329 ■ Test kontraktowy po stronie konsumenta dla Order History Service 330
- 10.1.4. Kontraktowe testy integracyjne interakcji asynchroniczne żądanie/odpowiedź 332
 - Przykładowy kontrakt dla asynchronicznego żądania/odpowiedzi 333
 - Kontraktowy test integracyjny po stronie konsumenta dla interakcji asynchroniczne żądanie/odpowiedź 334 ■ Pisanie testów kontraktowych po stronie dostawcy przeznaczonych do testowania konsumenta w zakresie asynchronicznej interakcji żądanie/odpowiedź 335
- 10.2. Tworzenie testów komponentów 336
 - 10.2.1. Definiowanie testów akceptacyjnych 337
 - 10.2.2. Pisanie testów akceptacyjnych z użyciem Gherkina 338
 - Wykonywanie specyfikacji Gherkina za pomocą Cucumber 339
 - 10.2.3. Projektowanie testów komponentów 340
 - Wewnątrzprocesowe testy komponentów 340 ■ Pozapprocesowe testowanie komponentu 341 ■ Jak tworzyć obiekty stubs dla usług w pozapprocesowych testach komponentów 341
 - 10.2.4. Pisanie testów komponentów dla Order Service z FTGO 342
 - Klasa OrderServiceComponentTestStepDefinitions 343 ■ Uruchamianie testów komponentów 345
- 10.3. Pisanie testów end-to-end 346
 - 10.3.1. Projektowanie testów end-to-end 347
 - 10.3.2. Pisanie testów end-to-end 347
 - 10.3.3. Uruchamianie testów end-to-end 348
- 11. Opracowywanie usług gotowych do produkcji 349**
 - 11.1. Opracowywanie bezpiecznych usług 350
 - 11.1.1. Przegląd kwestii bezpieczeństwa w tradycyjnej aplikacji monolitycznej 351
 - 11.1.2. Realizacja kwestii bezpieczeństwa w architekturze mikroserwisowej 354
 - Obsługa uwierzytelniania w bramie API 355 ■ Obsługa autoryzacji 357
 - Użycie JWT do przekazywania tożsamości użytkownika i roli 357 ■ Użycie OAuth 2.0 w architekturze mikroserwisowej 358

- 11.2. Projektowanie konfigurowalnych usług 361
 - 11.2.1. Użycie konfiguracji zewnętrznej opartej na udostępnianiu 363
 - 11.2.2. Użycie konfiguracji zewnętrznej opartej na pobieraniu 364
 - 11.3. Projektowanie obserwowalnych usług 366
 - 11.3.1. Użycie wzorca API kontroli działania 367
 - Realizacja punktu końcowego kontroli działania 369 ■ Wywołanie punktu końcowego kontroli działania 369
 - 11.3.2. Stosowanie wzorca agregacji dzienników 369
 - W jaki sposób usługa generuje dziennik? 370 ■ Infrastruktura agregacji dzienników 371
 - 11.3.3. Stosowanie wzorca rozproszonego śledzenia 371
 - Użycie biblioteki instrumentacji 373 ■ Serwer rozproszonego śledzenia 374
 - 11.3.4. Stosowanie wzorca metryk aplikacji 375
 - Zbieranie metryk na poziomie usługi 376 ■ Dostarczanie metryk do usługi metryk 377
 - 11.3.5. Stosowanie wzorca śledzenia wyjątków 377
 - 11.3.6. Stosowanie wzorca dzienników inspekcji 379
 - Dodawanie kodu dzienników inspekcji do logiki biznesowej 379 ■ Użycie programowania aspektowego 379 ■ Użycie pozyskiwania zdarzeń 379
 - 11.4. Tworzenie usług za pomocą wzorca szkieletów mikroserwisowych 380
 - 11.4.1. Korzystanie ze szkieletu mikroserwisowego 381
 - 11.4.2. Od szkieletu mikroserwisowego do service mesh 381
- 12. Wdrażanie mikroserwisów 384**
- 12.1. Wdrażanie usług jako pakietów specyficznych dla języka 387
 - 12.1.1. Zalety wzorca usługi jako pakietu specyficznego dla języka 389
 - Szybkie wdrażanie 390 ■ Efektywne wykorzystanie zasobów 390
 - 12.1.2. Wady wzorca usługi jako pakietu specyficznego dla języka 390
 - Brak enkapsulacji stosu technologicznego 390 ■ Brak możliwości ograniczenia zasobów zużywanych przez instancję usługi 391 ■ Brak izolacji podczas uruchamiania wielu instancji usługi na tej samej maszynie 391
 - Automatyczne określenie miejsca umieszczenia instancji usługi jest trudne 391
 - 12.2. Wdrażanie usług z użyciem wzorca usługi jako maszyny wirtualnej 391
 - 12.2.1. Zalety wdrażania usług jako maszyn wirtualnych 393

- Obraz maszyny wirtualnej hermetyzuje stos technologiczny 393 ■ Instancje usług są izolowane 393 ■ Wykorzystanie wyrafinowanej infrastruktury chmurowej 393
- 12.2.2. Wady wdrażania usług jako maszyn wirtualnych 394
 - Mniej wydajne wykorzystanie zasobów 394 ■ Wolne wdrożenia 394
 - Narzut związany z administracją systemem 394
- 12.3. Wdrażanie usług za pomocą wzorca usługi jako kontenera 394
 - 12.3.1. Wdrażanie usług za pomocą Dockera 396
 - Budowanie obrazu Dockera 397 ■ Wysyłanie obrazu Dockera do rejestru 398 ■ Uruchamianie kontenera Dockera 398
 - 12.3.2. Zalety wdrażania usług jako kontenerów 399
 - 12.3.3. Wady wdrażania usług jako kontenerów 400
- 12.4. Wdrażanie aplikacji FTGO za pomocą Kubernetesa 400
 - 12.4.1. Kubernetes 400
 - Architektura Kubernetesa 402 ■ Kluczowe pojęcia Kubernetesa 403
 - 12.4.2. Wdrażanie Restaurant Service w Kubernetesie 404
 - Tworzenie service w Kubernetesie 406 ■ Wdrażanie bramy API 407
 - 12.4.3. Wdrożenia bez przestojów 408
 - 12.4.4. Użycie service mesh w celu oddzielenia wdrożenia od wydania 409
 - Service mesh Istio 410 ■ Wdrażanie usługi za pomocą Istio 412
 - Tworzenie reguł routingu do wersji v1 413 ■ Wdrażanie wersji 2 Consumer Service 415 ■ Przekierowywanie części testowego ruchu do wersji 2 415
 - Kierowanie produkcyjnego ruchu do wersji 2 416
- 12.5. Wdrażanie usług za pomocą wzorca wdrożenia bezserwerowego 417
 - 12.5.1. Bezserwerowe wdrażanie za pomocą AWS Lambda 417
 - 12.5.2. Tworzenie funkcji lambda 418
 - 12.5.3. Wywoływanie funkcji lambda 419
 - Obsługa żądań HTTP 419 ■ Obsługa zdarzeń generowanych przez usługi AWS 419 ■ Definiowanie zaplanowanych funkcji lambda 419
 - Wywoływanie funkcji lambda za pomocą żądania usługi sieciowej 419
 - 12.5.4. Zalety użycia funkcji lambda 420
 - 12.5.5. Wady użycia funkcji lambda 420
- 12.6. Wdrażanie usługi RESTful z użyciem AWS Lambda i AWS Gateway 421
 - 12.6.1. Projekt AWS Lambda wersji Restaurant Service 421
 - Klasa FindRestaurantRequestHandler 423 ■ Wstrzykiwanie zależności z użyciem klasy AbstractAutowiringHttpRequestHandler 424 ■ Klasa AbstractHandler 425

- 12.6.2. Pakowanie usługi jako pliku ZIP 426
- 12.6.3. Wdrażanie funkcji lambda z użyciem frameworka Serverless 426

13. Refaktoryzacja do mikroserwisów 429

13.1. Refaktoryzacja do mikroserwisów 430

- 13.1.1. Dlaczego warto refaktoryzować monolit? 431
- 13.1.2. Duszenie monolitu 431

Szybkie i częste prezentowanie wartości dodanej 433 ■ Minimalizacja zmian w monolicie 434 ■ Techniczne aspekty infrastruktury wdrożeniowej: nie potrzebujemy wszystkiego od razu 434

13.2. Strategie refaktoryzacji monolitu do mikroserwisów 434

13.2.1. Realizacja nowych funkcjonalności jako usług 435

Integracja nowej usługi z monolitem 435 ■ Kiedy zaimplementować nową funkcjonalność jako usługę 435

13.2.2. Oddzielanie warstwy prezentacji od backendu 437

13.2.3. Wyodrębnianie zdolności biznesowych do usług 438

Podział modelu domeny 440 ■ Refaktoryzacja bazy danych 441
■ Replikowanie danych w celu uniknięcia rozległych zmian 442 ■ Jakie usługi wyodrębnić i kiedy 443

13.3. Projektowanie, w jaki sposób usługa i monolit będą współpracować 444

13.3.1. Projektowanie kodu integrującego 445

Projektowanie API kodu integrującego 445 ■ Wybór stylu interakcji i mechanizmu IPC 446 ■ Projektowanie Anti-Corruption Layer 448 ■ Jak monolit publikuje i subskrybuje zdarzenia domenowe 449

13.3.2. Utrzymywanie spójności danych w usłudze i monolicie 450

Wyzwanie związane ze zmianą monolitu, aby wspierał transakcje kompensujące 451 ■ Sagi nie zawsze wymagają od monolitu wspierania transakcji kompensujących 452 ■ Określanie kolejności wyodrębniania usług w celu uniknięcia konieczności implementacji transakcji kompensujących w monolicie 453

13.3.3. Obsługa uwierzytelniania i autoryzacji 455

LoginHandler monolitu ustawia plik cookie USERINFO 456 ■ Brama API mapuje plik cookie USERINFO na nagłówek Authorization 456

13.4. Wdrażanie nowej funkcjonalności jako usługi: obsługa niezrealizowanych zamówień 457

13.4.1. Projekt Delayed Delivery Service 458

- 13.4.2. Projektowanie kodu integrującego dla Delayed Delivery Service 459
 - Pobieranie informacji kontaktowych klienta z wykorzystaniem CustomerContactInfoRepository 460
 - Publikowanie i pobieranie zdarzeń domenowych Order i Restaurant 460
- 13.5. Rozbijanie monolitu: wyodrębnianie zarządzania dostawami 461
 - 13.5.1. Przegląd istniejącej funkcjonalności związanej z zarządzaniem dostawami 461
 - 13.5.2. Przegląd Delivery Service 463
 - 13.5.3. Projekt modelu domeny Delivery Service 464
 - Określenie encji i ich pól, które są częścią zarządzania dostawami 464
 - Decydowanie, które dane przenieść do Delivery Service 465
 - Projekt logiki domeny Delivery Service 466
 - 13.5.4. Projekt kodu integrującego Delivery Service 467
 - Projekt API Delivery Service 467
 - W jaki sposób Delivery Service będzie uzyskiwał dostęp do danych monolitu FTGO? 468
 - W jaki sposób monolit FTGO będzie uzyskiwał dostęp do danych Delivery Service? 468
 - 13.5.5. Dostosowanie monolitu FTGO do współpracy z Delivery Service 468
 - Definiowanie interfejsu DeliveryService 468
 - Refaktoryzacja monolitu w celu wywołania interfejsu DeliveryService 469
 - Implementacja interfejsu DeliveryService 470
 - Przelączanie funkcjonalności 470