# Seria prezentacji “Od modelu do implementacji na przykładzie the80by20”

## Utworzenie modelu systemu

na miro, z użyciem event storming (przykład: https://miro.com/app/board/uXjVOkhNSOs=/ )

- eksploracja domeny - es big picture, same zdarzenia, problemy (hotspoty), ułożenie zdarzeń w proces

- zapisywanie języka domeny, który będzie obowiązywał w komunikacji, na modelu i w kodzie ((ubiquitous language) wszędobylski język, ale uwzgledniający bounded contexty)

- utworzenie procesu / procesów tak jak powinny być (bo na es big picture pojawiają się problemy, pokazujemy wtedy jaki proces jest, a nie jeszcze jaki chcemy żeby był) - w tym celu użycie es process level
    - dodajemy readmodele (readmodelem może stać się zdarzenie interfejsowe z es big picture), rozkazy, inwarianty, próba rozwiązania problemów lepszym procesem, próba znalezienie osobnych procesów, próba uchwycenia pod-domen
    - użyciu picture that explains everything

- próba podziału systemu na bounded contexty przez: domena -> pod-domeny -> bounded contexty -> moduły
    - szukać po single source of truth (grupować zdarzenia zmieniające jakąś informację biznesową do jednego kontesktu)
    - heurystyka różnego nazywania jednego obiektu ze świata rzeczywistego: (np, książka w kontekście magazynu to towar, a w kontekście sprzedaży to bestseller, albo ta sama nazwa ale inne atrybuty - to może sugerować przynależność książki do osobnych kontestków (osobne obiekty, osobne table, wspólny id - uniknięcie splątania i couplingu)
    - kategorie pod-domen
        - core (nasz główny proces biznesowy , przewaga rynkowa)
        - supportive
        - generic - można kupić (np system płatności)
    - jakie informacje wymieniają ze sobą konteksty - starać się o jak najluźniejszy messaging
        - - najluźniej np przez styl event driven architecture - moduł A publishuje event na broker / szynę / kolejkę eventów moduł B subskrybuje się

- na podstawie w/w model modułów i messagingu między nimi
    - messaging - asynchroniczny / synchroniczny
    - komunikaty: rozkaz, query, event

- przekształcenie modelu es process level na model  es design level + wyznaczanie granic agregatów
    - użyciu picture that explains everything żeby zrobić es big picture
    - 4 zasady tworzenia agregatów
    - anemiczna encja z tym samym id co agregat, by agregat miał w sobie tylko informacje na cele inwariantów
    - model agregatu zawiera też informacje o serwis domenowym  (na cele orkiestracji agregatów), read model aktualizowany zdarzeniem agregata - skutek spersystowania zmiany stanu

- wozrców z ddd strategicznego (bounded contexty, ubiquitous language)
    - ddd taktyczne (building blocks taktycznego ddd)
    - utworzenie modelu es big picture, process level,
    - utworzenie modelu es design level, reguły wyznaczenia granic agregatów, modelowanie przy użyciu picture that explains everything
    - domena -> poddomeny -> bounded contexty -> moduły w systemie z różnymi architekturami (bo mają różne drivery)

# Implementacja, na podstawie modelu

- w głowie:
    - grasp, solid
    - niski coupling wysoka kohezja, hermetyzacja obiektów, obiekty komunikują się poprzez wysyłanie sobie rozkazów,
    - 4 zasady budowy agregatów
    - zachowanie w kodzie ubiquitous language
    - różne abstrakcje tego samego obiektu z rzeczywistości (nie robić uber tabeli uber obiektów - rozdzielać wg kontaktu, wspólne id snoflwkae)
- stabilne api zabezpieczone testami. bdd pisane na modłę test first,

- implementacja modułów na podstawie modelu bounded context i messagingu między nimi
- różne moduły mają różne architektury;  driverami jakie style architektoniczne w danym module użyć może być np.:
    - złożoność logiki, którą widać na modelu,
    - to czy model reprezentuje core systemu (bogaty model domenowy, **the80by20.core**), reprezentuje system raportowy reagujący na zdarzenia z core-a (analityka danych **the80by20.reports**), reprezentuje master data (crud słowników z soft delete i audytem danych **the80by20.masterdata**), czy reprezentuje etl, czy cross cutting - np security i zarządzanie userami  **the80by20.security**

// ramki na miro https://miro.com/app/board/uXjVOkhNSOs=/
- implementacja walking skeleton poszczególnych modułów z uwzględnieniem wzorców na potrzeby każdego z nich
    - **architektura warstwowa** warstwy: web.api, app, infrastructure, common (biblioteczka, która retencjonują all projekty), playground - testy, wiki (docs, adry, instrukcje - np uruchomienie / komendy do migracji, diagramy z modelami) -  wszędzie się przyda

    - **clean architecture** (porty (warstwy z logiką decydują o ich kształcie - jaki interfejs), adaptery (warstwy z infrastrukturą implementują)), odwrócenie zależności przepływ wykonania podczas działania aplikacji jest odwrotny do zależności między warstwami, kontener ioc w bootstrapper - wszędzie
    - **generyczne repo** - w modułach o logice typu CRUD (moduł master-data)
    - **cqrs**
        - 1 poziomu - osobno command i query, ale ta sama tabela  / store dla zapisu (**the80by20.security**)
        - 2 poziomu osobny command, a warstwa query jako reaktywny subscriber nasłuchujący na zdarzenia zmiany stanu -  osobna table / storem (**the80by20.core**)
    - **bogaty model domenowy** (zachowania w encjach, a nie serwajs + anemiczna encja)
        - dodatkowa warstwa domain
        - wzorce taktycznego ddd: agregat, encja, encja-agregatu, value object, serwis domenowy, policy, factory, readmodel (on raczej w warstwie app), custom-exceptiony (aplikacyjne, domenowe)
        - persystencja agregatu
        - patrzenie na logikę na poziomach: application, domain (decision support, policy, operations capability) i odpowiednie zgrupowania wzorców taktycznego ddd i cqrs do każdego z nich (można katalogami podzielić)
    - **event driven messaging** między modułami
        - np moduł A publikuje event na message brokera na który subskrybuje się moduł B (the80by20.reports)
    - **process manager** - asynchroniczny stanowy message driven - np process manager płatności w the 80by20\
    - **saga** - “transakcja” między kilkoma zmianami kilku agregatów z mechanizmami kompensacji
    - **skoncentrowanie na raporty (the 80by20.reports)**
        - hurtownia danych
        - znormalizowane struktury do odczytu
        - asnotracking w ef lub bezpośrednie sql

- implementacja agregatów, rozkazów, zdarzeń, read modeli (przełożenie picture that explains everything + CQRS, który mamy na modelu ES na kod)
    - Kroki implementacji modelu na https://miro.com/app/board/uXjVOkhNSOs=/
    - w tym testy bdd (stan-zmianastanu-then)

- implementacja cross cuttings
    - audyt
    - soft delete
    - logowanie + middleware exceptionów
    - sign in, sign up + mechanik persystencji userow ich rol, jwt token, reposektowanie rol na odczycie (asp.net identy, reuiemtns)
    - dal
        - migracje
        - dbcontext
    - ioc

        - dbcontext scoped
        - dekoracja handlerów unit of work transaction
        - rejestracja generyczna commands / queries handlers
    - swagger, postman, vscode http
    - testy z in-memory database, webhostbuidler,  - czy dobra persystencja agregatu i read modela - happy path
    - dobre kontrolery - routing, rest-style

## Modeling whirlpool - cykl: model, implementacja, repeat

## CI / CD

- ci/cd na ghub actions(w tym coś w stylu gitflow)
- dockeryzacja serwisów
- hosting na chmurze i usługi chmurowe

## tech stack

- .net6, ef core, asp.net core, rabbitmq/ lub cos azurowego do messegingu, sql server, testy, mediatr, różne pakiety
- angular, typescript, rxjs

- gh actions
- azure, messaging tam i baza
- docker, kubernetes

- https://miro.com/app/board/uXjVOkhNSOs=/
- https://github.com/SimonConrad/the80by20