Wybór pomiędzy programowaniem obiektowym (OOP) a programowaniem funkcjonalnym (FP) może mieć znaczący wpływ na biznes SaaS, szczególnie w zakresie rozwoju produktu, skalowalności, łatwości utrzymania i ogólnej opłacalności. Chociaż programowanie modułowe nie jest uważane za paradygmat na tym samym poziomie co OOP lub FP, pozostaje istotnym aspektem organizacji i zrozumienia kodu w projektach oprogramowania. Wyjaśnię te trzy elementy i ich wpływ na długoterminowy biznes SaaS.

Programowanie obiektowe a programowanie funkcjonalne

Ponieważ warstwa OOP i FP funkcjonuje na tym samym poziomie, najłatwiej będzie zrozumieć te koncepcje poprzez ich porównanie.

Co to jest programowanie funkcyjne?

Jako jeden z najstarszych paradygmatów programowania, programowanie funkcyjne buduje systemy, koncentrując się na ich funkcjach i wykorzystując je do manipulowania danymi bez zmiany samych danych. Wyobraź to sobie jako rurociąg, w którym dane przepływają przez różne kroki (funkcje) bez ryzyka przypadkowych zmian danych po drodze. Jest to przydatne do budowania systemów, które są przewidywalne i łatwe do testowania.

Obliczenia w FP opierają się na ocenie funkcji matematycznych przy użyciu czystych funkcji (brak efektów ubocznych), niezmienności (dane nie mogą być modyfikowane) i funkcji wyższego rzędu (funkcje, które akceptują lub zwracają inne funkcje). W FP funkcje są obywatelami pierwszej klasy (ponieważ są podstawą wszystkich operacji), co oznacza, że mogą być przypisywane do zmiennych, przekazywane jako argumenty lub zwracane przez inne funkcje. Funkcja jest przewidywalna: biorąc pod uwagę te same dane wejściowe, zawsze zwróci te same dane wyjściowe.

Przykładowe funkcje mogą służyć takim celom jak:

- sumowanie/odejmowanie i wszystkie inne operacje matematyczne wykonywane na przekazanych argumentach
- zapisywanie zmiennej na dysku lub w bazie danych
- przekazanie jako argument do innej funkcji i/lub zwrócenie danych z innej

funkcji

.

W tym artykule zajmiemy się tylko kilkoma szczegółami technicznymi. Można jednak śmiało powiedzieć, że programowanie funkcyjne świetnie nadaje się do tworzenia oprogramowania po stronie serwera, manipulowania danymi lub indeksowania stron internetowych.

Co to jest programowanie obiektowe?

Programowanie obiektowe (OOP) wykorzystuje kod do osiągnięcia pożądanego rezultatu krok po kroku. OOP organizuje oprogramowanie wokół rzeczywistych „obiektów”, z których każdy ma unikalne atrybuty (dane) i działania (metody), które definiują jego zachowanie. Pomyśl o tym jak o tworzeniu planu produktu ze szczegółami takimi jak kolor, kształt i akcje, które może wykonać. Takie podejście pomaga budować oprogramowanie, które jest modułowe i wielokrotnego użytku.

Projektowanie oprogramowania w OOP jest zorganizowane wokół danych lub obiektów, a nie funkcji i logiki. Zasady OOP obejmują hermetyzację (ukrywanie danych wewnątrz obiektów), dziedziczenie (współdzielenie atrybutów i metod z podobiektami) oraz polimorfizm (pozwalający na traktowanie różnych obiektów jako instancji tego samego typu).

Aby lepiej opisać działanie obiektów, weźmy „telefon” jako przykład obiektu:

- Obiekt: telefon
- Funkcje (metody w OOP): dzwonienie, wysyłanie wiadomości tekstowych, korzystanie z aplikacji
- Zmienne (właściwości w OOP): kolor, marka, model

Zasadniczo OOP jest idealny dla systemów wymagających wielu instancji tego samego obiektu, które zachowują się inaczej w oparciu o unikalne parametry i relacje.

Programowanie modułowe – stary paradygmat

czy nowoczesne rozwiązanie?

Programowanie modułowe jest alternatywą nie należącą stricte do żadnej z wyżej wymienionych metod (choć wielu twierdzi, że jest to jedna z metod programowania obiektowego). Metoda ta jest obecna w tworzeniu oprogramowania od lat 60. ubiegłego wieku. XX wieku i wciąż uważana jest za jedną z najwygodniejszych metod programowania. Programowanie modułowe jest ściśle powiązane z programowaniem obiektowym i funkcjonalnym. Wszystkie one mają ten sam cel – podzielenie budowy dużych systemów oprogramowania na mniejsze części.

W programowaniu modułowym funkcjonalność danego oprogramowania zostaje podzielona na możliwe do zarządzania, niezależne części. Innymi słowy, deweloper zamyka wszystkie elementy jednej funkcjonalności w jednym module. Niezależne moduły można łączyć ze sobą i układać w stosy, tworząc całą aplikację. Możesz być bardziej zaznajomiony z nazwą „biblioteka”, ponieważ czasami biblioteka zaimportowana do systemu jest (na poziomie programowania lub na poziomie koncepcji) pojedynczym modułem. Przyjrzyjmy się zatem bliżej czym są moduły.

Czym są „moduły” w programowaniu?

Moduły hermetyzują określoną funkcjonalność, wykonując odrębne operacje, które po połączeniu umożliwiają kompleksowe tworzenie oprogramowania. Takie podejście jest korzystne przy tworzeniu wielu aplikacji, głównie dlatego, że pozwala na wstępną produkcję określonych funkcji i ponowne ich wykorzystanie do tworzenia nowego oprogramowania. Jak dokładnie? Weźmy przykład notatnika. Na przykład w zwykłym notatniku w linie, wewnętrzne strony są standardowe, niezależnie od projektu okładki. Jedynym elementem, który się zmienia, jest okładka. Drukarnia mogłaby z łatwością mieć jeden zapisany plik o nazwie Lined Interior i nie musiałaby tworzyć nowego wnętrza za każdym razem, gdy chce wydać nowy notatnik w linie. Co za oszczędność najważniejszych zasobów – czasu i pieniędzy! Jest to uproszczona reprezentacja tego, jak działa ponowne wykorzystanie wcześniej napisanego kodu i jest bezpośrednią implementacją metody DRY (Do Not Repeat Yourself), która jest podstawową zasadą dla każdego dobrego programisty.

Jak moduły mogą pomóc w rozwoju oprogramowania?

Readability

Dzięki programowaniu modułowemu kod jest łatwy do odczytania. Warto jednak zauważyć, że moduły mogą być długie i złożone, z wieloma argumentami i zmiennymi. Dlatego też potrzeba doświadczonego programisty, aby tworzyć proste i przejrzyste moduły. Niemniej jednak, jeśli zostaną wykonane poprawnie, moduły powinny być łatwiejsze do odczytania niż kody monolityczne.

Łatwe testowanie i szybsze poprawki

Każdy kod musi przejść testy przed wydaniem oprogramowania. W końcu produkt końcowy musi działać. Programowanie modułowe umożliwia testowanie oddzielnych części całego oprogramowania. Oszczędza to czas (ponownie) i ostatecznie ułatwia przeprowadzanie testów. Pozwala również skupić się na bardziej złożonych funkcjach, które wymagają większej uwagi podczas fazy testowania. Ponadto, po zidentyfikowaniu wadliwego modułu, łatwo jest przepisać lub wymienić tylko ten jeden wadliwy moduł, a nie całą aplikację.

Reużywalność

Ta funkcja programowania modułowego została już pokrótce wspomniana. Ma ona jednak więcej aspektów, o których warto wspomnieć. Po pierwsze, dzięki modułom, które mogą być pobierane za pomocą interfejsów lub bibliotek, maksymalny rozmiar tworzonej aplikacji lub oprogramowania jest mniejszy niż byłby bez użycia modułów. Po drugie, moduły mogą być wykorzystywane zarówno w ramach jednego oprogramowania, jak i przy tworzeniu innych projektów.

Niskie ryzyko aktualizacji

Możesz zastanawiać się nad problemami, które mogą pojawić się wraz z aktualizacjami. Co jeśli wprowadzisz błędną zmianę w module używanym w wielu miejscach lub nawet aplikacjach? Może to zepsuć kod i spowodować większe spustoszenie niż kody monolityczne. Jest to uzasadniona obawa. Jednak moduł programistyczny to coś więcej niż tylko jego wypełnienie. Każdy projekt programowania modułowego ma zdefiniowaną warstwę API chroniącą przed wprowadzaniem zmian wewnątrz bibliotek. Dlatego też, bez zmiany API, istnieje niskie ryzyko przypadkowego zepsucia kodu.

Zarządzalność

Jest kilka aspektów tego elementu. Jednym z nich jest wspomniana wcześniej

testowalność. Drugim jest zastępowalność – co oznacza, że jeśli moduł jest kompatybilny, można go zastąpić innym modułem lub jego lepszą wersją bez wpływu na całą aplikację. Trzecim aspektem jest łatwiejsze kodowanie – programista może skupić się na pracy tylko nad konkretnym modułem, nie biorąc pod uwagę całej aplikacji przez cały czas – musi tylko wziąć pod uwagę wejścia i wyjścia danego modułu.

Czy są jakieś wady programowania modułowego?

Każda metoda ma wady i nie byłoby obiektywne, gdybyśmy je pominęli. Dlatego wyjaśnijmy pokrótce możliwe przeszkody w tworzeniu oprogramowania modularnego:

Wymagane są odpowiednie umiejętności

Programowanie modułów wymaga wykwalifikowanych specjalistów, co sprawia, że znalezienie wykwalifikowanych programistów lub firm, które skutecznie wdrażają struktury modułowe, jest zarówno czasochłonne, jak i potencjalnie kosztowne. Z drugiej strony, wykwalifikowany programista będzie znacznie bardziej efektywny w przypadku korzystania z programowania modułowego dla większych systemów.

Potrzebny czas na testowanie

O ile programowanie modułów zwykle oszczędza czas, o tyle ich testowanie może być czasochłonne. Dzieje się tak dlatego, że trzeba przetestować wiele instancji tego samego obiektu z różnymi parametrami, więc możliwych jest znacznie więcej różnych mutacji, które należy przetestować.

Kontrola złożoności

Złożoność wzrasta, jeśli istnieje zbyt wiele modułów lub jeśli moduły są zbyt sparametryzowane – więc musi istnieć równowaga między tymi dwoma. Chociaż można to skalować w przypadku większych aplikacji, stosując podejście drzewiaste lub „wyciszając” moduły do określonych zespołów i ograniczając komunikację między nimi do interfejsów API.

Overhead dla komunikacji

TTutaj może istnieć narzut na wydajność, ponieważ dodatkowa pamięć i czas procesora muszą być wykorzystane do komunikacji między modułami. Podczas gdy

niektóre języki pozwalają na kompilację wielu modułów w jedną instancję, kompilatory wciąż nie są doskonałe, a inne języki – zwłaszcza języki skryptowe – często nie są kompilowane. Nawet jeśli taki narzut będzie widoczny tylko dla ciężkich skryptów, takich jak gry.

Verdict dla programowania modułowego

Jak łatwo zauważyć, jest to oczywiste: gdy tylko jest to możliwe, jeśli umiejętności programisty na to pozwalają, powinien on używać programowania modułowego. Nawet jeśli programowanie modułowe nie jest metodą idealną, to wciąż jest to powszechne podejście wśród wielu deweloperów. Wszelkie wady można złagodzić, jeśli odpowiednio nauczy się sztuki programowania modułowego. Jego długoterminowy wpływ jest korzystny zarówno dla programistów, jak i wydajności biznesowej. Jedyną rzeczą, o której należy pamiętać, jest – jak zawsze – nie przesadzanie, ale znalezienie równowagi między złożonością modułów a ich ilością, a jeśli łatwość zarządzania spada – organizowanie ich w silosach.

Wpływ programowania obiektowego i funkcyjnego na biznes

Wybór między programowaniem obiektowym (OOP) a programowaniem funkcjonalnym (FP) może mieć znaczący wpływ na działalność SaaS, zwłaszcza w zakresie rozwoju produktu, skalowalności, łatwości konserwacji i ogólnej opłacalności.

Szybkość rozwoju i produktywność zespołu

Projektowanie obiektowe jest często bardziej intuicyjne dla zespołów zaznajomionych z tworzeniem złożonych, stanowych aplikacji, takich jak wiele rozwiązań SaaS. Umożliwia hermetyzację i organizację wokół „rzeczywistych” bytów (takich jak użytkownicy, konta lub projekty), co może uprościć mapowanie logiki biznesowej. Dla zespołów dobrze zorientowanych w OOP może to oznaczać szybszy początkowy rozwój. Z drugiej strony, programowanie funkcjonalne kładzie nacisk na niezmienność i czyste funkcje, co może pomóc zredukować liczbę błędów i sprawić, że kod będzie bardziej przewidywalny i łatwiejszy do przetestowania. Jeśli zespół programistów ma doświadczenie z językami FP (takimi jak Haskell, Elixir lub Scala), może faktycznie szybciej tworzyć bardziej niezawodny kod, ale krzywa uczenia się może być stroma dla nowicjuszy, wpływając na szybkość wdrażania.

Skalowalność i wydajność

Aplikacje SaaS opracowane w językach OOP często wykorzystują zmienne stany, co może prowadzić do wąskich gardeł wydajności w miarę skalowania liczby użytkowników, zwłaszcza jeśli architektura nie jest starannie zaprojektowana. Zarządzanie stanami może stać się wyzwaniem wraz z rozwojem aplikacji, a czasami śledzenie stanów może prowadzić do dużych plików dziennika historii. W przypadku programowania funkcjonalnego, ponieważ kładzie ono nacisk na niezmienność i funkcje bezstanowe, skalowanie może być prostsze. Na przykład architektury bezserwerowe, mikrousługi lub systemy sterowane zdarzeniami, powszechne w SaaS, dobrze pasują do zasad FP. FP może lepiej wspierać współbieżność i równoległość, pomagając poprawić wydajność wraz ze wzrostem zapotrzebowania użytkowników.

Łatwość utrzymania i czytelność kodu

Chociaż OOP może tworzyć czytelne i intuicyjne struktury do reprezentowania złożonych bytów, duże bazy kodu oparte na OOP mogą z czasem stać się trudne w utrzymaniu ze względu na ścisłe sprzężenie, głębokie hierarchie dziedziczenia lub złożone interakcje między klasami. Może to utrudniać wprowadzanie nowych funkcji lub refaktoryzację kodu, wpływając na długoterminową łatwość utrzymania. Z drugiej strony kod funkcjonalny, skoncentrowany na czystych funkcjach, często minimalizuje zależności między komponentami, ułatwiając izolację i refaktoryzację różnych części kodu. Jednak kod może być mniej czytelny dla osób niezaznajomionych z paradygmatami FP, szczególnie w przypadku intensywnego korzystania z rekurencji lub kompozycji funkcji. Może to utrudnić konserwację, jeśli zespół nie ma doświadczenia w programowaniu funkcjonalnym.

Testowanie i debugowanie

Kod zorientowany obiektowo często wiąże się z bardziej zmiennym stanem i efektami ubocznymi, co utrudnia testowanie, ponieważ funkcje zależą od określonych stanów obiektów. Strategie testowania, takie jak mocking i wstrzykiwanie zależności, mogą pomóc, ale zwiększają złożoność. Musisz przewidzieć wiele różnych kombinacji wielu różnych parametrów, aby pokryć wszystko zgodnie z wymaganiami. Poleganie na czystych funkcjach w FP’gdzie wyjścia zależą tylko od wejść, a nie od stanu zewnętrznego’może sprawić, że automatyczne testowanie będzie prostsze i bardziej niezawodne. FP redukuje efekty

uboczne, co ułatwia debugowanie, ponieważ w kodzie jest mniej „ruchomych części”. W przypadku aplikacji SaaS, w których niezawodność jest kluczowa, może to oznaczać mniej błędów docierających do produkcji.

Elastyczność w dodawaniu nowych funkcji

OOP jest ogólnie elastyczny w budowaniu nowych funkcji, jeśli aplikacja jest dobrze zaprojektowana, ponieważ można rozszerzać klasy lub tworzyć nowe do reprezentowania nowych bytów. Jednak modyfikowanie lub rozszerzanie głęboko zagnieżdżonych lub złożonych hierarchii może wprowadzać błędy regresji. Dodawanie funkcji w FP może wiązać się z tworzeniem nowych funkcji lub komponowaniem istniejących. Chociaż może to stworzyć wysoce modułowy i komponowalny kod, złożone wymagania biznesowe mogą być czasami trudniejsze do modelowania wyłącznie w FP, w zależności od języka i tego, jak ściśle egzekwuje on zasady FP.

>

Wpływ na koszty (infrastruktura i talenty)

- **Koszty infrastruktury**: Języki FP często dobrze sprawdzają się w środowiskach rozproszonych, opartych na chmurze i bezserwerowych, które są powszechne w SaaS, potencjalnie zmniejszając koszty infrastruktury dzięki zoptymalizowanemu wykorzystaniu zasobów. Języki OOP, zwłaszcza te, które obsługują ciężką wielowątkowość (np. Java, C#), mogą również dobrze się skalować, ale mogą wymagać więcej zasobów, jeśli zarządzanie stanem stanie się złożone. Chociaż można argumentować, że koszt infrastruktury jest bardziej zależny od frameworka używanego dla konkretnego języka.
- **Koszty talentu**: OOP jest bardziej powszechny, więc znalezienie programistów znających języki OOP jest generalnie łatwiejsze i potencjalnie mniej kosztowne. Eksperci FP, zwłaszcza ci znający języki takie jak Haskell czy Erlang, są mniej dostępni, co potencjalnie zwiększa koszty zatrudnienia. Można jednak argumentować, że jest to bardziej zależne od samego języka.

Dostosowanie do wymagań SaaS (mikrousługi, architektury sterowane zdarzeniami)

Aplikacje SaaS ze złożonymi, specyficznymi dla biznesu obiektami mogą korzystać z OOP, zwłaszcza jeśli architektura jest zbudowana wokół mikrousług. Każda usługa

może reprezentować odrębne jednostki, a projektowanie obiektowe mapuje je bezpośrednio na te jednostki. Z drugiej strony programowanie funkcjonalne może być szczególnie wydajne w mikrousługach lub architekturach sterowanych zdarzeniami, gdzie każda usługa przetwarza przepływy danych w bezstanowy, odizolowany sposób. Nacisk na niezmienność i bezstanowość FP'dobrze współgra z tymi wzorcami, umożliwiając tworzenie odpornych i niezależnie wdrażanych usług, które można łatwo aktualizować lub skalować.

Podsumowanie decyzji: Programowanie obiektowe vs. funkcjonalne w SaaS

Aspect	**Functional Programming (FP)**	**Object-Oriented Programming (OOP)**
Obsługa danych	Niezmienne dane i czyste funkcje	Obiekty z hermetyzowanymi danymi i stanem
Focus	Funkcje i transformacje danych	Modelowanie rzeczywistych bytów jako obiektów
Reusability	Funkcje są wielokrotnego użytku i komponowalne	Klasy i obiekty są wielokrotnego użytku i modularne
Łatwość testowania	Często łatwiejsze dzięki bezpaństwowym i przewidywalnym funkcjom	Testowanie zależy od dobrze zdefiniowanych interfejsów klas i hermetyzacji

Podsumowując, programowanie obiektowe jest bardziej korzystne, gdy:

- Wymagania biznesowe obejmują złożone, wzajemnie powiązane struktury danych.
- Zespół programistów ma duże doświadczenie w OOP.
- Kładzie się nacisk na szybkie wdrażanie i proste skalowanie zespołu.

Z drugiej strony, programowanie funkcjonalne może oferować przewagę, gdy:

- Oczekuje się, że aplikacja będzie skalowana poziomo z wysoką współbieżnością.
- Niezawodność i prostota w testowaniu i utrzymaniu są wysokimi priorytetami.

- Zespół posiada doświadczenie w zakresie FP, a w razie potrzeby jest miejsce na krzywą uczenia się.

W ostatecznym rozrachunku, wiele nowoczesnych projektów SaaS przyjmuje **podejście hybrydowe**: wykorzystując OOP dla obszarów wymagających ustrukturyzowanej reprezentacji danych i FP dla usług bezstanowych lub komponentów, w których współbieżność i odporność na błędy są krytyczne. Taka równowaga może często uchwycić mocne strony obu paradygmatów i skutecznie spełnić wymagania SaaS. Możesz być w stanie zidentyfikować, które elementy są OOP, a które FP na opisie technicznym twojej aplikacji, ale prawdopodobnie najlepiej jest pozostawić decyzję programistom, jeśli nie jesteś architektem aplikacji. To, co powinno być ważniejsze, jeśli chcesz uruchomić SaaS, to stworzenie doskonałego modelu lean canvas lub business canvas, który bardzo pomoże programistom w zrozumieniu celu aplikacji i strategii biznesowej, co ostatecznie doprowadzi do stworzenia prawdziwego rozwiązania na zamówienie.

Sailing Byte od lat korzysta z modułów

.

Dzięki niezliczonym zaletom programowania modułowego i naszemu bogatemu doświadczeniu na rynku tworzenia oprogramowania, możemy śmiało powiedzieć, że Sailing Byte jest ekspertem w programowaniu modułowym. Projektujemy moduły z wyprzedzeniem i planujemy, co zawrzeć w każdym z nich, aby były dobrze napisane i tak proste, jak to tylko możliwe. Wreszcie, ponownie wykorzystujemy modele, tworząc nasz styl tworzenia wysokiej jakości oprogramowania idealnie dostosowanego do potrzeb biznesowych naszych klientów. Zarezerwuj telefon już dziś, aby dowiedzieć się o wszystkich możliwościach programowania modułowego, które możemy zaoferować podczas tworzenia oprogramowania.

>