Michał Pędziwiatr

AGH

ARCHITEKTURA MONOLITYCZNA

Nowoczesne architektury
aplikacji webowych

PLAN PREZENTACJI

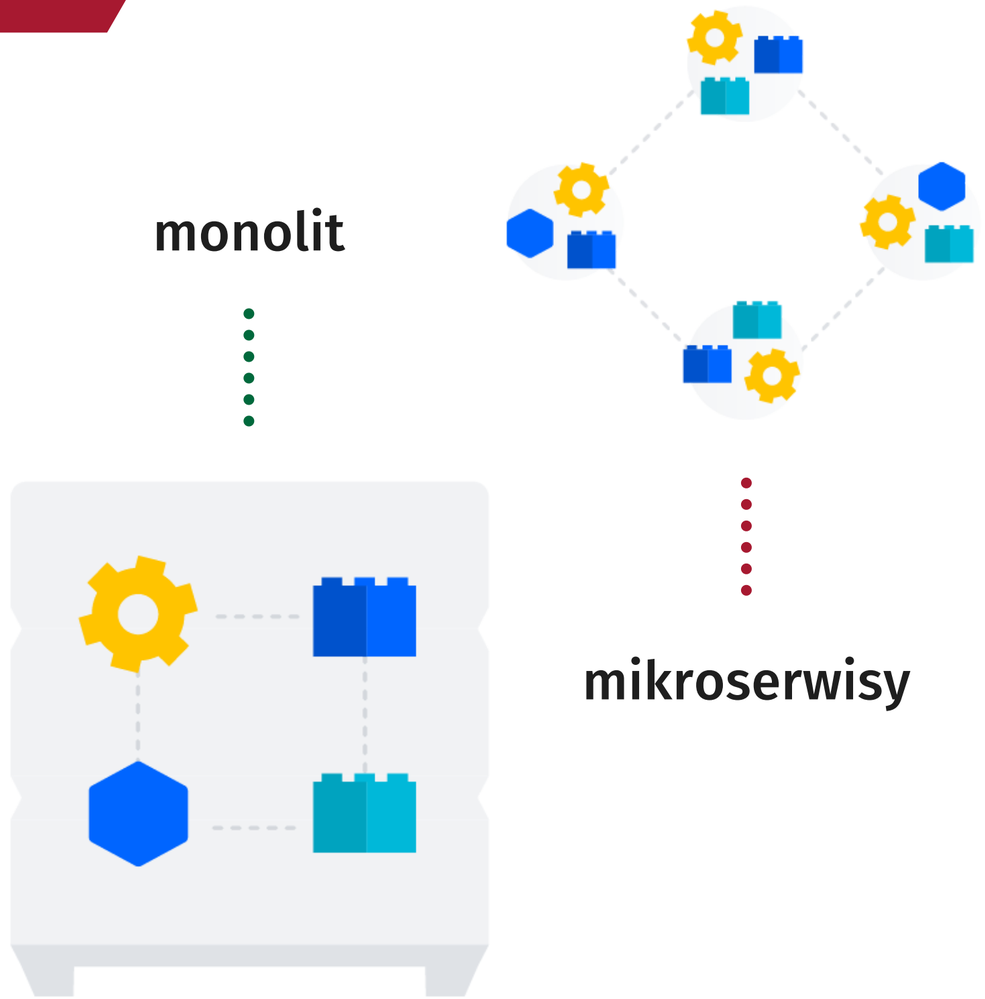
- Definicja i omówienie
- Wzorce architektoniczne
- Spojrzenie w przyszłość
- Cechy dobrej implementacji
- Zalety oraz wady
- Podsumowanie i źródła

Definicja

Architektura monolityczna

Architektura monolityczna to podejście do projektowania systemów informatycznych, w którym wszystkie operacje wykonują się **w jednym, niepodzielonym systemie**. Wszystkie dane gromadzone są w jednej centralnej bazie danych. Wydajność takiej aplikacji jest dedykowana jednej maszynie, co ma swoje zalety, takie jak łatwość zarządzania i monitorowania.

Wszystkie części składowe aplikacji są **ściśle połączone**, co oznacza, że lokalny błąd może wpłynąć na cały system. Te cechy powodują, że architektura monolityczna jest często wybierana do **tworzenia mniejszych aplikacji lub systemów**, gdzie integracja i współdziałanie komponentów jest kluczowe, a skomplikowane operacje można zrealizować w pojedynczym środowisku.

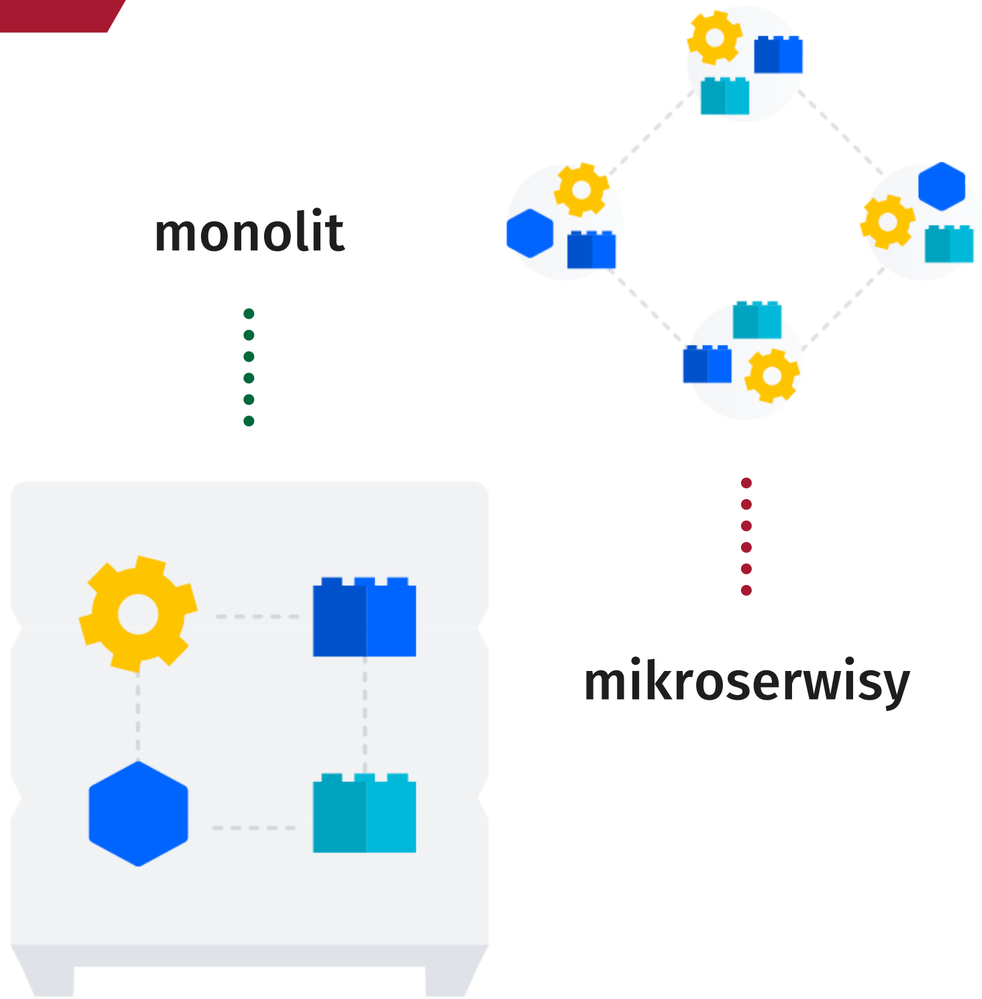


Definicja

Monolity opierają się na jednym, często skomplikowanym kodzie źródłowym, który łączy wszystkie funkcjonalności. Skalowanie w monolitach zazwyczaj oznacza **powielanie całego systemu**.

Monolity są często kojarzone — i to niesłusznie — z czymś przestarzałym. Jeśli są dobrze napisane i dostosowane do potrzeb biznesowych przedsiębiorstwa, i co istotne, dba się o nie regularnie, to mogą **efektywnie służyć przez lata**. Wiele odnoszących dziś sukcesy firm, takich jak m.in. Netflix, Spotify, Twitter, Amazon czy LinkedIn zaczynały swój biznes właśnie bazując na **monolicie**. Dopiero później, gdy przedsiębiorstwa te się rozrosły, zaczęły oferować więcej usług i stawiać na szybki rozwój, przerzuciły się na **architekturę mikroserwisów**.

Architektura monolityczna



Omówienie

Najważniejsze aspekty tworzenia i zarządzania aplikacją o architekturze monolitycznej

Wszystkie funkcje systemu, takie jak interfejs użytkownika, zarządzanie danymi, logika, są projektowane w ramach jednego systemu.

Kod jest pisany jako jedna aplikacja. Wszystkie funkcje są ściśle ze sobą powiązane i współdziałają ze sobą.

Wdrażanie jako jedna jednostka. Jeśli jedna funkcja wymaga aktualizacji lub modyfikacji, cała aplikacja musi być ponownie wdrożona.

Skalowanie polega na kopiowaniu i uruchomieniu całego systemu na większej ilości serwerów.

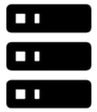
Błąd w jednym module może wpłynąć na cały system. Dlatego też, zarządzanie błędami i testowanie są kluczowe.

Aktualizacje mogą być trudne do zaplanowania i zarządzenia, ponieważ każda zmiana wymaga ponownego wdrożenia całego systemu.

Omówienie

Cechy dobrej implementacji aplikacji o architekturze monolitycznej

Czysty i zrozumiały kod: Kod powinien być napisany w sposób zrozumiały i łatwy do utrzymania. Powinien być dobrze skomentowany i zorganizowany, co ułatwia zarządzanie i rozwijanie aplikacji.



Modularność: Mimo że aplikacja monolityczna jest jedną jednostką, powinna być zorganizowana w sposób modularny. To oznacza, że różne funkcje i komponenty są oddzielone w ramach kodu, co ułatwia zarządzanie i rozwijanie aplikacji.

Wydajność: Aplikacja monolityczna powinna być wydajna i szybko reagować na żądania użytkowników. Powinna być zoptymalizowana pod kątem wydajności, aby zapewnić płynne i efektywne działanie.

Bezpieczeństwo: Bezpieczeństwo jest kluczowe w każdej aplikacji. Dobra aplikacja monolityczna powinna mieć silne mechanizmy bezpieczeństwa, aby chronić dane użytkowników i zapewnić bezpieczne działanie.

Skalowalność: Mimo że skalowanie może być wyzwaniem w aplikacjach monolitycznych, dobra aplikacja powinna być zaprojektowana w taki sposób, aby umożliwić skalowanie w miarę wzrostu liczby użytkowników lub zwiększenia obciążenia.

Omówienie

Cechy dobrej implementacji aplikacji o architekturze monolitycznej

Testowalność: Aplikacja powinna być łatwa do testowania. Powinna umożliwiać pisanie i uruchamianie testów jednostkowych, integracyjnych i funkcjonalnych, aby zapewnić, że wszystko działa poprawnie.

Zarządzanie błędami: Dobra aplikacja monolityczna powinna mieć skuteczne mechanizmy zarządzania błędami. Powinna być w stanie szybko wykrywać i naprawiać błędy, aby zapewnić ciągłość działania.

Integracja: W architekturze monolitycznej wszystkie komponenty są ściśle ze sobą zintegrowane. Dobra aplikacja monolityczna powinna zapewniać płynną integrację między różnymi komponentami, tak aby mogły one efektywnie współpracować i wymieniać się danymi.

Elastyczność: Dobra aplikacja monolityczna powinna być elastyczna. To oznacza, że powinna być w stanie dostosować się do zmieniających się wymagań biznesowych lub technologicznych bez konieczności całkowitego przepisywania kodu.

Dokumentacja: Każda dobra aplikacja powinna mieć kompletną i aktualną dokumentację. Dokumentacja powinna zawierać informacje o funkcjach aplikacji, jej architekturze, jak ją uruchomić i utrzymywać, a także jak rozwiązywać typowe problemy.

MVC
Model-View-Controller

MVC, czyli Model-View-Controller, to wzorzec projektowy często stosowany w architekturze aplikacji internetowych.

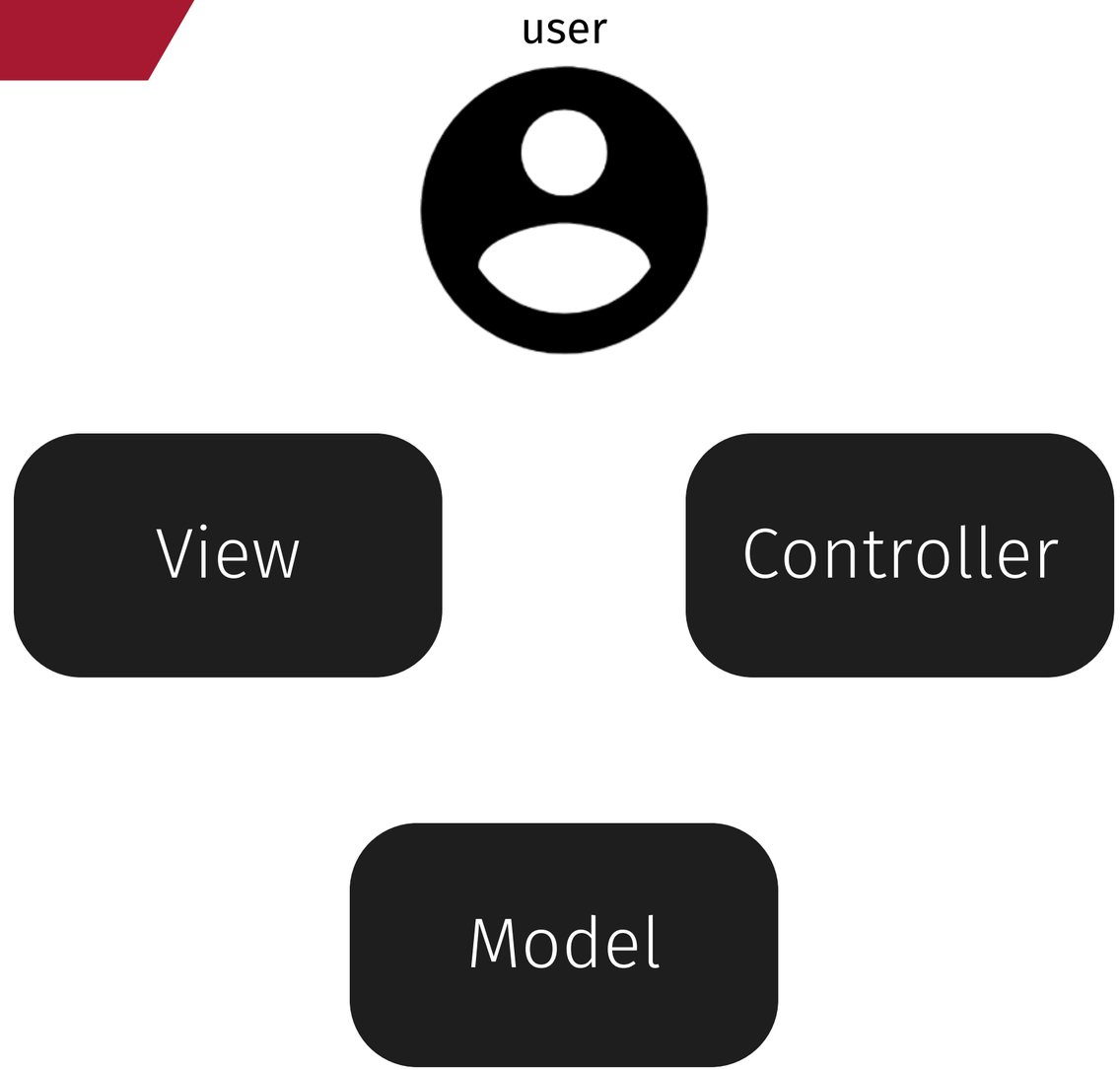
Model: Reprezentuje dane i logikę biznesową aplikacji, zarządza danymi, niezależnie od tego, czy pochodzą one z bazy danych, API czy obiektu JSON.

View: Odpowiada za to, co użytkownik widzi na ekranie. Jest to interfejs użytkownika aplikacji.

Controller: Jest “mózgiem” aplikacji, który kontroluje, jak dane są wyświetlane.

Wzorzec MVC jest często stosowany w nowoczesnych aplikacjach internetowych, ponieważ pozwala na skalowalność, utrzymanie i łatwą rozbudowę aplikacji.

Architektura monolityczna

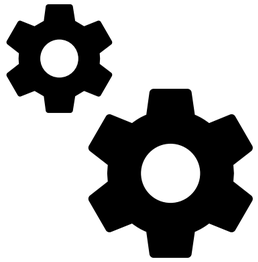


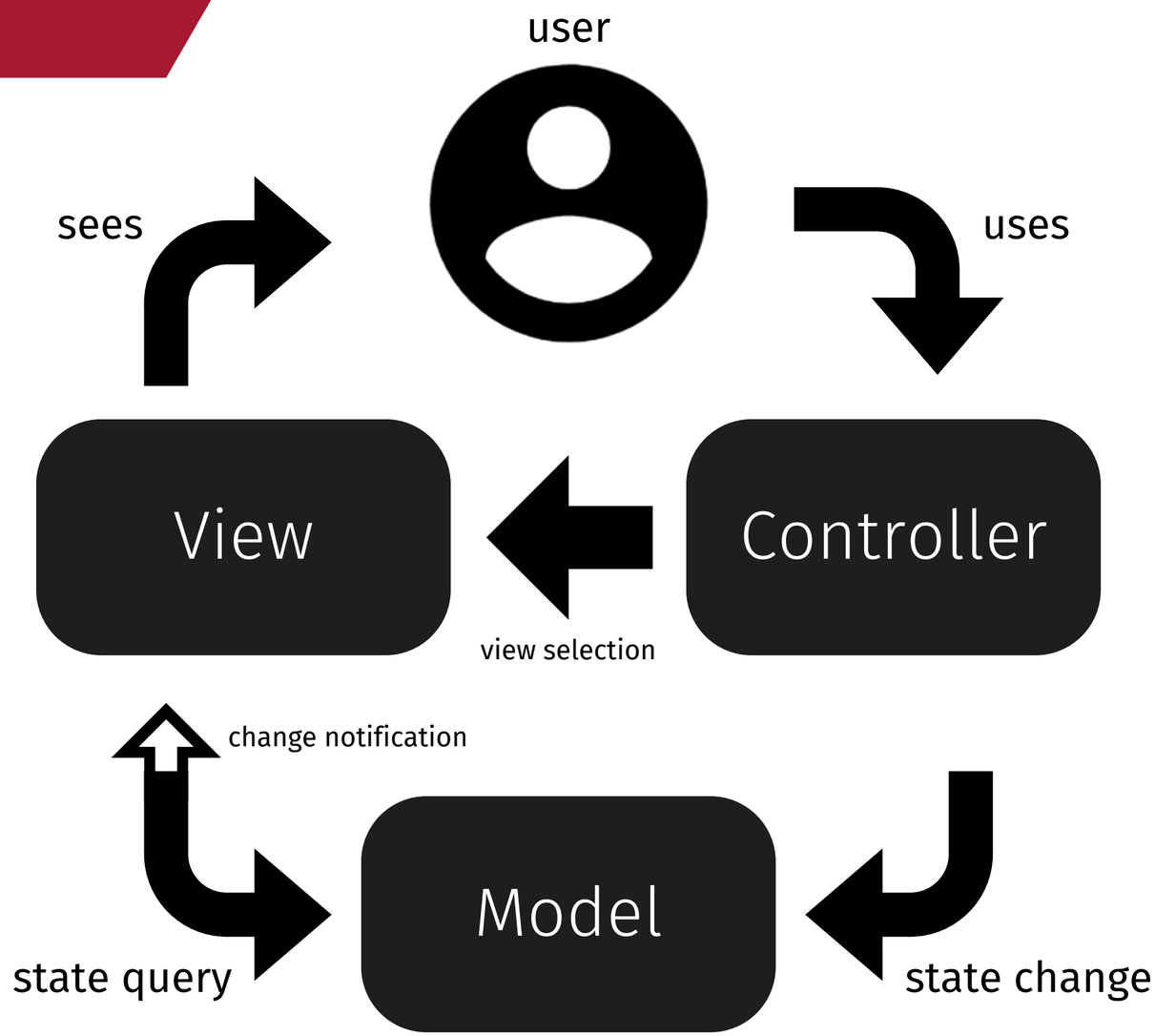
MVC

Model-View-Controller

Architektura monolityczna

Model związany jest z domeną aplikacji, zawiera jej elementy statyczne i behawioralne. Najważniejszą składową modelu jest **logika aplikacji** oraz stojąca za nią **logika biznesowa**. Innymi słowy, model określa działanie aplikacji oraz przetwarzane dane. Powinien być tak zaprojektowany, żeby był **niezależny** od wybranego rodzaju prezentacji oraz systemu obsługi akcji użytkownika (**model nic nie wie** o widoku i kontrolerze). Jedyne powiązanie wychodzące z **modelu** to powiadomienie widoku o aktualnych zmianach (change notification). W większości implementacji jest to rozwiązane za pomocą systemu komunikatów.



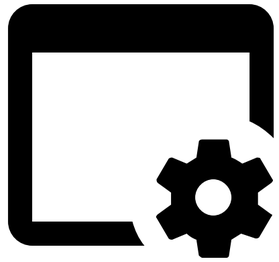


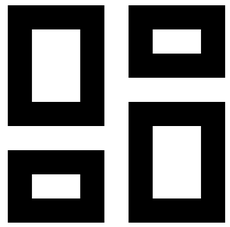
MVC

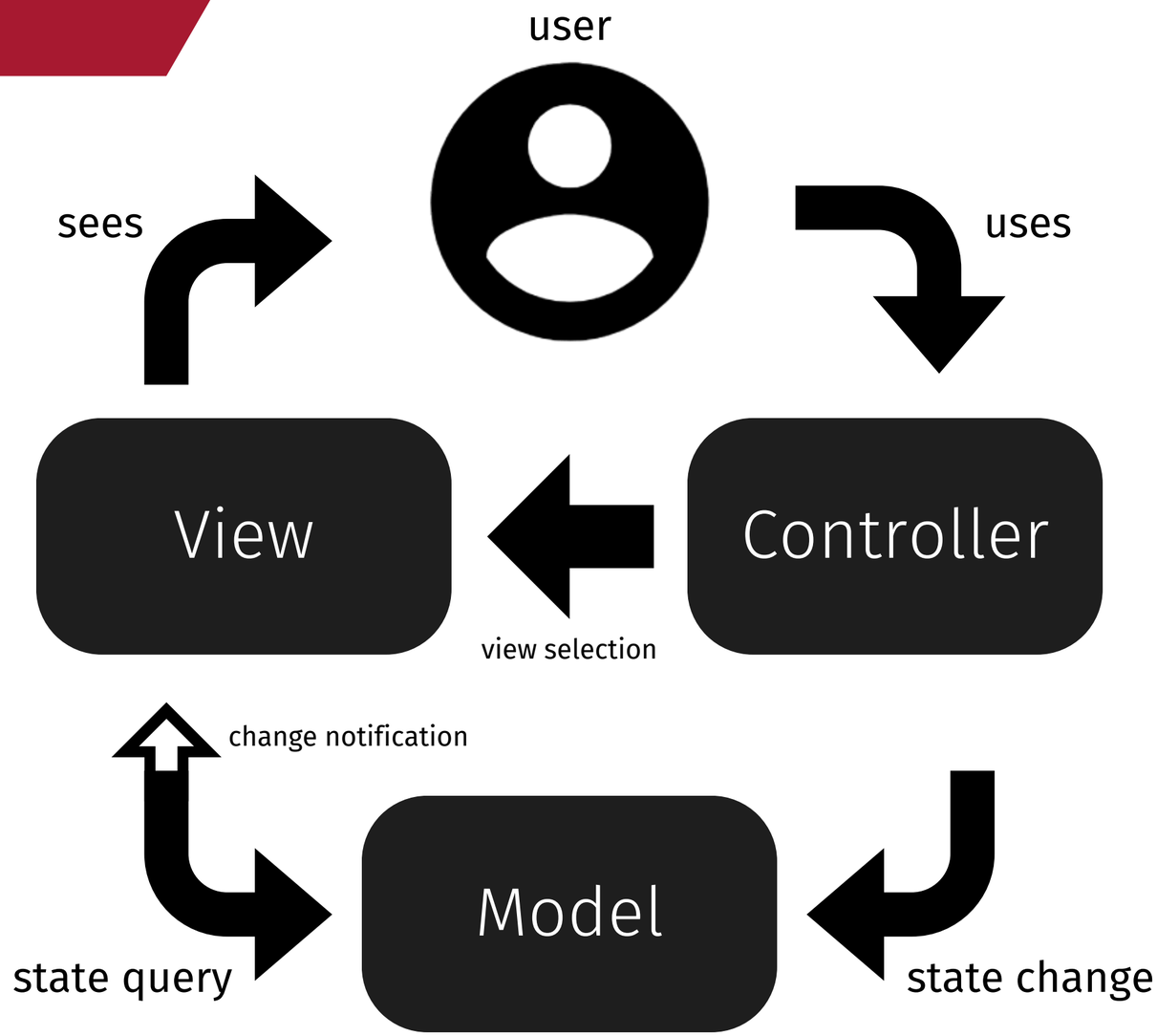
Model-View-Controller

Architektura monolityczna

Widok zarządza graficzną lub tekstową prezentacją modelu (jest jego **wizualnym odwzorowaniem**). Realizacja widoku jest już powiązana z konkretnym modelem. Prezentacja nie może zostać wygenerowana bez znajomości specyfiki danych czy operacji, które obrazuje użytkownikowi. Rysunek obok ilustruje to powiązanie: **widok** pobiera informacje z **modelu** (state query) ilekroć zostaje powiadomiony o jego **zmianie**. Z drugiej strony model **nie jest związany** ze sposobem jego prezentacji. W związku z tym zmiany **widoku** nie pociągają za sobą zmian w **modelu**.





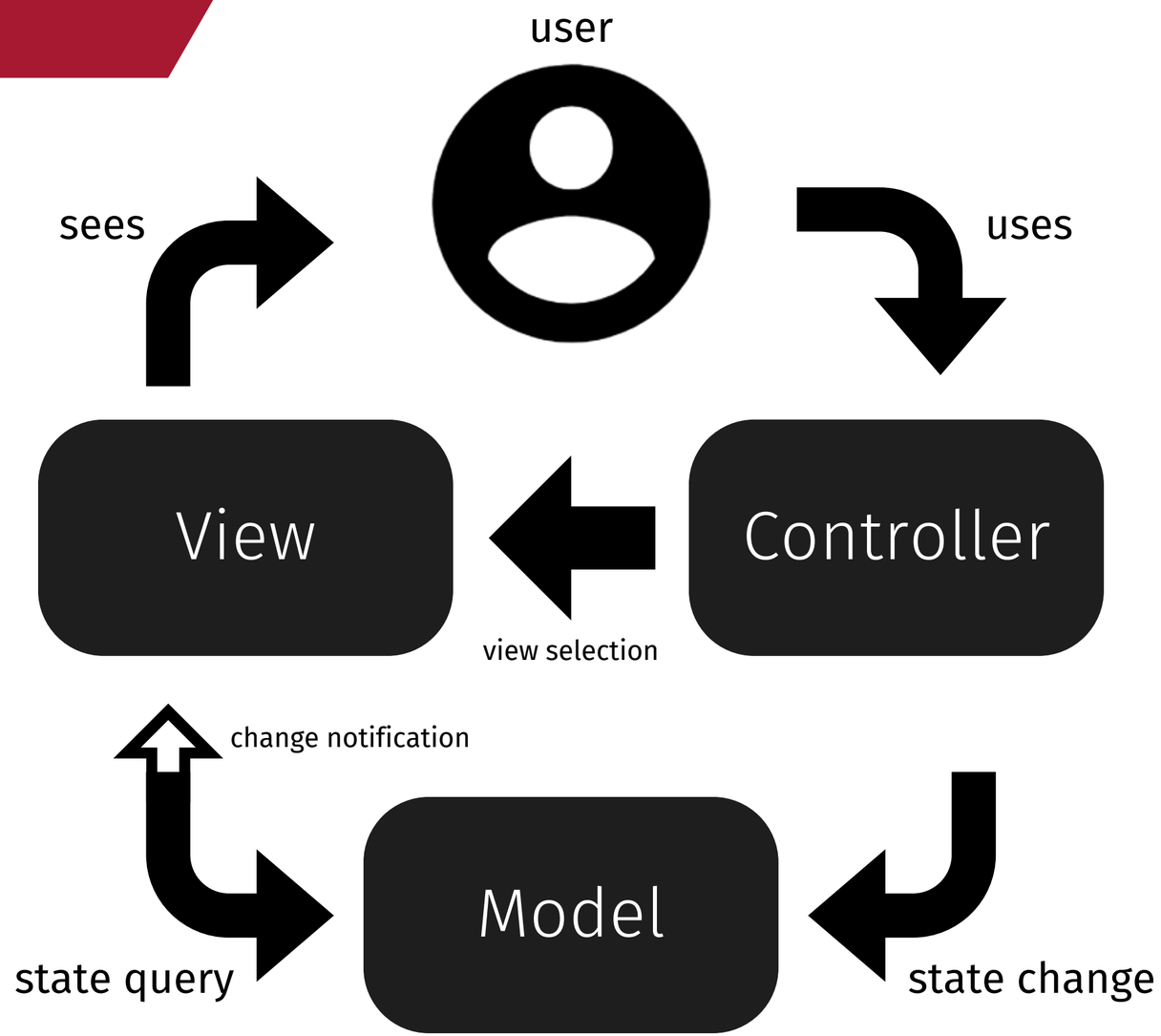


MVC
Model-View-Controller

Architektura monolityczna

Kontroler jest natomiast odpowiedzialny za reagowanie na **akcje użytkownika** (np. kliknięcia myszką), odwzorowując je na operacje zawarte w **modelu** (state change) oraz na zmiany **widoku** (view selection). W połączeniu z widokiem odpowiada za dwa aspekty interfejsu użytkownika (look and feel).

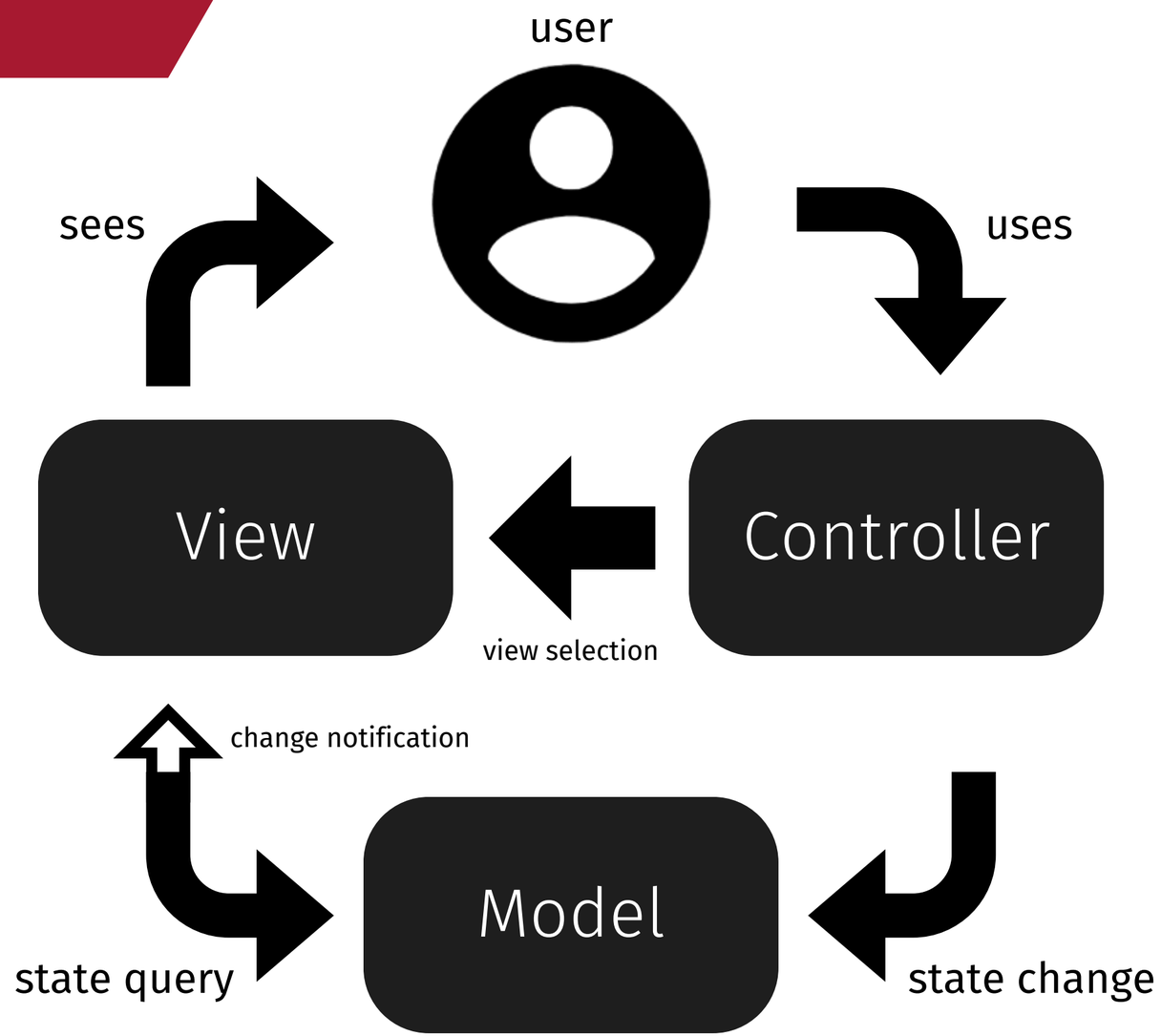
Element ten bywa **mylnie** uważany za segment reprezentujący aspekt behawioralny (działanie aplikacji) przy jednoczesnym założeniu, że **model** reprezentuje aspekt statyczny (dane). Tak rozumiany **kontroler**, jednocześnie odpowiedzialny za sterowanie widokiem (logika przetwarzania żądań użytkownika), **nie pozwala** na zmianę warstwy prezentacji bez modyfikacji logiki aplikacji (znajdującej się w tym przypadku w **kontrolerze**). Z tego względu **kontroler** nie powinien zawierać **logiki aplikacji**, jedynie odwołania do niej.



MVC
Model-View-Controller

Architektura monolityczna

Wartość szkieletu architektonicznego MVC leży w dwóch podstawowych zasadach.
Pierwsza to **separacja prezentacji i modelu**, umożliwiająca zmianę interfejsu użytkownika (np. udostępnienie usług aplikacji poprzez interfejs graficzny i tekstowy).
Druga zasada to **separacja widoku i kontrolera**. Klasycznym przykładem jest użycie dwóch kontrolerów związanych z jednym widokiem, które umożliwiają edycję widoku lub jej nie umożliwiają.
Tradycyjna postać MVC jest jednak często zdegenerowana. Degeneracja ta polega na mocnym połączeniu **kontrolera i widoku**. Znalazło to swoje odzwierciedlenie np. w bibliotece Swing na platformie Java. Projektanci biblioteki Swing zrezygnowali bowiem z dokładnego stosowania szkieletu MVC na rzecz architektury Model-Delegate.



MVC

Model-View-Controller

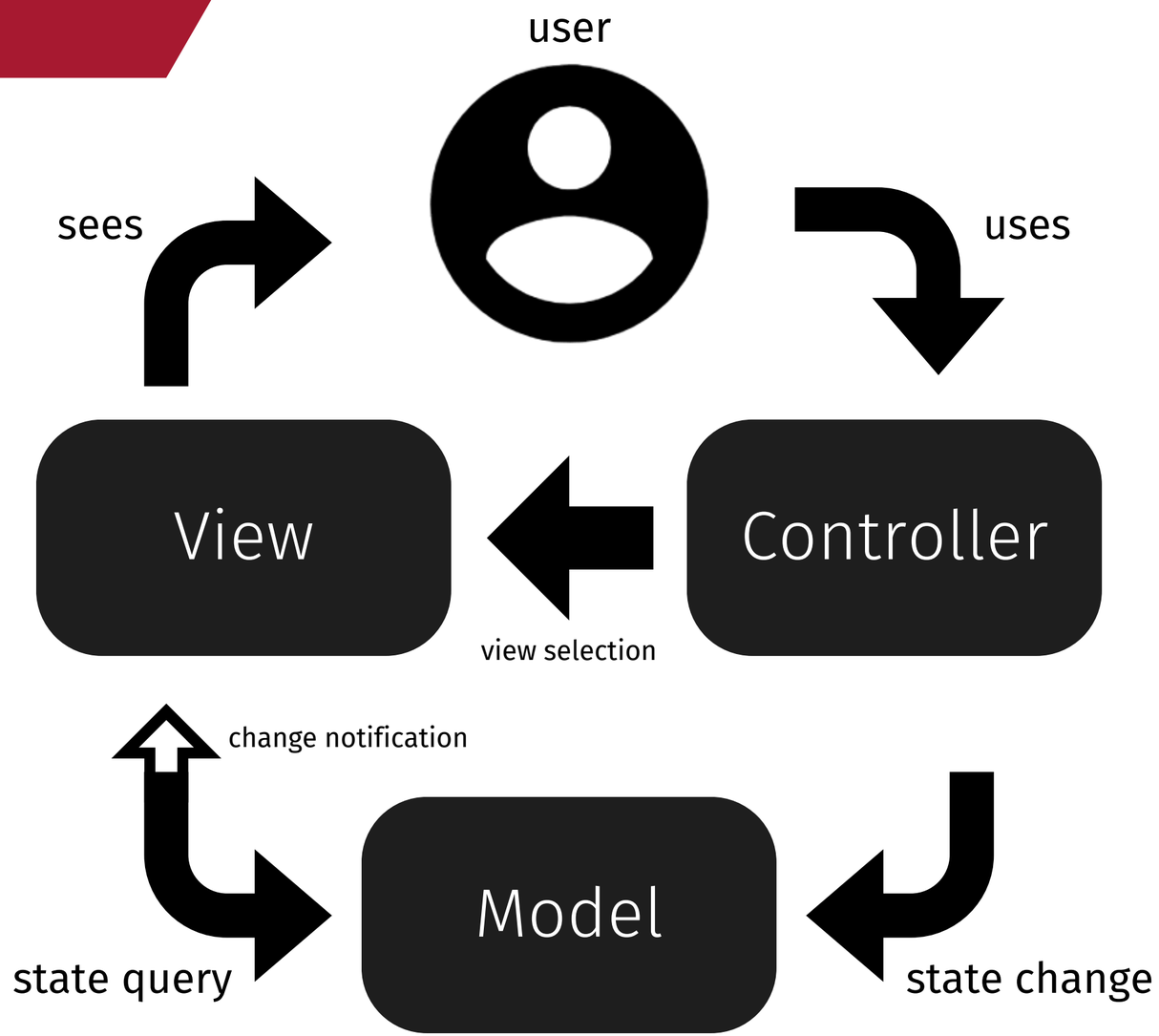
Architektura monolityczna

Zalety:

- Szybki rozwój aplikacji
- Współpraca
- Łatwość aktualizacji
- Łatwość debugowania
- Organizacja dużych aplikacji internetowych
- Wsparcie dla asynchroniczności
- Łatwość modyfikacji

Wady:

- Trudność zrozumienia
- Surowe zasady dotyczące metod
- Obciążenie dla mniejszych projektów



PCMEF

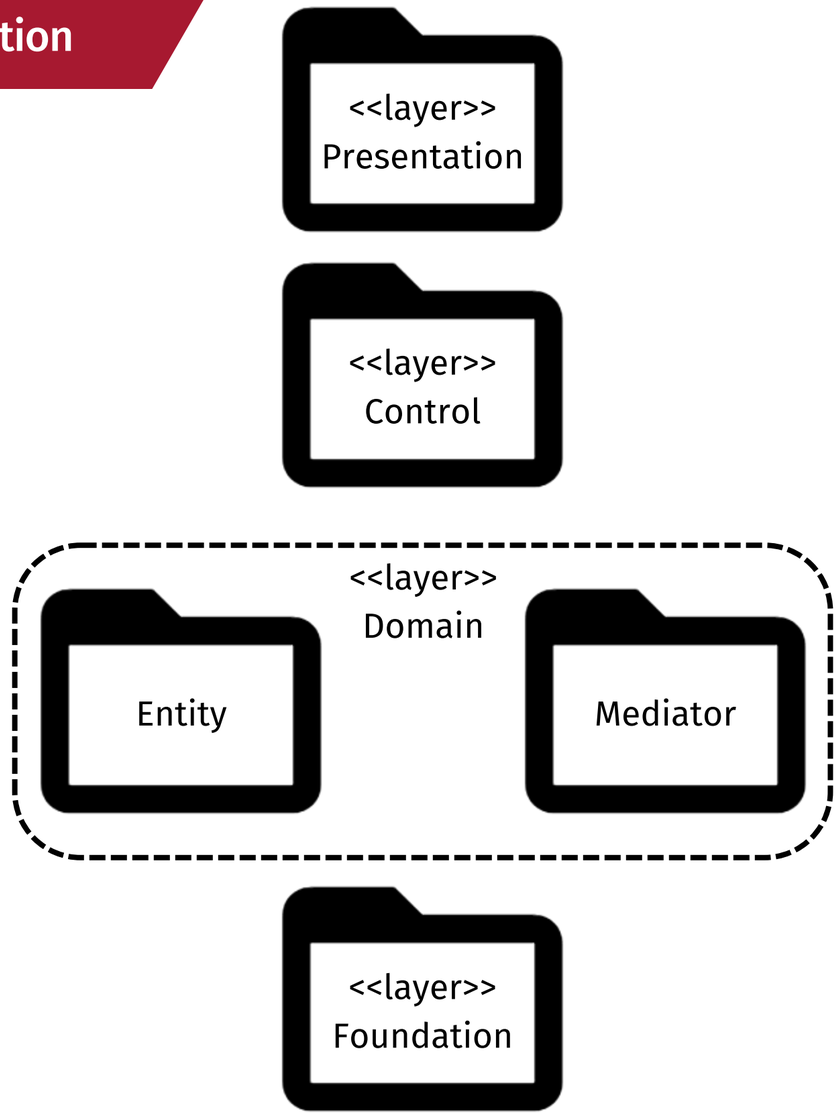
Presentation-Control-Mediator-Entity-Foundation

Architektura monolityczna

PCMEF to styl architektury składający się z **czterech warstw**. Jest pewnego rodzaju **ulepszeniem** w porównaniu do architektury MVC. Jest zalecany do celów zorientowanych na **interakcję, dane i komunikację**.

Podobnie jak inne ramy architektoniczne, takie jak MVC, **PCMEF** pozwala na budowanie dobrze strukturyzowanych aplikacji poprzez **minimalizację zależności** między modułami systemu.

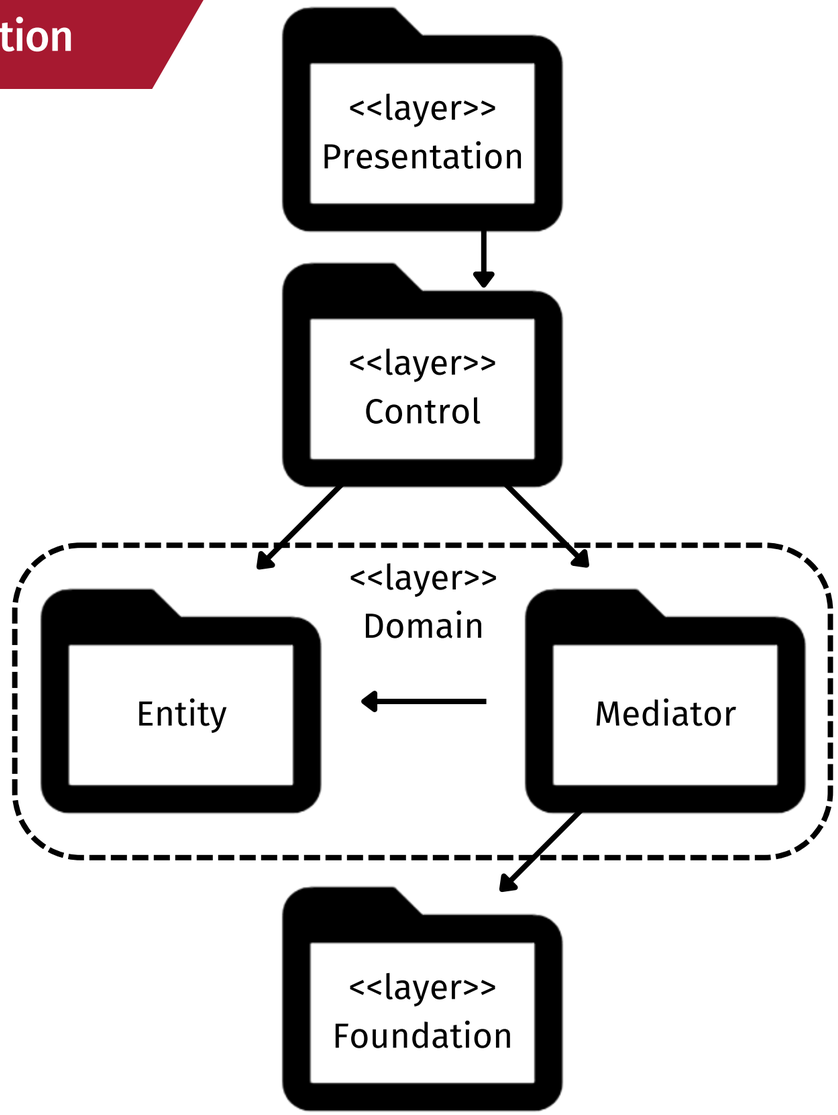
W kontekście architektury aplikacji internetowych, wzorzec **PCMEF** może być użyty do tworzenia aplikacji o dobrej strukturze, łatwej do utrzymania i skalowania.



PCMEF

Presentation-Control-Mediator-Entity-Foundation

Architektura monolityczna



Warstwa **Presentation** odpowiada za sposób **prezentacji aplikacji**. Składa się głównie z klas rozszerzających komponenty graficznych interfejsów użytkownika. Na przykład programując w języku Java, będą to klasy rozszerzające komponenty biblioteki Swing lub SWT. Elementem **MVC**, odpowiadającym warstwie **Presentation**, jest **widok** silnie połączony z **kontrolerem**. Z podobną sytuacją mamy do czynienia w przypadku wzorca Model-Delegat w bibliotece Swing.

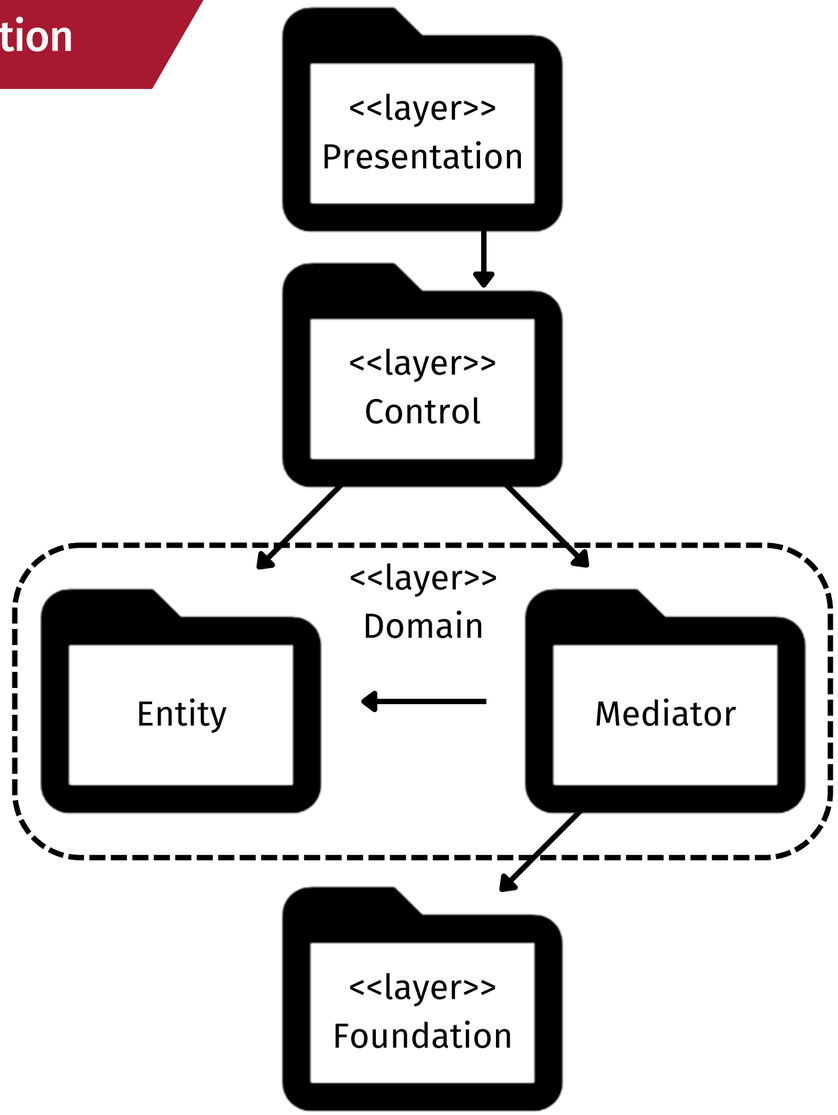
Warstwa **Control** odpowiada za obsługę żądań użytkownika przekazanych z wyższej warstwy. Zawiera głównie klasy odpowiedzialne za **logikę programu**. Przykładem klas należących do tej warstwy są implementacje interfejsów słuchaczy (ang. listeners), będących składowymi biblioteki Swing.

PCMEF

Presentation-Control-Mediator-Entity-Foundation

Pakiet **Entity**, należący do warstwy **Domain**, składa się z klas reprezentujących **obiekty biznesowe**. Są to obiekty, które zawierają operacje i dane biznesowe. Część z nich jest trwała (ang. persistent objects) i posiada odwzorowanie w zewnętrznych źródłach danych (np. w bazie danych).

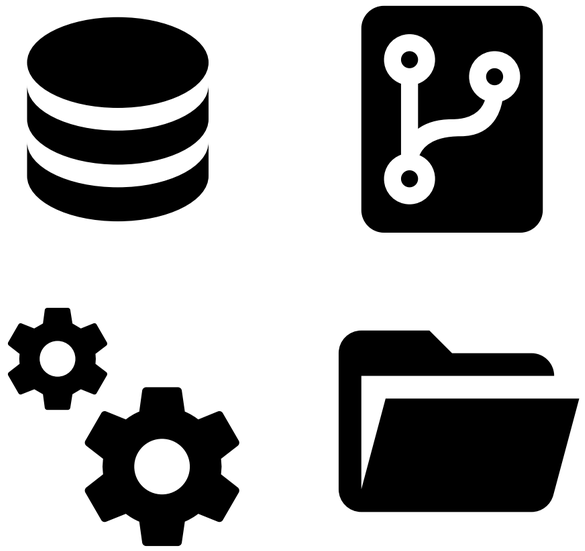
Pakiet **Mediator**, należący do warstwy **Domain**, pośredniczy pomiędzy pakietami **Control i Entity a pakietem Foundation**. Jego wprowadzenie ma na celu **usunięcie zależności Entity od Foundation**. Zlikwidowano w ten sposób konieczność modyfikacji **obiektów biznesowych** przy zmianie mechanizmów trwałości danych. Umożliwiono również **oddzielenie konstrukcji zapytań** do zewnętrznych źródeł danych od logiki aplikacji zawartej w warstwie **Control**.

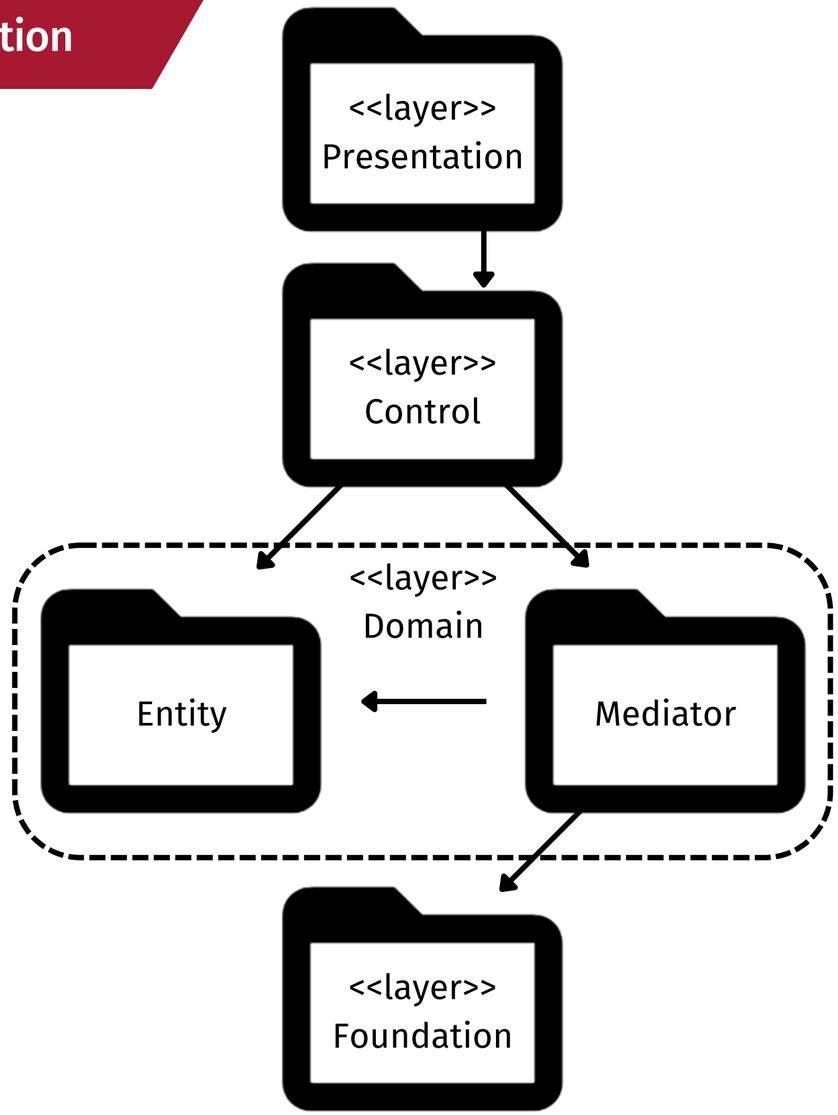


PCMEF

Presentation-Control-Mediator-Entity-Foundation

Warstwa **Foundation** zawiera klasy odpowiedzialne za obsługę zewnętrznych źródeł danych, takich jak bazy danych, repozytoria dokumentów, usługi sieciowe (ang. web services) czy systemy plików.





PCMEF

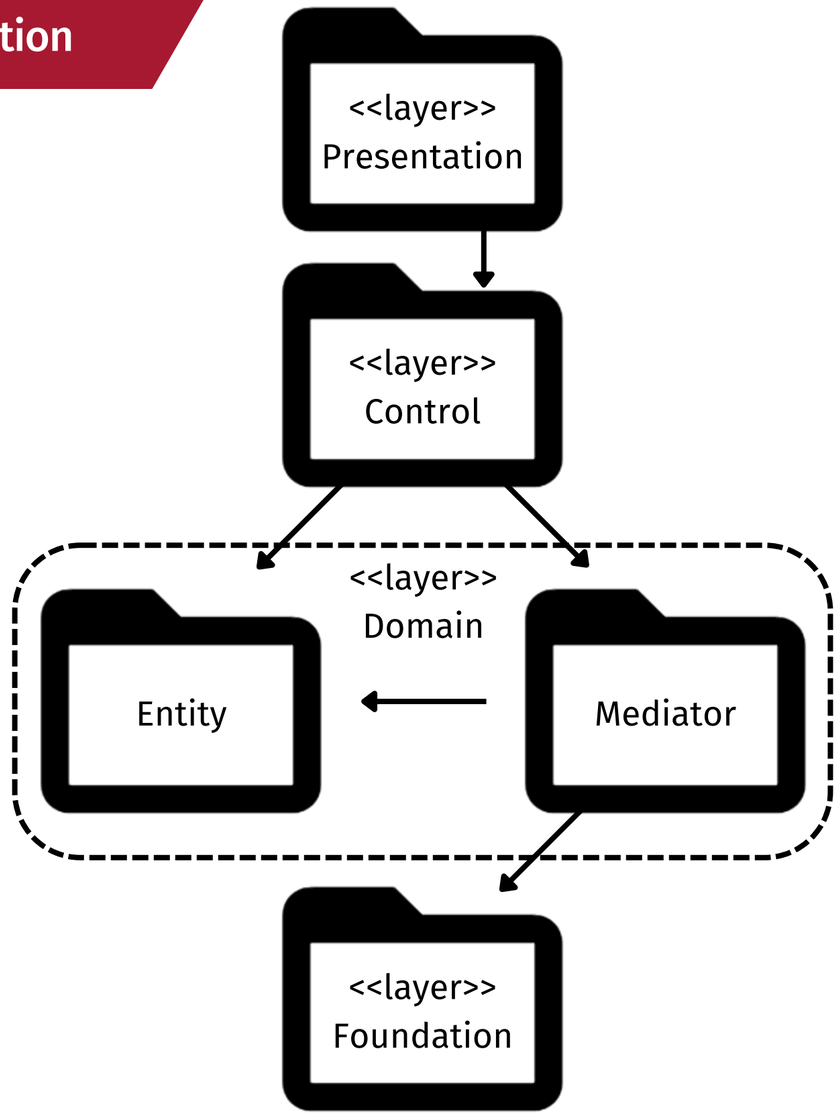
Presentation-Control-Mediator-Entity-Foundation

Zalety:

- Minimalizacja zależności i interkomunikacji obiektów
- Inspiracja uznawanymi wzorcami projektowymi
- Komunikacja oparta na zdarzeniach
- Dobra strukturyzacja
- Ewolucja istniejących podejść

Wady:

- Nieodpowiedni dla małych projektów
- Zmniejszona wydajność
- Trudność w utrzymaniu i skalowaniu
- Wymaga doświadczenia
- Wysoka złożoność



XWA

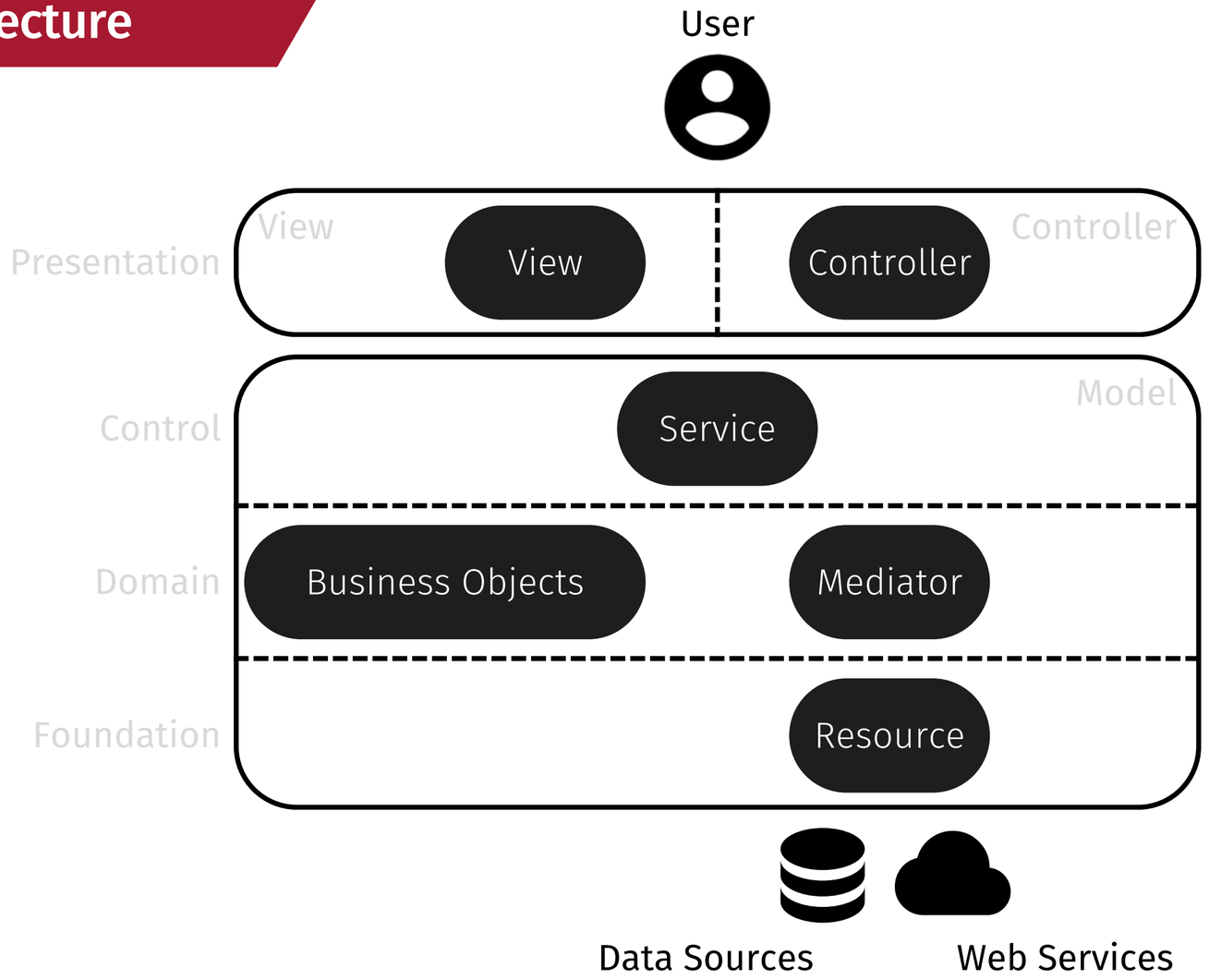
eXtensible Web Architecture

Architektura webowa **XWA**, znana jako **eXtensible Web Architecture**, jest wynikiem analizy istniejących wzorców architektonicznych, takich jak **MVC i PCMEF**.

Początki **XWA** sięgają badań nad strukturą i organizacją aplikacji internetowych. Właściwy wybór architektury dla tworzonego systemu stanowi poważne wyzwanie, szczególnie w przypadku dużych aplikacji internetowych wykorzystujących najnowsze możliwości technologiczne.

XWA to propozycja szkieletu architektonicznego, który uwzględnia specyfikę aplikacji internetowych. Szkielet ten umożliwia tworzenie dobrze ustrukturyzowanych aplikacji, **minimalizując zależności międzymodułowe**. XWA ma na celu uzyskanie systemu o odpowiedniej charakterystyce, takiej jak **utrzymywalność** (ang. maintainability) czy **skalowalność** (ang. scalability).

Architektura monolityczna



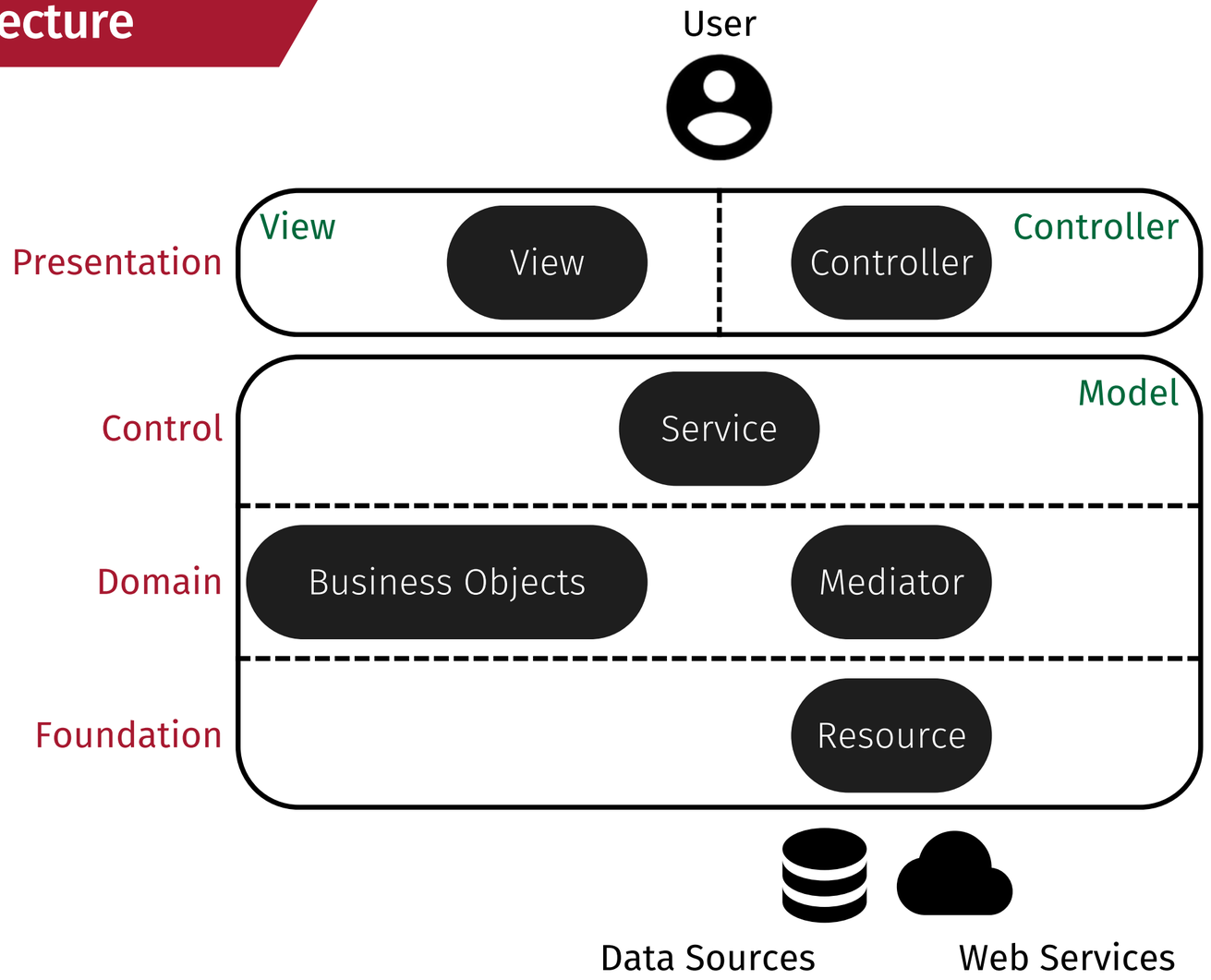
XWA

eXtensible Web Architecture

Architektura monolityczna

XWA jest **wariacją** omówionych wcześniej rozwiązań, dostosowaną do **specyfiki aplikacji internetowych**. Łączy zalety poprzedników zakładając separację klas widoku od kontrolera (zgodnie z **MVC**) oraz podział klas wewnętrznych modelu na strukturę hierarchiczną (zgodnie z **PCMEF**).

Jest to **architektura warstwowa** z wyraźnie wyodrębnioną triadą **MVC**. Składa się z **sześciu pakietów** ułożonych w **czteropoziomową hierarchię ilustrującą zależności** między pakietami (warstwy wyższe zależą od niższych). Zależności odwrotne są zminimalizowane do luźnych powiązań.

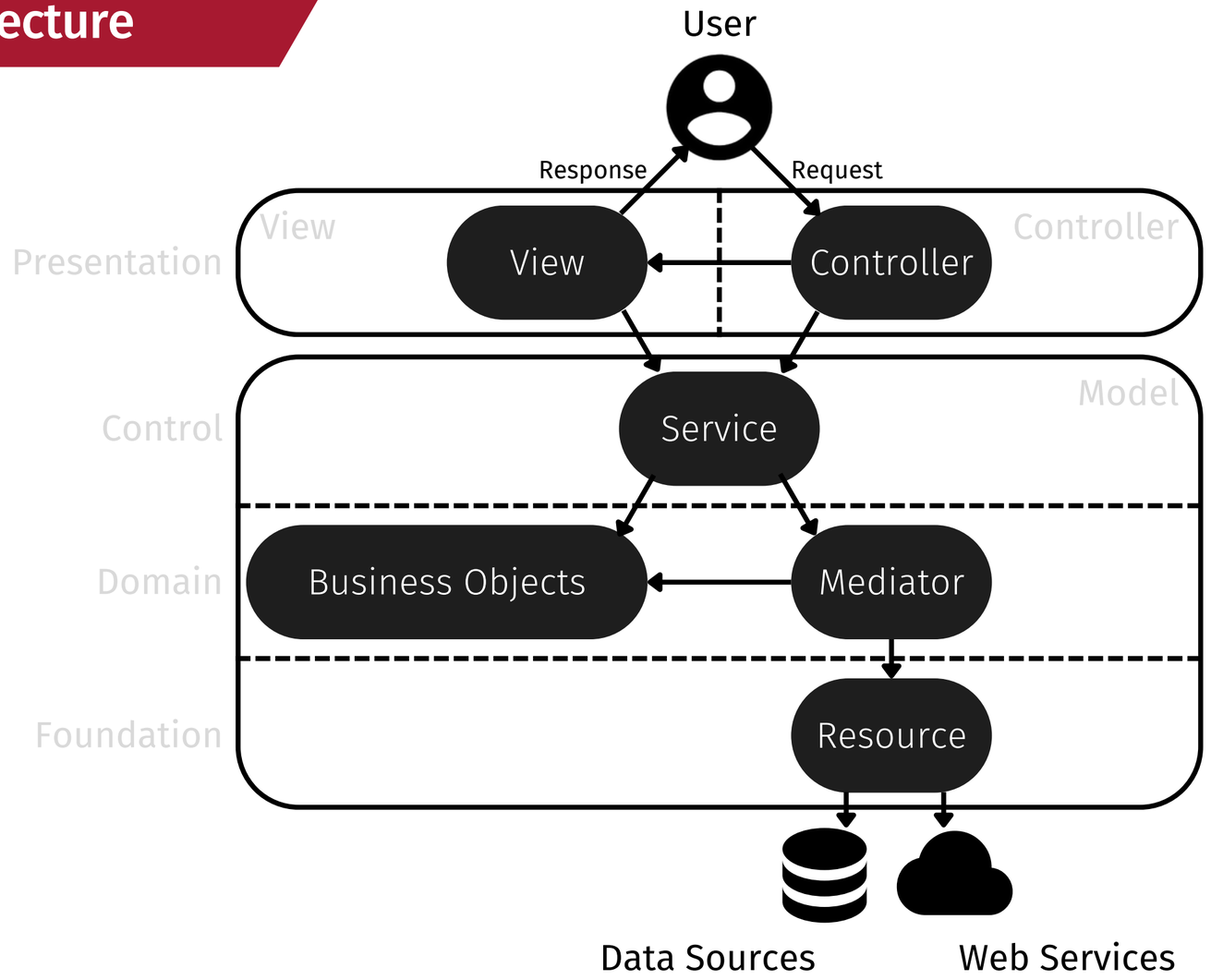


XWA
eXtensible Web Architecture

Pakiet **View** odpowiada za **prezentację aplikacji**. Jest to dokładny odpowiednik widoku w szkielecie **MVC**. W aplikacjach internetowych składa się z **plików opisujących wygląd stron WWW**. Mogą to być na przykład szablony w postaci plików HTML. Ciekawym rozwiązaniem wydaje się użycie technologii bazujących na metajęzyku XML w połączeniu z transformatami XSL. W takim przypadku pojawia się nowy (bazujący na XML) rodzaj kontraktów pomiędzy warstwami.

Pakiet **Controller** jest odpowiedzialny za **obsługę akcji użytkownika poprzez wywołanie logiki** zawartej w niższych warstwach. Jego zadaniem jest **oddzielenie specyfiki protokołu HTTP od logiki aplikacji**. Jest on odpowiedzialny za **sterowanie przepływem w ramach pojedynczej interakcji**, a także **sterowanie sekwencją interakcji**. Istotnym zadaniem tego pakietu jest również **sterowanie widokiem**.

Architektura monolityczna



XWA

eXtensible Web Architecture

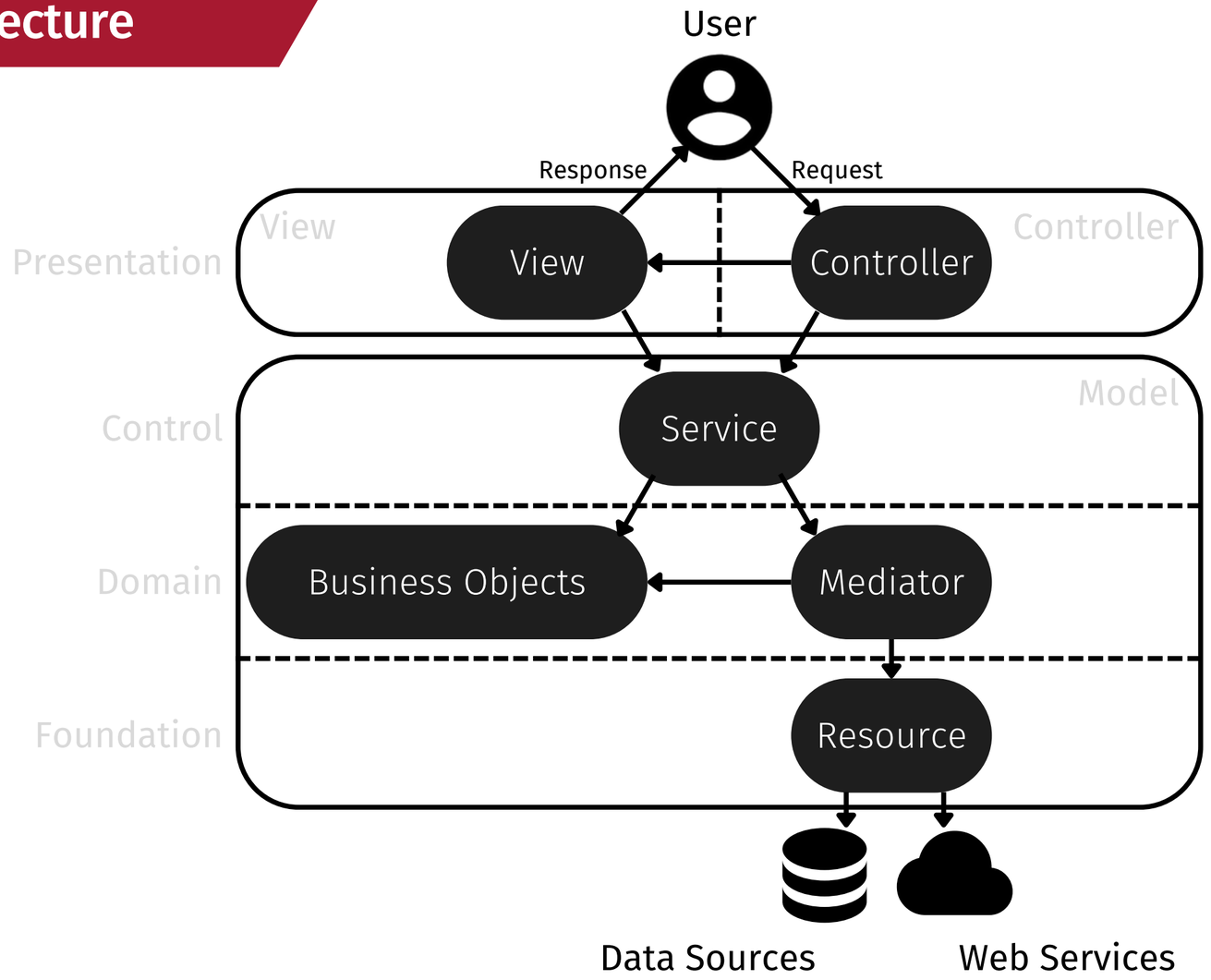
Pakiet **Service** jest odpowiedzialny za **udostępnienie usług systemu**. Centralizuje logikę aplikacji bazującej na wielu **obiektach biznesowych** oraz wymagającej dostępu do **zewnętrznych źródeł danych czy usług internetowych**. Klasy tego pakietu mogą realizować wzorce projektowe **Application / Service Layer**.

Pakiet **Business Objects** zawiera **obiekty biznesowe**, które tworzą model domenowy aplikacji. Ich implementacja bazuje na wzorcach **Business Object lub Domain Model**.

Pakiet **Mediator** jest warstwą pośredniczącą w dostępie do **zewnętrznych źródeł danych**, mechanizmów trwałości danych czy **usług sieciowych**. Klasy tego pakietu zazwyczaj realizują wzorzec **Data Access Object**.

Pakiet **Resource** realizuje **niskopoziomową obsługę dostępu** do zewnętrznych zasobów.

Architektura monolityczna



XWA

eXtensible Web Architecture

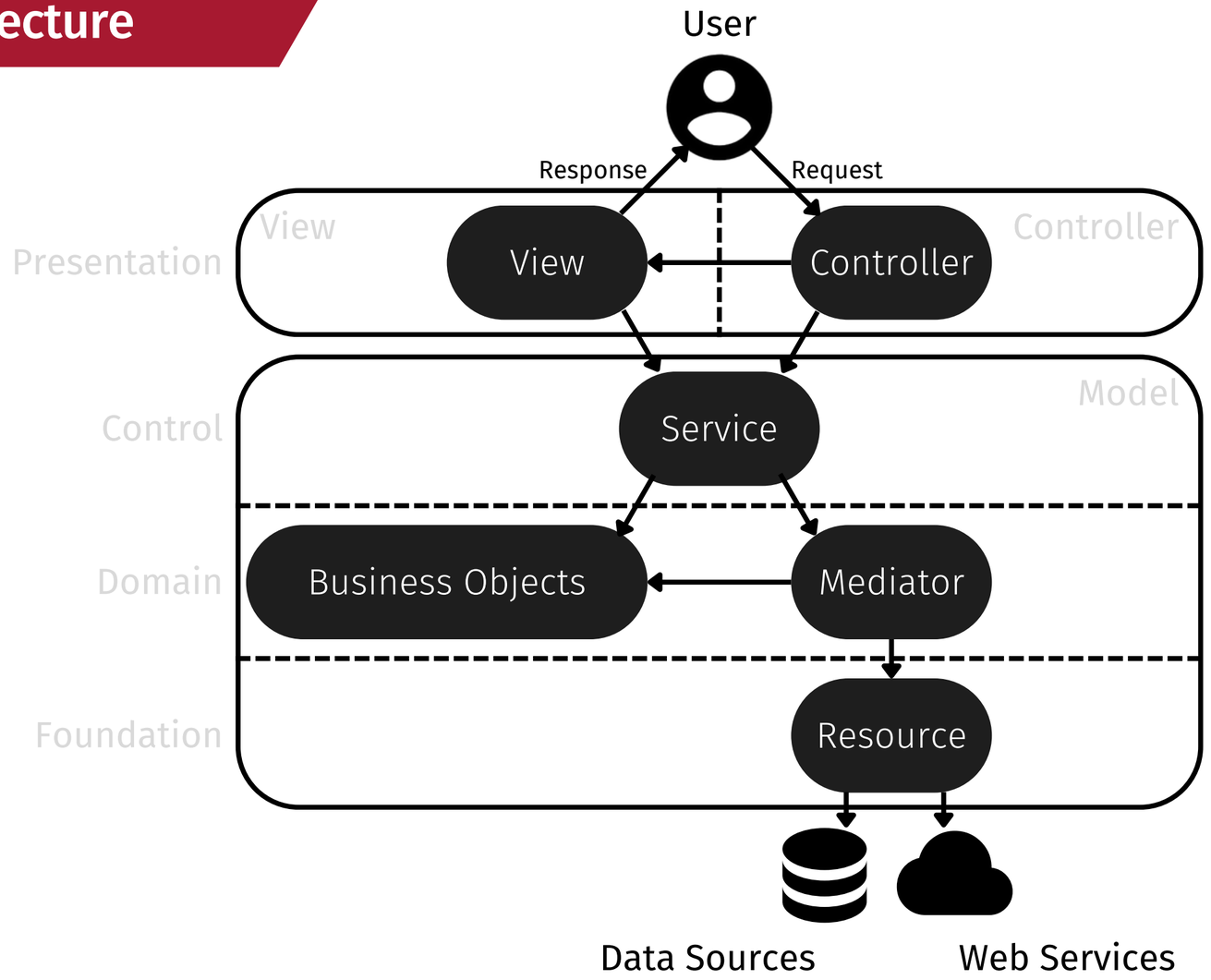
Architektura monolityczna

Zalety:

- Modularność
- Łatwa skalowalność
- Dobra utrzymywalność
- Wysoka elastyczność
- Ułatwiona integracja komponentów

Wady:

- Złożoność
- Długi czas implementacji
- Duże wymagania dotyczące zasobów
- Trudność w debugowaniu
- Zależność od innych technologii



Przyszłość

Spojrzenie w przyszłość na nowoczesne architektury aplikacji webowych

Wybór architektury aplikacji **zależy od wielu czynników**, takich jak skomplikowanie systemu, wymagania dotyczące skalowalności, wydajności, bezpieczeństwa, łatwości utrzymania i elastyczności.

Monolityczna architektura ma swoje zalety, takie jak prostota, łatwość wdrażania i testowania. Jest to szczególnie korzystne dla małych do średnich aplikacji, gdzie złożoność jest niska. Jednak w miarę jak aplikacja rośnie, monolit może stać się trudny do zarządzania i skalowania.

Z drugiej strony, **architektury oparte na mikrousługach**, które są przeciwieństwem monolitów, są bardziej skalowalne i elastyczne, ale są również bardziej złożone do zarządzania i wdrażania.

W związku z tym, czy monolityczna architektura będzie architekturą przyszłości, zależy w dużej mierze od **trendów technologicznych, wymagań biznesowych i ewolucji narzędzi i praktyk inżynierii oprogramowania**. W niektórych przypadkach monolit może być najlepszym rozwiązaniem, podczas gdy w innych przypadkach inne architektury mogą być bardziej odpowiednie.

AGH

Podsumowanie

Informacje na temat architektury monolitycznej w aplikacjach webowych, jej wzorców i cech

1. **Architektura monolityczna to model, w którym wszystkie funkcje aplikacji są zarządzane w jednym, niepodzielnym systemie. Wszystkie komponenty są ze sobą ściśle powiązane i działają jako jedna jednostka.**

2. **Dobry monolit powinien być prosty w utrzymaniu, łatwy do zrozumienia i efektywny w działaniu. Powinien również umożliwiać łatwe skalowanie i być odporny na błędy.**

3. **Model-View-Controller (MVC) to popularny wzorzec architektoniczny stosowany w monolitach, który oddziela logikę biznesową (Model) od interfejsu użytkownika (View) i sterowania (Controller).**

4. **PCMEF to styl architektoniczny, który stanowi pewnego rodzaju ulepszenie w porównaniu do architektury MVC . Składa się z czterech warstw: Presentation, Control, Domain, Foundation. Zaleca się go do celów związanych z interakcją, danymi i komunikacją.**

5. **XWA to propozycja szkieletu architektonicznego, który stara się odpowiedzieć na dotychczasowe wyzwania architektoniczne. Bazuje na dwóch dobrze znanych szkieletach (MVC i PCMEF), ale uwzględnia specyfikę aplikacji internetowych. XWA nie jest tylko zjawiskiem teoretycznym, ale ma swoją praktyczną implementację.**

Źródła

Przydatne źródła, materiały, artykuły i dokumenty użyte w tej prezentacji

1. Lech Madeyski, Michał Stochmiałek
madeyski.e-informatyka.pl | stochmialek.pl
“Architektura nowoczesnych aplikacji internetowych”

2. Medium (Transparent Data) | medium.com | transparentdata.pl
“Monolity vs. mikroserwisy — zalety i wady [porównanie]”

3. Boring Owl (Tomasz Kozon) | boringowl.io
“Mikroserwisy vs Monolit: Analiza i porównanie dwóch architektur”

4. Atlassian (Chandler Harris) | atlassian.com
“Porównanie mikrousług z architekturą monolityczną”

5. Don’t Code Tired (Jason Roberts) | dontcodetired.com
“The PCMEF architectural Framework”

6. WorldMaker (Max Battcher) | blog.worldmaker.net
“The Django Framework Architecture”