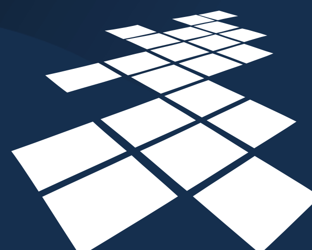
Projektowanie

wykład prowadzi
Mikołaj Morzy



UCZELNIA
ONLINE

Plan wykładu

- Projektowanie warstwy prezentacji
  - Sesja, kontrola dostępu, duplikacja, walidacja
- Złe praktyki w warstwie prezentacji
- Projektowanie warstwy biznesowej
  - Komponenty sesyjne, komponenty encyjne
- Złe praktyki w warstwie biznesowej
- Refaktoryzacja warstwy prezentacji
- Refaktoryzacja warstwy biznesowej

Celem wykładu jest przedstawienie podstawowych zagadnień związanych z poprawnym projektowaniem aplikacji internetowych. Podczas wykładu skupimy się na warstwach prezentacji i biznesowej. W pierwszej części omówione zostaną istotne elementy składające się na warstwę prezentacji: zarządzanie sesją, kontrola dostępu klienta, duplikacja formularzy, walidacja danych. Następnie przedstawione zostaną błędne praktyki i rozwiązania, które często są spotykane w warstwie prezentacji. W drugiej części przedstawimy najważniejsze zasady związane z projektowaniem warstwy biznesowej, w szczególności skupimy się na poprawnym stosowaniu komponentów sesyjnych i encyjnych. Podobnie jak poprzednio, przedstawiony zostanie katalog często spotykanych błędów projektowych. Trzecia część wykładu będzie poświęcona refaktoryzacji istniejących już aplikacji – przedstawione zostaną proste modyfikacje poprawiające istniejący projekt i ulepszające aplikację.

Zarządzanie sesją

- Sesja: konwersacja obejmująca wiele żądań i odpowiedzi przesyłanych między klientem i serwerem
- Stan sesji po stronie klienta HTTP
  - Ukryte pola formularza HTML
  - Cookies
- Stan sesji w warstwie prezentacji
  - Demarkacja sesji
- Stan sesji w warstwie biznesowej i warstwie zasobów

Sesją HTTP nazywamy konwersację, na którą składa się wiele żądań i odpowiedzi przesłanych między klientem HTTP i serwerem. Identyfikacja sesji jest zadaniem trudnym, w szczególności w przypadku wykorzystywania serwerów proxy lub współdzielenia komputerów przez wielu użytkowników. Protokół HTTP jest protokołem bezstanowym, co w dużej mierze utrudnia tworzenie aplikacji internetowych.

Stan sesji może być przechowywany po stronie klienta HTTP, konieczne jest wówczas umieszczanie stanu sesji w każdym dokumencie HTML wysyłanym klientowi. Stan sesji może być przekazany klientowi poprzez ukryte pola formularza HTML (znacznik <input type="hidden">). Rozwiązanie to posiada wiele wad. Dokumenty stają się duże, ponieważ każdy dokument przechowuje cały stan sesji, wzrasta obciążenie komunikacji sieciowej. Zjawisko to staje się bardzo niekorzystne w przypadku gdy sesja zawiera wiele danych. Dane sesji nie są szyfrowane i stan sesji jest widoczny w kodzie HTML. Stan sesji jest ograniczony tylko do danych tekstowych, referencje do obiektów muszą być serializowane. Innym rozwiązaniem jest przechowywanie stanu sesji w zmiennych cookies. Rozwiązanie to posiada podobne wady jak wykorzystanie ukrytych pól formularzy HTML. Dodatkowo, pojawiają się ograniczenia dotyczące rozmiaru i typu elementów stanu sesji u klienta oraz ograniczenie dotyczące długości nagłówka HTTP. Kolejnym problemem jest bezpieczeństwo i poufność stanu sesji po stronie klienta.

W aplikacjach klasy enterprise zaleca się przechowywanie stanu sesji po stronie serwera. Klient jest zobowiązany do dostarczenia identyfikatora sesji (przekazywany przez zmienną cookie lub jako część adresu URL), zmienne sesji są przechowywane w warstwie prezentacji. Takie rozwiązanie wymaga demarkacji sesji. Zazwyczaj sesja kończy się gdy (1) upłynął określony czas bezczynności (2) sesja została jawnie zakończona. Przechowywanie stanu sesji w warstwie prezentacji pozwala na przechowywanie stanu sesji postaci obiektów (brak konieczności serializacji obiektów), wspiera skalowalność i wpływa pozytywnie na wydajność systemu.

Istnieje także możliwość wykorzystania warstw biznesowej lub zasobów do przechowywania stanu sesji. W pierwszym przypadku stan sesji jest przechowywany w komponentach encyjnych, w drugim przypadku jest umieszczany w relacyjnej bazie danych. Oba rozwiązania są jednak mniej wskazane niż przechowywanie stanu sesji w warstwie prezentacji.

Kontrola dostępu klienta

- Ochrona widoku
  - Mechanizm ochrony "wszystko albo nic"
  - Dołączanie ochrony do fragmentów widoku
  - Ochrona widoku na podstawie ról
  - Ochrona przez konfigurację
    - Standardowe zasady bezpieczeństwa
    - Wykorzystanie katalogu /WEB-INF

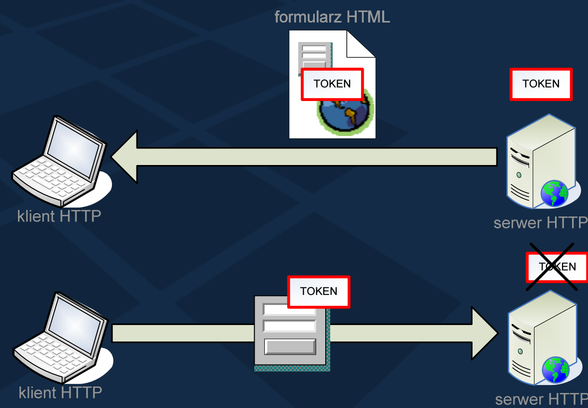
Ochrona widoku jest najczęściej związana z tym, że pewne fragmenty aplikacji powinny być dostępne tylko dla uprawnionych użytkowników, np. po identyfikacji i podaniu hasła dostępu. Dwie podstawowe metody ochrony widoku to (1) użycie kontrolera zarządzającego dostępem do elementów i odpowiednia konstrukcja logiki aplikacji (2) konfiguracja aplikacji w taki sposób, aby wywołanie widoku było możliwe tylko jako wewnętrzne wywołanie z innego zasobu aplikacji. W przypadku użycia kontrolera ochronie może podlegać cały widok (ochrona typu "wszystko albo nic"), wówczas logika związana z ochroną powinna zostać scentralizowana w kontrolerze a nie umieszczana w poszczególnych widokach. Możliwe jest również chronienie fragmentów widoku. Fragmenty widoku mogą być wyświetlane w przypadku gdy użytkownik dysponuje właściwą rolą lub w zależności od stanu aplikacji (np. fragment widoku zawierający koszyk zakupów nie jest wyświetlany do momentu dodania pierwszego produktu do koszyka).

Serwer aplikacji umożliwia ochronę zasobów poprzez właściwą konfigurację środowiska aplikacji internetowej. Podstawowym mechanizmem ochrony są deklaracje wpisywane w pliku wdrożenia web.xml. Przykładowo, dodanie znaczników <security-constraint> zawierających wzorce adresów URL (znacznik <url-pattern>) do pliku wdrożenia web.xml powoduje objęcie zasobów opisanych przez adresy URL zasadami ochrony. Zasoby chronione w ten sposób nie są dostępne bezpośrednio z poziomu klienta HTTP, natomiast można do nich się odwoływać z poziomu kontrolera. W przypadku prostej ochrony wystarcza umieszczenie zasobu w katalogu /WEB-INF, każdy zasób umieszczony w tym katalogu jest dostępny tylko i wyłącznie z poziomu kontrolera (np. przez mechanizm RequestDispatcher).

Zaawansowane aplikacje internetowe

Duplikacja formularzy HTML

- Dwukrotne wysłanie formularza
- Token synchronizujący (Déjà vu)



Projektowanie (5)

Możliwość wykorzystania przycisków przeglądarki Wstecz, Przeładuj i Stop powoduje, że użytkownicy mogą nieświadomie wielokrotnie przesyłać ten sam formularz HTML. Takie zachowanie jest bardzo niepożądane z punktu widzenia logiki aplikacji internetowej. W praktyce chcemy wychwytywać i obsługiwać zdarzenie duplikacji formularzy HTML. Podstawowym mechanizmem ochrony przed duplikacją formularzy HTML jest wprowadzenie tokenu synchronizującego (tzw. rozwiązanie "deja vu") do kontrolera sterującego. Token synchronizujący jest dodawany do każdego formularza HTML przesyłanego do klienta HTTP i przesyłany wraz z formularzem z powrotem. W serwerze HTTP następuje porównanie tokenu przechowywanego w sesji klienta z tokenem przesłanym w formularzu. Jeśli są identyczne, to oznacza to poprawne pierwsze wysłanie formularza, token z sesji jest usuwany lub modyfikowany. Jeśli tokeny są różne, to nastąpiła duplikacja zgłoszeń formularza.

Zaawansowane aplikacje internetowe

Walidacja

- Walidacja po stronie klienta
  - Język skryptowy umieszczony w widoku
  - Prosta walidacja: formaty, pola wymagane
- Walidacja po stronie serwera
  - Walidacja oparta na formularzu
  - Walidacja oparta na typach abstrakcyjnych

Projektowanie (6)

Walidacja, czyli sprawdzenie poprawności danych, może zachodzić zarówno po stronie klienta, jak i po stronie serwera. Walidacja po stronie klienta jest za zwyczaj znacznie prostsza i implementowana w postaci kodu napisanego w języku skryptowym (np. JavaScript) osadzonym w widoku. Tego typu walidacja najczęściej dotyczy podstawowych testów: czy wszystkie wymagane pola zostały wypełnione, czy wprowadzone dane mają poprawny format (liczba, kod pocztowy, nazwisko, adres email), ew. czy wprowadzone dane są zgodne z wzorcowymi wyrażeniami regularnymi. Walidacja po stronie klienta jest niewystarczająca, w celu przeprowadzenia zaawansowanych testów konieczne jest przeniesienie części walidacji na stronę serwera. Po stronie serwera walidacja może być przeprowadzona na dwa sposoby. Walidacja oparta na formularzu polega na powiązaniu każdego formularza z kodem walidującym, który może dokonać sprawdzenia złożonych ograniczeń integralnościowych. Takie rozwiązanie jest popularne w wielu architekturach szkieletowych, np. Apache Struts. Wadą jest brak centralizacji i rozbicie walidacji na wiele niezależnych modułów powiązanych z poszczególnymi formularzami. Alternatywą jest walidacja bazująca na typach abstrakcyjnych. Taka walidacja powoduje wyłączenie metadanych i ograniczeń narzuconych na stan modelu z logiki aplikacji oraz centralizację metod walidacji. Metadane i ograniczenia są zapisywane w plikach konfiguracyjnych lub bazie danych, poszczególne procedury obsługi formularzy nie posiadają własnego kodu walidującego lecz korzystają ze wspólnej, ujednoliconej wersji.

Złe praktyki w warstwie prezentacji

- Kod sterujący w wielu widokach
- Udostępnianie struktur danych warstwy prezentacji warstwie biznesowej
- Duplikacja formularzy
- Użycie skryptletów w widoku
- Znacznik <jsp:setProperty>

W trakcie projektowania warstwy prezentacji można popełnić wiele błędów. Najczęściej spotykane błędy i złe praktyki to:

1. Umieszczanie kodu sterującego (kontrola dostępu, walidacja) w postaci skryptletów lub własnych znaczników w wielu widokach: takie rozwiązanie powoduje, że pielęgnacja kodu staje się bardzo uciążliwa
2. Udostępnianie struktur danych warstwy prezentacji bezpośrednio warstwie biznesowej (np. przekazywanie do warstwy biznesowej specyficznych struktur danych warstwy prezentacji takich jak HTTPServletRequest): powoduje to niepotrzebne powiązanie warstwy biznesowej z warstwą prezentacji, niszczy modułowość i uniwersalność warstwy biznesowej, zwiększa liczbę wewnętrznych zależności wewnątrz aplikacji
3. Duplikacja formularzy: brak kontroli nad nawigacją między stronami w przeglądarce, możliwość wielokrotnego przesyłania tego samego formularza
4. Użycie skryptletów w widoku: powoduje nadmierną komplikację kodu, uniemożliwia modułową pracę nad widokiem (podział na osoby programujące HTML i logikę prezentacji), ujawnia logikę implementacji
5. Założenie, że <jsp:setProperty> zresetuje własności komponentu JavaBean: należy pamiętać, że <jsp:setProperty> nie modyfikuje cech komponentu JavaBean, którym odpowiadają parametry o wartości null

Rozwiązaniem powyższych problemów jest przeprowadzenie właściwych refaktoryzacji, opisanych szczegółowo w dalszej części wykładu

Korzystanie z komponentów sesyjnych

- Komponenty sesyjne stanowe i bezstanowe
- Skalowalność i wydajność
- Wybór w zależności od procesu biznesowego
- Przechowywanie stanu sesji

Komponenty sesyjne występują w dwóch wersjach: stanowej i bezstanowej. Bezstanowy komponent sesyjny nie przechowuje żadnej informacji o stanie sesji klienta i po jednorazowym wykonaniu metody wywołanej przez klienta ta sama instancja komponentu może być wykorzystana do obsługi innego żądania innego klienta. Stanowy komponent sesyjny pamięta stan sesji klienta, ta sama instancja komponentu może być wykorzystana do obsługi żądania nowego klienta dopiero po zakończeniu sesji aktualnie obsługiwanego klienta. Komponenty stanowe wymagają więc większej ilości zasobów i mają negatywny wpływ na skalowalność i wydajność aplikacji. Wybór rodzaju wykorzystanego komponentu sesyjnego powinien być jednak podyktowany nie skalowalnością, lecz logiką procesu biznesowego implementowanego przez komponent. Jeśli proces biznesowy jest bezkonwersacyjny (tzn. można zrealizować proces biznesowy za pomocą jednego wywołania metody), wówczas należy użyć komponentów bezstanowych. Jeśli proces biznesowy jest konwersacyjny, wówczas zdecydowanie należy wykorzystać komponenty stanowe. Wykorzystanie w drugim przypadku komponentów bezstanowych powoduje niepotrzebną komplikację aplikacji i wiąże się z koniecznością przesyłania stanu sesji w każdym wywołaniu.

Jeśli aplikacja jest prosta i dostęp do aplikacji zawsze odbywa się poprzez warstwę prezentacji (serwlety i widoki JSP), to wówczas stan sesji można przechowywać w warstwie prezentacji, np. za pomocą obiektu HTTPSession. Jeśli jednak aplikacja obsługuje różne rodzaje klientów HTTP komunikujące się także bezpośrednio z warstwą biznesową, wówczas stan sesji powinien być przechowywany w warstwie biznesowej za pomocą stanowych komponentów sesyjnych.

Korzystanie z komponentów encyjnych

- Komponenty encyjne
  - Obiektowy widok danych
  - Transakcyjność
  - Współdzielenie
  - Odporność na awarie kontenera
- Logika biznesowa w komponentach encyjnych
  - Brak logiki, metody dostępu do danych
  - Logika samowystarczalna obsługi własnych danych

Zgodnie ze specyfikacją EJB, komponenty encyjne są zdefiniowane jako obiektowa reprezentacja trwałych danych przechowywanych w bazie danych. Komponenty encyjne mogą uczestniczyć w transakcjach, są współdzielone przez wielu użytkowników, posiadają długi czas życia i są odporne na awarie kontenera. Komponenty encyjne są zatem współdzielonymi, rozproszonymi i trwałymi obiektami transakcyjnymi. Silne powiązanie komponentów encyjnych z danymi powoduje, że umieszczanie w nich logiki biznesowej staje się co najmniej problematyczne. Popularne jest przekonanie, że komponenty encyjne w ogóle nie powinny zawierać logiki biznesowej, a tylko i wyłącznie metody dostępu do danych (odczyt i zapis). Rzeczywiście, wprowadzenie logiki biznesowej do komponentów encyjnych może spowodować powstanie niepotrzebnych związków między komponentami encyjnymi czy przeniesie odpowiedzialność za zarządzanie przepływem interakcji z użytkownikiem do warstwy biznesowej. Uznaje się, że jeśli komponent encyjny ma zawierać logikę biznesową, powinna być to logika samowystarczalna z punktu widzenia danych przechowywanych przez komponent encyjny, ew. logika obsługująca obiekty zależne od komponentu encyjnego. Występowanie zależności między komponentami encyjnymi wiąże się z koniecznością wprowadzenia dodatkowego komponentu sesyjnego zarządzającego przepływem sterowania między powiązanymi komponentami encyjnymi.

Zaawansowane aplikacje internetowe

Złe praktyki w warstwie biznesowej

- Mapowanie modelu obiektowego na model komponentów encyjnych
- Mapowanie modelu relacyjnego na model komponentów encyjnych
- Korzystanie z komponentów encyjnych tylko do odczytu
- Osadzanie usług wyszukiwania po stronie klienta
- Rozdrabnianie komponentów encyjnych

Projektowanie (10)

Podobnie jak w warstwie prezentacji, w warstwie biznesowej łatwo jest popełnić wiele błędów projektowych. Poniżej przedstawiono najczęściej spotykane złe praktyki programistyczne występujące w warstwie biznesowej lub warstwie integracji.

1. Mapowanie modelu obiektowego aplikacji bezpośrednio na model komponentów encyjnych: odwzorowanie każdej klasy modelu obiektowego aplikacji na komponent encyjny powoduje powstanie zbyt dużej liczby komponentów, nadmierne obciążenie kontenera, wprowadzenie niepotrzebnych zależności między komponentami oraz ogólne zmniejszenie wydajności aplikacji
2. Mapowanie modelu relacyjnego bezpośrednio na model komponentów encyjnych: sposób reprezentacji związków 1:N i M:N (jeden-do-wiele i wiele-do-wiele) w modelu relacyjnym jest nieprzystający do modelu komponentów encyjnych, każdy komponent encyjny powinien reprezentować jedną encję biznesową, a nie jej tablicową reprezentację w bazie danych, takie odwzorowanie gwałtownie zwiększa liczbę komponentów i zależności między nimi, tym samym wpływając negatywnie na wydajność systemu
3. Większość kontenerów EJB nie obsługuje komponentów "tylko do odczytu" i traktuje takie komponenty jak zwykłe komponenty, synchronizując ich stan ze stanem bazy danych, stąd wykorzystywanie komponentów encyjnych jako obiektów tylko do odczytu powoduje marnotrawstwo zasobów
4. Osadzanie usług wyszukiwania po stronie klienta: wyszukiwanie komponentów encyjnych z poziomu aplikacji klienta nie powinno się odbywać za pomocą interfejsu JNDI, ponieważ grozi to umieszczeniem kodu wyszukującego w wielu miejscach aplikacji w warstwie klienta, rozwiązaniem jest wprowadzenie specjalnego mechanizmu lokalizacji zasobów, który zostanie omówiony później.
5. Rozdrabnianie komponentów encyjnych: jak już wcześniej wspomniano, komponenty encyjne reprezentują encje biznesowe, a nie encje występujące w bazie danych czy modelu warstwy prezentacji. Należy unikać niepotrzebnego podziału komponentów encyjnych na mniejsze jednostki.

Zaawansowane aplikacje internetowe

Refaktoryzacja

- Cele refaktoryzacji
  - Uproszczenie pielęgnacji aplikacji
  - Poprawienie modułowości aplikacji
  - Rozdzielenie ról członków zespołu projektowego
  - Wielokrotne wykorzystanie komponentów
  - Zwiększenie bezpieczeństwa
  - Redukcja komunikacji sieciowej

Projektowanie (11)

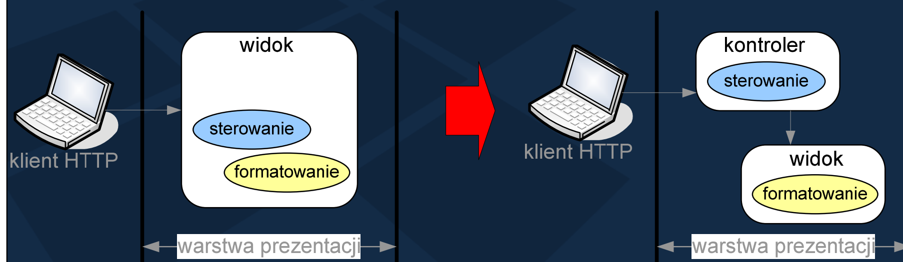
Refaktoryzacja polega na takiej zmianie kodu i architektury aplikacji, aby osiągnąć poprawę pewnych kryteriów (czytelności, efektywności, modułowości, itp.), zachowując jednocześnie własności i sposób zachowania aplikacji. Refaktoryzacja może dotyczyć zarówno etapu projektowania aplikacji, jak i wprowadzania zmian w istniejącym kodzie. Najważniejsze cele refaktoryzacji aplikacji internetowych to:

1. Uproszczenie pielęgnacji aplikacji poprzez zmniejszenie liczby zależności między komponentami aplikacji oraz uproszczenie, ujednolicenie i scentralizowanie kodu
2. Poprawienie modułowości poprzez identyfikację wspólnych modułów i ich scalenie
3. Rozdzielenie ról członków zespołu projektowego poprzez podział aplikacji na wyraźnie zarysowane moduły odpowiedzialne za obsługę wyznaczonych fragmentów działania aplikacji (generowanie widoku, obsługa bazy danych, lokalizacja usług, obsługa błędów i wyjątków, itp.)
4. Wielokrotne wykorzystanie komponentów w celu uniknięcia duplikowania tego samego kodu
5. Zwiększenie bezpieczeństwa przez ujednolicenie procedur kontroli dostępu, deklaratywne zarządzanie uprawnieniami dostępu, czy obsługę transakcyjności
6. Redukcja komunikacji sieciowej przez scalanie komponentów i tworzenie obiektów do transferu danych

Zaawansowane aplikacje internetowe

Wprowadzenie kontrolera

- Logika sterująca w jednej klasie sterującej która stanowi początkowe miejsce obsługi żądań klienta



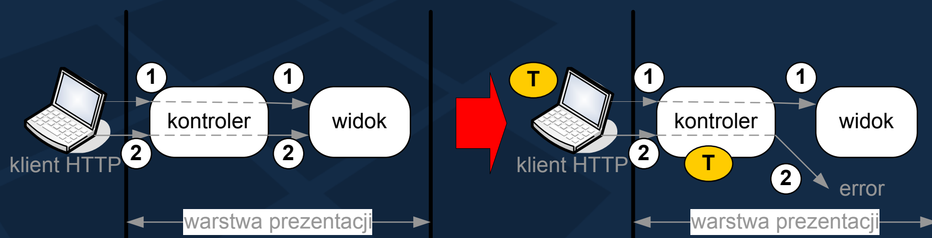
Projektowanie (12)

Jeśli logika kontroli przepływu sterowania jest rozproszona po wszystkich składowych aplikacji i dostęp do każdego widoku jest administrowany bezpośrednio w widoku (tzn. jeśli widoki posiadają zarówno kod formatujący wynik jak i kod sterujący), wówczas pielęgnacja takiej aplikacji bardzo się komplikuje. Rozwiązaniem jest wprowadzenie pomocniczego kontrolera (może być to jedna lub wiele klas), który stanowi początkowy punkt obsługi wszystkich żądań pochodzących od klientów. Dla każdego żądania kontroler przeprowadza konieczne operacje sterujące i przekierowuje żądanie do właściwego widoku. Takie rozwiązanie nadaje się najbardziej do ochrony zawartości typu "wszystko albo nic", ochrona fragmentów widoku z poziomu kontrolera sterującego jest utrudniona. Kontroler powinien być wprowadzony praktycznie we wszystkich, z wyjątkiem bardzo małych, aplikacji internetowych.

Zaawansowane aplikacje internetowe

Wprowadzenie tokenu synchronizującego

- Współdzielony token monitoruje i steruje przepływem żądań i dostępem klientów do zasobów, wymuszając określoną kolejność żądań

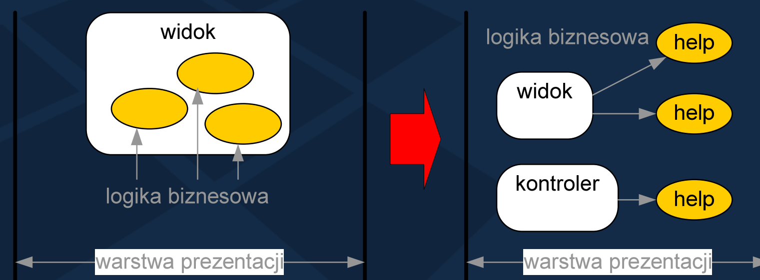


Projektowanie (13)

Klient HTTP może duplikować żądania poprzez wielokrotne wysłanie tego samego formularza lub może dokonywać nielegalnych prób bezpośredniego dostępu do zasobów z pominięciem ścieżki nawigacji. Praktycznie zawsze jesteśmy zainteresowani kontrolowaniem nadchodzących żądań, choćby w celu uniknięcia powielenia żądań klienta. Rozwiązaniem może być utworzenie pomocniczej klasy odpowiedzialnej za generowanie i porównywania unikalnych tokenów jednorazowego użytku. Po stronie aplikacji należy dodać logikę sprawdzającą, czy token przesłany przez użytkownika jest identyczny z tokenem zapamiętanym w sesji. W trakcie każdego żądania token przesłany przez klienta powinien być taki sam jak token przesłany klientowi przez serwer w poprzedniej odpowiedzi. Brak zgodności tokenów może wynikać z wielokrotnego przesłania formularza (w trakcie drugiego wysłania serwer oczekuje już na inny token) lub z powodu dostępu do zasobu przez niepoprawną ścieżkę (np. poprzez zakładkę w przeglądarce). Brak zgodności tokenów powinien być obsłużony za pomocą właściwej strony z komunikatem o zaistniałym błędzie. Unikalne tokeny są za zwyczaj generowane przez kontroler, choć można również dodać odpowiednią funkcjonalność bezpośrednio w widokach.

Podział logiki na niezależne fragmenty

- Logika biznesowa z widoków przeniesiona do jednej lub kilku klas pomocniczych, które są wykorzystywane przez widoki JSP lub serwlety



Jeśli widoki JSP zawierają zarówno kod formatowania jak i logikę biznesową, konieczne staje się wydzielenie logiki i podział logiki na niezależne fragmenty realizujące poszczególne procesy biznesowe. Każdy fragment jest zaimplementowany w postaci klasy pomocniczej, która może być wykorzystywana tak przez kontroler, jak i widok (np. jako komponent JavaBean lub element własnego znacznika). Takie rozwiązanie poprawia modułowość aplikacji, umożliwia podział zadań wewnątrz zespołu projektowego oraz ułatwia pielęgnację aplikacji.

Ukrycie szczegółów warstwy prezentacji przed warstwą biznesową

- Wszystkie odwołania do struktur danych, klas i protokołów komunikacyjnych w warstwie biznesowej muszą być usunięte, dane między warstwami przesyła się za pomocą struktur ogólnych

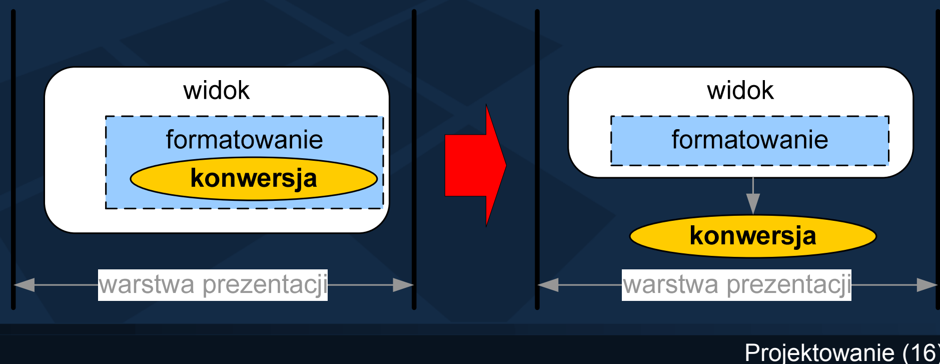
komponent prezentacyjny
HTTPServletRequest
komponent biznesowy
HTTPServletRequest
warstwa prezentacji
warstwa biznesowa

komponent prezentacyjny
HTTPServletRequest
komponent biznesowy
Informacje o żądaniu
warstwa prezentacji
warstwa biznesowa

Istotnym błędem projektowym jest prezentowanie szczegółów implementacyjnych jednej warstwy innym warstwom. Związanie usług w warstwie biznesowej z konkretnymi strukturami danych i protokołami warstwy prezentacji powoduje zmniejszenie zakresu wykorzystywania danej usługi biznesowej i ograniczenie jej tylko do wybranych klientów. Przykładowo, jeśli komponent warstwy biznesowej zostanie powiązany z komponentem prezentacyjnym przez parametr określonego wąskiego typu (np. HTTPServletRequest), to dany komponent biznesowy będzie dostępny tylko i wyłącznie z poziomu serwletu. Komunikacja danych między warstwami nie może wykorzystywać struktur specyficznych dla określonego typu klienta. W celu przekazania danych między warstwami posługujemy się strukturami ogólnymi, które nie są związane ściśle z żadną warstwą.

Usunięcie konwersji z widoku

- Cały kod konwersji musi być przeniesiony z widoku do klasy pomocniczej

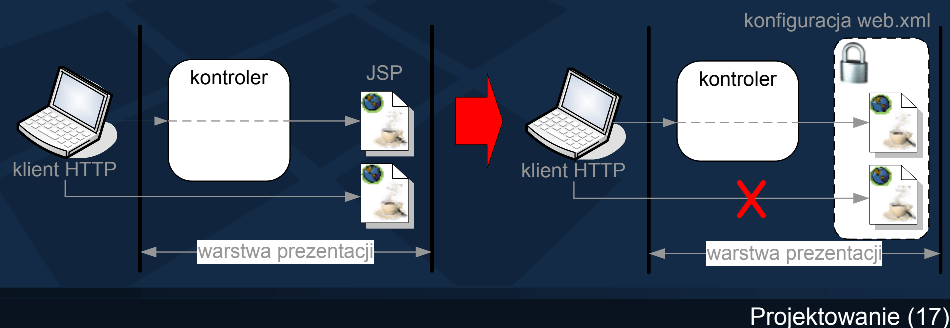


Zdarza się, że widok JSP zawiera kod konwersji odpowiedzialny za transformację modelu do postaci właściwej dla możliwości klienta (ekran komputera, ekran PDA, telefon komórkowy, itp.). Umieszczenie kodu konwersji bezpośrednio w widoku zmniejsza modułowość aplikacji i uniemożliwia wykorzystanie tego samego widoku na potrzeby innych klientów. Preferowanym rozwiązaniem jest wyłączenie kodu konwersji w postaci klasy pomocniczej i wywoływanie tego kodu po sformatowaniu danych wyjściowych.

Zaawansowane aplikacje internetowe

Ukrywanie zasobów przed klientem (1)

- Najbardziej istotne zasoby powinny być ukryte i chronione przed użytkownikiem poprzez konfigurację kontenera



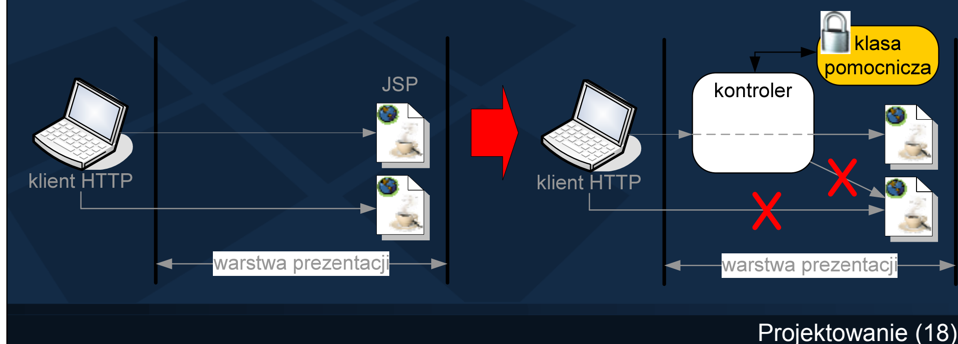
Projektowanie (17)

W przypadku, gdy dostęp do niektórych zasobów, np. wybranych widoków JSP, powinien być chroniony, należy zastosować ukrywanie zasobów przed klientem. Jeśli nie jest stosowane żadne ukrywanie zasobów, to każdy zasób jest dostępny z klienta zarówno przez kontroler, jak i bezpośrednio (np. na podstawie adresu zapisanego w zakładce). Wprowadzenie deklaratywnej kontroli dostępu w pliku web.xml powoduje, że bezpośredni dostęp do zasobu, z pominięciem kontrolera, przestaje być możliwy. Najpopularniejszym i najprostszym sposobem implementacji tej propozycji jest umieszczenie chronionych widoków w pliku WEB-INF/, dostęp do nich może się odbywać tylko przez kontroler. Innym rozwiązaniem jest wprowadzenie pomocniczej klasy zawierającej definicje ograniczeń. Wówczas przepływem sterowania kieruje kontroler odczytujący zasady zabezpieczeń z komponentu JavaBean.

Zaawansowane aplikacje internetowe

Ukrywanie zasobów przed klientem (2)

- Zasoby mogą być chronione przez kontroler i klasę pomocniczą zawierającą definicje ograniczeń



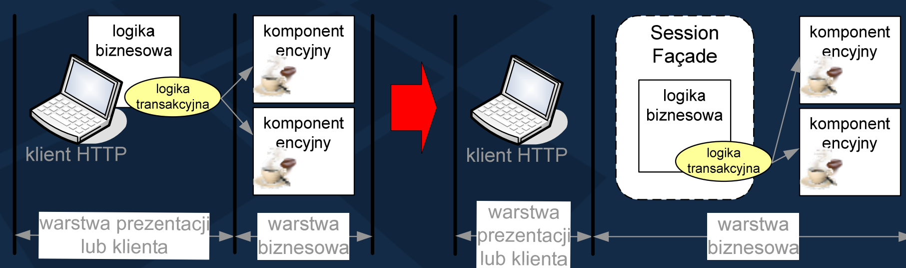
Projektowanie (18)

Alternatywnym sposobem ukrywania zasobów przed klientem jest wprowadzenie kontrolera i dodatkowej klasy pomocniczej zawierającej definicje ograniczeń dostępu. Wówczas kontroler kieruje przepływem sterowania, odczytując zasady dostępu z klasy pomocniczej. Takie rozwiązanie pozwala na bardziej elastyczną konfigurację kontroli dostępu.

Zaawansowane aplikacje internetowe

Ukrycie komponentów encyjnych

- Jeżeli komponenty encyjne w warstwie biznesowej są udostępniane klientom w innych warstwach, to należy je ukryć za komponentami sesyjnymi



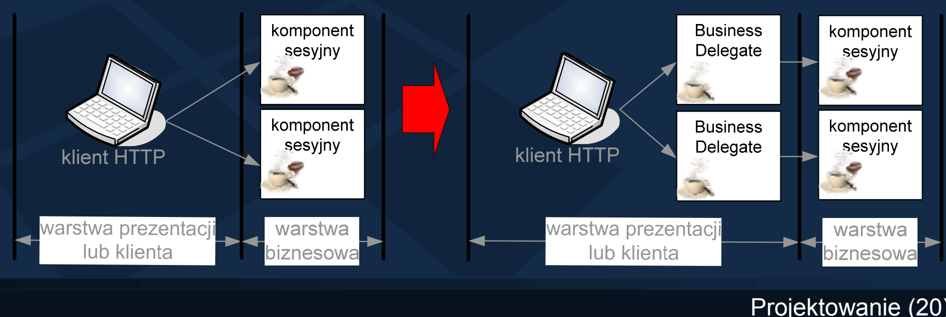
Projektowanie (19)

Bezpośrednie udostępnienie komponentów encyjnych obiektom innych warstw wprowadza duży narzut na komunikację sieciową, ponieważ każde wywołanie dotyczące komponentu encyjnego powoduje zdalne wołanie metody. Skutkuje to wyraźnym spadkiem wydajności aplikacji. Dodatkowo, jeśli logika biznesowa, a w szczególności logika transakcyjna, są implementowane po stronie klienta, osoba tworząca warstwę klienta lub warstwę prezentacji musi szczegółowo zrozumieć, zaprojektować i zarządzać transakcjami dotyczącymi wielu komponentów encyjnych. Powoduje to nadmierną komplikację kodu po stronie klienta (wyszukanie komponentów, pobranie interfejsu transakcji, operacje w ramach kontekstu transakcji) i niemożność wykorzystania zalet zarządzania transakcjami przez kontener. Rozwiązaniem problemu jest przeniesienie logiki biznesowej i logiki transakcyjnej do warstwy biznesowej pod postacią wzorca Session Façade (wzorzec zostanie szczegółowo omówiony w trakcie wykładu dotyczącego wzorców J2EE). Wzorzec Session Façade jest to komponent sesyjny zarządzający logiką biznesową i dostępem oraz współdziałaniem między komponentami encyjnymi. Logika transakcyjna może być zaimplementowana przez komponent sesyjny, a może być także przeniesiona bezpośrednio do kontenera, gdzie deklaratywne zarządzanie transakcjami definiuje się w deskryptorze wdrożenia. W obu przypadkach zdejmuje to z klienta obowiązek sterowania przebiegiem transakcji.

Zaawansowane aplikacje internetowe

Wprowadzenie obiektów Business Delegate

- Komponenty sesyjne warstwy biznesowej powinny być dostępne w innych warstwach poprzez obiekty Business Delegate



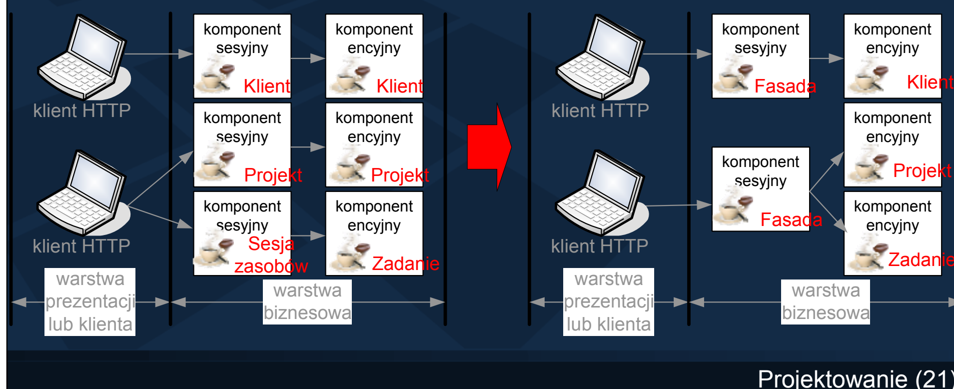
Projektowanie (20)

Komponenty sesyjne w warstwie biznesowej są używane przede wszystkim do ukrywania komponentów encyjnych. Jednak udostępnienie komponentów sesyjnych warstwy biznesowej bezpośrednio klientom jest złym rozwiązaniem, ponieważ silnie wiąże klientów z interfejsem oferowanym przez komponent sesyjny oraz wymusza na klientach obsługę wyjątków specyficznych dla komponentów sesyjnych. Takie rozwiązanie źle wpływa na elastyczność aplikacji i utrudnia wykorzystanie heterogenicznych klientów. W celu rozwiązania problemu należy wprowadzić obiekty Business Delegate jako obiekty przesłaniające dostęp do komponentów sesyjnych. Obiekt Business Delegate to zwykły obiekt Java (POJO) hermetyzujący dostęp do interfejsu komponentu sesyjnego oraz obsługujący wyjątki charakterystyczne dla komponentu sesyjnego. Każdy obiekt Business Delegate obsługuje jeden komponent sesyjny. Zysk polega na uniezależnieniu klienta od zmian w interfejsie komponentu sesyjnego oraz przeniesienie kodu odpowiedzialnego za lokalizację komponentów (JNDI) i kodu odpowiedzialnego za buforowanie do obiektów Business Delegate.

Zaawansowane aplikacje internetowe

Łączenie komponentów sesyjnych

- Komponenty sesyjne reprezentują usługi biznesowe i nie odwzorowują komponentów encyjnych 1:1

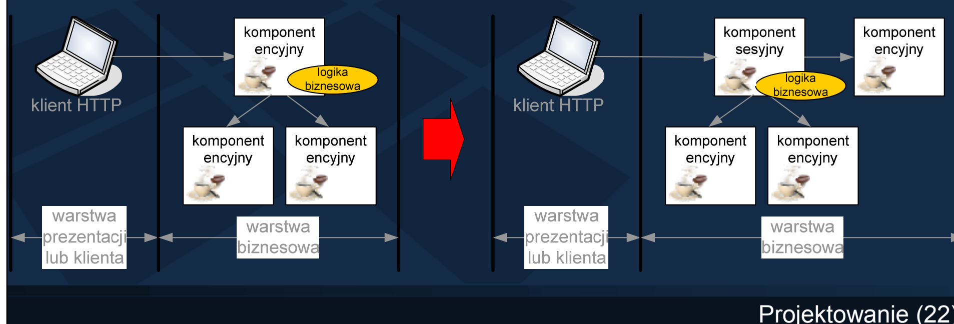


Projektowanie (21)

Jedno z najpoważniejszych nieporozumień związanych z projektową zasadą ukrywania komponentów encyjnych za fasadą komponentów sesyjnych polega na tym, że wiele osób rozumie tę zasadę jako nakaz odwzorowania każdego komponentu encyjnego na jeden komponent sesyjny. W rzeczywistości takie rozwiązanie przynosi niewiele korzyści, w zasadzie wprowadza tylko dodatkową warstwę pośredniczącą. Komponenty sesyjne tworzą fasady reprezentujące konkretne usługi biznesowe, a nie komponenty encyjne. Jeśli jakiś proces biznesowy wykorzystuje dwa komponenty encyjne (przykładowo, zarządzanie zadaniami szczegółowymi przypisanymi do projektu), to procesowi biznesowemu powinien odpowiadać jeden komponent sesyjny stanowiący fasadę dla obu komponentów encyjnych: komponentu reprezentującego projekty i komponentu reprezentującego zadania. Wszystkie komponenty sesyjne które pełnią tylko rolę pośredników dla komponentów encyjnych powinny zostać połączone w większe komponenty reprezentujące usługi biznesowe. Dzięki temu warstwa biznesowa jest widoczna dla klientów przez pryzmat interfejsu definiowanego przez komponenty sesyjne, a te z kolei są niezależne od modelu danych i reprezentują oferowane usługi biznesowe.

Logika biznesowa w komponentach sesyjnych

- Logika biznesowa wyrażająca relacje między komponentami encyjnymi jest przeniesiona do komponentu sesyjnego



Logika biznesowa komponentu encyjnego operująca na innych komponentach encyjnych wprowadza dodatkowe zależności między komponentami. Może prowadzić to do nadmiernej komplikacji komponentów encyjnych (które z definicji powinny być proste). Dobrym rozwiązaniem projektowym jest wprowadzenie komponentu sesyjnego, do którego można przenieść logikę biznesową wykorzystującą wiele komponentów encyjnych. Wiąże się to bezpośrednio z poprzednim zaleceniem, dotyczącym łączenia komponentów sesyjnych. Wprowadzony komponent sesyjny funkcjonuje jako fasada dla komponentów encyjnych, centralizuje i upraszcza logikę biznesową, oraz oferuje klientom spójny interfejs dostępu do usług biznesowych.

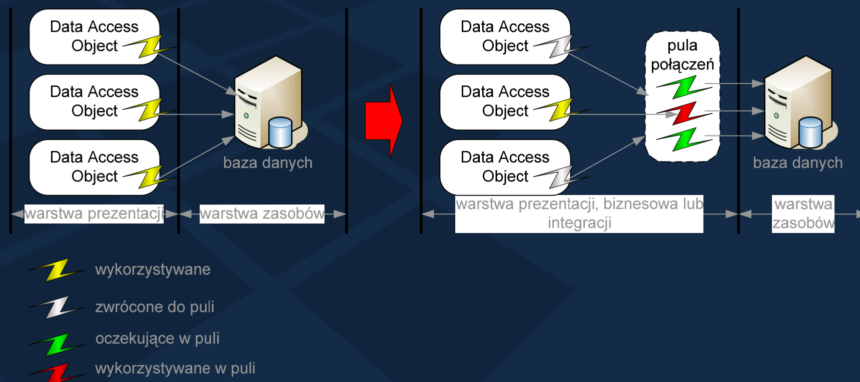
Zaawansowane aplikacje internetowe

Wydzielenie kodu dostępu do danych

- Cały kod dostępu do danych znajduje się w wydzielonej klasie pomocniczej zlokalizowanej logicznie i fizycznie blisko źródła danych

Projektowanie (23)

Rozwiązanie, w którym klasa sterująca, poza swoją normalną funkcjonalnością, jest także odpowiedzialna za nawiązywanie połączenia i komunikację z bazą danych, może stwarzać wiele problemów. Wprowadza niepotrzebne zależności między warstwami aplikacji, uzależnia klasy sterujące od zmian w warstwie zasobów, zmniejsza modułowość aplikacji i wymaga duplikacji kodu dostępu do danych we wszystkich klasach sterujących. Dobrym rozwiązaniem jest wydzielenie kodu odpowiedzialnego za dostęp do danych i umieszczenie tego kodu w wydzielonej klasie. W ten sposób hermetyzuje się kod dostępu do danych (najczęściej JDBC) i uniezależnia warstwę prezentacji lub biznesową od szczegółów implementacyjnych połączeń z bazą danych. Wydzielenie kodu dostępu do danych znajduje zastosowanie zarówno w warstwie prezentacji (np. wyłączenie kodu JDBC z serwletu pełniącego rolę kontrolera), jak i w warstwie biznesowej (np. przeniesienie kodu odpowiedzialnego za logikę trwałości danych z komponentu encyjnego do osobnej klasy). Rozwiązanie to powoduje wzrost elastyczności i modułowości aplikacji oraz umożliwia łatwą pielęgnację kodu w odpowiedzi na zmiany zachodzące w warstwie zasobów.



Ze względu na koszt związany z otwarciem i nawiązaniem połączenia z bazą danych w wielu sytuacjach opłaca się wykorzystywać pule połączeń. Unika się wówczas wielu niekorzystnych zjawisk, takich jak utrata skalowalności i wydajności czy wyczerpanie liczby dostępnych połączeń (w sytuacji gdy każdy klient trzyma własne połączenie). Rozwiązaniem tych problemów jest wprowadzenie puli połączeń, która zawiera pewną liczbę wcześniej zainicjalizowanych połączeń. Pula połączeń powinna oferować interfejs do pobierania i zwracania połączeń zgodnie z aktualnymi potrzebami klientów. Stosowanie JDBC 2.0 i interfejsu javax.sql.DataSource umożliwia skorzystanie z gotowej puli połączeń zaimplementowanej w ramach standardu JDBC.