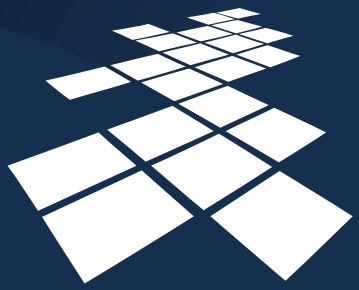
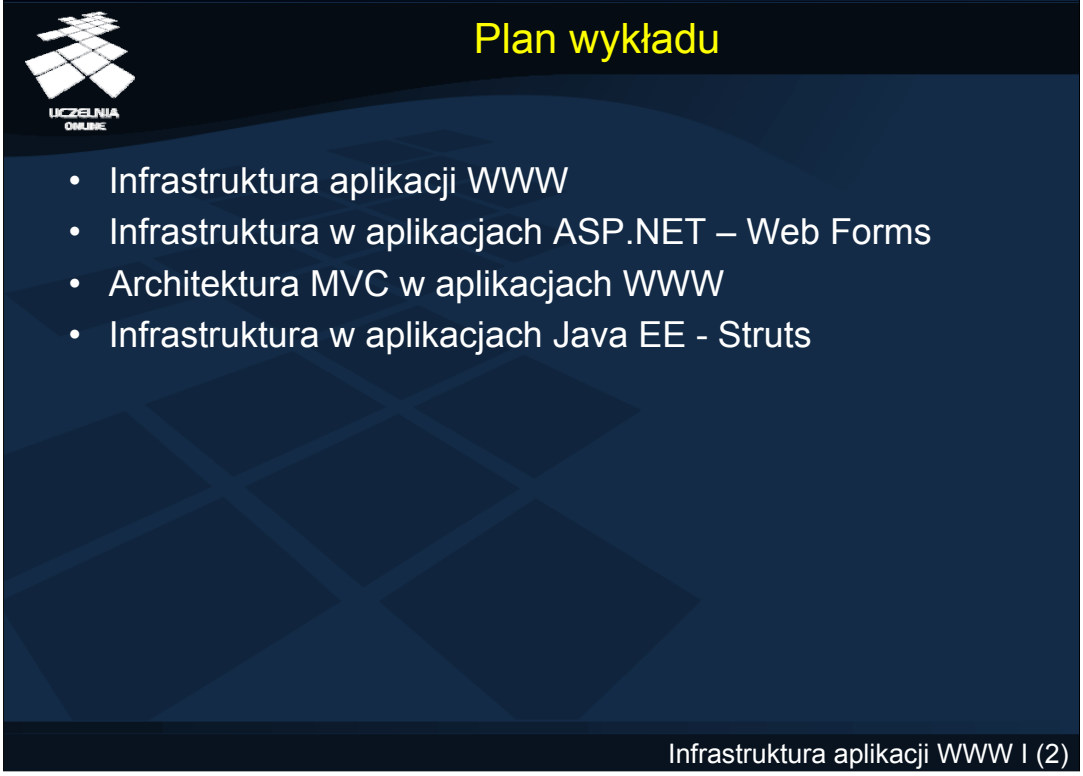
Infrastruktura aplikacji WWW I

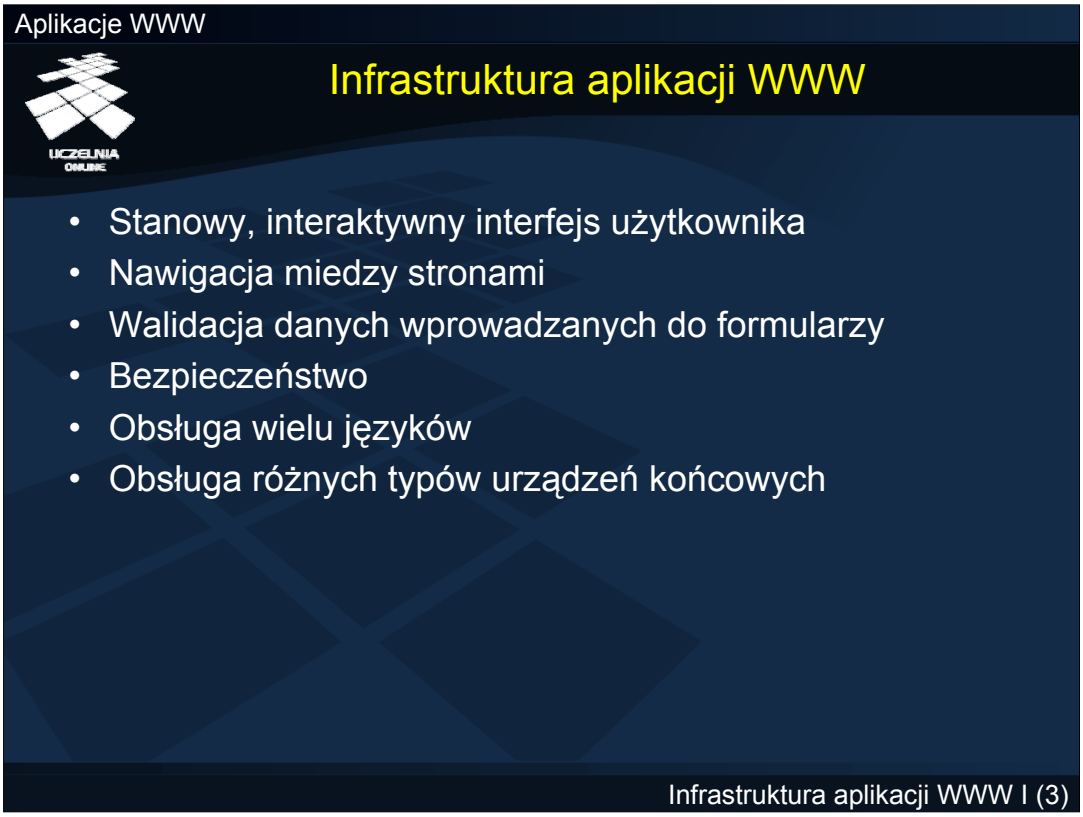
Wykład prowadzi:
Marek Wojciechowski



UCZELNIA
ONLINE



Celem wykładu jest przedstawienie typowej funkcjonalności wymaganej w aplikacjach internetowych, powtarzającej się niezależnie od konkretnych zastosowań i określanej mianem infrastruktury. Po ogólnym wprowadzeniu, przedstawiona będzie architektura Web Forms dla ASP.NET oraz architektura Model-View-Controller i jej popularna implementacja dla platformy Java EE – Struts. Najważniejsza, bo stanowiąca część specyfikacji firmy Sun, implementacja MVC dla Java EE zostanie omówiona w ramach następnego wykładu.



Oprócz implementacji logiki aplikacji specyficznej dla konkretnego zastosowania, twórcy aplikacji WWW muszą rozwiązać szereg typowych problemów, które powtarzają się w różnych aplikacjach WWW niezależnie od ich przeznaczenia. Te problemy to:

(a) zapewnienie stanowego, interaktywnego interfejsu użytkownika na gruncie bezstanowego protokołu HTTP i języka HTML;

(b) nawigacja między stronami, uwzględniająca automatyczne powroty do formularzy wprowadzania danych w przypadku błędów walidacji i zdefiniowana w sposób umożliwiający łatwe zmiany przepływu sterowania w aplikacji;

(c) walidacja danych wprowadzanych do formularzy;

(d) bezpieczeństwo, rozumiane jako kontrola dostępu do stron aplikacji w oparciu o uwierzytelnianie i autoryzację użytkowników;

(e) obsługa wielu języków, najlepiej z możliwością internacjonalizacji aplikacji i dodawania wsparcia dla kolejnych języków bez konieczności rekompilacji kodu;

(f) obsługa różnych typów urządzeń końcowych, a więc nie tylko komputerów, ale również urządzeń o mniejszym ekranie, typu PDA czy telefony.

Implementacja od podstaw infrastruktury aplikacji WWW wymaga wielu linii kodu i może zająć nawet więcej czasu w procesie tworzenia i testowania aplikacji niż implementacja specyficznej dla aplikacji logiki biznesowej. Aby zwiększyć produktywność twórców aplikacji WWW, zaproponowano gotowe rozwiązania w zakresie infrastruktury aplikacji dla poszczególnych technologii, w formie rozwiązań architektonicznych i szkieletów aplikacji. W ramach wykładu zostaną przedstawione rozwiązania w tym zakresie dla dwóch najważniejszych platform: Microsoft .NET i Java EE. Należy zwrócić uwagę, że poszczególne rozwiązania różnią się zakresem wsparcia i najczęściej ograniczają się do niektórych obszarów funkcjonalnych infrastruktury.

Na platformie .NET infrastruktura aplikacji jest zapewniana przez samą technologię ASP.NET z jej architekturą Web Forms. Inaczej jest w przypadku platformy Java EE, gdzie twórcy aplikacji mają do wyboru kilka tzw. szkieletów aplikacji (ang. frameworks), opartych o wzorzec projektowy Model-View-Controller. Najważniejsze z nich to Struts i JavaServer Faces (JSF). Struts pojawił się jako pierwszy, szybko zyskał popularność i stał się standardem de facto. JSF jest nowszy, widoczne są w nim wpływy zarówno Struts jak i Web Forms. JSF ma status oficjalnej specyfikacji składowej Java EE.

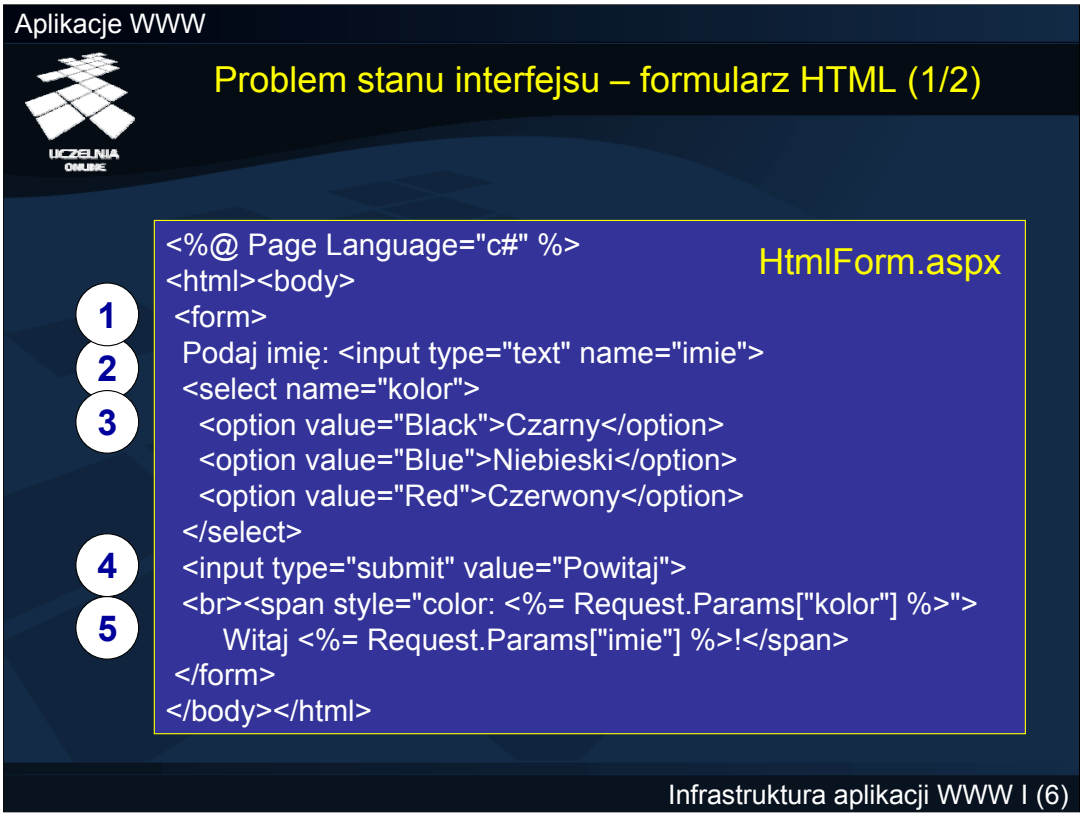
ASP.NET i Web Forms

- Nacisk na stanowy i interaktywny interfejs aplikacji na wzór aplikacji desktopowych
  - integracja z Visual Studio .NET
- Wsparcie dla:
  - walidacji danych
  - bezpieczeństwa
  - obsługi urządzeń mobilnych
  - nawigacji
  - internacjonalizacji aplikacji
  - zapewniania spójnego wyglądu stron i personalizacji
- Nie promuje wzorca Model-View-Controller (MVC)

Najważniejszą cechą ASP.NET, a w szczególności kluczowej dla niej koncepcji Web Forms, jest nacisk na stanowy i interaktywny interfejs aplikacji na wzór aplikacji desktopowych. Ważna jest w tym zakresie integracja ze środowiskiem do tworzenia aplikacji Visual Studio .NET, które umożliwia tworzenie interfejsu dla aplikacji WWW i oprogramowanie zdarzeń generowanych przez kontrolki interfejsu w sposób zbliżony do tworzenia aplikacji desktopowych.

Poza zapewnieniem stanowego interfejsu użytkownika, ASP.NET oferuje gotowe rozwiązania dla walidacji danych (dostarczając kilka gotowych kontrolek walidacyjnych), bezpieczeństwa (współpracując z trzema mechanizmami uwierzytelniania) i obsługi urządzeń mobilnych (poprzez ASP.NET mobile controls). Od wersji ASP.NET 2.0 oferowane jest również wsparcie dla deklaratywnego definiowania nawigacji w oparciu o hierarchiczną mapę serwisu (witryny), internacjonalizacji aplikacji w oparciu o pliki zasobów, a także zapewniania spójnego wyglądu i organizacji stron serwisu i personalizacji.

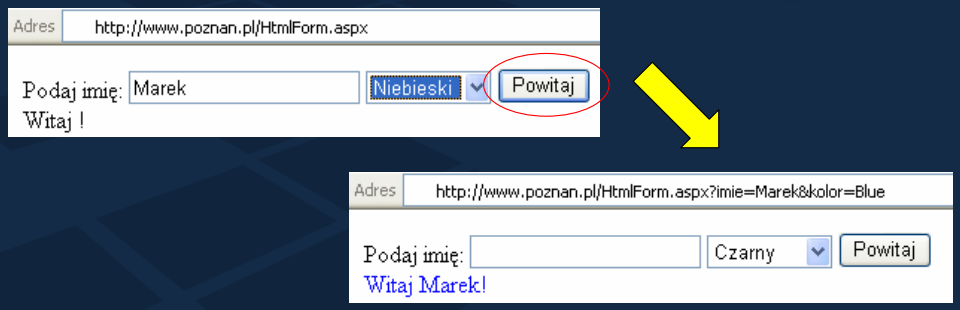
ASP.NET nie promuje wzorca Model-View-Controller (MVC) w takim zakresie jak Java EE, choć implementacja MVC jest możliwa, a jej sposób opisany w dokumentacji ASP.NET. W przypadku implementacji MVC dla ASP.NET w zakresie implementacji kontrolera bardziej typowy jest wzorzec projektowy Page Controller, z obsługą nawigacji w pliku Code Behind związanym ze stroną. Możliwa jest jednak również implementacja wzorca Front Controller, charakterystycznego dla implementacji MVC dla Java EE.



Na slajdzie przedstawiono kod strony ASP.NET wyświetlającej tradycyjny formularz HTML, zbudowany ze znaczników HTML, bez wykorzystania kontrolek ASP.NET. Znaczenie wyróżnionych fragmentów kodu jest następujące:

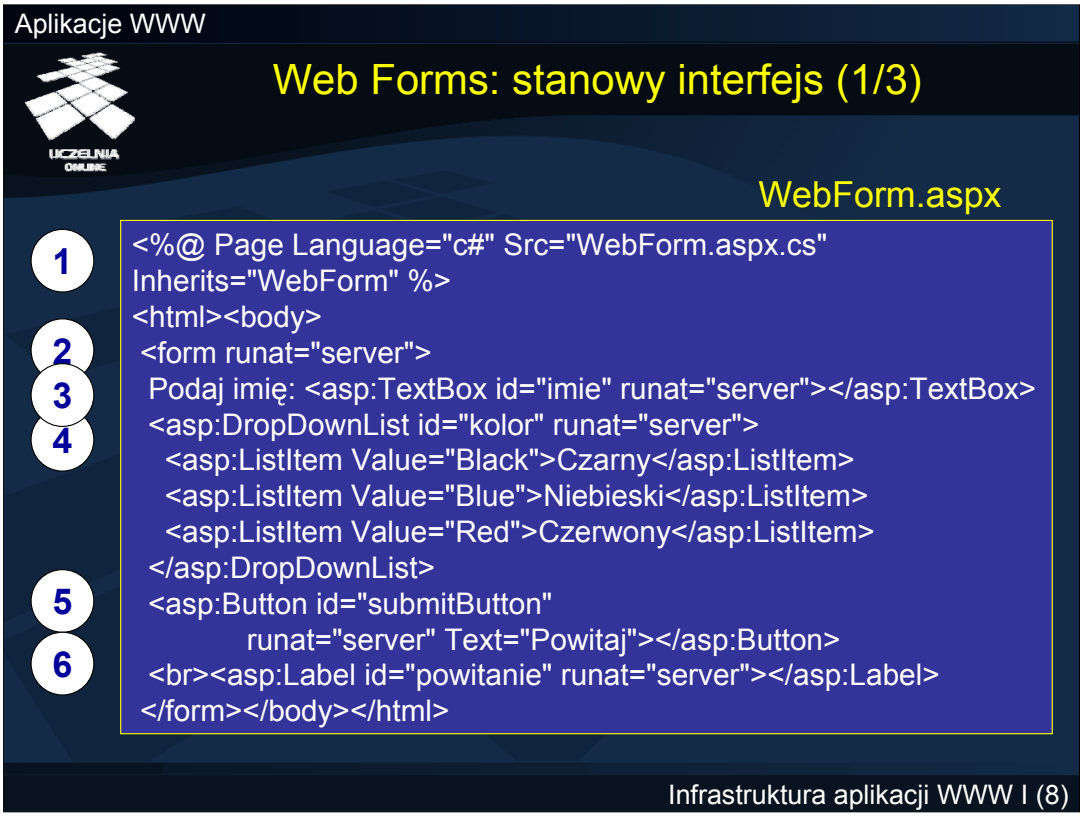
1. Formularz HTML wywołujący bieżącą stronę (brak atrybutu ACTION).
2. Pole tekstowe do wprowadzenia imienia
3. Lista rozwijana zawierająca kolory do wyboru.
4. Przycisk zatwierdzający formularz
5. Tekst powitalny zawierający wstawiane dynamicznie za pomocą konstrukcji ASP.NET ustawienie koloru za pomocą stylu oraz imię. Zarówno kolor jak i imię, przekazane jako parametry z formularza, są odczytywane z obiektu Request poprzez zawartą w nim tablicę asocjacyjną Params.

Problem stanu interfejsu – formularz HTML (2/2)



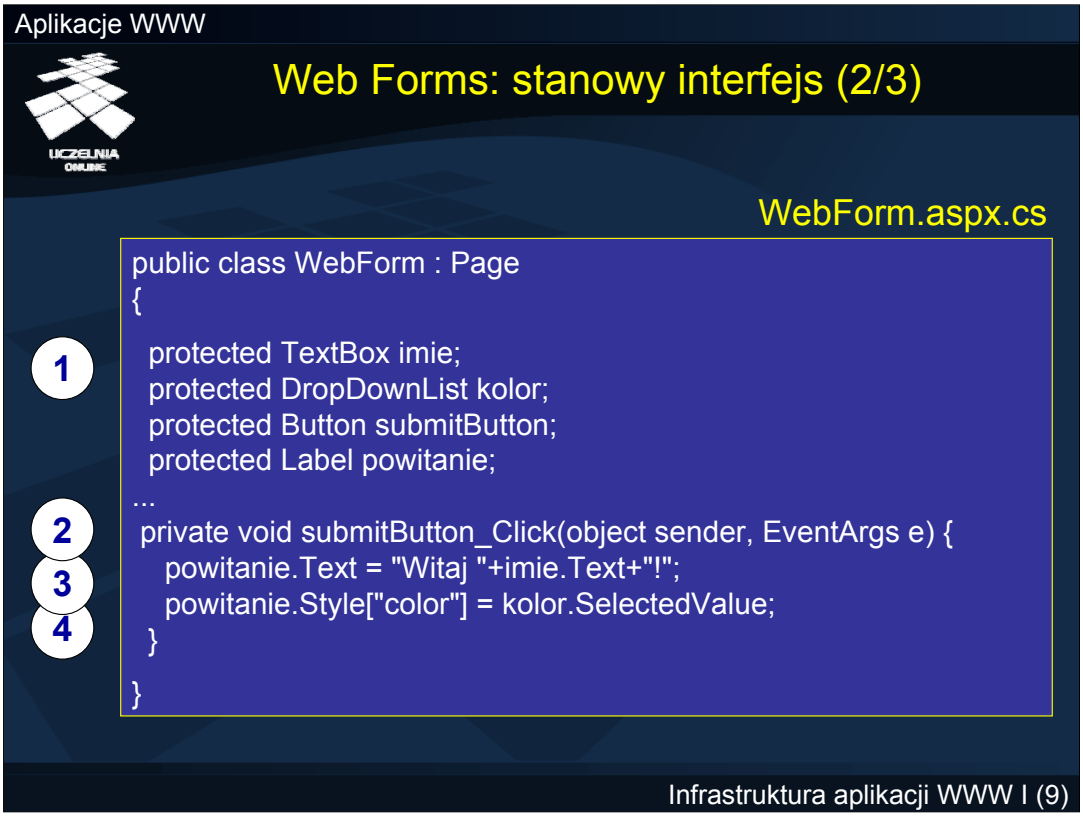
- Formularze HTML są bezstanowe
  - kolejne wywołania tej samej strony obsługiwane niezależnie
  - problem szczególnie dotkliwy przy walidacji danych

Slajd pokazuje efekt działania aplikacji ASP.NET, zawierającej klasyczny formularz HTML. Użytkownik wprowadza imię, wybiera kolor, a następnie zatwierdza formularz przyciskiem „Powitaj”. Formularz wywołuje sam siebie. Po jego ponownym wyświetleniu, widać że parametry z formularza zostały poprawnie przekazane i odebrane – zmieniła się treść i kolor tekstu powitalnego. Niestety wybór dokonany przez użytkownika w elementach formularza (polu tekstowym i liście rozwijanej) nie został zachowany. Dzieje się tak dlatego, że formularze HTML utworzone za pomocą „zwykłych” znaczników HTML są bezstanowe. Kolejne wywołania tej samej strony są obsługiwane przez serwer jako niezależne żądania. Zawartość pól formularza nie jest automatycznie odtwarzana przez serwer na podstawie przekazanych parametrów. Problem ten jest szczególnie dotkliwy gdy dane z formularza są walidowane i formularz jest wyświetlany ponownie w celu umożliwienia użytkownikowi poprawienia danych. W związku z niezapamiętaniem stanu formularza, użytkownik będzie musiał wypełnić wszystkie pola od początku.



Slajd pokazuje wersję aplikacji pozdrawiającej użytkownika, w której formularz został zdefiniowany jako Web Form, zbudowany z kontrolek ASP.NET (HTML i Web Controls). Znaczenie wyróżnionych fragmentów kodu jest następujące:

1. Strona wykorzystuje kontrolki ASP.NET, dla których kod obsługi zdarzeń został umieszczony w pliku Code Behind.
2. Formularz uruchamiany na serwerze.
3. Kontrolka ASP.NET typu pole tekstowe, do wprowadzenia imienia.
4. Kontrolka ASP.NET typu lista rozwijana, do wyboru koloru.
5. Kontrolka ASP.NET typu przycisk.
6. Kontrolka ASP.NET typu etykieta. W niej zostanie programowo umieszczony tekst powitalny.



Slajd pokazuje kod Code Behind obsługujący stronę ASP.NET, której źródło zostało przedstawione na poprzednim slajdzie. Pominięte zostały fragmenty kodu importujące przestrzenie nazw i kod inicjalizujący wskazujący procedury obsługi zdarzeń.

Znaczenie wyróżnionych fragmentów kodu jest następujące:

1. Definicje składowych klasy reprezentujących kontrolki ze strony ASP.NET, którą obsługuje dany kod Code Behind.

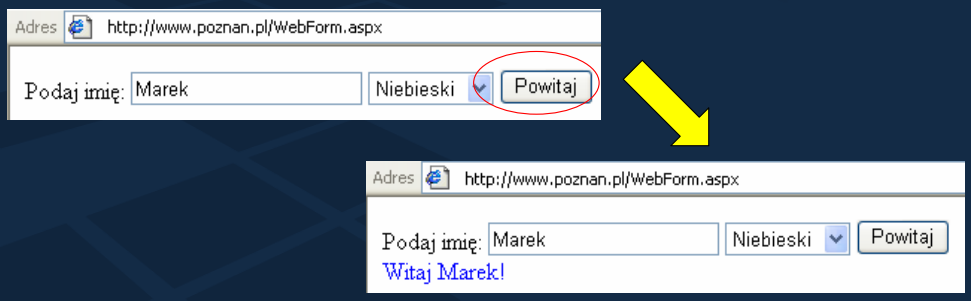
2. Metoda wskazana jako procedura obsługi zdarzenia Click przycisku submitButton.

3. Ustawienie zawartości tekstowej etykiety poprzez jej właściwość Text.

4. Ustawienie koloru etykiety poprzez jej właściwość Style, reprezentującą styl CSS elementu.

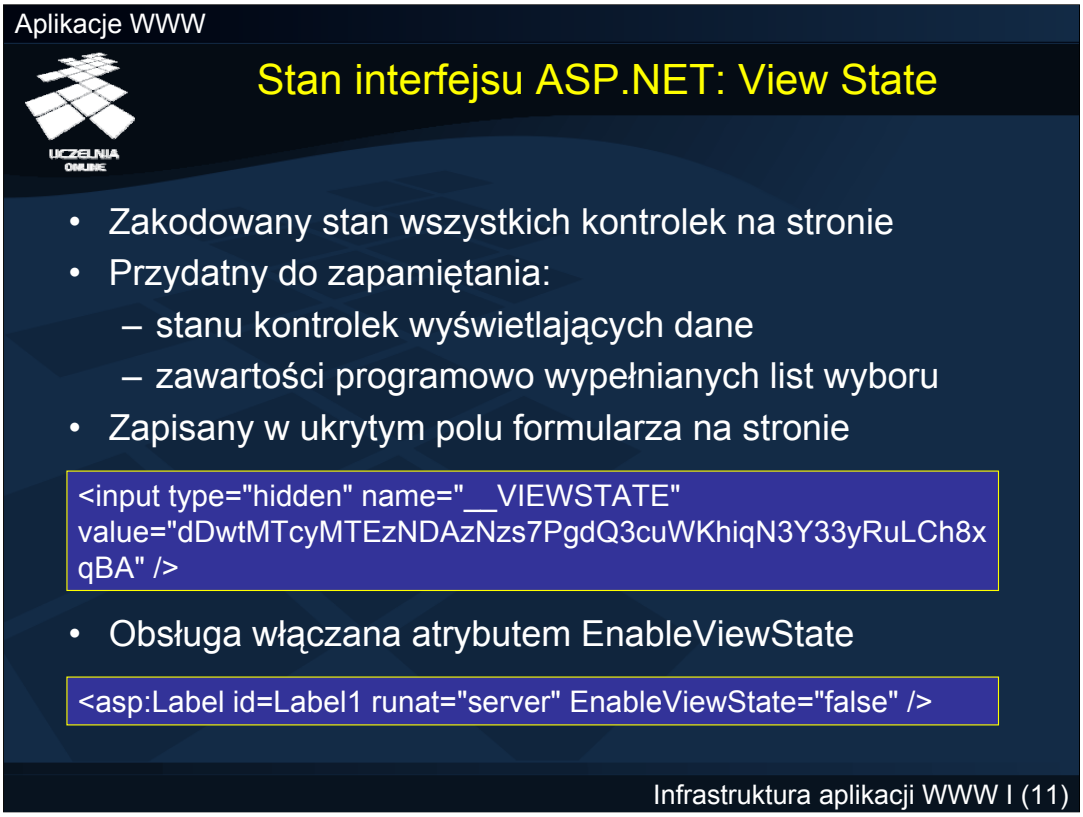
Należy zwrócić uwagę, że w tym wypadku podane przez użytkownika imię i kolor nie są odczytywane z obiektu Request (mimo że taka możliwość nadal istnieje), ale z kontrolek formularza Web Forms, do których zostały wprowadzone przy poprzednim wyświetleniu formularza. Jest to zalecany sposób dostępu do danych z formularzy, możliwy dzięki temu, że dla formularzy Web Forms ASP.NET automatycznie pamięta i odtwarza wartości do formularza wprowadzone.

Web Forms: stanowy interfejs (3/3)



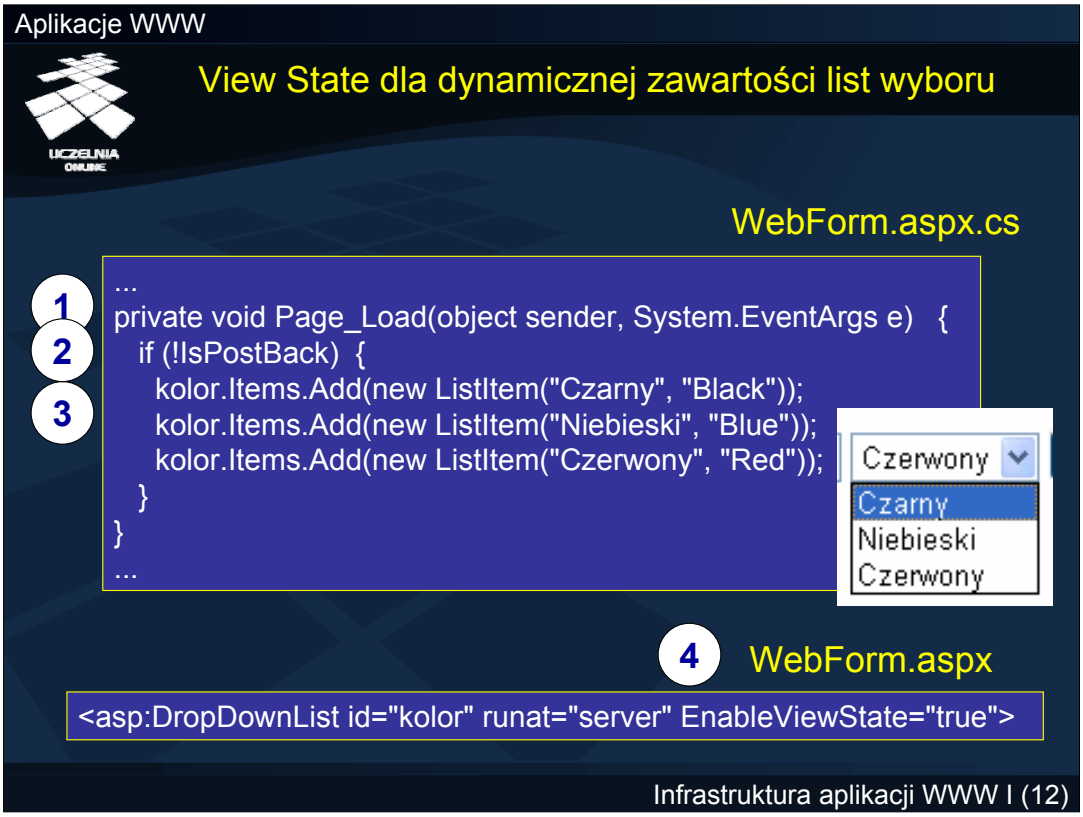
- Formularze Web Forms są stanowe
  - wartości wprowadzane do formularzy zawsze zachowywane
  - możliwe zachowanie stanu wszystkich kontrolek

Na slajdzie pokazano efekt działania aplikacji omówionej na dwóch poprzednich slajdach. Użytkownik wprowadza imię, wybiera kolor, a następnie zatwierdza formularz przyciskiem „Powitaj”. Formularz wywołuje sam siebie. Po jego ponownym wyświetleniu, zmieniła się treść i kolor tekstu powitalnego, a wartości wprowadzone przez użytkownika w elementach formularza (polu tekstowym i liście rozwijanej) zostały zachowane. Dzieje się tak dlatego, że formularze Web Forms są stanowe. Dla formularzy Web Forms ASP.NET automatycznie pamięta i odtwarza wartości wprowadzone/wybrane w formularzu, wysyłane do serwera wraz z kolejnym żądaniem (jako tzw. postback data). Tego zachowania nie można zmienić. Możliwe w ASP.NET jest również zachowanie pełnego stanu wszystkich kontrolek, co będzie omówione na kolejnych slajdach.



Poza odtwarzaniem wartości wprowadzonych lub wybranych w formularzu, ASP.NET umożliwia pamiętanie i odtwarzanie kompletnego stanu kontrolek na stronie poprzez obsługę stanu widoku (ang. View State). View State to zakodowany kodowaniem base64 stan wszystkich kontrolek na stronie. View State jest przydatny do zapamiętania stanu kontrolek wyświetlających dane i zawartości programowo wypełnianych list wyboru. Przypomnijmy, że zawartość kontrolek służących do wprowadzania danych w formularzu jest automatycznie odtwarzana przez ASP.NET niezależnie od mechanizmu View State.

View State jest zapisany w treści wynikowego dokumentu HTML w ukrytym polu formularza o nazwie __VIEWSTATE dodawanym automatycznie przez ASP.NET. Obsługę stanu poprzez View State można włączać i wyłączać atrybutem EnableViewState na poziomie serwera, strony i pojedynczych kontrolek. Domyślnie obsługa View State jest włączona. Sugerowane jest wyłączanie obsługi View State dla kontrolek, dla których nie jest konieczne pamiętanie stanu między wywołaniami strony, w celu zmniejszenia ilości informacji zakodowanych w ukrytym elemencie __VIEWSTATE, a w konsekwencji rozmiaru stron HTML generowanych przez ASP.NET.



Przykład na slajdzie pokazuje sposób dynamicznego wypełnienia listy rozwijanej wartościami przy pierwszym odwołaniu do strony i zachowania zawartości listy poprzez View State. Znaczenie wyróżnionych fragmentów kodu jest następujące:

1. Wypełnienie listy wartościami musi nastąpić w ramach inicjalizacji strony, czyli w metodzie Page_Load().

2. Wypełnienie listy wartościami w typowym scenariuszu powinno nastąpić przy pierwszym załadowaniu strony w ramach pracy użytkownika ze stroną. Takie podejście jest szczególnie odpowiednie, gdy lista nie zmienia się zbyt często, a odczyt danych dla niej jest czasochłonny np. z bazy danych. Informację o tym, czy strona została wyświetlona po raz pierwszy, czy też jest to wyświetlenie ponowne zawiera właściwość strony IsPostBack.

3. Elementy listy są zawarte w jej atrybucie Items, który jest kolekcją elementów ListItem. Kolejne elementy tworzone są konstruktorem ListItem() i dodawane do listy metodą Add().

4. Aby zawartość listy nie ginęła między wywołaniami strony, konieczne jest aby stan listy był zapisywany w ramach View State. Atrybut EnableViewState dla kontrolki listy rozwijanej musi mieć wartość „true” (jest to wartość domyślna, więc atrybut może być pominięty, jeśli mechanizm View State nie został wyłączony na poziomie strony lub serwera).

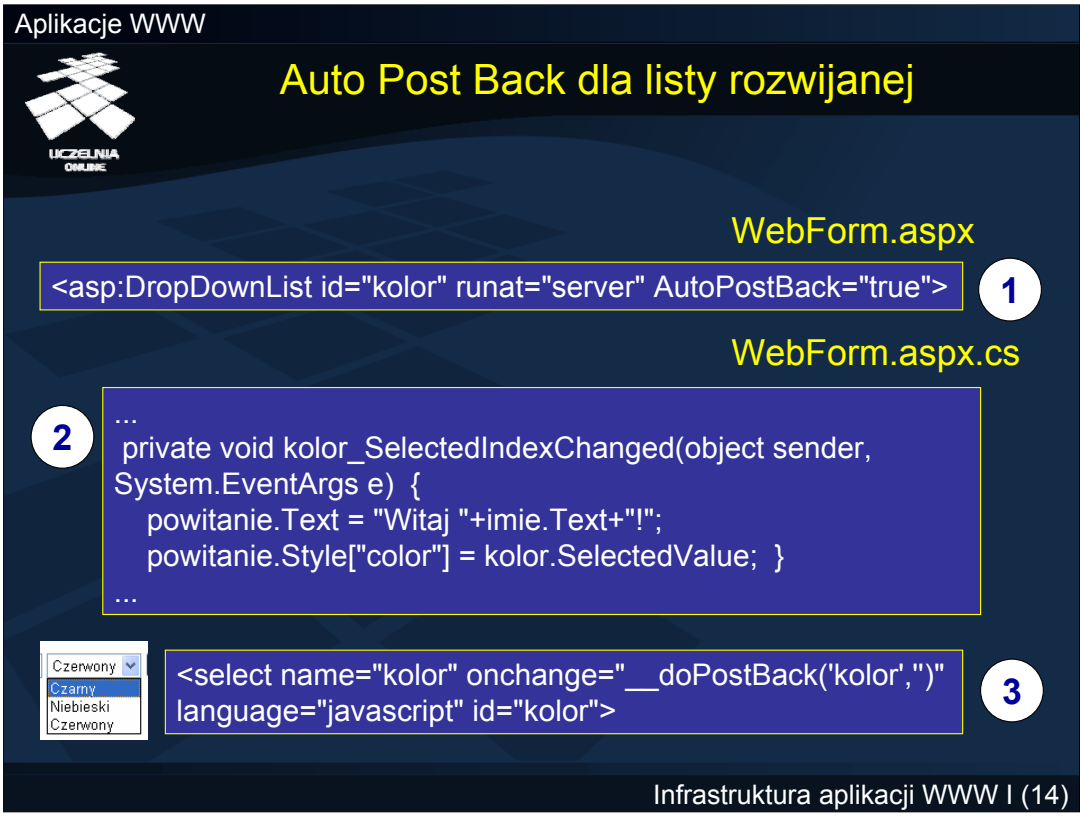
Mechanizm Auto Post Back

- Zdarzenia z kontrolek Web Forms obsługiwane na serwerze
- Odwołanie do serwera tylko w wyniku zatwierdzenia formularza
- Kontrolki Web Controls posiadają atrybut AutoPostBack, którego ustawienie spowoduje zatwierdzenie formularza po zmianie stanu kontrolki
  - poprzez automatycznie generowany kod JavaScript
- Mechanizm Auto Post Back zwiększa interaktywność aplikacji

Zdarzenia z kontrolek Web Forms obsługiwane są na serwerze. Aby została uruchomiona procedura obsługi zdarzenia, które wystąpiło w interfejsie użytkownika wyświetlanym w przeglądarce, konieczna jest interakcja z serwerem poprzez zatwierdzenie formularza. Domyślnie, formularz musi być zatwierdzony jawnie, za pomocą przycisku. Takie rozwiązanie ogranicza interaktywność aplikacji, gdyż obsługa pewnych zdarzeń jest odsuwana w czasie. Rozwiązaniem problemu jest mechanizm Auto Post Back, polegający na wymuszeniu zatwierdzenia formularza po zmianie stanu kontrolki w formularzu. Mechanizm ten można włączyć dla kontrolek z grupy Web Controls ustawiając im wartość „true” atrybutu AutoPostBack.

Od strony technicznej, automatyczne zatwierdzenie formularza w wyniku interakcji użytkownika z kontrolką inną niż przycisk zatwierdzający formularz jest realizowane przez kod JavaScript automatycznie generowany przez ASP.NET.

Mechanizm Auto Post Back zwiększa interaktywność aplikacji, kosztem zwiększenia ruchu w sieci.



Slajd pokazuje przykład wykorzystania mechanizmu Auto Post Back do zwiększenia interaktywności aplikacji pozdrawiającej użytkownika. W dotychczasowej postaci aplikacji, aby zmienić kolor tekstu powitalnego, należało najpierw wybrać żądany kolor z listy rozwijanej, a następnie zatwierdzić formularz przyciskiem. Dzięki modyfikacji przedstawionej na niniejszym slajdzie jawne zatwierdzanie formularza przyciskiem po wybraniu koloru z listy nie będzie już konieczne.

Aby zmiana stanu kontrolki powodowała zatwierdzenie formularza, a przez to natychmiastową obsługę zdarzenia zmiany stanu kontrolki, należy ustawić dla niej wartość atrybutu AutoPostBack na „true” (1). Następnie, konieczne jest oprogramowanie procedury obsługi zdarzenia zmiany stanu kontrolki – w naszym przypadku, dla listy rozwijanej jest to metoda kolor_SelectedIndexChanged() (2). Po dokonaniu powyższych zmian, generowany przez ASP.NET formularz HTML będzie zatwierdzany po zmianie wyboru pozycji z listy, dzięki kodowi JavaScript automatycznie generowanemu przez ASP.NET (funkcja __doPostBack() podpięta do zdarzenia onChange dla listy (3)).

Walidacja danych w ASP.NET

- Predefiniowane kontrolki walidacyjne z grupy Web Controls:
  - CompareValidator
  - CustomValidator
  - RangeValidator
  - RegularExpressionValidator
  - RequiredFieldValidator
  - ValidationSummary
- Walidacja po stronie klienta (JavaScript) i na serwerze
  - interaktywność, bezpieczeństwo

ASP.NET ułatwia walidację danych z formularzy, udostępniając zestaw predefiniowanych kontrolek walidacyjnych w ramach rodziny kontrolek Web Controls. Dostępne kontrolki walidacyjne to:

(a) CompareValidator – do porównania z wartością stałą lub zawartością innej kontrolki;

(b) CustomValidator – do walidacji wg specyficznej logiki aplikacji podanej przez twórcę aplikacji;

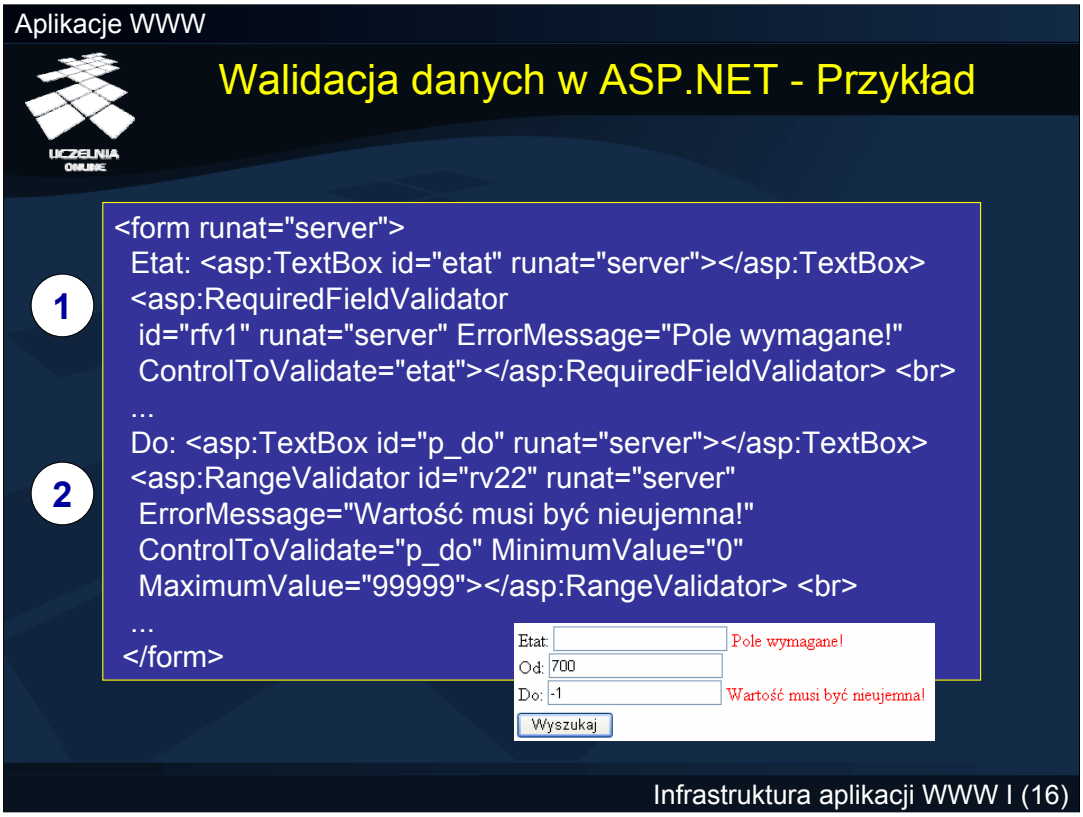
(c) RangeValidator – do testowania zawierania się w przedziale;

(d) RegularExpressionValidator – do testowania zgodności z wyrażeniem regularnym;

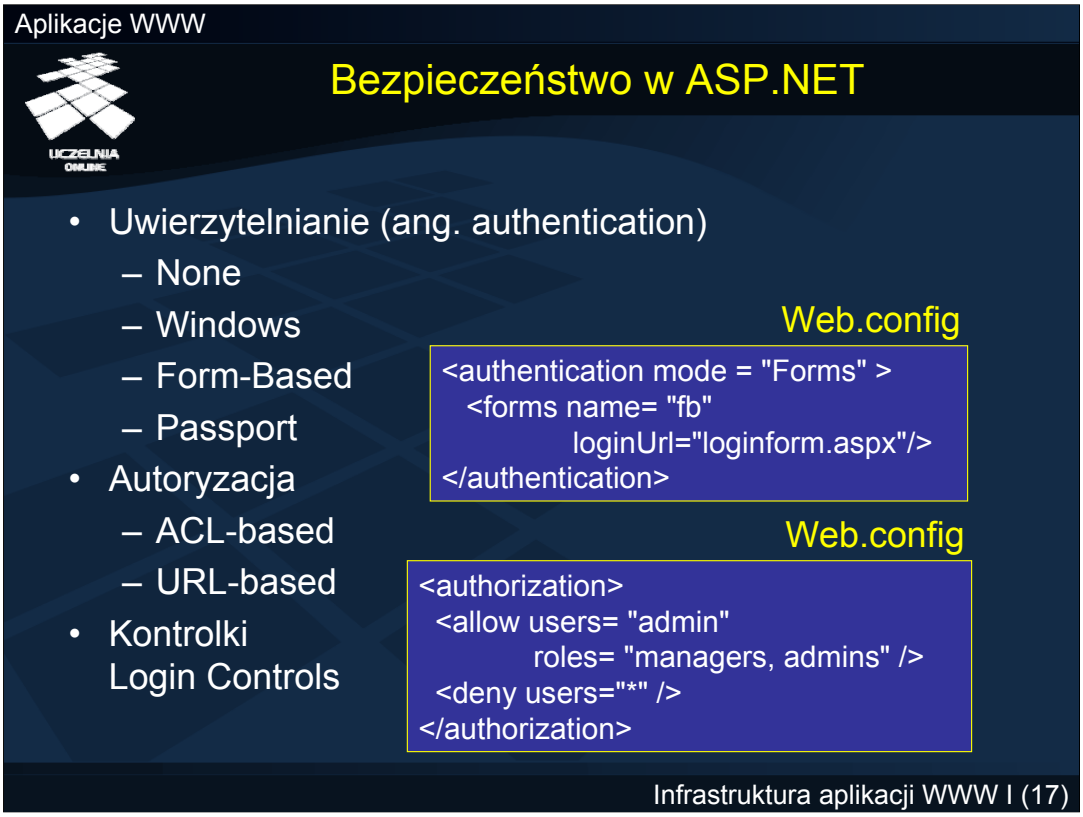
(e) RequiredFieldValidator – do wymuszenia obowiązkowości pola;

(f) ValidationSummary – do wyświetlania podsumowania wyników walidacji dla całego formularza.

Domyślnie włączona jest walidacja po stronie klienta poprzez automatycznie generowany kod JavaScript. Dzięki temu błędy sygnalizowane są zaraz po opuszczeniu pola, do którego została wprowadzona błędna wartość. Walidację po stronie klienta można wyłączyć ustawiając atrybut walidatora EnableClientScript na „false”. Niezależnie od walidacji po stronie klienta, zawsze odbywa się walidacja po stronie serwera. Zaletą walidacji w JavaScript jest szybka odpowiedź systemu na błędne dane. Nie można jednak polegać jedynie na walidacji w JavaScript, gdyż klient może mieć wyłączoną obsługę skryptów w przeglądarce. Możliwe są też próby modyfikacji kodu JavaScript po stronie klienta.



Na slajdzie pokazany został przykład wykorzystania dwóch spośród predefiniowanych walidatorów ASP.NET. Walidowany jest formularz służący do wyszukiwania pracowników według etatu i przedziału płac. Etat jest obowiązkowy, a wartości podane jako progi płacowe muszą być nieujemne. Do walidacji etatu został wykorzystany walidator RequiredFieldValidator (1), a do walidacji progów płacy walidator RangeValidator (2). Znacznik walidatora umieszcza się w kodzie strony w miejscu, w którym ma pojawiać się generowany przez niego komunikat o błędzie w przypadku niepowodzenia walidacji. Walidowana kontrolka jest wskazywana atrybutem ControlToValidate, a tekst komunikatu o błędzie jest zawarty w atrybucie ErrorMessage. Dla walidatora przedziałowego (2) minimalna i maksymalna dopuszczalna wartość są zawarte w atrybutach MinimumValue i MaximumValue.



Aplikacje WWW mogą być ogólnodostępne lub dostępne tylko dla uwierzytelnionych użytkowników. Powszechnie występują również aplikacje, w których pewne strony są ogólnodostępne, a inne wymagają uwierzytelnienia. Uwierzytelnienie (ang. authentication) i autoryzacja (ang. authorization) to dwa kluczowe etapy kontroli dostępu, a przez to zabezpieczania aplikacji. Uwierzytelnienie polega na zweryfikowaniu tożsamości użytkownika i najczęściej jest realizowane poprzez weryfikację nazwy użytkownika i hasła podanego przez użytkownika. Autoryzacja sprowadza się do określania jakie operacje na jakich zasobach może wykonać dany uwierzytelniony użytkownik.

ASP.NET oferuje następujące możliwości uwierzytelniania użytkowników:

(a) None – brak uwierzytelniania;

(b) Windows – uwierzytelnianie przez IIS (Basic/Digest) w oparciu o użytkowników i grupy z systemu operacyjnego Windows; jest to rozwiązanie akceptowalne w intranecie;

(c) Form-based – oparte o formularz logowania i zmienne Cookie; użytkownik jest uznany za uwierzytelnionego jeśli wraz z jego żądaniem przyszła odpowiednia zmienna Cookie, która jest ustawiana przez specjalnie przygotowany formularz, wskazany jako formularz logowania; jest to rozwiązanie stosowane najpowszechniej;

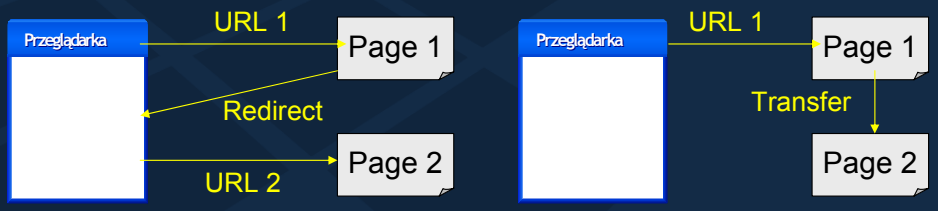
(d) Passport – z wykorzystaniem centralnego serwisu uwierzytelniania Microsoft Passport.

Autoryzacja może być: ACL-based , czyli oparta o ustawienia Access Control List na poziomie systemu plików NTFS serwera lub URL-based, czyli oparta o ścieżki adresów URL.

ASP.NET umożliwia deklaratywną konfigurację uwierzytelniania i autoryzacji poprzez odpowiednie sekcje w plikach konfiguracyjnych aplikacji Web.config.

Nawigacja w aplikacjach ASP.NET

- Formularz wywołuje sam siebie
- Kod obsługujący zdarzenie zatwierdzenia formularza może przekazać sterowanie do innej strony:
  - Response.Redirect()
  - Server.Transfer()



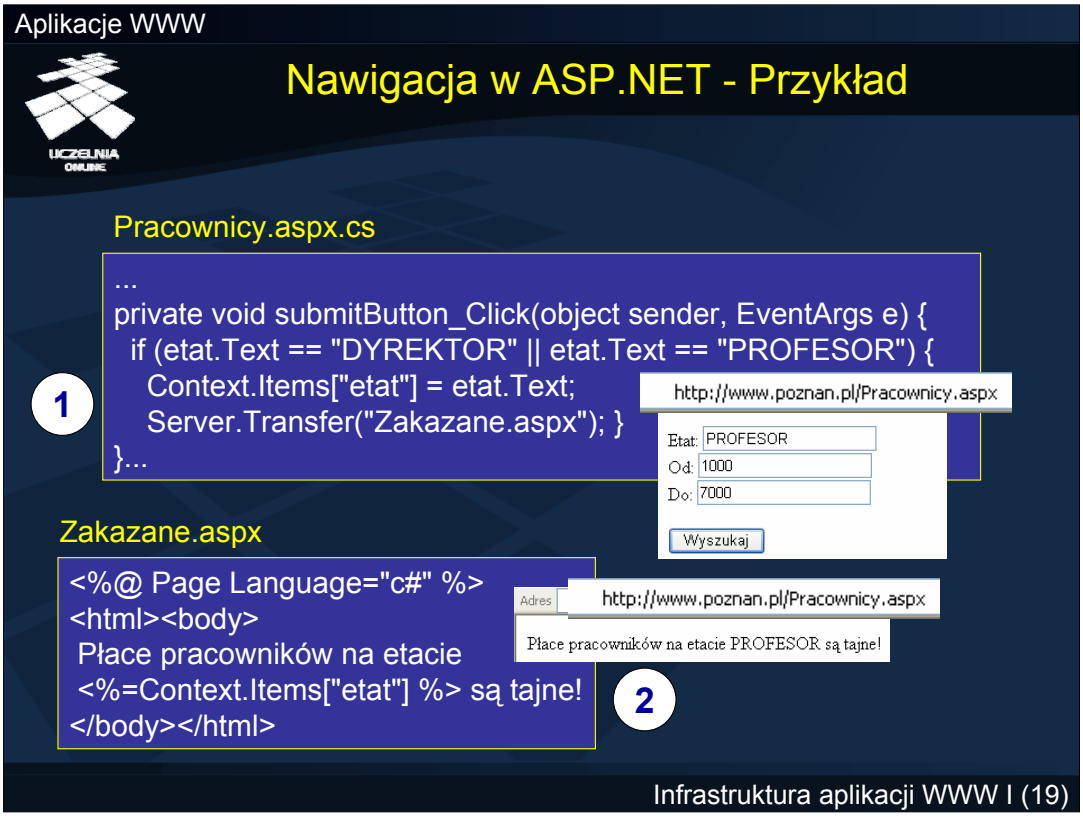
Formularze Web Forms (przetwarzane po stronie serwera – runat=”server”) po zatwierdzeniu wywołują same siebie. Jest to wygodne podejście z punktu widzenia np. walidacji danych i symulowania stanowego interfejsu użytkownika. W praktyce jednak aplikacje WWW obejmują wiele stron, między którymi następują warunkowe przekierowania. Reguły nawigacji są w ASP.NET najczęściej implementowane w kodzie strony, ewentualnie w pliku Code Behind.

Przekierowanie użytkownika do innej strony może być zrealizowane na dwa sposoby:

Response.Redirect() przekierowuje użytkownika do innej strony za pośrednictwem przeglądarki. Użytkownik ma świadomość, że ogląda inną stronę, gdyż wskazuje na to zmiana adresu URL w przeglądarce.

Server.Transfer() to przekierowanie odbywające się po stronie serwera. Jego wadą jest to, że z punktu widzenia przeglądarki nie nastąpiła zmiana adresu URL. Zalety to redukcja komunikacji między przeglądarką a serwerem oraz możliwość przekazania danych przez obiekt Context reprezentujący zasięg bieżącego żądania.

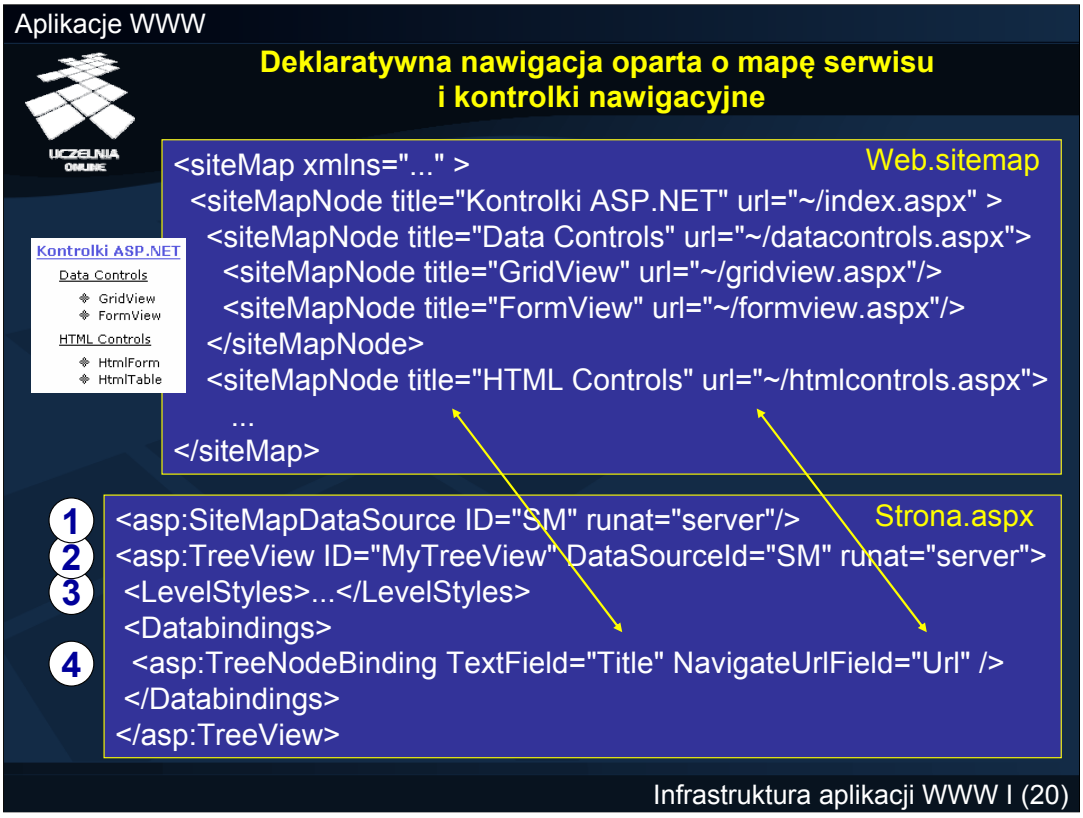
W przypadku Response.Redirect() przekazanie danych między stronami może się odbyć poprzez zasięg sesji lub zakodowanie ich jako parametrów wywołania w adresie URL. Wykorzystanie do tego celu zmiennych o zasięgu sesji może stanowić problem gdy użytkownik ma otwartych kilka okien przeglądarki dla tej samej aplikacji.



Na slajdzie pokazano przykład nawigacji między stronami ASP.NET metodą Server.Transfer(), preferowaną ze względu na lepszy sposób przekazywania danych. Pierwszą stroną aplikacji jest formularz do wprowadzania kryteriów wyszukiwania pracowników, a drugą - strona informująca o tym, że płace są tajne. Przekierowanie do tej drugiej strony następuje gdy użytkownik podejmie próbę wyszukania pracowników będących dyrektorami lub profesorami. Znaczenie przedstawionych fragmentów kodu jest następujące:

1. Warunkowa nawigacja z formularza wprowadzania danych jest realizowana w procedurze obsługi zdarzenia naciśnięcia przycisku zatwierdzającego formularz. Przekierowanie następuje pod warunkiem podania przez użytkownika etatu „DYREKTOR” lub „PROFESOR”. Przed samym przekierowaniem, nazwa wybranego etatu jest zapamiętywana w obiekcie Context pod kluczem „etat”. Przekierowanie do strony Zakazane.aspx jest realizowane metodą Server.Transfer().

2. Druga strona wyświetla tekst o utajnieniu płac na podanym przez użytkownika etacie. Nazwa etatu odczytywana jest z kontekstu. Po przekierowaniu użytkownika do drugiej strony, w jego przeglądarce dalej wyświetlany jest adres pierwszej strony. Dzieje się tak dlatego, że przekierowanie metodą Server.Transfer() odbywa się po stronie serwera.



Dla serwisów WWW o hierarchicznej strukturze, ASP.NET od wersji 2.0 umożliwia deklaratywną specyfikację nawigacji dla serwisu WWW w oparciu o mapę serwisu zdefiniowaną w pliku konfiguracyjnym Web.sitemap i kontrolki nawigacyjne.

Na slajdzie pokazano definicję struktury serwisu dla przykładowego serwisu prezentującego informacje o kontrolkach dostępnych w ASP.NET. Struktura serwisu z jedną wyróżnioną stroną główną i następnie kilkoma poziomami hierarchicznie zagnieżdżonych stron predysponuje go do wykorzystania deklaratywnej mapy serwisu i kontrolek nawigacyjnych o nią opartych.

Mapa serwisu jest zdefiniowana w pliku XML Web.sitemap. Element główny <siteMap> zawiera jeden element <siteMapNode>, reprezentujący stronę główną serwisu. W nim zawarte są hierarchicznie zagnieżdżone elementy <siteMapNode> reprezentujące pozostałe strony serwisu. Każdy element <siteMapNode> posiada atrybuty „title” i „url”, które będą wykorzystane przez kontrolki nawigacyjne jako nazwa i adres URL dla linków na stronie.

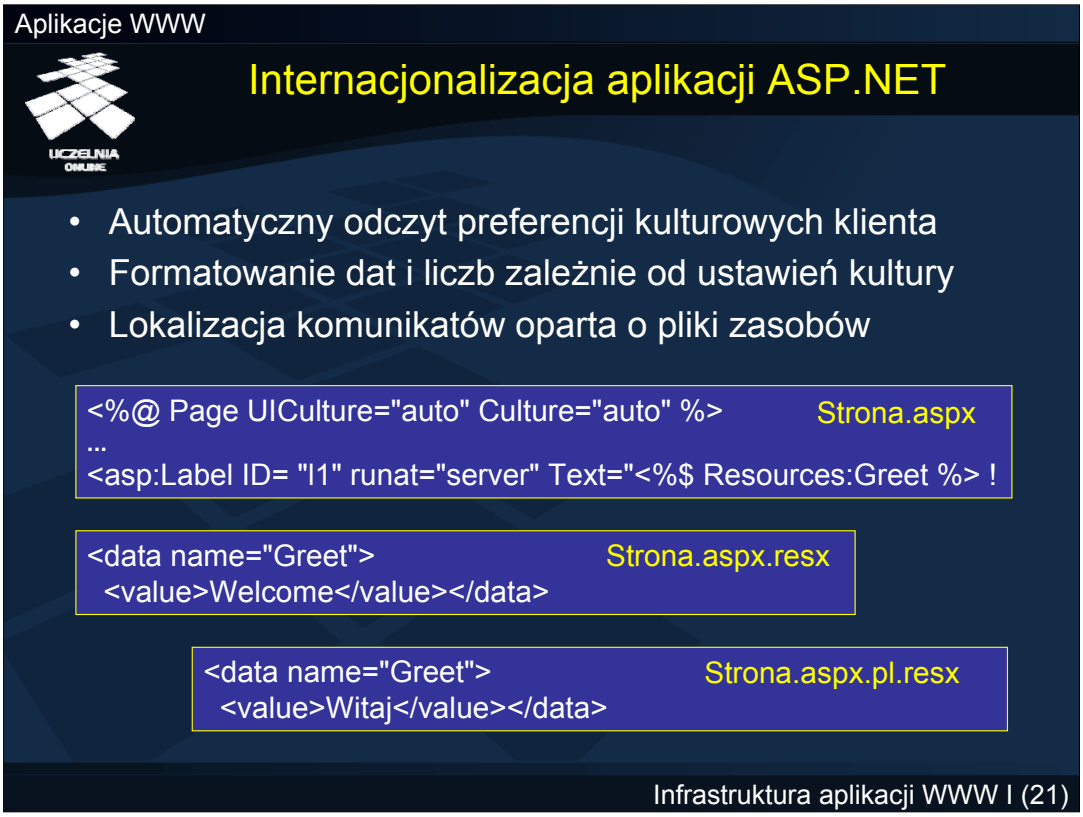
U dołu slajdu pokazany jest fragment strony ASP.NET definiujący kontrolkę drzewa nawigacyjnego, a w górnym lewym rogu jej wygląd w przeglądarce. Znaczenie wyróżnionych fragmentów kodu strony jest następujące:

1. Kontrolka źródła danych, udostępniająca mapę serwisu z pliku Web.sitemap.

2. Kontrolka drzewa, wykorzystująca jako źródło danych kontrolkę z mapą serwisu.

3. Sekcja <LevelStyles> zawierająca definicje wyglądu poszczególnych poziomów drzewa.

4. Sekcja <Databindings> wiążąca atrybuty opisu tekstowego i adresu URL do nawigacji węzłów drzewa z atrybutami węzłów zawartych w źródle danych.



Od wersji 2.0 ASP.NET oferuje silne wsparcie dla internacjonalizacji aplikacji obejmujące:

(a) Automatyczny odczyt preferencji kulturowych klienta (ang. culture) obejmujących język, nazwy miesięcy, dni tygodnia, sposób formatowania dat i liczb itp.

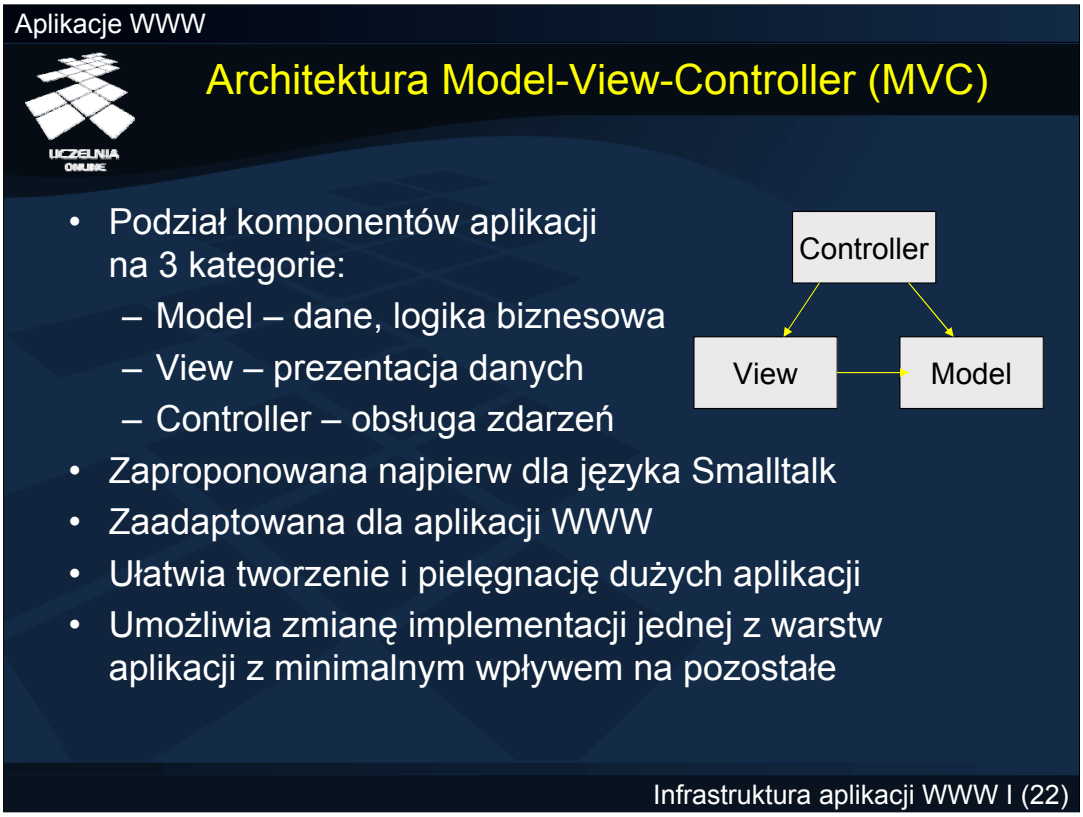
(b) Formatowanie dat i liczb zależnie od ustawień kultury.

(c) Lokalizację komunikatów opartą o pliki zasobów, z możliwością dodawania wsparcia dla kolejnych języków dla działającej aplikacji, poprzez dodawanie kolejnych zlokalizowanych plików zasobów (podobnie jak w aplikacjach języka Java).

Slajd pokazuje przykład strony, która wspiera języki angielski (domyślnie) i polski. Kod strony zawarty jest w pliku Strona.aspx. Deklaracja @Page strony zawiera dwa atrybuty związane z internacjonalizacją aplikacji: UICulture i Culture. Pierwszy wskazuje ustawienia lokalizacji dla komunikatów aplikacji, a drugi determinuje sposób formatowania dat i liczb. W przykładzie oba mają wartość „auto” co oznacza, że ustawienia mają być odczytane automatycznie z preferencji zawartych w żądaniu wysłanym z przeglądarki. Można oczywiście jako wartość tych atrybutów podać konkretny kod lokalizacji np. „pl” „de”, „en-US”.

Zlokalizowane komunikaty umieszczane są na stronie za pomocą dostępnego od ASP.NET 2.0 nowego typu wyrażeń, zastępowanych wartością przed parsowaniem strony (notacja ze znakiem „$”). W przykładzie, tekst związany w plikach zasobów z kluczem „Greet” jest umieszczany jako tekst prezentowany przez komponent etykiety.

U dołu slajdu umieszczone są dwa pliki komunikatów dla strony Strona.aspx. Pierwszy plik zawiera komunikaty neutralne kulturowo, w tym wypadku w języku angielskim, a drugi przygotowane dla lokalizacji polskiej. Klucz komunikatu zawarty jest w atrybucie „name” elementu <data>, a odpowiadająca mu wartość w zagnieżdżonym elemencie <value>. Pliki zasobów posiadają rozszerzenie .resx, poprzedzone kodem lokalizacji.



Model-View-Controller (MVC) to architektura aplikacji (uznawana niekiedy również za wzorzec projektowy) zakładająca podział komponentów aplikacji na 3 kategorie:

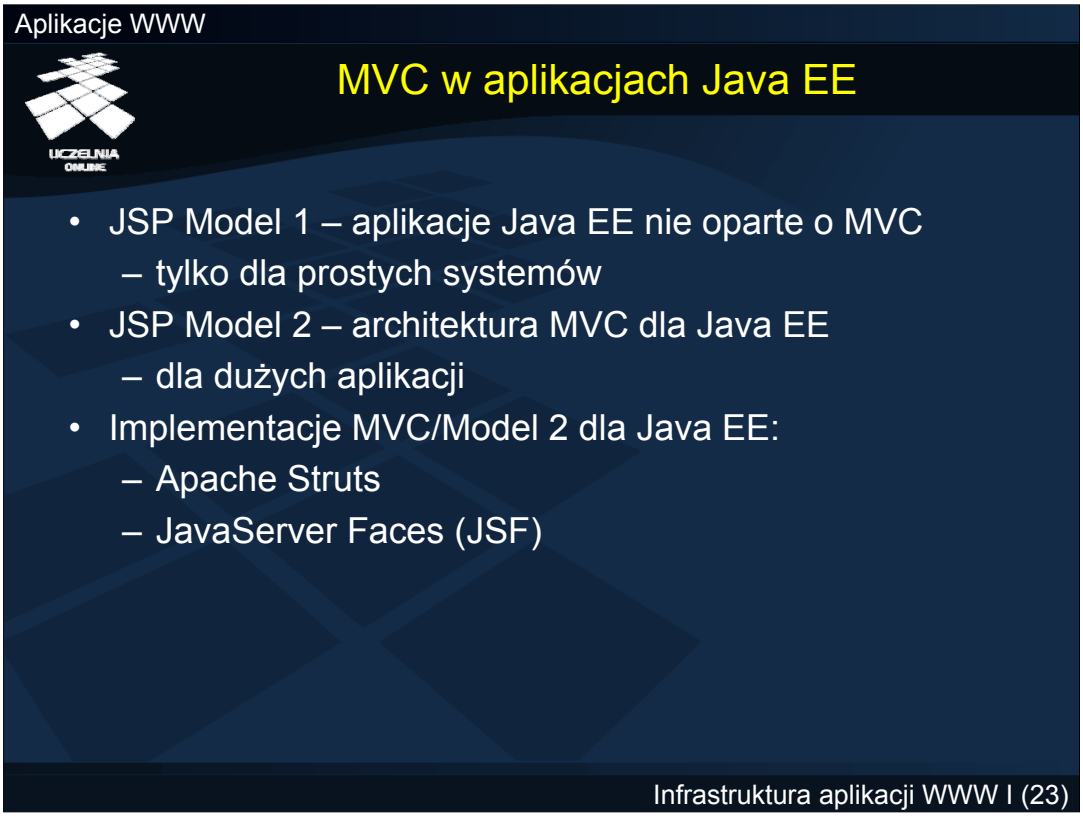
Model – komponenty modelu, reprezentujące dane, na których operuje aplikacja i zawierające implementację logiki biznesowej;

View – komponenty widoku, odpowiadające za prezentację danych zawartych w modelu, czyli interfejs użytkownika;

Controller – komponenty sterujące, odpowiadające za obsługę zdarzeń, najczęściej zachodzących w interfejsie użytkownika w wyniku interakcji użytkownika z aplikacją; do zadań kontrolera należy również modyfikowanie modelu i widoku.

Architektura MVC została zaproponowana najpierw dla języka Smalltalk i była wykorzystywana w implementacjach graficznego interfejsu użytkownika. Następnie została zaadaptowana dla aplikacji WWW, z zachowaniem ogólnych idei, ale z pewną specyfiką, np. obsługą nawigacji między stronami jako jednym z zadań kontrolera.

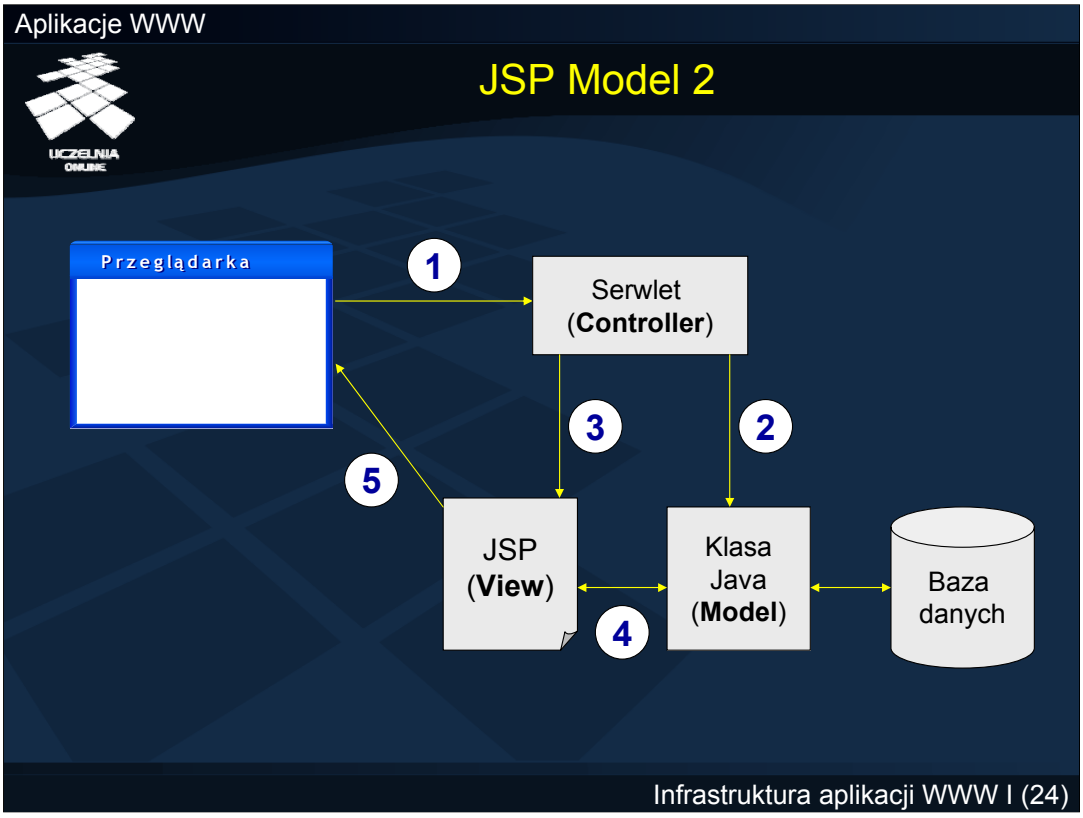
Architektura MVC ułatwia tworzenie i pielęgnację dużych aplikacji, prowadząc do bardziej przejrzystego i lepiej zorganizowanego kodu oraz umożliwiając zmianę implementacji jednej z warstw aplikacji z minimalnym wpływem na pozostałe.



Architektura MVC dla aplikacji WWW jest szczególnie popularna na platformie Java EE (Java Enterprise Edition). Przyczyny takiego stanu rzeczy są co najmniej dwie. Po pierwsze, Java EE jest obok .NET jedną z dwóch głównych platform odpowiednich dla złożonych aplikacji WWW. Po drugie, Java EE daje twórcom aplikacji duży wybór technologii składowych i typów komponentów, często z możliwością realizacji tego samego celu na kilka sposobów. Z czasem okazało się, że konieczne jest wypracowanie wzorców opisujących sposób tworzenia złożonych aplikacji z elementarnych technologii Java EE i rolę poszczególnych typów komponentów.

Tradycyjny model aplikacji, w którym logika prezentacji miesza się w kodzie JSP z logiką biznesową i logiką nawigacji określany jest obecnie jako JSP Model 1. JSP Model 1 jest akceptowalny jedynie dla prostych systemów. Nowszy, zalecany dla dużych aplikacji model jest oparty o architekturę MVC i określany jako JSP Model 2.

Istnieje wiele implementacji MVC/JSP Model 2 dla Java EE. Najważniejsze dwie to Apache Struts jako sprawdzona i najbardziej rozpowszechniona i JavaServer Faces (JSF) jako implementacja oficjalna, stanowiąca część specyfikacji Java EE.



W implementacjach MVC dla Java EE rolę widoku najczęściej pełnią strony JSP, stąd nazwa JSP Model 2, choć czasem wykorzystywana do implementacji widoku jest zamiast JSP np. technologia szablonów dla serwletów Velocity. Kontrolerem jest zawsze serwlet. W zakresie doboru technologii dla modelu jest największa swoboda. Najczęściej model stanowią zwykłe klasy Java lub komponenty EJB.

Na slajdzie pokazano typowy schemat obsługi żądania w modelu JSP Model 2:

1. Przeglądarka wysyła żądanie. Aplikacja jest tak skonfigurowana, że każde żądanie jest kierowane do serwletu – kontrolera.

2. Serwlet – kontroler analizuje żądanie i tworzy wymagane przez żądany widok obiekty klas zewnętrznych. Interakcja kontrolera z modelem może pociągać za sobą interakcję z bazą danych.

3. Serwlet przekazuje sterowanie do odpowiedniego widoku - strony JSP.

4. JSP pobiera dane z obiektów modelu przygotowanych przez kontroler. Obiekty te mogą udostępniać dane pobrane z bazy danych.

5. JSP generuje wynikowy dokument HTML przesyłany do przeglądarki.

Apache Struts

- Szkielet aplikacji (ang. framework) MVC dla Java EE
- Opracowane przez Craiga McClanahana, podarowane Apache Software Foundation
- Składniki Struts:
  - gotowy konfigurowalny serwlet – kontroler
  - klasy biblioteczne Java wykorzystywane podczas implementacji kodu przetwarzania żądań np. Action i ActionForm
  - bogaty zbiór bibliotek znaczników JSP

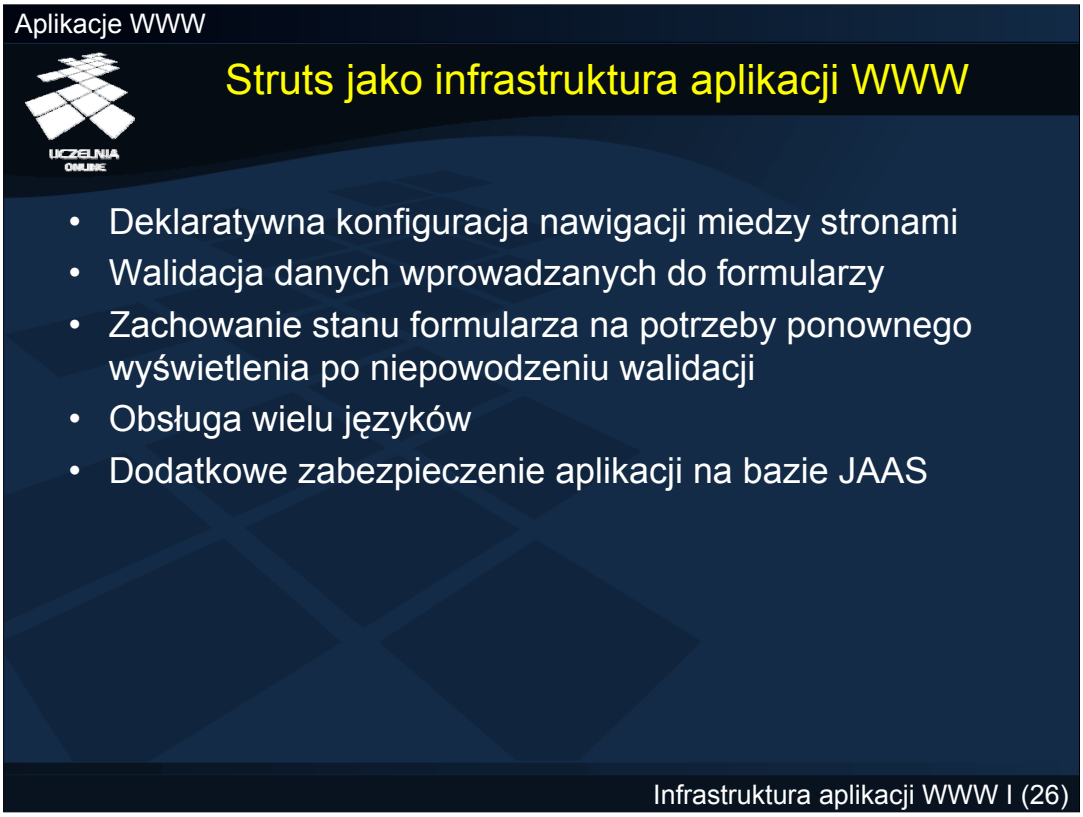
Apache Struts to szkielet aplikacji (ang. framework) MVC dla Java EE, opracowany przez Craiga McClanahana, a następnie podarowany Apache Software Foundation.

Składniki Struts to:

(a) gotowy konfigurowalny serwlet – kontroler ActionServlet;

(b) klasy biblioteczne Java wykorzystywane podczas implementacji kodu przetwarzania żądań np. Action do implementacji akcji obsługującej żądanie i ActionForm do reprezentacji danych z formularza;

(c) bogaty zbiór bibliotek znaczników JSP, z których część straciła na znaczeniu wraz z pojawieniem się JSTL.



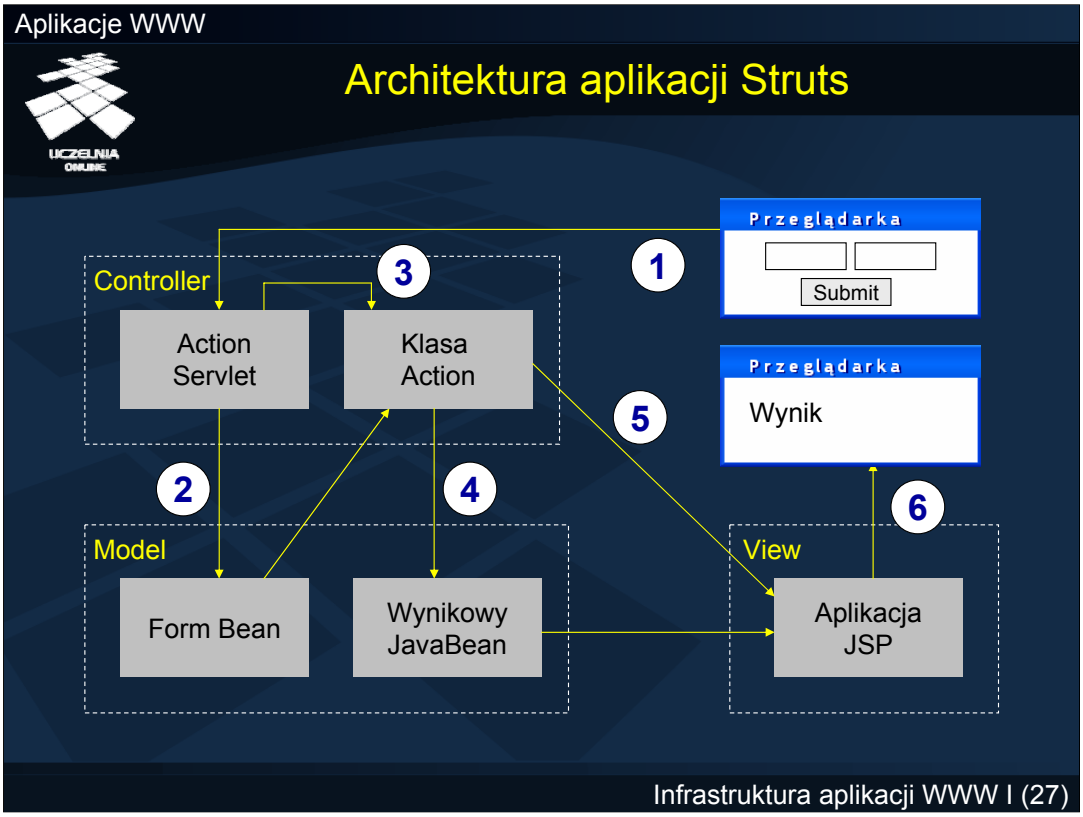
Wykorzystanie Struts jako szkieletu dla tworzonej aplikacji WWW zwalnia twórców aplikacji z oprogramowania wielu obszarów funkcjonalnych infrastruktury aplikacji. Główną zaletą Struts jest deklaratywna konfiguracja nawigacji miedzy stronami w pliku konfiguracyjnym kontrolera. W przypadku zmiany kolejności stron w nawigacji czy nazw stron JSP wystarczy zmodyfikować plik konfiguracyjny aplikacji Struts.

Struts kładzie również duży nacisk na walidację danych, oferując zestaw predefiniowanych walidatorów z możliwością walidacji po stronie klienta.

Struts słabo ułatwia tworzenie stanowego interfejsu użytkownika. Wsparcie w tym zakresie ogranicza się do odtwarzania zawartości pól formularza przy ponownym jego wyświetleniu w wyniku błędu walidacji.

Struts wspiera tworzenie aplikacji wielojęzycznych w oparciu o ogólny dla Javy mechanizm zbiorów zasobów z komunikatami (ang. resource bundles).

Struts oferuje też drobne uzupełnienie w zakresie deklaratywnej konfiguracji autoryzacji na gruncie usług uwierzytelniania i autoryzacji dla Javy (JAAS), umożliwiając specyfikowanie wymaganych ról dla poszczególnych akcji w aplikacji Struts.



Slajd pokazuje architekturę i przepływ sterowania w aplikacji Struts:

1. Żądanie HTTP z przeglądarki trafia do serwletu-kontrolera, który jest skonfigurowany jako centralny „punkt wejścia” do systemu.

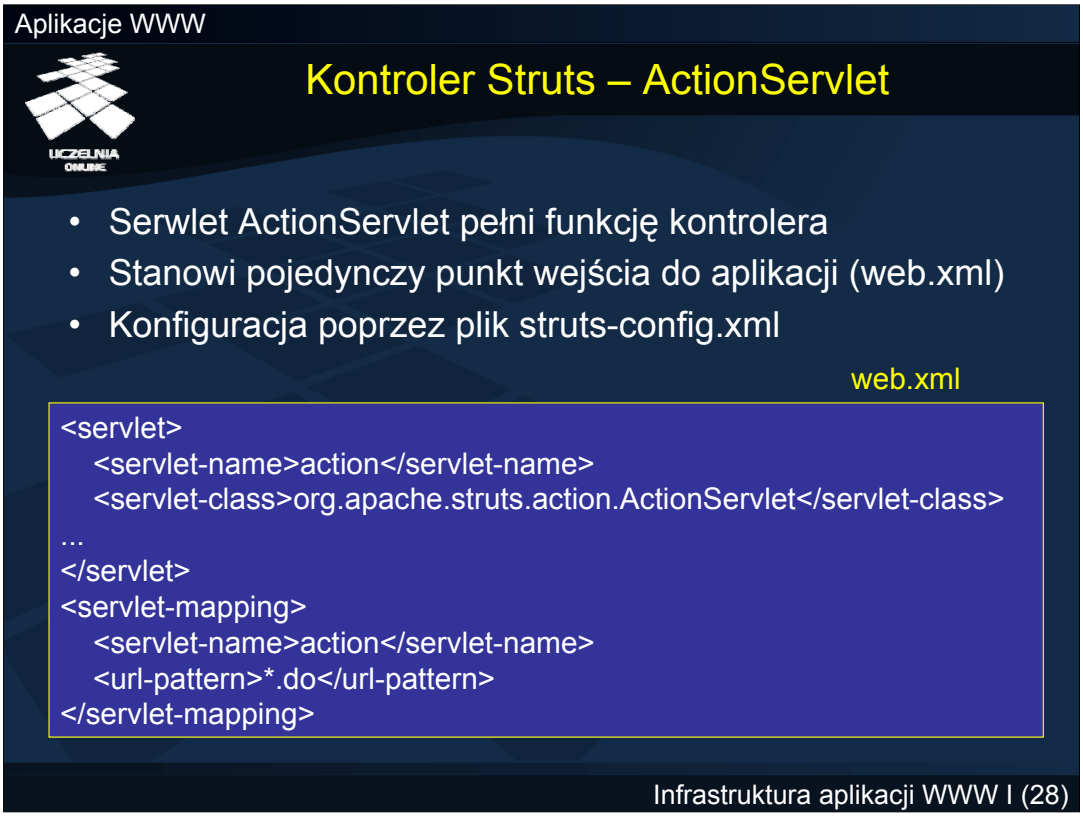
2. Kontroler tworzy obiekt Form Bean zawierający dane z formularza HTML, którego zatwierdzenie spowodowało wysłanie żądania.

3. Kontroler uruchamia akcję związaną z żądaniem, wywołując metodę execute() klasy Action.

4. Klasa Action realizuje logikę aplikacji pobierając dane wejściowe z komponentu Form Bean i tworzy wynikowy obiekt JavaBean, zawierający dane, które ma zaprezentować widok.

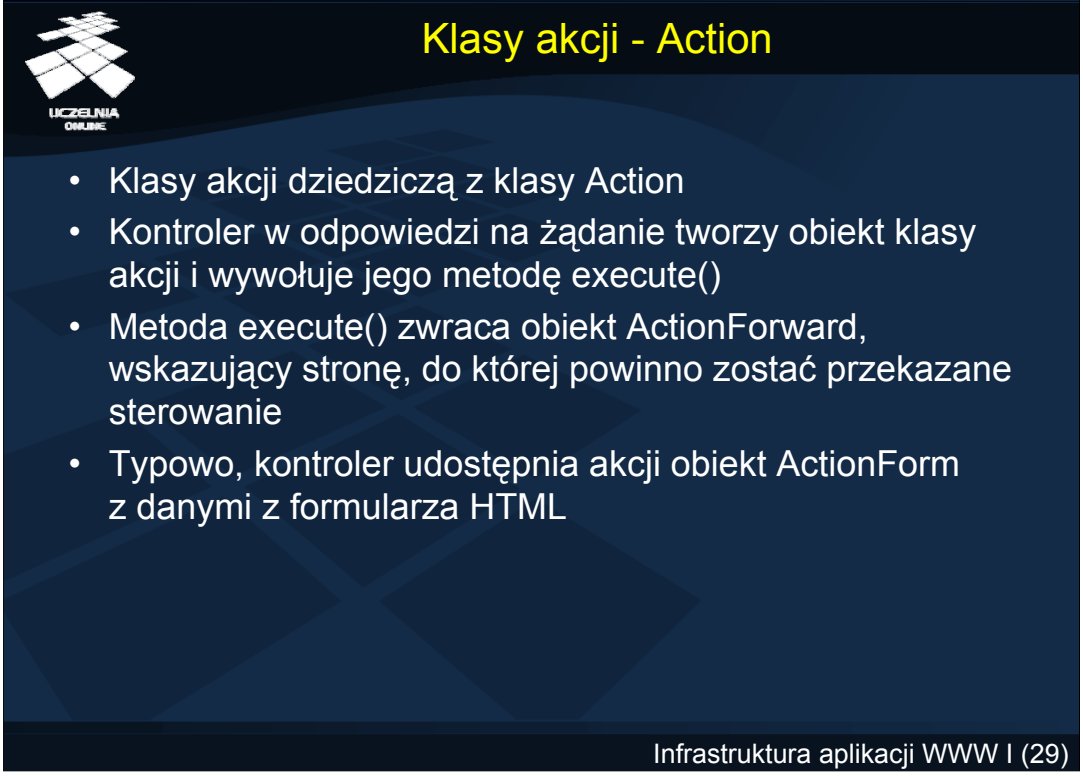
5. Zależnie od wyniku działania akcji, kontroler dokonuje warunkowego przekierowania do jednego z widoków (stron JSP).

6. Strona JSP generuje wynikowy dokument HTML, umieszczając w nim dane przygotowane przez akcję w postaci obiektu JavaBean.



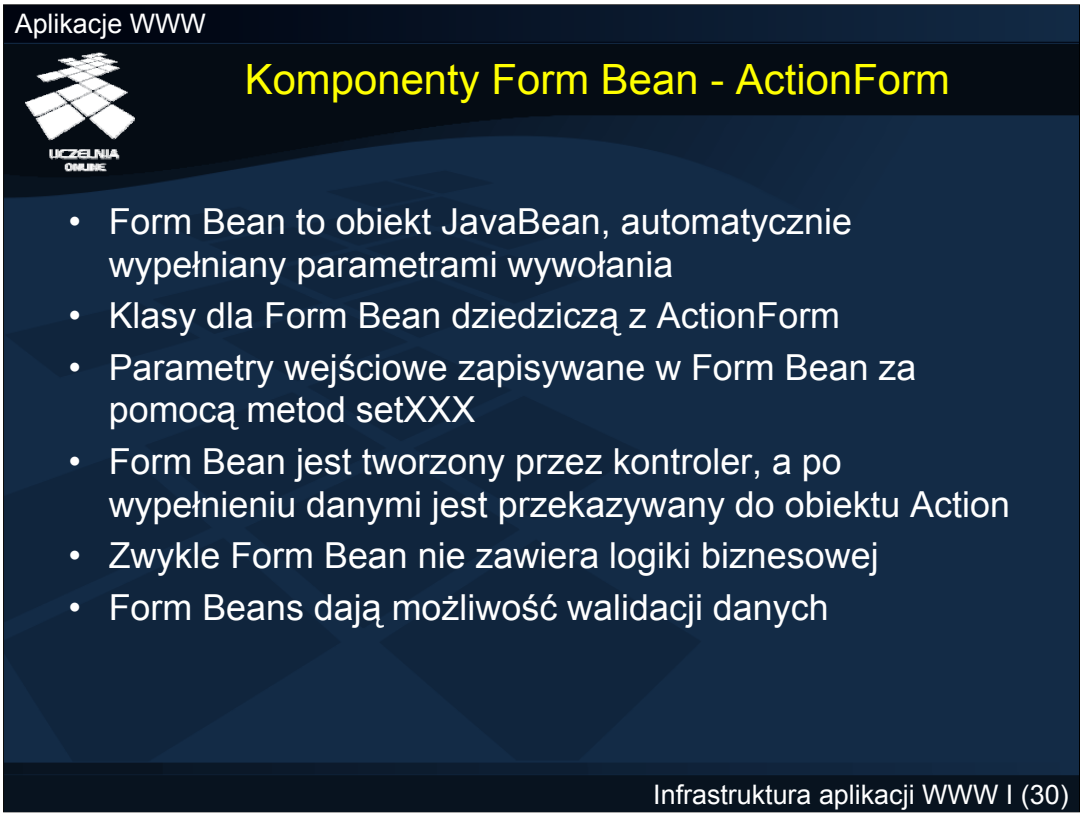
Serwlet ActionServlet (pakiet org.apache.struts.action) pełni w aplikacji Struts funkcję kontrolera. Dzięki odpowiedniej konfiguracji w pliku web.xml (pokazanej na slajdzie) stanowi on pojedynczy punkt wejścia do całego systemu aplikacyjnego. Zwyczajowo, serwlet ActionServlet reaguje na wszystkie żądania HTTP, których adres URL posiada rozszerzenie .do.

ActionServlet to gotowy, w pełni zaimplementowany uniwersalny kontroler. Dla konkretnej aplikacji należy go skonfigurować poprzez XML-owy plik konfiguracyjny – domyślnie WEB-INF/struts-config.xml. Plik ten zawiera m.in. definicje akcji, reguły nawigacji itp.



Klasy akcji tworzone przez programistę muszą dziedziczyć z klasy bibliotecznej Action. Kontroler w odpowiedzi na żądanie tworzy obiekt klasy akcji i wywołuje jego metodę execute().

Metoda execute() realizuje logikę aplikacji i zwraca obiekt ActionForward, wskazujący stronę, do której powinno zostać przekazane sterowanie. Typowo, kontroler udostępnia akcji obiekt ActionForm, nazywany komponentem Form Bean, zawierający dane z formularza HTML, z którego wywołana została akcja.



Form Bean to obiekt JavaBean, który jest automatycznie wypełniany parametrami wywołania pochodzącymi od użytkownika; wartości parametrów wywołania stają się wartościami właściwości obiektu JavaBean.

Programista tworzy klasę dla Form Bean, dziedzicząc z klasy bibliotecznej ActionForm. Każdy parametr wejściowy jest zapisywany w obiekcie Form Bean za pomocą wywołania metody setXXX; zadaniem programisty jest implementacja tych metod.

Obiekt Form Bean jest tworzony przez kontroler (ActionServlet), a po wypełnieniu danymi przekazywany do obiektu Action (jako argument metody execute()).

Zwykle klasa Form Bean nie zawiera kodu logiki biznesowej, a jedynie metody setXXX i getXXX, ustawiające i odczytujące wartości właściwości i ewentualnie logikę walidacji.

Komponenty Form Beans mogą być wykorzystane do walidacji danych wejściowych, stanowiąc jedną z możliwości w zakresie walidacji danych, dostępną w Struts.

Plik konfiguracyjny struts-config.xml

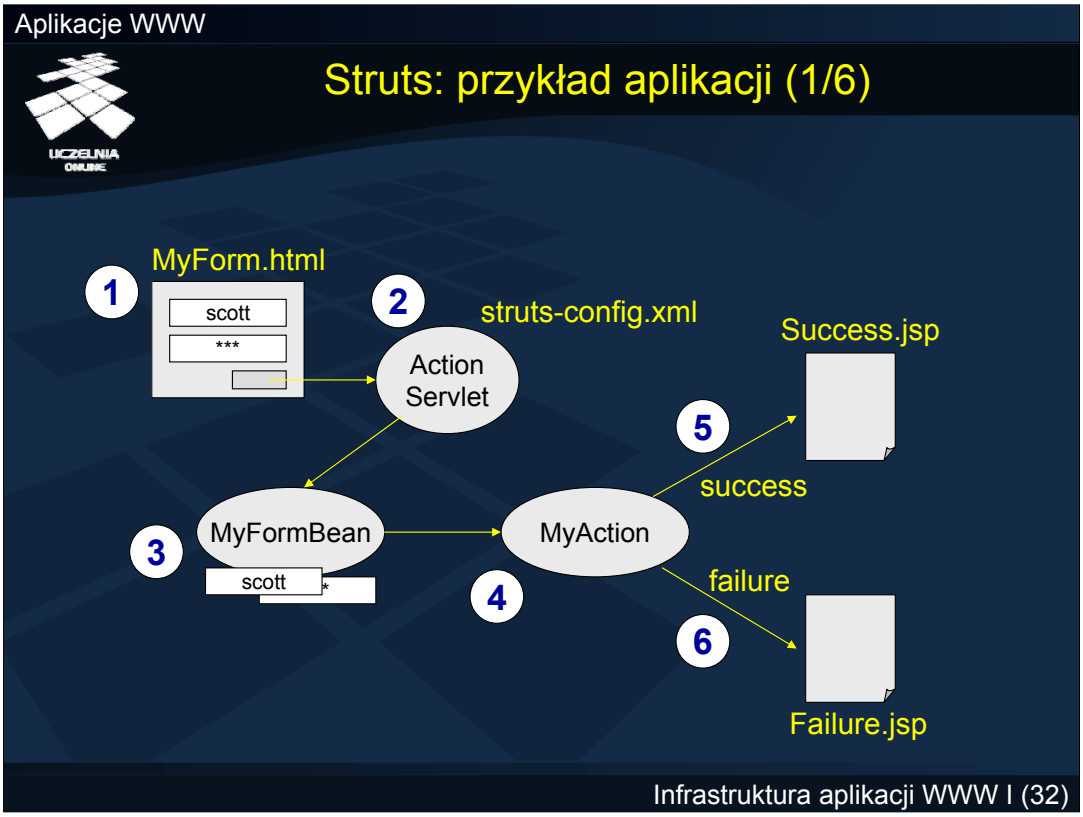
- Zawiera konfigurację serwletu kontrolera
- Główne sekcje:
  - <form-beans> - opisy wykorzystywanych komponentów Form Bean
  - <action-mappings> - definicje akcji wywoływanych przez ActionServlet
  - <forward> - definicje etykiet dla adresów URL, wykorzystywane podczas przekazywania żądań

Plik konfiguracyjny aplikacji Struts, zawierający konfigurację kontrolera zwyczajowo nazywa się struts-config.xml i znajduje się w katalogu WEB-INF aplikacji (nazwę i lokalizację wskazuje parametr inicjalizacyjny serwletu ActionServlet w pliku web.xml). Główne sekcje pliku struts-config.xml (w formie elementów XML) to:

<form-beans> - opisy wykorzystywanych komponentów Form Bean, dla każdego komponentu: name - nazwa komponentu, type - nazwa klasy;

<action-mappings> - definicje akcji wywoływanych przez ActionServlet, dla każdej akcji: path – ścieżka URL reprezentująca żądanie wykonania akcji (bez ".do"), type – nazwa klasy Action, name – nazwa komponentu Form Bean skojarzonego z tą akcją, validate – czy w obiekcie Form Bean wywołać metodę validate()?, input – ścieżka URL, do której nastąpi przekierowanie gdy walidacja się nie powiedzie;

<forward> - definicje etykiet dla adresów URL; etykiety te są wykorzystywane podczas przekazywania żądań w ramach nawigacji obsługiwanej przez kontroler; dla każdej etykiety: name – nazwa etykiety, path – reprezentowana ścieżka URL, redirect – przekazać sterowanie (forward) czy przekierować przeglądarkę (redirect)?



Na kolejnych slajdach tworzenie aplikacji Struts zostanie zilustrowane prostym przykładem. Przykładowa aplikacja obsługuje logowanie użytkownika do systemu.

1. Użytkownik wprowadza dane do formularza HTML.

2. Po zatwierdzeniu formularza obsługa żądania po stronie serwera rozpoczyna się od wywołania kontrolera.

3. Kontroler tworzy obiekt Form Bean i umieszcza w nim wartości parametrów przekazanych z formularza.

4. Kontroler uruchamia akcję wskazaną w żądaniu. Decyzję o tym jaką akcję wywołać i jaki obiekt Form Bean dla niej wcześniej przygotować, kontroler podejmuje w oparciu o zawartość pliku konfiguracyjnego struts-config.xml.

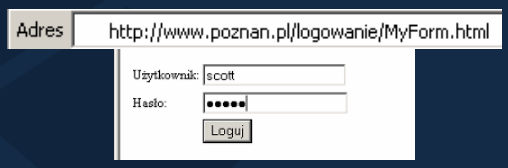
5. Jeśli akcja pozytywnie zweryfikuje dane podane przez użytkownika, to zwróci etykietę „success”, która spowoduje, że kontroler przekieruje użytkownika do strony Success.jsp.

6. W przeciwnym wypadku, akcja zwróci etykietę „failure”, która spowoduje, że kontroler przekieruje użytkownika do strony Failure.jsp.

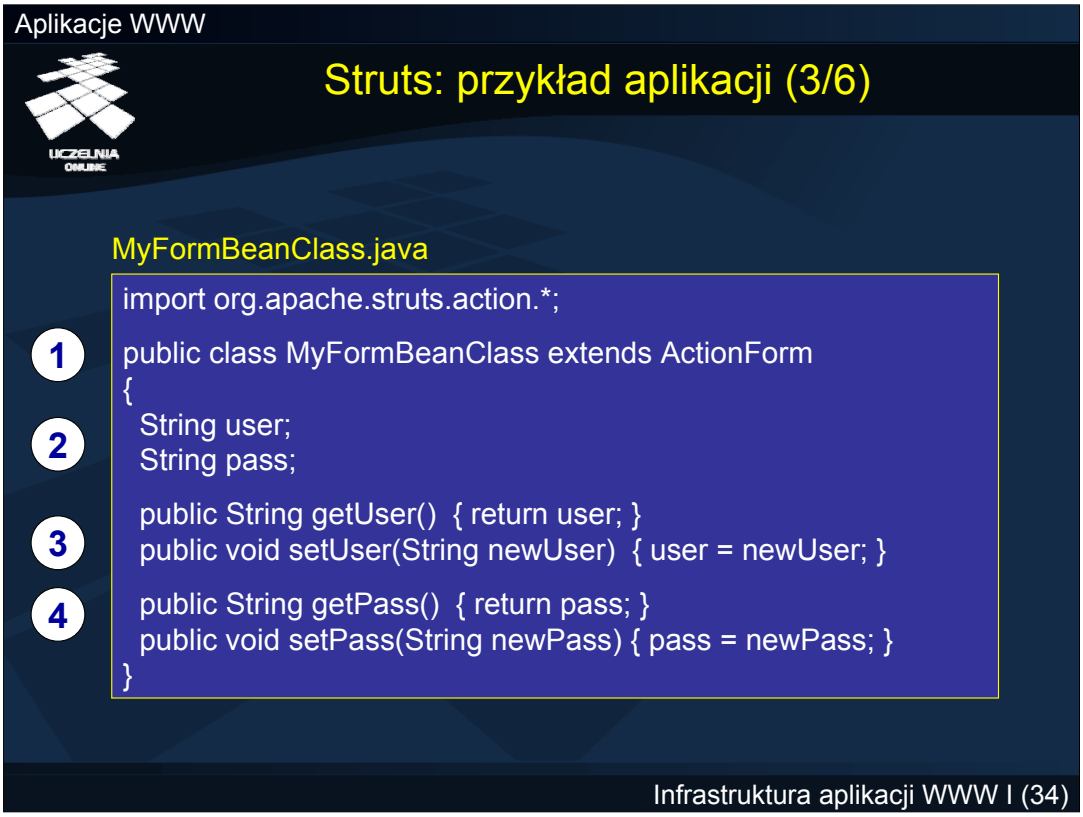
Struts: przykład aplikacji (2/6)

MyForm.html

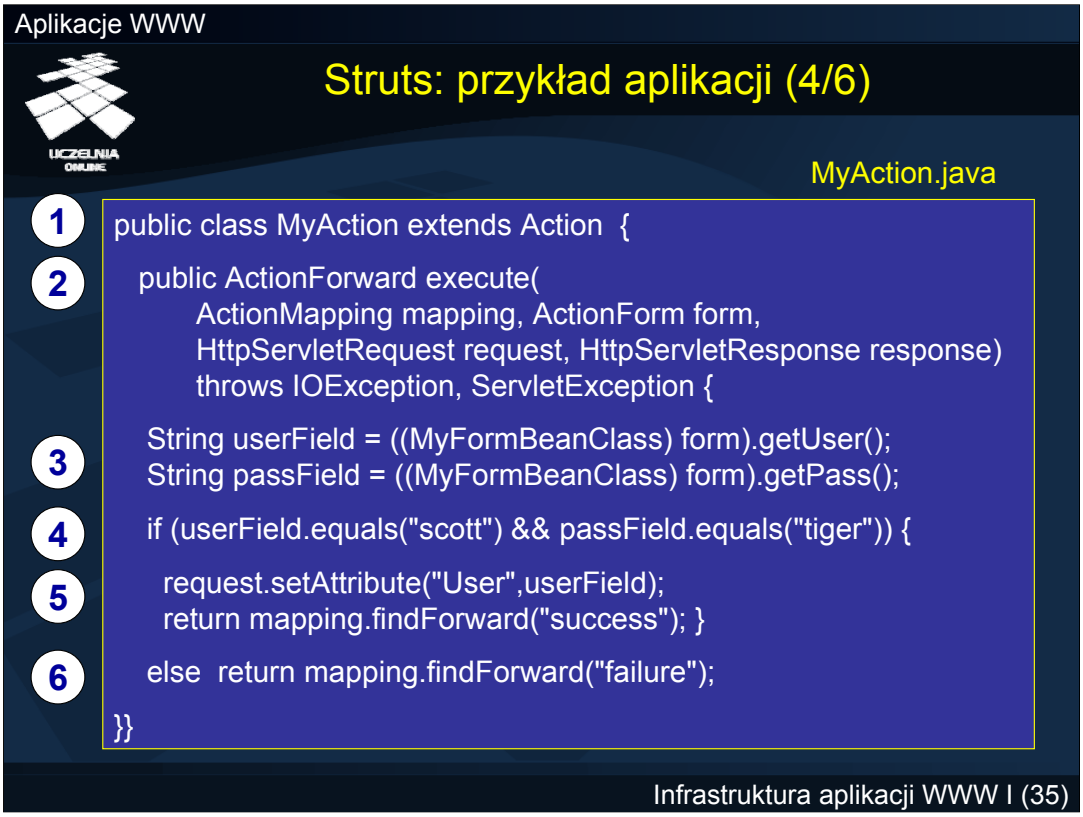
```
<FORM ACTION="MyAction.do">
Użytkownik: <INPUT TYPE="TEXT" NAME="user">
Hasło: <INPUT TYPE="PASSWORD" NAME="pass">
<INPUT TYPE="SUBMIT" VALUE="Loguj">
</FORM>
```



Slajd pokazuje formularz logowania. Jest to zwykły formularz HTML. Jego zatwierdzenie spowoduje wywołanie akcji Struts, co wskazuje wartość atrybutu ACTION elementu <FORM>: MyAction.do.



Slajd pokazuje kod klasy komponentu Form Bean, który będzie służył do przekazania do akcji parametrów wysłanych z formularza logowania przedstawionego na poprzednim slajdzie. Klasa dziedziczy z ActionForm (1), zawiera dwie właściwości: user i pass (2) odpowiadające parametrom z formularza oraz metody setXXX i getXXX dla obu właściwości (3)(4).



Slajd pokazuje kod klasy akcji (pominięte zostały dyrektywy import). Znaczenie wyróżnionych fragmentów kodu jest następujące:

1. Klasa akcji dziedziczy z Action.

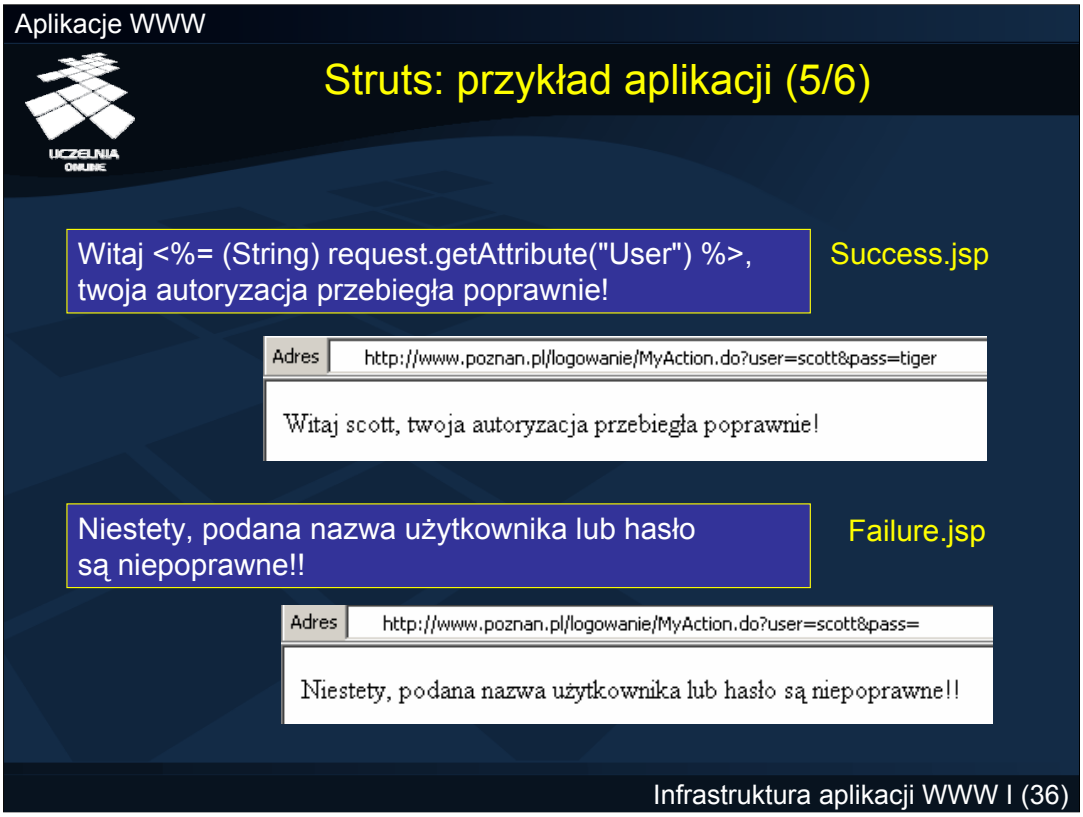
2. Logika akcji jest zawarta w metodzie execute(). Metodzie tej przekazane są następujące parametry: mapping – obiekt reprezentujący dostępne etykiety wynikowe, form – komponent Form Bean oraz udostępniane serwletom i JSP obiekty request i response reprezentujące żądanie i odpowiedź HTTP.

3. Działanie akcji rozpoczyna się od pobrania z komponentu Form Bean parametrów wywołania, czyli w naszym przypadku nazwy użytkownika i hasła.

4. W naszym przykładzie logowania akceptowany jest tylko użytkownik „scott” z hasłem „tiger”.

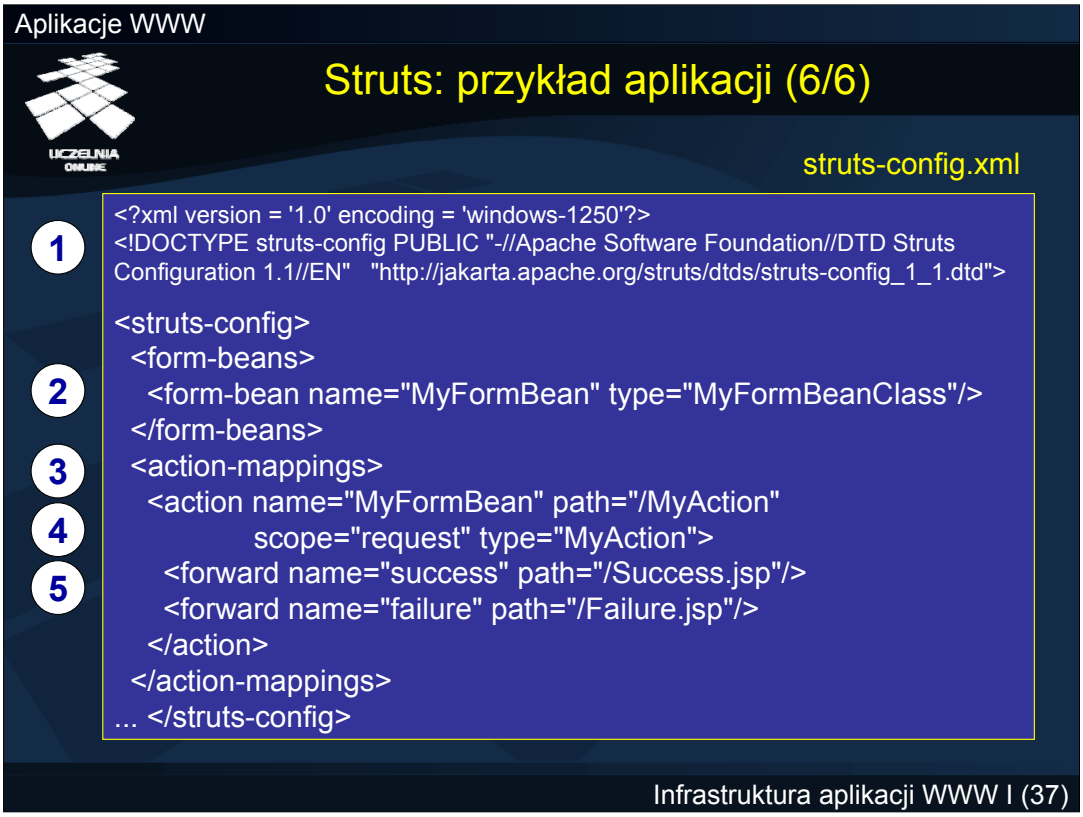
5. Jeśli dane użytkownika są poprawne, akcja rejestruje w zasięgu żądania nazwę użytkownika pod kluczem „User”, tak by była dostępna na stronie JSP, do której nastąpi przekierowanie. Następnie działanie metody akcji kończy się zwróceniem etykiety „success”.

6. W przypadku niepowodzenia logowania działanie metody akcji kończy się zwróceniem etykiety „failure”.



Slajd pokazuje kod stron JSP Success.jsp i Failure.jsp, do których następuje warunkowe przekierowanie z akcji w zależności od powodzenia logowania. Strona Success.jsp wyświetla nazwę zalogowanego użytkownika pobierając ją z zasięgu żądania, gdzie została umieszczona w akcji.

Ponieważ przekierowanie z akcji do strony JSP odbywa się domyślnie operacją forward po stronie serwera, adres w przeglądarce wskazuje na akcję, a nie na stronę JSP, która wygenerowała prezentowany w przeglądarce dokument HTML.



Elementem spinającym komponenty przykładowej aplikacji przedstawione na poprzednich slajdach jest plik konfiguracyjny aplikacji Struts pokazany na niniejszym slajdzie. Znaczenie wyróżnionych fragmentów pliku jest następujące:

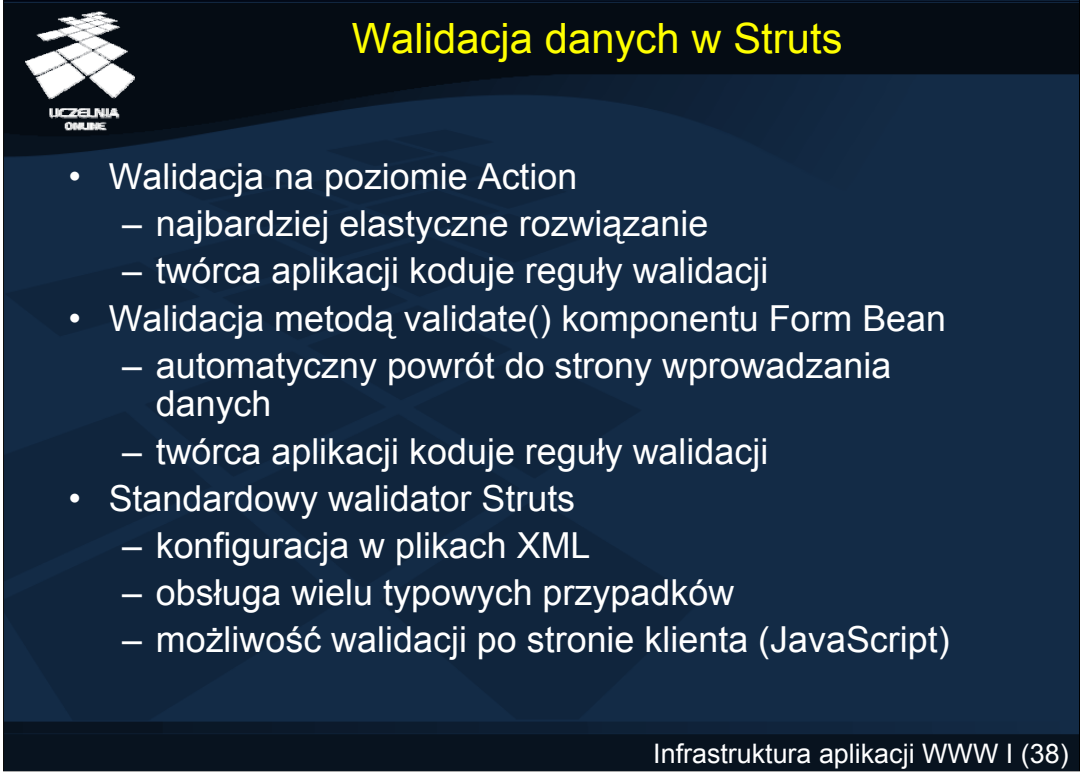
1. Prolog dokumentu XML, zawierający informację o typie dokumentu z odwołaniem do DTD.

2. Deklaracja komponentu Form Bean w sekcji <form-beans>.

3. Sekcja <action-mappings>, zawierająca deklaracje akcji i etykiet reprezentujących wyniki ich działania.

4. Deklaracja akcji, obejmująca nazwę komponentu Form Bean przekazującego parametry do akcji (name), ścieżkę URL wywołującą akcję (path), zasięg życia komponentu Form Bean (scope) i nazwę klasy akcji (type).

5. Etykiety reprezentujące możliwe wyniki działania definiowanej akcji. Każda etykieta ma nazwę (name) i adres URL, do którego ma nastąpić przekierowanie (path).

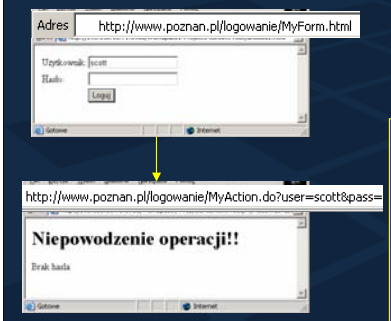


Struts oferuje wiele możliwości walidacji danych. Pierwszym sposobem jest walidacja w kodzie akcji. Jest to najbardziej elastyczne rozwiązanie, ale wymaga ręcznego kodowania reguł walidacji, a dodatkowo jeszcze ręcznego skonfigurowania powrotu do formularza wprowadzania danych w wypadku niepowodzenia walidacji.

Drugi sposób to walidacja na poziomie komponentu Form Bean. Komponenty Form Bean mogą posiadać metodę validate(), w której można ręcznie zakodować reguły walidacji. Zaletą skorzystania z metody validate() jest automatyczne przekierowanie do formularza wprowadzania danych, wskazanego w konfiguracji akcji, w wypadku niepowodzenia walidacji.

Trzeci sposób, wygodny w przypadku standardowej logiki walidacji, to wykorzystanie standardowego walidatora Struts. Jest to predefiniowany komponent walidatora, konfigurowany poprzez pliki XML, włączany do aplikacji jako wtyczka (ang. plug-in). Walidator ten obsługuje wiele typowych przypadków takich jak: obowiązkowość pól, weryfikacja długości łańcucha znaków, dopasowania do wzorca, zawierania się wartości w przedziale, walidacja numeru karty kredytowej, adresu e-mail i adresu url. Ważną zaletą standardowego walidatora Struts jest możliwość włączenia walidacji po stronie klienta, oprócz walidacji po stronie serwera. W tym wypadku, walidator Struts automatycznie wygeneruje odpowiedni kod JavaScript. Domyślnie walidacja po stronie klienta jest wyłączona.

Przykład walidacji metodą validate()



ApplicationResources.properties

```
error.nopass=Brak hasla<BR>
```

1

MyFormBeanClass.java

```
public ActionErrors validate(ActionMapping mapping,
HttpServletRequest request) {
   ActionErrors errs = new ActionErrors();
if (pass.equals("")) errs.add("noPass",
                        new ActionError("error.nopass"));
   if (errs.empty()) return null; else return errs;   }
```

2

ErrorPage.jsp

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<h1>Niepowodzenie operacji!!</h1>
<html:errors />
```

4

struts-config.xml

```
<action path="/MyAction" type="MyAction"
        name="MyFormBean" scope="request"
        input="/ErrorPage.jsp">  ...
```

3

Na slajdzie przedstawiony został przykład implementacji walidacji danych w aplikacji Struts w metodzie validate() komponentu Form Bean. Walidacja w przykładzie polega na sprawdzeniu czy zostało podane hasło. Implementacja walidacji wymaga realizacji kilku kroków:

1. Komunikat o błędzie należy zdefiniować w pliku zasobów. Domyślny, główny plik zasobów aplikacji Struts to ApplicationResources.properties (plik ten jest wskazany w pliku struts-config.xml). Ewentualne polskie znaki w pliku zasobów muszą być zakodowane w postaci ich kodów Unicode (ponieważ pliki .properties są plikami ASCII).

2. W klasie komponentu Form Bean należy zaimplementować metodę validate(). Metoda ta zwraca kolekcję błędów ActionErrors. Dodając błąd do tej kolekcji należy podać jego nazwę i przekazać obiekt ActionError związany z komunikatem o błędzie identyfikowanym przez jego klucz w pliku zasobów.

3. W definicji akcji w pliku struts-config.xml należy wskazać atrybutem INPUT stronę, do której ma nastąpić przekierowanie w przypadku niepowodzenia walidacji. Typowo jest to strona wprowadzania danych, ale niekoniecznie.

4. W naszym przykładzie w wypadku błędu walidacji ma nastąpić przekierowanie do strony z komunikatem o błędzie. Błędy walidacji Struts są wyświetlane znacznikiem <html:errors /> z jednej z bibliotek znaczników Struts (Struts Html). Istnieje również możliwość wyświetlania błędów pojedynczo, wskazując nazwę błędu atrybutem „property” znacznika <html:errors />.

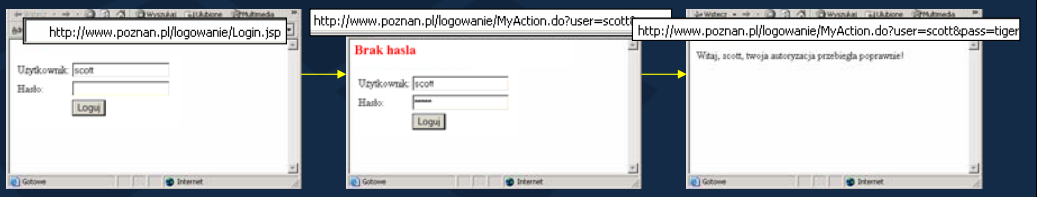
Stanowy formularz w Struts

Login.jsp

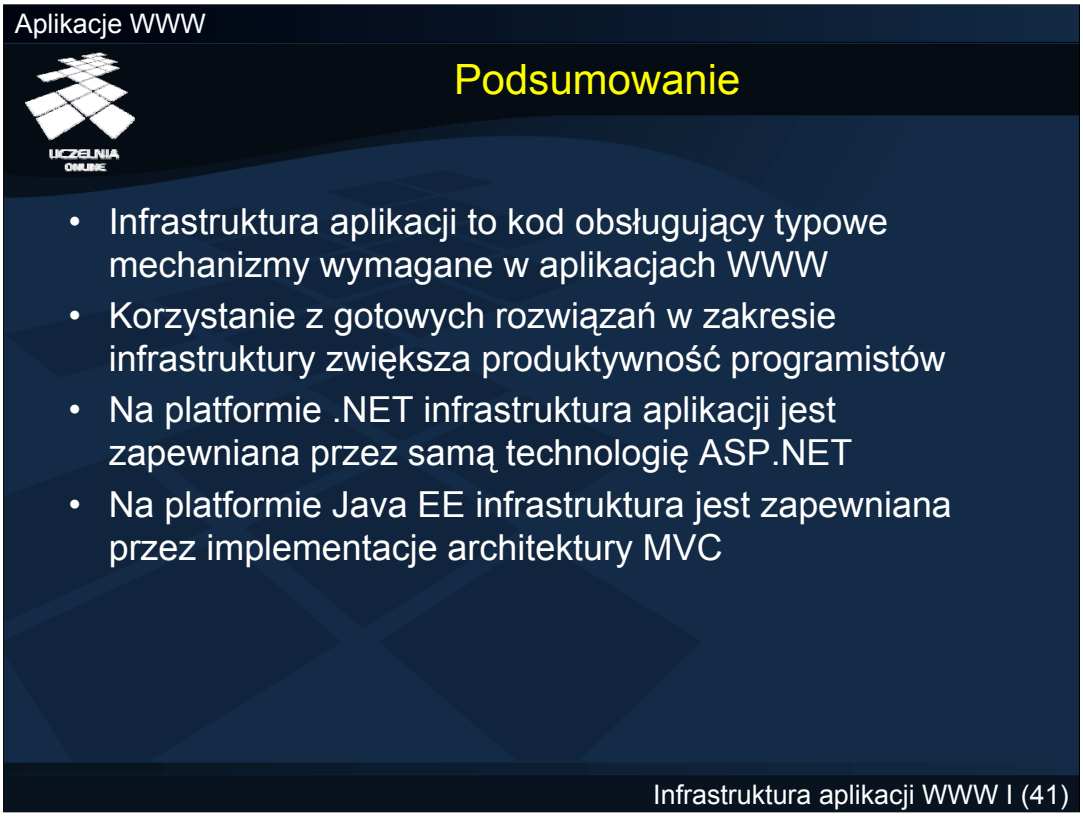
```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<html:form action="MyAction.do">
 <html:errors />
  Użytkownik: <html:text property="user" />
  Hasło: <html:password property="pass" />
 <html:submit value="Loguj" />
</html:form>
```

struts-config.xml

```
<action path="/MyAction" type="MyAction"
  name="MyFormBean" scope="request"
  input="/Login.jsp"> ...
```



Aby formularz HTML w aplikacji Struts zachowywał wprowadzone przez użytkownika dane po błędzie walidacji, musi on być utworzony za pomocą znaczników z biblioteki Struts Html, a nie zwykłych znaczników języka HTML. Na slajdzie pokazano zmodyfikowany formularz logowania, tym razem w postaci strony JSP wykorzystującej znaczniki Struts do utworzenia formularza i jego elementów. Adres formularza (/Login.jsp) został wskazany w pliku struts-config.xml jako adres formularza wprowadzania danych, do którego ma nastąpić przekierowanie w przypadku błędu walidacji w komponencie Form Bean związanym z akcją obsługującą logowanie.



Infrastruktura aplikacji to kod obsługujący typowe mechanizmy wymagane w aplikacjach WWW takie jak: stanowość interfejsu, obsługa nawigacji, czy walidacja danych.

Korzystanie z gotowych rozwiązań w zakresie infrastruktury pozwala twórcom aplikacji skupić się na logice specyficznej dla aplikacji, zwiększając ich produktywność.

Na platformie .NET infrastruktura aplikacji jest zapewniana przez samą technologię ASP.NET z jej architekturą Web Forms.

Na platformie Java EE infrastruktura jest zapewniana przez szkielety aplikacji implementujące architekturę Model-View-Controller. Najważniejsze szkielety aplikacji dla Java EE to Struts, omówiony w ramach niniejszego wykładu, i JSF, któremu poświęcony będzie następny wykład.

•.NET Framework Developer Center, http://msdn.microsoft.com/netframework/

•ASP.NET Quickstart Tutorial, http://www.asp.net/QuickStart/aspnet/

•Apache Struts, http://struts.apache.org/