

# Projektowanie aplikacji ADO.NET + ASP.NET

## Zestaw 5

Autentykacja, autoryzacja, Provider Model, własne formanty

23-11-2010

Liczba punktów do zdobycia: **10/45**

1. (**1p**) Zaprezentuj w praktyce przedstawiany na wykładzie mechanizm autentykacji **Windows**. Ściślej - przygotuj aplikację, w której użytkownik zostanie rozpoznany jako aktualnie zalogowany użytkownik systemu operacyjnego. Pokaż, że potrafisz sterować dostępem do poszczególnych zasobów aplikacji za pomocą mechanizmu autoryzacji (użytkownicy w odpowiednich rolach mają lub nie dostęp do wybranych stron).
2. (**1p**) Zaimplementuj i użyj we własnej aplikacji takiego dostawcę usługi uwierzytelniania (**MembershipProvider**), który potwierdzi tożsamość użytkownika w bazie danych Microsoft SQL Server, w tabeli **USERS**, w której zapisane będą nazwa użytkownika i SHA256 hasła. Zbuduj formularz dodawania użytkownika, który po utworzeniu konta poprawnie zapisze w tabeli **USERS** nazwę i skrót hasła.  
  
Logowanie do aplikacji oraz dodawanie użytkownika może być oparte o formanty biblioteczne, ale nie musi.
3. (**1p**) Poprzednie zadanie rozwiń o implementację usługi informowania o rolach, gdzie role zapisane byłyby w tabeli **ROLES**, a powiązanie wiele-do-wielu użytkowników z rolami w tabeli **USERSROLES**.  
  
Dostęp do zasobów można zabezpieczyć przez wskazanie ról użytkowników którzy mogliby do tych ról mieć dostęp. Pokaż, że można to robić zarówno dla pojedynczych zasobów (sekcja `location` w `web.config`) oraz całych podfolderów (osobny, zdegenerowany `web.config`).
4. (**1p**) Jak korzystać z informacji o rolach użytkowników w aplikacji?  
  
Pokaż, że potrafisz zablokować dostęp do podglądu i edycji **wybranego** wiersza **ListView** dla użytkowników będących w konkretnej roli.  
  
Na przykład pole **PESEL** powinien widzieć każdy, a edytować tylko użytkownik będący w roli **ADMINISTRATOR**, zaś pole **PENSJA** powinien widzieć i edytować tylko użytkownik w roli **PLACOWA**.
5. (**1p**) Pokaż, że potrafisz posługiwać się sekcją **UserData** ciastka **Forms**. Ściślej - napisz takiego dostawcę usługi informowania o rolach, który listę ról użytkownika zapamięta w sekcji **UserData** ciastka **Forms** w momencie logowania, a przy każdym żądaniu dostarczenia listy ról będzie wydobywał je z tej sekcji.  
  
Zadanie to ma ma celu oswojenie się ze strukturą ciastka **forms** oraz interfejsem, który pozwala na jego tworzenie oraz na dostęp do informacji w nim zawartych (klasa **FormsAuthenticationTicket**).

6. (1p) Udowodnij, że sposób uwierzytelniania Forms jest ogólniejszy niż Windows. Ścisłej - napisz takiego dostawcę usługi uwierzytelniania, który sprawdzi tożsamość użytkownika w systemie operacyjnym (formalnie - we wskazanej domenie).
- Wskazówka. Do potwierdzenia tożsamości użytkownika należy użyć protokołu LDAP, do którego dostęp mamy za pomocą obiektów `DirectoryEntry` i `DirectorySearcher`. Odpowiedni kod prawie na pewno znajdziesz na sieci. Nie wolno korzystać z bibliotecznej klasy `ActiveDirectoryMembershipProvider`.*
7. (1p) Korzystając z przykładu z wykładu napisz własnego dostawcę usługi "stopka/nagłówek", którego zadaniem będzie dostarczanie kodu html nagłówka i stopki witryny.
- Zaprojektuj infrastrukturę dla takiego dostawcy oraz dostarcz dwu różnych implementacji, np.
- pierwsza ma zwracać statyczne kawałki htmla, które może mieć zaszyte nawet w kodzie
  - druga ma zwracać wynik odwołania się do zewnętrznego zasobu `WebService`
- Pokaż, że potrafisz wymieniać dostawcę usługi bez rekompilacji aplikacji, a jedynie konfigurując dostawcę w `web.config`.
8. (1p) Utwórz własną kontrolkę użytkownika (`*.ascx`), `Login`, która będzie zawierać pola tekstowe **Nazwa użytkownika** i **Hasło** oraz przycisk **Loguj**.
- Po naciśnięciu przycisku kontrolka powinna symulować uwierzytelnianie użytkownika używając skonfigurowanego dostawcy usługi uwierzytelniania i w wypadku pomyślnego uwierzytelnienia przekierowywać kontekst do innej strony lub informować aplikację podnosząc jakieś zdarzenie.
9. (1p) Utwórz własną kontrolkę dziedziczącą z `WebControl`, która w zależności od stanu wartości `HttpContext.Current.User` będzie wypisywać informację "**Niezałogowany**" lub **Witaj: xyz**, gdzie `xyz` będzie nazwą użytkownika odczytaną z `HttpContext.Current.User`
10. (1p) Wzorując się na przykładzie z wykładu (walidator parzystości) napisać własny walidator działający po stronie klienta `PeselValidator`, który będzie walidował poprawność numeru PESEL wprowadzonego do pola tekstowego.
- Uwaga! Algorytm weryfikacji poprawności numeru PESEL jest opisany w sieci. Proszę również pamiętać o poprawnej implementacji walidacji po stronie serwera, tak aby właściwość `IsValid` zwracała poprawną wartość.*
- Uwaga! Walidator powinien działać tak, jak każdy inny walidator, tzn. nie dopuszczać do przesłania na serwer formularza w sytuacji niewłaściwej walidacji pola, do którego jest doczepiony. Walidator powinien poprawnie obsługiwać typowe właściwości, takie jak `ErrorMessage` czy `ValidationGroup`.*
- Wskazówka! Podobnie jak w przykładzie z wykładu głównym elementem walidatora będzie odpowiednio skonstruowana funkcja JavaScript przywiązana do zdarzeń pola tekstowego po stronie klienta.*

Wiktor Zychla