Projektowanie aplikacji ADO.NET + ASP.NET
Zestaw 1

Podstawy ASP.NET

04-10-2016

Liczba punktów do zdobycia: **10/10**
Zestaw ważny do: 25-10-2016

1. (**3p**) Przygotować przykładową aplikację z jedną stronę `*.aspx`, a w niej (formalnie: w jej kodzie serwerowym C#) spróbować otworzyć do odczytu wskazany plik tekstowy z dysku serwera (**StreamReader sr = ...**). Zawartość pliku wypisać na stronie. Jako serwer aplikacji wskazać IIS Express.

   Następnie na serwerze IIS 7/8 założyć dwie różne witryny w różnych fizycznych lokalizacjach na dysku. Dla każdej z witryny utworzyć osobną pulę aplikacji. Jako tożsamości pul aplikacji utworzyć i przypisać dwa **różne** konta w systemie operacyjnym.

   Pokazać jak przeprowadzić deployment aplikacji rozwijanej lokalnie przy pomocy IIS Express na pełny serwer IIS.

   Pokazać, że w przypadku IIS próba otwarcia pliku w kodzie serwerowym wymaga nadania jawnie w systemie operacyjnym uprawnień do pliku (lub foldera zawierającego plik) dla konta określającego tożsamość puli aplikacji (dlaczego tak się dzieje?).

   Pokazać co się dzieje w przypadku niewystarczających uprawnień (konto puli aplikacji nie ma uprawnień dostępu do pliku).

   Dlaczego nie jest dobrą praktyką przełączanie tożsamości puli aplikacji na LocalSystem?

   Którąś z utworzonych witryn udostępnić na dwóch różnych nagłówkach hosta (np. `ap1.myserver.com` i `ap2.myserver.com`). Pokazać, że witryna działa poprawnie adresowana na dwa różne sposoby. Nauczyć się korzystać z lokalnej mapy hostów (`Windows/System32/Drivers/etc/hosts`).

2. (**1p**) W konfiguracji witryny na serwerze aplikacji umieć wskazać miejsce definiowania filtrów kojarzących rozszerzenia zasobów z logiką ich przetwarzania. Co się stanie jeśli na serwerze aplikacji umieścimy zasób o rozszerzeniu, na które nie mapuje się żaden filtr, a następnie spróbujemy do serwera wysłać żądanie o wydanie tego zasobu?

   Zademonstrować to na przykładzie eksperymentu z zasobem o rozszerzeniu np. ***.foo**: utworzyć w aplikacji zasób o takim rozszerzeniu i spróbować go pobrać z poziomu przeglądarki.

3. (**1p**) Nauczyć się korzystać z debuggera warstwy TCP (np. Wireshark lub Microsoft Network Monitor) do podglądania ruchu klient-serwer na poziomie protokołów transportowych.

   Pokazać jak podglądać ramki TCP/IP i jak podglądać komunikację za pomocą protokołu HTTP.

4. (**1p**) Nauczyć się korzystać z debuggera warstwy HTTP, Fiddlera (`http://www.telerik.com/fiddler`).

   Pokazać jak przechwytywać ruch z przeglądarki do internetu - jak podglądać zawartości żądań i odpowiedzi i jak stawiać pułapki i za ich pomocą **modyfikować** żądania i/lub odpowiedzi (zakładka Inspectors).

   Pokazać jak za pomocą Fiddlera symulować żądania typu GET i żądania typu POST do własnej strony ASP.NET (czyli jak wyklikać w Fiddlerze żądanie, które do naszej aplikacji wyśle Fiddler, a nie przeglądarka, zakładka Composer!).

   Czym różni się żądanie GET od POST po stronie Fiddlera? Czym różnią się takie żądania po stronie kodu aplikacji ASP.NET? Pokazać jak w kodzie aplikacji po stronie serwerowa odczytać parametry przesłane przez GET a jak te przesłane przez POST.

5. (**1p**) Nauczyć się korzystać z debuggera warstwy HTTP, Burpa (`https://portswigger.net/burp/`).

   Pokazać jak przechwytywać ruch z przeglądarki do internetu - jak podglądać zawartości żądań i odpowiedzi i jak stawiać pułapki i za ich pomocą **modyfikować** żądania i/lub odpowiedzi (zakładka Proxy/Options, Proxy/Intercept).

   Pokazać jak za pomocą Burpa symulować żądania typu GET i żądania typu POST do własnej strony ASP.NET (czyli jak wyklikać w Burp żądanie, które do naszej aplikacji wyśle Burp, a nie przeglądarka, zakładka Repeater!).

6. (**1p**) Ucząc się języka HTML przyzwyczajamy się do tego, że dla żądań typu GET z przeglądarki przeznaczony jest odsyłacz (link)

   ```
   <a href="strona.aspx?p1=v1&p2=v2">żądanie GET</a>
   ```

   zaś dla żądań typu POST - przycisk

   ```
   <form method="post">
     <input type="text" name="p1" />
     <input type="text" name="p2" />
     <input type="submit" value="żądanie POST" />
   </form>
   ```

   Zadanie polega na tym, żeby pokazać jak to zrobić odwrotnie - to odsyłacz (link) powinien spowodować żądanie typu POST a przycisk żądanie typu GET.

7. (**2p**) Przygotować aplikację ASP.NET WebForms, która pozwala wprowadzić i wydrukować standardowy "pasek zgłoszenia zadań".

   Aplikacja ma składać się z dwóch stron `*.aspx`: formularza zgłoszenia i formularza wydruku.

   Na formularzu zgłoszenia użytkownik aplikacji powinien mieć możliwość wpisania imienia i nazwiska, daty, nazwy zajęć i numeru zestawu oraz kompletu wyników kolejnych deklarowanych 10 zadań z odpowiednią liczbą punktów. Program powinien kontrolować poprawność wpisywanych danych.

   Po zaakceptowaniu formularza zgłoszenia, użytkownik powinien w przeglądarce zobaczyć formularz wydruku: pasek zgłoszenia w postaci możliwej do natychmiastowego wydrukowania.

   Użyć dowolnej, wybranej przez siebie metody przekazywania danych między stronami (GET/POST, ciastka, kontenery serwerowe).