

Projektowanie aplikacji ASP.NET

Zestaw 3

Podstawy ASP.NET

2024-10-22

Liczba punktów do zdobycia: **10/26**

Zestaw ważny do: 2024-11-05

1. (**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 powinien powodować żądanie typu POST a przycisk żądanie typu GET.

*Wskazówka! Celem tego zadania jest zapoznanie się z wycinkiem interfejsu programistycznego Javascript w przeglądarce. W obu przypadkach należy bowiem wspomóc się odpowiednim kodem Javascript. I tak, żeby odsyłacz powodował żądanie typu POST, należy dodać mu obsługę zdarzenia **onClick** i wewnątrz kodu obsługi zdarzenia pozyskać referencję do obiektu formularza, a następnie wykonać metodę **submit** formularza. Żeby przycisk powodował żądanie typu GET należy dodać mu obsługę zdarzenia **onClick** i wewnątrz kodu obsługi zdarzenia wymusić zmianę bieżącej strony przeglądarki przez przypisanie wartości do **document.location.href**.*

2. (**1p**) Na przykładowej aplikacji pokazać jak dodawać, odczytywać i usuwać ciastka (**HttpCookie**) w kodzie po stronie serwera.

Jak naiwnie sprawdzić czy przeglądarka obsługuje ciastka? Ale jak to zrobić **wiarygodnie**, tzn. mieć pewność że przeglądarka w danej sesji obsługuje ciastka?

3. (**1p**) Przystudiować interfejsy (API) obiektów **Request**, **Server** i **Response**. Nauczyć się odczytywać nagłówki (Headers) żądania. Nauczyć się tworzyć własne nagłówki odpowiedzi. Nauczyć się mapować ścieżki względne (względem korzenia lokalizacji witryny na dysku) na ścieżki fizyczne (**Server.MapPath**).

Do czego przydaje się statyczna właściwość **HttpContext.Current**?

4. (1p) Nauczyć się różnic między kontenerami serwerowymi **Application**, **Session** i **Items**. Zademonstrować ich użycie w kodzie po stronie serwera. Zapoznać się z pewną szczególną techniką wykorzystania kontenerów serwerowych, w której obiekt o czasie życia kontrolowanym za pomocą kontenera, jest w interfejsie programistycznym udostępniony tak jakby był *singletonem*.

Por. <https://www.wiktorzychla.com/2010/12/container-based-pseudosingletons-in.html>

Dostęp do którego kontenera powinien być dodatkowo chroniony przed problemem dostępu z wielu równoległych wątków (**lock**) w akcesorze dostępu (**get**) i dlaczego? Którego kontenera mogłoby w ogóle nie być w interfejsie programistycznym, bo jest on równoważny pewnemu innemu prostemu mechanizmowi samego języka programowania?

5. (1p) Bez względu na sposób dostępu do danych, z kontekstu dostępu do danych zwykle korzystamy przy pomocy jakiegoś dedykowanego obiektu (**SqlConnection**, LINQowy **DataContext**, Hibernate'owy **ISession** itp).

Który kontener serwerowy jest najwłaściwszy do przechowywania takiego kontekstu dostępu do danych (**Application**, **Session** czy **Items**)? Jakie skutki uboczne miałyby przechowywanie kontekstu dostępu do danych nie w tym **właściwym**, ale w którymś z pozostałych kontenerów (co oczywiście technicznie jest możliwe)?

Zademonstrować kod, który opakowuje kontekst dostępu do danych w pseudosingleton przy wykorzystaniu tego właściwego kontenera.

W którym miejscu potoku przetwarzania poprawnie zwalniać zasoby kontekstu dostępu do danych (co jest istotne zwłaszcza wtedy jeśli obiekt dostępowy implementuje interfejs **IDisposable**)? Zademonstrować to zwalnianie na przykładzie.

6. (1p) Do czego służy plik **app_offline.htm**?
7. (2p) Przećwiczyć w praktyce przesyłanie danych binarnych w obie strony. Ściślej - wykonać aplikację, która pozwoli użytkownikowi wskazać plik lokalny na dysku i przesłać go na serwer (formant `<input type="file" ... />`), a po stronie serwera zostanie wyprodukowany i odesłany plik tekstowy (xml lub json lub nawet zwykły tekst):

```
<opis>
  <nazwa>nazwaprzesłanego pliku</nazwa>
  <rozmiar>rozmiarprzesłanego pliku</rozmiar>
  <sygnatura>suma bajtów pliku modulo 0xFFFF</sygnatura>
</opis>
```

Plik tekstowy powinien być budowany dynamicznie i odesłany do klienta **bez** zapisywania jego zawartości na dysku serwera. W przeglądarce użytkownika nadesłana odpowiedź powinna spowodować podniesienie się domyślnego okna Otwórz/Zapisz/Anuluj.

Przestudiować specyfikację semantyki nagłówka **Content-Disposition**. W jaki sposób określać nazwę zwracanego zasobu? Jak decydować o tym czy zwracany zasób ma być potraktowany jako zewnętrzny plik a jak zasugerować przeglądarce że jest to zasób do pokazania w oknie przeglądarki? Jak wykorzystać specyfikację RFC5987 do wykorzystywania znaków UTF-8 w nazwach zwracanych zasobów? I jak poradzić sobie w sytuacji kiedy przeglądarka nie obsługuje poprawnie tej specyfikacji?

8. (2p) Wykorzystać obiekt sesji do następującego **naiwnego** rozwiązania problemu autentykacji: w każdej stronie aplikacji w wypadku stwierdzenia w kodzie zdarzenia **Page_Load**

braku informacji o użytkowniku w sesji, kontekst przetwarzania przekierowywany jest do strony `Login.aspx`, na której po poprawnym potwierdzeniu tożsamości (login i hasło) informacja o tożsamości zapamiętywana jest w kontenerze sesji.

Dodatkowo punkt wejścia do aplikacji powinien być poprawnie przywrócony po autentykacji (użytkownik kieruje żądanie do którejkolwiek strony aplikacji, jest przekierowany do strony logowania, a następnie aplikacja sama powraca do strony, od której użytkownik chciał rozpocząć nawigację).

*Uwaga! O tym jak **poprawnie** implementować mechanizm autentykacji i autoryzacji będziemy rozmawiać na kolejnych wykładach. To zadanie ma wyłącznie pokazać, że referencyjne rozwiązanie, które poznamy w przyszłości, nie jest jedynym możliwym. W praktyce nie powinno się autentykacji opierać na sesji serwera (dlaczego?).*

Wiktor Zychla