Projektowanie aplikacji ASP.NET
Zestaw A

Zadania dodatkowe

2025-01-07

Zestaw ważny do końca semestru

Uwaga! Zestaw zawiera zadania dodatkowe, niewymagane, ale punktowane. Sam zestaw nie jest wliczany do maksimum punktowego, stąd proszę potraktować poniższe zadania jako okazję do zdobycia dodatkowych punktów, w sytuacji w której z różnych powodów nie udało się zdobyć odpowiedniej liczby punktów na dotychczasowych listach.

1. (**1p**) (**Autocomplete**) Zademonstrować w praktyce użycie atrybutu `autocomplete` formantu `input` (w dowolnym typie aplikacji, może być MVC). W rozwiązaniu pokazać i omówić co najmniej 10 różnych rodzajów, w tym zwrócić uwagę na działanie dwóch: `current-password` i `new-password`.

2. (**1p**) (**Potwierdzenie usuwania**) Usuwanie danych z poziomu interfejsu użytkownika zawsze rodzi wątpliwość taką oto czy nie zostało aktywowane przez użytkownika przypadkiem. Stąd standardowo wymaga się potwierdzenia usuwania. Dostępny po stronie przeglądarki `window.confirm` jest używany rzadko, ponieważ renderuje fragment interfejsu użytkownika "poza stroną". Lepsze podejście to ukryty fragment formularza, który aktywuje się dopiero po wybraniu przez użytkownika funkcji usuwania.

   W zadaniu należy zademonstrować taki mechanizm. Ściśle: użytkownik ogląda jakiś fragment danych (np. zamówienie) i gdzieś w interfejsie jest dostępny na stronie przycisk usuwania. Jego kliknięcie nie odsyła od razu strony na serwer, zamiast tego odsłania niewidoczny do tej pory fragment strony, na którym widać prośbę o potwierdzenie i dopiero wtedy widać przycisk który odsyła formularz na serwer (a na serwerze następuje usunięcie elementu).

3. (**2p**) (**Dwukrokowy formularz logowania**) Standardowy formularz logowania zawiera dwa pola tekstowe (nazwa użytkownika/hasło) i przycisk submitujący.

   Pokazać tzw. **dwukrokowy** formularz logowania. Taki formularz ma tylko pole nazwy użytkownika i przycisk do przejścia do drugiego kroku. Drugi krok to inny fragment formularza, w którym użytkownik podaje już tylko hasło i dopiero w drugim kroku jest przycisk submitujący. Przejście między oboma częściami formularza odbywa się wyłącznie po stronie klienta i jest realizowane chowaniem/pokazywaniem fragmentu strony (np. ustawieniem `display: none` w stylu odpowiedniej części formularza.

   Dwukrokowy formularz logowania ma następującą przewagę nad zwykłym: po wprowadzeniu nazwy użytkownika w pierwszym kroku, przeglądarka może wykonać dodatkowe żądanie do serwera (metoda `fetch` z poziomu Javascript strony) i sprawdzić czy użytkownik o podanej nazwie istnieje a jeśli tak, to czy jego logowanie wymaga spełnienia jakichś dodatkowych warunków.

Przykładowo: jeśli użytkownik ma skonfigurowane uwierzytelnianie dwuskładnikowe, to można w drugiej części formularza aktywować od razu możliwość wprowadzenia drugiego składnika. Inny przykład: jeśli użytkownik o takiej nazwie ma nieudaną poprzednią próbę logowania, to w drugiej części formularza oprócz wprowadzenia hasła można aktywować obowiązkowy składnik typu captcha.

W oczekiwanym rozwiązaniu tego zadania nie trzeba pokazywać tych dodatkowych rzeczy związanych z dwukrokowym formularzem logowania (logowanie dwuskładnikowe, captcha), wystarczy sam dwukrokowy fomualrz logowania.

4. (**5p**) (**Kupon rabatowy**) Aplikacja posiada formularz podsumowania zamówienia, na którym użytkownik widzi swoje zamówienie, jego cenę i przycisk potwierdzenia zamówienia. Po kliknięciu przycisku potwierdzenia, zamówienie jest zapisywane po stronie serwera, dla zalogowanego użytkownika. Zapisane w bazie zamówienie zawiera cenę.

   Formularz podsumowania zamówienia ma jednak dodatkowe wymaganie - możliwość wprowadzenia kodu rabatowego. Kod rabatowy obniża cenę zamówienia. Po stronie serwera, w bazie, znajduje się dodatkowa tabela z kodami rabatowymi, przy każdym kodzie rabatowym znajduje się informacja o kodzie, procencie rabatu i znacznik czy kod został już użyty.

   Zadanie polega na takim rozszerzeniu formularza podsumowania zamówienia, które pozwala na wprowadzenie takiego kodu rabatowego, ale muszą być spełnione warunki:

   - formularz powinien posiadać dwa przyciski - dotychczas istniejący przycisk potwierdzenia zamówienia (który zapisuje zamówienie) i nowy przycisk sprawdzenia poprawności kodu rabatowego
   - kliknięcie przez użytkownika przycisku sprawdzenia poprawności kodu powinno sprawdzić prawidłowość kodu rabatowego (tylko serwer wie czy wprowadzony kod jest prawidłowy) i jeśli kod jest prawidłowy to na formularzu podsumowania użytkownik powinien zobaczyć zaktualizowaną informację o cenie zamówienia (stara cena przed zastosowaniem kuponu i nowa cena po zastosowaniu kuponu)
   - użytkownik może wielokrotnie używać przycisku sprawdzenia poprawności kodu, w tym - może za każdym razem zmieniać zawartość pola z kodem rabatowym, wprowadzać prawidłowe lub nieprawidłowe kody. Za każdym razem kliknięcie przycisku sprawdzenia powinno prawidłowo aktualizować informację o podsumowaniu zamówienia - pokazywać albo informację że kod rabatowy jest prawidłowy (i pokazać starą/nową cenę) lub pokazać informację że kod jest nieprawidłowy lub pusty (i wtedy wrócić do oryginalnej ceny zamówienia)
   - niezależnie od tego czy i ile razy użytkownik kliknął przycisk sprawdzenia kodu rabatowego, w dowolnym momencie może kliknąć przycisk potwierdzenia zamówienia i wtedy do bazy należy zapisać taką cenę zamówienia jaka wynika z ostatniego sprawdzenia kodu rabatowego (czyli cenę obniżoną lub wyjściową, w zależności od tego czy kod rabatowy jest prawidłowy czy nie)
   - informacja o zastosowaniu kodu nie powinna być możliwa do "zhakowania", czyli użytkownik nieposiadający kodu rabatowego nie powinien móc obniżyć ceny zamówienia preparując w jakiś szczególny sposób żądanie na serwer (dokładając ciasteczko, parametry do formularza itp.)
   - kodu rabatowego można użyć tylko raz i po złożeniu zamówienia należy go oznakować jako zużyty. To oznacza ze należy zabezpieczyć się przed scenariuszem w którym użytkownik A na formularzu stosuje kod rabatowy ale jeszcze nie składa zamówienia,

uzytkownik B stosuje ten sam kod rabatowy (kod jest jeszcze nie zużyty bo uzytkownik A nie złożył zamówienia!) po czym obaj użytkownicy składają zamówienie i obaj korzystają z obniżki - jest to nadużycie, przed którym należy się zabezpieczyć

5. (**5p**) (**Captcha**) W tym zadaniu należy zaproponować mechanizm typu captcha. Należy wykonać własne rozszerzenie MVC, renderowane tak jak inne rozszerzenia (`Html.Captcha`), które w żądaniu typu GET renderuje captchę a w żądaniu typu POST nie tylko renderuje captchę ale też sprawdza czy *odpowiedź* użytkownika jest prawidłowa i daje programiście możliwość odróżnienia stanów (prawidłowa/nieprawidłowa) odpowiedź.

   Wskazówki: w żądaniu POST w którym użytkownik oddaje formularz na serwer i serwer spodziewa się rozwiazanej captchy, żeby w kodzie akcji kontrolera programista miał możliwość podjęcia decyzji (prawidłowa/nieprawidłowa), weryfikacja musi odbywać się przed renderowaniem. Dlatego o ile do renderowania captchy na stronie przygotowuje się rozszerzenie (`Html.Captcha`) o tyle do sprawdzania rozwiązania przygotowuje się dodatkowy, własny **filtr akcji**, umieszczany nad akcją kontrolera (podobnie jak chociażby **Authorize**). Taki filtr - jak pamiętamy - jest uruchamiany przed wykonaniem akcji a w którymś elemencie infrastruktury (np. kontener `Items`) pozostawia po sobie informację o tym czy rozwiązanie captchy jest prawidłowe czy nie. Istnieje jeszcze jedno ładne rozwiązanie - filtr akcji ma dostęp do kolekcji parametrów przekazywanych do funkcji implementującej akcję kontrolera i może tę kolekcję modyfikować, na przykład dodając do niej klucz-wartość czyli parametr akcji z wartością).

   Jeśli chodzi o rodzaj captcha to istnieje szereg możliwości: począwszy od prostych typu zadania arytmetyczne (łatwe do obchodzenia przez automaty) przez captche obrazkowe (trudniejsze) po captche wymagające interakcji użytkownika (na przykład przesuwania/klikania myszą w odpowiednie miejsca). Proszę użyć inwencji, poszukać w sieci przykładów i zaproponować jakieś interesujące podejście.

Wiktor Zychla