Laboratorium 6 – JavaScript

Połącz projekt interfejsu użytkownika menedżera ocen studentów z poprzednich zajęć z usługą REST ze wcześniejszych zajęć wykorzystując w tym celu język JavaScript oraz biblioteki jQuery i KnockoutJS 3.x oraz mapping plugin. Zachowaj pełną funkcjonalność wcześniejszych projektów!

Wprowadzenie

jQuery – to biblioteka programistyczna dla języka JavaScript, której głównym zadaniem jest rozszerzanie standardowego API przeglądarki o dodatkowe funkcje, implementacja (niektórych) nowych API w starszych przeglądarkach, oraz normalizacja różnic w działaniu przeglądarek. Zapoznaj się z podstawową funkcjonalnością w samouczku.

KnockoutJS – to biblioteka szablonów treści HTML dla aplikacji działających w architekturze MVVM (Model-View-ViewModel). W architekturze tej poza głównym modelem danych aplikacji, wyróżnia się model danych specjalizowany dla widoku (interfejsu użytkownika). KnockoutJS pozwala na deklaratywne powiązanie obiektów w modelu widoku z ich prezentacją (znacznikami HTML i/lub regułami CSS). Zapoznaj się z podstawową funkcjonalnością w samouczku.

Mapping plugin – to dodatek do KnockoutJS, który pozwala na automatyczną konwersję obiektów JSON (np.: pobranych z serwera) na obiekty obserwowalne (JavaScript z funkcjami) wykorzystane przez bibliotekę KnockoutJS. Zapoznaj się z podstawową funkcjonalnością w samouczku.

Zadania szczegółowe

Grupa zadań K (1p):

1. Pobieranie i zapisywanie danych na serwerze
   a. Do usługi REST dodaj klasę CustomHeaders i zarejestruj ją podczas uruchamiania, przy użyciu metody `ResourceConfig.register()`. Klasa ta dodaje nagłówki CORS do odpowiedzi usługi pozwalające na odwołania do usługi ze skryptów na stronach internetowych znajdujących się na innych domenach i/lub portach niż usługa REST. Jeśli korzystasz z innej technologii niż JAX-RS, dodaj odpowiednie nagłówki w sposób zgodny z tą technologią.
   b. Pobierz pliki *.js w/w bibliotek, umieść je w katalogu aplikacji i podłącz do dokumentu HTML znacznikami `<script type="text/javascript" src="{ścieżka}"></script>`. Zwróć uwagę, że znacznik `<script>` nie może być zamknięty w miejscu, tzn. zapis `<script/>` jest niepoprawny, a skrypt podłączony w ten sposób nie zostanie uruchomiony. Zwróć uwagę również, że podłączenie zewnętrznego skryptu z użyciem atrybutu `src` wyklucza się z osadzeniem kodu JS wewnątrz znacznika `<script>`.
   Zastanów się nad wadami i zaletami serwowania plików bibliotek bezpośrednio z serwera aplikacji oraz z publicznych serwerów CDN firm trzecich.
   c. Utwórz własny plik *.js, umieść go w katalogu aplikacji i podłącz do dokumentu HTML. Wszystkie skrypty aplikacji powinny być umieszczone w tym pliku. Na początku pliku umieść dyrektywę `"use strict";` przełączającą silnik JS w tryb ścisły.
   d. Utwórz generyczny obiekt tablicy obiektów dziedziczący po `ko.observableArray` pozwalający na pobieranie danych z serwera, aktualizację pojedynczej krotki (obiektu) na serwerze, dodanie nowej krotki (obiektu) na serwerze oraz usunięcie pojedynczej krotki (obiektu) na serwerze. Zmiany realizowane na serwerze powinny odpowiadać widokowi danych w przeglądarce. Wykorzystaj funkcję `$.ajax()` do komunikacji z serwerem oraz funkcjonalność ko.mapping do konwersji obiektów JSON pobranych z serwera do obiektów obserwowalnych. Pamiętaj o ustawieniu nagłówków żądania:
   `Content-Type: application/json`

`Accept: application/json`
zależnie, czy dane są wysyłane do serwera, czy pobierane.

e. Utwórz model widoku (viewmodel) dla wszystkich widoków swojej aplikacji. Możesz utworzyć jeden wspólny model (zalecane), lub wiele oddzielnych modeli.
f. Umieść w dokumencie HTML szablony (template) i powiązania do modelu (binding), określające deklaratywnie sposób wypełnienia dokumentu HTML danymi z modelu (tabele i nagłówki, np.: w przypadku listy ocen studenta, niech jego dane pojawią się w nagłówku tabeli). Pamiętaj, że powiązania są dwukierunkowe, a więc zmiana wartości przez użytkownika w polu formularza spowoduje uaktualnienie wartości w modelu oraz ewentualną aktualizację innych powiązań. Wykorzystaj powiązania `foreach`, `value`, `text`, `options` lub inne, zależnie od potrzeb.
g. Jawnie zasubskrybuj zdarzenia modyfikacji modelu i prześlij modyfikacje wykonywane przez użytkownika do usługi REST (dodanie nowej krotki – POST, aktualizacja – PUT, usunięcie – DELETE).

Grupa zadań L (1p):

2. Wyszukiwanie danych
   a. Rozszerz funkcjonalność usługi REST o możliwość wyszukiwania po wartości każdego pola w modelu. Wyszukiwanie powinno działać poprzez wyszukiwanie podciągów znaków wprowadzonych przez użytkownika z ignorowaniem wielkości liter.
      Przykład: Dla wyszukiwania studenta o imieniu „an” powinni zostać znalezieni studenci o imionach:
      i. J**an**
      ii. J**an**ek
      iii. **An**drzej
   b. Stwórz nowe właściwości w modelu widoku reprezentujące wartości wpisane przez użytkownika w pola wyszukiwarki. Wykorzystaj te właściwości w zapytaniach wykonywanych do usługi REST.
   c. W dokumencie HTML dodaj powiązania pól formularza z w/w zmiennymi w modelu. Wykorzystaj powiązanie `textInput`, uaktualniające model po wpisaniu każdego znaku. Nie używaj dodatkowego przycisku uruchamiającego wyszukiwanie.
   d. Jawnie zasubskrybuj zdarzenia modyfikacji modelu i wykonaj zapytania do usługi REST w celu wyszukania danych. Możesz wykorzystać funkcje `ko.mapping.toJS()` oraz `$.param()` do konwersji zmiennych obserwowalnych modelu do JSONa oraz do serializacji JSONa do zapytania w URLu.