Programowanie wizualne

opracował: Wojciech Frohmberg

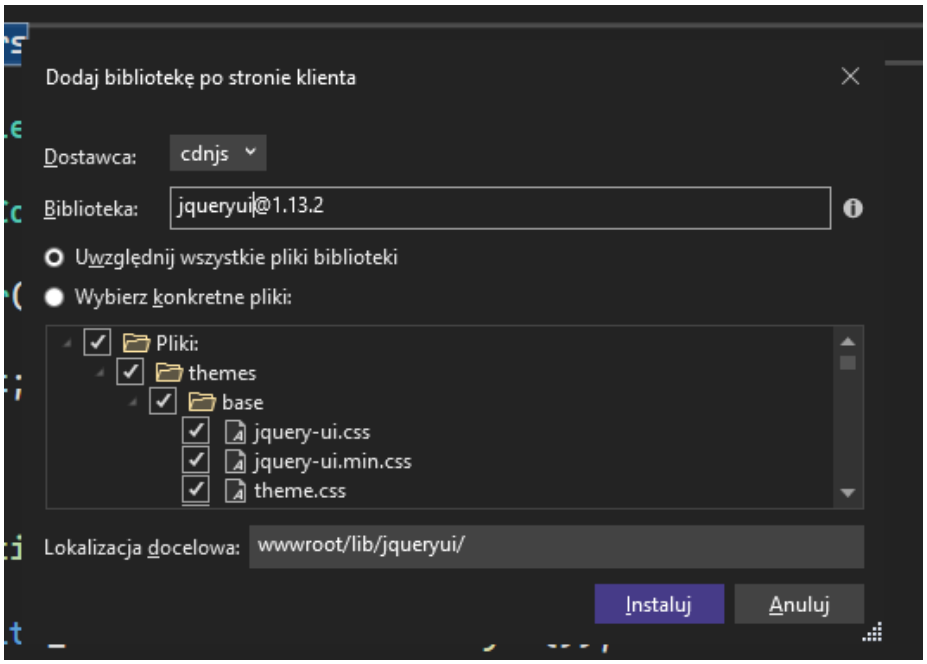
Lab 7

Zagadnienia do opanowania:

- Biblioteka Selenium
- XPath 1.0
- Wykorzystanie managera bibliotek LibMan

Zarys problemu:
Kontynuując projekt z naszego cyklu zajęciowego chcielibyśmy wpiąć funkcjonalność dzielenia zadań pomiędzy użytkowników w interfejs graficzny użytkownika aplikacji webowej. W celu weryfikacji poprawności wspomnianego wykorzystania utworzymy testy e2e zaproponowanej funkcjonalności. Jeśli nie udało Ci się w ramach poprzednich zajęć dokończyć implementacji wszystkich zadań możesz skorzystać z projektu ujętego w ramach zasobów zajęć.

1. W ramach projektu TaskShare wygeneruj kod brakujących kontrolerów wraz z widokami. Na potrzeby zajęć wystarczą kontrolery dla modeli Issue oraz User (patrz skrypt Lab 5. zadanie 16).
2. Na potrzeby dodawania elementów w połączeniu x do wiele w ramach klasy Issue chcielibyśmy dodać do projektu bibliotekę jqueryui zawierającą pole formularza z podpowiedziami możliwych wartości (tzw. Autocomplete). W celu dodania biblioteki naciśnij prawym przyciskiem na projekt i z podmenu Dodaj wybierz opcję „Biblioteka po stronie klienta…”.
3. W pole biblioteka wpisz „jqueryui” i naciśnij enter celem wybrania przez paczkę zarządzającą bibliotekami (LibMana) wersji biblioteki kompatybilnej z zainstalowanymi w ramach naszego projektu bibliotekami. Następnie wybierz opcję Instaluj.



4. W ramach eksploratora projektów w katalogu wwwroot dostępna jest teraz wskazana biblioteka, jednak póki co nie mamy jej wykorzystanej w template’cie naszego projektu. Celem jej fizycznego użycia musimy do pliku layoutu naszej strony dodać odwołanie do arkusza stylów biblioteki oraz jej skryptu. Do tego celu otwieramy plik Views/Shared/_Layout.cshtml i w sekcji head dodajemy odniesienie do pliku:

~/lib/jqueryui/themes/base/jquery-ui.min.css natomiast pod tagiem footer w miejscu gdzie znajdują się wszystkie skryptu dodajemy odniesienie do pliku skryptu biblioteki: ~/lib/jqueryui/jquery-ui.min.js

5. Do pliku skryptu strony tj. wwwroot/js/site.js przekopiuj kod przykładu użycia tej funkcjonalności ze strony: https://jqueryui.com/autocomplete/ z przykładu „Multiple values” (do pliku przekopiuj tylko i wyłącznie kod zawarty w ramach tagu skrypt on będzie jeszcze modyfikowany w ramach kolejnych zadań).
6. Z kodu usuń deklarację tablicy availableTags zastąpimy ją deklaracją wartości w ramach konkretnego widoku.
7. Kod:
   $("#tags")

   zamień na:
   $(".autocompletebyid").each(function (i, el) {
   var availableTags = tags[el.id];
   $(el)

   Przy czym dodaj element zamykający wywołanie metody each:
   });

   Wskazana zmiana służy uzależnieniu wartości podpiętych pod nasz tag od id inputu, które ma być autouzupełniane.
8. W ramach widoku Views/Issues/Create.cshtml pod grupą właściwości Description dodaj grupę zawierającą podobne pole jak powyżej, jednak usuń atrybut asp-for zarówno dla labela jak i dla inputa. Usuń span z asp-validation-for="Description"
9. W tekście elementu label wpisz „Users” podobnie w ramach id skopiowanego inputa oraz jego atrybutu name.
10. Do wspomnianego inputa Users dodajmy jeszcze klasę autocompletebyid.
11. W ramach sekcji skrypt na końcu pliku widoku dodaj kod:
    ```
    <script>
       var tags = {};
       tags["Users"] = @Json.Serialize(ViewBag.ListOfPseudonyms);
    </script>
    ```
12. Żeby podany kod zadziałał zgodnie z oczekiwaniami, do strony będziemy musieli przekazać w naszej zmiennej ViewBag listę pseudonimów dodanych użytkowników. Do tego celu w akcji Create typu GET naszego kontrolera IssuesController dodajmy linijkę:

    ViewBag.ListOfPseudonyms = _context.Users.Select(u => u.Pseudonym).ToList();

13. Podobną czynność powtórzmy dla skojarzonych modeli Tasks tj. Utwórzmy odpowiednią grupę elementów tuż pod grupą formularza z inputem przeznaczonym do uzupełniania użytkowników. Wypełnijmy tekst labelu na Tasks oraz id inputu oraz jego atrybut name na Tasks, upewnijmy się że nasz input jest obdarzony klasą autocompletebyid.
14. Do kodu naszego skryptu dodajmy:

    tags["Tasks"] = @Json.Serialize(ViewBag.ListOfLabels);

15. W kontrolerze Issues dodajmy linijkę odpowiadającą za wypełnienie listy wszystkich etykiet tasków:

    ViewBag.ListOfLabels = _context.Tasks.Select(u => u.Label).ToList();

16. Do akcji Create typu POST naszego kontrolera IssuesController dodajmy parametry: [Bind("Users")]string Users, [Bind("Tasks")] string Tasks

    oraz tuż pod dodaniem naszego issue do bazy:

```
issue = _context.Issues.Include("Users")
                .Include("Tasks").FirstOrDefault(m => m.Id == issue.Id);
var listOfUsers = Users
   .Split(",")
   .Select(pseudo => pseudo.Trim())
   .Join(_context.Users,
      pseudo => pseudo,
      user => user.Pseudonym,
      (pseudo, user) => user)
   .ToList();

var listOfTasks = Tasks
   .Split(",")
   .Select(desc => desc.Trim())
   .Join(_context.Tasks,
      desc => desc,
      bill => bill.Label,
      (desc, bill) => bill)
   .ToList();
issue.Users = listOfUsers;
issue.Tasks = listOfTasks;

_context.Issues.Update(issue);
_context.SaveChanges();
```

    Kod zapewni zinterpretowanie ciągów inputów na elementy skojarzonych z issue modeli.
17. Do widoku View/Issues/Index.cshtml dodajmy nowy przycisk do każdego wpisu przy użyciu którego wywołamy działanie naszego algorytmu podziału zadań z danego Issue. W ramach tego widoku przed linkiem Edit dodajmy kod:
    <a asp-action="Suggest" asp-route-id="@item.Id">Suggest split |
18. W ramach kontrolera IssuesController dodaj metodę akcji Suggest o sygnaturze metody Edit oraz ciele:

```
if (id == null || _context.Issues == null)
{
   return NotFound();
}

var issue = await _context.Issues
    .Include("Users")
    .Include("Tasks")
    .FirstOrDefaultAsync(i => i.Id == id);

if (issue == null)
{
   return NotFound();
}
```

```
var algorithm = new BipartitionAlgorithm<Task>();
algorithm.SubsetsCount = issue.Users.Count();
algorithm.Predicate = i => i.TimeCost;
ViewBag.Result = algorithm.Run(issue.Tasks);

return View(issue);
```

19. W ramach metody zaproponuj sposób przechwytywania sytuacji wyjątkowych wynikających z działania algorytmu oraz przekazywania informacji o zaistniałej sytuacji do widoku.
20. Kliknij na nowo dodaną metodą prawym przyciskiem myszy i wybierz opcję „dodaj widok” a następnie wybierz opcję „Widok razor”.
21. W kolejnym oknie wybierz szablon „Details” klasa modelu „Issue” i naciśnij przycisk „Dodaj”.
22. Usuń link:

```
<a asp-action="Edit" asp-route-id="@Model?.Id">Edit</a> |
```

    oraz wypełnienie elementów dt oraz dd. Pozostaw tylko pojedynczą parę dt oraz dd i wypełnij ją w pętli wartościami z Model.Users oraz ViewBag.Result lub zapronowanymi informacjami dotyczącymi zaistniałej wyjątkowej sytuacji.
23. Przetestuj ręcznie dodanie zestawu zadań użytkowników i podpięcie ich do issue.
24. Do rozwiązania dodaj nowy projekt typu „Projekt testowy NUnit”. Projekt nazwij AlgorithmIntegrationTests.
25. Do projektu doinstaluj paczki – Selenium.WebDriver oraz driver konkretnej przeglądarki na której chcesz przeprowadzać testy np. Selenium.WebDriver.ChromeDriver
26. W ramach projektu nie musisz dodawać referencji do projektu TaskShare – ponieważ będziesz dostawać się do niego poprzez jego interfejs użytkownika.
27. Do projektu dodaj klasę testów interfejsu, dodaj i zainicjuj w konstruktorze klasy pole _driver typu IWebDriver. Alternatywnie inicjalizacji możesz dokonać w dedykowanej metodzie SetUp, podejście to jednak będzie wymagało otworzenia tylu przeglądarek ile w ramach projektu jest instancji testów. Dodaj do klasy metodę czyszczącą wszystkie testy i oznacz ją atrybutem OneTimeTearDown. W ramach metody zamknij instancję przeglądarki.
28. W ramach klasy utwórz też metodę testującą dodawanie użytkownika. Skorzystaj do tego celu z atrybutu TestCaseSource podające metodę generującą użytkowników testowych. Każdy test zwieńcz asercją testującą czy użytkownik rzeczywiście został dodany. Użytkownika w ramach testu dodaj przy użyciu formularza twojej usługi znajdującego się pod adresem:
    https://localhost:[port]/Users/Create
29. Do podanej metody dodaj atrybut Order określający kolejność przeprowadzania testów. Ustaw odpowiednią wartość liczbową w ramach parametru tak, by dodawanie użytkowników przeprowadzane było na początku testów.
30. Do klasy dodaj metodę testującą usuwanie użytkownika i skorzystaj do utworzenia instancji testów z podobnej metody generującej pseudonimy użytkownika jak w przypadku metody testującej dodawanie użytkowników. Do celu usunięcia odpowiedniego wpisu z użytkownikiem z bazy znajdź link usuwający użytkownika znajdujący się w ramach listy ze wszystkimi użytkownikami ze strony https://localhost:[port]/Users/Index/. Pamiętaj by do metody dodać atrybut Order i ustawić priorytet dodawania tak by metoda testująca uruchamiała się na końcu wszystkich metod testujących naszej klasy.
31. Do klasy testowej dodaj metodę testującą dodawanie i usuwanie zadania. Podobnie jak w przypadku użytkownika do celu wypełnienia danych skorzystaj z odpowiedniego formularza usługi i zadbaj o dodanie odpowiedniej asercji testującej czy atrybutów Order. Pamiętaj, że

zadania mogą być podłączone na raz tylko do jednego obiektu Issue stąd też przemyśl w jaki sposób rozplanować liczbę i wartości poszczególnych dodawanych obiektów zadania tak by przeprowadzić sensowne testy uruchamiania algorytmu.

32. Do klasy testowej dodaj metodę testującą dodawanie i usuwanie Issue. W trakcie dodawania zadbaj o podpięcie pod Issue odpowiednich użytkowników oraz zadań.
33. Utwórz metodę testującą wywołanie algorytmu. Zadbaj by przetestować zarówno przypadki pozytywnego zakończenia algorytmu, jak i przypadki, w których algorytmowi nie udało się znaleźć równomiernego podziału zadań pomiędzy użytkowników.
34. Uruchamianie projektu testów przy wykorzystaniu biblioteki Selenium będzie wymagało działania naszej aplikacji w tle. Stąd też żeby przeprowadzać testy zawsze będą potrzebne dwie instancję VisualStudio 2022 – na jednej uruchomisz aplikację webową, a na drugiej – testy.