Programowanie wizualne

opracował: Wojciech Frohmberg

Lab 8

Zagadnienia do opanowania:

- Biblioteka Selenium
- XPath 1.0
- Wykorzystanie managera bibliotek LibMan

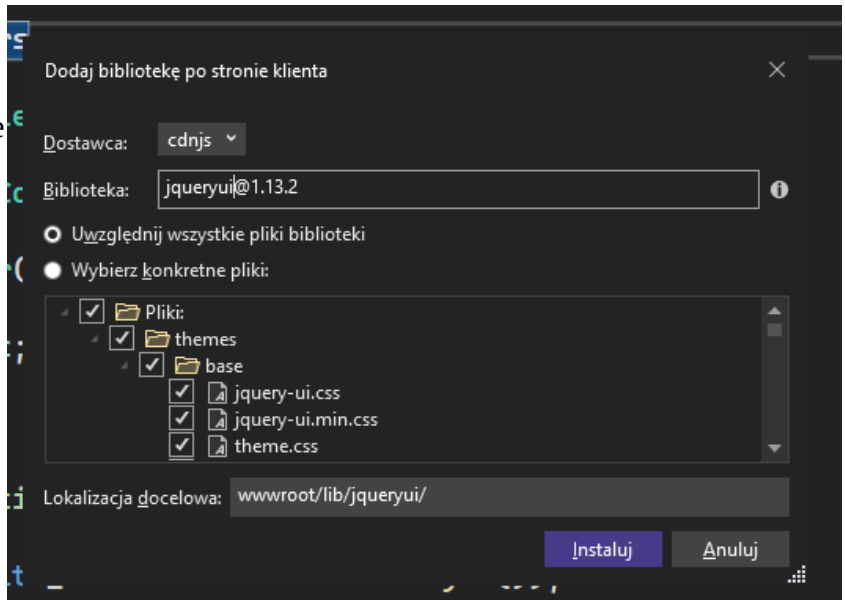
Zarys problemu:
Kontynuując projekt z naszego cyklu zajęciowego chcielibyśmy wpiąć funkcjonalność dzielenia zadań pomiędzy użytkowników w interfejs graficzny użytkownika aplikacji webowej. W celu weryfikacji poprawności wspomnianego wykorzystania utworzymy testy e2e zaproponowanej funkcjonalności. Jeśli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

Ostrzeżenie:
Jeśli korzystasz z dotnet cli przy użyciu wsl możesz mieć problem z uruchomieniem przeglądarki z poziomu komendy uruchamiającej testy. Do wykonywania poleceń zaleca się korzystać z dotnet cli w macierzystym systemie.

1. Póki co w ramach zadania z drugiego laboratorium testowaliśmy możliwość dodania zadania. Niestety generowany kod, który zadziałał w tamtym przypadku, przy próbie dodania np. dewelopera zakończy się niepowiedzeniem. Problem polega na asocjacjach, którymi powiązany jest deweloper z innymi modelami. Kontroler dewelopera przy próbie dodania nowego obiektu dewelopera do bazy waliduje dane wstawione do nowo utworzonego obiektu modelu poprzez właściwość ModelState (por. plik Controllers/DevelopersController.cs Create w wersji POST). By walidacja zaczęła działać musimy wymusić wyłączenie jej dla właściwości Issues oraz Assignments. Można to zrobić poprzez dodanie przed walidacją kodu (testem `if (ModelState.IsValid)`):

   ```
   ModelState.Remove("Issues");
   ModelState.Remove("Assignments");
   ```

2. Żeby dać możliwość połączenia danego Issue z wieloma zadaniami oraz wieloma deweloperami potrzebujemy biblioteki, która umożliwi przypisanie wielu obiektów składowanych w naszej bazie do elementu Issue. Potrzebujemy zatem biblioteki po stronie frontendu, która umożliwi wybieranie wielu elementów najlepiej z autopodpowiadaniem. Jedną z możliwych bibliotek które możemy w tym celu zastosować jest jqueryui zawierającą pole formularza



z podpowiedziami możliwych wartości (tzw. Autocomplete). W celu dodania biblioteki (jeśli korzystasz z VS2022) naciśnij prawym przyciskiem na projekt AssignmentSharing i z podmenu Dodaj wybierz opcję „Biblioteka po stronie klienta…”.
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.
Jeśli używasz dotnet cli w ramach projektu dodaj plik libman.json o następującej treści:

```
{
  "version": "1.0",
  "defaultProvider": "cdnjs",
  "libraries": [
    { "library": "jqueryui@1.13.2",  "destination": "wwwroot/lib/jqueryui/" }
  ]
}
```

Następnie uruchom komendę instalującą narzędzie libman:

```
dotnet tool install Microsoft.Web.LibraryManager.Cli
```

Pamiętaj że do tego celu musisz mieć utworzony tool-manifest (patrz poprzednie zajęcia). Następnie skorzystaj z libmana żeby dodać narzędzia po stronie klienta przy użyciu komendy:

```
dotnet libman restore
```

3. W ramach eksploratora projektów w katalogu wwwroot w podkatalogu lib dostępna jest teraz wskazana biblioteka (folder jqueryui). Póki co jednak 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 (nie trzeba korzystać z atrybutu `asp-append-version="true"`) natomiast pod tagiem footer w miejscu gdzie znajdują się wszystkie skryptu dodajemy odniesienie do pliku skryptu biblioteki: ~/lib/jqueryui/jquery-ui.js (tutaj również nie ma potrzeby wykorzystywania atrybutu `asp-append-version="true"`)
4. Do pliku skryptu strony tj. wwwroot/js/site.js przekopiuj kod przykładu użycia tej funkcjonalności ze strony: https://jqueryui.com/autocomplete/#multiple z przykładu „Multiple values” (do pliku przekopiuj tylko i wyłącznie kod zawarty w ramach tagu skrypt on będzie jeszcze modyfikowany w ramach kolejnych zadań).
5. Z kodu usuń deklarację tablicy availableTags zastąpimy ją deklaracją wartości w ramach konkretnego widoku.
6. 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.

7. 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"`
8. W tekście elementu label wpisz „Developers” podobnie w ramach id skopiowanego inputa oraz jego atrybutu name:
   ```
   <input id="Developers" name="Developers" class="form-control"/>
   ```
9. Do wspomnianego inputa Developers dodajmy jeszcze klasę (po spacji) `autocompletebyid`.
10. W ramach sekcji skrypt na końcu pliku widoku dodaj kod:
    ```
    <script>
       var tags = {};
       tags["Developers"] = @Json.Serialize(ViewBag.ListOfPseudonyms);
    </script>
    ```
11. Żeby podany kod zadziałał zgodnie z oczekiwaniami, do strony będziemy musieli przekazać w naszej zmiennej ViewBag listę pseudonimów dodanych deweloperów. Do tego celu w akcji Create typu GET naszego kontrolera IssuesController dodajmy linijkę:

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

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

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

14. W kontrolerze Issues dodajmy linijkę odpowiadającą za wypełnienie listy wszystkich etykiet zadań:

    ```
    ViewBag.ListOfLabels = _context.Assignments.Select(a => a.Label).ToList();
    ```

15. Do akcji Create typu POST naszego kontrolera IssuesController dodajmy parametry:
    ```
    [Bind("Developers")] string Developers, [Bind("Assignments")] string Assignments
    ```

    oraz tuż pod dodaniem naszego issue do bazy:

    ```
            issue = _context.Issues.Include("Developers")
                .Include("Assignments").FirstOrDefault(m => m.Id == issue.Id);
             var listOfDevelopers = (Developers ?? "")
                .Split(",")
                .Select(pseudo => pseudo.Trim())
                .Join(_context.Developers,
                    pseudo => pseudo,
                    d => d.Pseudonym,
                    (pseudo, d) => d)
                .ToList();

             var listOfAssignments = (Assignments ?? "")
                .Split(",")
                .Select(desc => desc.Trim())
                .Join(_context.Assignments,
                    desc => desc,
                    a => a.Label,
    ```

```
                (desc, a) => a)
            .ToList();
        issue.Developers = listOfDevelopers;
        issue.Assignments = listOfAssignments;

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

Kod zapewni zinterpretowanie ciągów inputów na elementy skojarzonych z issue modeli.

16. Do tego samego pliku przed testem stanu modelu dodaj linijki usuwające walidację skojarzonych asocjacji deweloperów i zadań:

```
ModelState.Remove("Developers");
ModelState.Remove("Assignments");
```

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</a> |
```

18. W ramach kontrolera IssuesController dodaj metodę akcji Suggest o sygnaturze metody Edit (w wersji GET) oraz ciele:

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

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

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

try
{
    var algorithm = new BipartitionAlgorithm<Assignment>();
    algorithm.SubsetsCount = issue.Developers.Count();
    algorithm.InterpretAsWeight = i => i.TimeCost;
    ViewBag.Result = algorithm.Run(issue.Assignments);
}
catch (InappropriateWeightException e)
{
    ViewBag.Error = "Invalid time cost of one of the assignments";
}
catch (InputSetUnsplittableException e)
{
    ViewBag.Error = "Cannot split the set of assignments";
}
catch (UnsupportedSubsetsCountException e)
{
```

```
                ViewBag.Error = "Cannot split the set of assignments into " +
                                issue.Developers.Count() + " subsets";
            }


            return View(issue);
```

19. Kliknij na nowo dodaną metodą prawym przyciskiem myszy i wybierz opcję „dodaj widok” a następnie wybierz opcję „Widok razor”.
20. W kolejnym oknie wybierz szablon „Details” klasa modelu „Issue” i naciśnij przycisk „Dodaj”.
21. Wypełnij stronę następującym kodem:

```
@model AssignmentSharing.Models.Issue

@{
    ViewData["Title"] = "Suggest";
}

<h1>Suggest</h1>
@if (ViewBag.Error != null)
{
<div class="alert alert-danger" role="alert">
    @ViewBag.Error
</div>
}
else
{
<div>
    <h4>Suggested assignment of the issue split is as follows:</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">
            First developer:
        </dt>
        <dd class="col-sm-10">
            @foreach(dynamic e in ViewBag.Result[0])
            {
                @e.Label
            }
        </dd>
        <dt class="col-sm-2">
            Second developer:
        </dt>
        <dd class="col-sm-10">
            @foreach (dynamic e in ViewBag.Result[1])
            {
                @e.Label
            }
        </dd>
    </dl>
</div>
}
<div>
    <a asp-action="Edit" asp-route-id="@Model?.Id">Edit</a> |
    <a asp-action="Index">Back to List</a>
</div>
```

22. Przetestuj ręcznie dodanie zestawu zadań oraz deweloperów i podpięcie ich do issue. Następnie przetestuj sugestie podziału zadań pomiędzy deweloperów.
23. Do rozwiązania dodaj nowy projekt typu „Projekt testowy NUnit”. Projekt nazwij AlgorithmIntegrationTests.
24. Do projektu doinstaluj paczki – Selenium.WebDriver oraz driver konkretnej przeglądarki na której chcesz przeprowadzać testy np. Selenium.WebDriver.ChromeDriver. Możesz do tego celu np. skorzystać z konsoli menedżera pakietów i instrukcji:

    ```
    Install-Package Selenium.WebDriver
    Install-Package Selenium.WebDriver.ChromeDriver
    ```

    lub dla cli

    ```
    dotnet add package Selenium.WebDriver
    dotnet add package Selenium.WebDriver.ChromeDriver
    ```

25. W ramach projektu nie musisz dodawać referencji do projektu AssignmentSharing – ponieważ będziesz dostawać się do niego poprzez jego interfejs użytkownika.
26. Do projektu dodaj klasę testów interfejsu (InterfaceTests), dodaj i zainicjuj w konstruktorze klasy testów pole _driver typu IWebDriver:

    ```
    private IWebDriver _driver;

    public InterfaceTests()
    {
        _driver = new ChromeDriver();
    }
    ```

    Pamiętaj by dodać using do użycia odpowiednie przestrzenie nazw (`OpenQA.Selenium` oraz `OpenQA.Selenium.Chrome`)

27. Dodaj do klasy metodę czyszczącą wszystkie testy i oznacz ją atrybutem OneTimeTearDown. W ramach metody zamknij instancję przeglądarki:

    ```
    [OneTimeTearDown]
    public void OneTimeTearDown()
    {
        _driver.Quit();
    }
    ```

28. W ramach klasy utwórz też metodę testującą dodawanie dewelopera. 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
    Przykładowy kod testów może wyglądać następująco:

    ```
    private static IEnumerable<string> UserNameGenerator()
    {
        for (int i = 0; i < 10; i++)
        {
    ```

```
            yield return $"generated-developer{i}";
        }
    }

    [TestCaseSource(nameof(UserNameGenerator))]
    public void AddDevelopersTests(string pseudonym)
    {
        _driver.Navigate().GoToUrl("https://localhost:7068/Developers/Create");
        _driver.FindElement(
             By.XPath(
                "//*[text()='FirstName']/following::input[1]"
             )
        ).SendKeys("John");
        _driver.FindElement(
             By.XPath(
                 "//*[text()='LastName']/following::input[1]"
             )
        ).SendKeys("Doe");
        _driver.FindElement(
             By.XPath(
                 "//*[text()='Pseudonym']/following::input[1]"
             )
        ).SendKeys(pseudonym);
        _driver.FindElement(By.XPath("//*[@value='Create']")).Click();
    }
```

Pamiętaj by zmienić numer portu ujęty w ramach testu na ten odpowiadający Twojej aplikacji.

29. Do podanej metody testowej dodaj atrybut Order określający kolejność przeprowadzania testów. Ustaw odpowiednią wartość liczbową w ramach parametru tak, by dodawanie deweloperów przeprowadzane było na początku testów:

```
[TestCaseSource(nameof(UserNameGenerator)), Order(1)]
public void AddDevelopersTests(string pseudonym)
```

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. Kod może wyglądać następująco:

```
[TestCaseSource(nameof(UserNameGenerator)), Order(100)]
public void DeleteDevelopersTests(string pseudonym)
{
    _driver.Navigate().GoToUrl("https://localhost:7068/Developers");
    _driver.FindElement(
        By.XPath(
            $"//td[contains(.,'{pseudonym}')]/following::td[1]/a[3]"
        )
    ).Click();
    _driver.FindElement(By.XPath("//input[@value='Delete']")).Click();
}
```

31. Jeśli korzystasz z VS2022 celem uruchomienia testów niezbędne będzie w jednej z instancji Visual Studio uruchomienie aplikacji. W drugiej instancji, w której otwarte jest to samo rozwiązanie uruchomienie testów np. poprzez wybranie menu Test->Uruchom wszystkie testy. W przypadku korzystania z dotnet cli nie ma takiego problemu.
32. 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.
33. Do klasy testowej dodaj metodę testującą dodawanie i usuwanie Issue. W trakcie dodawania zadbaj o podpięcie pod Issue odpowiednich użytkowników oraz zadań.
34. 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.