Programowanie Aplikacji Internetowych

Laboratorium nr 3

Wzorzec MVC w JavaScript - AngularJS

Poniżej znajdują się zadania, które należy wykonać w ramach laboratoriów, a następnie sporządzić sprawozdanie w formie archiwum .zip. Plik archiwum powinien mieć nazwę zgodną ze wzorem: PAI_Lab<nr_laboratorium>_<pierwsza_litera_imienia>.<nazwisko_bez_polskich_znaków>.zip, np. PAI_Lab2_J.Kowalski.zip. W archiwum powinny znajdować się wszystkie pliki z poniższych zadań.

Zadanie 1 Wprowadzenie w AngularJS

Zaczynamy od podstawowej struktury pliku *kalendarz.html*:

```
<!DOCTYPE html>

<html lang="pl">
<head>
    <title>Kalendarz</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>

<body>

</body>
</html>
```

Kolejnym krokiem będzie zadeklarowanie zakresu aplikacji AngularJS oraz dołączenie odpowiedniej biblioteki:

```
<html lang="pl" ng-app='calendar'>
<head>
    <title>Kalendarz</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular «
.min.js"></script>
</head>
```

W powyższy sposób zadeklarowaliśmy, że będziemy tworzyć naszą aplikację w obrębie modułu o nazwie *calendar.* Jego inicjalizację rozpoczniemy w pliku *app.js:*

```
(function() {
      var app = angular.module('calendar', []);

})();
```

Plik ten oczywiście należy dołączyć do pliku *kalendarz.html*:

```
<head>
      <meta http-equiv="Content-Type" content="text/html;charset=utf-8" >
      <script  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/ «
angular.min.js"></script>
      <script src="app.js"></script>
</head>
```

Dodamy również arkusz stylów *style.css*, który na razie pozostanie pusty (tworzymy pusty plik). Dołączamy go do pliku *kalendarz.html*:

```
<head>
    <title>Kalendarz</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link rel='stylesheet' type='text/css' href='style.css'>
    <script  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular «
.min.js"></script>
    <script src="app.js"></script>
</head>
```

Nasza aplikacja będzie przechowywała serie notatek. W tym celu zaczniemy od utworzenia modelu. Ułatwi nam to przygotowanie widoku.

W pliku *app.js* tworzymy zmienną, która przechowuje tablice tablic obiektów:

```
(function() {
	var app = angular.module('calendar', []);

	var model = [
		[	// pojedyncza seria
			{	// pojedynczy obrazek dnia
				"imgUrl": 'img/smile1.png',
				"date": '12.04.2014',
				"description": '',
				"stars": 3,
				"evaluated": true
			},
			{
				"imgUrl": 'img/smile2.png',
				"date": '12.04.2014',
				"description": '',
				"stars": 3,
				"evaluated": true
			},
			{
				"imgUrl": 'img/smile3.png',
				"date": '12.04.2014',
				"description": '',
				"stars": 3,
				"evaluated": true
			}
		],
		[
			{
				"imgUrl": 'img/smile4.png',
				"date": '12.04.2014',
				"description": '',
				"stars": 3,
				"evaluated": true
			},
			{
				"imgUrl": 'img/smile2.png',
				"date": '12.04.2014',
				"description": '',
				"stars": 3,
				"evaluated": true
			}
		]
	];
})();
```

Zaprezentujmy powyższe dane na widoku. W tym celu w pliku *kalendarz.html* dopisujemy:

```
<html lang="pl">
<head>
    ...
</head>
<body>
	<div class="table">
	  <div class="row">
		<div class="cell">
		    <h4></h4>
		    <img >
		    <h5></h5>
		    <span></span>
		    <p></p>
		    <code></code>
		</div>
```

```
        </div>
    </div>
</body>
</html>
```

Na chwilę obecną nic się nie wyświetli na ekranie przeglądarki, ale gdy dopiszemy poniższy kod do pliku *style.css* zobaczymy pustą komórkę:

```
.table {
    display: table;
}

.row {
    display: table-row;
}

.cell {
    display: table-cell;
    padding: 5px;
    border: 1px solid black;
    text-align: center;
    vertical-align: middle;
}
```

Do wypełnienia danymi przygotowanej struktury tabeli posłużymy się kontrolerem AngularJS. W tym celu w pliku *kalendarz.html* dopisujemy:

```
<body ng-controller='CalendarController as calCtrl'>
      <div class="table">
        <div class="row" ng-repeat='days in calCtrl.series track by $index'>
            <div class="cell" ng-repeat='day in days'>
                <h4>{{ $index + 1 }}</h4>
                <img ng-src="{{ day.imgUrl }}">
                <h5>{{ day.date }}</h5>
                <span>{{ day.description }}</span>
                <p>{{day.stars}}</p>
                <code>{{day.evaluated}}</code>
            </div>
        </div>
    </div>
</body>
```

W powyższym kodzie pierwszą dyrektywą jaką dodajemy jest dyrektywa *ng-controller*, która deklaruje zasięg kontrolera o nazwie *CalendarController*. Daje nam to dostęp do zmiennych tego kontrolera (jeszcze ich nie zdefiniowaliśmy). Zakładamy, że zmienna *calCtrl.series* jest kolekcją obiektów, więc za pomocą dyrektywy *ng-repeat* będziemy po niej iterować. Dalej zakładamy, że obiektami wewnątrz tej kolekcji będą także kolekcje, więc ponownie używamy dyrektywy *ng-repeat* aby przeiterować po obiektach, które znajdują się wewnątrz tych kolekcji. W końcu docieramy do najgłębszego zagnieżdżenia, gdzie wyświetlamy poszczególne informacje. Zmienna $index przechowuje aktualny indeks obiektu (począwszy od 0).

Powyższy kod nie będzie oczywiście działać póki nie zdefiniujemy kontrolera *CalendarController* w pliku *app.js*:

```
(function() {
      var app = angular.module('calendar', []);

      app.controller('CalendarController', function() {
            this.series = model;
      });

      ...
})();
```

Przetestujmy aplikację.

Usprawnijmy pewną rzecz, a mianowicie zadbajmy o to aby ilość gwiazdek była wyświetlana tylko w wypadku gdy wpis został oceniony, a gwiazdki niech rzeczywiście będą gwiazdkami. Zmiany należy dokonać w pliku *kalendarz.html*:

```
<div class="cell" ng-repeat='day in days'>
    <h4>{{ $index + 1 }}</h4>
    <img ng-src="{{ day.imgUrl }}">
    <h5>{{ day.date }}</h5>
    <span>{{ day.description }}</span>
     <p ng-show='day.evaluated'><img ng-repeat="x in calCtrl.range(day.stars)" ng-
src= «
'img/star.png'></p>
</div>
```

oraz w pliku *app.js* dodajemy funkcję pomocniczą:

```
app.controller('CalendarController', function() {
            this.series = model;

            this.range = function(range) {
                  var arr = [];
                  for (var i = 0; i < range; i++) {
                        arr += i+1;
                  }
                  return arr;
            };
      });
```

Ponownie przetestujmy aplikację.

Czas na dodanie możliwości manipulowania danymi. Dodamy funkcjonalność dodawania kolejnego wpisu do danej serii oraz rozpoczynania nowej serii. W tym celu rozbudujmy plik *kalendarz.html*:

```
<div class="row" ng-repeat='days in calCtrl.series'>
  <div class="cell" ng-repeat='day in days'>
    <h4>{{ $index + 1 }}</h4>
    <img ng-src="{{ day.imgUrl }}">
    <h5>{{ day.date }}</h5>
    <span>{{ day.description }}</span>
     <p ng-show='day.evaluated'><img ng-repeat="x in calCtrl.range(day.stars)" ng-
src= «
'img/star.png'></p>
  </div>
  <div class="cell">
    <ul>
        <li>
            <a    href='#openModal'    ng-click='calCtrl.setSeriesNo($index)'>Dodaj
wpis</a>
        </li>
        <li>
            <a ng-click='calCtrl.addSeries(days)'>Dodaj serię</a>
        </li>
    </ul>
  </div>
</div>
```

Powyższy kod dodaje następną komórkę do naszych wierszy. Znajdują się w niej dwa linki. Pierwszy będzie otwierał okno, w którym znajdować się będzie formularz dodawania nowego wpisu. Drugi link będzie dodawał nowy pusty wiersz do tabeli.

Zacznijmy od dodawania nowego wiersza. W tym wypadku wystarczy dodać odpowiednią funkcję do naszego kontrolera w pliku *app.js*:

```
app.controller('CalendarController', function() {
      ...
      this.seriesNo = -1;
```

```
	this.setSeriesNo = function(numb) {
		this.seriesNo = numb;
	};
	this.addSeries = function(object) {
		var idx = this.series.indexOf(object) + 1;
		this.series.splice(idx, 0, []);
	};
});
```

Dodajmy również kilka linijek kodu do pliku *style.css* aby nasza aplikacja lepiej wyglądała:

```
...
.cell li {
    list-style-type: none;
    padding-left: 0px;
    margin-top: 15px;
}

a[ng-click], a[href]:link, a[href]:visited {
    color: black;
    text-decoration: none;
    cursor: pointer;
    font-weight: bold;
}
```

Zajmijmy się teraz oknem formularza. Mechanizm pojawiającego się okna wykorzystuje wyłącznie HTML5 i CSS3. W pliku *kalendarz.html* dopisujemy:

```
<html lang="pl">
<head>
    ...
</head>
<body ng-controller='CalendarController as calCtrl'>
	<div class="table">
	  ...
	</div>
	<div id="openModal" class="modalDialog">
		<div>
			<a href="#close" title="Close" class="close">X</a>
			<h2>Dodaj nowy wpis</h2>
		</div>
	</div>
</body>
</html>
```

W pliku *style.css* dopisujemy:

```
...

.modalDialog {
	position: fixed;
	font-family: Arial, Helvetica, sans-serif;
	top: 0;
	right: 0;
	bottom: 0;
	left: 0;
	background: rgba(0,0,0,0.8);
	z-index: 99999;
	opacity:0;
	-webkit-transition: opacity 400ms ease-in;
	-moz-transition: opacity 400ms ease-in;
	transition: opacity 400ms ease-in;
	pointer-events: none;
}

.modalDialog:target {
	opacity:1;
```

```
      pointer-events: auto;
}

.modalDialog > div {
      width: 400px;
      position: relative;
      margin: 10% auto;
      padding: 5px 20px 13px 20px;
      border-radius: 10px;
      background: #fff;
      background: -moz-linear-gradient(#fff, #999);
      background: -webkit-linear-gradient(#fff, #999);
      background: -o-linear-gradient(#fff, #999);
}

.close {
      background: #606061;
      color: #FFFFFF;
      line-height: 25px;
      position: absolute;
      right: -12px;
      text-align: center;
      top: -10px;
      width: 24px;
      text-decoration: none;
      font-weight: bold;
      -webkit-border-radius: 12px;
      -moz-border-radius: 12px;
      border-radius: 12px;
      -moz-box-shadow: 1px 1px 3px #000;
      -webkit-box-shadow: 1px 1px 3px #000;
      box-shadow: 1px 1px 3px #000;
}

.close:hover { background: #00d9ff; }
```

Przetestujmy aplikację.

Zajmijmy się kodem formularza w pliku *kalendarz.html*:

```
    <div id="openModal" class="modalDialog">
      <div>
            <a href="#close" title="Close" class="close">X</a>
            <h2>Dodaj nową obserwację</h2>
            <form action='#close'>
                <div class='form-field'>
                    <span class='sp'>Oceń:</span>
                    <input type='checkbox'>
                </div>
                <div class='form-field'>
                    <span class='sp'>Ocena:</span>
                    <input type='radio'>
                </div>
                <div class='form-field'>
                    <span class='sp'>Podaj opis (opcjonalnie):</span><br />
                    <textarea></textarea>
                </div>
                <div class='form-field'>
                    <input type='submit' value='Dodaj'>
                </div>
            </form>
        </div>
    </div>
```

Przetestujmy aplikację. Godne uwagi jest to, iż zarówno klikając link *X*, jak i przycisk *Dodaj* zamyka się okno. W tym momencie jeszcze nie ma różnicy, z którego sposobu skorzystamy efekt będzie ten

sam.

Rozbudujmy teraz nasz formularz za pomocą AngularJS i przede wszystkim dodajmy pole wyboru obrazka, które musimy utworzyć samodzielnie. W pliku *kalendarz.html* dopiszmy:

```
<form action='#close' ng-controller='FormController as formCtrl'
            ng-submit='formCtrl.add(calCtrl.getSeries())'>
      <div class='form-field'>
            <span class='sp'>Oceń:</span>
            <input type='checkbox' ng-model='record.evaluated'>
      </div>
      <div class='form-field' ng-show='record.evaluated'>
            <span class='sp'>Ocena:</span>
            <input type='radio' ng-repeat='x in calCtrl.range(5)'
                                    ng-model='record.stars' value='{{x}}'>
      </div>
      <div class='form-field'>
            <span class='sp'>Podaj opis (opcjonalnie):</span><br />
            <textarea ng-model='record.description'></textarea>
      </div>
      <div class='form-field'>
            <input type='submit' value='Dodaj obserwację'>
      </div>
</form>
```

Powyższe zmiany uwzględniają dodanie:

- kontrolera, który obsługuje formularz;
- metod nasłuchujących zmiany *ng-model*;
- mechanizmu warunkowego pojawiania się (*ng-show*); oraz
- powielenia pól radio (*ng-repeat*), które odpowiadają ilości gwiazdek, które można przyznać podczas oceniania.

Dopisując w pliku *app.js* poniższy kod sprawimy, że formularz zacznie poprawnie działać:

```
(function() {
      var app = angular.module('calendar', []);

      app.controller('CalendarController', function() {
            ...
            this.getSeries = function() {
                  return this.series[this.seriesNo];
            };
      });

      app.controller('FormController', ['$scope', function($scope) {
            $scope.record = {
                  stars: 0,
                  imgUrl: 'img/smile0.png'
            };

            var getDate = function() {
                  d = new Date();
                  day = d.getDay();
                  mth = d.getMonth();
                  year = d.getFullYear();
                  day = (day > 9) ? day : '0'+day;
                  mth = (mth > 9) ? mth : '0'+mth;
                  return day + '.' + mth + '.'+ year;
            }

            this.add = function(series) {
                  $scope.record.date = getDate();
                  series.push($scope.record);
                  $scope.record = {
```

```
                        stars: 0,
                        imgUrl: 'img/smile0.png'
                  };
            };
      }]);
      ...
})();
```

Wciąż brakuje pola wyboru obrazka. Zajmijmy się tym teraz edytując plik *kalendarz.html*:

```
<form action='#close' ng-controller='FormController as formCtrl'
            ng-submit='formCtrl.add(calCtrl.getSeries())'>
      <div class='form-field'>
            <span class='sp'>Wybierz obrazek:</span><br />
            <div class="selection-wrapper">
                  <div class="selected-item-box">
                        <span class="dropdown-icon"></span>
                        <ul class="items-list">
                              <li>
                                    <img />
                              </li>
                        </ul>
                  </div>
                  <div class='list">
                        <ul class="items-list">
                              <li>
                                    <img  />
                              </li>
                        </ul>
                  </div>
            </div>
      </div>
      <div class='form-field'>
            ...
      </div>
      ...
</form>
```

Tym razem rozbudowę widoku zacznijmy od napisania konstruktora w pliku *app.js*:

```
(function() {
      ...
      app.controller('DropDownListController', ['$scope', function($scope) {
            this.clicked = false;
            this.itemsList = [];

            for (var i = 1; i <= 4; i++)
                  this.itemsList.push('img/smile'+ i +'.png');


            this.toggle = function() {
                  this.clicked = !this.clicked;
            };

            this.select = function(id) {
                  this.clicked = false;
                  $scope.record.imgUrl = this.itemsList[id];
            };
      }]);
      ...
})();
```

Mając gotowy kontroler możemy powrócić do rozbudowy funkcjonalności w pliku *kalendarze.html*:

```
<div class='form-field'>
      <span class='sp'>Wybierz obrazek:</span><br />
```

```
      <div class="selection-wrapper"
                              ng-controller='DropDownListController as ddList'>
            <div class="selected-item-box" ng-click='ddList.toggle()'>
                  <span class="dropdown-icon"></span>
                  <ul class="items-list">
                        <li>
                              <img ng-src="{{ record.imgUrl }}"/>
                        </li>
                  </ul>
            </div>
            <div class="list" ng-show='ddList.clicked'>
                  <ul class="items-list">
                        <li ng-click='ddList.select($index)'
                                    ng-repeat='item in ddList.itemsList'>
                              <img ng-src="{{ item }}" />
                        </li>
                  </ul>
            </div>
      </div>
</div>
```

W powyższym kodzie najważniejsza jest dyrektywa definiująca zakres kontrolera *DropDownListController*. W tym przypadku zagnieżdżamy go w zakresie kontrolera *FormController* przez co ten pierwszy dziedziczy po nim. Posługujemy się tutaj zmienną $scope.record, która jest wspólna dla obu kontrolerów. Kontroler *DropDownListController* przekazuje informacje o wybranym obrazku kontrolerowi *FormController* za pośrednictwem tej zmiennej. Dyrektywa *ng-click* dodaje obsługę zdarzenia kliknięcia myszą na tym elemencie. Przy wystąpieniu tego zdarzenia zostanie wywołana funkcja *ddList.toggle()* (rozwija ona listę dostępnych obrazków).

Pozostaje już tylko dodanie odpowiednich stylów w pliku *style.css*:

```
...
.selection-wrapper {
    width: 200px;
    position: relative;
}
.selection-wrapper .selected-item-box {
    list-style-type: none;
    box-shadow: inset 1px -1px 5px 0 #CCCCCC;
    -webkit-box-shadow: inset 1px -1px 5px 0 #CCCCCC;
    -moz-box-shadow: inset 1px -1px 5px 0 #CCCCCC;
    -o-box-shadow: inset 1px -1px 5px 0 #CCCCCC;
    cursor: pointer;
    border: solid 1px #C7C6C7;
    padding: 5px;
    height: 80px;
    background-color: #fff;
    border-radius: 5px;
    -mox-border-radius: 5px;
    -webkit-border-radius: 5px;
    -o-border-radius: 5px;
}
.selected-item-box .items-list {
    list-style-type: none;
    width: 100%;
    padding: 0;
    margin: 0;
}
.selected-item-box .items-list li {
    display: inline;
}
.selected-item-box .items-list li img {
    display: inline;
    padding-left: 10px;
```

```
    height: 80px;
    font-size: 20px;
}
.selection-wrapper .list {
    height: 300px;
    overflow-y: auto;
    overflow-x: hidden;
    border-left: solid 1px #C7C6C7;
    border-right: solid 1px #C7C6C7;
    border-bottom: solid 1px #C7C6C7;
    z-index: 9999;
    position: absolute;
    width: 99%;
    border-radius: 5px;
    -mox-border-radius: 5px;
    -webkit-border-radius: 5px;
    -o-border-radius: 5px;
}
.list .items-list {
    list-style-type: none;
    width: 100%;
    padding: 0;
    margin: 0;
}
.list .items-list li {
    margin: 0;
    width: 100%;
    border-bottom: solid 1px #C7C6C7;
    padding: 5px;
    background-color: #fff;
}
.list .items-list li img {
    padding-left: 10px;
    padding-right: 6px;
    height: 80px;
}
.dropdown-icon {
    position: absolute;
    top: 50%;
    right: 0;
    margin-top: -8px;
    width: 20px;
    height: 20px;
    background-image: url("img/menu_arrow_down.png");
    background-repeat: no-repeat;
    background-position: 0 center;
}
```

Przetestujmy aplikację.

Zadanie 2 Implementacja dodatkowych funkcjonalności

W ramach samodzielnej pracy należy zaimplementować funkcjonalność:

a) edycji,
b) dodawania na początku i w środku serii,
c) zmiany kolejności,
d) usuwania,
e) sortowania,
f) filtracji,
g) itp.