SPRAWOZDANIE

Zaawansowane aplikacje WWW

Frameworki front-endowe oraz tworzenie Web API

14.12.2015

Karol Suwalski 125NCI B

https://github.com/SuwalskiKarol/orw.git

ASP.NET MVC WEB API

Web API jest frameworkiem ułatwiającym pisanie serwisów **REST**.

Czym jest REST?Jest to odpowiednio zapisany ciąg znaków w formie adresu URI, który następnie przysyłamy do zewnętrznej aplikacji metodą GET(na poziomie 2 i 3 na niższych jest chyba tylko POST, ale ja tam się nie znam to się nie wypowiadam.), co skutkuje zwrotem żądanych danych. Zapytania, które wysyłamy mogą być w formacie JSON, XML, ale może to być też zwykły tekst czy dane binarne. Czyli jak widać do komunikacji korzystamy z protokołu HTTP.

Można więc powiedzieć, że Web API ułatwia nam tworzenie serwisów HTTP. Web API jest również rozszerzeniem MVC, więc naturalnym staje się tworzenie aplikacji Web API przy wykorzystaniu wzorca MVC. MVVM się nie na daje do pisania ASP.NET, ten wzorzec jest polecany programistom WPF oraz Silverlight ze względu na wiązania.

Zasadnicza różnica między Web API a ASP.NET to sposób, w jaki wybierane są wywoływane metody. W Web Api akcje zwykle zwracają konkretne dane, a nie ActionResult. Akcja nie określa, w jaki sposób (JSON czy XML) mają zostać zwrócone. W ASP.NET MVC często akcję REST definiowało się w następujący sposób:

```
public ActionResult GetData()
        {
            Person persons = _repository.GetPersons();
return Json(persons);
        }
```

Analogiczna metoda w WebApi wygląda następująco:

```
public Person[] GetData()
        {
            Person persons = _repository.GetPersons();
return persons;
        }
```

AngularJS

Jestto framework JavaScript stworzony przez inżynierów z Google. Służy on do szybkiego i łatwego budowania aplikacji internetowych, tak zwanych – single app. Pozwala pogodzić idee JavaScript i modelu MVC, co czyni go idealnym frameworkiem do pracy z ASP.NET MVC Web API.

Kilka podstawowych elementów AngularJS wykorzystanych przy tworzeniu operacji CRUD.

DataBinding – a dokładnie "Two Way Data-Binding". Polega na automatycznej synchronizacji danych między widokiem a innymi JavaScriptowymi elementami aplikacji np. między kontenerem. Przykład:

HTML:

```
<divng-controller="MyCtrl">

<inputdata-ng-model="myVar"/>

<h1>jakiś napiś:{{myVar}}! - {{myVar.length}}</h1>

</div>
```

JAVASCRIPT

```
var myApp = angular.module('myApp', []);

function MyCtrl($scope) {
    $scope.myVar = 'Hello World';
}
```

W widoku HTML jak i w kontrolerze mamy zmienna myVar. Zmieniając wartość pola tekstowego automatycznie zmienia nam się wartość zmiennej po obu stronach.

Module- Moduł jest podstawową koncepcją frameworka AngularJS. Modułem może być w zasadzie wszystko: kontrolery, dyrektywy, serwisy itp. itd. Również sama aplikacja to tak na prawdę taki główny moduł.Deklaruje się to mniej więcej tak:

```
var app;


(function () {
    app = angular.module("crudModule", []);
})();
```

Jako pierwszy parametr podajemy nazwę modułu. Z kolei drugi parametr to lista nazw modułów, od których zależy dany moduł.

Controller – Jest to zwykły kontroler jak to przy MVC bywa. Jednak przy wykorzystaniu AngularJS możemy rozbudować kontroler o kilka funkcjinalności.

Na rzecz modułu wywołujemy funkcję controller, której jako pierwszy parametr podajemy nazwę kontrolera, a jako drugi, funkcję wywołania zwrotnego. Funkcja ta przyjmuje parametr **$scope** – jest to taka specjalna zmienna, którą można rozpatrywać, jako coś w rodzaju modelu. Przypisuje się jej właściwości, które później można „zbindować” do widoku – wszystkie te właściwości są automatycznie obserwowalne, więc każda ich zmiana w jednym miejscu na widoku powoduje zmianę także w innym miejscu.

```
app.controller('crudController', function ($scope, crudService) {

$scope.save = function () {
var Employee = {
            EmpNo: $scope.EmpNo,
            EmpName: $scope.EmpName,
            Salary: $scope.Salary,
            DeptName: $scope.DeptName,
Designation: $scope.Designation
        }

}
```

Services - Serwisy to obiekty, które mogą być dodane jako zależność w naszej aplikacji. Służyć mogą nam po to, aby odciążyć kontroler od nadmiaru funkcjonalności oraz co najważniejsze – wykorzystać tą moc można do udostępniania funkcjonalności między kontrolerami, które to kontroler nie posiada w standardzie.

Typy serwisów: provider, factory, service, value, constant.

Typ**Service** wykorzystuje wzorzec projektory, jakim jest constructor a więc zachowywać się będzie jak zwykła klasa.W AngularJS nie musimy serwisów deklarować w ten sposób:

```
var MojSerwis = new MojSerwis();
```

Wystarczy go zadeklarować jak constant czy value.

```
app.service('crudService', function ($http) {

this.post = function (Employee) {
var request = $http({
            method: "post",
            url: "/api/EmployeesAPI",
data:  Employee
        });
return request;
    }

}
```

Kilka dyrektyw wykorzystanych podczas pisania aplikacji.

ng-app–Stosowany w widoku. Informuje Angulara, że wszystko, co znajduje się wewnątrz elementu body to nasza aplikacja.

```
<htmlng-app="crudModule">
```

ng-controller– Stosowany przy kontenerach, z którymi wiążemy kontroler.

```
<tableid="tblContainer"ng-controller="crudController">
```

ng-model –Służy dopowiązania zmiennej modelu z innym elementem kodu HTML np. z polem tekstowym.Jeśli użytkownik zmieni wartość pola tekstowego, zmienna modelu zostanie automatycznie zaktualizowana i zmiana ta będzie od razu widoczna wszędzie tam, gdzie ma miejsce „bindowanie” do tej właściwości.

```
<inputtype="text"id="eno"readonly="readonly"ng-model="EmpNo"/>
```

ng-repeat - wszystko to, co znajduje się wewnątrz kontenera posiadającego tę dyrektywę wyrenderuje się tyle razy ile jest elementów tablicy list.

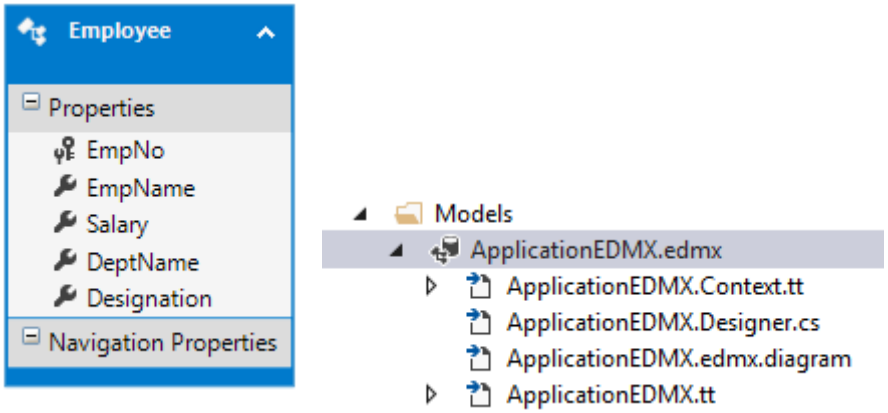
```
<tbodyng-repeat="EmpinEmployees">
<trng-click="get(Emp)">
<td><span>{{Emp.EmpNo}}</span></td>
<td><span>{{Emp.EmpName}}</span></td>
<td><span>{{Emp.Salary}}</span></td>
<td><span>{{Emp.DeptName}}</span></td>
<td><span>{{Emp.Designation}}</span></td>
</tr>
</tbody>
```

ASP.NET MVC WEB API w praktyce na przykładzie operacji CRUD

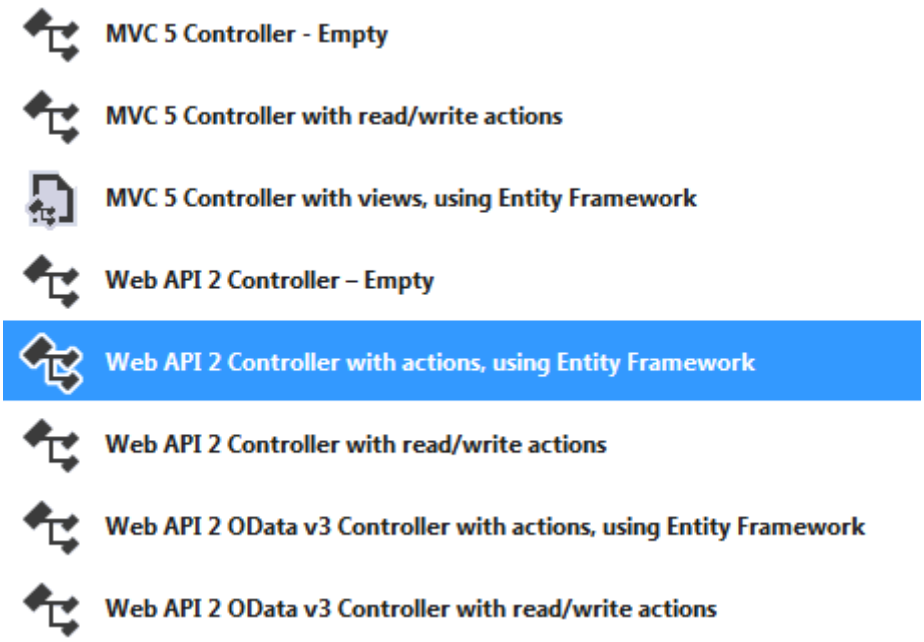
1. Na pewno potrzebne są dane, więc stwórzmy SQL server database w folderze przeznaczonym do przechowywania danych czyli App_Data.

```
CREATETABLE [dbo].[Employee](
    [EmpNo]       INTIDENTITY (1, 1)NOTNULL,
    [EmpName]     VARCHAR (100)NOTNULL,
    [Salary]      INTNOTNULL,
    [DeptName]    VARCHAR (50)NOTNULL,
    [Designation] VARCHAR (50)NOTNULL,
PRIMARYKEYCLUSTERED ([EmpNo] ASC)
      );
```

2. Następnie musimy stworzyć reprezentację utworzonej bazy danych w Modelu. Wykorzystajmy do tego ADO.NET Entity Data Model. Utworzy to nam całą strukturę bez potrzeby pisania modelu własnoręcznie.



3. Kolejnym krokiem jest stworzenie kontrolera Web API, który będzie zawierał metody HTTP.



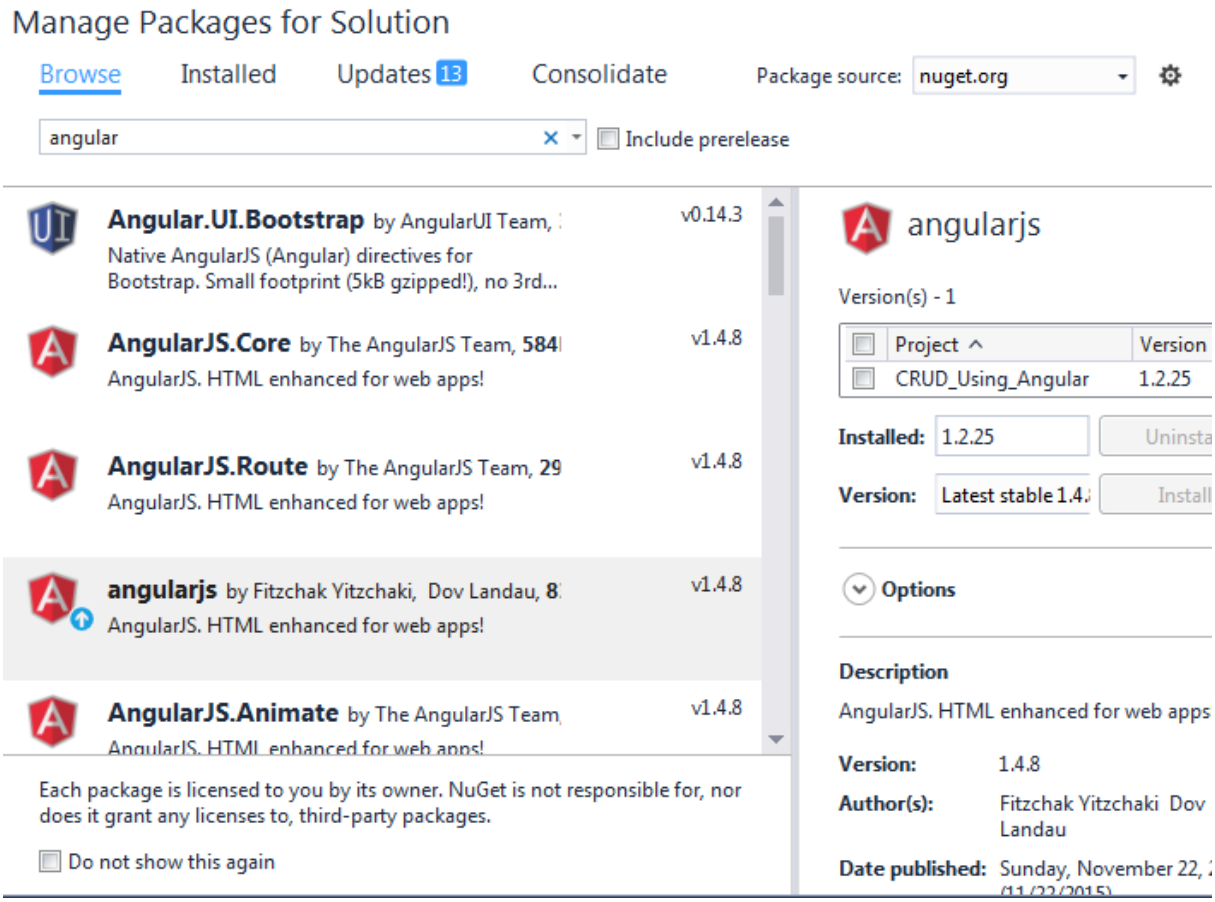
Dzięki Entity większość kodu utworzy się automatycznie. Przykład metody PUT

```
// PUT: api/EmployeesAPI/5
        [ResponseType(typeof(void))]
publicIHttpActionResult PutEmployee(int id, Employee employee)
        {
if (!ModelState.IsValid)
            {
return BadRequest(ModelState);
            }

if (id != employee.EmpNo)
            {
return BadRequest();
            }

            db.Entry(employee).State = EntityState.Modified;

try
            {
                db.SaveChanges();
            }
catch (DbUpdateConcurrencyException)
            {
if (!EmployeeExists(id))
                {
return NotFound();
                }
else
                {
throw;
                }
            }

return StatusCode(HttpStatusCode.NoContent);
}
```

4. Potrzebny jest pusty kontroler, który zwróci nam wszystko doIndex.cshtml w widoku.

```
publicclassEmployeeController : Controller
    {
// GET: Employee
publicActionResult Index()
        {
return View();
}
          }
```

5. Dzięki narzędziu *Manager NuGet Package* możemy pobierać nowe funkcjonalności do naszego projektu bez potrzeby używania konsoli *Package Manager Console*. Dodajemy AngularJS.



6. Tworzymy trzy skrypty:

Module.js

Tworzy nam moduł o nazwie crudModule. Potrzebny do inicjalizacji projektu przy dyrektywie ng-app.

```
var app;

(function () {
    app = angular.module("crudModule", []);
})();
```

Service.js

Używany do połączenia się z Web API używając metod $http get, post, put oraz delete. $http jest Angularowym wrapperem obiektu XmlHttpRequest.

- $http.$get(url): wysyła HTTP GET request do wskazanego adresu URL
- $http.post(url, dataToBePosted): Wysyła HTTP POST request do adresu URL
- $http.put(url, data): Wysyła HTTP PUT request
- $http.patch(url, data): Wysyła HTTP PATCH request
- $http.delete(url): Wysyła HTTP DELETE request

Przykład metody tworzenia nowego rekordu:

```
app.service('crudService', function ($http) {


//Create new record
this.post = function (Employee) {
var request = $http({
            method: "post",
            url: "/api/EmployeesAPI",
            data:  Employee
        });
return request;
        }
```

Controller.js

Może zawierać dodatkowe informacje o $http.W tym przypadku są to metody do wykonywania operacji HTTP: loadRecords, Save, Delete, Get, Clear.
Przykład operacji usuwania rekordu:

```
//Method to Delete
$scope.delete = function () {
var promiseDelete = crudService.delete($scope.EmpNo);
        promiseDelete.then(function (pl) {
            $scope.Message = "Usuwanie zakończone";
            $scope.EmpNo = 0;
            $scope.EmpName = "";
            $scope.Salary = 0;
            $scope.DeptName = "";
            $scope.Designation = "";
            loadRecords();
        }, function (err) {
            console.log("Err" + err);
});
    }
```

7. Czas na stworzenie widoku Index.cshtml. Oprócz czystego kodu cshtml czy też CSS musimy dodać Angularowe dyrektywy w celu zachowania zasady DataBinding oraz ng-app na początku pliku.

```
    <htmlng-app="crudModule">
```

Przykładowy wycinek kodu:

```
<tr>
<td>
<span>Tytuł</span>
</td>
<td>
<inputtype="text"id="desig"requiredng-model="Designation"/>
</td>
</tr>
<tr>
<td>
<inputtype="button"id="new"value="New"ng-click="clear()"/>
</td>
<td>
<inputtype="button"id="save"value="Save"ng-click="save()"/>
</td>
<td>
<inputtype="button"id="delete"value="Delete"ng-click="delete()"/>
</td>
    </tr>
```

Oczywiście nie możemy zapomnieć o dodaniu informacji Widokowi z jakich skryptów korzystamy.

```
<scriptsrc="~/Scripts/angular.js"></script>
<scriptsrc="~/Scripts/angular-route.js"></script>
<scriptsrc="~/Scripts/MyScripts/Module.js"></script>
<scriptsrc="~/Scripts/MyScripts/Service.js"></script>
<scriptsrc="~/Scripts/MyScripts/Controller.js"></script>
```