ერთგვერდიანი JAVASCRIPT - პლატფორმების ANGULARJS და EMBERJS განხილვა და შედარება

გიორგი კენჭოშვილი
საქართველოს ტექნიკური უნივერსიტეტი

რეზიუმე

განიხილება AngularJs და EmberJs SPA-პლატფორმები. ჩატარებულია ამ ორი პლატფორმის შედარებითი ანალიზი და გამოვლენილია თითოეულის უპირატესობანი და სირთულეები ვებ-აპლიკაციების აგების პროცესში. საილუსტრაციო მაგალითის სახით განხორციელებულია მარტივი SPA-აპლიკაციის იმპლემენტაცია ორივე პლატფორმის შემთხვევაში – AngularJs-ის და EmberJs-ის გამოყენებით.

საკვანძო სიტყვები: ერთგვერდიანი აპლიკაცია. SPA. AngularJS პლატფორმა. EmberJS პლატფორმა. თხელი კლიენტი. სქელი სერვერი.

1. შესავალი

ერთგვერდიანი აპლიკაცია (SPA – Single Page Application) ვებ-დანართია, რომელიც იტვირთება ერთ გვერდში და მომხმარებლის ურთიერთქმედებების საპასუხოდ, დინამიკურად ახდენს ვებ-გვერდის განახლებას [1]. ერთგვერდიანი აპლიკაციების კონცეფციის შემუშავებამდე, ვებ-აპლიკაციის ლოგიკის კოდირება სრულდებოდა მთლიანად სერვერზე, პრინციპით – „thin-client, thick-server“.

ერთგვერდიანი აპლიკაციების შექმნასთან ერთად, ამ ლოგიკის დიდმა ნაწილმა გადაინაცვლა კლიენტზე და ფრონტ-აპლიკაციების დეველოპერი უკვე კარგად უნდა იყოს გათვითცნობიერებული და ფლობდეს საკმაო გამოცდილებას ამ განხრით. მეტწილად, ერთგვერდიანი აპლიკაციის პლატფორმების დიზაინი სრულდება MVC ფორმატის მიხედვით, ან მისი ვარიაციით. ამ ვარიაციებს ასახელებენ ფორმატებში MV* პლატფორმებად. SPA-აპლიკაციებთან მუშაობას აიოლებს დღეს-დღეობით არსებული ღია კოდის მქონე (Open Source) JavaScript-პლატფორმები, როგორიცაა AngularJS და EmberJS [2,3].

ნაშრომში განიხილება AngularJs და EmberJs JavaScript პლატფორმები და მათი შედარება. საილუსტრაციო თვალსაზრისით, განხორციელებულია მარტივი SPA-აპლიკაციების იმპლემენტაცია.

2. მოდელები

მოდელები – ობიექტებია, რომლებიც წარმოადგენს აპლიკაციის მიერ გამოყენებად მონაცემებს. ყველა მოდელის კომბინაცია განსაზღვრავს აპლიკაციის მდგომარეობას (state). AngularJs-ის და EmberJs-ის მსგავს ერთ-გვერდიან პლატფორმებს აქვს მეთოდები – მოდელებთან ურთიერთქმედების და მართვის მიზნით.

2.1. მოდელები AngularJs -ში

AngularJs-ი მოდელებისთვის იყენებს გეგმას „ol' ჯავასკრიპტის ობიექტები“ (POJSOs).

```
var myModel = {id: 1, name: 'foo1'};
```

ამ მოდელის მდგომარეობის (state) შეცვლა ასევე ძალიან მარტივია, რადგანაც სინტაქსი არ განსხვავდება POJSOs-ის მანიპულირებისაგან:

```
myModel.name = 'myNewName';
```

მსგავსად ამისა, state-ზე წვდომა ძალზე მარტივია:

```
myModel.name;  // (evaluates to "myNewName")
```

რაში მდგომარეობს ამ სიმარტივის საიდუმლო ? ამ მოდელების გამოყენების წინ ისინი უნდა დაყენდეს როგორც $scope ცვლადის თვისება („Property”) კონტროლერში:

amisa:$scope.myModel = myModel;

თუმცა თავისი შედეგები მოყვება POJSOs-ის გამოყენებას მოდელების სახით, როდესაც იგება მონაცემები მოდელებსა და წარმოდგენებს (view-ებს) შორის. გამოდის, რომ საჭიროა „ცვლილებების შემოწმება“ („dirty checking“) - თითოეული მოდელის ასლის შენახვა წინარე მდგომარეობაში და კონკრეტულ ინტერვალზე განსხვავებების შემოწმება.

აღიშნული დაკავშირებულია მძიმე თვისობრივ გაუარესებასთან, როდესაც ბევრი მოდელია, ან როდესაც მოდელები ძალზე კომპლექსური ხდება; მსგავსი შეზღუდვებით სამუშაოების შესრულება საჭიროებს $scope, $watch, „ცვლილებების შემოწმების“ და შესრულების ციკლის („run loops“) ღრმა ცოდნას. როგორც ითქვა, „ცვლილებების შემოწმება“ საკმაოდ ღირებული უპირატესობაა უმეტეს შემთხვევებში, რადგანაც ვებ-აპლიკაციების უმეტესობას არ გააჩნია ბევრი მოდელი, რომლებზეც ერთდროულად უნდა გამახვილდეს ყურადღება, ან არ აქვს მოდელები, რომლებიც საკმაოდ კომპლექსურია, იმისათვის, რომ თვისობრივი გაუარესება გამოიწვიოს.

2.2. მოდელები EmberJs -ში

EmberJs-ში, მოდელის გამომხატველი სინტაქსის გამოყენება უფრო კომპლექსურია ვიდრე AngularJs-ში, მსგავსად POJSOs-სა არ შეიძლება იყოს პირდაპირ გამოყენებული, და ამის მაგივრად უნდა იყოს შემოხვეული ("wrapped") EmberJs-ის ობიექტების გამოყენებით, შემდეგის მსგავსად:

```
var Foo = Ember.Object.extend();
var myModel = Foo.create({ id: 1, name: 'foo1' });
```

აქ აღსანიშნავია ის ფაქტი, რომ თავდაპირველად განისაზღვრება კლასი მოდელისთვის Foo, ხოლო შემდეგ იქმნება ამ კლასის რეალიზაცია (instance), myModel. ეს სრულდება იმისათვის, რომ EmberJs-ში მოდელები უნდა შეესაბამებოდეს კონკრეტულ ინტერფეისს, რათა შეძლოს დანარჩენ ფრეიმვორკთან მუშაობა. POJSOs-ი არ უზურნველყოფს ამ ინტერფეისს, აქედან გამომდინარე უნდა განისაზღვრავთ კლასი ამ მოდელებისათვის მანამ, სანამ შესრულდება რეალიზაცია. ყველაზე მნიშვნელოვანი და ამ ინტერფეისის ხშირად გამოყენებადი ნაწილებია წვდომის და ცვლილების თვისებები:

myModel.set('name', 'newFooName'); ცვლის მოდელის მდგომარეობას.
myModel.get('name'); კითხულობს მოდელის მდგომარეობას.

ზოგადად შეიძლება დაისვას კითხვა, რომ თუ AngularJs-ს შეუძლია გამოიყენოს POJSOs-ი მოდელებისათვის, რატომ არ შეიძლება იგივე შეასრულოს EmberJs-მას? ამის ერთ-ერთი მთავარი მიზეზია ის, რომ შეუძლებელია მეთვალყურის („observers“) შექმნა თვისებების ცვლილებისთვის POJSOs-ზე. ამრიგად, ვრეპერის კლასი გამოიყენება თვისებების ცვლილების მეთვალყურეობისათვის, მაშინ როდესაც .get() და .set() არის გამოძახებული.

შეიძლება ითქვას, რომ ვრეპერის ობიექტები წვდომებით (EmberJs' მიდგომა) და „ცვილებების შემოწმება“ (AngularJs' მიდგომა), არის უბრალოდ ერთი და იგივე პრობლემის გადაჭრის სხვადასხვა გზა, მოდელებზე მდგომარეობის (state) ცვლილების მეთვალყურეობა.

3. წარმოდგენა (View)

რა ფუნქციური დატვირთვა აქვს წარმოდგენას ? View-ებს ორივე შემთხვევაში - AngularJs-ის და EmberJs-ის დროს იშვიათად ეხება დეველოპერი. ეს იმიტომ ხდება, რომ მათში არის „შიგა ელემენტები", რომლებიც ზრუნავს რთულად მოსაგვარებელ საკითხებზე:

- წყვეტენ როდის გამოიტანონ ნიმუშები;
- ორმხრივი გადაბმა (two way binding).

მოვლენის დამუშავება (event handling) / პროქსირება(proxing) / ბაბლინგი (bubbling).

აღნიშნული სუფთა ჰაერის ნაკადივითაა მათთვის, ვისაც ვებ-აპლიკაციები შეუქმნია BackboneJs-ის გამოყენებით, ეს არის ერთ-ერთი რუტინული საქმე, რომლის შესრულება თქვენ არასდროს აღარ დაგჭირდებათ. თუმცა, დეველოპერებმა უნდა გამოიყენონ ვიუები, როდესაც ისინი უფრო რთული დონის საქმეს ასრულებენ, მაგალითად:

- მესამე მხარის ბიბლიოთეკასთან ინტეგრაციის დროს, რომელიც არ უზრუნველყოფს AngularJs ან EmberJs ვრაპერებას;
- რაიმე სპეციფიურის შესრულება დამუშავების დასრულებისთანავე.

სხვადასხვა ვიუებს ან კლასებს შორის კოდის გაცვლისთვის „ნაზავების" (mixins) გამოყენება ან მემკვიდრეობით მიღება.

View-ებისთვის AngularJs-ში არ მოიძებნება რაიმე ჯავასკრიპტი. ამის სანაცვლოდ, უბრალოდ გამოიძახება ngView დირექტივა ნიმუშში. ნებისმიერი ჩვეულებრივი View-ის ლოგიკა უნდა იყოს გამოყენებული მასზე პასუხისმგებელ კონტროლერში.

EmberJs-ში შესაძლებელია View-ს აწყობა ჯავასკრიპტში. როგროც ზემოთ აღვნიშნეთ, ერთი მიზეზი იმისა, რომ ჩვენ ამის შესრულება მოვინდომოთ, არის მესამე პირის ბიბლიოთეკასთან ინტეგრაცია, რომელიც არ არის აუცილებელი რომ Ember-მა გაიგოს.

```
varmyFooView = Ember.View.extend ({
  didInsertElement: function(){
     this._super.apply(this, arguments);
     varsvg = this.$('svg').get()[0];
     d3.select(svg); //do something with d3 and the <svg> element }
});
```

აქ ჩვენ არ გაგვითვალისწინებია View-ების didInsertElement ფუნქცია, რადგანაც ხდება მათი გაშვება ვიუს მიერ მისი კონტენტის გამოტანისას და ელემენტის მანიპულაციის გაშვება მასში რეალიზაციის დროს. ამის სადმე სხვაგან შესრულება არ იქნება შედეგიანი, იმიტომ რომ d3-ს არ ესმის თუ როდის (ან არ უნდა) შეასრულოს მისი ქმედებები EmberJs ვიუში, რადგანაც ის ვერ ცნობს EmberJs.

4. კონტროლერები

კონტროლერები AngularJs-ში არ არის, მკაცრად რომ ვთქვათ, კონტროლერები. ფაქტობრივად ისინი არ ირქმევს MVC პლატფორმის სახელს. ამის ნაცვლად ისინი MVW (მოდელი-ვიუ-სხვარამ) ფრეიმვორკად ისახელებს თავს. პრაქტიკული თვალსაზრისით, AngularJs-ის კონტროლერები კონტრო-ლერებია. სინტაქსი:

```
angular.module('application', []).controller('FooCtrl', function($scope){
    $scope.someProperty = 'More exclamation marks';
    $scope.someAction = function(){
       $scope.someProperty += '!';     };
});
```

$scope ობიექტი არის პროტოტიპულად მემკვიდრეობით მიღებული მისი მშობელი $scope ობიექტისგან - ამ შემთხვევაში მთავარი გამოყენების ობიექტი. ეს გახდა ხელმისაწვდომი AngularJs-ის დამოკიდებულების საინექციო პლატფორმით. ეს თავისთავად მომხიბლავი თემაა და უთუოდ განსახილველია, რადგანაც წარმოადგენს პროგრამული უზურნველყოფის მშვენიერ საინჟინრო და არქიტექტურულ მაგალითს. მისი გაგება დაკავშირებულია AngularJs-ის ცოდნის სათავეებთან.

ორმხრივი გადაბმა (Two way binding) AngularJs-ში სინტაქსი:

```
<divng-controller="FooCtrl">
<buttonng-click="someAction()">Press me</button>
<p>{{someProperty}}</p>
<inputtype="text"value="{{someProperty}}">
</div>
```

ნიმუში არ არის საკმარისი. თუმცა, ის უზურნველყოფს ერთი და იმავე თვისებების კონტროლერის $scope -ზე მოდიფიკაციის ორ განსხვავებულ გზას და ამრიგად ის ემსახურება ორმხრივი გადაბმის მოკლე დემონსტრირებას. ეს არის ორმხრივი გადაბმა ქმედებაში, პირველი თანმიმდევრობით - ვიუდან მოდელამდე და მეორე თანმიმდევრობით - მოდელიდან უკან ვიუმდე.

EmberJs-ის კონტროლერები მათი წმინდა MVC მნიშვნელობას ადასტურებენ. სინტაქსი:

```
App.FooController = Ember.Controller.extend({
    someProperty: 'More exclamation marks',
    actions: {
      someAction: function(){
        this.set('someProperty', this.get('someProperty') + '!');  }
    }
  });
```

EmberJs-ს ქმედებაში მოყავს თვისებებსა და ქმედებებს შორის წმინდა გაყოფა კონტროლერის შიგნით - ყველა ქმედება დაჯგუფებულია ერთ ჰეშში.

აქვე აღსანიშნავია სხვა შემთხვევა, AngularJs-ისგან განსხვავებით, EmberJs-ი განასხვავებენ კონტროლერზე თვისებების პარამეტერებს და კონტროლერის მიერ უტილიზირებულ მოდელებს. მოდელები ტიპურად იქმნება და გადაეცემა კონტროლერს Ember.Route ობიექტის მიერ, რომელსაც ჩვენ შევეხებით მარშრუტიზატორის განხილვის დროს. ეს იმიტომ ხდება, რომ წარმოდგენილია ნებისმიერი რეალიზებული კონტროლერის („controller instantiated“) მხოლოდ ერთი რეალიზაცია და, ამრიგად მისი მდგომარეობის (state) გადატვირთვა არ სრულდება ყოველ ჯერზე. ეს შეიძლება დამაბნეველიც იყოს EmberJs-თან მუშაობის პირველ ეტაპზე.

ასევე აღსანიშნავია, რომ ჩვენ მოგვიწია ამ კონტროლერისთვის FooController სახელის მინიჭება. AngularJs-ში, ჩვენ გვეძლეოდა კონტროლერისთვის სახელების მინიჭების სრული თავისუფლება - BarCtrl ზუსტად ისევე იმუშავებდა როგორც FooCtrl. თუმცა, EmberJs-ში, ჩვენ უნდა გამოვიჩინოთ სიფრთხილე კონტროლერებისთვის სახელების შერჩევისას - და ასევე სხვა მრავალი სახის ობიექტებისთვის - სახელები ენიჭება დასახელების შერჩევის ზოგადად მიღებული წესის დაცვით.

ორმხრივი გადაბმა (Two way binding) EmberJs-ში. სინტაქსი:

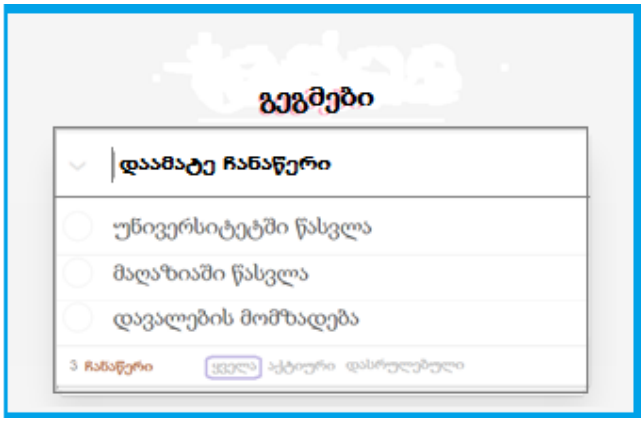
```
<div>
<button{{action 'someAction'}}>Exclaim harder</button>
<p>{{someProperty}}</p>
{{inputtype="text' value=someProperty}}
</div>
```

EmberJs-ში ორმხრივი გადაბმა მუშაობს ზუსტად იმავენაირად, როგორც AngularJs-ში, ამიტომ ჩვენ მხოლოდ განსხვავებებს განვიხილავთ.

<p> ტაგის კონტენტი ებმება და მისი ქცევა ამართლებს მოლოდინებს. თუმცა, ელემენტების ატრიბუტები პრობლემატური ხასიათისაა - ჩვენ არ შეგვიძლია ნებისმიერი მოდელის მნიშვნელობის გადაბმა DOM ელემენტის ატრიბუტებთან ჩვეულებრივი ფიგურული ფრჩილების სინტაქსის გამოყენებით. ამის მიზეზია ის, რომ EmberJs-ი სვავს <script> ტაგებს - ერთს წინ და ერთს შემდგომ - ნიმუშის თითოეულ გადაბმულ სექციაში; და სანამ ეს კარგად მუშაობს ელემენტების გარეთ, აღნიშნული სათანადოდ არ მუშაობს ატრიბუტებისთვის. ამიტომ, ჩვენ უნდა გამოვიყენოთ {{bind-attr}} „Handlebars helper („საჭის დამხმარე“). რადგანაც ეს არის ჩვეულებრივად გამოყენებადი <input>ტაგებისთვის, არსებობს {{input}} საჭის დამხმარე რომელიც ამას ჩვენთვის შეასრულებს.შედეგად, ორმხრივი გადაბმის ამუშავება EmberJs-ში უფრო რთული შეიძლება აღმოჩნდეს, რადგანაც სინტაქსის უფრო დიდი მოცულობაა შესასწავლი.

5. SPA - მარტივი აპლიკაციის იმპლემენტაცია AngularJs და EmberJs პლატფორმებზე

ახალი ტექნოლოგიების შესაძლებლობები დემონსტრირებულია აპლიკაციაში Todos (გეგმები), რომელიც წარმოადგენს SPA-კონცეფციაზე შემუშავებულ ვებ-საიტს, AngularJs და EmberJs ფლატფორმებზე (ნახ.1). სამომხმარებლო ინტერფეისში ინტერაქტიულობას უზრუნველყოფს AngularJs ან EmberJs პლატფორმა, ანიმაციებისა და ტრანსფორმაციების ეფექტი მიღწეულია CSS3 გამოყენებით. სერვისის დონე იმპლემენტირებულია ASP.NET Web API ტექნოლოგიით, რომელიც უზრუნველყოფს JSON მონაცემების მიღება/გაგზავნას.



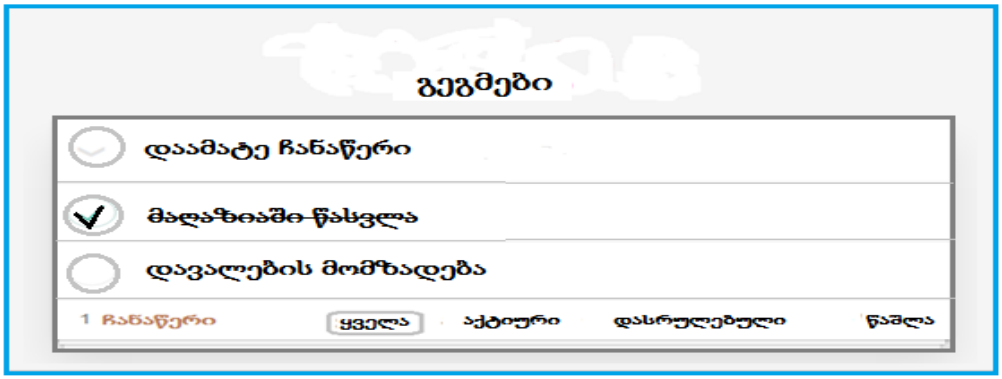
ნახ..1. AngularJs პლატფორმა, მომხმარებლის გეგმები

Todos ეს არის მარტივი აპლიკაცია, რომელიც სთავაზობს მომხმარებლებს ჩაინიშნონ საკუთარი განსახორციელებელი გეგმები და აწარმოონ მათი შესრულება. აპლიკაციას გააჩნია ფუნქციონალი ახალი გეგმის დამატების, შესრულების, შესრულებული გეგმების წაშლის, აქტიური გეგმების ნახვის და ა.შ. (ნახ.2).

ვებ-აპლიკაცია შემუშავებულია Visual Studio ხელსაწყოში. ახალი პროექტის შექმნისას გამოყენებულია ASP.NET Web Application პროექტის ტიპი და ჩართულია MVC და Web API შაბლონების გამოყენება. მიუხედავად იმისა, რომ წარმოდგენილია ერთი აპლიკაცია ორი სხვადასხვა პლატფორმით, მომხმარებელი მას აღიქვამს როგორც ერთიდაიმავე აპლიკაციას. ამიტომ, დანამდვილებით შეგვიძლია ვთქვათ, რომ სამომხმარებლო დონეზე ამ ორ პლატფორმას შორის განსხვავება არ არსებობს, განსხვავება საგრძნობია ტექნიკურ ნაწილში.

6. დასკვნა

ერთგვერდიანი აპლიკაციები (SPA) სრულიად ახალი, განსხვავებული მიდგომაა თანამედროვე ვებ-აპლიკაციების დაპროგრამების მეთოდოლოგიაში. ამ ორი ერთგვერდიანი პლატფორის სხვადასხვა ასპექტების შედარებისას EmberJs-ის შეთავაზება გაცილებით უფრო რთულია და რთულად შესასწავლი, ვიდრე AngularJs-ისა. ამის მიზეზი ისაა, რომ AngularJs-ის პლატფორმის შეთავაზებები არის დაბალი დონის და უფრო მოქნილია.



ნახ.2. EmberJs პლატფორმა: გეგმის დასრულების მონიშვნა

EmberJs-ში გადაწყვეტილებების უმეტესობა მომხმარებლის მაგივრად არის მიღებული - ისინი შეიძლება იყოს უხეში „გამკაცრებული ელემენტის, იზოლირებული შინაარსის, ვირტუალური შემცველობის" და ა.შ. სასარგებლოა იმის ცოდნა, რომ ორივე პლატფორმა თანხმობაში მოდის სპეციფიკაციებთან, იმიტომ რომ ამ პლატფორმების საშუალებით შექმნილი აპლიკაციები თავისთავად თითქმის მთლიანად შესაბამისობაშია სტანდარტებთან და, აქედან გამომდინარე, ისეთ სარგებლობას მოგვცემს, როგორიც არის მომავალი „ქროს-ბრაუზერის" მხარდაჭერა და მომგებიანობა.

ლიტერატურა:

1. Freeman A. (2014). Pro AngularJS (Expert's Voice in Web Development). 1st ed., Apress, ch. 3, pp. 45-50.

2. Guides and Tutorials. https://guides.emberjs.com/v2.2.0/ . გადამოწმ. 05.12.2015

3. კივილაძე ნ. (2015). რევოლუცია ASP.NET პლატფორმაზე: Web API და Angular.js ტექნოლოგიების გამოყენებით ერთგვერდიანი აპლიკაციების შემუშავება. საერთ.სამეცნ.კონფ. ივ.ფრანგიშვილის 85 წ.: „საინფორმაციო და კომპიუტ.ტექნოლოგიები, მოდელირება და მართვა". სტუ. თბ., გვ.229-233.

DISCUSSION AND COMPARISON OF SINGLE PAGE APPLICATION JAVASCRIPT PLATFORMS ANGULARJS AND EMBERJS

Kentchoshvili Giorgi
Georgian Technical University

Summary

Considers SPA-platforms AngularJs and EmberJs. A comparative analysis of these platforms and their advantages identified, and implementation complexity when building web applications. As an illustrative example of a simple implementation of proposed SPA applications using both platforms AngularJs and EmberJs.

ОБСУЖДЕНИЕ И СРАВНЕНИЕ ОДНОСТРАНИЧНЫХ JavaScript ПЛАТФОРМ AngularJs И EmberJs

Кенчошвили Г.
Грузинский Технический Университет

Резюме

Рассматриваются SPA-платформы AngularJs и EmberJs. Проведен сравнительный анализ этих платформ и выявлены их преимущества и сложности реализации при построении веб-аппликаций. В качестве иллюстративного примера предлагается имплементация простой SPA аппликации с использованием обеих платформ AngularJs и EmberJs.