გელა ღვინეფაძე

AngularJS

„ტექნიკური უნივერსიტეტი“

saqarTvelos teqnikuri universiteti

გელა ღვინეფაძე

AngularJS

რეკომენდებულია საქართველოს
ტექნიკური უნივერსიტეტის
სარედაქციო-საგამომცემლო საბჭოს
მიერ, 29.03.2017, ოქმი №1

თბილისი

2017

uak 681.3.06

წიგნში განხილულია Angular ფრეიმვორკი - Google Inc კორპორაციის მიერ შექმნილი, თვისებრივად გაუმჯობესებული ვებგამოყენებების (დანართების) დასაპროექტებლად განკუთვნილი, JavaScript ენაზე დაწერილი და MVW (Model-View-Whatever) მიდგომაზე აგებული პროგრამული პროდუქტი.

დამხმარე სახელმძღვანელო განკუთვნილია ინფორმატიკის სპეციალობათა შემსწავლელი სტუდენტებისა და ვებდანართების დაპროექტების საკითხებით დაინტერესებული პირებისათვის.

რეცენზენტები: საქართველოს ტექნიკური უნივერსიტეტის
პროფესორი გია სურგულაძე,

საქართველოს ტექნიკური უნივერსიტეტის
პროფესორი ვლადიმერ კეკენაძე

ISBN 978-9941-20-794-5

http://www.gtu.ge

AngularJS

1. შესავალი

*რას წარმოადგენს **AngularJS**?*

Angular არის JavaScript ენაზე დაწერილი, თვისებრივად გაუმჯობესებული ვებგამოყენების (დანართების) შესაქმნელად განკუთვნილი MVW (Model-View-Whatever) ფრეიმვორკი.

იგი დამუშავებულია და ვითარდება Google Inc კორპორაციის მიერ და უშუალო დანიშნულების გარდა, წარმოდგენას გვიქმნის, თუ როგორი იქნება ვებსივრცის მომავალი.

ტერმინ MVW (Model-View-Whatever)-ის არსი დაახლოებით ასე შეიძლება განიმარტოს:

მისი მთავარი ღირსებაა მოქნილობა - ნებისმიერი გარემოს ამსახველი ინფორმაციული ბლოკის სტრუქტურული ელემენტების ამა თუ იმ შაბლონის სახით წარმოდგენა და მათი არჩევის ხელმისაწვდომობა, რაც მნიშვნელოვნად ამარტივებს ვებგამოყენების შექმნის პროცესს. მაგალითად, ამ მიზნით,

შეგვიძლია ავირჩიოთ MVC (Model-View-Controller, იხ. დანართი_1) ან MVVM (Model-View-ViewModel) მოდელი.

AngularJS ფრეიმვორკს ახალ დონეზე აჰყავს HTML ენისა და მასთან ურთიერთობის შესაძლებლობები, ამ ენის ელემენტებს სძენს დინამიკურობას.

AngularJS თავის თავში მოიცავს როგორც კლიენტის მხარეს მოქმედი JavaScript ენაზე დაწერილი სცენარების, ასევე სერვერული ენებისათვის დამახასიათებელ კონცეფტუალურ შესაძლებლობებსაც.

ამ ფრეიმვორკში მონაცემებთან მუშაობისათვის გამოყენებული მიდგომა განსხვავდება ისეთი ფრეიმვორკების აგების კონცეფციისაგან, როგორიცაა, მაგალითად, Backbone.js და Ember.js. AngularJS არ არის მიბმული კონკრეტულ DOM-ზე და ოპერირებს სიტუაციის მიხედვით: DOM-ს განაახლებს JavaScript-ის ობიექტებში მოდელის შესახებ არსებულ ინფორმაციაზე დაყრდნობით. ამრიგად, აქ ეკრანზე გამოსატანი HTML მონაცემები მოდელში ინახება მცირე ზომის შაბლონის სახით. შედეგად, ელემენტზე ჩატარებული საჭირო გარდაქმნები ადვილად ტირაჟირდება ყველა საჭირო შემთხვევაში. მაშასადამე, AngularJS არ არის დამოკიდებული HTML-ზე, მეტიც - იგი მას აუმჯობესებს, სძენს ახალ თვისებებს, რადგანაც საქმე გვაქვს მონაცემების დაფიქსირების **ორ შრესთან:**

მოდელთან და ეკრანზე გამოსულ **სახესთან.**

ამ შრეებს შორის კავშირი ორმხრივია - მოდელში განხორციელებული ცვლილებები გავლენას ახდენს ეკრანზე გამოსატანი მონაცემის სახეზე და პირიქით. მოდელისა და სახის სინქრონიზაციისათვის კი გამოიყენება JavaScript-ში არსებული მარტივი ობიექტები.

შესაბამისი გარდაქმნებისათვის კი AngularJS მიმართავს JSON ფორმატს. ამასთან აღსანიშნავია, რომ მის მიერ განსაკუთრებით ეფექტიანად გამოიყენება REST მეთოდი.

2. ერთფურცლოვანი დანართები **(Single Page Application, SPA),** მათი მდგომარეობის მართვა და **Ajax (HTTP)**

SPA **ერთფურცლოვანი დანართის** ფუნქციონირებისათვის საჭირო კოდი (HTML, CSS და JavaScript) სერვერიდან ან ერთბაშად გადმოიტვირთება კლიენტის მხარეს, ანდა იგი ჩაიტვირთება დინამიკურად, როგორც წესი,

მომხმარებლის ქმედებებიდან გამომდინარე. ხშირად SPA დანართის სერვერთან ურთიერთობის აღნიშნული პროცესი სრულდება ფონურ რეჟიმში, ვებფურცლის გადატვირთვის გარეშე.

ადრინდელი მიდგომებისას, როდესაც ინფორმაცია პროგრამის მდგომარეობის შესახებ სერვერზე ინახებოდა, ზოგჯერ ქსელში იგვიანებდა მომხმარებლის ქმედებებზე დროულად რეაგირება, რის გამოც კლიენტის მხარეს ფურცლის სახე და სერვერზე არსებული მისი მოდელი ერთმანეთისაგან განსხვავდებოდა. ამასთანავე, ზოგჯერ ვერ ხერხდებოდა ამ ცვლილებების დამახსოვრებაც, ისინი იკარგებოდა და აუცილებელი ხდებოდა ვებფურცლის განახლება სერვერიდან მისი ხელახლა მთლიანად გადმოქაჩვით.

AngularJS ფრეიმვორკის საშუალებებზე დაყრდნობით კი ინფორმაცია დანართის მდგომარეობის შესახებ ინახება ბროუზერში, ხოლო ფურცელზე შეტანილი ცვლილებების შესახებ შეტყობინებები სერვერს გადაეცემა Ajax (HTTP)-ის მეშვეობით, რომელიც ამ მიზნით იყენებს GET, POST, PUT და DELETE მეთოდებს.

API დანართების შექმნისას AngularJS-ის საშუალებების გამოყენებაზე ორიენტირებული მიდგომის უდავო ღირსებაა ის, რომ ამ დანართების სტრუქტურის ფორმირებისა და გამართვა-ტესტირების პროცესები ერთმანეთის მსგავსი და გამარტივებულია.

3. მოდულები

Angular გამოყენებებს (დანართებს) ქმნის მოდულების მეშვეობით. მოდული შესაძლებელია იყოს ავტონომიური ან სხვა მოდულებზე დამოკიდებული. იგი დანართის სხვადასხვა ნაკვეთისათვის ასრულებს არაერთხელ გამოყენებადი კონტეინერის როლს.

მოდულის შესაქმნელად გამოიყენება გლობალური ობიექტი Object, ფრეიმვორკის სახელთა სივრცე და module მეთოდი.

3.1 სეტერები (setters).

გამოყენებისათვის გათვალისწინებულია (დანიშნულია) ერთი, app სახელწოდების მქონე მოდული:

```
angular.module('app',[]);
```

მეორე არგუმენტი მასივია და ჩვეულებრივ მიუთითებს იმ მოდულებზე, რომლებზეც დამოკიდებული არის მოცემული მოდული და რომლებიც მიუერთდება მოცემულს (აქ მასივი ცარიელია).

3.2 გეტერები (Getters)

Controllers, Directives, Services და სხვა შესაძლებლობებისადმი მიმართვისათვის უნდა დავეყრდნოთ არსებულ მოდულს. ამ შემთხვევაში მეორე არგუმენტი აღარ გამოიყენება:

```
angular.module('app');
```

3.3 მოდულებთან მუშაობა

მოდული ამგვარად გამოიძახება და შეინახება ცვლადის დახმარებით:

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

ამის შემდეგ უკვე შესაძლებელი ხდება, `app` ცვლადი გამოყენებული იქნეს დანართის შესაქმნელად.

3.4 HTML ბუტსტრაპი

დანართი, ჩვეულებრივ, DOM-ში ინახება `<html>` ელემენტის მიერ შესაბამისი ატრიბუტის გამოყენების შედეგად:

```
<html ng-app="app">

  <head></head>

  <body></body>

</html>
```

თუ JavaScript-ის სცენარების შემცველი ფაილების ჩატვირთვა ასინქრონულად ხდება, app დანართი ჩაიტვირთება ოპერატორით:

angular.bootstrap(document.documentElement, ['app']);

4. რა არის $scope?

$scope განიმარტება ხილვადობის უბნად, ასედაც - DOM-ში არსებულ მონაცემთა და JavaScript-ის სცენარს შორის გადებულ ავტომატურ ხიდად. ითვლება, რომ $scope სცენარში ასრულებს მოდელის წარმომდგენის (ViewModel) როლს. იგი გამოიყენება მხოლოდ კონტროლერების შიგნით და ახდენს მათში გამოცხადებული მონაცემების მიბმას ამ მონაცემების ეკრანულ წარმოდგენასთან (View-სთან) .

მოგვყავს კონტროლერში მონაცემების გამოცხადების მაგალითი:

```
$scope.someValue = 'Hello';
```

შემდეგ დგება საკითხი DOM-ში ამ `someValue` მნიშვნელობის ასახვისა. იგი წყდება Angular-სთვის DOM-ში მონაცემის ჩასადგმელი ადგილის მითითებით ანუ HTML-ის ელემენტთან კონკრეტული კონტროლერის მიბმით, მაგალითად, ამგვარად:

```
<div ng-controller="AppCtrl">

  {{ someValue }}

</div>
```

ცხადია, JavaScript-ის სცენარში არსებული ნებისმიერი ტიპის მქონე მონაცემებისათვის რაც მეტი კონტროლერი შეიქმნება და DOM-ის მონაცემებთან კავშირი დამყარდება, საქმე გვექნება მით მეტ ხედვის უბანთან და მათ იერარქიაში გასარკვევად მიმართავენ $rootScope ცვლადს.

4.1 $rootScope

$rootScope ცვლადის არსს განმარტავენ, როგორც საწყისი (ფესვური) დონის $scope ობიექტს. სწორედ მისგან გავდივართ ყველა სხვა დონის ხედვის უბანზე. $rootScope შესაძლებელია გამოყენებული იქნეს ხედვის ერთი უბნიდან მეორეში მონაცემების გადასაცემად.

5. კონტროლერები

კონტროლერი ორმხრივ კავშირს ამყარებს წარმოდგენასა და მოდელს შორის - ცვლილებები ერთ მათგანში აისახება მეორეშიც. ამრიგად, კონტროლერი წარმოდგენასა და მოდელს შორის შუამავალია - იგი იღებს მოთხოვნას მომხმარებლისაგან, აანალიზებს მას და შემდგომი დამუშავებისათვის გადასცემს სისტემის შესაბამის რგოლს, რითაც ხორციელება ერთმანეთისაგან განცალკევებული ბიზნესლოგიკისა და წარმოდგენის (პრეზენტაციის) ლოგიკის შეთანხმებული მუშაობა.

როგორც უკვე ვნახეთ, კონტროლერის შექმნა ამგვარად ხდება:

```
angular

  .module('app', [])
  .controller('MainCtrl', function () {
});
```

ჩანს, რომ კონტროლერს გადაეცემა ორი არგუმენტი - გამოძახებისათვის განკუთვნილი, მისი მაიდენტიფიცირებელი სახელი და შესასრულებელი ფუნქცია, რომელიც რეალურად კონტროლერის სხეულს წარმოადგენს.

შესაძლებელია, ზემოთ მოტანილ ჩანაწერს ასეთი, უფრო გამჭვირვალე სახეც მივცეთ (ქვემოთ მოყვანილ მაგალითში ფუნქცია Angular-ის გარეთ არის გატანილი და საკუთარი სახელი აქვს მიცემული. ნათლად ჩანს, რომ ფუნქცია დამოუკიდებელია Angular-ისა და მისი სინტაქსისაგან):

```
function MainCtrl () {

}
angular
  .module('app', [])
  .controller('MainCtrl', MainCtrl);
```

ქვემოთ მოყვანილ მაგალითებში იგულისხმება, რომ Angular-ის მოდული უკვე შექმნილია.

5.1 მეთოდები და პრეზენტაციების ლოგიკა

პრეზენტაციების ფორმატში ბიზნესლოგიკის ინტერპრეტირებისათვის კონტროლერი მიმართავს **სერვისს** და $scope ობიექტის მეშვეობით ახდენს

სერვერიდან მონაცემების (პირდაპირი ან მოდიფიცირებული სახით) ადგილზე (View-ში) გადაცემას. განახლებული View კი უკუპროცესით იმავე სერვისის მეშვეობით ფიქსირდება სერვერზეც.

კონტროლერის ფუნქციონირებაში უკეთ გარკვევის მიზნით, ჯერ მასში ვქმნით რამდენიმე ობიექტს (ბიზნესლოგიკის საკითხს შემდგომ განვიხილავთ):

```
function MainCtrl ($scope) {

  $scope.items = [{
     name: 'მენიუ',
     id: 7297510
  },{
     name: 'კერძი_1',
     id: 0278916
  },{
     name: 'კერძი_2',
     id: 2389017
  },{
     name: 'კერძი_3',
     id: 1000983
  }];
}
angular
  .module('app')
  .controller('MainCtrl', MainCtrl);
```

$scope-ის მეშვეობით იქმნება მასივი, რომელიც შემდგომ შესაძლებელია Angular-ის ng-repeat დირექტივის მეშვეობით ციკლში ჩათვალიერდეს და, რაიმე შაბლონზე დაყრდნობით, ამა თუ იმ სტრუქტურის სახით, DOM-ში წარმოგვიდგეს:

```
<div ng-controller="MainCtrl">

  <ul>
      <li ng-repeat="item in items">
```

```
            {{ item.name }}
        </li>
    </ul>
</div>
```

5.2 ახალი, controllerAs სინტაქსი

კონტროლერები კლასებს წააგავს, მაგრამ არის განსხვავებაც - მათი გამოყენება ხდება არა this გასაღებური სიტყვის, არამედ $scope ობიექტების მეშვეობით, თუმცა Angular-ის დამპროექტებლებმა ასეთი მიდგომა დაუშვეს სპეციალურად შემოღებული controllerAs გადაწყვეტისათვის, რაც ამგვარად აისახა სინტაქსში:

ქვემოთ გამოტოვებულია $scope-ის გამოძახება და მის ნაცვლად გამოიყენება this პრეფიქსი:

```
function MainCtrl () {

  this.items = [{
    name: 'მენიუ',
    id: 7297510
  },{
    name: 'კერძი_1',
    id: 0278916
  },{
    name: 'კერძი-2',
    id: 2389017
  },{
    name: 'კერძი_3',
    id: 1000983
  }];
```

```
}
angular
  .module('app')
  .controller('MainCtrl', MainCtrl);
```

ამის შემდეგ DOM-ის იმ ადგილებში, სადაც საჭიროა კონტროლერის ეგზემპლარის შექმნა (მისი გამოყენება ამ წესით ხდება), ვამატებთ as ტერმინს.

ჩანაწერი MainCtrl as main მიგვითითებს, რომ ყველა მონაცემი ინახება main ცვლადში, რის გამოც წინა მაგალითში გამოყენებულ items ტერმინს ვცვლით main.items-ზე.

```
<div ng-controller="MainCtrl as main">

  <ul>
      <li ng-repeat="item in main.items">
          {{ item.name }}
      </li>
  </ul>
</div>
```

შედეგად, ხდება არა მარტო იმის გარკვევა, თუ რომელ კონკრეტულ კონტროლერს მიეკუთვნება ესა თუ ის თვისება, არამედ ასეთი მიდგომით თავიდან ვიცილებთ ერთნაირი სახელების მქონე თვისებებს შორის კონფლიქტური სიტუაციის წარმოშობის შესაძლებლობასაც:

მართლაც, controllerAs-ის გამოყენებით საჭირო აღარაა თითოეული $scope-ის მეთოდისათვის მშობელი $parent ობიექტის მითითება, ხოლო უფრო მაღალი დონისათვის - $parent.$parent-ის და ა.შ. მათ ნაცვლად გამოიყენება მხოლოდ ცვლადის სახელი.

6. სერვისები და ფაბრიკები

სერვისი მეთოდია, რომელიც ქმნის ახალ ობიექტს მასში მოდელის მონაცემებისა და ბიზნესლოგიკის შენახვის შესაძლებლობით.

6.1 service მეთოდი.

ვნახოთ, როგორ იქმნება სერვისი (ქვემოთ მოყვანილ მაგალითში სერვისის სახელია 'UserService') და მასთან კავშირდება შესაბამისი ფუნქცია:

```
function UserService () {
  this.sayHello = function (name) {
    return 'მოგესალმებით ' + name;
  };
}
angular
  .module('app')
  .service('UserService', UserService);
```

ამის შემდეგ შესაძლებელი ხდება, სერვისი კონტროლერში ჩაიდგას:

```
function MainCtrl (UserService) {
  this.sayHello = function (name) {
    UserService.sayHello(name);
  };
}
angular
  .module('app')
  .controller('MainCtrl', MainCtrl);
```

აღვნიშნავთ, რომ სერვისის წინ ვერ ხერხდება კოდის შესრულება, რადგანაც მეთოდები იქმნება ობიექტების სახით, ქვემოთ განხილული ფაბრიკებისაგან განსხვავებით.

6.2 ფაბრიკული მეთოდები

ფაბრიკული მეთოდები გვიბრუნებს ობიექტს ან ფუნქციას, მაშასადამე, ამ შემთხვევაში შესაძლებელი ხდება:

- გამოყენებული იქნას უკუჩართვის მექანიზმი;
- დაიშვება `host` ობიექტის დაბრუნება, რომელსაც მიებმება მეთოდები;
- აგრეთვე მიმართავენ პრივატული და საჯარო ხილვადობის უბნების შექმნასაც.

ფაბრიკებს ხშირად სერვისების სახელითაც მოიხსენიებენ. ქვემოთ ფაბრიკული მეთოდის გამოყენებით ვქმნით UserService-ს:

```
function UserService () {

  var UserService = {};
  function greeting (name) {
    return 'მოგესალმებით ' + name;
  }
  UserService.sayHello = function (name) {
    return greeting(name);
  };
  return UserService;
}
angular
  .module('app')
  .factory('UserService', UserService);
```

პირველი მიდგომის გამოყენებისას ახდენენ ხილვადობის პრივატული უბნის ემულირებას. საერთოდ კი, შესაძლებელი იყო იმავე მექანიზმის გამოყენებით service constructor მეთოდის შიგნით ასეთივე შედეგის მიღება, მაგრამ მაშინ ნათლად აღარ გამოჩნდებოდა, რა გვიბრუნდება და რა რჩება სერვისის ხილვადობის უბნის შიგნით. ასევე შესაძლებელია პრივატული დამხმარე ფუნქციების შექმნაც, რომლებიც ხილვადობის არეში რჩება ფუნქციის შედეგის დაბრუნების შემდეგაც და დასაშვებია მათი გამოძახება გარე მეთოდებიდან.

ამგვარივე წესით არის შესაძლებელი პრივატული დამხმარე ფუნქციების შექმნაც, რომლებიც დარჩება ხედვის არეში ფუნქციის შესრულების შემდეგაც. ამასთან, მათი გამოყენება შესაძლებელია საჯარო მეთოდებიდანაც. შევნიშნავთ, რომ კონტროლერის შიგნითაც დასაშვებია ამ სერვისების ზუსტად ასეთივე ხერხებით გამოყენება.

```
function MainCtrl (UserService) {

  this.sayHello = function (name) {
    UserService.sayHello(name);
  };
}
angular
```

```
  .module('app')
  .controller('MainCtrl', MainCtrl);
```

სერვისები, როგორც წესი, გამოიყენება არა პრეზენტაციების ლოგიკის, არამედ ბიზნესლოგიკის შრისათვის, რომელთანაც კავშირი მყარდება Ajax-ისა და REST მეთოდის (პროტოკოლის) მეშვეობით.

7. შაბლონების გამოყენება Angular-ის ბირთვისადმი მიმართვით

აქამდე განვიხილავდით Angular-ის პროგრამისტულ მხარეს, ამჯერად კი ვაჩვენებთ, თუ როგორ ხერხდება მისი გამოყენება HTML კოდიდან. აქცენტირება ხდება შაბლონების გამოყენების მძლავრ შესაძლებლობებზე.

7.1 გამოსახულებები

გამოსახულებები Angular-ში ორმაგ ფიგურულ ფრჩხილებში ექცევა - {{}}. ციკლები და პირობითი ოპერატორები მათში არ გამოიყენება, თუმცა დაშვებულია ლოგიკური (მაგალითად, || და &&) და ტერნალური ოპერატორის (value? true: false) ჩართვა.

```
function MainCtrl () {

  this.items = [{
    name: 'მენიუ',
    id: 7297510
  },{
    name: 'კერძი_1',
    id: 0278916
  },{
    name: 'კერძი_2',
    id: 2389017
  },{
    name: 'კერძი-3',
    id: 1000983
  }];
}
```

```
angular
  .module('app')
  .controller('MainCtrl', MainCtrl);
```

აი, როგორ შეიძლება მივიღოთ ინფორმაცია მასივის სიგრძის შესახებ:

```
<div ng-controller="MainCtrl as main">

  მასივის სიგრძეა {{ main.items.length }}

</div>
```

length თვისებას Angular შეცვლის მისი მნიშვნელობით, რომელიც ეკრანზეც აისახება.

7.2 ძირითადი დირექტივების გამოყენება

ზემოთ განხილული ქმედებების შედეგად მასივში შეტანილი ცვლილებები ავტომატურად აისახება DOM-ში და, ცხადია, მონიტორის ეკრანზეც. მაგალითად, მასივიდან ერთი ელემენტის ამოგდებისას დაიწერება: „მასივის სიგრძეა 3“.

საერთოდ, Angular იძლევა ng-* პრეფიქსის მქონე მრავალი დირექტივის გამოყენების საშუალებას, რითაც მომხმარებელი ქმნის უფრო მეტი შესაძლებლობის მქონე HTML ელემენტებს და ატრიბუტებს.

თავდაპირველად მოკლედ აღვწეროთ ზოგიერთი უმნიშვნელოვანესი დირექტივის დანიშნულება:

ng-app - აცხადებს დანართისათვის ფესვურ ელემენტს;
ng-bind - HTML ელემენტის ტექსტს ავტომატურად ანიჭებს გადაცემულ მნიშვნელობას;
ng-model - წინას ანალოგიურია, იმ განსხვავებით, რომ ელემენტსა და მოდელის მნიშვნელობებს შორის მყარდება ორმხრივი კავშირი-შესაბამისობა;
ng-class - განსაზღვრავს დინამიკური ჩატვირთვის კლასებს;
ng-controller - აფორმირებს JavaScript-კონტროლერს HTML-გამოსახულებების გამოსათვლელად;
ng-repeat - კოლექციის თითოეული ელემენტისათვის ქმნის ეგ ზემპლარს;

ng-show და ng-hide - ლოგიკურ გამოსახულებაზე დაყრდნობით ეკრანზე გამოჰყავს ელემენტი ან მალავს მას;
ng-switch - მოქმედებს დაპროგრამებაში კარგად ცნობილი switch გადამრთველის ანალოგიურად;
ng-view - ეს გაზისური დირექტივა კონტროლერებით მართვადი შაბლონების მეშვეობით ამუშავებს ისეთ მარშრუტებს, რომლებიც იმართება JSON-ზე დაყრდნობით.

დირექტივებთან უფრო საფუძვლიანი გაცნობა დავიწყოთ ng-click-დან:

```
<div ng-controller="MainCtrl as main">
  <div>
    {{ main.items.length }} ელემენტია თქვენს
განკარგულებაში.
  </div>
  <ul>
      <li ng-repeat="item in main.items" ng-
click="main.removeFromStock(item, $index)">
          {{ item.name }}
      </li>
  </ul>
</div>
```

ng-click დირექტივის მეშვეობით ვუკავშირდებით ფუნქცია main.removeFromStock()-ს, რომელსაც გადავცემთ ელემენტს ციკლში დასამუშავებლად. შესაბამისი თვისება გვეხმარება ელემენტის მასივიდან ამოგდებაში - საჭირო აღარაა „ხელით“ გამოვთვალოთ მიმდინარე ელემენტის ინდექსი.

ამის შემდეგ უკვე შესაძლებელია, ფუნქცია კონტროლერს დავუმატოთ. ამ ფუნქციას გადაეცემა $index და მასივის ელემენტი, მეთოდი კი ასე მოქმედებს:

```
function MainCtrl () {

  this.removeFromStock = function (item, index) {
    this.items.splice(index, 1);
  };
```

```
  this.items = [...];
}
angular
  .module('app')
  .controller('MainCtrl', MainCtrl);
```

მეთოდის შექმნისას გასათვალისწინებელია, რომ this-ის მნიშვნელობა კონტექსტის მიხედვით სხვადასხვა მნიშვნელობას იღებს. თუ გამოვიყენებთ ასეთ მიდგომას და კონტროლერზე შევქმნით დაყრდნობას:

`var vm = this;` (აქ `vm` აღნიშნავს `ViewModel`-ს), მაშინ ეს დაყრდნობა აღარ დაიკარგება.

გადამუშავებულ კონტროლერს ასეთი სახე ექნება:

```
function MainCtrl () {
  var vm = this;
  vm.removeFromStock = function (item, index) {
    vm.items.splice(index, 1);
  };
  vm.items = [...];
}
angular
  .module('app')
  .controller('MainCtrl', MainCtrl);
```

ამ წესით იქმნება ინტერფეისთან მომუშავე პრეზენტაციების ლოგიკა. გამოტოვებულია მხოლოდ ერთი ბიჯი - მოდელის განახლება.

vm.items-დან ელემენტის ამოგდების წინ უნდა გაიგზავნოს DELETE მოთხოვნა ბეკ-ენდზე. დადებითი პასუხის მიღების შემდეგ მოხდება ელემენტის ამოგდება მასივიდან. ამრიგად, DOM განახლდება მხოლოდ მოთხოვნის შესრულებისას და მომხმარებელს შეცდომა აღარ მოუვა.

8. დირექტივები

დირექტივები აადვილებს მომავალში პროგრამების დაწერას. ისინი ორი სახისაა:

1. ერთნი უკვე ჩაშენებულია Angular-ში და მათზე გავდივართ კავშირების დახმარებით;
2. მეორენი იქმნება მომხმარებელთა მიერ მათთვის სასურველი კონკრეტული შედეგების მისაღებად.

ჯერ გავეცნოთ არსებულთ და შემდეგ გადავიდეთ საკუთარი დირექ-ტივების შექმნის საკითხზე.

8.1 ng-repeat

ამ დირექტივას უკვე გავეცანით:

```
<ul>

  <li ng-repeat="item in main.items">
     {{ item }}
  </li>
</ul>
```

ng-repeat ახდენს ელემენტის კოპირებას და შემდეგ მის აღწარმოებას მასივიდან აღებული ობიექტების მონაცემებზე დაყრდნობით. თუ მასივიდან ელემენტს ამოვაგდებთ, DOM ავტომატურად განახლდება.

8.2 ng-model

გამოიყენება როგორც ახლად შესაქმნელი მოდელის ინიციალიზა-ციისათვის, ისე უკვე არსებულთან მის მისაბმელად.

შეტყობინება: {{ main.message }}

თუ $scope-ის main.message თვისებაში ინახება მნიშვნელობა, იგი გადაიცემა input-ში. საწინაღმდეგო შემთხვევაში უბრალოდ მოხდება მისი ინიციალიზება. შესაძლებელია ამ მნიშვნელობების სხვა, მაგალითად, ng-click დირექტივაში გადაცემა.

8.3 ng-click

onClick=”...” ხდომილობისაგან განსხვავებით, ng-click დირექტივა გლობალური სახის არ არის - იგი მოქმედებს მხოლოდ საკუთარი ხედვის არეში. ამასთან, ელემენტთან მისი მიბმა და დირექტივისათვის რაიმე პირობის

შესრულებისას შესაბამისი ხდომილობის დამმუშავებლის გამოძახება ავტომატურად ხდება.

```
<input type="text" ng-model="main.message">

<a href=" ng-click="main.showMessage(main.message);">
      აჩვენეთ შეტყობინება
</a>
```

ზემო მაგალითში main.message გადაიცემა main.showMessage მეთოდში, რომელშიც Angular მას დაამუშავებს, როგორც ჩვეულებრივ JavaScript ობიექტს.

აქ იკვეთება Angular-ის მნიშვნელოვანი ღირსება - DOM-ში არსებული მონაცემების ჯგუფები წარმოგვიდგება ადვილად მოსაძიებელი ობიექტების სახით, რომლებზეც შესაძლებელია ჩატარდეს საჭირო მოქმედებები, კერძოდ, გარდაქმნები Json ფორმატში და გადაგზავნები.

8.4 ng-href/ng-src

Angular ფრეიმვორკი ბროუზერებთან ურთიერთობისათვის href და src პარამეტრების ნაცვლად იყენებს ng-href და ng-src დირექტივებს:

```
<a ng-href="{{ main.someValue }}">Go</a>

<img ng-src="{{ main.anotherValue}}" alt="">
```

8.5 ng-class

elem.addClass(className) და elem.removeClass(className) ტრადიციული მიმართვების (გამოძახებების) ნაცვლად Angular ფრეიმვორკი, კლასების დამატებისა და ამოგდების მიზნით, იყენებს სწორედ ამ დირექტივას:

```
<div class="notification" ng-class="{

  warning: main.response == 'error',

  ok: main.response == 'success'
}">
  {{ main.responseMsg }}
```

```
</div>
```

იგი აანალიზებს main.response მნიშვნელობას და მის საფუძველზე შესაბამისად მოქმედებს.

8.6 ng-show/ng-hide

ამ დირექტივებით ხდება ელემენტის დამალვა-გამოჩენა, მისი რომელიმე თვისების მნიშვნელობიდან გამომდინარე.

გადართვა ხდება ng-click ხდომილობისადმი მიმართვით:

```
<a href=" ng-click="showMenu = !showMenu">მენიუს
გადართვა!</a>

<ul ng-show="showMenu">
  <li>1</li>
  <li>2</li>
  <li>3</li>
</ul>
```

მათ შორის განსხვავება ვლინდება მითითებაში, თავდაპირველად როგორ მდგომარეობაში უნდა იმყოფებოდეს ელემენტი.

8.7 ng-if

ng-if დირექტივა არათუ მალავს, არამედ ანადგურებს კიდეც ელემენტს. ოღონდ ელემენტის ამა თუ იმ თვისების მიერ შესაბამისი მნიშვნელობის მიღებისას შესაძლებელი ხდება ახალი $scope-ის გაზაზე ამ ელემენტის აღდგენაც.

შევნიშნავთ, რომ ng-if დირექტივის გამოყენების მეშვეობით ფრეიმვორკის სწრაფმოქმედება იზრდება.

```
<div ng-if="main.userExists">

  შეიტანეთ ლოგინი!
</div>
```

8.8 ng-switch

ng-switch დირექტივა იმგვარადვე მოქმედებს, როგორც პროგრამირებაში კარგად ცნობილი case / switch ოპერატორი. ამრიგად, შესამოწმებელი მნიშვნელობის მიხედვით, $scope-ში რამდენიმე ელემენტიდან ამოირჩევა ერთ-ერთი:

```
<div ng-switch on="main.user.access">

  <div ng-switch-when="admin">
    <!-- code for admins -->
  </div>
  <div ng-switch-when="user">
    <!-- code for users -->
  </div>
  <div ng-switch-when="author">
    <!-- code for authors -->
  </div>
</div>
```

8.9 ng-bind

როგორც ვნახეთ, DOM-ში მნიშვნელობების ჩასმა ხდება {{ value }} სინტაქსის მეშვეობით, თუმცა არსებობს სხვა გზაც - ng-bind დირექტივაც, რომლის დადებითი მხარეა ის, რომ ფურცლის ჩატვირთვისას შედეგის გამოთვლამდე კონტენტი არ ეკრანზე აისახება და ამის გამო ციმციმს ადგილი აღარ ექნება.

```
<p>{{ main.name }}</p>

<p ng-bind="main.name"></p>
```

8.10 ng-view

რათა ერთფურცლოვანი გამოყენებების განახლება ავტომატურად მოხდეს ფურცლის ხელახლა ჩატვირთვის გარეშე XMLHttpRequest მოთხოვნით

გადმოგზავნილი, დინამიკურად ჩასადგამი კოდის მისაღებად, ვიყენებთ ng-view ატრიბუტს ცარიელი ელემენტით.

ng-view-ის ადგილას ჩაიდგმება სხვადასხვა view (URL-ში მითითებული გზის მიხედვით), მაგალითად, Angular ჩადგამს login.html-ს, თუ URL-ის მნიშვნელობა იქნება myapp.com/#/login და ა.შ.

8.11 HTML-ის გაფართოება

როგორც ვნახეთ, ჩაშენებული დირექტივები მნიშვნელოვნად ამაღლებს HTML-ის ფუნქციონირების დონეს, თუმცა ზოგიერთი შემთხვევისათვის მათი შესაძლებლობები საკმარისი აღარაა და ენისადმი გაზრდილი მოთხოვნების დასაკმაყოფილებლად Angular-ის დამპროექტებლებმა კიდევ უფრო სრულყოფილი გახადეს ეს ფრეიმვორკი მასში მომხმარებელების მიერ საკუთარი დირექტივების შექმნის შესაძლებლობის დამატებით.

ქვემოთ განიხილება API ასეთი დირექტივების შესაქმნელად.

9. გაწყობადი დირექტივები

Angular-ისათვის განკუთვნილი გაწყობადი დირექტივები იყოფა სამ ჯგუფად:

- გაწყობადი ელემენტები,
- ჩრდილოვანი DOM,
- HTML-ის იმპორტი.

9.1 გაწყობადი ელემენტები

აღსანიშნავია, რომ გაწყობადი ელემენტების ჯგუფს მიაკუთვნებენ არა მარტო HTML ელემენტებს, არამედ ასევე - ატრიბუტებს, კლასებს და კომენტარებსაც.

პრაქტიკაში ყველაზე მეტად პოპულარულობა მოიპოვა გაწყობადმა ატრიბუტებმა მათი არაერთხელ და შაბლონის სახით გამოყენების შესაძლებლობის გამო. აქვე აღვნიშნავთ, რომ ისინი სხვადასხვა სახის ბროუზერებისათვის ყველაზე ნაკლებ პრობლემას ქმნის.

აი, გაწყობადი ელემენტებისადმი (საკუთრივ Element, ასევე, Attribute, Class და Comment) მიმართვის 4-ვე წესი:

```
<my-element></my-element>
```

```
<div my-element></div>
```

```
<div class="my-element"></div>
```

```
<!-- directive: my-element -->
```

ამ დირექტივების კუთვნილი restrict თვისებით შესაძლებელია მათი გამოყენების ვარიანტების შეზღუდვა. კერძოდ, restrict თვისებისთვის 'EA' მნიშვნელობის მინიჭებით გათვლა ხდება მხოლოდ ელემენტებსა და ატრიბუტებთან, 'C'-ით - კლასებსა, ხოლო 'M'-ით - კომენტარებთან მუშაობაზე.

9.2 ჩრდილოვანი DOM

ჩრდილოვანი DOM უზრუნველყოფს მასში იმპორტირებული როგორც წმინდა HTML კოდის, ასევე კონტენტის ასახვას გაწყობად ელემენტში. შედეგად, ჩვეულებრივ, DOM-ში შესაძლებელი ხდება ჩადგმული დოკუმენტების გადაგზავნაც.

აი, გაწყობად ელემენტში ტექსტის განთავსების მაგალითი:

```
<my-element>
  მოგესალმებით!
</my-element>
```

ამ ტექსტის გადმოწერა შესაძლებელი ხდება ჩრდილოვანი DOM-იდან.

9.3 HTML კოდის შაბლონიზება და იმპორტი

ქვემოთ განიხილება დირექტივებში შაბლონების გამოყენების 3 წესი:

9.3.1 template თვისება

template თვისებით გამოცხადებული სტრიქონის სახის მქონე შაბლონი კომპილირდება Angular-ის მიერ და DOM-ში ჩაისმება.

მაგალითი:

```
{
```

```
  template: '<div>' +
    '<ul>' +
      '<li ng-repeat="item in vm.items">' +
        '{{ item }}' +
      '</li>' +
    '</ul>' +
  '</div>'
}
```

ზედა ფრაგმენტს შესაძლებელია ასეთი სახეც მიეცეს:

```
  {

    template: [
      '<div>',
        '<ul>',
          '<li ng-repeat="item in vm.items">',
            '{{ item }}',
          '</li>',
        '</ul>',
      '</div>'
    ].join('')
  }
```

JavaScript-იდან ჩვენთვის ნაცნობი [].join('') კონსტრუქციის გამოყენება კოდს უფრო ადვილად წაკითხვადს ხდის.

9.3.2 templateUrl თვისება

templateUrl თვისება მიუთითებს იმ გარე რესურსსა თუ ელემენტზე, რომელიც საჭირო შაბლონს შეიცავს.

დავუშვა, მოცემულია:

```
{
  templateUrl: 'items.html'
}
```

Angular ჯერ ეცდება DOM-ში მოძებნოს შესაბამისი id-ის მქონე ელემენტი, წარუმატებლობისას კი ამ ფაილზე მოთხოვნას HTTP GET-თი სერვერზე გადააგზავნის.

```
<script type="text/ng-template" id="/hello.html">

  <div>
    <ul>
      <li ng-repeat="item in vm.items">
        {{ item }}
      </li>
    </ul>
  </div>
</script>
```

ზედა მაგალითში თავდაპირველად შაბლონს ტიპად განესაზღვრება text/ng-template მნიშვნელობა, რათა ბროუზერმა, JavaScript-ის მსგავსად, გზავნილი ტექსტად არ აღიქვას. ამ გზით შესაძლებელი ხდება დანიშნულების ადგილას რამდენიმე შაბლონის ერთი ფაილის სახით გადაგზავნა. ამასთანავე, დასაშვებია `template` თვისების გამოყენებაც, როდესაც შაბლონი სტრიქონში ინახება. Angular იმახსოვრებს ჩატვირთულ შაბლონს, რომლის გამოყენება შემდგომ შესაძლებელი ხდება ng-include და ng-view დირექტივებით. თუ შაბლონი ადგილზე ვერ მოიძებნა, Angular სერვერს მიმართავს GET-მოთხოვნით.

აღვნიშნავთ, რომ დანართი $templateCache ყველა ჩატვირთულ შაბლონს მეხსიერებაში მუდმივად ინახავს.

9.4 API დირექტივების შექმნა

განვიხილოთ საკუთარი დირექტივების შექმნის უმარტივესი მაგალითი მხოლოდ `return` ინსტრუქციის დახმარებით, რომელიც ობიექტს გვიბრუნებს:

```
function someDirective () {

  return {
  };
}
```

```
angular
  .module('app')
  .controller('someDirective', someDirective);
```

ქვემოთ ნაჩვენებია `someDirective` დირექტივისადმი მიმართვისას `return` ინსტრუქციით დაბრუნებული ობიექტის ყველაზე პოპულარული თვისებების გადმოცემის მაგალითი:

```
function someDirective () {
  return {
    restrict: 'EA',
    replace: true,
    scope: true,
    controllerAs: 'something',
    controller: function () {
    },
    link: function ($scope, $element, $attrs) {
    },
    template: [
      '<div class="some-directive">',
        'My directive!',
      '</div>'
    ].join('')
  };
}
angular
  .module('app')
  .controller('someDirective', someDirective);
```

9.4.1 restrict

როგორც უკვე ვნახეთ, `restrict` თვისებისათვის `A', 'E', 'M'` ან `'C'` მნიშვნელობების მიცემის შედეგად იზღუდება დირექტივის მოქმედების არეალი.

9.4.2 replace

`replace`-ის მეშვეობით დირექტივის თავდაპირველი ელემენტი შეიცვლება სასურველით - სკრიპტის მუშაობის შედეგით. მაგალითად, თუ ვიყენებთ `<some-directive></some-directive>` დირექტივას, `replace: true`-ს მეშვეობით ფურცლის შექმნის შემდეგ მოხდება თავდაპირველი ელემენტის შეცვლა.

9.4.3 scope

საშუალებას იძლევა, გამოყენებული იქნეს მოცემული დირექტივისათვის მიმდინარე ან მშობელი არის (კონტექსტის) `$scope`-ის მემკვიდრეობა. აგრეთვე შესაძლებელია იზოლირებული `$scope`-ის შექმნაც და გაწყობადი დირექტივებით მისთვის რაიმე მნიშვნელობების გადაცემა.

9.4.4 controllerAs

ჩვენ უკვე გავეცანით ამ თვისებას - იგი დირექტივის შიგნით განსაზღვრავს კონტროლერის სახელს. თუ ვთქვათ, ამ მიზნით ვწერთ *`controllerAs: 'something'`-ს,* მაშინ კონტროლერისადმი მიმართვებს ექნება ასეთი სახე:

```
something.myMethod();
```

9.4.5 controller

მისი მეშვეობით ხდება არსებული კონტროლერისადმი მიმართვა ან ახლის შექმნა.

თუ კონტროლერი `MainCtrl` სახელით უკვე არსებობს, მას განვსაზღვრავთ ასე:

```
controller: 'MainCtrl'.
```

ინკაფსულაციის შესანარჩუნებლად ახალ კონტროლერს ყოველ ჯერზე ამგვარად ვაცხადებთ:

```
controller: function () {}
```

კონტროლერის უკუგამოძახების ფუნქცია ამუშავებს ცვლილებებს ViewModel-ში და განაგებს სერვისებთან ურთიერთობას.

9.4.6 link

link ფუნქციისადმი მიმართვა შესაძლებელია ელემენტის კომპილირებისა და DOM-ში ჩადგმის შემდეგ. კონტროლერიდან DOM-ში ცვლილებების შეტანა არ ხდება, link ფუნქციაში კი დაშვებულია როგორც ეს ქმედება, ასევე, - $scope-ის, შაბლონის ფესვური $element ელემენტისა და $attrs ობიექტის ჩასმაც (ეს უკანასკნელი DOM-ის ელემენტის ყველა თვისებას შეიცავს).

link ფუნქციის შიგნით შესაძლებელია პლაგინების განსაზღვრა, ხდომილებათა დამმუშავებლების გამოძახება და Angular-ის სერვისების ჩადგმაც კი.

9.5 დირექტივების შექმნა

ქვემოთ მოგვყავს მაგალითი ისეთი დირექტივის შექმნისა, რომელიც იძლევა კოდში "email" კომპონენტის (შესაბამისი ველებიანად) ჩადგმის შესაძლებლობას.

ვქმნით <compose-email> ელემენტს, რომლის ადგილზე შემდგომ ჩაიდგმება შესაბამისი კონტენტი. DOM-ში ამ ელემენტის ჩადგმა დასაშვებია სხვადასხვა სასურველ ადგილას თითოეულში კომპონენტის ცალკეული ეგზემპლარის სახით.

ვიწყებთ შაბლონებით:

```
function composeEmail () {
  return {
    restrict: 'EA',
    replace: true,
    scope: true,
    controllerAs: 'compose',
    controller: function () {
    },
    link: function ($scope, $element, $attrs) {
```

```
    },
    template: [
      '<div class="compose-email">',
        '<input type="text" placeholder="To..." ng-
model="compose.to">',
        '<input type="text" placeholder="Subject..." ng-
model="compose.subject">',
        '<textarea placeholder="Message..." ng-
model="compose.message"></textarea>',
      '</div>'
    ].join('')
  };
}
angular
  .module('app')
  .controller('composeEmail', composeEmail);
```

ამრიგად, უკვე შესაძლებელი ხდება `composeEmail` დირექტივის რამდენიმე ადგილზე ჩასმა - საჭირო აღარაა HTML-ის კოდის ფრაგმენტის კოპირება - Angular-ის მიერ დირექტივის სახელის პარსირების შედეგად `<compose-email></compose-email>` ელემენტი გადის HTML-ის კოდის საჭირო ფრაგმენტზე - კომპონენტის ეგზემპლარზე.

10. ფილტრები

ფილტრების მეშვეობით ხდება მათთვის გადაცემული ინფორმაციის რაიმე ლოგიკის შესაბამისად გადამუშავება და მისთვის სასურველი სახის მიცემა, მაგალითად:

- თარიღის საჭირო ფორმატში გამოტანა;
- გვარების სიიდან იმ ელემენტების ამორჩევა, რომლებიც განსაზღვრული ასოდან იწყება და სხვ.

არჩევენ 2 სახის ფილტრს:

- HTML სინტაქსით: `{{ filter_expression | filter : expression : comparator }}`

- და JavaScript სინტაქსით: `$filter('filter')(array, expression, comparator);`

პრაქტიკაში ძირითადად იყენებენ პირველი სახის ფილტრს.

10.1 ფილტრები თარიღისათვის

Angular აადვილებს თარიღებთან მუშაობას ამ ამოცანისადმი შემდგომი მიდგომით (დრო აითვლება მილიწამებში):

```
$scope.timeNow = new Date().getTime():

<p>
  დღეს გვაქვს: {{ timeNow | date:'dd-MM-yyyy' }}
</p>
```

10.2 JSON ფილტრი

JSON-ის ჩადგმული ფილტრები JavaScript-ის ობიექტს გარდაქმნის JSON-სტრიქონად. მიმზიდველი სახით დასაფორმატებლად იყენებენ შემდეგ ტეგებს:

```
</code>
<source lang="html">
<pre>
{{ myObject | json }}
```

10.3 limitTo და orderBy

ზემოთ აღწერილი ფილტრების მუშაობა მარტივ წესებს ექვემდებარება: ფილტრი შესასვლელზე იღებს ერთ რაიმე მნიშვნელობას და მას სასურველ სახეს აძლევს.

ვნახოთ, როგორ მიმდინარეობს მონაცემთა კრებულებების ფილტრით დამუშავება. limitTo ლიმიტს აწესებს View-ში გადასაცემი მონაცემების რაოდენობაზე. მისი გამოყენება მოხერხებულია ng-repeat-ის შიგნით:

```
<ul>
  <li ng-repeat="user in users | limitTo:10">
```

```
    {{ user.name }}
  </li>
</ul>
```

ეს ფრაგმენტი ინფორმაციას გვაწვდის მხოლოდ 10 მომხმარებლის შესახებ. `orderBy` ახდენს გამოსაცემი მასივის ელემენტების სორტირებას ობიექტის რომელიმე თვისების მიხედვით.

```
{
  name: 'Todd Motto',
}
```

მოგვყავს ელემენტების ალფაბეტის მიხედვით სორტირებისა და ეკრანზე ასახვის მაგალითი:

```
<ul>
  <li ng-repeat=" user in users | orderBy:'name' ">
    {{ user.name }}
  </li>
</ul>
```

ქვემოთ გავეცნობით საკუთარი ფილტრების შექმნის მაგალითებს.

11. საკუთარი ფილტრები

საკუთარი ფილტრების შესაქმნელად `Angular`-ში იყენებენ შესაბამისი დანიშნულებს მქონე `API`-ს. მონაცემთა შორის ორმხრივი კავშირი მარტივად მყარდება. შემუშავებული ფილტრების გამოძახება ხდება `$digest` ციკლში.

11.1 ფილტრები ერთადერთი მნიშვნელობისათვის

დავიწყოთ მარტივი მაგალითით - ფილტრს გადაეცემა ერთადერთი მნიშვნელობა და `.filter()` მეთოდით მოგვეწოდება გაფილტრული კონტენტი. თითოეული ასეთი ფილტრი გლობალური სახისაა და იგი მიწვდომადია ხილვადობის ნებისმიერი უბნიდან.

ამ დანიშნულების ფილტრისათვის არსებობს შემდეგი სახის შაბლონი:

```
function myFilter () {
```

```
  return function () {
    // შედეგის დაბრუნება
  };
}
angular
  .module('app')
  .filter('myFilter', myFilter);
```

ფილტრებისათვის არგუმენტები ავტომატურად გადაიცემა. ქვემოთ ვქმნით ფილტრს ტექსტის ქვედა რეგისტრში გადასაყვანად (აქვე აღვნიშნავთ, რომ ასეთი დანიშნულების მქონე ჩაშენებული ფილტრი Angular-ში ისედაც არსებობს):

```
function toLowercase () {
  return function (item) {
    return item.toLowerCase();
  };
}
angular
  .module('app')
  .filter('toLowercase', toLowercase);
```

item პარამეტრი ფილტრს გადაეცემა, როგორც ლოკალური ცვლადი. შექმნილ ფილტრს ზუსტად ისევე მივმართავთ, როგორც - ჩაშენებულთ:

```
<p>{{ user.name | toLowercase }}</p>
```

11.2 მონაცემთა კრებულების დამმუშავებელი ფილტრები

შევქმნათ ფილტრი, რომელიც მოცემული კრებულიდან ირჩევს 'A' ასოთი დაწყებულ სიტყვებს:

```
function namesStartingWithA () {

  return function (items) {
    return items.filter(function (item) {
      return /$a/i.test(item.name);
```

```
    });
  };
}

angular
  .module('app')
  .filter('namesStartingWithA', namesStartingWithA);
```

გამოვიყენოთ ეს ფილტრი `ng-repeat` დირექტივაში:

```
<ul>
  <li ng-repeat="item in items | namesStartingWithA">
    {{ item }}
  </li>
</ul>
```

ფილტრში არგუმენტების გადაცემა შესაძლებელია ორწერტილის მეშვეობით:

```
<ul>
  <li ng-repeat="item in items | namesStartingWithA:something">
    {{ item }}
  </li>
</ul>
```

`something` ავტომატურად გადაიცემა ფილტრის ფუნქციაში:

```
function namesStartingWithA () {
  return function (items, something) {
    // შეღწევადია "items" და something-ზე
  };
}

angular
  .module('app')
  .filter('namesStartingWithA', namesStartingWithA);
```

11.3 კონტროლერების ფილტრები

ფილტრები შესაძლებელია შეიქმნას `.filter()` მეთოდის გამოყენების გარეშეც, კონტროლერში ფილტრის როლის მქონე ფუნქციის ჩართვით. ასეთ შემთხვევაში ფილტრი მიწვდომადია მხოლოდ ამ კონტროლერიდან ანუ იგი გლობალური სახის ველარ იქნება.

კონტროლერისათვის ვქმნით `this.namesStartingWithA` ფუნქციას, მიზნის მისაღწევად ვიყენებთ `controllerAs` სინტაქსს:

```
function SomeCtrl () {

  this.namesStartingWithA = function () {


  };
}

angular
  .module('app')
    .controller('SomeCtrl', SomeCtrl);
```

DOM-ში ფილტრის გამოძახების სინტაქსი რამდენადმე განსხვავებული სახისაა:

```
<ul>

  <li ng-repeat="item in vm.items |
filter:namesStartingWithA">
    {{ item }}
  </li>
</ul>
```

12. დინამიკური როუტინგის ორგანიზება $routeProvider-ზე დაყრდნობით

(ერთფურცლოვანი) გამოყენების შესაქმნელად `Angular` იყენებს `ngRoute` სახელწოდების როუტერს, რომლის მდგომარეობა `URL`-ის შემცველობაზეა დამოკიდებული. საჭირო მოდულის მიერთება ამგვარად ხდება:

```
angular

  .module('app', [
    'ngRoute'
```

```
  ]);
```

ამის შემდეგ გზების მისათითებლად ხდება `$routeProvider`-ის ჩადგმა და გაწყობა `.config()` მეთოდში და უკვე შესაძლებელია მივმართოთ `.when()` მეთოდს.

დავუშვათ, ჩვენს განკარგულებაშია ელ. ფოსტის წამკითხველი დანართი და მოითხოვება, `inbox`-ზე დაწკაპუნებისას გამოგვივიდეს წერილების სია. ასეთ შემთხვევაში პირველი არგუმენტი უნდა იყოს შესაბამისი `URL`-ის აღმწერი სტრიქონი, მეორე კი - დამატებითი გაწყობების შესაძლებლობის მქონე ობიექტი.

არარსებული გზების დასამუშავებლად აგრეთვე გათვალისწინებული არის `.otherwise()` მეთოდიც.

```
function router ($routeProvider) {

  $routeProvider
  .when('/inbox', {})
  .otherwise({
    redirectTo: '/inbox'
  });
}
angular
  .module('app')
    .config(router);
```

გზის გაწყობისათვის თავდაპირველად ვირჩევთ რაიმე შაბლონს, მაგალითად, `inbox.html`-ს:

```
$routeProvider
.when('/inbox', {
  templateUrl: 'views/inbox.html'
})
.otherwise({
  redirectTo: '/inbox'

   });
```

თითოეულ view-ს ესაჭიროება კონტროლერი. ამ შესაძლებლობის უზრუნველსაყოფად Angular-სათვის გათვალისწინებული არის თვისებები: controller და controllerAs.

```
$routeProvider
.when('/inbox', {
  templateUrl: 'views/inbox.html',
  controller: 'InboxCtrl',
  controllerAs: 'inbox'
})
.otherwise({
  redirectTo: '/inbox'
});
```

მიზნად დავისახოთ კიდევ ერთი view-ის გაწყობა - დავუშვათ, გვსურს, შემომავალ წერილზე დაწკაპუნებისას იგი წავიკითხოთ. ამ შემთხვევაში აუცილებელი ხდება დინამიკური როუტინგის გამოყენება, რადგანაც სხვადასხვა წერილს განსხვავებული URL აქვს. ამგვარი როუტინგის მონაცემების გადაცემისას დინამიკური ჯგუფის სახელის წინ ისმება ორწერტილი. მაგალითად, id 173921938 იდენტიფიკატორიანი წერილი-სათვის გზა ასეთი სახით იქნება ნაჩვენები:

/inbox/email/173921938,

ხოლო გზის აღწერა კი შემდეგნაირად მოხდება:

'/inbox/email/: id'.

როდესაც დანართს გზის ასარჩევად გადაეცემა ესა თუ ის მნიშვნელობა (მოცემულ შემთხვევაში /inbox/email/173921938), Angular ჩამო-ტვირთავს იმავე შაბლონსა და კონტროლერს, რომლებიც გამოიყენებოდა /inbox/email/902827312-ის შემთხვევაში.

საბოლოოდ გვექნება:

```
function router ($routeProvider) {
  $routeProvider
  .when('/inbox', {
    templateUrl: 'views/inbox.html',
    controller: 'InboxCtrl',
    controllerAs: 'inbox'
  })
   .when('/inbox/email/:id', {
    templateUrl: 'views/email.html',
    controller: 'EmailCtrl',
    controllerAs: 'email'
  })
  .otherwise({
    redirectTo: '/inbox'
  });
});
angular
  .module('app')
  .config(router);
```

ამის შემდეგ უნდა მიეთითოს, რომელ ადგილას უნდა ჩაიდგას ფურცელზე დამუშავებული შაბლონი, რისთვისაც გათვალისწინებულია `ng-view` დირექტივა:

```
<div ng-view></div>
```

`Angular` ფრეიმვორკი თვალყურს ადევნებს `URL`-ში განხორციელებულ ყველა ცვლილებას, არკვევს ფურცელზე სხვა შაბლონის დაყენების საჭიროებას და ახორციელებს შესაბამის ქმედებებს.

12.1 $routeParams

`$routeParams` სერვისი ავტომატურად ახდენს `URL` მისამართის პარსირებას და მისგან გამოყოფს პარამეტრების კრებულს, რომელსაც ობიექტად გარდაქმნის. წინა მაგალითში, დინამიკური როუტინგის `:id` სახით განსაზღვრისას, `URL`-იდან ამოირჩეოდა გადმოგზავნილი წერილის `id` იდენტიფიკატორი.

მოვიყვანოთ $routeParams-დან პარამეტრების გადამცემი კონტროლერის მაგალითი:

```
function EmailCtrl ($routeParams, EmailService) {

  // $routeParams { id: 20999851 }
  EmailService
  .get($routeParams.id) // გადაიცეს ობიექტი
  .success(function (response) {})
  .error(function (reason) {});
}

angular
  .module('app')
  .('EmailCtrl', EmailCtrl);
```

13. ფორმების შემოწმება

პირველ რიგში აუცილებელია ფორმას მიეცეს სახელი - იგი ფორმისათვის განსაზღვრავს ხილვადობის არეს:

```
<form name="myForm"></form>
```

Angular აფიქსირებს ფორმას და თვალყურს ადევნებს, დაიცვა თუ არა მომხმარებელმა მისი შევსებისას ყველა საჭირო წესი.

13.1 HTML5

HTML5 ენაში დამატებული იქნა pattern ატრიბუტი. იგი ბროუზერს შესაძლებლობას აძლევს, შეამოწმოს კოდის შესაბამისობა ამ ენის მოთხოვნებთან. ასეთი კონტროლის შესაძლებლობა Angular-შიც არის გათვალისწინებული, დამატებით კი ეს ფრეიმვორკი ng-required დირექტივაში required-ით მონიშნული ველებისათვის განუწყვეტლივ ამოწმებს მოდელის მდგომარეობას.

ქვემოთ გავეცნოთ Angular-ის ზოგიერთ სხვა შესაძლებლობასაც.

13.2 $pristine

ფურცლის თავდაპირველად შექმნისას `Angular` ფორმისათვის ამატებს `$pristine` თვისებას და `ng-pristine` კლასში შეჰყავს ის ელემენტები, რომელთაც ჯერ ცვლილება არ განუცდია.

13.4 $dirty

`$dirty` თვისებით პირიქით - იდენტიფიცირდება შეცვლილი ელემენტები. ამასთან, მათ ემატება `ng-dirty` კლასებიც, ხოლო `ng-pristine` კლასები კი ნადგურდება. ფორმა ფურცლის გადატვირთვის გარეშე `$pristine` მდგომარეობას ვერ დაუბრუნდება.

13.5 $valid

`$valid` თვისების მეშვეობით შეტანის თითოეული ველი შესაძლებელია გამოცხადდეს, როგორც ვალიდური. მაგალითად, თუ ველს აქვს `ng-required` ატრიბუტი და მომხმარებელმა ის შეავსო, ველს ენიჭება `ng-valid` კლასი.

13.6 $invalid

წინას ანტიპოდია. დუმილით, ითვლება, რომ ყველა ფორმა `$invalid` მდგომარეობაში იმყოფება და მათ მინიჭებული აქვს `ng-invalid` კლასი. განხილულ 2 მდგომარეობას შორის გადართვები ხდება მომხმარებლის მიერ ინფორმაციის შეტანისას.

13.7 შეტანის ელემენტების ჩართვა-გამორთვა

ზოგჯერ მოითხოვება ინფორმაციის შეტანის ელემენტების (ტექსტური ველის, ღილაკის) ჩართვა-გამორთვა. მაგალითად, ქვემო ფრაგმენტში შეტანის ღილაკი ჩაირთვება მხოლოდ მაშინ, როდესაც მომხმარებელი შეტანის ველში რაიმე სიმბოლოს აკრეფს და თუ ეს არ მოხდა, ფორმა ვერ განახლდება:

```
<input type="text" ng-model="user.name"
placeholder="შეიტანეთ სახელი!">
<button ng-disabled="!user.name.length">
```

```
  განვაახლოთ სახელი!
</button>
```

ღილაკი ჩაირთვება მაშინ, როდესაც user.name.length მნიშვნელობა true გახდება. სანამ ფორმასთან მუშაობა მიმდინარეობს, ეს მნიშვნელობა განუწყვეტლივ მოწმდება.

14. სერვერთან ინფორმაციის გაცვლა $http და $resource დანართების (API) მეშვეობით

სერვერთან მაღალი დონის `$http` და `$resource` კავშირების საშუალებებითაც შესაძლებელია `$digest` ციკლების გაშვება და ჩვენი მონაცემების აქტუალიზება.

14.1 $http

სანამ **$http** მეთოდის დანიშნულებას გავეცნობოდეთ, სასურველია გავიხსენოთ, თუ როგორ მოქმედებდა `jQuery`-ში გათვალისწინებული `$.ajax` მეთოდი.

`$http` მეთოდი შესაძლებელია გამოყენებული იქნეს, როგორც ფუნქცია და ასევე, როგორც ობიექტი.

ქვემოთ მოგვყავს `GET` მარტივი `HTTP`-მოთხოვნის ფორმირების მაგალითი:

```
$http.get('/url')
.success(function (data, status, headers, config) {})
.error(function (data, status, headers, config) {});
```

შესაძლებელია კიდევ უფრო მარტივი მიდგომის გამოყენებაც:

```
$http.get('/url')
.success(function (response) {})
.error(function (reason) {});
```

არსებობს ასეთი გზაც:

```
 $http.get('/url')
```

```
.then(function (response) {
  // პასუხი წარმატებისას
}, function (reason) {
  // ინფორმაცია წარუმატებლობისას
});
```

jQuery ბიბლიოთეკისაგან განსხვავებით, Angular-ი *$http* გამოძახებებს განათავსებს $scope.$apply()-ში, რის შედეგად სერვერზე არსებული მონაცემებისათვის გაიშვება მოთხოვნების ციკლი და ხდება კავშირების განახლება.

14.2 $resource

$http მეთოდთან ერთად შესაძლებელი არის მოდულით სარგებლობაც, რომელშიც არსებული API $resource აადვილებს CRUD (create, read, update, delete) ოპერაციების ჩატარებას.

ეს რესურსი ქმნის ობიექტს, რომლის მეშვეობითაც მონაცემთა წყაროებთან ურთიერთობა მყარდება REST პროტოკოლებზე დაყრდნობით.

```
function MovieService ($resource) {

  return $resource('/api/movies/:id', { id: '@_id' },
    {
      update: {
        method: 'PUT'
      }
    }
  );
}
angular
  .module('app')
  .factory('MovieService', MovieService);
```

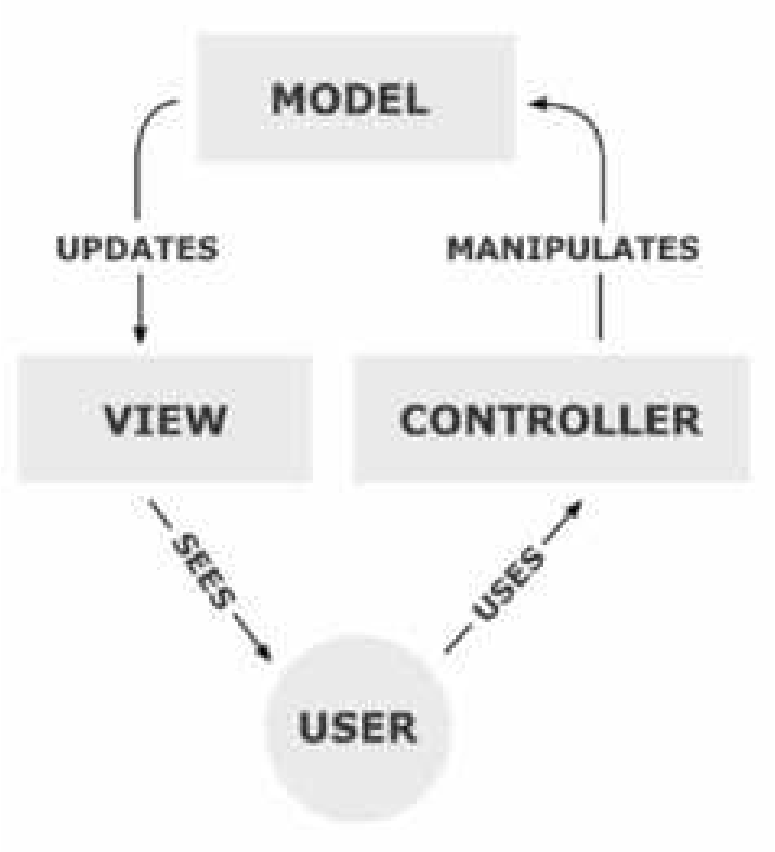
შესაძლებელია, დამოკიდებულება ჩაისვას მარტივ Movies ფაბრიკაში ჩვენ მიერ კონტროლერებში ფორმირებულ და შედეგად წვდომა გვქონდეს ყველა საჭირო მონაცემზე.

```
function MovieCtrl (MovieService) {

  var movies = new MovieService();

  // მოხდა განახლება

  movies.update(/* some data */);

}

angular

  .module('app')

  .controller('MovieCtrl', MovieCtrl);
```

დანართი_1

MVC (Model-View-Controller) მოდელის

სქემა



დანართი_2

მასალები ლაბორატორიული სამუშაოებისათვის

(I ნაწილი - ნიმუშები)

Angular-თან პირველი შეხება

შესავალი

```
<!DOCTYPE html>
<html>
  <!- ქვედა სტრიქონში ხდება ფრეიმვორკის ჩატვირთვა! ->
```

```
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
 <h1 align="center">AngularJS </h1>
 <p></p>

<table align="center">
  <tr><td><h4>     AngularJS აფართოებს HTML ენის შესაძლებობებს მისი ელემენტებისათვის ახალი ატრიბუტების შემოტანის გზით.</h4></td></tr>
  <tr><td><h4>     AngularJS იდეალური და ამასთან, ადვილად ასათვისებელი ინსტრუმენტი გახლავთ ე.წ. ერთგვერდიანი (SPA) დანართების <br/> (გამოყენებების) შესაქმნელად.</h4></td></tr>
  <tr><td><h4>     ამ სახელმძღვანელოთი თავდაპირველად შევისწავლით AngularJS-ის საფუძვლებს <br/> (დირექტივებს, გამოსახულებებს, ფილტრებს, მოდულებს და კონტროლერებს), <br/>
შემდეგ კი გადავალთ ხდომილობების, მათი დამმუშავებლების, DOM ფორმების, მოთხოვნების <br/> და სხვა საკითხების განხილვა-შესწავლაზე.</h4></td></tr>
  <tr><td><h4>     სანამ Angular-ის შესწავლას შევუდგებოდეთ, მოითხოვება ვერკვეოდეთ HTML, CSS და JavaScript-ის შესაძლებლობებში.</h4></td></tr>
 </table>

<h4 align="center" style="color: blue">ქვემოთ მოგვყავს AngularJS-ის მეშვეობით (SPA) დანართის შექმნა-გამოყენების მაგალითი</h4>
  <!- ng-app, ng-model ატრიბუტებთან (დირექტივებთან) და გამოსახულებებთან გაცნობა ->
<p></p>

<div align="center" ng-app="">    <!- აქ ხდება გამოყენებისათვის ფესვური ელემენტის განსაზღვრა! ->
    <p>Input something in the input box:</p>
    <!- ქვემოთ ng-model დირექტივით იქმნება name სახელის ცვლადის მოდელი! ->

    <p>Name: <input type="text" ng-model="name" placeholder="Enter name here"></p>
    <!- შეგვაქვს ტექსტი, რომელიც მომენტალურად, ასო-ასო გამოდის ეკრანზე ქვემოთ h1 ელემენტში! ->

<h1>Hello {{name}}</h1>   <!- ფრჩხილებში თავსდება გამოსახულება, რომელიც გამოითვლება და შესაბამისი ტეგით ეკრანზე აისახება ->
  </div>
</body>
</html>
```

იგივე ამოცანა პლუს გამოსვლა

```
<!DOCTYPE html>
<html>
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
  <body>
    <div ng-app="">
```

```
    <p>აკრიფეთ რაიმე ტექსტი (სახელი) შეტანის ველში:</p>
    <p>სახელი: <input type="text" ng-model="name" placeholder="Enter name here"></p>
    <h1>Hello {{name}}</h1>
   </div>
  <center> <A HREF="#" onClick="window.close()"> შ ე ვ წ ყ ვ ი ტ ო თ!</a> </center>
 </body>
</html>
```

კომენტარების გამოტანა-წაშლა

```
<!DOCTYPE html>
<html>
 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
 <body>
  <center> <A HREF="#"  onClick =" document.all.item('GEO').innerHTML ='კომენტარი ქართულ ენაზე';  "> კომენტარის გამოყვანა </a> </center>
  <center> <A HREF="#"  onClick =" document.all.item('ENG').innerHTML ='Comment in English'; "> COMMENT GET </a> </center>

  <center> <A HREF="#"  onClick =" document.all.item('GEO').innerHTML ='';  "> კომენტარის წაშლა  </a> </center>
  <center> <A HREF="#"  onClick =" document.all.item('ENG').innerHTML ='';  "> COMMENTs DELETE</a> </center>
 <center>
  <div ng-app="">
    <p>შეტანის ველში აკრიფეთ სახელი:</p>
    <p>სახელი: <input type="text" ng-model="name" placeholder="Enter name here"></p>
    <h1>Hello {{name}}</h1>
   </div>
 </center>
<p  id="GEO"></p>
<p  id="ENG"></p>
</body> </html>
```

შევისწავლოთ Angular-ის ზოგიერთი ატრიბუტის (დირექტივის) დანიშნულება

ng-bind ატრიბუტი

```
<!DOCTYPE html>

<html>

  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

 <!- ng-bind ატრიბუტთან (დირექტივასთან) გაცნობა ->
```

```
 <div ng-app="">

   <p>შეტანის ველში აკრიფეთ რაიმე ტექსტი (მაგალითად, სახელი): </p>

   <!- ქვემოთ ng-model დირექტივით იქმნება name სახელის მქონე ცვლადის მოდელი! ->

   <p>სახელი: <input type="text" ng-model="name" placeholder="Enter name here"></p>

   <!- შემაქვს ტექსტი, რომელიც მომენტალურად, ასო-ასო გამოდის ეკრანზე ქვემოთ h3 და p ელემენტებში! ->


   <h3 align="center">შეტანის ველში თქვენ აკრიფეთ <span ng-bind="name"></h3>

   <!- ng-bind დირექტივით გავდივართ name ცვლადზე (მის მოდელზე) ->

   <p>შეტანის ველში თქვენ აკრიფეთ <strong><span ng-bind="name"></strong></p>

   <!-  ng-bind დირექტივით გავდივართ name ცვლადზე (მის მოდელზე)  ->

 </div>

</body>

</html>
```

ng-init და ng-bind ატრიბუტები

```
<!DOCTYPE html>

<html>

 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

 <!-  ng-init ატრიბუტთან გაცნობა და ng-bind-ით მის მნიშვნელობაზე გასვლა  ->


 <div ng-app=""  ng-init="firstName='John'"> <!- ფაქტობრივად, ეს არის ცვლადის გამოცხადება და ხდება მისი ინიციალიზება. ->

 <!-  საერთოდ, ცვლადების ინიციალიზებას ამჯობინებენ კონტროლერების ან მოდულის მეშვეობით (ამ მიდგომას შემდგომ გავეცნობით).  ->


   <p>მოცემულ დანართში გამოცხადებულია ცვლადი შემდეგი საწყისი მნიშვნელობით - <strong><span ng-bind="firstName"></span>.</strong></p>

 </div>
```

```
</body>

</html>
```

ვატარებთ ექსპერიმენტს

```
<!DOCTYPE html>
<html>
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>

 <div>   <!- ამჯერად Angular-ის ატრიბუტი აღარ გამოგვიყენებია. ->
   <p>My first expression: {{ 5 + 5 }}</p>    <!- გამოსახულება აღარ გამოითვლება და იგი ეკრანზე აისახება ჩვეულებრივი ტექსტის სახით! ->
 </div>

</body>
</html>
```

ლიტერალები, ცვლადები, ოპერატორები,

გამოსახულებები

გამოსახულების დამუშავების მაგალითები

```
<!DOCTYPE html>
<html>
   <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
  <body>
   <div ng-app="">
<!-  ქვემოთ ხდება ორმაგ ფიგურულ ფრჩხილებში მოთავსებული გამოსახულების დამუშავება-გამოთვლა  ->
     <p>My first expression: <strong> {{ 5 + 5 }} </strong> </p>
   </div>

 </body>
</html>

<!DOCTYPE html>
<html>
```

```
 <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>

 <div ng-app>
<!-  ვატარებთ ექსპერიმენტს - ng-app ატრიბუტის მნიშვნელობას არ ვუთითებთ.  ->
  <p>My first expression: <strong> {{(5 + 5)*2 }} </strong> </p>
<!-  ამ შემთხვევაშიც ხდება გამოსახულების გამოთვლა!  ->

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

html ელემენტისათვის დინამიკურად

ფერის შეცვლა

```
<!DOCTYPE html>
<html>
   <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>

  <p align="center"> ქვემოთ ველში შეცვალეთ ტექსტი - შეიტანეთ რომელიმე ფერის დასახელება ი ნ გ ლ ი ს უ რ ენაზე!</p>

  <div  ng-app="" ng-init="myCol='lightblue'">

    <input style="background-color:{{myCol}}" ng-model="myCol" value="{{myCol}}">
     <! - გაანალიზეთ ამ ელემენტის შემცველობა. რა ხდება?! ->

  </div>

  <p>AngularJS resolves the expression and returns the result.</p>

  <p>The background color of the input box will be whatever you write in the input field.</p>

        <! - დ ა ვ ა ლ ე ბ ა: გადააკეთეთ ეს კოდი HTML-ის სხვადასხვა ელემენტისათვის ფერის შესაცვლელად! ->

</body>
</html>
```

ცვლადების შემცველი გამოსახულების

გამოთვლა

```
<!DOCTYPE html>

<html>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
```

```
<body>

 <div ng-app="" ng-init="quantity=1;cost=5">

  <p>Total in dollar: <strong>{{ quantity * cost }}</strong></p>  <! ცვლადების შემცველი გამოსახულების გამოთვლა ->

 </div>

      <! - დ ა ვ ა ლ ე ბ ა: დაწერეთ კოდები უფრო რთული სახის გამოსახულებების გამოსათვლელად! ->

</body>

</html>
```

მონიტორზე გამოსატანი ტექსტის აწყობა

ცვლადებზე დაყრდნობით

```
<!DOCTYPE html>

<html>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

 <body>


  <p align="center"> ტექსტის აწყობა ცვლადებზე დაყრდნობით </p>


  <div ng-app=""  ng-init="firstName='John';  lastName='Doe'">


    <p>The full name is: <strong> {{ firstName + " " + lastName }} </strong></p>


    <! -  დ ა ვ ა ლ ე ბ ა: დაწერეთ კოდები სხვადასხვა  ცვლადების გამოყენებით ტექსტების ასაწყობად და ეკრანზე ასახვისათვის! ->

  </div>


 </body>

</html>
```

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>

  <div ng-app="" ng-init="firstName='John'; lastName='Doe'">

   <p>The full name is: <span ng-bind="firstName + ' ' + lastName"></span></p>

  </div>

  <! -  დ ა ვ ა ლ ე ბ ა:  დაწერეთ კოდები მსგავსი მოქმედებების განსახორციელებლად! ->

</body>
</html>
```

Angular-ის ობიექტები მსგავსია

Javascript-ის ობიექტების

```
<!DOCTYPE html>
<html>
```

```
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>

  <!- გავეცნოთ Angularis-ის მეშვეობით ობიექტის გამოცხადებისა და მისი ეგზემპლარის თვისებებისათვის მნიშვნელობის განსაზღვრის ხერხებს! ->
  <!- მივაქციოთ ყურადღება - ყოველივე ეს ხდება Javascript-ში მიღებული წესების მსგავსად! ->

 <div ng-app="" ng-init="person={firstName:'John', lastName:'Doe'}">
  <p>The name is {{ person.lastName }}</p>
 </div>

 <!- დ ა ვ ა ლ ე ბ ა:  დაწერეთ კოდები სხვადასხვა ობიექტების შესაქმნელად, დაუნიშნეთ მათ თვისებები,  განუსაზღვრეთ ამ თვისებებს მნიშვნელობები და ასახეთ ისინი ეკრანზე!
 ->
</body>
</html>

<!DOCTYPE html>
<html>
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>

 <div ng-app="" ng-init="person={firstName:'John',lastName:'Doe'}">

<p>The name is <strong><span ng-bind="person.lastName"></strong></span></p> <!- აქ ng-bind ატრიბუტით გავდივართ person ობიექტის ეგზემპლარის lastName თვისების მნიშვნელობაზე ->
 </div>
<! დ ა ვ ა ლ ე ბ ა:  დაწერეთ კოდები სხვადასხვა ობიექტის შესაქმნელად, დაუნიშნეთ მათ თვისებები და განუსაზღვრეთ ამ თვისებებს მნიშვნელობები, ჩართეთ  ისინი აბზაცში span ელემენტის მეშვეობით და ასახეთ ეკრანზე!  ->
</body>
</html>
```

```
<!DOCTYPE html>

<html>

  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>


 <div ng-app="" ng-init="points=[1,15,19,2,40]">  <!- ობიექტის როლში გვევლინება მასივი ->


   <p>The third result is {{ points[4] }}</p>      <!- მივაქციოთ ყურადღება - მასივის პირველი ელემენტის ნომერი არის ნ უ ლ ი ! ->


 </div>


 <!- დ ა ვ ა ლ ე ბ ა:  დაწერეთ კოდები სხვადასხვა მასივის შესაქმნელად და მისი ელემენტების ნაწილი ასახეთ ეკრანზე! ->


</body>

</html>



<!DOCTYPE html>

<html>

  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>


 <div ng-app="" ng-init="points=[1,15,19,2,40]">  <!- ობიექტის როლში გვევლინება მასივი ->


  <p>The third result is {{ points[5] }}</p>

  <! მივაქციოთ ყურადღება - მასივის პირველი ელემენტის ნომერი არის ნ უ ლ ი ! ამის გამო მასივის მეხუთე ელემენტი, ფაქტობრივად კი მეექვსე, ვერ მოიძებნა! ->


 </div>
```

```
 <! დ ა ვ ა ლ ე ბ ა:  დაწერეთ კოდები მსგავსი მაგალითებისათვის და შედეგები ასახეთ ეკრანზე! ->


</body>

</html>


<!DOCTYPE html>

<html>

  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>


 <div ng-app="" ng-init="points=[1,15,19,2,40]">


  <p>The third result is <span ng-bind="points[4]"></span></p>  <!- ობიექტთან დაკავშირება ხდება Angular-ის შესაბამისი, ng-bind დირექტივის მეშვეობით. ->


 </div>


</body>

</html>
```

შესაძენი პროდუქტების სიის ფორმირებისათვის

Angular-ის დახმარებით დანართის შექმნის

მაგალითი

```
<!doctype html>

<html ng-app="purchaseApp">  <!-  html  ელემენტი ყველაზე მაღალი დონეა, რომელზეც
    შესაძლებელია დანართის მიბმა  ->

<head>

  <meta charset="utf-8">
```

```
 <link rel="stylesheet"
   href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css/bootstrap.min.css">
</head>
<body ng-controller="purchaseController">   <!- აქ ცხადდება კონტროლერი ->
  <div class="page-header">
    <h1> შენაძენთა სია </h1>
  </div>
  <div class="panel">                    <!- begin “panel” ->
    <div class="form-inline">                  <!-  begin “form-inline”  ->
      <div class="form-group">
        <div class="col-md-8">
          <input class="form-control" ng-model="text" placeholder = "დასახელება" />
        </div>
      </div>
      <div class="form-group">
       <div class="col-md-6">
        <input type="number" class="form-control" ng-model="price" placeholder="ფასი" />
       </div>
      </div>
      <div class="form-group">
        <div class="col-md-offset-2 col-md-8">
          <button class="btn btn-default" ng-click="addItem(text, price)">
            დამატება
          </button>
        </div>
      </div>
    </div>                                  <!- end “form-inline”  ->
    <table class="table table-striped">
```

```
        <thead>
          <tr>
            <th>საგანი</th>
            <th>ფასი</th>
            <th>შეძენილია</th>
          </tr>
        </thead>
        <tbody>
          <tr ng-repeat="item in list.items">
            <td>{{item.purchase}}</td>
            <td>{{item.price}}</td>
            <td><input type="checkbox" ng-model="item.done" /></td>
          </tr>
        </tbody>
      </table>
    </div>                                  <!-  end “panel”   ->

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular.min.js"></script>
<script>
var model = {
    items: [
      { purchase: "პური", done: false, price: 15.9 },
      { purchase: "ზეთი", done: false, price: 60 },
      { purchase: "კარტოფილი", done: true, price: 22.6 },
      { purchase: "ყველი", done: false, price:310 }
    ]
};
```

```
var purchaseApp = angular.module("purchaseApp", []);

   purchaseApp.controller("purchaseController", function ($scope) {

   $scope.list = model;

   $scope.addItem = function (text, price) {

      price = parseFloat(price);   // რიცხვად გარდაქმნა

      if(text != "" && !isNaN(price))   // თუ შეტანილია რაიმე ტექსტი  და იგი რიცხვია, ვამატებთ:

      {

         $scope.list.items.push({ purchase: text, price: price, done: false });

      }

   }

});

</script>

</body>

</html>
```

ლიტერატურა

1. WEB-ტექნოლოგიების სტანდარტების საიტი http://www.w3schools.com
2. http://metanit.com/web/angular/
3. გ. ღვინეფაძე. Javascript და მისი შესაძლებლობების განმავითარებელი თანამედროვე ტექნოლოგიები. *"ტექნიკური უნივერსიტეტი".* 2016 წ. ISBN 99940-14-80-3.
4. გ. ღვინეფაძე. WEB-დაპროგრამება WEB 2.0, XML, AJAX. *"ტექნიკური უნივერსიტეტი".* 2013 წ.
5. http://www.intuit.ru/
6. http://learn.javascript.ru/json

7. http://thewebland.net/angularjs-tutorial/

redaqtori b. cxadaZe

gadaeca warmoebas 24.04.2017. xelmowerilia dasabeWdad 26.04.2017. qaRaldis zoma 60X84 1/16. pirobiTi nabeWdi Tabaxi 3.

sagamomcemlo saxli `teqnikuri universiteti~, Tbilisi, kostavas 77

i.m. saxli `goCa dalaqiSvili~, q. Tbilisi, varkeTili 3, korp. 333, bina 38