**Introduction**

Jedan od popularnih i moćnih frameworka današnjice u webu je Angular.js koji je razvio google i izuzetno ga je teško naučiti ili nije. Možda naučiti ustvari znači stvarno ga razumeti.

Ja sam vaš instruktor Zoran Vulanovic, petnaestogodisnji veteran u razvoju web aplikacija.

Pored mysql i php u doba dok je bila još 4 verzija u upotrebi, razvila se potreba za učenjem JavaScripta kao jezika zbog potreba na frontu. Ne samo da je bio izazov učiti JavaScript u to vreme, nego ga je stvarno trebalo razumeti, masterizovati da bi stvarno mogli samostalno napravili SPA aplikaciju. Što sam na kraju i postigao. Napravio sam aplikaciju jos davne 2009 koja je objedinjavala logiku otkupa sirovina sa materijalnim, finansijskim knjigovodstvom i koja radi kao SPA za potrebe firme “Business Park” iz Bačkog Petrovca. Razvijanje takve aplikacije na frontu je bila noćna mora, razvoj je bio spor uključivalo se puno bibloteka desavalo se da jedan javascript fajl završi sa par hiljada linija koda doduše čitljivo i razumljivo ali sporo i lošje organizovano nego danas . Kompletnu logiku sam morao da napisem sam, sve u jezgru js-u i uz pomoć Jquery-a koji mi je pomagao oko DOM manipulacije.

Ono želim da izrazim jeste da Ideja o Single Page Application dalje u textu SPA aplikacijama ne potiče od samih izumitelja danas poznatih frontend frameworka. Potreba za SPA aplikacijama javila se mnogo ranije samo nije bilo tehnologije koja bi to podržavala.

Pojavljvanjem prvih SPA tehnologija javascript programerima se život promenio na bolje. Medju njima želim da pomenem trenutno najbolje. Angular.JS, Vue.js, Ember.js, Meteor.js i React koji je više biblioteka nego full baked framework.

Ja mislim da je Angular.js stavarno dobar framework , Angular.js predstavlja resenje koji svaki web developer, dizjner treba da bude upoznat. Sa druge strane izuzetno je težak za pojedine ljude zato što sadrži koncepte i vokalne izraze koji su teški za razumevanje. Cilj ovog kursa jeste da prodjemo zajedno kroz ključne elemente samog frameworka.

Ne verujem u imitiranje koda i mislim lično da to nikuda ne vodi osim u ćorsokak. Ukoliko naiđete na zid tokom razvoja, nećete biti u stanju da ga preskočite ako budete radili imitirajući nečiji kod. Pričam o scenariju iz realnog života. Mislim..., možda i hoćete uobičajenim googlanjem rešenja ili oslanjanje na nivo vašeg poznavanja samog JavaScripta kao jezika, ali u tom slučaju vreme radi protiv vas. U stvari ono što želimo da postignemo ovim kursom jeste da ne želimo da imitiramo kod već da ga u potpunosti razumemo, ubrzamo i unapredimo razvoj web aplikacija .

Na ovom kursu nećete samo imitirati kako pisem kod već ćete naučiti i osnovne javascript principe koji su ukjuceni u Angular.js, kako Angular.js radi i ono sto treba da bude moj posao jeste da vam pojasnim sve primere koje budem prezentovao da bi ih u potpunosti razumeli i da kasnije implementirate vaše ideje u Angular.js aplikacije i u vašu karijeru pre svega.

**Probem koji angular js želi da resi**

Problem koji Angular.js mora da resi jeste da se aplikacije i web sajtovi ponasaju više prirodno i tečno i naravno da smanjimo nivo posla da bi napravili stvarno dobru web aplikaciju. Koji su to problemi koje Angular.js stvarno želi da reši i koji će vam pomoći da stvarno razumete kako da radi i kako da uspešno rešite probleme. Da pogledamo jedan primer kod.

**Primer 1.1**

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document 1.1</title>

<link href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.0.0-beta/js/bootstrap.min.js" rel="style">

<script src="https://code.jquery.com/jquery-3.2.1.min.js" integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4=" crossorigin="anonymous"></script>

<script src="app.js"></script>

<style>

html, body, input, select, textarea{

font-size: 1.05em

}

</style>

</head>

<body>

<div class="row">

<div class="col-md-12">

<input id="btnStep1" type="button" value="Step 1">

<input id="btnStep2" type="button" value="Step 2">

<div id="step1" class="panel">

<div class="panel-body">

<h1>Step 1</h1>

<p> Info or Step 1</p>

</div>

</div>

<div id="step2" class="panel">

<div class="panel-body">

<h1>Step 2</h1>

<p> Info or Step 2</p>

</div>

</div>

</div>

</div>

</body>

</html>

**app.js**

$(document).ready(function(){

var currentStep = null;

$('#step1, #step2').hide();

$('#btnStep1').click(function(){

$('#step1').show();

$('#step2').hide();

currentStep = 1;

//update database with ajax

});

$('#btnStep2').click(function(){

$('#step1').hide();

$('#step2').show();

currentStep = 2

//update database with ajax

});

})

Na očigled vidimo jedan jQuery kod koji se ponaša uobicajeno u startu sakrijemo dva div elemena koji možemo prikazati i sakriti klikom na dugmiće Step 1 i Step 2. Ovo je dobar koncept i postavka gledano sa development strane ali vremenom ćete želeti da imate 10 ili 20 koraka i programiranje može da postane poprilično nezgodno i neuredno. Morali bi smo dobro da razmislimo kako da izorganizujemo kod a da ne ponavljamo pisanje istog i da sa silnim registrovanim eventima opteretimo memoriju racunara.

Ovakave probleme Angular.js uspesno resava bez upotrebe mnogo koda i ovde nam treba koncept.

**Angular koncept**

Imamo podatke sa jedne strane, html formu sa druge i želimo da povežemo jedno sa drugim. Odnosno želimo da postignemo da onaj podatak koji se promeni na html starni promeni i u kodu samog izvora podatka i obrnuto bez toga da se ta radnja obavlja manuelno. Ovo je ustvari koncept koji postoji već veoma puno vremena i naziva se Model View Controller (MVC). Postoje i drugi koncepti poput Model View View Model (MVVM). U stvari Angular ne podržava ni jedan od ovih arhitektura već Model View Wathewer (MV\*). Šta to znači?

Pretpostavimo da imamo “Model” što predstavlja podatak, ustvari model definiše taj podatak. Kada gledamo kroz javascript uzmimo naprimer variablu:

var firstName=”Marko”

Zatim imate View to je ustvari pretstavlja ono sto korisnik vidi na User Interfaceu dalje u textu UI-u. U slučaju same aplikacije to je .html document:

<h1>Marko</h1>

U ovom konceptu MV\* Whatever je nesto sto veže (two way binding - eng) podatak odnosno “Model” za View I obratno bez toga da to radimo manuelno u samom kodu.

VIEW

<h1>Marko</h1>

MODEL

var firstName=”Marko”

Stvari koje ostvaruju povezivanje izmedju View-a I Modela I obrnuto uključuju nešto što se zove Controller ili View Model ili bilo šta. Angularu ne smeta kako ćete sve ovo zvati sve dok ostvaruje ovaku vezu u realnom vremenu izmedju podatka I UI-a. Mi ćemo baznu postavku u Angularu zvati zvati (MV\*). I to je ono sto Angular.js radi.

**Napomena: HTML strana – custom attributes**

Treba naglasiti da angular radi u okviru html sa custom atributima u okviru HTML dokumenta koje angular koristi i obicno pocinje sa ng-naziv\_atributa angular korisiti nekoliko takvih framework built in atributa koje ćemo upoznati tokom rada.

**Primer 1.2**

**Napomena: JavaScripta strana – global name space**

Ukoliko u javascript fajlu app.js kreiramo varijablu.

var person = “Marko”;

console.log(person);

U konzoli vidimo da se u delu memorije računara nalazi variabla sa vrednošću „Marko“ . Ova varijabla se nalazi na globalnom objektu i dostupna je u svakom momentu. Ovo može biti veliki problem u realnom scenariju. Naprimer ako stavimo jos jedan Javascript fajl u okviru naseg index.html fajla odnosno ako taj fajl predstavlja neku drugu biblioteku nazovimo ga utility.js i postavimo je u nasem html kodu iznad app.js.

U utility fajlu kreiramo sledece:

var person = „Dejan“;

function logPerson(){

console.log(person);

}

Kada osvežimo browser opet vidimo da ispuisuje “Marko” zasto se ovo desava. Prvi js fajl je utility I prvo iz njega smesta u memoriju racunara funkciju logPerson() I vidimo da je variabla person zauzela mesto u memoriji sa vrednoscu “Dejan”. Posle tog fajla gleda na fajl app.js gde se opet nalazi varijabla person koja ima vrednost “Marko” I kada smesta vrednost u memoriju, racunar konta da u memoriji vec postoji odnosno obe varijable su na global namespace-u I smesta novu vrednost “Marko” na mestu te varijable koja sedi u memoriji racunara, zatim izvrsava funkciju koja ima zadatak da iskonzolira varijablu person i Ispisuje u konzoli “Marko”. Shvatate problem. Mi to nismo hteli ali to se desilo. U ovakvim scenarijam čak je tesko odraditi I debug.

Da bi smo izbegli ovakve scenarije treba da se kreira nesto sto se yove namespace.

To radimo tako sto u utility.js fajlu kreiramo prazan objekat dejanApp u nju smeštamo property person I funkciju logPerson() kao metodu. U console.log koja se nalazi u funkciji smestamo property dejanApp.person.

var dejanApp = {}

dejanApp.person = „Dejan“;

dejanApp.logPerson = function (){

console.log(person);

}

U app.js fajlu moramo da isparavimo na poziv inicijalizacije same metode na dejanApp.logPerson():

var person = “Marko”;

dejanApp.logPerson();

Na refresovanju browsera sada vidimo da u konzoli ispisuje ”Dejan” iz razloga što se sada vrednost I sama metoda nalazi na objektu dejanApp a person gde je smestena vrednost “Marko” se jos uvek nalazi na global name spaceu. U koliko želim da mi ispise ponovo vrednost “Marko” u konzoli, sve što treba da uradimo jeste da iznad pozivanja funkcije dejanApp.logPerson() u app.js dodamo još jednu liniju koda

var person = “Marko”;

dejanApp.person = person;

dejanApp.logPerson();

Dakle inkapsulacija nam osigurava da odredjene varijable ne dohvata sa global name spacea. Ovaj vid gradjenja obezbedjuje izradu kompleksnog softvera i veoma bitna stavar koju morate da razumete ako želimo da gradimo angular aplikaciju koja ce biti lako održiva i koja će osigurati da se odredjene komponente mogu upotrebiti na razlicite angular aplikacije u bilo kom trenutku. (reusable components).

Moramo obezbediti nasu aplikaciju koju budemo gradili da budemo sigurni kada budemo deklarisali odredjene delove da se ne nalaze na global namespaceu I da se nebi sudarale sa varijablama sa raznih biblioteka koje budete dovlačili da koristite zajedno sa angularom sa tim povećavate I security same aplikacije.

**Dependency Injection**

Pre nego što krenemo dalje trebalo bi objasniti šta je dependency injection. Ovo je veoma važam i fundamentalan koncept i nešto što ćete čuti od ljudi koji su radili sa angular frameworkom. Pretpostavljam da već znate šta nači ali ipak ćemo objasniti. Ustavari to je jedna od stvari koju treba da razumete pre nego krenemo da radimo sa angularom jer ćemo konstantno koristiti dependency injection.

Dependecy injection predstavlja u stvari davanje objekta funkciji.

Umesto kreiramo objekta unutar same funkcije mi cemo objekat proslediti funkciji. Šta to znači?

var Person = function(firstname, lastname){

    this.firstname = firstname;

    this.lastname = lastname;

}

function logPerson(){

    var marko = new Person('Marko', 'Marković'); //ova funkcija zavisi od variable marko to je dependency.

    //Ukoliko želite nesto da se promeni u okviru objekta person

    //mora se menjati unutar ove funkcije

    console.log(marko)

}

logPerson();

Ovaj primer je loš i ne bi trebali tako da ga radite. Umesto ovog primera uradicemo dependency injection.

var Person = function(firstname, lastname){

    this.firstname = firstname;

    this.lastname = lastname;

}

function logPerson(person){

    console.log(marko)

}

var marko = new Person('Marko', 'Marković'); //Kreiramo objekat van funkcije i prosledjujemo objekat funkciji

logPerson(marko);

Dobili smo istu stvar samo smo objekat kreirali van funkcije i tako inicijalizovan prosleđen je funkciji logPerson(). Zar nije jednostavno. Ovo je dependency injection i angular js ih na ovaj način resava kroz kontrolere i ostale instance. Ovakav rad osigurava da imate snažnu arhitekturu prilikom kreiranja angular instanci.

**Modules, Apps, and Controllers**

Do sada nismo napisali nijedan angular kod ali od ovog trenutka to počinjemo. Udjimo u strukturu same angular aplikacije a to su Modules, Apps I Controllers.

Pocnimo tako sto ćemo u index.html kodu smestiti angular biblioteku.

**Primer 1.3**

**Index.html**

<!DOCTYPE html>

<html lang="en" ng-app="app">

<head>

<meta charset="UTF-8" >

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document 1.1</title>

<link href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.0.0-beta/css/bootstrap.min.css" rel="stylesheet">

<script src="https://code.jquery.com/jquery-3.2.1.min.js" integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4=" crossorigin="anonymous"></script>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.6/angular.min.js"></script>

<script src="app.js"></script>

<style>

html, body, input, select, textarea{

font-size: 1.05em

}

        .btn{

            margin:5px 5px 0px 5px;

        }

</style>

</head>

<body>

<div class="row" ng-controller="mainController">

<div class="col-md-12">

         <button type="button" class="btn" ng-repeat="($i, btn) in btnStepsData" ng-click="setActiveBtnIndex($i)">{{ btn.title}}</button>

         <div ng-if="$i===activeBtnIndex" ng-repeat="($i, btnStep) in btnStepsData" id="step{{$i}}" class="panel">

            <div class="panel-body">

                <h1>{{btnStep.title}}</h1>

                <p>{{btnStep.content}}</p>

            </div>

        </div>

</div>

</div>

</body>

</html>

app.js

(function(){

//MODULE

    angular.module("app", []);

})();

(function(){

//CONTROLLER

    var mainController = function($scope){

        $scope.activeBtnIndex = 0;

        $scope.btnStepsData = [

            {

                title:"Step 1",

                content: "Info about Step 1"

            },{

                title:"Step 2",

                content: "Info about Step 2"

            },{

                title:"Step 3",

                content: "Info about Step 3"

            },{

                title:"Step 4",

                content: "Info about Step 4"

            },{

                title:"Step 5",

                content: "Info about Step 5"

            },{

                title:"Step 6",

                content: "Info about Step 6"

            },{

                title:"Step 7",

                content: "Info about Step 7"

            },{

                title:"Step 8",

                content: "Info about Step 8"

            },{

                title:"Step 9",

                content: "Info about Step 9"

            },{

                title:"Step 10",

                content: "Info of Step 10"

            }

        ];

        $scope.setActiveBtnIndex = function(i){

            $scope.activeBtnIndex = i;

        }

    }

    mainController.$inject = ['$scope'];

    angular.module("app").controller("mainController", mainController);

}());