

AngularJS

Servisi

AngularJS Servisi

- Servisi su specijalizovani za određeni zadatak – obezbeđuju interfejs sa metodama namenjenim za neku specifičnu funkciju
 - Primer: tajmer ili komunikacija preko HTTP
 - Predstavljaju reusable delove koda deljenje između različitih kontrolera, direktiva, filtera i samih servisa
 - Povezuje se putem injekcije zavisnosti (dependency injection)
- Singleton objekti (objekti koji se instanciraju samo jednom po aplikaciji)

\$timeout i \$interval servis

- **\$timeout**
 - Predstavlja Angular wrapper oko JavaScript funkcije `window.setTimeout` – pozovi funkciju nakon što istekne zadati period (izvršava se samo jednom)
- **\$interval**
 - Predstavlja Angular wrapper oko JavaScript funkcije `window.setInterval` – ponavlja poziv funkcije svaki put nakon što istekne zadati period (izvršava se periodično)

```
var TimerController = function($scope, $interval) { ← injektujemo $interval servis u kontroler
  ...
  var countdownInterval = null;
  var startCountdown = function() {
    countdownInterval = $interval(decrementCountdown, 1000, $scope.secondsLeft);
  };
  startCountdown();

  $scope.cancelCountdown = function(){
    $interval.cancel(countdownInterval);
  };
  ...
}
```

funkcija koja se poziva svakih 1000ms

broj ponavljanja

\$interval vraća (promise) objekat: čuvamo ga i posle koristimo da bi smo mogli prekinuti periodični poziv funkcije

TimerController.js

\$log servis

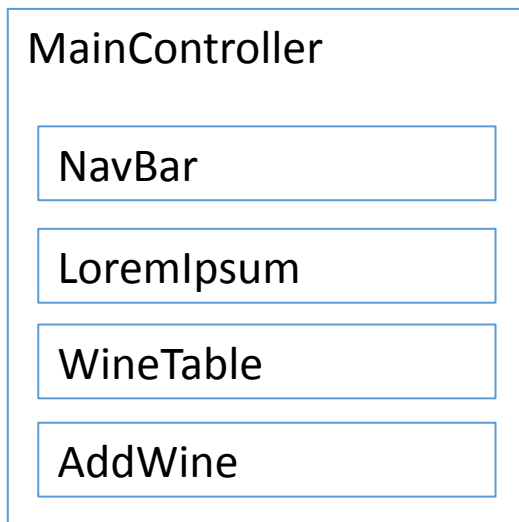
- Ispis poruka u browser konzolu
- Metode:
 - `log()`
 - `info()`
 - `warn()`
 - `error()`
 - `debug()`
- Korisno je prilikom razvoja aplikacije jer uvodi veću semantiku u konzolni ispis

\$window servis

- Najčešće, kada želimo da izmenimo prikaz, radimo to izmenom modela kroz kontroler
- Postoje situacije kada želimo da izmenimo prikaz direktno:
 - Navigacija
 - Prikaz modalnog dijaloga
 - ...
- Za ovo možemo koristiti servise (\$window, \$browser, \$location, \$animate,...)
- \$window – referenca na window objekat browser-a
- Redirekcija na eksterni URL: `$window.location.href = 'http://www.google.com';`
- Alert: `$window.alert("Hello!");`

Kreiranje custom servisa

- Pored već ugrađenih servisa moguće je definisati sopstvene servise
 - Izbegavanje dupliciranja koda
 - Smanjenje složenosti programa
 - Skladištenje deljenih podataka (servis je singleton objekat. Ako imamo istovremeno nekoliko kontrolera koji koriste isti servis sa podacima, svi kontroleri vide iste podatke)
- Primer: problem na koji smo naišli u wineCellar primeru je razmena podataka između „sestrinskih“ kontrolera:



`$scope.wines` objekat nam je bio potreban u WineTable kontroleru i u AddWine kontroleru. Problem smo rešili prebacivanjem podataka u MainController, iako im tamo možda nije mesto

Kreiranje custom servisa

- Najjednostavniji (i najčešće korišćen) način registracije servisa je korišćenje *factory* funkcije dostupne kroz angular.module API

```
(function(){  
  var dataService = function(){  
    var wines = [];  
    var init = function(){  
      var data = {...}; wines = data.wines;  
    };  
    init();  
  
    var getWines = function(){ return wines; }  
    var addWine =function(newWine){ wines.push(newWine); }  
    return {  
      getWines : getWines,  
      addWine : addWine  
    };  
  };  
  var module = angular.module("wineModule");  
  module.factory("dataService", dataService);  
})();
```

Ovaj servis će biti singleton objekat i možemo ga koristiti u celoj aplikaciji. Npr. možemo ga injektovati u neki kontroler i onda pozivati njegove metode, npr. `var WinesController = function($scope, dataService)`

„Revealing module“ šablon: u servisu su nam dostupne dve funkcije: `getWines` i `addWine` (ovo je API servisa)

Registracija servisa u modulu korišćenjem njegove factory funkcije – nije jedini način

Drugi načini registracije custom servisa

- Svi servisi su singleton objekti – instanciraju se jednom po aplikaciji
- Mogu biti bilo kog tipa – primitiva, objekat, funkcija, ...
- Servis je moguće kreirati na 5 načina: **value**, **constant**, **service**, **factory** i **provider**
- value:
 - najjednostavniji
 - `angular.module('app').value('myObj', {prop1:"val1", prop2:"val2"});`
 - u value servis se ne mogu injektovati zavisnosti
- constant:
 - jedina razlika u odnosu na value je što constant servis može biti injektovan u konfiguracionoj fazi app, dok value service ne može
 - koristimo ga za podatke koji treba da su dostupni u konfiguracionoj fazi inicijalizacije modula
 - `angular.module('app').constant('myObj', {prop1:"val1", prop2:"val2"});`

Drugi načini registracije custom servisa

- service
 - Manja mogućnost konfiguracije u odnosu na factory
 - `angular.module('app').service('MyService1',['dep1', function(dep1){
 this.prop1="val1"; this.prop3=dep1.doWork();
}]);`
- factory
 - service objekat se kreira putem factory funkcije, a zavisnosti se injektuju kao argumenti factory funkcije
 - omogućava: korišćenje drugih servisa (zavisnosti), inicijalizaciju servisa, odloženu inicijalizaciju
 - najčešće korišćen
 - `angular.module('app').factory('MyService2', ['dep1', function (dep1) {
 var service = { prop1: "val1", prop3: dep1.doWork() };
 return service;
}]);`
- provider
 - omogućava maksimalnu kontrolu prilikom kreiranja i konfiguracije servisa – još jedan nivo apstrakcije, moguće je konfigurisati factory
 - `angular.module('app').provider('myService3', function () {...}`

Komunikacija sa serverom

- U opštem slučaju front-end aplikacija će se kačiti na server u cilju autentifikacije, autorizacije, dobavljanja/čuvanja podataka, validacije, itd.
- Svaka klijent-server interakcija se u suštini svodi na slanje HTTP zahteva i dobijanje odgovora sa servera
 - Do sada smo videli AJAX zahtev/odgovor mehanizam
- Kao podršku AJAX-baziranoj komunikaciji AngularJS obezbeđuje dva servisa:
 - `$http` – osnovna komponenta za interakciju sa udaljenim serverom putem AJAX-a (slično `ajax` funkciji u jQuery)
 - `$resource` – apstrakcija izgrađena nad `$http` servisom koja olakšava komunikaciju sa RESTful servisima

\$http

- Servis koji obezbeđuje komunikaciju sa serverom putem HTTP protokola
 - Objekat sa metodama koje se mogu pozvati u cilju slanja GET, POST, PUT, DELETE zahteva
- Osnovna sintaksa (retko korišćena): `$http(config)`
- Češće se koriste skraćene notacije:
 - `$http.get(url, [config])`
 - `$http.post(url, data, [config])`
 - `$http.put(url, data, [config])`
 - `$http.delete(url, [config])`
 - `$http.head(url, [config])`
 - `$http.jsonp(url, [config])`
- `config` – konfiguracioni objekat (url udaljenog servera, tip zahteva (GET, POST,...), headeri koji se šalju sa zahtevom,...)

\$http

- AJAX zahtevi: asinhrona komunikacija – browser neće čekati da podaci pristignu sa servera da bi nastavio sa daljim procesiranjem. Da bi obradili pristigle podatke moramo definisati callback funkciju koja će biti pozvana kada stigne odgovor sa servera
- \$http servis se poziva se asinhrono i **ne vraća podatke** nego **obećava da će vratiti podatke** (u budućnosti). Vraća **promise** objekat
- Promise objekat će se razrešiti u budućnosti kada stignu podaci (ili greška)

Promise API

- Metode koje se mogu pozvati nad promise objektom:
 - `then(successCallback, errorCallback, notifyCallback)`
 - *successCallback*: funkcija koja se poziva nakon uspešnog razrešenja promise objekta. Kao parametar se funkciji prilikom poziva prosleđuje razrešena vrednost
 - *errorCallback*: funkcija koja se poziva ukoliko promise bude razrešen greškom. Kao parametar se funkciji prilikom poziva prosleđuje uzrok greške
 - *notifyCallback*: funkcija koja se poziva za izveštaj o progresu promise-a. Ovo je korisno kod asinhronih metoda koje se dugo izvršavaju jer mogu da notifikuju o svom egzekucionom progresu. Funkcija može biti pozvana više puta
 - `then` metoda za provratnu vrednost ima promise objekat
 - `catch(errorCallback)`
 - Skraćena notacija za `then(null, errorCallback)`
 - `finally(callback)`
 - funkcija koja se poziva nakon razrešenja promise objekta nezavisno od toga da li je razrešen uspehom ili greškom. Funkciji se ne prosleđuju parametri. Korisna je kada treba osloboditi neki resurs nakon razrešenja promise objekta

\$http format podataka

- Handlovanje JSON objekata je veoma jednostavno
 - Ukoliko GET zahtev vrati JSON string, on će se automatski konvertovati u JavaScript objekat
 - Za POST i PUT zahteve objekti se automatski serijalizuju i automatski se postavlja odgovarajući header (Content-Type: application/json)
- Ni ostali formati nisu problematični
 - \$http je generički AJAX servis koji može da rukuje bilo kojim formatom zahteva i odgovora

Primer korišćenja \$http servisa

```
(function(){  
  var winesService = function($http){  
    var getAll = function(){  
      var url =...;  
      return $http.get(url).then(onSuccess);  
    }  
  
    var post = function(data){  
      var url =...;  
      return $http.post(url, data).then(onSuccess);  
    }  
  
    var onSuccess = function(response){return response.data;}  
    return {  
      getAll : getAll,  
      post : post  
    };  
  };  
  var module = angular.module("wineModule");  
  module.factory(„winesService", winesService);  
})();
```

winesService servis zavisi od \$http servisa (\$http servis se injektuje u winesService servis)

\$http.get i \$http.post vraćaju promise objekat. Funkcija then takođe vraća promise – povratna vrednost getAll funkcije je promise objekat (promise da je izvršen HTTP zahtev i da se izvršila onSuccess funkcija prosleđena u then metodu

onSuccess funkcija vraća tražene podatke koji se nalaze u data property-ju objekta koji predstavlja HTTP odgovor (response)

\$Primer korišćenja \$http servisa – nastavak

```
var WinesController = function($scope, winesService) {  
    $scope.wines = null;  
  
    var onSuccess = function(data){  
        $scope.wines = data;  
        ---  
    }  
    var onError = function(reason){  
        $scope.wines = null;  
        $scope.message = "Could not fetch the data";  
    }  
    winesService.getAll().then(onSuccess, onError);  
};
```

injektovanje winesService u kontroler

Funkcija koja se poziva ukoliko je promise objekat uspešno razrešen – uraditi sve što je potrebno kada smo dobili podatke

Funkcija koja se poziva ukoliko je promise objekat razrešen sa greškom

Poziv getAll metode winesService – metoda vraća promise pa definišemo successCallback i errorCallback funkcije

Šta se dešava ukoliko imamo grešku u url-u?

\$resource servis

- Gotovo svi javni i privatni servisi (Google, Facebook, Twitter,...) imaju API
- Danas je većina HTTP API-ja RESTful
- REST (REpresentational State Transfer) – arhitektura koja komponente sistema definiše kao resurse
 - Resursi su identifikovani putem URL-a
 - Nad resursima se obavljaju jednostavne operacije
 - Jednostavne operacije nad resursima kao HTTP metode:
 - GET- čitanje resursa
 - POST – kreiranje resursa
 - PUT – ažuriranje resursa
 - DELETE – brisanje resursa
- \$resource je apstrakcija izgrađena nad \$http servisom koja olakšava konzumaciju RESTful servisa

\$resource servis

`$resource(url, [paramDefaults], [actions]);`

- url: URL endopinta
 - Može biti parametrizovan argumentima kojima se dodaje prefiks :, npr. /collection/:identifier
 - Ukoliko vrednost parametra nije dostupna prilikom poziva, parametar se uklanja iz URL-a
- paramDefaults
 - Za parametrizovane URL-ove može da propiše default vrednosti
 - Ostatak parametara (koji nisu default vrednosti) se dodaje na URL
 - Npr. neka URL glasi /users/:name

paramDefaults	Rezultujući URL
{}	/users
{name: 'david'}	/users/david
{search: 'david'}	/users?search=david
{name: 'david', search: 'out'}	/users/david?search=out

- Prilikom poziva ovi paramDefaults se mogu override-ovati

\$resource servis

- actions

- JavaScript funkcija koja se kači na \$resource
- \$resource ima standardni set operacija (get, query, save, delete). Parametar actions omogućava da se lista operacija proširi custom operacijama ili da se redefiniše postojeća operacija
- actions: {action1 : config, action2 : config} definiše dve akcije i konfiguracije za ove akcije (config je isti obekat koji smo koristili kod \$http, 11. slajd)
- Za default metode standardni config je:

```
{ 'get': {method:'GET'},  
  'save': {method:'POST'},  
  'query': {method:'GET', isArray:true},  
  'remove': {method:'DELETE'},  
  'delete': {method:'DELETE'}  
};
```

\$resource servis

- `var myResource = $resource(url, [paramDefaults], [actions]);`
 - `myResource.query()` – vrati sve objekte iz kolekcije
 - `myResource.get({id : 'test'})` – vrati objekat sa `_id=='test'`
 - `myResource.save({}, data)` – snimi objekat u kolekciju
- Za metode bazirane na GET sintaksa je:
 - `myResource.actionName([parameters], [successCallback], [errorCallback])`
- Za metode bazirane na POST/PUT sintaksa je:
 - `myResource.actionName([parameters], postData, [successCallback], [errorCallback])`
- Kada se pozove resource akcija povratna vrednost je:
 - Resource objekat (ukoliko je `isArray=false`)
 - prazan niz (ukoliko je `isArray=true`)
 - Angular će popuniti objekat/niz podacima kada stigne objekat sa servera
- Povratna vrednost ima parametre:
 - `$promise` – promise objekat slično kao kod `$http` servisa
 - `$resolved` – ima vrednost `true` ukoliko je promise razrešen, inače `false`

Korišćenje \$resource servisa

- \$resource je deo ngResource modula
 - moramo include-ovati angular-resource.js u index.html
 - dodati 'ngResource' u listu zavisnosti našeg modula (app.js)
 - u WinesService injektujemo \$resource servis

```
(function(){  
    var app = angular.module("wineModule", ['ngResource']);  
})();
```

app.js

```
...  
<script src="angular-resource.js"></script>  
...
```

index.html

```
(function(){  
    var winesService = function($resource){  
        var url = "https://localhost:3000/api/wines/:id";  
        return $resource(url, {update: {method: 'PUT'}});  
    };  
    var module = angular.module("wineModule");  
    module.factory("winesService", winesService);  
})();
```

Kreira Resource klasu sa 5 default metoda (get/save/query/remove) i 1 custom metodom (update)

WinesService.js

Korišćenje \$resource servisa

- Prikaz svih vina:

```
var WinesController = function($scope, winesService) {  
    winesService.get().$promise.then(function(data){  
        $scope.wines = data;  
        adjustPagination();  
    });  
    ...  
};  
WinesController.js
```

winesService.query odmah vraća prazan niz (da smo stavili \$scope.wines = winesService.query, \$scope.wines bi se u budućnosti popunio podacima sa servera). Ovde smo međutim iskoristili promise objekat kako bi se adjustPagination() izvršilo tek kada podaci stignu sa servera

- Update vina:


```
$scope.updateWine = function(winId){  
    $scope.showPanels.newWine  
        = winesService.get({id: winId});  
    ...  
};  
WinesController.js
```

Vraća vino sa servera sa specificiranim id. Povratna vrednost je resource objekat

- Pozivom get metode nad **Resource klasom** dobijamo **resource objekat**
\$scope.showPanels.newWine = winesService.get({id: winId}); - resource objekat
\$scope.wines = wineService.query – kreira niz koji će biti popunjen resource objektima
- Nad resource objektom imamo definisane iste akcije kao nad Resource klasom, samo što njihova imena imaju prefiks \$ (\$get, \$save, \$update, \$delete, ...)

Korišćenje \$resource servisa

```
$scope.addWine = function(){  
  if($scope.showPanels.newWine._id){ Ukoliko je definisan id, updatujemo vino  
    $scope.showPanels.newWine.$update({id: $scope.showPanels.newWine._id},  
                                       onSuccessSave);  
  }else{ Inače definišemo id i snimimo ga u bazu  
    $scope.showPanels.newWine._id = $scope.showPanels.newWine.name.split(' ').join('_');  
    winesService.save($scope.showPanels.newWine, onSuccessSave);  
  }  
}
```

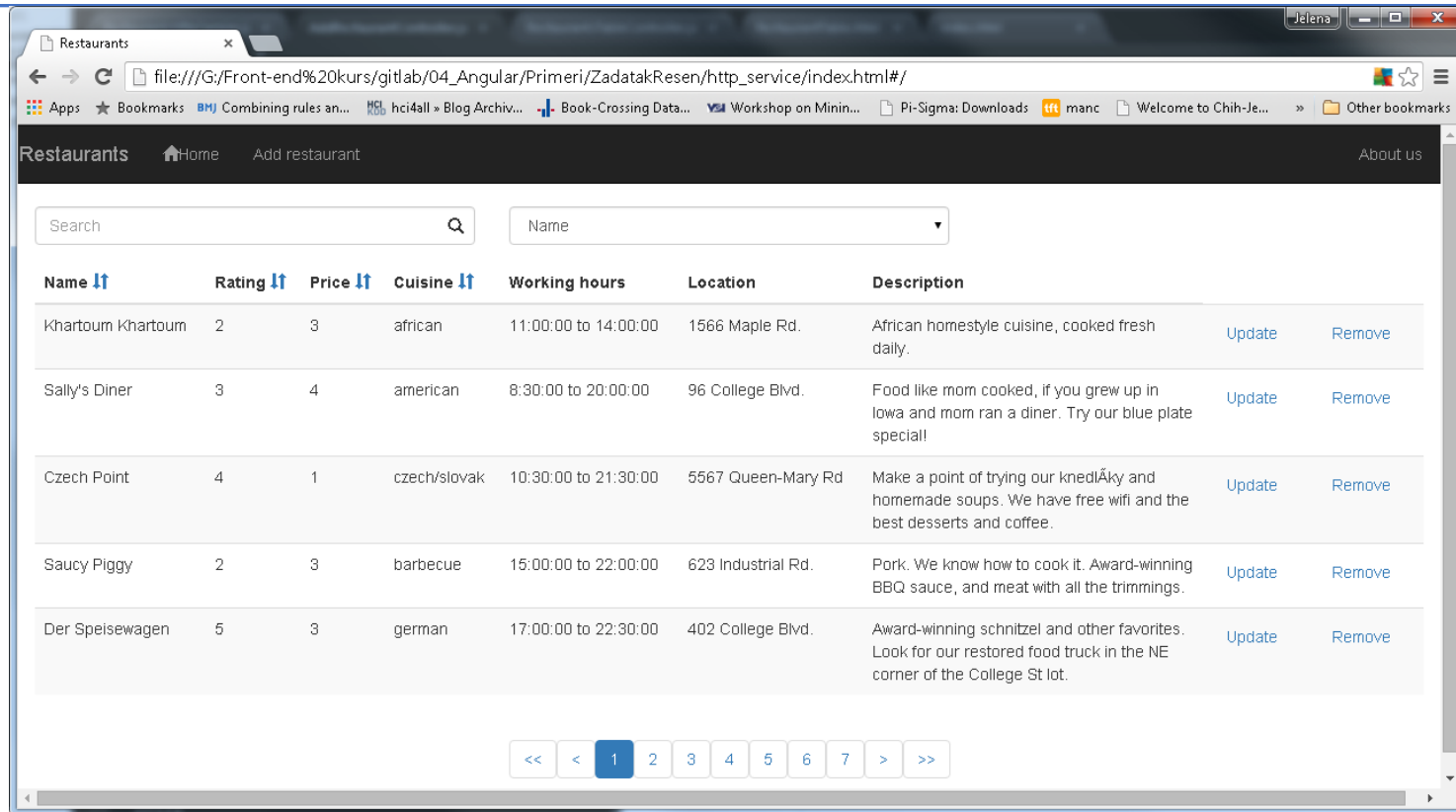
Funkcija koja se poziva nakon uspešnog snimanja/učitavanja  [AddWineController.js](#)

- Akcije nad resource objektom izvlače podatke iz samog resource objekta (za winesService.save potreban je payload, odnosno prosleđivanje wine objekta), dok za newWine.\$update nije
- Koristiti akcije nad resource objektom u slučaju da se radi o kontekstu jednog item-a (update i delete operacije). Inače, koristiti \$resource servis

Zadatak

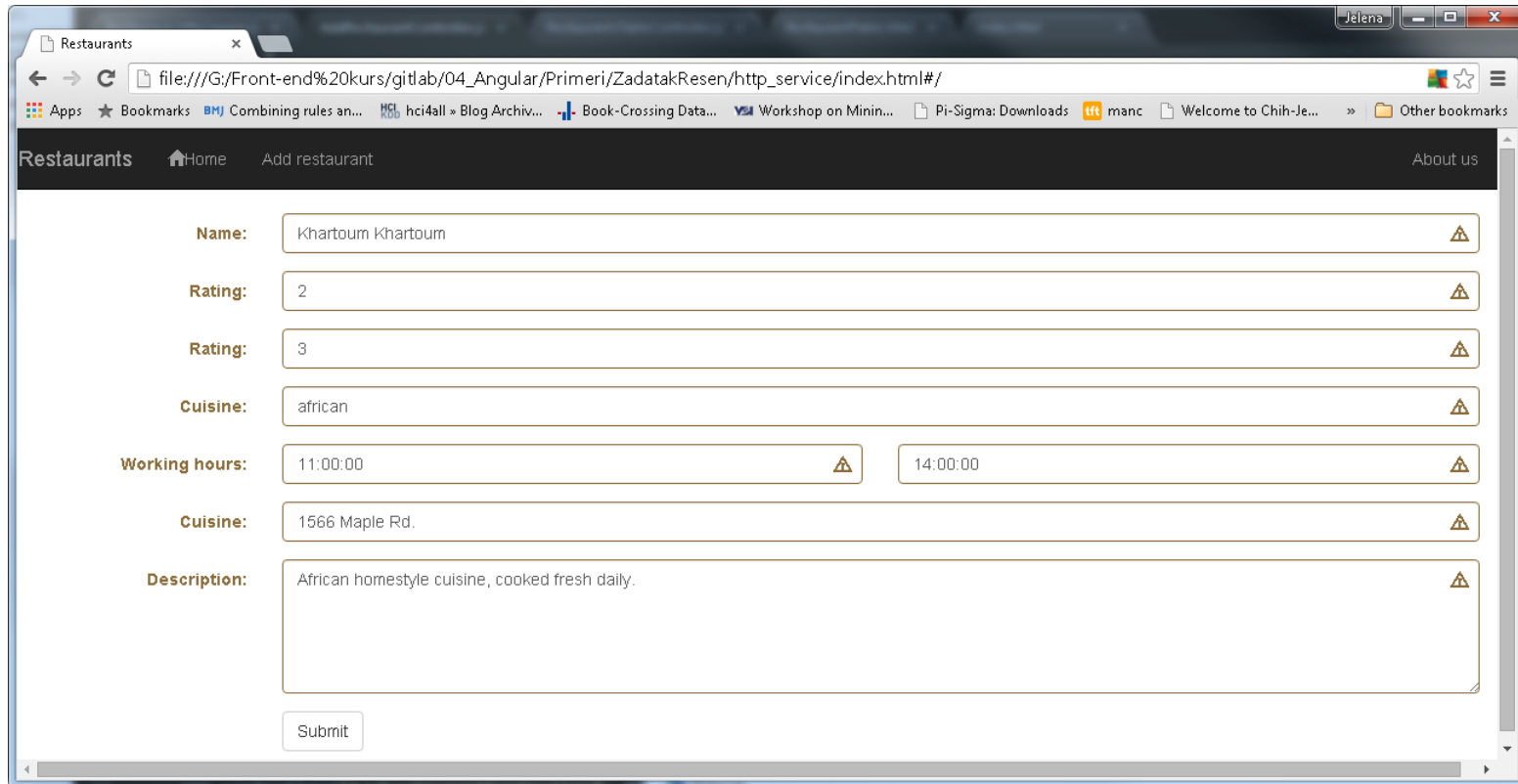
- U fajlu `restaurantsData.json` je dat JSON objekat koji predstavlja niz restorana.
- Napraviti web sajt za pregled/ažuriranje restorana koji obezbeđuje sve CRUD (create, retrieve, update i delete) operacije nad restoranima putem
 1. `$http servisa`
 2. `$resource servisa`
- Omogućiti detaljan prikaz za svakog restorana i CRUD operacije nad stavkama menija

Zadatak - izgled aplikacije



- Ukoliko korisnik klikne na **Add restaurant** prebaciti se na formu za unos restorana (AddRestaurant.html). Polja za unos treba da su prazna
- Ukoliko korisnik klikne na **Remove** ukloniti red iz tabele
- Ukoliko korisnik klikne na **Update** prebaciti se na prikaz forme za unos restorana (AddRestaurant.html). U formi polja treba da su popunjena podacima o restoranu koji se ažurira (slika na sledećem slajdu). Podatke dobiti pomoću GET zahteva upućenog serveru.

Zadatak – dodavanje i ažuriranje



The screenshot shows a web browser window with the title 'Restaurants'. The address bar shows the file path: `file:///G:/Front-end%20kurs/gitlab/04_Angular/Primeri/ZadatakResen/http_service/index.html#/`. The browser's bookmark bar is visible. The page has a dark header with 'Restaurants' and navigation links: 'Home', 'Add restaurant', and 'About us'. The main content area contains a form with the following fields:

- Name:** Khartoum Khartoum
- Rating:** 2
- Rating:** 3
- Cuisine:** african
- Working hours:** 11:00:00 to 14:00:00
- Cuisine:** 1566 Maple Rd.
- Description:** African homestyle cuisine, cooked fresh daily.

A 'Submit' button is located at the bottom of the form.

- Na slici se nalaz primer ažuriranja restorana. Nakon submit korisnik se prebacuje na prikaz tabele sa restoranima.
- Radi provere da li je restoran ažuriran osvežiti stranicu (još nismo videli svu funkcionalnost Angulara da bi smo imali elemente kojima bi se stranica automatski osvežila prema podacima sa servera).