AngularJS

Rutiranje

Single Page Application (SPA)

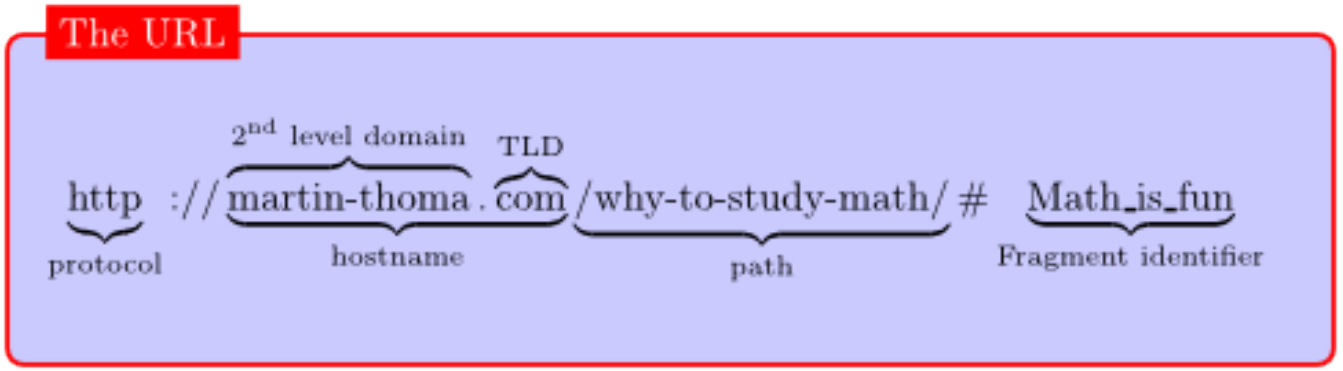
- U SPA, sadržaj se ne učitava sav odjednom već se delovi sadržaja preuzimaju AJAX pozivima u onom trenutku u kome su potrebni
  - Ni u jednom trenutku se ne radi osvežavanje cele stranice niti se vrši prebacivanje na neku drugu stranicu unutar aplikacije
  - Stvara se samo privid navigacije – zapravo se dešava samo logička promena stranica, iako je korisnik sve vreme na istoj stranici
  - Glavna ideja: renderovati i menjati samo elemente korisničkog interfejsa za koje je to potrebno
- Postoje mnoge biblioteke koje služe za izgradnju SPA (AngularJS, Backbone.js, Ember.js, ...)

Rutiranje

- Pomoću AngularJS-a razvijamo single-page aplikacije
- Rutiranje omogućava:
  - Navigaciju među različitim prikazima
  - Razmenu parametara među kontrolerima koji upravljaju prikazima
  - Back i forward u browser-u

Definisanje pravila navigacije

- Kako znamo u kom se delu aplikacije korisnik nalazi? URL obezbeđuje jednoznačno lociraje resursa
  - Na serveru (tradicionalne aplikacije)
  - Na serveru ili **na klijentu** (single-page aplikacije)

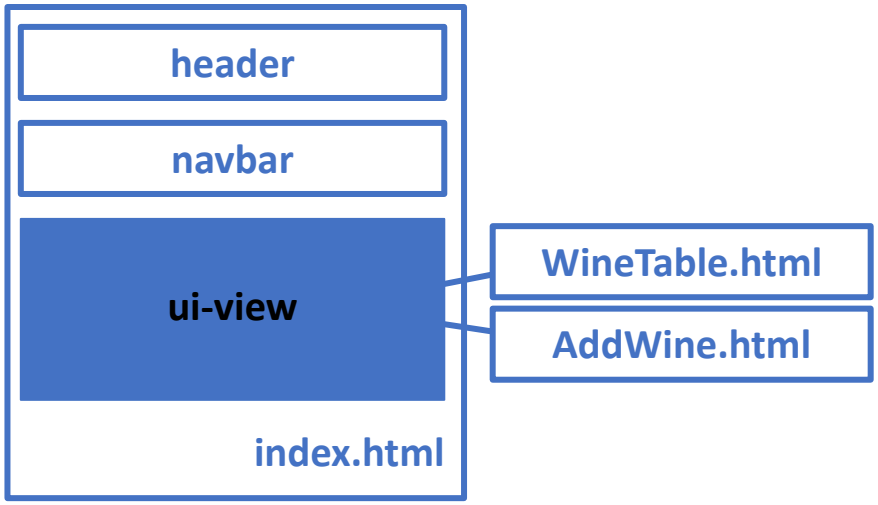


- Primer – URL je /users/angular
  - Engine za rutiranje treba da prepozna URL i da odredi šta će se dešavati – klijent želi da vidi podatke o korisniku „angular“
  - Za to se koriste pravila – ako je URL /users/:username prikazuje se prikaz sa detaljima o korisniku

ui-router

- **Layout view**: osnovna html stranica ima statički sadržaj (koji se prikazuje na svakom prikazu) i sekciju u kojoj se smenjuju templejti

Potrebna nam je 3rd party biblioteka angular-ui-router.js
http://angular-ui.github.io/ui-router/release/angular-ui-router.js




```
<script src="angular-ui-router.js"></script>
...
<body>
    <div ng-include="'Header.html'"></div>
    <div data-spy="affix" data-offset-top="280">
        <div ng-include="'Navbar.html'"></div>
    </div>
    <div ui-view></div>
</body>
```

index.html

Dodajemo zavisnost od modula ui-route:

```
var app = angular.module(" uiRouterExample ", ['ui.router']);
```

Menja se struktura osnovne html stranice:

- Koristimo direktivu **ui-view** da definišemo gde će se umetati templejti u stranicu
- Pri tome se automatski ažurira istorija pregledanja (back i forward)

ui-router – stanje aplikacije

- Menja aplikaciju na osnovu **stanja aplikacije**

- Stanje (state)
  - Predstavlja mesto na kome se trenutno nalazimo u aplikaciji
  - Opisuje kako izgleda i šta radi korisnički interfejs (putem kontrolera, templejta)
  - Stanja često imaju dosta toga zajedničkog, pa postoji mogućnost hijerarhijskog organizovanja stanja (pravljenje ugnježdenih stanja)

- Definisanje stanja (u module.config(…)):

```
$stateProvider
.state("firstState", {
    url: "/state1",
    template: "<h2>FirstPage</h2>"
})
```

app.js

ui-router – templejti

- Kada se stanje aktivira, templejti će se automatski umetnuti unutar ui-view

```
<body>
    <div ui-view></div>
</body>
```
index.html

- Template je moguće zadati:

```
$stateProvider                                  kao string
.state("firstState", {
            url: "/state1",
            template: "<h2>First Page</h2>"
})
```

```
$stateProvider                                  kao URL
.state(“secondState", {
            url: "/state2",
            templateURL: “SecondPage.html"
})
```

I na druge načine: https://github.com/angular-ui/ui-router/wiki

ui-router – aktivacija stanja

- Stanje se aktivira na jedan od sledećih načina:
  1. Klikom na link sa URL-om stanja

```
<a href="#/state1">First page</a>
```
index.html

  2. Klikom na link koji sadrži ui-sref direktivu

```
<a ui-sref="secondState">Second page</a>
```
index.html

ui-sref ukazuje na stanje aplikacije. Na osnovu stanja (ukoliko stanje ima URL) će se kasnije izgenerisati i postaviti vrednost href atributa

     - ui-sref direktiva vezuje link za stanje
     - Ako stanje ima URL, direktiva će automatski izgenerisati i postaviti href atribut
     - Moguća je navigacija sa prenosom parametara
       - ui-sref='stateName({param1: value, param2: value})'

  3. Pozivom $state.go() funkcije

Konfiguracija aplikacije

```
var app = angular.module("uiRouterExample", ['ui.router']);
```
u modul dodati ui-router zavisnost

```
app.config(function($stateProvider, $urlRouterProvider){
```
Pravila rutiranja zadajemo preko provajdera $stateProvider i $urlRouterProvider

```
    $urlRouterProvider.otherwise('/state1');
```
ako nismo ništa pronašli, redirektujemo se na /state1

```
    $stateProvider
    .state("firstState", {
```
naziv stanja

```
        url: "/state1",
```
bookmarkable URL za parcijalni prikaz (http://<hostname>/#/state1)

```
        template: "<h2>First page</h2>"
    })
    .state("secondState", {
        url: "/state2",
        templateUrl: "SecondPage.html"
    });
});
```

01_ui-router/index.html

Config/run faza u AngularJS-u

- U prethodnim slajdovima smo pomenuli da se pravila rutiranja definišu prilikom konfiguracije aplikacije
- Kada je DOM stranice spreman, AngularJS traži ng-app direktivu i počinje inicijalizaciju modula
  - učitava modul definisan pomoću ng-app
  - učitava sve dependecy-je koje ima dati modul, moduli od kojih on zavisi, ...
- Prilikom inicijalizacije svaki modul prolazi kroz dve faze:
  - config
    - služi za inicijalne postavke, npr. konfiguracija pravila rutiranja
    - app.config(...) – definiše se callback funkcija koja će se pozvati prilikom faze konfiguracije
    - u callback funkciju se ne mogu injektovati servisi ili filteri kao zavisnosti. Injekcija $routeProvider je moguća jer je to specijalna klasa servisa - *provider*
  - run
    - u ovoj fazi aplikacija je u potpunosti konfigurisana i spremna za korišćenje

ui-router – kontroleri

- Kontroler je moguće zadati:

navođenjem

```
.state("firstState", {
      url: "/state1",
      template: "<h2>{{title}}</h2>",
      controller: function($scope){
            $scope.title = "First Page";
      }
})
```

zadavanjem naziva

```
.state("secondState", {
      url: "/state2",
      templateUrl: "SecondPage.html",
      controller : 'SecondPageController'
});
```

***Napomena*: u ovom slučaju kontrolere ne navoditi u templejtima pomoću ng-controller direktive!**

I na druge načine: https://github.com/angular-ui/ui-router/wiki

02_ui-router_kontroleri/index.html

Resolve

- Koristi se da kontroleru obezbedi podatke ili sadržaj koji odgovara datom stanju
- Opciona mapa zavisnosti koje se injektuju u kontroler. Mapa sadrži ključ/vrednost parove:
  - key (String) – ime zavisnosti koja se injektuje u kontroler
  - value (String ili funkcija)
    - Ukoliko je String, radi se o servisu
    - Ukoliko je funkcija, njena povratna vrednost se tretira kao zavisnost
- Među zavisnostima se može pojaviti jedan ili više promise objekata:
  - **svi** promise objekti će se razrešiti i konvertovati u vrednost **pre nego što se kontroler instancira**
  - Da bi se route promenio, **svi promise objekti moraju biti uspešno razrešeni**, ukoliko se bilo koji promise razreši neuspehom neće doći do redirekcije na prikaz
  - Ovo simplifikuje kod za inicijalizaciju kontrolera koji kontroliše dati prikaz (podaci se prosleđuju kontroleru umesto da ih on dobavlja)

Resolve primer

- Situacija: Imamo tabelu sa listom vina. Korisnik može da odabere detaljan prikaz vina za koga je potrebno dobaviti neke dodatne podatke sa servera. Pre nego što se redirektujemo na stranicu sa detaljnim prikazom vina, želimo da budemo sigurni da su ti podaci dobavljeni i da možemo da ih prikažemo.

```
app.config(function($stateProvider, $urlRouterProvider){
    ...
    .state("wineDetails", {
        url: "/main/:id",
        templateUrl: "wineDetails.html",

        controller : function($scope, wine){
            $scope.wine = wine.data;
        },

        resolve: {
            wine: function(WinesDataService, $stateParams){
                    return WinesDataService.get($stateParams.id);
                }
        }
});
```

```
<a href="#/main/{{wine._id}}">Details</a>   index.html
```

Angular : vidi kao parametar – podatak koji treba da ekstrahuje iz URL-a i prosledi drugim komponentama

Kada se promise uspešno razreši, property wine se može injektovati u AddWineController

Povratna vrednost funkcije je promise: redirektovaćemo se na prikaz samo ako uspešno dobavimo wine objekat

Pomoću $stateParams servisa se može pristupiti parametrima novog route-a. Ovaj servis se može injektovati u kontrolere i servise za pristup parametrima. Ima jedan ključ po URL parametru

Moguće je definisati više resolve property-ja

Primeri 03_resolve i 04_wines_ui_route

Resolve

- Prednost: Neće se dešavati da se stranica učita pa da se naknadno pojavljuju njeni delovi koji zavise od dobavljanja neophodnih podataka
- Mana: ukoliko dobavljanje podataka traje dugo, korisnik može pomisliti da nije pokrenuo akciju redirekcije

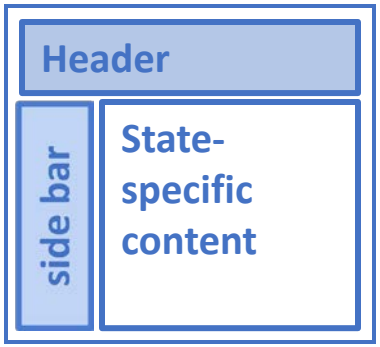
- Ukoliko se koristi resolve, kontroler se mora definisati u pravilima rutiranja (ne može putem ng-controller direktive) jer se ne moze injektovati property resolva ako se injektuje preko html-a

Angularov ng-route

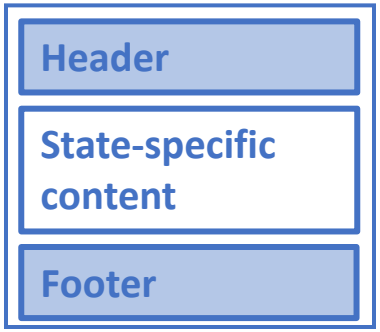
- Pored ui-routera koji je 3rd party biblioteka postoji i Angularov ng-route
- Nešto manje mogućnosti u odnosu na ui-router
- Angular-ov ng-route
  - osnovna stranica sa sekcijom u koju se umeću templejti
  - jedan templejt – jedno pravilo rutiranja – jedan kontroler, omogućava definiciju samo jednog glavnog prikaza (putem ng-view)
  - glavne prikaze ne možemo dalje izdeliti na logičke celine što bi doprinelo da kod bude kraći i čistiji (nije moguće imati ugnježdene prikaze)

ui-router: ugnježđena stanja

- Prilikom izgradnje web sajta često nailazimo na situaciju da više različitih web stranica imaju zajednički layout, npr.



1. grupa stranica: uvek se pojavljuju header i side bar:
- Napravićemo roditeljsko stanje (template: header i side bar)
- Napravićemo specifična stanja koja nasleđuju roditeljsko stanje (template: sadržaj specifičan za stanje)



2. grupa stranica: uvek se pojavljuju header i footer:
- Napravićemo roditeljsko stanje (template: header i footer)
- Napravićemo specifična stanja koja nasleđuju roditeljsko stanje (template: sadržaj specifičan za stanje)

ui-router: ugnježđena stanja

ovde će se ugraditi roditeljsko stanje

```
...
<body>
  <div ui-view></div>
</body>
...
```

index.html

```
<h1>Nested views examle</h1>
<p>This is the main state</p>
<div>
      <a ui-sref=".state1" class="btn btn-primary">State 1</a>
      <a ui-sref=".state2" class="btn btn-primary">State 2</a>
</div>
<p><div ui-view></div></p>
```

main.html

za povezivanje sa child stanjima koristimo tačka notaciju

Ugrađeni view ima svoj ui-view: child state se ovde ugrađuje u parent state

```
$stateProvider
    .state("main", {
        url: "/main",
        templateUrl: "main.html„})
    .state("main.state1", {
        url: "/state1",
        template: "<div>This is state 1</div>„})
    .state("main.state2", {
        url: "/state2",
        template: "<div>This is state 2</div>"});
    });
```

URL je /main/state2

app.js

Vezu roditelj-dete je moguće zadati koristeći tačka notaciju .state('parentStateName.childStateName'). Drugi način je korišćenje parent property-ja (parent: 'main')

05_nested_states/index.html

ui-router: ugnježđena stanja

- ui-router omogućuje da se templejti umeću jedan u drugi kroz ugnježđavanje stanja
- Kada je neko stanje aktivno, aktivna su i sva njegova roditeljska stanja
- Ugnježđena stanja nasleđuju razrešene zavisnosti (resolve property $stateProvider-a)
- Kontroleri se ne nasleđuju – svako ugnježđeno stanje može imati svoj zaseban kontroler
- Stanje može biti i apstraktno
  - Može da ima ugnježđena (child) stanja
  - Samo apstraktno stanje ne može biti nikada aktivirano. Aktivno je samo ukoliko je aktivno neko od njegovih child stanja
  - Primena: želimo da imamo URL koji prethodi URL-ovima svih child stanja, imamo resolve koji bi trebao da prethodi tranziciji na sva child stanja, želimo da definišemo delove templejta zajedničke za child stanja, ...
  - Stanje postavljamo da bude apstraktno postavljanjem parametra abstract:true prilikom konfiguracije stanja
  - Bez obzira što je apstraktno, templejt stanja mora da sadrži <ui-view/> u koji će se ugraživati child stanja

ui-router: više imenovanih prikaza

- U istom templejtu je moguće imati više ui-view direktiva
  - npr. imamo sidebar koji ima popularne postove, nedavne postove, korisnike, itd. Svaki deo se može izdvojiti u poseban prikaz i injektovati u template
  - svakim prikazom može da se upravlja pomoću posebnog kontrolera
  - ovo čini aplikaciju mnogo čistijom, a takođe nam obezbeđuje reusability
- Potrebno je imenovati prikaze za šta se koristi *views* svojstvo stanja

```
body>
  <h1>Multiple views example</h1>
  <div ui-view="firstView"></div>
  <div ui-view="secondView"></div>
</body>
```
index.html

```
$scope.message = "This is the 1st view";
```
Controller1.js

```
$scope.message = "This is the 2nd view";
```
Controller2.js

```
$stateProvider
.state("main", {
      url: "/main",
      views: {
          "firstView": {
              template: "<p>{{message}}</p>",
              controller: "Controller1”},
          "secondView": {
              template: "<p>{{message}}</p>",
              controller: "Controller2"}
      }
})
```
app.js

06_multiple_views/index.html

$q servis

- Asinhroni pozivi u AngularJS-u oslanjaju se na $q servis, deferred objekte i promise objekte
- $q servis omogućuje asinhrone pozive funkcija
- Pomoću $q servisa možemo da kreiramo deffered objekat koji modeluje odložen završetak izvršavanja funkcije

deffered objekat

- Dobija se pozivom $q.defer()
- Omogućava kreiranje promise objekta
- Nudi API kroz koji se promise objektu signalizira da je aktivnost završena uspešno ili neuspešno:
  - resolve(value) – razrešava izvedeni promise objekat i postavlja mu vrednost value
  - reject(value) – odbacuje (neuspešno razrešava) izvedeni promise objekat uz navođenje razloga
  - notify(value) – pruža obaveštenje o statusu izvršavanja. Može se pozvati više puta pre nego što se promise objekat razreši

07_q_service/index.html

Ulančavanje promise objekata

- Primer: želimo da izračunamo izraz (*number*+1) + *number**2. Imamo servis sa dve operacije: *increaseByOne* i *multiplyByTwo.* Svakoj od ovih operacija je potrebno 2 sekunde da se izvrši. Nakon što su se obe operacije izvršile sabiramo 2 broja

```
calcService.increaseByOne(number, function(retVal){
    $scope.retVal1 = retVal;
    calcService.multiplyByTwo(number, function(retVal){
        $scope.retVal2 = retVal;
        $scope.finalRes = $scope.retVal1 + $scope.retVal2;
    });
})
```

Ugnježdavanje callback funkcija – kod je dosta nerazumljiv

- Rezultat *then* funkcije je promise objekat – ovo nam omogućava ulančavanje promise-a:

```
calcService.increaseByOne(number)
    .then(function(retVal){
            $scope.retVal1 = retVal;
            return calcService.multiplyByTwo(number); })
    .then(function(retVal){
            $scope.retVal2 = retVal;
            $scope.finalRes = $scope.retVal1 + $scope.retVal2;});
```

- **Kod je razumljiviji. Ali nema smisla čekati da se prvi broj uveća za 1 da bi se drugi množio sa dva**
- **Ulančavanje bi imalo smisla kada bi nam za dugu operaciju bili potrebni podaci dobavljeni u 1. operaciji. Npr. dobavljanje stavki menija restorana ima smisla tek kada smo dobavili restoran**

$q.all

- $q.all prihvata niz promise objekata
  - Omogućava da se registruje funkcija koja će se izvršiti tek kada se svi promise objekti razreše
  - Ovim mehanizmom možemo paralelno izvršiti više nezavisnih zahteva, a obezbediti se da su svi izvršeni pre nego što izvršimo neki posao sa dobavljenim podacima

```
var promise1 = calcService.increaseByOne(number);
var promise2 = calcService.multiplyByTwo(number);

$q.all( [promise1, promise2] ).then(function(data){
    $scope.retVal1 = data[0];
    $scope.retVal2 = data[1];
    $scope.finalRes = data[0] + data[1];
});
```

niz promise objekata

$q.all takođe vraća promise. Agregirani promise se može razrešiti

- **uspehom – svi promise objekti iz niza su uspešno razrešeni**
- **neuspehom – barem jedan promise objekat je neuspešno razrešen**

pomoću *then* smo registrovali successCallback

povratna vrednost: niz razrešenih vrednosti koji vrati svaki od promise-a

(u slučaju da je agregirani promise neuspešno razrešen povratna vrednost je ista kao kod neuspešno razrešenog promise-a koji je rezultovao da agregirani promise bude rejected)

- Primer realne upotrebe: kod resolve – učitavanje više entiteta pre prikaza stranice

08_chaining_promises/index.html

Custom filteri

- Filteri: formatiranje podataka za prikaz (bez izmene samih podataka)
- AngularJS ima jednostavan API za kreiranje filtera:

```
var app = angular.module(“moduleName");
app.filter(“filterName", function() {
        return function(input, optional1, optional2){
                var output;
                // ovde obaviti posao filtriranja
                return output;
        }
});
```

Slično deklaraciji kontrolera

Ime filtera

Factory funkcija: kreira i vraća drugu funkciju (koja vrši posao filtera)

Obavezan parametar: podaci nad kojima filter radi

Imamo mogućnost da prosledimo više dodatnih parametara koji se opciono mogu proslediti u filter

Funkcija bi trebala da bude stateless i idempotentna (ponavljanje poziva funkcije sa istim parametrima će uvek davati isti rezultat)

Custom filteri – primer

- Ordinalni prikaz broja, npr. ukoliko je originalni broj 3, želimo da se prikaže kao “3rd”

```
var app = angular.module("filterTestModule");
app.filter('ordinal', function(){
    return function(number){
        if( isNaN(number) || number<1){
            return number;
        }else{
            var lastDigit = number % 10;
            if(lastDigit === 1){
                return number + "st";
            }else if(lastDigit === 2){
                return number + "nd";
            }else if(lastDigit === 3){
                return number + "rd“;
            }else if(lastDigit > 3){
                return number + "th";
            }
        }
    }
});
```

ordinalNumFilter.js

```
<ul ng-repeat="num in array">
    <li>{{num | ordinal}}</li>
</ul>
```

index.html

Obavezan parametar: broj koji se pretvara u ordinalni zapis

Custom filteri – primer

- Filter koji kapitalizuje ili prvo slovo ili slovo pod rednim brojem koga mi specificiramo prosleđivanjem dodatnog parametra

```
app.filter('capitalize', function() {
    return function(input, char) {
        if (isNaN(input)) { // ukoliko se ne radi o broju
            var char = char - 1 || 0; // ukoliko je char undefined imaće vrednost 0
            var letter = input.charAt(char).toUpperCase();
            var out = [];
            for (var i = 0; i < input.length; i++) {
                if (i == char) {out.push(letter);}
                else {out.push(input[i]);}
            }
            return out.join('');
        } else { return input;}
 }
```

capitalizingFilter.js

```
<ul ng-repeat="word in array">
    <li>{{word | capitalize:3}}</li>
</ul>
```

index.html

Prosleđivanje dodatnog parametra filteru

09_custom_filter/index.html

Zadatak

U zadataku iz prethodnog termina (aplikacija za pregled/ažuriranje restorana) podesiti rutiranje primenom ui-router-a. Prilikom prelaza na stranicu za ažuriranje restorana koristiti resolve kako bi se podaci dobavili pre nego što se prikaže stranica sa detaljnim prikazom vina (AddRestaurants.html)