AngularJS projekat

Struktura projekta i prateći alati

Struktura datoteka

- Zlatno pravilo: Jedna komponenta po datoteci

```
/* izbegavati */
angular
    .module('app',[])
    .controller('MainController', MainController)
    .factory('mainFactory', mainFactory)

function MainController() { … }

function mainFactory() { … }
```

```
/* preporučeno */
//app.module.js
angular
    .module('app', [])
```

```
//main.controller.js
angular
    .module('app')
    .controller('MainController', MainController)
function MainController() { … }
```

```
//main.service.js
angular
    .module('app')
    .factory('mainFactory', mainFactory)
function mainFactory() { … }
```

Struktura datoteka

- Datotekama i komponentama davati značajne nazive
- Odrediti konvenciju imenovanja i pridržavati se nje tokom razvoja aplikacije
- Preporuka šablona za imenovanje datoteka:
  - <komponenta aplikacije>.<angular komponenta>.js
  - primer: `wines.controller.js` (dok bi naziv kontrolera bio `WinesController`)
- Konvencije:
  - Kad postoji više modula glavni se zove `app.module.js`, dok ostali dobijaju ime spram onoga što predstavljaju (npr. `admin.module.js`)
  - Odvajati konfiguraciju modula u posebnu datoteku (npr. za `app.module.js` formirati `app.config.js`)
  - Odvojiti konfiguracije vezane za route (URL putanje) u posebnu datoteku (npr. `app.route.js` i `admin.route.js`)

Struktura datoteka

- Konvencije (nastavak):
  - Komponente definisati u samopozivajuće funkcije
  - Koristiti imenovane funkcije prilikom definisanja komponenti

```
/* izbegavati */
angular
    .module('app')
    .factory('logger', function() {
        …
    });
```

```
/* preporučeno */
(function() {
    'use strict'

    angular
        .module('app')
        .factory('logger', logger);

    function logger() { … }
})();
```

ControllerAs sintaksa

- Do sada smo koristili `$scope` kada god smo želeli da povežemo kontroler i HTML
- `{{name}}` smešten u HTML-u će tražiti name polje u `$scope` objektu koji je ubačen u kontroler vezan za HTML
- Šta ako želimo da pristupimo polju koje je u roditelju od aktuelnog scope-a?
  - `$scope.$parent.name`
- Šta ako želimo da pristupimo polju koje se nalazi tri stepena iznad?
  - `$scope.$parent.$parent.$parent.name`

ControllerAs sintaksa

- ControllerAs sintaksa rešava prethodni problem tako što ne koristimo `$scope`, već imenujemo kontroler
  - `ng-controller="WineController as wc"`
- Svakom polju u HTML-u pristupamo putem <ime kontrolera>.<ime polja>, bez obzira u kom kontroleru se trenutno nalazimo (primer: `{{wc.wine.name}}`)
- U kontroleru se ne injektuje `$scope` (osim ako nam ne trebaju posebne scope funkcije) već se koristi ključna reč this (primer: `this.wine = {name: "chardonnay"}`)
- Bolja praksa: na vrhu kontrolerske funkcije napisati `var wc = this`, i onda i u kontroleru pristupati svemu kao u HTMLu (`wc.wine = {name: "chardonnay"}`)

primer 1

Dependency injection

- Potrebno je ručno identifikovati svaku zavisnost koju komponenta ima kako ne bi nastao problem prilikom minifikacije
- Ovo je moguće uraditi putem `$inject` atributa funkcije komponente ili putem *inline* niza

```
/* izbegavati */
angular
  .module('app')
  .controller('Controller', Ctrl);

function Ctrl(someService) { … }
```

```
/* preporučeno */
angular.module('app')
  .controller('Controller', Ctrl);
Ctrl.$inject = ['someService']
function Ctrl(someService) { … }

/* ili */
angular.module('app')
  .controller('Controller', ['someService', Ctrl]);
function Ctrl(someService) { … }
```

- Napomena: Redosled navođenja zavisnosti mora biti isti kao u funkciji komponente

Struktura projekta

- U ranoj fazi razvoja projekta struktura direktorijuma nije bitna
- Na kraće staze ovo omogućava brzi razvoj koda, ali na duže staze utiče na održavanje i snalaženje u kodu
- Kako organizovati datoteke?
  - Segmentirati po angular komponentama
  - Segmentirati po komponentama aplikacije
  - Kombinacija prethodna dva pristupa

Segmentacija po angular komponentama

```
app/
----- controllers/              // Sadrži sve kontrolere aplikacije
---------- mainController.js
---------- otherController.js
----- directives/               // Sadrži sve direktive aplikacije
---------- mainDirective.js
---------- otherDirective.js
----- services/                 // Sadrži sve servise aplikacije
---------- mainService.js
---------- otherService.js
views/                          // Sadrži sve html stranice
----- mainView.html
----- otherView.html
assets/
----- img/                      // Slike i ikonice
----- css/                      // Datoteke gde se definišu stilovi
----- js/                       // Spoljne biblioteke koje se koriste
index.html
```

Segmentacija po angular komponentama

- Naveden pristup je dobar kada se pravi mala aplikacija
- Angular komponente su jasno odvojene i kada je aplikacija mala jednostavno je pronaći svaki deo koda
- Problem nastaje kod većih aplikacija, gde se pojavljuje veći broj kontrolera, direktiva, HTML stranica itd.
- Ako želimo da menjamo neku komponentu aplikacije potrebno je da “skupimo” sve datoteke koje ta komponenta koristi pre početka rada

primer 2

Segmentacija po komponentama aplikacije

```
app/
----- shared/                        // Sadrži komponente koje se koriste na više mesta
---------- sidebar/
--------------- sidebar.directive.js
--------------- sidebar.html
---------- article/
--------------- article.directive.js
--------------- article.html
----- components/                    // Svaka komponenta je mini angular aplikacija
---------- blog/
--------------- blog.controller.js
--------------- blog.service.js
--------------- blog.html
--------------- blog.module.js
--------------- blog.routes.js
---------- home/
--------------- ...
----- app.module.js
----- app.routes.js
assets/                             // Sadrži stilove, slike, spoljne biblioteke...
index.html
```

Segmentacija po komponentama aplikacije

- index.html - stoji u korenskom direktorijumu i glavna svrha mu je učitavanje svih javaskript biblioteka i angular elemenata
- Assets fascikla - sadrži sve dodatne stvari koje su potrebne za aplikaciju koje nisu vezane za angular komponente
- App fascikla - sastoji se od javaskript datoteka koje definišu modul (`app.module.js`, `app.config.js`, `app.route.js`), fascikle koja objedinjuje deljene komponente (`shared` ili `common`) i fascikle koja objedinjuje ostale komponente (`components`)
- Shared fascikla - najčešće sadrži direktive i servise koji se koriste na više mesta u aplikaciji, kao i servise i kontrolere tih komponenti
- Components fascikla - sadrži segmente aplikacije, odnosno stranice ili delove stranica (sa propratnim kontrolerom i drugim angular komponentama)
- Preporuka je da svaka komponenta ima svoj angular modul
- Jedna preporuka za imenovanje modula (ne datoteka) je:
  - <ime glavnog modula>.<ime modula komponente>

primer 3

LIFT princip

- Locate - lociranje koda treba da bude intuitivno, jednostavno i brzo
- Identify - pogledom u datoteku treba da se zna šta sadrži i šta predstavlja
- Flat - iako treba podeliti datoteke po fasciklama treba voditi računa o dubini fascikla i težiti ka što manjoj dubini
- Try to stay DRY (Don’t Repeat Yourself) - izbegavati ponavljanje informacije (npr. ako je jasno iz konteksta da je articles.html view komponenta nema potrebe nazivati je articles-view.html)

Saveti za strukturiranje projekta

- Konvencije:
  - Svaka veća komponenta (ono što smo zvali mini aplikacije) treba da ima svoj modul, dok `app` modul treba idealno da sadrži samo logiku za povezivanje cele aplikacije
  - Svaka komponenta koja se može koristiti na više mesta treba da ima svoj modul (upravljanje greškama, logovanje, sigurnost, itd.)
  - Ukoliko jedan modul postane previše velik razmotriti deljenje modula na više delova
  - Svaki projekat treba da sadrži `core` ili `layout` fasciklu u `components` fascikli, u kom će se nalaziti delovi stranica zajednički za celu aplikaciju, poput zaglavlja, footer-a, navigacione trake i slično

Spoljne javaskript datoteke

- Kako koristiti spoljnu javaskript datoteku:
  - Pronaći na internetu .js datoteku koja nam treba, vodeći računa o verziji koja nam treba
  - Preuzeti datoteku i smestiti je u, na primer `assets/js`.
  - Dodati <script> tag u index.html koji pokazuje na datu datoteku
- Ukoliko je potrebno promeniti verziju javaskript datoteke koja se koristi potrebno je ponoviti proces naveden iznad
- Alternative:
  - CDN (content delivery network)
  - Bower
  - Jam, Component, Ender, Volo,...

Content delivery network

- Otklanja brigu o stranim .js datotekama tako što se navede u index.html nešto slično ovom:

```
<script src='https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.5/angular.js'></script>
```

- Većina stranih javaskript datoteka koje bi želeli da koristimo se mogu preuzeti sa servisa čija svrha je distribucija tih datoteka
- Olakšava rad jer ne moramo brinuti o tim datotekama u našoj strukturi projekta
- Ubrzava rad aplikacije jer se strane datoteke preuzimaju sa drugog servera
- Stvara zavisnost rada aplikacije od rada stranog servera
- Pošto se strane javaskript datoteke nalaze van projekta nije moguće manipulisati tim datotekema u procesu izgradnje projekta (više kasnije)

Bower

- Bower - sistem za kontrolu javaskript biblioteka koje naša aplikacija koristi
- Instalira se koristeći npm (node package manager) sa sledećom komandom u terminalu:

```
npm install bower -g
```

- Ako želimo da instaliramo neki konkretan modul, potrebno je otići na `http://bower.io/search/` ili na drugi način saznati kako je modul prijavljen kod Bower-a i u terminalu ukucati:

```
bower install <ime modula>
```

- Prethodna komanda stvara `bower_components` fasciklu (može se promeniti ime u `.bowerrc` datoteci) u tekućem direktorijumu i u nju smešta preuzeti modul, kao i sve zavisnosti koje dati modul ima

primer 4

bower.json

- Kako Bower zna koje zavisnosti neki modul ima?
- bower.json
  - ime modula
  - opis modula
  - ključne reči
  - glavna javaskript datoteka modula
  - lista zavisnosti
- Komanda `bower install` bez parametra za ime modula traži `bower.json` u tekućem direktorijumu i, ukoliko ga pronađe, preuzima sve module definisane u listi zavisnosti, kao i zavisnosti svakog od tih modula

Bower lista zavisnosti

- Postoje dve liste zavisnosti:
  - dependencies - zavisnosti koje se uvek koriste
  - devDependencies - zavisnosti koje su potrebne samo tokom proizvodnje aplikacije, ali ne i nakon što je ona proizvedena (alati za testiranje, izgradnju dokumentacije i sl.)
- Sintaksa za navođenje zavisnosti u listu:

```
"<ime modula>": "<verzija modula>"
```

- Verzija se sastoji od 3 broja (major, minor, patch), gde su drastične razlike između istog modula sa verzijama koje se razlikuju po prvom broju, a gotovo neprimetne između verzija koje se razlikuju po trećem broju
- Verziju modula je moguće navesti na više načina (semver jezik)

primer 5

Bower

- Jednostavno preuzimanje novih modula (`bower install`)
- Jednostavno ažuriranje postojećih modula (`bower update`)
- Jednostavna distribucija svih zavisnih modula (potrebno je da se drugoj strani prosledi samo `bower.json`, a ne sve datoteke)
- Potrebno je ručno navesti sve skripte koje se koriste (u npr. `index.html`), ali kako su javaskript datoteke fizički prisutne moguće ih je obrađivati u, na primer, procesu izgradnje projekta
- Kreiranje i inicijalizacija `bower.json` može preko `bower init` komande

Upravljanje procesom izgradnje projekta

- Angular aplikacije su SPA aplikacije, što podrazumeva da su sve javaskript datoteke učitane na klijentu od starta
- Ako bi sve javaskript datoteke bile spojene u jedan dokument pre puštanja aplikacije u produkciju to bi ubrzalo rad web čitača i olakšalo posao programera (index.html treba da sadrži jedan script tag)
- Ako bi se tako agregiran javaskript dokument minifikovao to bi drastično smanjilo veličinu te datoteke i ubrzao rad web čitača
- Ručno spajati javaskript datoteke je težak posao, a ručno minifikovati je nemoguće
- Treba alat koji će voditi računa o prevođenju projekta iz oblika koji je pogodan programeru za rad u oblik koji daje najbolje performanse aplikaciji

Gulp

- Alat za izgradnju javaskript projekat i automatizaciju određenih zadataka (taskova):
  - Agregiranje i minifikacija javaskript i css datoteka
  - Osvežavanje web čitača kada se desi izmena neke datoteke
  - Pokretanje unit testova
  - Pokretanje analizatora koda
  - Kompajliranje Less/Sass u CSS
  - Kopiranje izmenjenih datoteka u određen direktorijum

Gulp instalacija

- Instaliramo gulp putem npm-a sa komandom:

```
npm install -g gulp
npm install gulp --save-dev
```

- `--save` i `--save-dev` dodaje u `package.json` (koji se nalazi u tekućem direktorijumu) informaciju da dati projekat koristi naveden modul
- `package.json` je za npm ono što je `bower.json` za bower
- Kad se npm paket instalira lokalno (bez `-g`) smešta se u `node_modules`
- Treba izbegavati previše alata instalirati globalno sa `-g` već vezati npm zavisnosti za projekat (izuzetak može biti bower, pa i gulp)
- Kada preuzimamo strani projekat ukoliko sadrži `package.json` potrebno je instalirati sve zavisnosti sa `npm install` komandom, slično kao i za bower

Gulp API i gulpfile.js

- https://github.com/gulpjs/gulp/blob/master/docs/API.md
- U osnovi gulp sadrži 4 funkcije:
  - `gulp.task` - definiše task
  - `gulp.watch` - osmatra navedene datoteke na promenu
  - `gulp.src` - učitava navedene datoteke
  - `gulp.dest` - šalje učitane (i potencijalno transformisane) datoteke određen direktorijum
- `gulpfile.js` predstavlja manifest u kom definišemo taskove
- Da bi pokrenuli task potrebno je da izvršimo komandu `gulp <ime taska>` u direktorijumu u kom se nalazi `gulpfile.js`
- Task ima ime, opcioni niz taskova koji treba da se izvrše pre pokretanja datog taska i funkciju koja opisuje šta task radi

primer 6

gulp.watch

- Većina funkcija gulp alata i njegovih dodataka radi sa datotekama i direktno ih transformiše kroz cev (`.pipe` funkcija)
- `gulp.watch` funkcija posmatra određene datoteke na promenu i ukoliko se promena desi aktivira izvršavanje određenih taskova
- Struktura funkcije:

```
gulp.watch(<posmatrane datoteke>, <taskovi koji se izvršavaju na promenu>)
```

primer 7

Gulp dodaci

- Postoje razni dodaci za gulp koji proširuju funkcionalnost:
  - `gulp-clean` - omogućava brisanje sadržaja odabranog direktorijuma
  - `gulp-concat` - spaja navedene datoteke u jednu
  - `gulp-uglify` - minifikuje navedene datoteke
  - `gulp-rename` - preimenuje datoteku
  - `gulp-gzip` - vrši kompresiju datoteka
  - `gulp-sass` - kompajlira scss u css
  - `gulp-jshint` - otkriva greške i probleme u javaskript kodu
  - `gulp-webserver` - http server sa automatskim osvežavanjem
- Svaki od ovih dodataka se instalira sa `npm install <ime dodatka> --save-dev` u korenskom direktorijumu projekta (odakle se poziva gulp)

primer 8

Napomene

- Postoji alternativa za Gulp zvana Grunt koja radi slične stvari
- Pravljenje dobrog procesa izgradnje projekta može bitno da utiče na efikasnost razvoja same aplikacije
- Voditi računa da prilikom minifikacije angular kod može da prestane da radi ako zavisne komponente nisu dobro navedene (bitan je redosled)

```
/* izbegavati */
angular
    .module('app')
    .controller('AppController', AppController)

function AppController($scope, $location) {
    …
}
```

```
/* preporučeno */
angular
    .module('app')
    .controller('AppController', AppController)

AppController.$inject = ['$scope', '$location'];
function AppController($scope, $location) {
    …
}
```

primer 9