Klijentske web-aplikacije i Angular

Grđan, Barbara

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Barbara Grđan

KLIJENTSKE WEB-APLIKACIJE I ANGULAR

Diplomski rad

Voditelj rada:
doc. dr. sc. Zvonimir Bujanović

Zagreb, rujan 2017.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Hvala mami, tati, bratu i zaručniku na podršci i strpljenju.

Sadržaj

Uvod

U svijetu web-aplikacija sve je popularniji trend pomicanja aplikacijske logike na klijentsku stranu, uz web-stranice koje simuliraju klasične aplikacije sa stolnih računala. Kod takvih web-aplikacija, sav kod se dohvaća jednim zahtjevom prema web-serveru, a dodatni resursi se na stranicu učitavaju i dodaju dinamički, obično kao reakcija na akcije korisnika. Ovakav pristup rezultira znatno složenijim kodom na klijentskoj strani web-aplikacije, pa su se pojavili brojni razvojni okviri koji olakšavaju njezinu izradu. Jedan od najpopularnijih je Angular, koji kroz koncepte komponenti i modula daje elegantnu tehniku za složene klijentske aplikacije.

Cilj ovog diplomskog rada je opisati platformu Angular i istaknuti razliku u dizajnu web-aplikacija koje prenose aplikacijsku logiku na klijentsku stranu u odnosu na klasične web-aplikacije. Uz sami Angular, upoznat ćemo se sa programskim jezikom TypeScript, jezikom u kojem je Angular napisan, te ćemo istaknuti neke razlike između njega i jezika od kojeg je nastao i u kojeg se prevodi – JavaScript-a.

Mogućnosti Angular-a demonstrirat će se izradom web-aplikacije za prikaz vremenske prognoze. Aplikacija će sa servera dohvaćati podatke o vremenskim prognozama koji su prethodno dohvaćeni sa više različitih web servisa, a zatim ih uspoređivati međusobno i sa stvarnim podacima o vremenu, te na kraju rezultate prikazivati u obliku grafa.

Poglavlje 1

Uvod u Angular

1.1 Angular

Angular je razvojni okvir (engl. *framework*) namijenjen razvoju klijentskih web aplikacija. Pisan je u programskom jeziku TypeScript, otvorenog je koda (engl. *open source*), a razvio ga je Google. Zadnja službena verzija Angular-a je verzija 4.1.x.

Za izradu aplikacija koriste se HTML i TypeScript. Angular aplikacije pišu se komponirajući HTML predloške, pišući klase za komponente koje upravljaju tim predlošcima, dodajući logiku aplikacije u usluge i pakirajući komponente i usluge u module. Aplikacija se pokreće tako da se pokrene njezin glavni modul. Angular zatim prezentira sadržaj aplikacije u internet pregledniku i reagira na interakcije korisnika prateći instrukcije koje su mu dane.

1.2 Povijest

Prva verzija Angular-a bila je AngularJS. AngularJS dostupan je od 2010. godine, a zadnja verzija iz 2017. je verzija 1.6.x. On je pisan u JavaScript-u, a mnoge svjetski poznate web stranice ga koriste i danas, na primjer Paypal, Netflix, The Guardian i druge. Detaljnije o AngularJS-u možemo pronaći na [9].

Angular 2.0 objavljen je u drugoj polovici 2014. godine. Angular 2.0 je napisan (prepisan) iz početka, koristeći TypeScript. AngularJS i Angular 2.0 nisu kompatibilni, pa se zato pod nazivom Angular podrazumijeva svaka verzija Angular-a veća od 2.0. Na taj način izbjegavaju se zabune i jasnije se razlikuje AngularJS od Angular-a. Alpha i beta verzije Angular-a 2.0 objavljivane su kroz 2015. godinu, a u rujnu 2016. godine objavljena je prva službena verzija. Službena web stranica Angular-a je: `https://angular.io/`.

Već krajem 2016. godine najavljen je Angular 4. Verzija 3 je preskočena, kako bi se izbjegli nesporazumi povezani s nekim paketima čija verzija je već bila objavljena kao

v3.3.0. Posljednja dostupna verzija Angular-a je 4.1.0, objavljena u travnju 2017. godine.

Iako se skok sa verzije 2 na verziju 4 čini velik, u Angular-u 4.0 nema tako drastičnih promjena kao što je bilo u Angular-u 2.0, u odnosu na AngularJS. Politika verzioniranja je takva da, zbog brzine izlaska novih verzija, svaku novu verziju Angular-a (4.0, 5.0, ...) zapravo možemo gledati kao podverzije (1.4, 1.5, ...).

Izlazak Angular verzije 5.0 planira se u rujnu 2017. godine, a verzije 6.0 i 7.0 već su najavljene za ožujak i rujan 2018. godine. Bitno je napomenuti da su sve verzije Angular-a veće od 2.0 kompatibilne s prethodnim verzijama (do verzije 2.0), a taj princip planira se slijediti i dalje.

U ovom radu proučavat ćemo izradu web-aplikacija koristeći verziju Angular-a 2.0.

1.3 Prednosti

Web aplikacije

Angular koristi mogućnosti modernih web platformi kako bi web stranice prikazao u obliku aplikacija, a za razvoj samih aplikacija nije potreban pristup mreži, dok se instalacija potrebnog softvera obavlja u jednom koraku.

Generiranje programskog koda

Angular iz predložaka generira programski kod koji je optimiziran za današnje JavaScript virtualne strojeve. Na taj način dobivaju se sve prednosti ručno pisanog koda, uz visoku produktivnost koju pruža razvojni okvir.

Univerzalan

Angular aplikacije pokreću se na poslužiteljima node.js, .NET, PHP i drugim, a iscrtavaju se u web-pregledniku koristeći samo HTML i CSS.

Dijeljenje koda

Aplikacije u Angularu koriste automatsko dijeljenje programskog koda, što omogućava korisniku da učita samo one dijelove koda koji su mu potrebni za iscrtavanje trenutnog dijela aplikacije.

Razvojne okoline

Razvoj aplikacije u nekoj od popularnih razvojnih okolina (WebStorm, Visual Studio ili Angular IDE) donosi prednosti kao što su prijedlozi programskog koda, trenutni prikaz greški i ostale povratne informacije.

Predlošci

Brzo stvaranje dijelova korisničkog sučelja koristeći jednostavnu, ali moćnu sintaksu predložaka.

1.4 TypeScript

TypeScript je besplatan jezik otvorenog koda, razvijen i održavan od strane Microsoft-a. Sintaksa mu je strogi nadskup JavaScript-a. TypeScript podržava pisanje programskog koda u JavaScript-u, korištenje biblioteka za JavaScript i pozivanje TypeScript koda iz samog JavaScript-a. Drugim riječima, svaki JavaScript program je zapravo i TypeScript program. TypeScript se pomoću specijalnog programa, tzv. *transpiler-a* prevodi u čisti i jednostavan JavaScript programski kod koji se lako pokreće u bilo kojem internet pregledniku i na bilo kojem JavaScript pokretaču (engl. *engine*) koji podržava standard ECMAScript 3 ili noviji. TypeScript se koristi za razvoj klijentskih JavaScript aplikacija, ali i serverskih (Node.js) aplikacija.

Glavne prednosti programskog jezika TypeScript su to što je objektno orijentiran i to što su tipovi svih varijabli poznati u vrijeme prevođenja. Na taj način prevoditelj (engl. *transpiler*) može otkriti velik broj trivijalnih pogrešaka već u najranijim fazama razvoja. Ova svojstva olakšavaju razvoj srednjih i velikih aplikacija. S obzirom da je JavaScript podskup TypeScript-a, programeri dobivaju dodatne mogućnosti kombiniranjem elemenata ta dva programska jezika. TypeScript također donosi mnoge mogućnosti standarda ECMAScript, kao što su lambda funkcije, moduli i klase.

Promotrimo jedan jednostavan primjer koda u TypeScript-u, kako bismo ustanovili razlike u odnosu na JavaScript:

```
1 class Person {
2     name: string;
3     constructor (message: string) {
4         this.name = message;
5     }
6     printName() {
7         return "Hello, " + this.name;
8     }
9 }
```

Ovdje možemo vidjeti deklaraciju klase `Person`, te eksplicitno navođenje tipa varijabli `name` i `message`. Deklaracije funkcija ne zahtijevaju ključnu riječ `function`. Odgovarajući JavaScript kod:

```
var Person = (function () {
    function Person(message) {
        this.name = message;
    }
    Person.prototype.printName = function () {
        return "Hello, " + this.name;
    };
    return Person;
})();
```

Uvođenjem klasa u TypeScript došlo je do potrebe za dopunjavanjem i mijenjanjem postojećih JavaScript klasa (engl. *class*) i članova klasa novim mogućnostima. Upravo to su omogućili dekoratori (engl. *decorator*). Dekorator je poseban oblik deklaracije koji se može pridružiti deklaraciji klase, metode, funkcije za pristup svojstvu (engl. *accessor*), svojstva (engl. *property*) ili parametra. Dekoratori imaju oblik `@izraz`, gdje je `izraz` funkcija koja će se pozvati tijekom izvođenja aplikacije. Dekoratori primjenjeni na različite deklaracije unutar klase se primjenjuju sljedećim redoslijedom:

1. dekoratori parametara, metoda, funkcija za pristup svojstvu i svojstava za članove instance (engl. *instance member*);
2. dekoratori parametara, metoda, funkcija za pristup svojstvu i svojstava za statičke članove;
3. dekoratori parametara za konstruktor;
4. dekoratori klase su primijenjeni na klase.

Dekorator svojstva

Dekorator svojstva deklarira se neposredno prije deklaracije svojstva. Izraz dekoratora pozvat će se kao funkcija, sa dva argumenta:

1. konstruktor klase za statične članove, ili prototip klase za članove instance;
2. ime člana.

```
function ReadOnly(target: any, key: string) {
  Object.defineProperty(target, key, { writable: false });
}

```

```
5  class Test {
6    @ReadOnly
7    name: string;
8  }
9
10 let t = new Test();
11 t.name = 'myName';
12 console.log(t.name); // 'undefined'
```

U primjeru iznad, varijabli `name` smo pridružili dekorator `ReadOnly` i na taj način onemogućili pridruživanje nove vrijednosti toj varijabli, tj. ispisat će se `undefined`.

Dekorator klase

Dekorator klase deklarira se neposredno prije deklaracije klase. Dekorator klase primjenjuje se na konstruktor klase, a koristi se za proučavanje, modificiranje ili zamjenu definicije klase. Izraz dekoratora klase će biti pozvan s konstruktorom dekorirane klase kao jedinim parametrom. Ako dekorator klase vrati vrijednost, ona će zamijeniti deklaraciju klase s pridruženim konstuktorom. Brojne primjere dekoratora klase vidjet ćemo nešto kasnije, u radu s Angular-om.

Dekorator metode

Dekorator metode deklarira se neposredno prije deklaracije metode. Primjenjuje se na opis svojstva (engl. *property descriptor*) metode. On se koristi za proučavanje, modificiranje ili zamjenu definicije metode. Izraz dekoratora metode pozvat će se kao funkcija, s tri argumenta:

1. konstruktor klase za statične članove, ili prototip klase za članove instance;
2. ime metode;
3. novi opis svojstva metode.

Dekorator funkcije za pristup svojstvu

Dekorator funkcije za pristup svojstvu deklarira se neposredno prije deklaracije funkcije za pristup svojstvu. Primjenjuje se na opis svojstva funkcije za pristup svojstvu, te prati, modificira ili zamjenjuje definiciju funkcije za pristup svojstvu. Izraz dekoratora pozvat će se kao funkcija, sa tri argumenta:

1. konstruktor klase za statične članove, ili prototip klase za članove instance;

2. ime člana;

3. novi opis svojstva člana.

Ne možemo dekorirati i get i set funkcije za pristup svojstvu istog člana. Umjesto toga, svi dekoratori članova moraju biti primjenjeni na prvu funkciju za pristup svojstvu u dokumentu.

Dekorator parametra

Dekorator parametra deklarira se neposredno prije deklaracije parametra. Primjenjuje se na konstruktor klase ili deklaraciju metode. Izraz dekoratora pozvat će se kao funkcija, sa tri argumenta:

1. konstruktor klase za statične članove, ili prototip klase za članove instance;

2. ime metode;

3. indeks parametra u listi parametara funkcije.

Povratna vrijednost dekoratora parametra se ignorira, tj. dekorator parametra može se koristiti samo za proučavanje parametra.

```
function logPosition(target: any, propertyKey: string, parameterIndex:
    number) {
  console.log(parameterIndex);
}

class TestClass {
testFunction(firstParam: string, @logPosition secondParam: boolean) {
    console.log(firstParam);
  }
}

new TestClass().testFunction('hello', false);
// output: 1 (newline) hello
```

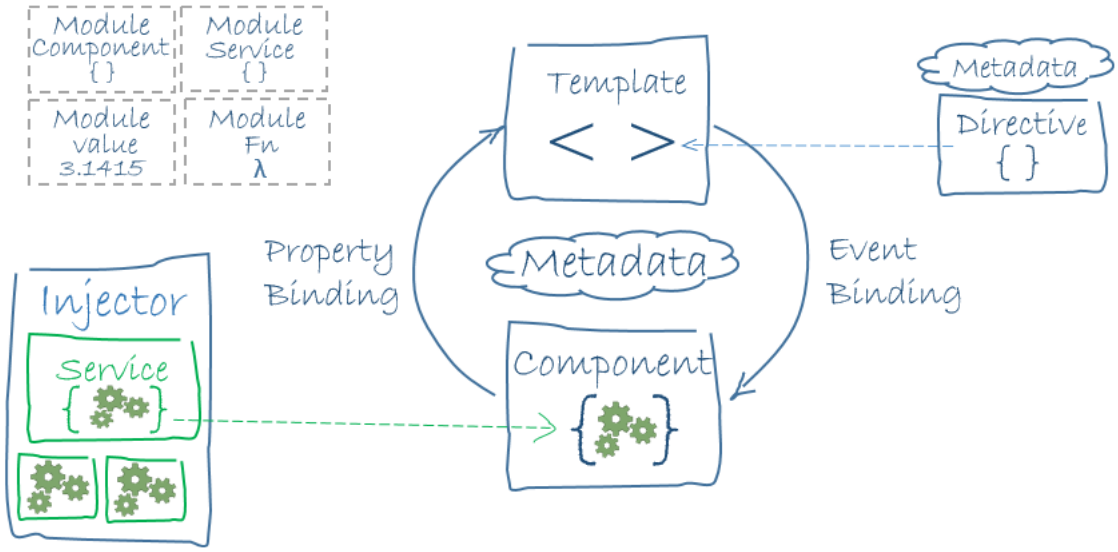
U primjeru iznad, parametru `secondParam` smo pridružili dekorator `logPosition` koji ispisuje poziciju tog parametra u listi parametara funkcije. Možemo vidjeti da se dekorator pozvao prije tijela funkcije.

Više primjera koda u TypeScript-u i dekoratora nalazi se na [8].

Poglavlje 2

Arhitektura Angular-a

U ovom poglavlju ćemo za početak samo pobrojati i definirati osnovne elemente koji čine aplikaciju u Angular-u, te neke njegove dodatne mogućnosti. U idućem poglavlju slijedi primjer osnovne aplikacije u kojem će postati jasna uloga i međudjelovanje osnovnih elemenata, dok će se ostali pojmovi objasniti u zadnjem poglavlju.



Slika 2.1: Struktura aplikacije u Angular-u

2.1 Moduli

Moduli

Aplikacije u Angular-u su modularne i Angular ima svoj sustav modula (engl. *modularity system*) koji se naziva `NgModules`. Svaka aplikacija sadrži barem jednu klasu modul (engl. *module class*) koji nazivamo korijenski modul (engl. *the root module*), u kodu obično nazvan `AppModule`. On je dovoljan samo za jako male aplikacije. Veće aplikacije s više mogućnosti sadrže veći broj modula, a svaki taj modul predstavlja blok koda posvećen jednoj značajki aplikacije.

Modul je klasa koja ima `@NgModule` dekorator. `NgModule` je dekoratorska funkcija koja uzima jedan metapodatak objekt (engl. *metadata object*) čija svojstva opisuju modul. Neki od najvažnijih svojstva kojima opisujemo module su:

- `declarations` - klase pogleda (engl. *view classes*) koje pripadaju modulu. Angular ima tri tipa takvih klasa pogleda, a to su komponente (engl. *components*), direktive (engl. *directives*) i *pipes*.
- `exports` - podskup deklaracija koje bi trebale biti vidljive i upotrebljive u predlošcima komponenata (engl. *component templates*) iz drugih modula.
- `imports` - moduli čije su izvozne klase (engl. *exported classes*) potrebne predlošcima komponenata deklariranih u ovom modulu.
- `providers` - popis usluga (engl. *services*) koje ovaj modul pruža. Te usluge postaju dostupne svim dijelovima aplikacije.
- `bootstrap` - glavni pogled aplikacije, koji se naziva korijenska komponenta (engl. *the root component*). On sadrži sve ostale poglede aplikacije. Samo glavni modul, tj. `AppModule` smije postaviti ovo svojstvo.

```
1  import { NgModule }      from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  @NgModule({
4    imports:      [ BrowserModule ],
5    providers:    [ Logger ],
6    declarations: [ AppComponent ],
7    exports:      [ AppComponent ],
8    bootstrap:    [ AppComponent ]
9  })
10 export class AppModule { }
```

Aplikacija se pokreće podizanjem (engl. *bootstrapping*) modula `AppModule`.

```
import { platformBrowserDynamic } from '@angular/platform-browser-
    dynamic';
import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule);
```

Aplikacije su najčešće namijenjene izvođenju u internet pregledniku, pa tako i ovdje pozivamo funkciju `bootstrapModule` koja se nalazi unutar modula `platformBrowserDynamic`, i kojoj kao parametar proslijeđujemo glavni modul naše aplikacije.

Moduli u JavaScript-u

JavaScript također sadrži svoj sustav modula za upravljanje kolekcijama objekata. On se u potpunosti razlikuje od sustava modula Angular-a i nije povezan s njim. U JavaScript-u, svaka datoteka je modul i svi objekti definirani u toj datoteci pripadaju tom modulu. Takav modul deklarira neke objekte javnima (engl. *public*) pomoću ključne riječi `export`, dok pomoću ključne riječi `import` modul pristupa javnim objektima drugih modula.

```
import { NgModule }      from '@angular/core';
import { AppComponent } from './app.component';

export class AppModule { }
```

Biblioteke u Angular-u

Angular dolazi kao kolekcija JavaScript modula. Možemo ih zamišljati kao biblioteke (engl. *library*) u drugim programskim jezicima. Ime svake biblioteke u Angular-u započinje prefiksom `@angular`.

```
import { Component } from '@angular/core';
```

2.2 Komponente

Komponenta upravlja nekim dijelom ekrana koji se naziva pogled. Logika komponente aplikacije definira se unutar klase. Ta klasa komunicira s pogledom pomoću svojstava i metoda. Angular automatski stvara, osvježava i uništava komponente kako se korisnik kreće kroz aplikaciju.

2.3 Predlošci

Pogled jedne komponente unutar aplikacije definira se pomoću njemu pripadnog predloška (engl. *template*). Predložak je kod napisan u HTML-u obogaćen direktivama koje signaliziraju Angularu kako prikazati komponentu.

```
<!-- app.component.html -->

<h2>Title</h2>
<p><i>Choose city:</i></p>
<ul>
    <li *ngFor="let city of cities" (click)="selectedCity(city)">
        {{city.name}}
    </li>
</ul>
<city-detail *ngIf="selectedCity" [city]="selectedCity"></city-detail>
```

Kao što vidimo u kodu, predložak je u osnovi HMTL datoteka, ali on zapravo proširuje sintaksu HTML-a Angular-ovom sintaksom. U kodu kao primjer Angular-ove sintakse imamo: `*ngFor`, {{`city.name`}}, `(click)` i <`city-detail`>. `selectedCity` je zapravo metoda unutar komponente `AppComponent`, dok je `city` objekt unutar iste komponente. Tag <`city-detail`> je posebni element koji predstavlja Angular-ovu komponentu `CityDetailComponent`.

Angular-ova sintaksa smješta se unutar HTML elemenata. To znači da se unutar jedne HTML datoteke miješaju Angular-ova i HTML sintaksa.

2.4 Metapodaci

Unutar Angular-a, pojam metapodaci (engl. *metadata*) označava podatke koji govore Angular-u kako procesirati određenu klasu. Već smo vidjeli da su komponente u Angular-u zapravo obične klase, ali iz definicije te klase nigdje se ne može vidjeti veza s Angular-om. Zapravo, svaka komponenta je samo klasa u programskog jeziku TypeScript, sve dok se Angular-u ne kaže da ona postoji. Kako bi Angular znao da je određena klasa zapravo komponenta, pridružujemo joj metapodatke.

```
@Component({
    selector:     'city-detail',
    templateUrl:  './city-detail.component.html',
    providers:  [ MyService ]
})
export class CityDetailComponent  {
 . . .
}
```

U TypeScript-u, metapodatke pridružujemo pomoću dekoratora. Ovdje vidimo dekorator `@Component`, koji klasu napisanu neposredno ispod njega označava kao Angular komponentu. Dekoratorska funkcija `@Component` kao parametar prima konfiguracijski objekt (engl. *configuration object*) s informacijama koje su Angular-u potrebne za stvaranje i prikaz komponente, te njoj pripadnog pogleda. Ovo su neke od najvažnijih konfiguracijskih opcija `@Component` dekoratora:

- `selector` - govori Angular-u da stvori i umetne instancu ove komponente na svakom mjestu unutar pripadne HTML datoteke na kojem pronađe odgovarajući tag (`<city-detail></city-detail>`).
- `templateUrl`- relativna adresa predloška ove komponente (`'./city-detail.component.html'`).
- `providers` - niz pružatelja usluga za usluge koje su potrebne ovoj komponenti. Kasnije ćemo reći više o uslugama i jednom od važnih dijelova Angular-a, koji se naziva *dependency injection*.

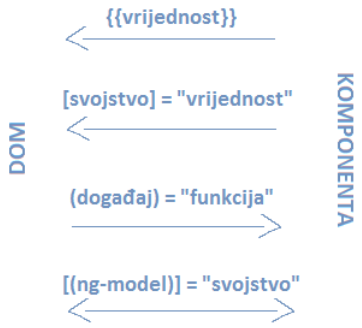
Slika 2.2: Pogled

Predložak, metapodaci i komponenta zajedno opisuju jedan pogled. Postoje razni drugi dekoratori poput `@Injectable`, `@Input` i `@Output` koji na sličan način preko metapodataka određuju ponašanje aplikacije.

2.5 Vezanje podataka

Jedna od važnih prednosti Angular-a je njegovo takozvano vezanje podataka (engl. *data binding*). Pojam vezanje podataka označava mehanizam koji koordinira dijelove predloška

s dijelovima komponente. To se postiže korištenjem posebnih oznaka unutar HTML predloška koje govore Angular-u kako ga povezati s komponentom.



Slika 2.3: Vezanje podataka

Na slici 2.3 možemo vidjeti četiri načina vezanja podataka. Svaki od njih ima različiti smjer: prema *DOM-u*, od *DOM-a* ili u oba smjera.

1. Ako u komponenti imamo člana `name`, tada u predlošku te komponente možemo pisati:

```
<p>City name: {{name}}</p>
```

   Na mjestu {{`name`}} ispisat će se vrijednost člana `name`.

2. Ako u komponenti imamo člana `isHelloVisible`, tada u predlošku te komponente možemo pisati:

```
<div [hidden]="isHelloVisible">Hello World!</div>
```

   Vidljivost ovog elementa ovisit će o vrijednosti boolean člana `isHelloVisible`.

3. Ako u komponenti imamo funkciju `onButtonClick()`, tada u predlošku te komponente možemo pisati:

```
<button (click)="onButtonClick()"> Click me </button>
```

   Ako pritisnemo ovaj element, pozvat će se funkcija `onButtonClick()` unutar odgovarajuće komponente.

4. Najvažniji od njih je dvostrano vezanje podataka (engl. *two-way data binding*), tj. zadnji primjer na slici. Kada na ovaj način povežemo podatke, svaka promjena varijable unutar komponente automatski će se vidjeti u samom pogledu aplikacije, ali i obrnuto, svaka promjena koju korisnik napravi unutar aplikacije automatski će promjeniti vrijednost varijable unutar komponente. Ako u komponenti imamo varijablu `name`, tada u predlošku te komponente možemo pisati:

```
<input [(ngModel)]="name">
```

   Svaka izmjena u `input` objektu od strane korisnika automatski će izmjeniti varijablu `name`, ali i obrnuto, svaka izmjena varijable `name` odmah će se vidjeti u `input` objektu.

Osim što ima važnu ulogu u komunikaciju između predloška i komponente, vezanje podataka također je važno za komunikaciju između komponenti roditelja i djece.

2.6 Direktive

Sljedeće bitno svojstvo predložaka u Angular-u je to što su dinamični (engl. *dynamic*). Kada ih Angular iscrtava na ekran, on mijenja elemente *DOM-a* s obzirom na instrukcije koje mu dolaze od direktiva. Direktiva je klasa definirana pomoću dekoratora `@Directive`. Zapravo, komponenta u Angular-u je spoj direktive i predloška, tj. dekorator `@Component` je zapravo `@Directive` dekorator proširen sa značajkama predloška. Dakle, iako je komponenta tehnički direktiva, komponente u Angular-u su toliko karakteristične i predstavljaju središta aplikacija, da se sa stajališta arhitekture moraju odvojiti jedna od druge.

Postoje još dvije vrste direktiva: strukturalne (engl. *structural*) i atributne (engl. *attribute*). Strukturalne direktive mijenjaju izgled stranice tako što dodaju, brišu ili zamjenjuju elemente unutar *DOM-a*.

```
<ul>
<li *ngFor="let city of cities">{{city.name}}</li>
</ul>
```

Strukturalna direktiva `*ngFor` govori Angular-u da ispiše onoliko elemenata `<li>` koliko lista `cities` ima članova. Još jedan primjer strukturalne direktive je `*ngIf`.

```
<div *ngIf="city" >{{city.name}}</div>
```

`*ngIf` ispituje izraz u navodnicima. U ovom primjeru, element `div` bit će vidljiv samo ako objekt `city` postoji.

Atributne direktive utječu na izgled ili ponašanje postojećeg elementa. One izgledaju kao obični HTML atributi. Jedan od primjera atributne direktive je direktiva `ngModel`, koja implementira dvostrano vezanje podataka. `ngModel` modificira ponašanje postojećeg

elementa, najčešće <input> elementa, tako što istovremeno prikazuje vrijednost varijable i odgovara na događaje kao što su promjena vrijednosti unutar <input> elementa. Još neke direktive su: `ngSwitch` (strukturalna), `ngStyle` i `ngClass` (atributne). Također, moguće je napisati i vlastite direktive. Više o direktivama možemo pronaći na [3].

2.7 Usluge

Usluga (engl. *service*) je široka kategorija koja obuhvaća bilo koju vrijednost, funkciju ili mogućnost koju aplikacija treba. Usluga je obično klasa s usko definiranom svrhom. Na primjer, to mogu biti: usluga logiranja, podatkovna usluga ili kalkulator poreza. U Angular-u nema posebne definicije usluge, niti nema bazne klase (engl. *base class*) za usluge, ali svejedno, usluge su neizostavan dio svake Angular aplikacije.

Usluge unutar aplikacije su najčešće korištene od strane komponenti. Cilj je da komponente budu što jednostavnije, odnosno da se uslugama prepuste svi složeniji zadaci kao što su dohvaćanje podataka sa servera ili validacija korisničkog unosa. Naime, komponenta bi trebala biti posrednik između pogleda koji se iscrtava koristeći predložak, i logike aplikacije. Dobro napisana komponenta sadrži samo varijable i metode za vezanje podataka, a sve netrivijalne zadatke prenosi uslugama. Zanimljivo je da Angular nigdje ne nameće ovakav stil razvoja aplikacije, ali bitno olakšava praćenje ovih smjernica, tj. prebacivanje logike aplikacije na usluge, te korištenje tih usluga unutar komponenti.

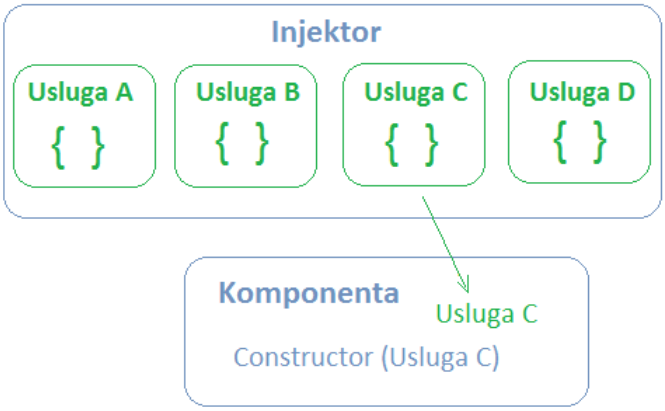
2.8 Dependency injection

U softverskom inženjerstvu, *dependency injection* je često korišteni oblikovni obrazac koji služi kako bismo novoj klasi `A` omogućili korištenje metoda iz postojeće klase `B`, bez potrebe da eksplicitno stvorimo objekt klase `B` kao člana od `A`. Jedna od tehnika implementacije ovog obrasca jest da se konstruktoru klase `A` proslijedi postojeća instanca klase `B`. Kod Angular-a, *dependency injection* se koristi kako bismo novim komponentama davali pristup uslugama koje su im potrebne. Kako bi saznao koje usluge određena komponenta treba, Angular gleda tipove parametara u konstruktoru komponente.

```
1 constructor(myService : MyService) {
2     this.cities = myService.getCityList();
3 }
```

Ova komponenta zahtjeva uslugu MyService. Kada Angular kreira komponentu, prvo od injektora (engl. *injector*) saznaje koje usluge ta komponenta zahtjeva. Injektor održava spremnik (engl. *container*) svih instanci pojedinih usluga koje su već ranije kreirane. Ako tražena usluga već nije u spremniku, injektor kreira novu instancu te usluge i dodaje ju u

spremnik. Nakon što su sve tražene usluge kreirane, Angular poziva konstruktore komponenti s traženim uslugama kao argumentima. Ovo je način na koji je u Angular-u implementiran *dependency injection*.



Slika 2.4: Injektor

Kako bi injektor znao koje usluge su mu dostupne i kako ih kreirati, potrebno je registrirati pružatelja usluge za svaku uslugu. Pružatelj usluge samo kreira ili vraća uslugu, obično instancu klase određene usluge. Najbolje je pružatelje usluge dodati u glavnom modulu, na taj način biti će dostupne svim dijelovima aplikacije. Osim toga, pružatelja usluge je moguće registrirati i samo na razini komponente, unutar svojstva `providers` metapodatka dekoratora `@Component`. Na ovaj način će se za svaku instancu komponente također stvoriti i nova instanca usluge.

Ukratko, *dependency injection* je dio Angular-a i koristi se na mnogo mjesta unutar koda web-aplikacije. Injektor je glavni mehanizam - on održava spremnik s instancama usluga koje je kreirao, a nove instance usluga stvara pomoću pružatelja usluga. Pružatelj usluge je kao recept za stvaranje usluge. Pružatelje usluge potrebno je registrirati pomoću injektora.

2.9 Usmjeravanje

Za navigaciju između pogleda unutar aplikacije zadužen je dio Angular-a koji nazivamo usmjernik (engl. *router*). Kao primjer, promotrimo navigaciju internet preglednika:

- unosom URL adrese internet preglednik otvara odabranu web stranicu;
- pritiskom na poveznicu unutar web stranice internet preglednik otvara odgovarajuću web stranicu;

- pritiskom na tipku `natrag` ili `naprijed` unutar internet preglednika, on navigira kroz povijest posjećenih web stranica.

Angular-ov usmjernik ima puno sličnosti s ovim modelom. Angular može interpretirati URL adresu internet preglednika kao instrukciju za navigaciju do određenog pogleda aplikacije. Na isti način, komponenti ciljanog pogleda mogu se proslijediti opcionalni parametri pomoću kojih će komponenta odrediti koji sadržaj prikazati u pogledu. Angular-ov usmjernik može se povezati poveznicama unutar aplikacije koji vode do odgovarajućih pogleda ako korisnik na njih klikne. Također je moguće navigirati do nekog pogleda kao reakciju na neku korisnikovu aktivnost kao što je pritisak tipke ili odabir stavke padajućeg izbornika. Navigacija između pogleda se bilježi u povijesti internet preglednika, tako da automatski rade preglednikove tipke `natrag` i `naprijed`.

Prva stvar koju je potrebno napraviti je unutar datoteke `index.html` dodati element <base> kao prvo dijete elementa <head>. Tada će usmjernik znati kako pravilno slagati navigacijske URL-ove. Najčešće je direktorij `app` u korijenskom direktoriju aplikacije i u tom slučaju <base> tag mora izgledati ovako:

```
<base href="/">
```

Nakon toga u datoteku `app.module.ts` uvozimo potrebne klase iz biblioteke `@angular/router`:

```
import { RouterModule, Routes } from '@angular/router';
```

Kada se URL adresa internet preglednika promijeni, usmjernik će potražiti odgovarajuću rutu (engl. *route*) iz koje može odrediti koju komponentu mora prikazati. Rute definiramo unutar `app.module.ts datoteke`, a u sljedećem primjeru možemo vidjeti pet različitih načina za definiranje ruta:

```
const appRoutes: Routes = [
  { path: 'city-list', component: CityListComponent },
  { path: 'city/:id', component: CityComponent },
  { path: 'cities',
    component: CitiesComponent,
    data: { title: 'Cities page' }
  },
  { path: '',
    redirectTo: '/cities',
    pathMatch: 'full'
  },
  { path: '**', component: PageNotFoundComponent }
];

@NgModule({
  imports: [
```

```
17      RouterModule.forRoot(appRoutes)
18    ],
19    ...
20 })
```

Niz `appRoutes` je niz ruta koji opisuje kako navigirati između komponenti. Taj niz proslijeđuje se metodi `RouterModule.forRoot`, koja zatim konfigurira usmjernik. Svaka ruta je zapravo preslikavanje URL putanje u komponentu. Putanje mogu biti relativne, kao u primjeru, a usmjernik će pretvoriti te putanje u apsolutne.

Token `:id` u drugoj ruti je token za parametar rute. Na primjer, ako imamo putanju `/city/13`, tada će vrijednost parametra `id` biti 13. Odgovarajuća komponenta `CityComponent` imat će pristup tom parametru.

Unutar svojstva `data` u trećoj ruti smještaju se podaci vezani za konkretnu rutu.

U četvrtoj ruti vidimo praznu putanju koja odgovara početnoj putanji aplikacije. Ta početna putanja preusmjerava se na putanju `/cities`, pa će pri otvaranju aplikacije biti prikazan `CitiesComponent`.

U zadnjoj ruti putanja ** predstavlja sve ostale putanje koje nisu obuhvaćene prethodno definiranim putanjama. Tada možemo korisnika preusmjeriti na početnu stranicu ili mu prikazati prikladnu poruku o nepostojanju stranice na toj putanji.

Kako bismo povezali usmjernik s predloškom koristimo sljedeći element:

```
1 <router-outlet></router-outlet>
```

Element postavljamo na željeno mjesto u predlošku, a usmjernik će na to mjesto ubaciti odgovarajuću komponentu. Odgovarajući pogled prikazuje se neposredno nakon prethodnog elementa. Nakon što smo implementirali rute, preostaje nam omogućiti navigaciju koja je najčešće vezana za URL poveznice ili gumbe (engl. *button*) unutar pogleda.

```
1 <nav>
2   <a routerLink="/city-list" routerLinkActive="active">Citylist</a>
3   <a routerLink="/cities" routerLinkActive="active">Cities page</a>
4 </nav>
```

U ovom primjeru možemo vidjeti dvije URL poveznice, a pritiskom na neku od njih otvara se odgovarajući pogled.

Detaljnije informacije o elementima aplikacije u Angular-u, te o dodatnim mogućnostima možemo pronaći na [10].

Poglavlje 3

Priprema okruženja i osnovna aplikacija

3.1 Instalacija

Aplikacije u Angular-u najbolje je razvijati lokalno, na vlastitom računalu. Prvi korak je instalacija paketa `node.js` i `npm`.

`Node` i `npm` paketi esencijalni su za razvoj modernih web aplikacija u Angular-u, ali i na drugim platformama. `Node` je potreban za izgradnju (engl. *build*) same aplikacije, a paket `npm` je zadužen za instalaciju dodatnih JavaScript biblioteka. `Node` i `npm` mogu se zajedno preuzeti sa: `https://docs.npmjs.com/getting-started/installing-node`.

Angular zahtjeva verziju `node-a` v4.0.0 i veću, dok je najmanja potrebna verzija `npm-a` 3.0.0. Verzije je moguće provjeriti u terminalu ili konzoli, pokretanjem naredbi `node -v` i `npm -v`.

Nakon instalacije `node-a` i `npm-a`, potrebno je napraviti sljedeće:

1. Kreirati direktorij projekta;
2. Klonirati ili preuzeti minimalni projekt sa github-a, koji se naziva `QuickStart` (`https://github.com/angular/quickstart/`);
3. Izvesti naredbu `npm install`;
4. Za pokretanje izvesti naredbu `npm start`.

```
1 git clone https://github.com/angular/quickstart.git quickstart
2 cd quickstart
3 npm install
4 npm start
```

QuickStart seed

QuickStart projekt je baza za lokalni razvoj svake nove Angular aplikacije. Sada ćemo opisati strukturu i dijelove tog projekta. Tri najbitnije TypeScript datoteke u projektu su `main.ts`, `app.module.ts` i `app.component.ts`.

Gotovo svaka aplikacija trebala bi sadržavati te tri bazne datoteke. Svaka od njih ima određenu svrhu, i mijenja se neovisno o drugima kako aplikacija raste.

Datoteke izvan direktorija `src/` su namjenjene za izgradnju, puštanje u pogon (engl. *deploy*) i testiranje aplikacije. Osim toga, tamo se nalaze konfiguracijske datoteke i vanjske biblioteke potrebne našoj aplikaciji.

Datoteke unutar direktorija `src/` pripadaju aplikaciji. Unutar tog direktorija dodajemo nove TypeScript, HTML ili CSS datoteke (obično unutar `src/app/` direktorija). Sada ćemo opisati tri datoteke koje smo naveli na početku:

- `app.component.ts` (`src/app/app.component.ts`) - ovdje definiramo baznu komponentu `AppComponent` koja će zapravo postati stablo ugnježđenih komponenti kako se aplikacija razvija.
- `app.module.ts` (`src/app/app.module.ts`) - u ovoj datoteci definiramo glavni modul aplikacije, `AppModule`. Za sada modul sadrži samo komponentu `AppComponent`.
- `main.ts` (`app/main.ts`) - pomoću datoteke `main.ts` prevodimo aplikaciju i podižemo `AppModule` koji se pokreće u internet pregledniku.

3.2 Jednostavna aplikacija

Kreirat ćemo jednostavnu aplikaciju, bez korištenja QuickStart projekta, koja na zaslon ispisuje tekst. Kroz razvoj aplikacije upoznat ćemo se s osnovnim dijelovima Angular-a. Detaljnije, pokazat ćemo:

- što su komponente;
- kako uvoziti (engl. *import*) programski kod iz drugih datoteka i koristiti ih u aplikaciji;
- kako programski kod aplikacije razdvojiti u Angular module;
- kako Angular aplikaciju prebaciti na server i pokrenuti u internet pregledniku.

Komponenta

Aplikaciju započinjemo stvaranjem datoteke `hello.component.ts`, te u njoj moramo kreirati klasu `HelloComponent`.

```
class HelloComponent {
}
```

Kao što smo naveli u prethodnom poglavlju, da bi klasa postala komponenta, moramo joj pridružiti metapodatke koristeći dekorator `@Component`.

```
@Component({
  selector: 'hello'
})
class HelloComponent {
}
```

Na ovaj način povezali smo komponentu `HelloComponent` sa HTML tagom <hello>. Kada god Angular naiđe na HTML kod oblika <hello></hello>, na tom mjestu instancira objekt klase `HelloComponent`.

Imports

Prije nego što koristimo dekorator `@Component`, moramo ga uvesti.

```
import {Component} from '@angular/core';
```

Ovako uvozimo kod za dekorator `Component` iz modula `@angular/core`.

Predložak

Kako bismo prikazali i koristili našu komponentu, moramo dodati tag <hello></hello> u HTML datoteku (`index.html`).

```
<body>
  <hello></hello>
</body>
```

Za sada se još ništa ne prikazuje na ekranu. Želimo da Angular zamijeni tagove <hello></hello> nekim HTML predloškom.

Dakle, sljedeće ćemo napraviti datoteku `hello.component.html` u kojoj ćemo napisati HTML predložak.

```
<h1> Hello World! </h1>
```

Zadnje što preostaje je povezati ovaj HTML predložak s komponentom `HelloComponent`. Za to koristimo atribut dekoratora `@Component`, ključne riječi `templateUrl`.

```
@Component({
  selector: 'hello',
  templateUrl: './hello.component.html'
})
class HelloComponent {
}
```

Moduli

Aplikacija kakvu smo implementirali do sada i dalje nije funkcionalna. Definirali smo svoju komponentu, povezali ju s predloškom i dodali tag komponente u svoju HTML datoteku, ali i dalje nismo rekli Angular-u da ga želimo koristiti na ovoj datoteci. To ćemo napraviti kreiranjem glavnog modula `AppModule`. Napravimo datoteku `app.module.ts`.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports: [BrowserModule],
  declarations: [HelloComponent],
  bootstrap: [HelloComponent]
})
export class AppModule {
}
```

Ovdje smo kreirali klasu `AppModule` kojoj smo pridružili dekorator `@NgModule`. Najbitniji atribut je `bootstrap` koji govori Angular-u da se pri pokretanju aplikacije pokreće komponenta `HelloComponent`. Atribut deklaracije sadrži listu komponenti koje pripadaju ovom modulu.

Jedino što nam preostaje je napraviti datoteku `main.ts` u kojoj pokrećemo aplikaciju pomoću modula `AppModule`.

```
import { platformBrowserDynamic } from '@angular/platform-browser-
    dynamic';
import { AppModule } from './app.module';
platformBrowserDynamic().bootstrapModule(AppModule);
```

Ovdje vidimo funkciju `platformBrowserDynamic()` koju koristimo za pokretanje aplikacije u internet pregledniku. Zadnje što je potrebno je stvoriti datoteku `index.html`. Datoteka `index.html` izgleda ovako:

```
<!DOCTYPE html>
<html>
  <head>
    <title>HelloApp</title>
    <base href="/HelloApp/src/">
    <meta charset="UTF-8">
```

```
7     <meta name="viewport" content="width=device-width, initial-scale=1">
8     <link rel="stylesheet" href="styles.css">
9
10    <!-- Polyfill(s) for older browsers -->
11    <script src="https://unpkg.com/core-js/client/shim.min.js"></script>
12    <script src="https://unpkg.com/zone.js@0.8.4?main=browser"></script>
13    <script src="https://unpkg.com/systemjs@0.19.39/dist/system.src.js
      "></script>
14    <script src="systemjs.config.js"></script>
15
16    <script>
17      System.import('main.js').catch(function(err){ console.error(err);
      });
18    </script>
19
20  </head>
21
22  <body>
23    <hello></hello>
24  </body>
25 </html>
```

Poglavlje 4

Razvoj složenije web aplikacije

U ovom poglavlju ćemo opisati jednu složeniju web-aplikaciju izrađenu pomoću Angular-a za potrebe ovog diplomskog rada. Svrha aplikacije je omogućiti laku usporedbu kvalitete vremenske prognoze koju daje nekoliko poznatih web-servisa.

Aplikacija se sastoji od dva glavna dijela: serverske i klijentske strane. Klijentska aplikacija napisana je u Angular-u, ali zbog potrebe za prikazom većeg broja mogućnosti Angular-a, napisana je i manja serverska aplikacija preko koje Angular aplikacija dohvaća podatke koje koristi.

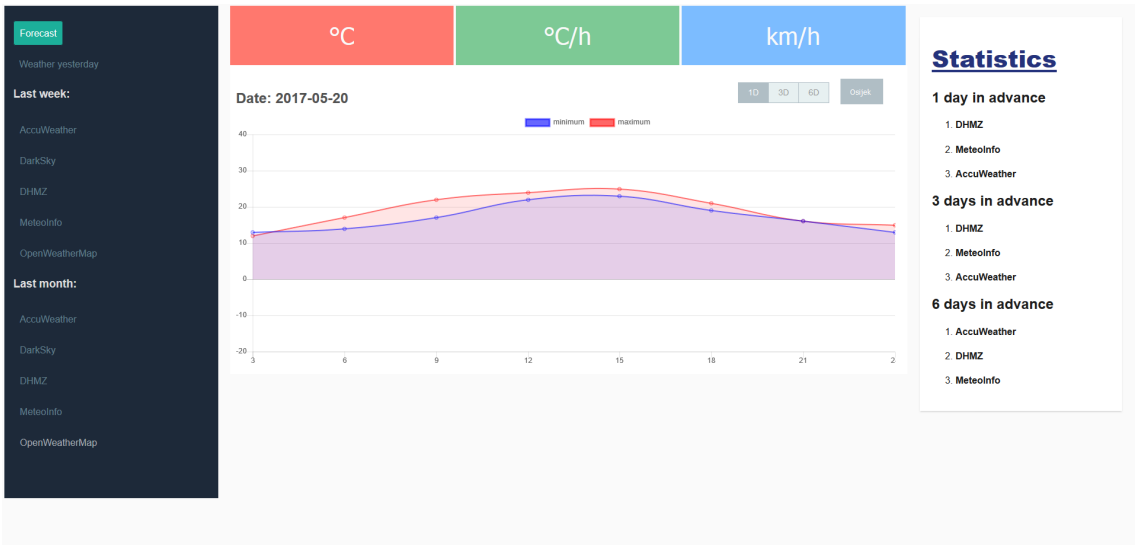
4.1 Opis web aplikacije

Klijentska aplikacija je aplikacija koja prikazuje vremensku prognozu, a korisnik može birati za koji dan i koju vrstu prognoze želi vidjeti. Prikazuje se vremenska prognoza dohvaćena s pet različitih web stranica ([1], [7], [5], [2], [4]), i moguće je međusobno usporediti te prognoze. Postoje tri grupe prikaza vremenske prognoze: vremenska prognoza za budućnost, vremenska prognoza za jučer, te vremenska prognoza kroz vremensko razdoblje u prošlosti. Također, postoje tri tipa vremenske prognoze: minimalna i maksimalna dnevna temperatura zraka, maksimalna dnevna brzina vjetra, te dnevna temperatura zraka po satima.

Vremenska prognoza za budućnost jednostavno prikazuje vremensku prognozu. Te prognoze su kratkoročne (odnose se na sutrašnji dan), srednjeročne (tri dana unaprijed) i dugoročne (šest dana unaprijed). Za svaki dan dodatno možemo birati jedan od tri tipa vremenske prognoze.

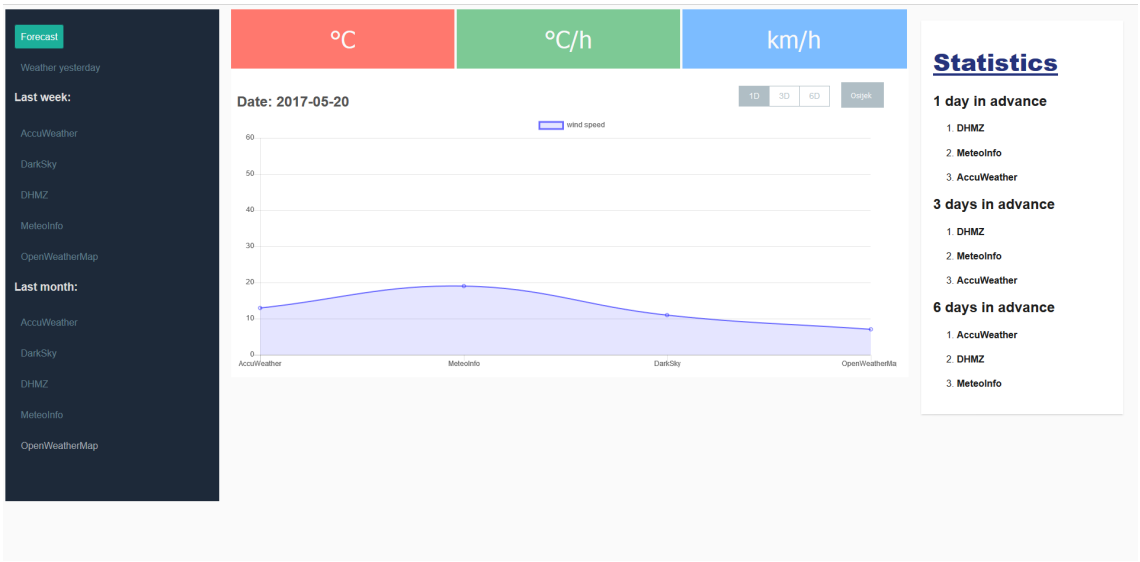
Kod prikaza vremenske prognoze za jučer, korisnik može vidjeti jučerašnju prognozu vremena zajedno sa stvarnim podacima o vremenu. Na taj način, određivanjem odstupanja izmjerene i prognozirane temperature zraka i brzine vjetre, moguće je odrediti točnost

Slika 4.1: Početni prikaz klijentske aplikacije



Slika 4.2: Dnevna temperatura zraka po satima

pojedine prognoze. Stvarni podaci o vremenu dohvaćaju se sa svakog web servisa na kraju dana, kada su ti podaci poznati. Ovdje su stvarni podaci uvijek fiksirani, ali možemo vidjeti što su za jučer web stranice prognozirale jedan, tri ili šest dana prije. Dodatno, za svaki prikaz možemo birati jedan od tri tipa vremenske prognoze.

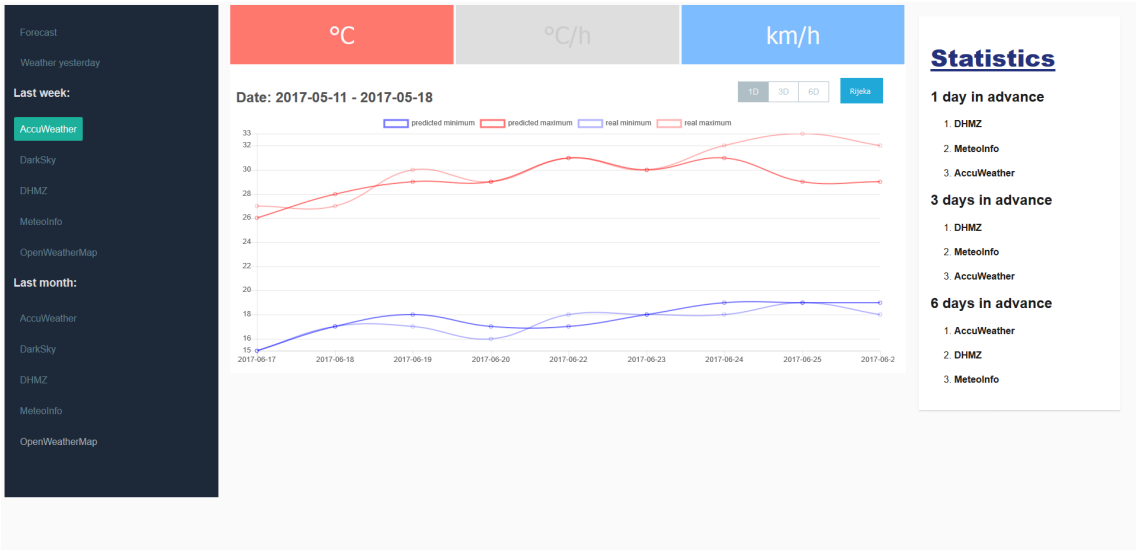


Slika 4.3: Dnevna brzina vjetra

Vremenska prognoza kroz vremensko razdoblje u prošlosti donosi nam prikaz vremenskih prognoza za prošli tjedan i za prošli mjesec, za pojedinu web stranicu. Kod ovog prikaza možemo vidjeti koliko je pojedina web stranica bila točna u svojem prognoziranju, kroz vremensko razdoblje. Za svaku web stranicu i za svaki dan u tjednu/mjesecu prikazana je predviđena vremenska prognoza i uz nju stvarni podaci o vremenu. Ovdje su stvarni podaci također fiksirani, ali možemo vidjeti što su za konkretni dan web stranice prognozirale jedan, tri ili šest dana prije. Za svaki prikaz možemo birati između dva tipa vremenske prognoze: minimalna i maksimalna dnevna temperatura zraka, ili maksimalna dnevna brzina vjetra.

Svaki od ovih prikaza vremenske prognoze dostupan je za sljedeće gradove: Zagreb, Split, Rijeka, Osijek.

U aplikaciji još možemo vidjeti statistiku o točnosti web stranica u svojoj prognozi. Statistika se odnosi na točnost u prognoziranju temperature zraka i brzine vjetra. Na temelju odstupanja vrši se i rangiranje, odnosno, može se odrediti koja web-stranica daje najbolju kratkoročnu, srednjoročnu ili dugoročnu prognozu.



Slika 4.4: Vremenska prognoza za vremensko razdoblje u prošlosti

4.2 Serverska strana aplikacije

Serverska strana napisana je u programskom jeziku PHP, a podaci o vremenskim prognozama spremaju se u bazu podataka. Postoje četiri tipa vremenske prognoze koju spremamo u bazu. To su dnevna vremenska prognoza, vremenska prognoza po satima u danu, stvarni današnji podaci o vremenu i stvarni današnji podaci o vremenu po satu. Svaki od tih podataka preuzima se sa svake web stranice za prognozu vremena svakodnevno, za svaki od četiri grada.

Baza podataka

Baza podataka sastoji se od četiri tablice koje odgovaraju tipovima prognoze. To su: `Forecast`, `TodayForecast`, `HourlyForecast` i `CurrentHourlyForecast`.

Tablice `Forecast` i `HourlyForecast` odgovaraju podacima o dnevnoj vremenskoj prognozi i podacima o vremenskoj prognozi po satima u danu. Tablice `TodayForecast` i `CurrentHourlyForecast` odgovaraju stvarnim podacima o vremenu u nekom danu i o vremenu po satu u danu.

Dio atributa u sve četiri tablice je isti, a to su `webpage` (web stranica od koje preuzimamo podatke), `date` (datum vremenske prognoze) i `city` (grad za koji je vremenska prognoza). U tablici `Forecast` dodatno imamo atribute `minTemp` i `maxTemp` (minimalna i maksimalna dnevna temperatura), `daysBefore` (koliko dana ranije je preuzeta prognoza za taj dan),

`rain` (predviđena količina kiše ili vjerojatnost da će pasti kiša), `isRainProbability` (označava da li atribut `rain` sadrži količinu kiše ili vjerojatnost da će pasti kiša) i `wind` (brzina vjetra). U tablici `TodayForecast` imamo sve atribute iste kao u tablici `Forecast`, osim atributa `daysBefore`. Taj atribut nije nam potreban, jer stvarne podatke o vremenu za neki dan preuzimamo točno na taj dan. Tablica `HourlyForecast` ima zajedničke atribute `webpage`, `date` i `city`, te dodatne atribute `temp` (temperatura u tom satu) i `hour` (sat u danu za koji preuzimamo temperaturu). Tablica `CurrentHourlyForecast` ima sve atribute iste kao i tablica `HourlyForecast`.
Za spremanje podataka korištena je MySQL baza podataka. Za spajanje na bazu i dohvaćanje podataka iz nje korišteno je sučelje PDO unutar programskog jezika PHP.

Ostale datoteke

Skripta `save.php` sadrži neke pomoćne funkcije za spremanje podataka u bazu podataka. Funkcije `save()`, `saveToday()`, `saveHourly()` i `saveCurrentHourly()` koristimo za unos podataka redom u tablice `Forecast`, `TodayForecast`, `HourlyForecast` i `CurrentHourlyForecast`. Osim ovih funkcija, datoteka sadrži još neke funkcije koje vraćaju određene datume u formatu koji je pogodan za spremanje u bazu podataka.

Za preuzimanje dnevne vremenske prognoze koristimo skripte `zagreb.php`, `split.php`, `rijeka.php` i `osijek.php`. U svakoj od njih preuzimamo podatke za taj grad, sa svake od pet web stranica koje pratimo. Podatke preuzimamo za jedan, tri i šest dana unaprijed. Dodatno preuzimamo podatke o vremenskoj prognozi po satu s dvije web stranice na kojima su ti podaci dostupni. Tri od pet stranica koje pratimo imaju dostupan besplatan API preko kojeg se dohvaćaju podaci. S ostale dvije stranice podaci se preuzimaju parsiranjem izvornog HTML koda.

Stvarni podaci o vremenu u nekom danu preuzimaju se pomoću skripte `saveToday.php`. Za svaku web stranicu i za svaki grad u njoj postoji funkcija koja preuzima podatke za tu web stranicu i za taj grad, te ih posprema u bazu podataka u tablicu `TodayForecast`. Kao i u prethodnoj datoteci, podaci s nekih web stranica dohvaćaju se preko API-ja web stranice, a neki se parsiraju iz izvornog HTML koda web stranice. Stvarni podaci o vremenu po satu u nekom danu preuzimaju se pomoću skripte `saveCurrentHour.php`. Ovdje postoje samo dvije web stranice koje imaju dostupne podatke, pa se za njih i za svaki grad podaci pospremaju u bazu podataka u tablicu `CurrentHourlyForecast`.

Skripte navedene iznad automatski se pozivaju koristeći uslugu `cron` svakog dana u određeni sat. One nam služe za dohvaćanje podataka sa web-servisa i pospremanje u bazu podataka. Sada ćemo navesti ostale skripte koje se pozivaju kada netko koristi aplikaciju. One nam služe za dohvaćanje podataka o vremenskim prognozama sa servera.

Skripta `get.php` dohvaća sve podatke o vremenskoj prognozi za određeni datum u određenom gradu. Parametri `datum` i `grad` su proslijeđeni preko `GET` zahtjeva toj skripti. Ako želimo dohvatiti stvarne podatke o vremenu na taj dan (u prošlosti), tada parametar `today` postavimo na `true`. U suprotnom, ako želimo podatke o vremenskoj prognozi za taj dan, tada parametar `today` postavimo na `false`. U prvom slučaju, podaci se dohvaćaju iz tablice `TodayForecast`, dok se u drugom slučaju podaci dohvaćaju iz tablice `Forecast`.

Skripte `getHourly.php` dohvaća sve podatke o vremenskoj prognozi po satu za određeni datum u određenom gradu. Parametri i postupak dohvaćanja je identičan kao u datoteci `get.php`, samo što se u ovom slučaju podaci dohvaćaju iz tablica `CurrentHourlyForecast` i `HourlyForecast` (ovisno o parametru `today`).

Skripte `getLastWeek.php` i `getLastMonth.php` dohvaćaju sve podatke o vremenskoj prognozi u određenom gradu unutar određenog vremenskog okvira, tj. u zadnjih tjedan dana ili zadnjih mjesec dana, ovisno o datoteci. Datoteka prima parametar `city` koji označava grad i parametar `today` koji označava želimo li stvarne podatke ili podatke o vremenskoj prognozi. Ovisno o parametru `today` dohvaćaju se podaci iz tablice `TodayForecast`, odnosno tablice `Forecast`.

Skripta `getLastTwoWeeks.php` dohvaća sve podatke o vremenskoj prognozi u zadnja dva tjedna. Ovi podaci služe nam za statistiku o točnosti web stranice.

Skripte `getCityList.php` i `getWebpageList.php` vraćaju nam redom popis svih gradova i popis svih web stranica za koje imamo podatke u bazi podataka.

4.3 Klijentska strana aplikacije

Početna datoteka koja se učitava otvaranjem web stranice je `index.html`. U njoj prvo navodimo biblioteke koje su potrebne za pokretanje aplikacije napisane u Angular-u:

```
<script src="node_modules/core-js/client/shim.min.js"></script>
<script src="node_modules/zone.js/dist/zone.js"></script>
<script src="node_modules/systemjs/dist/system.src.js"></script>
<script src="systemjs.config.js"></script>
```

Dodatno navodimo biblioteku za crtanje grafova ([6]) koja je potrebna za prikaz vremenske prognoze.

```
<script src="node_modules/chart.js/dist/Chart.bundle.min.js"></script>
```

Zatim učitavamo datoteke koje koristi naša aplikacija:

```
<script>
      System.import('main.js').catch(function(err){ console.error(err);
   });
</script>
```

Na samom kraju, unutar elementa <body> postavljamo glavni element koji predstavlja našu aplikaciju:

```
<body>
  <my-app>Loading ... </my-app>
</body>
```

`main.js` je jednostavna JavaScript datoteka unutar koje navodimo glavni modul aplikacije, tj. `AppModule`.

`AppModule` je definiran unutar datoteke `app.module.ts`. U njemu smo pobrojali sve dodatne module i komponente koje koristimo unutar aplikacije. Niz `imports` je niz modula koje aplikacija koristi. Niz `declarations` je niz komponenti od kojih se sastoji naša aplikacija.

```
declarations: [ AppComponent, ChartTypeComponent, StatisticsComponent,
    AppMenuComponent, ForecastChartComponent, YesterdayChartComponent,
    HistoryChartComponent ]
```

Unutar niza `providers` nalazi se samo `ForecastService`, usluga koja će nam dobavljati sve podatke o vremenskim prognozama. Na kraju navodimo glavnu komponentu koja predstavlja aplikaciju, tj. `AppComponent`.

Komponente

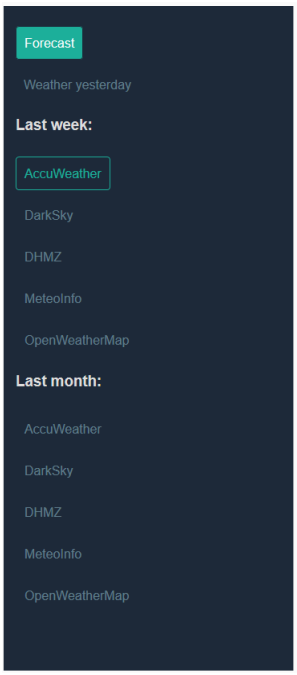
Komponenta `AppComponent` je deklarirana unutar datoteke `app.component.ts`. Ta komponenta nema članskih varijabli niti funkcija, već nam služi za iscrtavanje glavnog prikaza koji sadrži ostale komponente naše aplikacije. Kao predložak koristi se datoteka `app.component.html`. U njoj možemo vidjeti podjelu aplikacije na tri dijela: navigacija aplikacije, glavni dio aplikacije s grafovima vremenskih prognoza, i statistika o točnosti.

```
<div class="menuLeft">
  <app-menu></app-menu>
</div>

<div class="mainChartMiddle">
  <router-outlet></router-outlet>
</div>

<div class="statisticsRight">
  <statistics></statistics>
</div>
```

Element <app-menu> predstavlja komponentu `AppMenuComponent` definiranu u datoteci `app-menu.component.ts`. Ova komponenta služi nam za navigaciju kroz aplikaciju. Pomoću nje odabiremo vrstu vremenske prognoze i web stranicu koju želimo analizirati u povijesnom prikazu vremenske prognoze.



Slika 4.5: Navigacija

Komponenta izgleda ovako:

```
@Component({
  moduleId:module.id,
  selector: 'app-menu',
  templateUrl: './app-menu.component.html',
  styleUrls: ['./app-menu.component.css']
})
export class AppMenuComponent {
  webpageList: string[];
  constructor(private forecastService: ForecastService) {}
  ngOnInit(): void {
    this.forecastService.getWebpageList().then(webpageList => this.
    webpageList = webpageList);
  }
}
```

Ovdje možemo vidjeti primjer korištenja usluge `ForecastService`. Za prikaz navigacije potrebna nam je lista web stranica koje možemo odabrati u povijesnom prikazu vremenske prognoze. Tu listu dohvatit ćemo pomoću usluge `ForecastService`. Uslugu navodimo u konstruktoru komponente, kako bi Angular-ov injektor znao da je ovoj komponenti potrebna ta usluga. Funkcija `ngOnInit` automatski se poziva u trenutku kada

je komponenta inicijalizirana i spremna za korištenje. Unutar te funkcije stvaramo poziv instanci usluge `ForecastService`, koristeći njezinu funkciju `getWebpageList`. Kada je usluga spremna i traženi podaci su joj dostupni, tada se jednostavnom lambda funkcijom lista web stranica posprema u člansku varijablu, točnije niz `webpageList`. Uslugu `ForecastService` detaljnije ćemo opisati kasnije.

Predložak komponente `AppMenuComponent` nalazi se u datoteci `app-menu.component.html`. Unutar nje možemo vidjeti primjer korištenja usmjernika. Prvi link unutar navigacije vodi na prognozu vremena u budućnosti:

```
<a routerLink="/forecast" routerLinkActive="active">Forecast</a>
```

Drugi link vodi na jučerašnju vremensku prognozu:

```
<a routerLink="/yesterday" routerLinkActive="active">Yesterday weather</
    a>
```

Za navigaciju po prikazu vremenskog razdoblja korisimo Angular-ovu direktivu `ngFor`:

```
<li *ngFor="let webpage of webpageList">
  <a [routerLink]="['/history', 7, webpage.webpage]" routerLinkActive="
    active">{{webpage.webpage}}</a>
</li>
```

Ovdje možemo vidjeti primjer dva parametra. Prvi je broj 7, koji označava da želimo prikaz prognoze tjedan dana unatrag, a drugi je ime web stranice za koju želimo prikaz. `ngFor` direktiva će za svaku web stranicu unutar niza `webpageList` stvoriti po jednu poveznicu. Ključna riječ `routerLink` označava nam putanju koju će ruter obraditi i prikazati odgovarajuću komponentu.

Usluga `ForecastService` implementirana je u datoteci `forecast.service.ts`. Ona sadrži funkcije za dohvaćanje raznih vrsta vremenske prognoze. Sve funkcije su međusobno slične, a kao primjer pogledat ćemo funkciju `getForecast`:

```
getForecast(city: string, date: string): Promise<Forecast[]> {
  const url = `${this.forecastUrl}get.php?city=${city}&date=${date}`;
  return this.http.get(url, this.options)
    .toPromise()
    .then(response => response.json() as Forecast[])
    .catch(this.handleError);
}
```

Ova funkcija kao ulaz prima ime grada i datum za koji želimo vremenske prognoze. Povratni tip ove funkcije je objekt tipa `Promise` koji je parametriziran kao niz objekata tipa `Forecast`. Ovdje koristimo tip `Promise` zato što niz vremenskih prognoza dohvaćamo iz baze podataka, pa nam taj niz neće biti dostupan odmah, nego nakon nekog vremena. Podaci će se dohvatiti asinkrono, u pozadini, a tek kad budu dostupni bit će vraćeni na

mjesto poziva funkcije. Podatke dohvaćamo jednostavnim `GET` pozivom serverske skripte `get.php` opisane u prethodnoj sekciji. Kao odgovor dobiti ćemo niz vremenskih prognoza u JSON formatu, koje pospremamo u niz objekata tipa `Forecast[]`.

Središnji dio aplikacije je usmjernik koji prikazuje odgovarajuće komponente s grafovima o vremenskoj prognozi. Usmjernik je definiran u datoteci `chart-routing.module.ts`. On se sastoji od četiri jednostavne rute.

```
const routes: Routes = [
  { path: '', redirectTo: '/forecast', pathMatch: 'full' },
  { path: 'forecast', component: ForecastChartComponent },
  { path: 'yesterday', component: YesterdayChartComponent },
  { path: 'history/:days/:webpage', component: HistoryChartComponent }
];
```

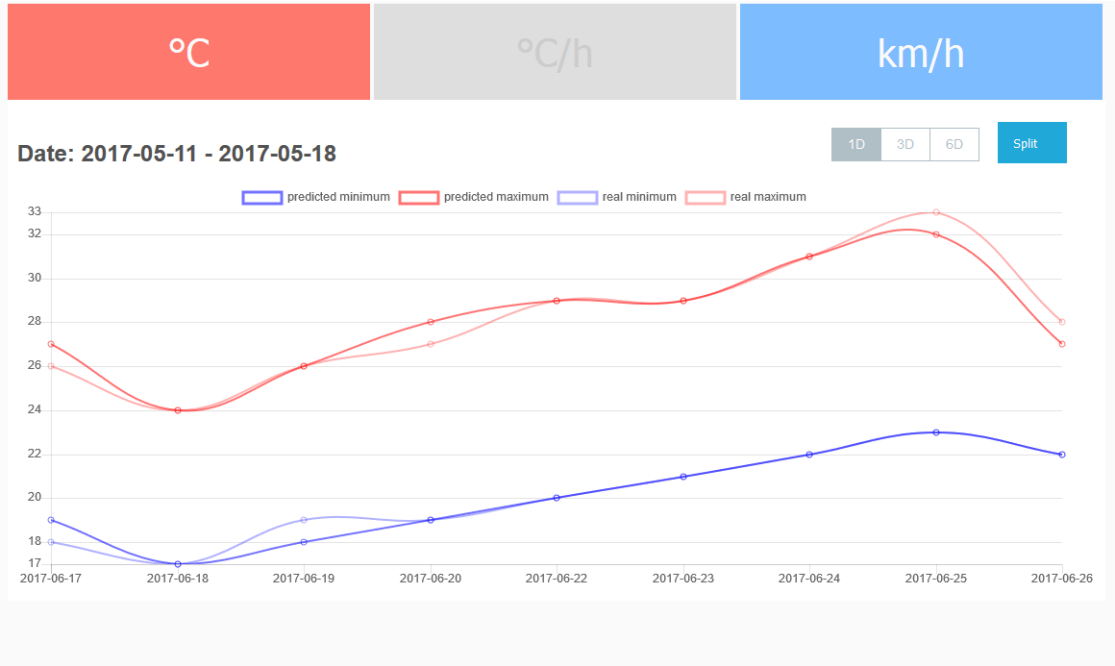
Prva ruta je ruta početne stranice, koja je preusmjerena na putanju `/forecast`. Sljedeće tri rute predstavljaju nam tri grupe prikaza vremenske prognoze. Prva od njih je putanja `forecast`, a pridružena joj je komponenta `ForecastChartComponent`. Druga je putanja `/yesterday`, a pridružena joj je komponenta `YesterdayChartComponent`. Zadnja putanja `history/:days/:webpage` je putanja koja sadrži dva parametra koja smo opisali ranije. Ona vodi do komponente `HistoryChartComponent`.

Svaka od tih komponenti sadržava pomoćnu komponentu za odabir tipa vremenske prognoze. To je komponenta `ChartTypeComponent`, a definirana je u datoteci `chart-type.component.ts`. Njezin predložak (`chart-type.component.html`) sadrži samo tri elementa <button> koji predstavljaju jedan od tri tipa vremenske prognoze: dnevna temperatura, temperatura po satu i jačina vjetra. Na pritisak nekog od njih poziva se `onSelect` metoda komponente. Klasa drugog <button> elementa postavlja se automatski ovisno o vrijednosti varijable `hourlyButtonClass` pripadne komponente. Neke vremenske prognoze nemaju dostupan drugi tip, pa na taj način možemo onemogućiti odabir drugog tipa vremenske prognoze, mijenjajući samo klasu elementa <button>. Sama komponenta izgleda ovako:

```
export class ChartTypeComponent {
  selectedType : Number = 0;
  hourlyButtonClass : string = 0;

  @Output() onTypeSelect = new EventEmitter<Number>();

  onSelect(newSelectedType: Number){
    this.selectedType = newSelectedType;
    this.onTypeSelect.emit(newSelectedType);
  }
}
```



Slika 4.6: Primjer nedostupnog tipa vremenske prognoze (temperatura po satima je zasivljena)

Varijabla `selectedType` sadrži trenutno odabrani tip vremenske prognoze. Unutar metode `onSelect` pridružujemo joj novu odabranu vrijednost. Ovdje također emitiramo novi događaj preko varijable `onTypeSelect`. Svaka komponenta koja sadrži ovu komponentu može pratiti događaj `onTypeSelect` i na taj način će biti obaviještena o svakoj promjeni tipa vremenske prognoze.

Sve tri glavne komponente međusobno su slične, a kao primjer ćemo analizirati komponentu `ForecastChartComponent`. Ona je implementirana u datoteci `forecast-chart.component.ts`, a njezin predložak nalazi se u datoteci `forecast-chart.component.html`. Predložak se sastoji od dva dijela, odabira tipa vremenske prognoze i grafa koji prikazuje vremensku prognozu.

```
1 <chart-type (onTypeSelect)="onTypeSelect($event)"></chart-type>
```

Na mjesto ove linije ubacit će se komponenta `ChartTypeComponent` i omogućiti biranje tipa prognoze. Nakon ovog elementa nalaze se tri tipke za odabir koliko dana unaprijed želimo vidjeti vremensku prognozu. Na odabir neke od njih poziva se funkcija

`OnChange()`. Odabrani broj dana povezan je s varijablom `selectedDaysBefore`, koristeći dvostrano vezanje podataka. Osim toga, možemo birati i grad za koji želimo vidjeti vremensku prognozu. Odabir dana implementiran je pomoću elementa `<select>`

Slika 4.7: Odabir gradova

i Angular-ove direktive `ngFor`. Odabrani grad povezan je s varijablom `selectedCity`, također koristeći dvostrano vezanje podataka.

```
<select [(ngModel)]="selectedCity" (change)="OnChange()">
<option *ngFor="let city of cityList" value={{city.city}}>{{city.city
    }}</option>
</select>
```

Na kraju predloška vidimo primjer grafa. Svi parametri grafa povezani su s varijablama komponente, i na taj način graf prilagođavamo određenoj vremenskoj prognozi.

Komponenta prati sve događaje koje korisnik pokrene, kao što su promjena tipa vremenske prognoze ili promjena grada. Funkcija `onTypeSelect` poziva se kada korisnik promjeni tip vremenske prognoze u komponenti `ChartTypeComponent`. Tada pozivamo metodu `OnChange` koja će se pobrinuti o novim promjenama. Ovdje možemo vidjeti primjer korištenja usluge `ForecastService`.

```
this.forecastService.getForecast(this.selectedCity, this.getDateString(
    this.selectedDaysbefore)).then(forecasts => this.UpdateChart(
    forecasts));
```

Od usluge tražimo sve prognoze vezane za odabrani grad i za odabrani datum. Kada su prognoze dostupne, pozivamo funkciju `UpdateChart` koja osvježava graf novim podacima.

Za crtanje grafova koristimo biblioteku u JavaScript-u `Chart.js`. Ona omogućuje jednostavnu implementaciju grafa na sljedeći način:

```
<div class="chart-wrapper">
  <canvas baseChart class="chart"
    [datasets]="mainChartData"
    [labels]="mainChartLabels"
    [options]="mainChartOptions"
    [colors]="mainChartColors"
    [legend]="mainChartLegend"
    [chartType]="mainChartType"
    (chartHover)="chartHovered($event)"
    (chartClick)="chartClicked($event)">
  </canvas>
</div>
```

Atribute grafa (`datasets`, `labels`, `options`, `colors`, `legend`, `charType`) definiramo preko pripadnih članova komponente koja koristi predložak sa grafom. Na taj način omogućeno nam je dinamičko mijenjanje grafa. Na kraju navodimo i dva događaja, `chartHover` i `chartClick` koji pozivaju funkcije `chartHovered` i `chartClicked` pripadne komponente kada korisnik prelazi kursorom miša iznad grafa, odnosno, kada klikne na graf.

Na primjer, `chartData` mogao bi izgledati ovako:

```
{
  label: 'Zagreb',
  data: [12, 19, 3, 17, 6, 3, 7],
  backgroundColor: "rgba(153,255,51,0.4)"
},
```

dok bi `chartLabels` mogao izgledati ovako:

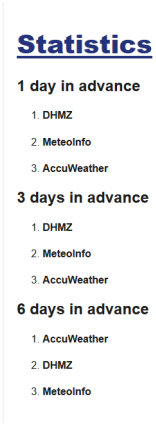
```
['Pon', 'Uto', 'Sri', 'Cet', 'Pet', 'Sub', 'Ned']
```

Na kraju imamo prikaz statistike o točnosti vremenskih prognoza. Predložak statistike sastoji se od tri liste koje popunjavamo Angular-ovim direktivama `ngFor`.

```
<li *ngFor="let webpage of statistics[0]">
<h4> {{webpage[0]}} </h4>
</li>
```

Podatke dohvaćamo korištenjem servisa `ForecastService`, odmah nakon što se aplikacija pokrene.

```
this.forecastService.getLastTwoWeeks().then(forecasts => this.OnLoad(
    forecasts));
```



Slika 4.8: Statistika

Ovim smo opisali strukturu aplikacije i neke njezine važnije implementacijske detalje. Potpuni izvorni kod aplikacije dostupan je na CD-u priloženom uz ovaj rad, a aplikacija se nalazi na: `http://somniumgames.com/diplomski/ngApp/src/index.html`.

Bibliografija

[1] AccuWeather, https://www.accuweather.com, posjećena svibanj 2017.

[2] Dark Sky, https://darksky.net/forecast/45.1667,15.5/si24/en, posjećena svibanj 2017.

[3] A. Hussain, *Angular 2: From Theory To Practice*, Daolrevo Ltd, 2016.

[4] Meteo-info, http://www.meteo-info.hr/, posjećena svibanj 2017.

[5] Open Weather Map, https://openweathermap.org, posjećena svibanj 2017.

[6] Web stranice biblioteke ChartJS, https://chartjs.org/, posjećena srpanj 2017.

[7] Web stranice Državnog hidrometeorološkog zavoda, http://meteo.hr, posjećena svibanj 2017.

[8] Web stranice programskog jezika TypeScript, https://www.typescriptlang.org/, posjećena srpanj 2017.

[9] Web stranice razvojnog okvira AngularJS, https://angularjs.org/, posjećena srpanj 2017.

[10] Web stranice razvojnog okvira Angular, https://angular.io, posjećena srpanj 2017.

Sažetak

U ovom diplomskom radu predstavljeni su dijelovi i arhitektura razvojnog okvira Angular. Upoznali smo se sa programskim jezikom TypeScript, te smo istaknuli neke razlike između njega i jezika od kojeg je nastao i u kojeg se prevodi, JavaScript-a. Opisali smo komponente, osnovne građevne jedinice Angular-a, te način na koji se komponente pakiraju u module. Sintaksu Angular-a smo upoznali i kroz predloške, tj. HTML datoteke obogaćene dodatnim direktivama koje Angular koristi za prikaz komponente na ekranu.

Kako bismo lakše prikazali funkcionalnosti Angular-a, kreirali smo web aplikaciju za prikaz vremenske prognoze. Web aplikacija sa servera dohvaća podatke o vremenskim prognozama, a koje je server već ranije pohranio u svoju bazu podataka koristeći više web-servisa specijaliziranih za vremenske prognoze. Zatim međusobno uspoređuje podatke sa različitih web servisa, a vremenske prognoze iz prošlosti uspoređuje sa stvarnim podacima o vremenu. Na taj način mogu se usporediti točnosti prognoza pojedinih web servisa. Osim sa komponentama i predlošcima, kroz aplikaciju smo se upoznali i sa usmjernikom, tj. dijelom Angular-a koji koristimo za navigaciju kroz komponente.

Summary

In this thesis we presented parts and architecture of the Angular framework. We introduced the TypeScript programming language, and we highlighted some differences between TypeScript and the language from which it originated and to which it compiles – JavaScript. We described components, the basic building blocks of Angular, and how the components are packed into modules. We got acquainted with Angular syntax through templates – the enriched HTML files that Angular uses to display the components on the screen.

To make it easier to show Angular features, we created a web application for displaying weather forecasts. The web application retrieves weather data from the server, which has already downloaded it from multiple web services and stored it in a database. The web application then compares the data from different web services, and it also compares the past weather forecasts with actual, measured data. This way we can compare the accuracy of individual web services. Apart from the components and templates, through the app we also studied the router – a part of Angular that is used for navigation between components.

Životopis

Osobni podaci:

- Prezime i ime: Grđan Barbara
- Datum rođenja: 13. srpnja 1993.
- Mjesto rođenja: Zagreb, Republika Hrvatska

Obrazovanje:

- 2015.-2017. Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet, Matematički odsjek, diplomski studij Računarstvo i matematika
- 2012.-2015. Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet, Matematički odsjek, preddiplomski studij Matematika, smjer nastavnički
- 2008.-2012. Gimnazija Velika Gorica, Velika Gorica
- 2000.-2008. Osnovna škola Jurja Habdelića, Velika Gorica

Zvanje:

- srpanj 2012. "Sveučilišna prvostupnica edukacije matematike", Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet, Matematički odsjek, preddiplomski studij Matematika, smjer nastavnički