Izvedba MVC arhitekture primjenom AngularJS razvojnog okvira i TypeScripta

Klobučar, Marko

Undergraduate thesis / Završni rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University North / Sveučilište Sjever**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:122:509085

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

University North Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište
Sjever

Završni rad br. 558/MM/2017

Izvedba MVC arhitekture primjenom AngularJS razvojnog okvira i TypeScripta

Marko Klobučar, 0316/336

Varaždin, rujan 2017. godine

Sveučilište Sjever

Multimedija, oblikovanje i primjena

Završni rad br. 558/MM/2017

Izvedba MVC arhitekture primjenom AngularJS razvojnog okvira i TypeScripta

Student

Marko Klobučar, 0316/336

Mentor

Vladimir Stanisavljević, mr.sc.

Varaždin, rujan 2017. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Odjel za multimediju, oblikovanje i primjenu

PRISTUPNIK: Klobučar Marko | MATIČNI BROJ: 0316/336

DATUM: 19.09.2017. | KOLEGIJ: Programski alati 3

NASLOV RADA: Izvedba MVC arhitekture primjenom AngularJS razvojnog okvira i TypeScripta

NASLOV RADA NA ENGL. JEZIKU: Implementation of MVC architecture through AngularJS framework and TypeScript

MENTOR: mr.sc. Vladimir Stanisavljević | ZVANJE: viši predavač

ČLANOVI POVJERENSTVA:

1. doc.dr.sc. Dejan Valdec - predsjednik
2. dr.sc. Andrija Bernik, pred. - član
3. mr.sc. Vladimir Stanisavljević, v.pred. - mentor
4. dr.sc. Robert Logožar, v. pred. - zamjenski član
5.

Zadatak završnog rada

BROJ: 558/MM/2017

OPIS:

Programski jezik JavaScript osnova je rada svih dinamičkih web stranica. Dugi niz godina standardiziran je kroz ECMA script specifikaciju. Razvojni okviri olakšavaju rad s JavaScriptom.

U ovom radu potrebno je:
* opisati osnove AngularJS-a razvojnog okvira i TypeScripta-a, njihovu ulogu u izradi Web stranica te analizirati njihove prednosti i ograničenja,
* opisati postupak instalacije i osnovnog korištenja AngularJS-a i TypeScripta-a,
* detaljno obraditi mogućnosti i sintaksu AngularJS-a i TypeScripta te na primjerima pokazati njihovo korištenje,
* osmisliti i oblikovati demo web stranicu na kojoj će biti prikazane glavne mogućnosti, načini korištenja AngularJS-a i TypeScripta-a, a u radu objasnit primjere koda,
* ukratko usporediti TypeScript sa sličnim sustavima

Detaljno opisati sve korištene tehnologije i korake u radu potrebne da bi se ostvarilo traženo te detaljno opisati stečena iskustva i postignute rezultate.

REPUBLIKA HRVATSKA
SVEUČILIŠTE SJEVER

ZADATAK URUČEN: 19.09.2017. | POTPIS MENTORA:

Sadržaj

1. Uvod 1
1.1 O pojmu arhitekture u softverskom razvoju 2
1.2 Povijest MVC arhitekture 2
1.3 Opis MVC softverske arhitekture 2

2. TypeScript 4
2.1 Povijest TypeScript-a 4
2.2 Mogućnosti TypeScript-a 5
2.2.1 Tipovi podataka 5
2.2.2 `let` naspram `var` 6
2.2.3 Klase u TypeScriptu 8
2.3 Usporedba TypeScripta s ostalim jezicima 13

3. AngularJS 14
3.1 Povijest AngularJS razvojnog okvira 14
3.2 Mogućnosti AngularJS razvojnog okvira 15
3.3 Glavni dijelovi AngularJS-a 17
3.3.1 Moduli 17
3.3.2 Rute 19
3.3.3 Kontroleri 20
3.3.4 Direktive 20
3.3.5 Servisi 24
3.4 Usporedba AngularJS-a s konkurencijom 26
3.5 Instalacija AngularJS-a i TypeScripta 27
3.5.1 AngularJS 27
3.5.2 TypeScript 27

4. Praktični primjer aplikacije 28
4.1 Google FireBase 28
4.2 Razvojna okolina 31
4.3 Osnova aplikacije 32
4.4 Predočaji 38
4.4.1 Naslovna strana 38
4.4.2 Kviz 41
4.4.3 Rezultati 45
4.5 Komponente 50

5. Zaključak 53

6. Popis slika 55

7. Popis tablica 56

8. Popis koda 57

9. Popis literature 58

Predgovor

Zahvaljujem se Tihomiru Dmitroviću, na njegovom strpljenju i volji za podučavanjem.
Zahvaljujem se profesoru Vladimiru Stanisavljeviću, na mentoriranju ovog rada.
Zahvaljujem se svim svojim prijateljima na neprestanoj podršci i bodrenju u mojim naporima.
Ali ponajprije, zahvaljujem se svojoj supruzi, bez koje nikada ne bih zakoračio u svijet programiranja.
Hvala vam.

Sažetak

Internet je mjesto bogato sadržajem, od svog začeća pogonjen jednim od danas najpopularnijih jezika – JavaScriptom. Međutim, JavaScript nije bez mana, pa su kao odgovor na mnoštvo njegovih problema nastale nebrojene programske zbirke, razvojni okviri, pa čak i čitavi novi jezici. Danas, jedan okvir i jedan jezik su postali popularniji od drugih, a to su AngularJS framework i TypeScript jezik. Ovaj rad predstavit će njihove prednosti, mane i primjere istih. Isto tako, rad pokazuje kako napisati bogate web aplikacijem putem MVC arhitekture, koja je dizajnirana radi lakšeg održavanja tijela kôda. Kombinacijom tih tehnologija i uzoraka moguće je na lakši i bolji način stvoriti web aplikacije bogate sadržajem, na primjeru jednostavne web aplikacije za pomoć pri odabiru mehaničkog prekidača za tipkovnicu.

Ključne riječi: Razvojni okvir, deklarativno programiranje, klase, objekti

Abstract

The internet is a place rich with content, which has, since its inception, been powered by one of today's most popular programming languages – JavaScript. However, JavaScript is not without its share of flaws. As a response to that, there have arisen countless libraries, frameworks, and even entirely new languages. Today, one framework and one language have risen above others, and they are the AngularJS framework and TypeScript language. This paper aims to introduce their advantages, flaws and to provide examples. In addition, this paper shows how to design and write rich web applications via the MVC architecture, which has been designed to ease the maintenance of large bodies of code. Utilizing a combination of the aforementioned technologies and patterns, it is possible to more easily and efficiently create web applications rich in content, based on an example web application designed to help with choosing a mechanical keyboard switch.

Keywords: Framework, declarative programming, classes, objects

Popis korištenih kratica

HTML **H**yper**T**ext **M**arkup **L**anguage
Hipertekstualni označni jezik

DOM **D**ocument **O**bject **M**odel
Objektni model dokumenta

URL **U**niform **R**esource **L**ocator
Ujednačeni lokator resursa

npm **N**ode **P**ackage **M**anager
Node-ov upravitelj paketa

Pojmovnik

Engleski	**Hrvatski**
Pattern	Uzorak
Interface	Sučelje
Namespace	Imenski prostor
Framework	Razvojni okvir
Two-way binding	Dvosmjerno vezivanje
Dirty checking	Brzinska provjera
Modular	Sastavan
Model	Ogled
View	Predočaj
Controller	Upravljalo
Separation of concerns	Odvajanje briga
Inversion of control	Obrtanje kontrole
Uniform Resource Locator	Jedinstveni lokator resursa
Expression	Izraz

1. Uvod

Kroz prethodne godine, najpopularniji jezik za kreiranje bogatih web aplikacija je postao JavaScript. Premda su drugi, konkurentni jezici pokušali uzeti krunu, JavaScript je prevladao zbog primarno jednog faktora – fleksibilnosti. No, ta fleksibilnost, uz sve prednosti koje nosi, također donosi i neke probleme. Ponajprije, nedostatak tipova podataka predstavlja problem mnogim programerima i dan danas, te je sve više rastuća potreba za bogatim, interaktivnim web aplikacijama pridonjela ovom problemu. Kao odgovor su se pojavile mnoge programske zbirke i razvojna okruženja, pa čak i jezici, no u ovom radu, fokus je na onim najutjecajnijima i, za sada, najboljima.

AngularJS, kao razvojni okvir, svojim do tada jedinstvenim pristupom razvoju aplikacija predstavio je revoluciju u području dizajna web platformi, dok je Microsoftov TypeScript na scenu stupio s ciljem da razriješi najveće probleme koje su mučile JavaScript programere godinama.

No, sve to nije moguće bez odgovarajuće filozofije, uzorka koji će sve to objediniti i učiniti stvaranje fleksibilnih, ali stabilnih aplikacija lakšim. Trygve Reenskaug će, davne 1979. godine, osmisliti softverski uzorak koji će danas omogućiti tisućama programera da uspiješnije i lakše razvijaju aplikacije. Ovaj rad će redom obraditi sve teme, te će pojasniti proces na praktičnom primjeru web aplikacije.

1.1. O pojmu arhitekture u softverskom razvoju

Prije nego se objasni što točno MVC arhitektura (engl. *architecture*) znači, valja objasniti pojam arhitekture unutar okvira softverskog (engl. *software*) razvoja. Kao što se vidi u [4], *softverska arhitektura* je pojam izgradnje programa koji predstavlja tzv. *najbolje prakse* (engl. *best practices*) u pisanju računalnih programa.

Programska arhitektura je skup pravila i navoditelja uz pomoć kojih je rezultirajući znakovnik (engl. *code*) uredno ustrojen, lako održiv, te idealno sastavan (engl. *modular*), odnosno da se bez poteškoća mogu ubacivati, micati i izmjenjivati dijelovi programa. Na taj način se dugoročno ušteđuje novac i vrijeme izgradnje računalnih programa, no da bi se postigao taj rezultat potrebno je dobro planirati i posvetiti se izgradnji samog ustroja programa prije nego se krene na pisanje.

1.2. Povijest MVC arhitekture

MVC uzorak osmislio je računalni znanstvenik Trygve Mikkjel Heyerdahl Reenskaug, rođen 21. 6. 1930. Danas radi kao *professor emeritus* na Sveučilištu u Oslu [6]. Reenskaug o ovom uzorku kaže sljedeće:

> MVC je zamišljen kao općenito rješenje za problem korisnika koji kontroliraju velike i kompleksne skupove podataka. Najteži dio je bio dogovoriti se oko imena svih komponenti. *Model-View-Editor* je bio prvi prijedlog. Nakon duge diskusije, primarno s Adele Goldberg, došli smo do termina *Model-View-Controller* [6].

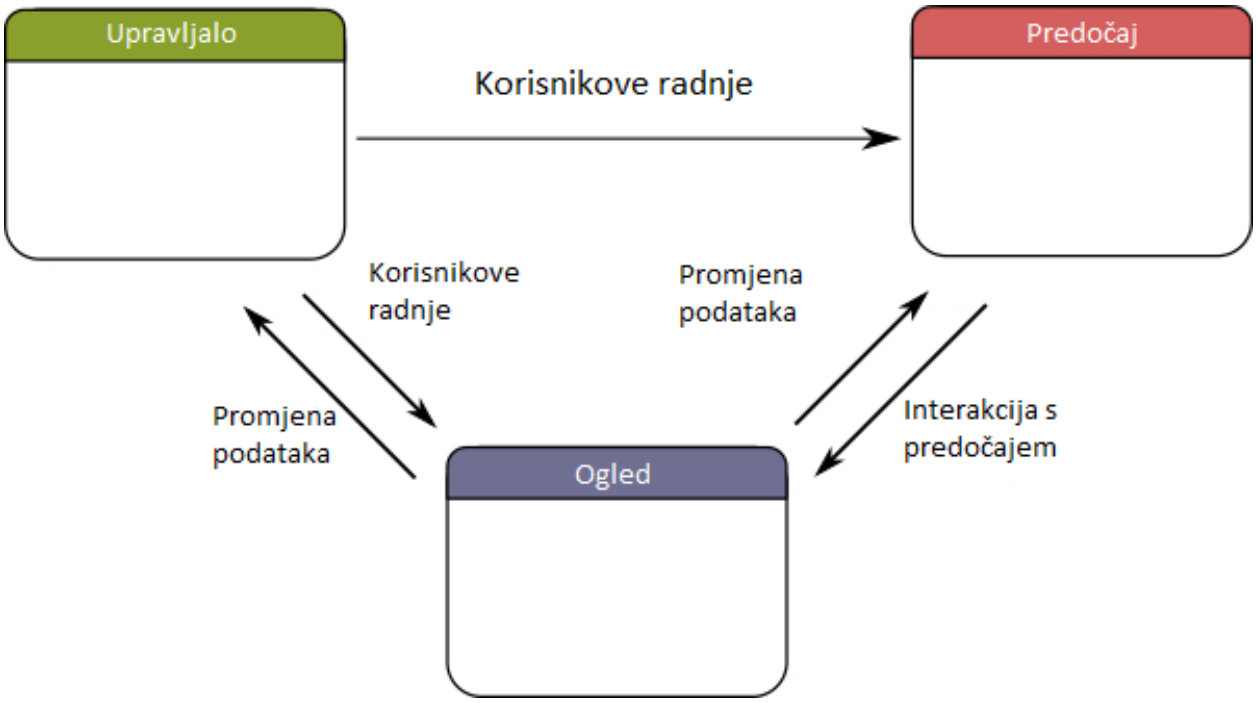
Slika 1.1: Trygve Reenskaug u 2010.
Izvor: [6]

MVC uzorak je osmišljen tijekom posjeta prof. Reenskauga Xerox Palo Alto istraživačkom središtu (engl. *Xerox Palo Alto Research Center*) 1979. godine, dok je njegov prvi softverski projekt bio ”Autokon”, od kojega je nastao uspješan CAD/CAM programa koji se prvo koristio 1963., te se nastavio koristiti na brodogradilištima diljem svijeta narednih 30 godina [6].

1.3. Opis MVC softverske arhitekture

MVC je softverska arhitektura koja je stvorena kako bi olakšala razvoj računalnih programa, te ga predstavila na način koji je prirodniji ljudima. Veoma rano u povijesti računalnog programiranja

pojavio se pojam koji danas poznajemo kao *odvajanje briga* (engl. *separation of concerns*) [8], koji označava odvajanje različitih dijelova programa na taj način da je čovjeku lakše razlučiti koji dio je odgovoran za što. Isto tako, uporabom ovog principa postiže se prije spomenuta sastavnost programa koja olakšava daljnje dograđivanje ili ponovo korištenje. Sukladno tom principu stvorena su mnoga rješenja, među kojima je skup programskih arhitektura, od kojih je među prvima naslovna MVC programska arhitektura. Kao što samo ime govori, ova arhitektura sastoji se od tri glavna dijela, s time da je treći, *Controller*, lako izmjenljiv ovisno o potrebama programa i razvojnog tima, no primarno se koristi u aplikacijama u kojima je potrebno uvesti korisničko sučelje [5].



Slika 1.2: Grafikon MVC arhitekture
Izvor: [7]

Princip rada MVC programske arhitekture je u svojoj srži veoma jednostavan. Kao što se vidi na slici 1.2, riječ je o kružnom procesu koji se ”neprekidno” vrti [5].

Uzmimo kao početnu točku ogled. Njegova je uloga sadržavanje trenutnog stanja sirovih podataka. Ogled ne smije znati što se trenutačno prikazuje korisniku, niti se od ogleda izravno traži da opskrbljuje podatke. To znači da korisnik sa svoje strane, odnosno sa strane korisničkog sučelja, ne smije imati mogućnost posezati izravno u ogled. Svo izmjenjivanje, umetanje i brisanje podataka se odvija neizravno preko upravljala.

Korisniku se u svakom trenutku prikazuje predočaj. On sadržava trenutni *izgled* podataka, dodatno obrađen da bi korisniku bio što korisniji. Na predočaj također ne dolaze svi podatci, već se filtriraju neke osjetljive informacije, ili pak informacije koje nisu od koristi klijentu. Primjerice, u relacijskom modelu nekakvih ustanova, svaka pojedina ustanova može biti označena jedinstvenim

identifikatorom koji korisniku nije od koristi, pa se taj podatak ne prikazuje na predočaju.

Kao posrednik između prijašnja dva dijela jest upravljalo. Kad god korisnik izvrši kakvu radnju na sučelju, upravljalo dobije obavijest o toj radnji, te primi sve potrebne podatke vezane uz tu radnju (npr., skup uvjeta kod kakvog filtriranja). Potom poseže u model od kojega se traži novi skup podataka ili izmjena već postojećih. Nakon toga, nove podatke upravljalo prebacuje na predočaj da bi korisnik mogao vidjeti rezultat svojih radnji.

Ovakva kružna priroda uzorka, kako je možda očito, odvija se u jednom ciklusu primanja naredbi, modificiranja podataka te osvježavanja prikaza. Time se jasno određuje zadaća svakog pojedinog dijela kôda, te se kasnije mogu izmjenjivati pojedini dijelovi bez da se poremeti rad čitavog programa. To se zove labavo spajanje (engl. *loose coupling*) [16].

MVC arhitektura je zaglavni kamen u razvoju web aplikacija sa TypeScriptom i AngularJS-om, koji omogućuje ubrzan i poboljšan razvoj web aplikacija. Slijedi pregled tehnologija s pomoću kojih će se razviti web aplikacija.

2. TypeScript

TypeScript je jezik za razvoj suvremenih internetskih aplikacija. Razvijen od strane Microsofta, TypeScript je nastao kao odgovor na najčešće kritike JavaScripta – ponajprije loše podržan objektno orijentirani razvoj te nedostatak tipova podataka (od čega potječe ime). TypeScript se *compileira* u čisti JavaScript te je stoga *de facto* podržan u svim modernim internetskim preglednicima.

2.1. Povijest TypeScript-a

TypeScript je prvo pušten u javnost u listopadu 2012. godine, pod verzijom 0.8, nakon dvije godine razvoja u Microsoftu. Nedugo nakon najave, Miguel de Icaza (programer poznat po započinjanju GNOME, Mono i Xamarin projekata [11]) hvalio je sâm jezik, no kritizirao je nedostatak stabilne i zrele razvojne okoline za jezik. U to vrijeme, jedina razvojna okolina koja je podržavala TypeScript je bio Microsoftov vlastiti Visual Studio, koji nije dostupan za OS X i Linux operativne sustave [10].

2013. godine podrška za TypeScript se pojavila i u ostalim razvojnim okolinama, ponajprije u Eclipse okolini, u vidu proširenja kojega je kreirala firma *Palantir Technologies* [10]. Razni uređivači teksta, poput Emacs-a, Vim-a, Sublime Text-a, Webstorm-a i Atom-a također počinju podržavati TypeScript.

Typescript 0.9, izdan 2013., uvodi podršku za općenite tipove, kolekcije, klase i sl [10].

TypeScript 1.0 je izdan 2014. na Microsoftovoj *Build* konferenciji. Nadogradnja 2 za Visual Studio 2013 uvodi ugrađenu podršku za TypeScript [10].

U srpnju 2014. godine, tim koji stoji iza TypeScripta objavljuje novosti o novom skupniku (engl. *compiler*) za TypeScript, za koji tvrde da je do 5 puta brži [10]. Istovremeno, kôd koji se do tada nalazio na mrežnoj usluzi *CodePlex* seli se na *GitHub*.

2.2. Mogućnosti TypeScript-a

Prvotna mogućnost TypeScripta koja se ne nalazi u JavaScriptu je, kako i samo ime govori, uvođenje tipova podataka. Dok se u JavaScriptu tipovi zaključuju za vrijeme izvođenja, TypeScript je jezik koji se mora *compileirati* u JavaScript, što omogućuje provjeru tipova prije nego se program počne izvršavati.

2.2.1. Tipovi podataka

Sintaksa deklariranja varijabli i tipova je vrlo jednostavna:

```
let varName: varType = "value";
```

Programski kôd 2.1: Primjer sintakse označavanja tipova varijabli.

Gdje je varName ime varijable, varType tip varijable, i izborno inicijaliziranje uz vrijednost. Za razliku od JavaScripta, koji dozvoljava nešto poput:

```
var name = "Marko";
name = 3; // U redu, name je sada 3;
```

Programski kôd 2.2: Dinamičko mijenjanje vrijednosti neovisno o tipu

TypeScript ne dozvoljava tako što.

```
let name: string = "Marko";
name = 2; // Greška!
```

Programski kôd 2.3: TypeScriptovo strogo provjeravanje tipa

Međutim, TypeScript podržava tip `any` koji se ponaša poput standardnog JavaScripta.

```
let name: any = "Marko";
name = 3; // U redu
```

Programski kôd 2.4: `any` tip u TypeScriptu

Primjer sintakse navođenja tipova slijedi:

```
let userAge: number = 26; // Broj
let userName: string = "Jeff"; // String, odnosno niz znakova
let userPets: string[] = ["Sparky", "Fluffy"]; // Polje stringova.
// Polja se označavaju sa []
```

Programski kôd 2.5: Primjeri tipova podataka kod varijabli

Funkcijama se također može naznačiti tip podataka koje vraćaju, te se argumentima može označiti tip podataka na sličan način kao i kod varijabli:

```
// Funkcija 'add' prima dva argumenta,
// od kojih oba moraju biti 'number'
// i vraća 'number' kao rezultat.
function add(a: number, b: number): number {
^^Ireturn a + b;
}
```

Programski kôd 2.6: Tipovi podataka kod funkcija

2.2.2. `let` naspram `var`

Varijable u JavaScriptu, pa tako i TypeScriptu se označavaju ključnim riječima `var` ili, u modernijim inačicama jezika, `let`.
Razlika između tih ključnih riječi je sljedeća [12, 13]:

```
function a() {
    // Varijabla "index" nije vidljiva ovdje!
    console.log(index); // Error: 'index' is not defined.
    for (let index = 0; index < 10; index++) {
        // Varijable označene ključnom riječi 'let' vidljive su isključivo
        // U bloku unutar kojega su deklarirane, ne izvana.
        console.log(index); // 0, 1, 2, 3...
    }
    // Varijabla "index" nije vidljiva niti ovdje,
    // premda je ovo mjesto nakon for petlje, tj. nakon
    // što je varijabla "index" deklarirana.
}

function b() {
    // Varijabla "counter" JEST vidljiva ovdje!
    // Interpreter preglednika povlači deklaraciju,
    // no ne i inicijalizaciju
    // varijable na vrh funkcije b:
    // var counter;
    console.log(counter); // undefined
    for (var counter = 0; counter < 10; counter++) {
        // Ovdje je također vidljiva.
        // Varijable označene sa 'var' su "podignute"
        // na vrhovnu razinu funkcije u kojoj su deklarirane
        console.log(counter); // 0, 1, 2, 3...
    }
    // I ovdje je varijabla "counter" vidljiva!
    // Dostupna je VAN granica bloka petlje u kojoj je definirana.
    console.log(counter); // 10
}
```

Programski kôd 2.7: `var` naspram `let`

TypeScript uvodi ključnu riječ `let` iz JavaScripta u koji je uvedena u ECMAScript 2015 standardu [14]. Glavna prednost te ključne riječi je smanjivanje broja sukoba kod slučajnog netočnog imenovanja varijabli. Isto tako, pošto varijable imaju kraći životni vijek (traju samo dok je aktivan blok u kojemu su deklarirane), opterećenje sistema je manje.

2.2.3. Klase u TypeScriptu

Druga najbitnija stvar TypeScripta je proširena mogućnost rada s klasama. Sve do ECMAScript inačice 6, u JavaScriptu je jedini način za objektno orijentirano programiranje bio sustav prototipa. Počevši s ECMAScript 6, u JavaScript je uvedena ključna riječ `class`.

```
class Person {
    constructor(name) {
        this._name = name;
    }

    speak() {
        return 'My name is ' + this._name + '.';
    }
}

let jeff = new Person('Jeff');

jeff.speak(); // My name is Jeff.
```

Programski kôd 2.8: Primjer klase u JavaScriptu

Kao što se vidi u primjeru 2.8, definirana je jednostavna klasa Person, koja pri instanciranju prima jedan argument, ime. Metoda `speak()` vraća *string*. No, kao što je vidljivo iz primjera, nigdje nije naznačen tip podatka koji mora biti proslijeđen u konstruktor. Isto tako, jedna od najvećih mana klasa u JavaScriptu jest ta što ne postoji koncept dozvole pristupa – to će reći, ne postoje privatna, zaštićena i javna polja i metode, već je sve javno i dostupno. To znači da se s klasom iz kôda 2.8 može napraviti sljedeće:

```
let jeff = new Person('Jeff');

jeff.name; // 'Jeff'
```

Programski kôd 2.9: Nedostatak dozvole pristupa u JavaScript klasama

U primjeru 2.9 vidi se da je slobodan pristup svim poljima klase Person, premda je to nečuveno

za strogo objektno orijentirano programiranje. TypeScript uvodi mnoge preinake i poboljšanja za klase, od već spomenutog naznačivanja tipova, pa do kontrole prava pristupa.

U TypeScriptu, zamišljena klasa Animal bi izgledala kao u primjeru 2.10:

```
class Animal {
    private _name: string;
    private _sound: string;

    public constructor(name: string, sound: string) {
        this._name = name;
        this._sound = sound;
    }

    public speak(): string {
        return this._name + ' says: "' + this._sound + '".';
    }
}

let paul: Animal = new Animal('Paul', 'Woof'); // varijabla 'paul' mora biti tip 'Animal'
paul.speak(); // Paul says: "Woof".
paul.name; // Error: Property 'name' is private and is only accessible within class 'Animal'
```

Programski kôd 2.10: Primjer tipova i dozvole pristupa

Kao što je vidljivo u kôdu 2.10, TypeScript omogućava izričito naznačivanje dostupnosti i anotaciju tipova podataka. To nam omogućava da zabranimo pristup poljima unutar klase, jer bi u suprotnom moglo doći do grešaka u funkcioniranju klase ako se prepiše preko već postojećeg polja. Valja napomenuti da je dobra navika uvijek izričito naznačivati razine pristupa (private, protected, public), jer ako se ne naznači, TypeScript *de facto* podrazumijeva public razinu (dostupno svima).

Također se vidi da kasnije varijable možemo navesti pod tipom klase, jer klasa je u svojoj srži kompleksan tip podataka.

Uz ovo proširivanje mogućnosti klasa, TypeScript također nastavlja poboljšavati objektno orijentirane aspekte uvođenjem par stvari[1], ponajprije:

- **Sučelja** (engl. *interfaces*) su način da se standardizira funkcioniranje klasa. Uz pomoć njih možemo odrediti koje metode, s kakvim potpisima sve klase koje *implementiraju* sučelje moraju imati.

- **Apstraktne klase** (engl. *abstract class*) su klase koje se ne mogu instancirati, odnosno ne može se napraviti pojedinačna inačica. Apstraktne klase služe za par stvari, ponajprije

[1] Lista nije sveobuhvatna.

za dijeljenje zajedničke funkcionalnosti među svim klasama koje od nje nasljeđuju, ili za stvaranje pomoćne funkcionalnosti koja je sveprisutna (npr., formatiranje teksta za ispis).

- **Imenski prostor** (engl. *namespace*) pružaju način da se zaobiđu sukobi u imenovanju. Metaforički, imenski prostori su kao prezimena kod ljudi. Ako postoje dvije osobe imena Ivan, može ih se razlikovati po prezimenu (u većini slučajeva). Isto tako, imenski prostori služe za obuhvaćanje funkcija, metoda i sl. u svoj djelokrug [15].

Kao primjer sučelja i apstraktnih klasa uzmimo sljedeće:

```
interface ITalks {
    speak(): string;
}

abstract class LivingBeing implements ITalks {
    protected _name: string;
    protected _species: string;
    protected _sound: string;

    public constructor(name: string, species: string, sound: string) {
        this._name = name;
        this._species = species;
        this._sound = sound;
    }

    abstract speak(): string;
}

class Animal extends LivingBeing {
    public speak(): string {
        return `${this._name} the ${this._species} says '${this._sound}'.`
    }
}

class Person extends LivingBeing {
    public speak(): string {
        return `${this._sound}! I'm ${this._name}.`;
    }
}
```

Programski kôd 2.11: Primjer sučelja i apstraktnih klasa 1/2

```
class ConversationManager {
    private _members: ITalks[];

    public constructor(members: ITalks[]) {
        this._members = members;
    }

    public logConversation(): void {
        console.log('The conversation starts.');

        for (var c of this._members) {
            console.log(c.speak());
        }

        console.log('The conversation ends.');
    }
}

let jeff: Person = new Person('Jeff', 'human', 'Hello');
let doge: Animal = new Animal('Doge', 'dog', 'Woof');

let cm: ConversationManager = new ConversationManager([jeff, doge]);

cm.logConversation();

/* Output:
*  The conversation starts.
*  Hello! I'm Jeff.
*  Doge the dog says "Woof".
*  The conversation ends.
*/
```

Programski kôd 2.12: Primjer sučelja i apstraktnih klasa 2/2

U primjerima 2.11 i 2.12 postupak je sljedeći:

1. Definirano je sučelje `ITalks`, koje osigurava da će klasa koja implementira ovo sučelje morati isto tako implementirati metodu `speak()` koja ne prima argumente te mora vraćati vrijednost tipa `string`

2. Deklarirane je apstraktna klasa `LivingBeing` koja implementira sučelje `ITalks`. Apstraktna klasa, kao što je već spomenuto, ne može biti instancirana, tako da nije moguće napisati `let being = new LivingBeing();` – to bi rezultiralo greškom. Ova apstraktna klasa sadrži sva polja koja ostale mogu naslijediti: ime, vrstu i zvuk koji proizvode. Definiran je i konstruktor za instanciranje klase. Najzad, definirana je apstraktna metoda speak, implementirana iz sučelja `ITalks`, no i dalje nije definirana. Apstraktne metode moraju biti definirane u prvoj konkretnoj klasi koja nasljeđuje od apstraktne klase.

3. Deklarirane su dvije klase koje nasljeđuju od `LivingBeing`, `Animal` i `Person`. U ove dvije klase mora se definirati metoda `speak()` koja je naslijeđena od roditeljske klase `LivingBeing`.

4. Deklarirana je nova klasa `ConversationManager` koja sadrži privatno polje `_members`. Izuzetno je važno napomenuti da je kao tip ovog polja navedeno ime *sučelja*, što iscrtava još jednu važnu ulogu sučelja – mogu se koristiti da bi se osiguralo da određena varijabla sadržava polja i/ili metode definirane u tom sučelju. U ovom slučaju, osigurano je da će svi unosi u polju `_members` imati metodu `speak()`.

5. Javna metoda `logConversation()` prolazi kroz svakog člana polja `_members` te poziva metodu `speak()` na svakom članu.

6. Najzad, inicijalizirane su dvije varijable, *jeff*, klase `Person` i *doge*, klase `Animal`. Potom su te dvije varijable proslijeđene novoj instanci klase `ConversationManager`, te je pozvana metoda `logConversation()`, što rezultira ispisivanjem ”razgovora” svih sudionika.

Iz navedenoga je jasna moć sučelja i apstraktnih klasa. Uz pomoć sučelja s lakoćom je napisana metoda `logConversation()` koja poziva metodu `speak()` na svim članovima, metodu koja je upravo zbog sučelja morala biti implementirana.

Uz pomoć apstraktne klase može se dijeliti funkcionalnost između svih ostalih klasa, tako da za klase `Person` i `Animal` nisu morala biti navedena polja ime, vrsta i zvuk, već su to preuzele iz roditeljske klase `LivingBeing`.

```
namespace First {
    export function log() {
        console.log('Called from first.');
    }
}

namespace Second {
    export function log() {
        console.log('Called from second.');
    }
}

First.log(); // Called from first.
Second.log(); // Called from second.
```

Programski kôd 2.13: Primjer imenskog prostora

Što se tiče imenskih prostora, u primjeru 2.13, deklarirana su dva imenska prostora, imenom `First` i `Second` (engl. prvi i drugi). Unutar oba prostora su definirane funkcije istog imena – `log()`. Isto tako, vidljivo je da ne dolazi do sukoba u imenovanju, jer jednu funkciju `log()` obuhvaća imenski prostor First, a drugu Second, tako da su, u svojoj suštini, dvije različite, jedinstvene funkcije.

Ključna riječ `export` označava *izvoz* člana imenskog prostora tako da je vidljiv drugdje, te da se može koristiti – ne tako različito od javne (*public*) metode na kakvoj klasi.

2.3. Usporedba TypeScripta s ostalim jezicima

TypeScript, kao takav, nema ”konkurenciju” kakvu imaju razvojni okviri poput AngularJS-a. Međutim, postoje drugi jezici koji se kompajliraju u JavaScript, isto kao i kod TypeScript-a. Među ostalima, najpoznatiji jezik je CoffeeScript.

CoffeeScript je funkcionalan programski jezik koji je zamišljen da ”izvuče dobre dijelove JavaScripta” [22]. Dizajner jezika je Jeremy Ashkenas. Jedna od najznačajnijih preinaka je izostavljanje ključnih riječi `var`, `let`, `const` i sl. CoffeeScript također izostavlja vitičaste zagrade te kao takav nalikuje Pythonu. Isto tako, funkcije se zapisuju u notaciji sa strelicom, primjerice:

```
    square = (x) -> x * x
    square 3 # 9
```

Programski kôd 2.14: Primjer funkcijskog zapisa u CoffeeScriptu

Ova funkcija prima jedan argument, x, te vraća taj broj pomnožen samim sobom. CoffeeScript također omogućava pozivanje funkcija bez zagrada, no predlaže se pisati ih za ugnježđene pozive. Valja napomenuti da se u CoffeeScript funkcijama ne mora nužno pisati riječ `return`, već se posljednji izraz implicitno vraća.

TypeScript i CoffeeScript su polarne suprotnosti – gdje TypeScript postrožuje pisanje uvođenjem tipova podataka i pravog rada s klasama, CoffeeScript pruža eleganciju i čitljivost nauštrb strogoće. TypeScript je bolji za veće, kompleksnije projekte, dok je CoffeeScript elegantniji te izražajniji, tj. može postići puno uz relativno malo kôda.

U ovom poglavlju predstavljen je TypeScript jezik te koje prednosti donosi nad tradicionalnim JavaScriptom. Iduće će biti obrađen AngularJS, koji, u kombinaciji sa TypeScriptom, čini moćan skup alata za programere.

3. AngularJS

AngularJS je razvojni okvir razvijen od strane Miška Hevarya, trenutno zaposlenika Googlea. AngularJS funkcionira primarno na *digest* ciklusu, doslovno prevedeno ciklus probave. To je glavna petlja koja prolazi kroz svaki aspekt aplikacije, uspoređuje trenutno stanje varijabli i vrijednosti sa onim prošlim, i ako je došlo do promjena, AngularJS automatski osvježava predočaje. Ovaj način rada se zove deklarativno programiranje [17]. U deklarativnom programiranju, redoslijed naredbi nema semantičku ulogu u funkcioniranju programa – umjesto ispisivanja što napraviti, programi se pišu na način da se opisuje što nešto jest. primjerice, linija koda `int a = b + c;` bi se čitala ne kao ”Zbroji vrijednosti varijabli b i c te spremi u a”, već kao ”Vrijednost varijable a je jednaka zbroju varijabli b i c”. Na taj način, ako se promijeni vrijednost jedne ili obje varijable, sukladno se mijenja i varijabla a [17].

3.1. Povijest AngularJS razvojnog okvira

AngularJS počeo je svoj razvoj 2009. godine u firmi Brat Tech LLC. Izvorno, `<angular/>` je bio softver koji je pogonio internetsku uslugu za pohranjivanje podataka. Cilj je bio prodavati usluge skladištenja s tarifnom jedinicom od jednog megabajta. Međutim, zbog malenog uspjeha, Miško Hevery, tvorac AngularJS-a, odlučio je izdati AngularJS kao besplatnu programsku zbirku [18].

3.2. Mogućnosti AngularJS razvojnog okvira

AngularJS je veoma moćan razvojni okvir u čijoj je srži ciklus razgradnje i razdvajanje aspekata programa. Svaki djelić aplikacije se sastoji od komada HTML-a (izborno, ako je riječ o servisu nema HTML-a) te JavaScript-a ili TypeScript-a koji njime upravlja. AngularJS je također izričito modularan. Ovaj pristup bodri razdvajanje dijelova programa u smislene, razdvojene strukture koje su kasnije lako izmjenljive i lako održive i proširive. Tako, primjerice, možemo imati modul koji se bavi prikazom prodaja, modul koji je zadužen za obavljanje transakcija, modul koji se bavi pomoćnim (engl. *utility*) funkcijama, te najzad korijenski modul koji to sve veže.

Slijedi minimalan primjer HTML i TypeScript kôda:

```
<!DOCTYPE html>
<html ng-app="app">
<head>
    <title>Hello world!</title>
</head>
<body>
    <div ng-controller="MainController">
        {{ greetings }}
    </div>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.6/angular.min.js">
    </script>
    <script src="script.js"></script>
</body>
</html>
```

Programski kôd 3.1: Minimalan HTML prikaz stranice

```
angular
    .module('app', [])
    .controller('MainController', function($scope) {
        $scope.greetings: string = "Hello world!";
    });
```

Programski kôd 3.2: Minimalan TypeScript prikaz stranice

U primjerima 3.1 i 3.2 demonstriran je minimalan funkcionirajući primjer stranice koju pogoni AngularJS razvojni okvir. Ako se pokrene HTML datoteka u bilo kojem modernom pregledniku, na bijeloj pozadini će se crnim slovima ispisati legendarni pozdrav ”Hello world!”. Slijedi opis kôda.

Prva stvar koju AngularJS zahtijeva jest da se označi djelokrug AngularJS aplikacije. `ng-app` je tzv. *direktiva*, za koju službena dokumentacija govori sljedeće:

> Na visokom nivou, direktiva je označitelj na DOM elementu, bilo u vidu atributa, komentara, CSS klase ili imena elementa, koja AngularJS-ovom DOM *compileru* govori da na taj dio DOM-a prikvači određen set funkcionalnosti [19].

`ng-app` je direktiva koja AngularJS-u govori da se čitava funkcionalnost aplikacije odnosi na taj specifični element i svo njegovo potomstvo u hijerarhijskoj strukturi DOM-a, koja se prikazuje u vidu stabla, gdje je korijen stranice `<html>` čvor, dok su `<head>` i `<body>` sestrinski elementi te djeca `<html>` čvora [1].

`ng-controller` je, kao i `ng-app`, direktiva unutar AngularJS-a. Kao što joj ime govori, kontroleri služe za upravljanje nad određenim segmentom aplikacije. Kao što je spomenuto u prijašnjim poglavljima, kontroleri su sastavni dio MVC arhitekture, te se ponašaju kao vezivo između ogleda i predočaja. Kada se na DOM element postavi `ng-controller` direktiva, njen djelokrug postaje taj element, te sva njegova djeca.

`scope`, tj. djelokrug, je unutarnji objekt unutar AngularJS okvira koji služi za vezivanje određene funkcionalnosti za djelokrug unutar kojega direktiva djeluje. Taj se objekt obično proširuje sa prilagođenim metodama i/ili varijablama. U ovom slučaju, objekt `scope` proširen je varijablom *greetings* čiji je tip podatka *string*, te joj je dodana vrijednost ”Hello world!”.

U HTML dijelu aplikacije, primjer 3.1, na retku 8 vidi se specifična sintaksa za ekstrapolaciju vrijednosti varijable. Između dva para vitičastih zagrada napiše se izraz (engl. *expression*), te potom Angular traži podatke na zadanom kontroleru (u ovom slučaju, *MainController*), te na to mjesto stavlja vrijednost varijable.

Ekstrapolacija podataka, tj. dvosmjerno vezivanje, je jedna od definirajućih karakteristika AngularJS-a. To je dio koji omogućava programerima da pišu u deklarativnoj paradigmi [17] koja potiče predvidljivost radnji tako što se definira što nešto jest, za razliku od klasične imperativne paradigme u kojoj se programu govori točno što napraviti u svakome trenutku. Na taj način naglasak je na konačnom rezultatu rada – radi se na predlošcima i okvirima koji se kasnije popunjavaju podacima.

Način na koji AngularJS je prilično jedinstven, a naziva se ”ciklus probave”, odnosno *digest cycle*. Način na koji radi je sljedeći [2]:

1. AngularJS prolazi kroz sve vrijednosti koje zadovoljavaju određene uvjete (one koje će se mijenjati) te za njih postavlja tzv. promatrače (engl. *watchers*) koji vode računa o trenutnoj vrijednosti određene varijable.

2. AngularJS prolazi kroz sve vrijednosti koje trenutno imaju zadanog promatrača, te uspoređuje vrijednost iz prijašnje iteracije s trenutnim stanjem.

   Neka se te dvije vrijednosti zovu A_{old} za staru vrijednosti i A_{new} za novu. Tada slijedi:

   - Ako je stanje jednako, tj. ako vrijedi

     $$A_{old} = A_{new}$$

     Angular ne radi ništa, već ide na sljedeću stavku.

   - Ako vrijedi da

     $$A_{old} \neq A_{new}$$

     Angular poziva posebnu metodu `apply()` koja prima jedan argument, `exp`, skraćeno od *expression* (hrv. izraz). `apply()` potom izračunava vrijednost izraza koji je primljen te poziva `$scope.$digest()` koji ažurira stanje podataka na predočajima.

Valja napomenuti da AngularJS, da bi se spriječilo beskonačno pozivanje `apply()` metode, ograničava maksimalan broj poziva prema toj metodi na 10. Ukoliko se prijeđe taj broj, Angular odašilje tzv. *InfiDig* grešku, skraćeno za *infinite digest loop*.

3.3. Glavni dijelovi AngularJS-a

Slijedi pregled nekolicine najvažnijih dijelova AngularJS-ove arhitekture – segmenti koji se koriste u 80% slučajeva.

3.3.1. Moduli

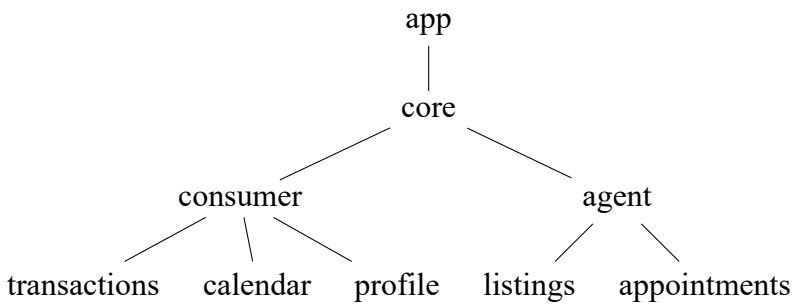
AngularJS je u svojoj srži izuzetno modularan. To omogućava programerima da logički grupiraju komponente, servise, kontrolere i ostale dijelove aplikacije u cjeline koje su lako održive i zamjenjive. Angularov sustav modula je jednostavan; na globalni `angular` objekt pozove se metoda `module()` koja prima dva argumenta, s tim da je drugi izborni.

- Prvi argument je tipa string, koji označava ime modula.

- Drugi argument je puno bitniji, i diktira ponašanje metode `module()`, a pruža se u vidu polja stringova:

    1. Ako drugi argument nije naveden, Angular će pokušati pronaći već postojeći modul te ga vratiti. Ako ga ne nađe, javlja grešku da modul ne postoji.
    2. Ako je specificiran drugi argument, Angular kreira novi modul pod imenom prvog argumenta, te definira zavisnosti tog modula u drugom argumentu – novi modul se ne može instancirati ako se sve njegove zavisnosti (drugi moduli) ne mogu instancirati. Ukoliko programer želi kreirati novi modul bez zavisnosti, potrebno je samo pružiti prazno polje, `[]`.

Uzmimo sljedeći dijagram kao primjer:



Slika 3.1: Dijagram zavisnosti modula
Izvor: Vlastita slika

Na slici 3.1 vidi se princip zavisnosti modula. Krenuvši od vrha, gdje je glavni modul ”app”, dijagram se spušta prema dolje, te linije predstavljaju vezu zavisnosti (X zavisi od Y, koji zavisi od Z...).

Nomenklatura modula je standardno sljedeća: Počevši od druge razine (u 1-indeksiranoj okolini), ako je P roditelj, a C dijete, svaki zavisni modul dobiva prefiks sintakse

$$P_{name}.C_{name}$$

Primjerice: *transactions*, od kojeg zavisi *consumer*, od kojeg zavisi *core* bi dobio ime *core.consumer.transactions*

Kada se u AngularJS-u pišu kontroleri, tvornice, direktive, servisi, itd., moraju se registrirati na određeni modul, što poboljšava modularnost i održivost kôda.

3.3.2. Rute

Kod pisanja AngularJS web aplikacije, radi se o jednostraničnoj aplikaciji koja, ovisno o URL-u, prikazuje određeni predložak HTML-a kojim upravlja određen kontroler. Ovaj postupak daje *iluziju* tradicionalne višestranične aplikacije, dok se u stvarnosti sve odvija u istom dokumentu, na istoj stranici, gdje AngularJS sakriva i prikazuje određene predloške po potrebi.

Primjer putanja slijedi:

```
1  angular
2      .module('app')
3      .config(routeConfig);
4
5  routeConfig.$inject = ['$routeProvider'];
6
7  function routeConfig($routeProvider) {
8      $routeProvider
9          .when('/', {
10             templateUrl: 'home.tmpl.html',
11             controller: 'MainController',
12             controllerAs: 'mainVm'
13         })
14         .when('/search', {
15             templateUrl: 'search.tmpl.html',
16             controller: 'SearchController',
17             controllerAs: 'searchVm'
18         })
19         .otherwise({
20             redirectTo: '/'
21         });
22 }
```

Programski kôd 3.3: Definiranje putanja u AngularJS-u

Na temelju primjera 3.3 valja primijetiti par stvari:

1. Pošto se Angular temelji na modulima. Svaki modul je moguće konfigurirati određenim parametrima. U slučaju primjera 3.3, konfigurira se servis koji se bavi rutama, odnosno putanjama.

2. `inject()`, na retku 5 primjera 3.3, je posebna metoda koja određenoj funkciji kao ovisnosti daje naznačene servise da bi se njima okoristila. Ovo se zove obrtanje kontrole (engl. *inversion of control*). Pojednostavljeno rečeno, sve usluge se odvijaju unutar spremnika koji je zadužen za instanciranje novih servisa i pružanje ovisnosti tim servisima. To sprječava

redundanciju i greške koje nastaju kada svaki servis sam sebi generira i instancira svoje zavisnosti [3].

3. Definirana je funkcija `routeConfig()` koja prima servis koji upravlja putanjama kao argument. U toj funkciji postavljena su pravila koja diktiraju koji će se dio pokazivati ovisno u URL-u.
   Pretpostavimo da je osnovni URL stranice `http://www.mojastranica.hr`:

   (a) Metoda `when()` na `routeProvider` servisu prima dva argumenta:

      i. Podatak tipa string, koji govori na koji pod-URL se odnosi pravilo. Npr.
      `http://www.mojastranica.hr/`
      ili `http://www.mojastranica.hr/search`

      ii. Objekt koji sadrži par stavki, primarno putanju prema HTML predlošku koji se koristi kada korisnik dođe na taj URL, kontroler koji upravlja tim HTML-om, te izborno pseudonim (skraćeno ime) za kontroler.

   (b) Metoda `otherwise()` prima jedan argument, a to je objekt koji sadrži pravila koja stupaju na snagu ako korisnik pokuša pristupiti bilo kojem drugom URL-u osim onih koji su definirani `when()` pravilima. U primjeru 3.3, korisnika se odvodi na korijen web mjesta.

3.3.3. Kontroleri

Kontroleri, kao što je prije spomenuto, su zaduženi za upravljanje određenim djelokrugom DOM-a. Kontroleri unutar AngularJS-a imaju više zadaća, no prvotna im je da vežu određene varijable za predočaj nad kojim upravljaju, te da vrše pozive prema servisima koji posežu u ogled, dohvaćaju podatke, vrše nekakve radnje nad njima te ih vraćaju upravljalu, odnosno kontroleru, koji potom ažurira predočaj. Temeljni pojam kontrolera je prije spomenut `scope` objekt koji služi kao apstrakcija veziva između upravljala i predočaja – metode i varijable koje proširuju `scope` se mogu pozivati i pregledavati sa predočaja. Paralele se povlače s prije spomenutim kontrolama dozvole pristupa, objašnjene na stranici 9, s tim da bi javni podaci bili svi oni koji proširuju `scope` objekt, dok je sve ostalo privatno unutar upravljala.

3.3.4. Direktive

Direktive, o kojima osnovno stoji na stranici 16, je jedna od najkorištenijih vrsta usluge unutar AngularJS-a, ponajprije jer omogućava programerima da stvaraju vlastite DOM elemente, uz to

što mogu dodatnu funkcionalnost pridodavati već postojećima.

Postoji mnoštvo već postojećih direktivi, s time da su Angularove unutarnje direktive prefiksirane sa ng-. Jedne od najpoznatijih su `ng-if`, koja kao vrijednost prima izraz. Ako je izraz istinit, taj dio DOM-a se stvara i prikazuje. U suprotnom se taj dio DOM-a miče iz postojanja. Zatim postoji i `ng-hide`, koja također prima izraz koji se evaluira u boolean (vrijednost koja je istinita ili lažna). Ukoliko je izraz istinit, taj dio DOM-a se sakriva (no ne miče u potpunosti!) iz pogleda, a u suprotnom ostaje vidljiv. `ng-for`, kada se postavi na određeni DOM element, prima izraz oblika `item in items`, a može se semantički čitati kao "za svaku stavku iz zbirke, kreiraj novu instancu DOM elementa", dok se ponajprije koristi za ispisivanje listi.

No najveća fleksibilnost stoji u mogućnosti da se kreiraju vlastite direktive. Slijedi primjer štoperice:

```
1 <div class="stopwatch">
2 {{ hours }}:{{ minutes }}:{{ seconds }}
3 </div>
```

Programski kôd 3.4: HTML predložak štoperice

```
angular
    .module('app')
    .directive('stopwatch', stopwatch);

stopwatch.$inject = ['$interval'];

function stopwatch($interval) {
    return {
        restrict: 'E',
        scope: false,
        templateUrl: 'stopwatch.tmpl.html',
        link: function($scope, $element, $attributes) {
            var interval = null;
            // Vrijeme u sekundama za odbrojavanje
            var maxTime = 5000;

            calcTime();

            interval = $interval(calcTime, 1000);

            function calcTime() {
                if (!maxTime) {
                    $interval.cancel(interval);
                    return;
                }

                $scope.hours = Math.floor(maxTime / 3600);
                $scope.minutes = Math.floor((maxTime % 3600) / 60);
                $scope.seconds = Math.floor(maxTime % 60);
            }
        }
    }
}
```

Programski kôd 3.5: JavaScript segment štoperice

Na primjeru 3.4 vidi se jednostavan HTML predložak koji je teoretski spremljen pod imenom stopwatch.tmpl.html.

Primjer 3.5 sadržava registriranje direktive na modul. Bitno je napomenuti da ime direktive postaje i način na koji se kasnije koristi. U ovom slučaju, direktiva se umeće tako da se napiše `<stopwatch></stopwatch>`. Isto tako valja napomenuti da imena direktiva moraju počinjati malim slovom.

Potom slijedi funkcija koja vraća objekt koji ima nekoliko bitnih svojstava:

Svojstvo **restrict** (hrv. ograničiti) ograničava način korištenja direktive. Tip podataka je string, a moguće vrijednosti su bilo koja kombinacija idućeg:

1. 'E' – ograničava korištenje direktive na HTML element:

   ```
   <stopwatch></stopwatch>
   ```

2. 'A' – ograničava korištenje direktive na HTML atribut:

   ```
   <div stopwatch></div>
   ```

3. 'C' – ograničava korištenje direktive na CSS klasu:

   ```
   <div class="stopwatch"></div>
   ```

4. 'M' – ograničava korištenje direktive na HTML komentar:

   ```
   <!-- directive: stopwatch -->
   ```

Valja napomenuti da se preporučuje korištenje modova E i A, nikada C ili M.

scope Ovo svojstvo diktira hoće li se za direktivu stvoriti novi, izolirani djelokrug ili ne. Može primiti tri vrijednosti:

1. `false` – *De facto* izbor, vrijednost `false` ne kreira poseban djelokrug već koristi roditeljski.
2. `true` – Stvara se novi djelokrug od roditelja koji prestaje biti vezan za njega i predaje se djetetu.
3. `{...}` – Ako se pruži objekt, stvara se novi, izolirani djelokrug koji ne nasljeđuje od roditelja. U tom objektu se definira skup svojstava koja su vidljiva na djelokrugu u vidu atributa na elementu na kojemu je direktiva pozvana. Više o mogućnostima predavanja atributa na [20].

`templateUrl` je string koji sadržava relativnu ili apsolutnu putanju do predloška za direktivu.

`link()` – funkcija koja se poziva onog trenutka kad je direktiva aktivna unutar DOM-a. Toj funkciji se prosljeđuju tri argumenta: `scope` objekt, referenca na DOM element koji predstavlja direktivu, te atributi koji su vezani za nju. Jednom kad se pozove, ova funkcija se ponaša veoma slično standardnim kontrolerima. Unutar primjera 3.5, definiran je interval koji odbrojava od `maxTime` sekundi prema 0, te formule koje pretvaraju broj preostalih sekundi u sate, minute i sekunde. Taj interval se poziva svakih 1000 milisekundi, odnosno svaku sekundu. Referenca na interval je spremljena u varijablu *interval*, te onog trenutka kada brojač dođe do 0, interval se prekida i funkcija prekida sa izvršavanjem.

Direktive su jedan od primarnih načina za pisanje vlastitih funkcionalnosti unutar AngularJS-a, te vještina koju potencijalni programer mora brzo savladati. Direktive pružaju način da se pišu

malene, izolirane komponente koje je moguće koristiti na više mjesta odjednom, te ih je lako nadograđivati, micati ili izmjenjivati.

3.3.5. Servisi

Servisi su dijelovi funkcionalnosti koji nemaju predočaj – služe kao alati te su čista funkcionalnost za sebe. Iz prakse, servisi obično služe za komunikaciju sa serverom, za obradu raznih podataka, formatiranje nekih informacija i sl. Servisi su analogni tipičnim klasama u drugim programskim jezicima koji podržavaju objektno orijentirano programiranje, no ne ponašaju se kao statične klase, što znači da ih se mora instancirati. AngularJS instanciranje radi automatski, čim se deklarira zavisnost nekog kontrolera ili direktive o nekom servisu.

Slijedi primjer fiktivnog servisa za formatiranje vremena u sate, minute i sekunde iz sekundi:

```
angular
    .module('app')
    .service('TimeFormattingService', TimeFormattingService);

function TimeFormattingService() {
    this.formatTimeFromSeconds = function(stamp) {
        stamp = parseInt(stamp, 10);

        return {
            hours: Math.floor(stamp / 3600),
            minutes: Math.floor((stamp % 3600) / 60),
            seconds: Math.floor(stamp % 60)
        };
    }
}
```

Programski kôd 3.6: Jednostavan servis

Sada kada je kreiran servis, bilo kojem drugom servisu, kontroleru ili direktivi se može kao zavisnost dati novi servis u glavnim funkcijama koje ih definiraju. Primjer 3.5 se može sada napisati na sljedeći način:

```
angular
    .module('app')
    .directive('stopwatch', stopwatch);

stopwatch.$inject = ['$interval', 'TimeFormattingService'];

function stopwatch($interval, TimeFormattingService) {
    return {
        restrict: 'E',
        scope: false,
        templateUrl: 'stopwatch.tmpl.html',
        link: function($scope, $element, $attributes) {
            var interval = null;
            // Vrijeme u sekundama za odbrojavanje
             var maxTime = 5000;

            calcTime();

            interval = $interval(calcTime, 1000);

            function calcTime() {
                if (!maxTime) {
                    $interval.cancel(interval);
                    return;
                }

                var newTime =
                    TimeFormattingService
                        .formatTimeFromSeconds(--maxTime);

                $scope.hours = newTime.hours;
                $scope.minutes = newTime.minutes;
                $scope.seconds = newTime.seconds;
            }
        }
    }
}
```

Programski kôd 3.7: Nova direktiva štoperice

Vidljivo je iz primjera 3.7 da direktiva štoperice sada koristi servis za formatiranje vremena. Na ovaj način logika računanja vremena premještena je u servis, tako da direktivino upravljalo sada ne zna kako formatirati vrijeme, dok servis ne zna gdje se sve koristi formatirano vrijeme – ovo je dobar primjer labavog vezivanja [16].

Uloga servisa je dodatni sloj apstrakcije između upravljala i ogleda – mjesto da upravljala/kontroleri direktno posežu u ogled, servisi služe kao posrednici; bilo za dobavljanje podataka iz baze, obradu

nečega i sl.

3.4. Usporedba AngularJS-a s konkurencijom

Autor teksta je osobno na MOP Festu, održan u svibnju 2017. godine, govorio na izlaganju čija je tema bila poredba tri ”titana” web frameworkova. Uspoređivali su se AngularJS, React i Vue.js.

React je razvojna okolina napravljena od strane Facebooka, te je popularna među današnjim programerima. Kao i AngularJS, React je deklarativan, što znači da se fokusira na to što stvari jesu, nego na slijedove procedura. Za razliku od AngularJS-a, React ne poznaje koncepte kontrolera ili servisa, već se sve bazira na komponentama – dijelovi aplikacije slični direktivama u AngularJS-u. Međutim, React služi isključivo za izradu korisničkih sučelja, jer se ne bavi logikom aplikacije. Često se koristi uz drugu popularnu zbirku Redux.

Vue.js, čiji autor je Evan You, je veoma sličan AngularJS-u u skoro svakom pogledu. Za razliku od AngularJS-a, Vue.js ne koristi *dirty checking* za praćenje promjena, već koristi sofisticirani sustav praćenja zavisnosti kako bi utvrdio kada se koja promjena događa. Međutim, kao i React, Vue.js sadrži samo komponente, dok se servisi implementiraju kao direktive bez predloška. Za razliku od AngularJS-a, Vue.js je puno manji, sa manje funkcionalnosti, no zato je puno brži za izvođenje. Slijedi prikaz prednosti i mana svakog od okvira.

	Prednosti	**Mane**
AngularJS	Robustan Pun mogućnosti Modularan	Nezgrapan Ponekad sadrži previše mogućnosti Spor
React	Brz lagan odličan za izradu sučelja	Služi **samo** za izradu sučelja Veoma ograničavajuća licenca
Vue.js	Lagan Kompaktan Brz	Manjak mogućnosti Relativno mlad

Tablica 1: Usporedba AngularJS-a i konkurencije

3.5. Instalacija AngularJS-a i TypeScripta

3.5.1. AngularJS

AngularJS je izuzetno lagano instalirati. Štoviše, potrebno je samo kopirati poveznicu sa jednog od mnogih popularnih CDN-ova te postaviti u HTML dokument. Primjerice:

```
<!DOCTYPE html>
<html>
    <head>
        <title>AngularJS primjer</title>
    </head>
    <body>
        <h1>AngularJS</h1>
        <!-- Poveznica na CDN -->
        <script
        src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.6/angular.min.js"></script>
    </body>
</html>
```

Programski kôd 3.8: Primjer instalacije AngularJS-a

Ugrađivanje preko CDN-a je, za mnoge svrhe, najlakši i često najefikasniji način za započeti razvoj AngularJS aplikacije. Međutim, postoji i opcija za preuzimanje AngularJS-a sa službenih stranica koje se nalaze na poveznici https://code.angularjs.org/
Na toj poveznici nalaze se sve javno dostupne verzije AngularJS-a. Potrebno je preuzeti odgovarajuću .zip datoteku, otpakirati sadržaje u radni direktoriji te potom unutar HTML-a navesti unutar `<script>` elementa putanju do nove programske zbirke.

3.5.2. TypeScript

TypeScript, na žalost, nije tako jednostavno instalirati, no niti sljedeća procedura nije isuviše zamršena. TypeScript je distribuiran preko tzv. npm-a. Node.js je, prema službenoj stranici[2]:

> Node.js je JavaScript izvršna okolina temeljena na Chrome-ovom V8 JavaScript pogonitelju (engl. *engine*). Node.js koristi model temeljen na događajima (engl. *event-driven*) koji ne blokira unos i ispis (engl. *I/O*) te ga to čini laganim i efikasnim. Node.js-ov ekosustav paketa, **npm**, je najveći ekosustav zbirki otvorenog koda na svijetu [21].

[2] Slobodan prijevod autora

Zaključno iz toga, preko npm-a distribuiraju se najpopularniji paketi i programske zbirke, ponajprije za razvoj web aplikacija, među kojima se nalazi i TypeScript. Za početak, sa službenih stranica Node.js-a i npm-a potrebno je preuzeti Node.js i npm. Nakon što su programi preuzeti i instalirani, sa komandne linije (ili terminala na UNIX-baziranim sustavima poput MacOS-a i Linuxa), potrebno je pozvati naredbu `npm install -g typescript`. Unutar tog paketa uključen je i TypeScript kompajler, koji se može pozvati također iz komandne linije naredbom `tsc datoteka.ts`. Međutim, popularne razvojne okoline poput Visual Studio Code-a (koji će se koristiti u daljnjim poglavljima) ili WebStorm-a, taj proces je automatiziran da bi se povećala programerova efikasnost.

Napomena: Preko npm-a moguće je instalirati i AngularJS, naredbom
`npm install angularjs`.

U ovom poglavlju predstavljen je konačni dio slagalice, AngularJS, te na koje sve načine ovaj framework olakšava razvoj aplikacija. Slijedi primjena ovih tehnologija na primjeru web aplikacije.

4. Praktični primjer aplikacije

Većina najbitnijih mogućnosti TypeScript-a i AngularJS-a bit će demonstrirana na temelju web aplikacije koja korisniku pomaže, na temelju određenih preferenci, odabrati mehaničku tipkovnicu, točnije, koji prekidač (engl. *switch*) bi mu najbolje odgovarao.

Aplikacija se može pronaći na poveznici: https://sqa-zavrsni.firebaseapp.com

Aplikacija je zamišljena kao veoma kratak kviz od tri pitanja, na kojemu korisnik unosi svoje preference za određene aspekte, te potom aplikacija prema tim kriterijima iz baze podataka koja se nalazi na Googleovom FireBase servisu pronalazi prekidač za korisnika.

4.1. Google FireBase

Google FireBase je Google-ov servis za pohranu podataka u online bazi podataka. Ne radi se o SQL relacijskoj bazi, već o bazi koja pohranjuje podatke u obliku JSON-a, te je utoliko slična MongoDB bazi. Za potrebe ove aplikacije, na Google FireBase-u pospremljeni su podatci o prijevodima koji imaju sljedeći oblik:

```
{
    "translations": {
        "en": {
            "sectionName": {
                "stringName": "Translation"
            }
        },
        "hr": {
            "sectionName": {
                "stringName": "Prijevod"
            }
        }
    }
}
```

Programski kôd 4.1: Primjer strukture prijevoda

Na kôdu 4.1 prikazana je struktura prijevoda. Prijevodi su podijeljeni u dva pod-objekta, ”en” i ”hr” za Engleske i Hrvatske prijevode. Potom su unutar svakog ugniježđeni pod-pod-objekti koji nose ime sekcija, u primjeru označeno kao općenito `"sectionName"`, primjerice ”home”, ”answers”, ”questions”, itd., te potom svaka sekcija nosi svoje prijevode koji se nalaze pod određenim naslovom, prikazano kao par `"stringName": "Translation/Prijevod"`

Podatci o prekidačima se spremaju na sličan način. Strukturirani su tako da svaki prekidač ima svoj identifikator koji je zapravo kombinacija kriterija. Primjerice, prekidač koji za proizvodnju buke ima vrijednost 1, za taktilnost 2 te za potrebnu silu pritiska 2, njegov je identifikator 112. Polje ”switches” u objektu koji je pospremljen na Google FireBase-u sastoji se od ključeva identifikatora koji su zapravo polja objekata, jer pod jedan identifikator može spadati više prekidača. Primjerice:

```
"switches": {
    "111": [
        {
            "description": {
                "en": "Quiet, linear, soft (45g force)",
                "hr": "Tihi, linearni, lagani (45g sile)"
            },
            "img_url": "placeholder",
            "name": {
                "en": "Cherry MX Speed Silver",
                "hr": "Cherry MX Brzi Srebrni"
            }
        },
        {
            "description": {
                "en": "Quiet, linear, soft (45g force)",
                "hr": "Tihi, linearni, lagani (45g sile)"
            },
            "img_url": "placeholder",
            "name": {
                "en": "Logitech ROMER-G",
                "hr": "Logitech ROMER-G"
            }
        }
    ]
}
```

Programski kôd 4.2: Primjer strukture objekta prekidača

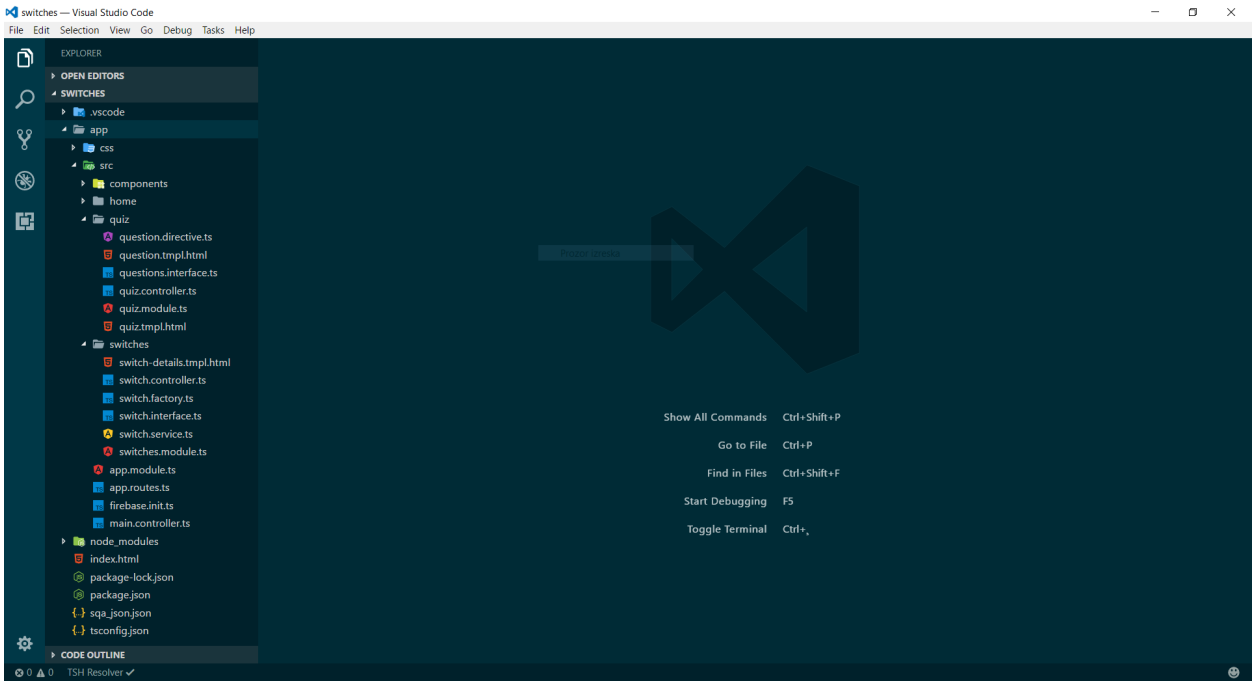
U kôdu 4.2 može se vidjeti kako svaki prekidač ima par svojstava:

- Opis, na Engleskom i Hrvatskom jeziku.
- Polje `img_url` označava putanju na sliku prekidača, radi ilustracije.
- Ime, također na Engleskom i Hrvatskom.

Ta će se svojstva koristiti na prikazu rezultata.

4.2. Razvojna okolina

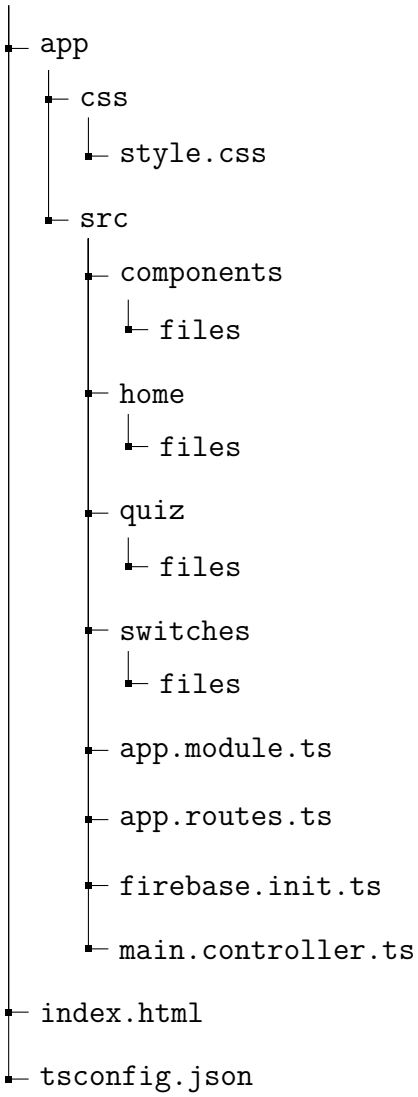
Sav razvoj aplikacije odvija se u Microsoftovom Visual Studio Code razvojnoj okolini. Visual Studio Code je besplatan program za razvoj aplikacija u više jezika, te podržava moćne mogućnosti poput inteligentnog nadopunjavanja, editiranje više redaka odjednom, preimenovanje varijabli i/ili metoda/funkcija, i slično. VS Code je besplatan te je podržan na svim popularnim desktop operativnim sustavima: Windows, MacOS i Linux.



Slika 4.1: Prikaz prozora VS Code-a
Izvor: Vlastita slika

4.3. Osnova aplikacije

Čitava struktura direktorija izgleda ovako:



Slika 4.2: Struktura direktorija
Izvor: Vlastita slika

Gdje `files` predstavlja ostatak datoteka koje se nalaze u tim poddirektorijima.

Aplikacija će se sastojati od tri predočaja, uz korijenski prikaz:

- Korijenska datoteka u kojoj je smještena direktiva koja iscrtava (engl. *renders*) ostale predočaje ovisno o trenutnoj ruti.
- Predočaj početnog stanja na kojem piše kratki uvod u mehaničke tipkovnice i zašto ih razmotriti.
- Predočaj kviza, kratke pitalice od tri pitanja koja će odrediti kriterije mehaničkih tipkovnica koje odgovaraju korisniku.
- Predočaj rezultata temeljenih na korisnikovom odabiru kriterija.

Svaki od tih predočaja stoji u vlastitom pod-direktoriju unutar projekta.

Kriteriji su razina tolerancije na zvuk, primarna upotreba tipkovnice i silina koju korisnik upotrebljava pri tipkanju. Sva tri kriterija se kreću u bodovima od 1 do 3. Pri odabiru rezultata, namjena korištenja i potrebna sila za pritisak se strogo filtriraju (odabiru se rezultati koji odgovaraju točno odabranim razinama), dok je tolerancija na zvuk slab kriterij (odabiru se svi rezultati koji odabranu razinu ili manje, npr. ako je odabrana razina 3 odabiru se rezultati s razinama 3, 2 i 1).

Prva stvar koju valja postaviti su rute. Kao što je prije spomenuto, rute su simulacije URL-ova, što znači da premda se korisniku čini da navigira prema drugim stranicama, i da se te promjene reflektiraju u adresnoj traci njihovog preglednika, korisnik u stvarnosti čitavo vrijeme ostaje na istoj stranici. Ovu simulaciju podržava činjenica da svaka ruta ima svoj HTML predložak i svoje pripadajuće upravljalo. Slijedi primjer postavljenih ruta u aplikaciji:

`routeProvider` je objekt koji je pružen u konfiguracijskoj funkciji unutar Angular-a. To je servis koji upravlja postavljanjem ruta ovisno u URL-u. U ovom slučaju, metoda `when` prima dva argumenta:

1. Podatak tipa string koji označava putanju za koju se definiraju pravila. '/' označava korijen stranice. Npr. ako je URL stranice www.mojastranica.hr, korijen bi bio upravo to. U slučaju '/test', URL bi bio www.mojastranica.hr/test.
2. Konfiguracijski objekt s nekoliko parametara koji diktiraju kako će se aplikacija ponašati kad korisnik navigira na prethodno navedenu URL putanju. Tu se ističu sljedeći parametri:
   (a) `templateUrl`: Putanja, apsolutna ili (češće) relativna do HTML predloška koji će se koristiti za taj predočaj.
   (b) `controller`: Ime upravljala koje je zaduženo za taj predočaj.

```
    $routeProvider
        .when('/', {
            templateUrl: 'app/src/home/home.tmpl.html',
            controller: 'HomeController',
            controllerAs: 'homeVm'
        })
        .when('/quiz', {
            templateUrl: 'app/src/quiz/quiz.tmpl.html',
            controller: 'QuizController',
            controllerAs: 'quizVm'
        })
        .when('/switch', {
            templateUrl: 'app/src/switches/switch-details.tmpl.html',
            controller: 'SwitchViewController',
            controllerAs: 'switchVm'
        })
        .otherwise({
            redirectTo: '/'
        });
```

Programski kôd 4.3: Rute za aplikaciju definirane u TypeScriptu

(c) `controllerAs`: sinonim za pozivanje metoda na upravljalu; umjesto `SwitchViewController.displaySwitch()` se može upotrijebiti `switchVm.displaySwitch()`

Druga metoda koja se vidi na kraju lanca metoda, `otherwise`, prima jedan argument, objekt konfiguracije sličan onome koji se šalje u prethodnoj metodi. Metoda `otherwise` označava što se događa u slučaju da korisnik navigira na bilo koji URL koji nije naveden u prethodnoj konfiguraciji.

Nakon što su putanje konfigurirane, valja postaviti osnovni HTML dokument koji će biti korijen čitave stranice:

```
<!DOCTYPE html>
<html lang="en" ng-app="app">
<head>
    <base href="/">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <style>
        body * { font-family: sans-serif; }
    </style>
    <link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro" rel="stylesheet">
    <link rel="stylesheet" href="app/css/style.css" type="text/css">
    <title>Switch Quiz App</title>
</head>
<body ng-controller="MainController">
    <select ng-model="preferredLang" ng-change="setPreferredLanguage(preferredLang)">
        <option value="hr">Hrvatski</option>
        <option value="en">English</option>
    </select>
    <div ng-view></div>
    <!-- JavaScript skripte idu ispod ove linije -->
</body>
</html>
```

Programski kôd 4.4: Osnovni HTML dokument

U programskom kodu 4.4 postavljen je kostur tipične HTML5 stranice. Prva linija, `<!DOCTYPE html>` označava vrstu dokumenta koja omogućuje preglednicima da interpretiraju dokument na točan način. Na čitavom `<html>` elementu definiran je djelokrug aplikacije u vidu direktive `ng-app="app"`. U `<head>` elementu definirane su određene meta-informacije vezane uz stranicu, kao npr. koje enkodiranje znakova se koristi (UTF-8), putanje na CSS stilove, itd.

Na `<body>` elementu postavljen je kontroler direktivom `ng-controller`, imena `MainController`. Taj kontroler sadržava opće informacije koje se tiču cijele aplikacije, primarno u vezi jezika. Naime, uvedena je mogućnost promjene jezika u bilo kojem trenutku uz pomoć izbornika. Ovo je sadržaj datoteke `main.controller.ts`:

```
namespace app {
    export interface IMainControllerScope extends ng.IScope {
        setPreferredLanguage(lang: string): void;
        getPreferredLanguage(): string;

        preferredLang: string;
    }

    export interface IMainControllerRootScope extends ng.IRootScopeService {
        preferredLang: string;
        getPreferredLanguage(): string;
    }

    class MainController implements ng.IController {
        static $inject = ['$scope', '$rootScope'];

        public constructor(private $scope: IMainControllerScope,
            private $rootScope: IMainControllerRootScope)
        {
            this.setPreferredLanguage('hr');

            this.$scope.preferredLang = 'hr';

            this.$scope.setPreferredLanguage = this.setPreferredLanguage;
            this.$rootScope.getPreferredLanguage = this.getPreferredLanguage;

            this.$scope.$watch('preferredLang', (lang: string): void => {
                this.setPreferredLanguage(lang);
            });
        }

        private setPreferredLanguage = (lang: string): void => {
            this.$rootScope.preferredLang = lang;
        }

        private getPreferredLanguage = (): string => {
            return this.$rootScope.preferredLang;
        }
    }

    angular
        .module('app')
        .controller('MainController', MainController);
}
```

Programski kôd 4.5: Sadržaj datoteke main.controller.ts

Na kodu 4.5 valja primijetiti par stvari. Prvo, čitav blok je enkapsuliran u imenski prostor

”app”, a kao što je prije spomenuto, imenski prostori i Angularovi moduli idu ruku pod ruku. Zatim, definirana su dva sučelja koja proširuju već postojeća, ona koja Angular definira. Razlog tomu je to što, ako se pokušaju pozvati vlastite metode na `scope` ili `rootScope` objekt, TypeScriptov *compiler* javlja grešku da metoda nije pronađena na tipu podatka, primjerice, `ng.IScope`. Proširivanjem postojećih sučelja sa vlastitim metodama i atributima taj problem je riješen.

Angularovi kontroleri, direktive, servisi i sl. su, u objektno orijentiranom pogledu, klase. Tako se definira klasa MainController koja implementira sučelje `ng.IController`. Ovo omogućava alatima za razvoj (npr. Visual Studio Code) da nude inteligentne preporuke i nadopunjavanja koda, ali isto tako garantira da će klasa pratiti definiciju kontrolera kakvu poznaje Angular.

Na retku 15 primjera 4.5 može se naći posebna sintaksa koja primarno služi kao zaštita protiv minifikacije koda. Naime, opća je praksa pri razvoju internetskih aplikacija rezultantni kod sažeti što je više moguće kako bi datoteka zauzimala što manje mjesta, ponajprije radi bržeg prijenosa korisniku. U tom procesu, određeni alati za smanjivane mijenjaju imena varijabli, što može rezultirati u tome da zavisnosti određenog kontrolera ili servisa dobiju drugo ime koje Angular ne prepoznaje. Injekcija imena zavisnosti u obliku stringova omogućuje Angularu da prepozna zavisnosti u slučaju da im se ime promjeni.

Konstruktori klasa su metode koje se interno pozivaju kada programer napiše kôd formata `var c = new ClassName()`. Oni su zaduženi za postavljanje parametara pri tzv. instanciranju klase, odnosno stvaranju novog objekta. U ovom slučaju, inicijaliziraju se određene postavke koje se tiču jezika. Kao što je spomenuto, glavno upravljalo je zaduženo primarno za praćene trenutno postavljenog jezika, te kao takvo sadrži metode koje s bilo kojeg djelokruga mogu uzeti trenutnu postavku ili postaviti novu. U konstruktoru klase u primjeru 4.5 vlastoručno se postavlja *promatrač* (engl. *watcher*) koji je zadužen za promatranje varijable `preferredLang`, te svaki put kada se ona promijeni pozove funkciju koja kao argument dobije novu vrijednost te potom postavi vrijednost varijable na *rootScope* djelokrugu na novozaprimljenu vrijednost.

Druga datoteka je app.module.ts, s veoma jednostavnim i kratkim sadržajem:

```
namespace app {
    angular.module('app', [
            'ngRoute',
            'app.home',
            'app.quiz',
            'app.switches',
            'app.components'
        ]);
}
```

Programski kôd 4.6: Sadržaj datoteke app.module.ts

Ova datoteka opisuje odnos modula. Moduli su objašnjeni u prijašnjim poglavljima. U ovom slučaju, definiran je modul ”app” koji će služiti kao korijen aplikacije koji je zavisan od ostalih modula koji će kasnije biti obrađeni. Također je kao zavisnost naveden modul ”ngRoute”, koji je programska zbirka zadužena za konfiguraciju prije spomenutih ruta.

Datoteka `firebase.init.ts` sadržava konfiguracijski objekt za spajanje na Google-ov FireBase servis. Na kraju se poziva metoda `firebase.initializeApp(config)`, te je potom moguće spajanje i manipulacija bazom podataka.

4.4. Predočaji

Iduća stavka su predočaji koji će se prikazivati korisniku. U primjeru 4.3 navedeni su predlošci i kontroleri zaduženi za određene rute.

4.4.1. Naslovna strana

Prvo valja definirati podmodul za naslovnu stranu:

```
module app.home {
    angular
        .module('app.home', []);
}
```

Programski kôd 4.7: Podmodul naslovne strane

Nakon toga, stvoren je kontroler koji će upravljati nad HTML predloškom:

```
1  namespace app.home {
2      interface IHomeControllerScope extends ng.IScope {
3          homeStrings: app.components.ITranslationPair;
4          ready: boolean;
5      }
6
7      class HomeController implements ng.IController {
8          static $inject = ['$log', '$scope', '$rootScope', 'TranslationService'];
9
10         public constructor(
11             protected $log: ng.ILogService,
12             private $scope: IHomeControllerScope,
13             private $rootScope: app.IMainControllerRootScope,
14             private TranslationService: app.components.ITranslationService)
15         {
16             this._init();
17         }
18
19         private _init(): void {
20             this.$log.log('HomeController initialized');
21             this._getTranslations();
22
23             this.$rootScope.$watch('preferredLang',
24                 (lang: string, oldLang: string): void => {
25                     if (lang === oldLang) return;
26                     this._getTranslations();
27                 });
28         }
29
30         private async _getTranslations(): Promise<void> {
31             this.$scope.ready = false;
32
33             const translations= await this.TranslationService
34                 .getTranslation(this.$rootScope.preferredLang, 'home');
35
36             this.$scope.homeStrings = translations;
37             this.$scope.ready = true;
38             this.$scope.$applyAsync();
39         }
40     }
41
42     angular
43         .module('app.home')
44         .controller('HomeController', HomeController);
45 }
```

Programski kôd 4.8: Kontroler za naslovnu stranu

Novitet u ovom upravljalu jest asinkrona funkcija na retku 30. Asinkrone funkcije se označuju

ključnom riječju `async` ispred imena funkcije. Asinkrone funkcije omogućuju programeru da ispred metode koja vraća tzv. *obećanje* (engl. *promise* stavi ključnu riječ `await`. Funkcija u kojoj se nalazi ta ključna riječ će pauzirati svoje izvršavanje dok se očekivano obećanje ne ostvari. Obećanja su, u programerskom svijetu, posebne vrste objekata koje služe olakšavanju radnji koje su po svojoj prirodi asinkrone, kao npr. pozivi prema serveru i sl. Nakon što je obećanje razriješeno (ispunjeno), funkcija nastavlja svojim normalnim tokom. U metodi `_getTranslations()`, koja se poziva unutar promatrača koji osmatra odabran jezik, poziva se metoda `getTranslation` na servisu za prevođenje [3]. Servis za prevođenje se spaja na Googleov FireBase (radnja koja je po svojoj prirodi asinkrona), te potom dohvaća prijevod za naslovnu stranu za trenutno odabran jezik.

Valja primijetiti da se na kraju metode `_getTranslations` poziva metoda `this.scope.applyAsync()`. Premda Angular budnim okom prati izmjene nad većinom varijabli i podataka, ne reagira na sve. Ako se dogodi promjena na koju Angular sam po sebi ne reagira, potrebno je te promjene *primijeniti* na djelokrug uz pomoć metoda `apply()` ili `applyAsync()`, s tim da `applyAsync()` ne blokira druge promjene jer se odvija asinkrono. Napose, na podmodul `app.home` registrira se novo upravljalo imena HomeController.

Predložak za ovaj predočaj izgleda ovako:

[3] O ovom servisu i ostalim komponentama bit će riječi u kasnijim poglavljima

```
<div ng-if="ready" class="container">
    <div class="main-title">
        <h1>{{ homeStrings['title'] }}</h1>
    </div>
    <section class="hero-section">
        <div class="hero-section__wrapper">
            <h2 class="hero-section__subtitle">
                {{ homeStrings['what_are_they'] }}
            </h2>
            <div class="hero-section__divider hero-section__divider--left">
                <hr/>
            </div>
            <p class="hero-section__paragraph">
                {{ homeStrings['explanation'] }}
            </p>
        </div>
    </section>
    <section class="hero-section">
        <div class="hero-section__wrapper">
            <h2 class="hero-section__subtitle">
                {{ homeStrings['why_title'] }}
            </h2>
            <div class="hero-section__divider hero-section__divider--right">
                <hr/>
            </div>
            <p class="hero-section__paragraph">
                {{ homeStrings['why_get_them'] }}
            </p>
        </div>
    </section>
    <a class="button button__quiz" href="#/quiz">
        {{ homeStrings['take_the_quiz'] }}
    </a>
</div>
```

Programski kôd 4.9: HTML predložak predočaja početne strane

Čitav HTML predložak na kôdu 4.9 prikazuje se jedino ako je varijabla `ready` istinita. Isto tako, vidljivo je da na čitavom predlošku nema izravno unesenog teksta, već se sav sadržaj ekstrapolira iz objekta `homeStrings` koji sadrži prijevode za predočaj imena "home".

4.4.2. Kviz

Prije nego se definira bilo što drugo, potrebno je registrirati podmodul:

```
namespace app.quiz {
    angular
        .module('app.quiz', []);
}
```

Predočaj kviza je kompleksniji od prethodnog predočaja. Njegov kontroler sadrži slične metode za pribavljanje prijevoda, uz razliku što se u polje stringova odvojeno spremaju prijevodi za pitanja i odgovore. Odabrane vrijednosti pitanja se spremaju u sljedeći objekt

```
private _selected: {[key: string]: number} = {
    "sound": -1,
    "force": -1,
    "usage": -1
}
```

Programski kôd 4.10: Objekt za pohranu rezultata

Inicijalno, sve vrijednosti za kategorije postavljene su na -1 radi provjere valjanosti – gumb za ”naprijed” je onemogućen dok je ijedna od tri vrijednosti jednaka -1.

```
private _isValid = (): boolean => {
    for (const category in this._selected) {
        if (this._selected[category] === -1)
            return false;
    }

    return true;
}
```

Programski kôd 4.11: Metoda za provjeru rezultata

Klikom na gumb ”dalje”, korisnika se vodi na predočaj rezultata. Točnije, pozivaju se sljedeće dvije metode:

```
private _checkResult = (): void => {
    const params: string = this._constructParamsFromSelected(this._selected);

    this.$location.url(`/switch?${params}`);
    this.$route.reload();
}

private _constructParamsFromSelected(selected: {[key: string]: number}): string {
    const params: string = `s=${selected.sound}&t=${selected.usage}&f=${selected.force}`;

    return params;
}
```

Programski kôd 4.12: Metode za konstrukciju parametara

Kao što se vidi u kôdu 4.12, prva metoda `_checkResults()` poziva drugu, `_constructParamsFromSelect`. Parametri, u ovom kontekstu, su dijelovi URL-a koji primaju format `?attribute=value&attr2=val2`, a služe za izvlačenje informacija iz URL-a. Parametri mogu biti puno duži od navedenog primjera, te premda ne postoji tehničko ograničenje dužine, preporučuje se da čitav URL ne bude duži od 2000 znakova. Nakon što metoda konstruira parametre, prva metoda preusmjeri korisnika na URL koji odgovara rezultatima kviza.

HTML predložak kviza je veoma jednostavan:

```
<h1>{{ quizStrings['intro'] }}</h1>
<question ng-repeat="q in questions track by $index" question="q"></question>
<hr>
<button type="button" ng-disabled="!isValid()" ng-click="checkResult()">
    {{quizStrings.done}}
</button>
```

Programski kôd 4.13: HTML predložak predočaja kviza

Gumb koji vodi korisnika na rezultate je, kao što je već spomenuto, onemogućen ako korisnik nije odgovorio na sva pitanja. Također valja primijetiti kako pitanja nisu ručno unesena u predložak, već se iterira po objektu u djelokrugu upravljala kviza, `questions`, te se za svako pitanje kreira nova instanca direktive.

```
1  namespace app.quiz {
2      interface IQuestionDirectiveScope extends ng.IScope {
3          selected: {[key: string]: number};
4          question: IQuestion;
5      }
6
7      class QuestionDirective implements ng.IDirective {
8          public restrict: string = 'E';
9          public templateUrl: string = 'app/src/quiz/question.tmpl.html';
10         public scope: {[boundProperty: string]: string} = {
11             question: '='
12         };
13
14         private constructor(private $log: ng.ILogService,
15             private $rootScope: ng.IRootScopeService) {}
16
17         public link = ($scope: IQuestionDirectiveScope,
18             $element: ng.IAugmentedJQuery,
19             $attributes: ng.IAttributes): void =>
20         {
21             $scope.selected = {};
22
23             $scope.$watch('selected', (s: {[key: string]: number}): void => {
24                 this.$rootScope.$emit('answerSelected', s);
25             }, true);
26         }
27
28         static instance = (): ng.IDirectiveFactory => {
29             const directive = ($log: ng.ILogService,
30                 $rootScope: ng.IRootScopeService) => new QuestionDirective($log, $rootScope);
31             directive.$inject = ['$log', '$rootScope'];
32             return directive;
33         }
34     }
35
36     angular
37         .module('app.quiz')
38         .directive('question', <any>QuestionDirective.instance());
39 }
```

Programski kôd 4.14: Direktiva pitanja

Na retku 10 kôda 4.14 je uspostavljeno dvosmjerno vezivanje objekata na djelokrug direktive pitanja. Preko atributa se direktivi pruži objekt, te se on potom veže na polje `scope.question`.

Premda je struktura direktive objašnjena u prijašnjim poglavljima, valja obratiti pozornost na retke 28 i 38 primjera 4.14. Rad s klasama uvodi određene probleme u AngularJS okvir – u ovom slučaju, nije moguće registrirati klasu kao novu direktivu, već je potrebno registrirati metodu koja

vraća instancu klase – što zapravo predstavlja direktivu. To se vidi na metodi `instance` koja vraća instancu klase, koja se potom registrira u obliku `QuestionDirective.instance()`.

Na retku 24 uvodi se pojam komunikacije između djeteta i roditelja u hijerarhiji. Na `rootScope` djelokrug emitira se događaj imena ”answerSelected”, uz koji se asocira objekt imena ”s”, koji sadrži ime odabranog kriterija te broj asociran uz taj odgovor.

HTML predložak pitanja izgleda ovako:

```
1  <div class="question">
2      <h1 class="question__title">
3          {{ question['question'] }}
4      </h1>
5      <label ng-repeat="c in question.choices track by $index">
6          <input type="radio" ng-value="c.value" ng-model="selected[c['category']]">
7          {{ c.title }}
8      </label>
9  </div>
```

Programski kôd 4.15: HTML predložak direktive pitanja

Osnova pitanja je da se iterira kroz odgovore asocirane uz pitanje, te se za svaki odgovor kreira nov natpis i element za unos tipa ”radio button”. Radio tipke su tako nazvane jer je omogućeno biranje samo jedne vrijednosti, kao što je nekoć bilo na starim radio prijemnicima.

`ng-value` atribut veže vrijednost elementa unosa za varijablu na djelokrugu direktive pitanja, tako da se odabrana vrijednost može emitirati.

4.4.3. Rezultati

Prije svega, valja definirati podmodul za prekidače:

```
1  module app.switches {
2      angular
3          .module('app.switches', []);
4  }
```

Programski kôd 4.16: Registracija podmodula za prekidače

Ovaj predočaj, u strukturi direktorija prikazanoj na strani 32 na slici 4.2 nazvan ”switches”, je zaslužan za prikazivanje rezultata upitnika nakon što korisnik klikne na gumb ”Dalje”. Najbitniji dio ovog predočaja je tzv. ”tvornica” (engl. *factory*) koja proizvodi objekte prekidača pogodnih za prikaz temeljem odabranih kriterija. *Factory* je jedan od osnovnih obrazaca, odnosno *patterna*

u objektno orijentiranom programiranju, a služi za kreiranje objekata za koje parametri unaprijed nisu poznati. Laički rečeno, ”kada treba objekt, a ne znamo koji”. Tvornica izgleda ovako:

```
namespace app.switches {
    export interface ISwitchFactory {
        getSwitches(switchId: string, lang: string): Promise<Array<ISwitch>>;
    }

    export class SwitchFactory implements ISwitchFactory {
        static $inject = ['TranslationService', '$rootScope'];

        private _firebaseDao: app.components.IFirebaseDao;

        public constructor() {
            this._firebaseDao = new app.components.FirebaseDao();
        }

        public async getSwitches(switchId: string, lang: string,
            existingData?: ISwitch[]): Promise<Array<ISwitch>> {
            const splitId: Array<string> = switchId.split('');

            const rawInfo: any[] =
                await this._firebaseDao
                    .getInformation(`switches/${switchId}`);

            const refinedData: Array<ISwitch> = rawInfo
                .map((data) => this._mapDataToSwitch(data, splitId, lang));

            let soundTolerance: number = parseInt(splitId[0], 10);

            let switchData: ISwitch[] = existingData || [];

            if (existingData)
                switchData = Array.prototype.concat(existingData, refinedData);
            else
                switchData = refinedData;

            if (soundTolerance > 1) {
                const newId: string = (--soundTolerance + '') + splitId[1] + splitId[2];
                return this.getSwitches(newId, lang, switchData);
            }

            return switchData;
        }
    }
}
```

Programski kôd 4.17: Tvornica prekidača (1/2)

Dok metoda `_mapDataToSwitch()` izgleda ovako [4]:

```
private _mapDataToSwitch(data: {[key: string]: string},
            splitId: string[], lang): ISwitch {
                return {
                    name: data['name'][lang],
                    description: data['description'][lang],
                    img_url: data['img_url'],
                    sound: parseInt(splitId[0], 10),
                    usage: parseInt(splitId[1], 10),
                    force: parseInt(splitId[2], 10)
                } as ISwitch;
}
```

Programski kôd 4.18: Tvornica prekidača (2/2)

Na primjeru 4.17, retku 24, može se vidjeti pozivanje metode `.map()` na polju `rawInfo`. `map`, u matematičkom smislu, se može predstaviti na sljedeći način:
Ako postoji funkcija

$$f(x)$$

Te skup elemenata

$$S = \{E_1, E_2, E_3, ..., E_n\}$$

Tada je funkcija mapiranja

$$M(S) = \{f(E_1), f(E_2), f(E_3), ..., f(E_n)\}, E \in S, n \geq 0$$

Odnosno, mapiranje je primjenjivanje određene funkcije na svaki element unutar nekog skupa, polja ili niza. U kôdu 4.17, preko FireBase-a se dobave podatci o svim prekidačima koji spadaju pod određeni identifikator. Potom se od tih podataka, uz pomoć metode na kôdu 4.18, ti sirovi podatci pretvore u podatke koji su pogodni za prikaz. Metoda `getSwitches` je zadužena za dobavljanje tih podataka. Isto tako, ako je kriterij tolerancije na zvuk veći od 1, tada se rekurzijom ponovno poziva ta metoda, no ovaj put se već postojeći podatci spajaju sa dodatnim podatcima.

Kontroler predočaja izgleda ovako:

[4] Kôd podijeljen na dvije strane zbog prostora

```
namespace app.switches {
    interface ISwitchViewControllerScope extends ng.IScope {
        switches: Array<ISwitch>;
    }

    class SwitchViewController implements ng.IScope {
        static $inject = ['$log', '$scope', '$rootScope', '$route', 'SwitchService'];
        private _id: string;

        public constructor(
            private $scope: ISwitchViewControllerScope,
            private $rootScope: ISwitchViewControllerRootScope,
            private $route: ng.route.IRouteService,
            private _switchService: ISwitchService)
        {
            const params: ng.route.IRouteParamsService = this.$route.current.params;
            this._id = this._constructSwitchIdFromParams(params);

            this._initView();

            this.$rootScope.$watch('preferredLang', (lang: string, oldLang: string): void => {
                    if (lang === oldLang) return;
                    this._initView();
                });
            }

            private async _initView(): Promise<void> {
                this.$scope.switches = await this._switchService
                    .getSwitchesById(this._id);
                this.$scope.$applyAsync();
            }

            private _constructSwitchIdFromParams =
            (params: ng.route.IRouteParamsService): string => {
                const sound: string     = params['s'];
                const tactility: string = params['t'];
                const force: string     = params['f'];

                return '' + sound + tactility + force;
            }
    }

    angular
        .module('app.switches')
        .controller('SwitchViewController', SwitchViewController);
}
```

Programski kôd 4.19: Upravljalo predočajem rezultata

Kao i kod prijašnjih upravljala, ovo upravljalo je zaduženo za osluškivanje promjene odabranog jezika te za ažuriranje prijevoda kad god se jezik promijeni. Zatim, iz parametara URL-a koji su poslani sa prethodnog predočaja (kviz) konstruira se identifikator, te upravljalo potom zatraži podatke o prekidačima koji odgovaraju tom identifikatoru. Međutim, pošto je TypeScript samo nadskup JavaScripta, i dalje se javljaju određeni problemi. Točnije, na retku 39, potrebno je pretvoriti prvi broj u string da bi se dobio ispravan identifikator. Tako bi u slučaju 3 + 2 + 2, da nema konkatenacije sa praznim stringom, umjesto željenog ”322” dobio broj 7. Na kraju se, naravno, registrira kontroler na podmodul.

Sve skupa povezuje servis koji nudi dvije metode za dobavljanje informacija o prekidačima:

```
1  namespace app.switches {
2      export interface ISwitchService {
3          getSwitchesById(switchId: string): Promise<Array<app.switches.ISwitch>>;
4      }
5
6      class SwitchService implements ISwitchService {
7          static $inject = ['$log', '$rootScope'];
8
9          private _switchFactory: ISwitchFactory;
10
11         public constructor(private $rootScope: app.IMainControllerRootScope) {
12             this._switchFactory = new SwitchFactory();
13         }
14
15         public async getSwitchesById(switchId: string): Promise<Array<app.switches.ISwitch>> {
16             const lang = this.$rootScope.getPreferredLanguage();
17
18             return await this._switchFactory.getSwitches(switchId, lang);
19         }
20     }
21
22     angular
23         .module('app.switches')
24         .service('SwitchService', SwitchService);
25 }
```

Programski kôd 4.20: Servis prekidača

Ovaj servis služi kao most između tvornice prekidača i ostatka aplikacije. HTML predložak predočaja rezultata izgleda ovako:

```
<div ng-repeat="switch in switches track by $index">
<h1>{{ switch['name']}}</h1>
<h2>{{ switch['description'] }}</h2>
<img ng-src="{{ switch['img_url'] }}" height="256">
</div>
```

Programski kôd 4.21: HTML predložak prekidača

Za svaki prekidač u polju prekidača, kreira se novi `<div>` element koji sadrži podatke o prekidaču.

4.5. Komponente

Komponente su dijelovi aplikacije koji ne spadaju pod nijedan od prije navedenih predočaja. U ovoj skupini se nalaze tzv. servis za prijevode i *Firebase DAO*. U objektno orijentiranom programiranju, DAO je obrazac koji sučeljava aplikaciju prema bazi podataka ili bilo kakvom drugom mehanizmu pohrane. U svojoj suštini, DAO je obrazac koji služi za apstrahiranje operacija nad bazom od ostatka aplikacije – pozivi za spremanje podataka, pretragu, brisanje i transformaciju su implementirani u tim objektima.

Redom, servis za prijevod, odnosno `translation.service.ts` izgleda ovako:

```
namespace app.components {
    export interface ITranslationService {
        getTranslation(lang: string, section: string): Promise<ITranslationPair>
    }

    class TranslationService implements ITranslationService {
        private _translationDao: IFirebaseDao;

        public constructor()
        {
            this._translationDao = new FirebaseDao();
        }

        public async getTranslation(lang: string, section: string): Promise<ITranslationPair>
        {
            const t = await this._translationDao.getInformation('translations');

            return t[lang][section];
        }
    }

    angular
        .module('app.components')
        .service('TranslationService', TranslationService);
}
```

Programski kôd 4.22: Servis za prijevod

Bitna metoda na servisu za prevođenje jest `getTranslation()`, koja se preko FireBase DAO objekta spaja na bazu te vraća prijevod za traženi jezik i sekciju.

FireBase DAO izgleda ovako:

```
namespace app.components {
    export interface IFirebaseDao {
        getInformation(section: string): Promise<any>
    }

    export class FirebaseDao implements IFirebaseDao {
        private _db: firebase.database.Database;

        public constructor() {
            this._db = firebase.database();
        }

        public async getInformation(section: string): Promise<any> {
            const data: firebase.database.Reference =
                await this._db.ref(`${section}`);

            const snapshot: firebase.database.DataSnapshot =
                await data.once('value');

            return snapshot.val();
        }
    }
}
```

Programski kôd 4.23: FireBase DAO objekt

Na kôdu 4.23 se vidi smisao iza apstrahiranja konekcije prema bazi od ostatka infrastrukture. FireBase DAO je jedini dio aplikacije koji zna kako se spojiti na FireBase te kako vratiti tražene podatke, što je vidljivo u metodi `getInformation()`, koja je također označena kao `async`.

5. Zaključak

AngularJS, uz TypeScript, pruža jedan potpuno drugačiji pogled na razvoj internetskih aplikacija. Ono što je nekoć bilo teško, nezgrapno i naporno za pisati, uz ove dvije komponente postaje puno lakše i intuitivnije. AngularJS je jedan od prvih, tzv. deklarativnih razvojnih okvira koji donosi mnoštvo preinaka u razvoj web aplikacija, na način koji je ljudima razumljiviji, te koje isto tako poboljšavaju kvalitetu kôda i čine ga lakšim za održavanje.

TypeScript je osmišljen kako bi odgovorio na primarne zamjerke JavaScripta, ponajprije nedostatak tipova podataka te loše implementiran rad s klasama. Tipovi podataka su, osim što smanjuju broj grešaka prilikom izvođenja programa, odličan alat za pomoć programeru, koji u bilo kojem trenutku zna s kakvim podatcima barata.

Iza svih softverskih alata stoji metodologija koja je začela mnoštvo njih – *Model-View-Controller*. Bez nje ništa od spomenutog u ovom radu ne bi bilo moguće. Istraženo je kako ta metodologija funkcionira te kako omogućava dugoročno održavanje sustava uz pomno planiranje i odvajanje briga.

Svi ovi čimbenici skupa čine skup alata s kojima je razvoj modernih, stabilnih, fleksibilnih i proširivih aplikacija ne samo moguć, već uvelike olakšan, te su otvorena vrata u novu, svijetliju budućnost.

Sveučilište
Sjever

SVEUČILIŠTE
SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, MARKO KLOBUČAR *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/~~ica~~ završnog/~~diplomskog~~ *(obrisati nepotrebno)* rada pod naslovom IZVEDBA MVC ARHITEKTURE PRIMJENOM AMAZARIS RAZVOJNOG OKVIRA I TYPESCRIPTA *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, MARKO KLOBUČAR *(ime i prezime)* neopozivo izjavljujem da sam suglasan/~~na~~ s javnom objavom završnog/~~diplomskog~~ *(obrisati nepotrebno)* rada pod naslovom IZVEDBA MVC ARHITEKTURE PRIMJENOM AMAZARIS RAZVOJNOG OKVIRA I TYPESCRIPTA *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)

(vlastoručni potpis)

6. Popis slika

Popis slika

1.1 Trygve Reenskaug u 2010. 2
1.2 Grafikon MVC arhitekture 3
3.1 Dijagram zavisnosti modula 18
4.1 Prikaz prozora VS Code-a 31
4.2 Struktura direktorija 32

7. Popis tablica

Popis tablica

1 Usporedba AngularJS-a i konkurencije 26

8. Popis koda

Popis programskog kôda

2.1 Primjer sintakse označavanja tipova varijabli. . . . 5
2.2 Dinamičko mijenjanje vrijednosti neovisno o tipu . . . 5
2.3 TypeScriptovo strogo provjeravanje tipa . . . 5
2.4 `any` tip u TypeScriptu . . . 6
2.5 Primjeri tipova podataka kod varijabli . . . 6
2.6 Tipovi podataka kod funkcija . . . 6
2.7 `var` naspram `let` . . . 7
2.8 Primjer klase u JavaScriptu . . . 8
2.9 Nedostatak dozvole pristupa u JavaScript klasama . . . 8
2.10 Primjer tipova i dozvole pristupa . . . 9
2.11 Primjer sučelja i apstraktnih klasa 1/2 . . . 10
2.12 Primjer sučelja i apstraktnih klasa 2/2 . . . 11
2.13 Primjer imenskog prostora . . . 13
2.14 Primjer funkcijskog zapisa u CoffeeScriptu . . . 13
3.1 Minimalan HTML prikaz stranice . . . 15
3.2 Minimalan TypeScript prikaz stranice . . . 15
3.3 Definiranje putanja u AngularJS-u . . . 19
3.4 HTML predložak štoperice . . . 21
3.5 JavaScript segment štoperice . . . 22
3.6 Jednostavan servis . . . 24
3.7 Nova direktiva štoperice . . . 25
3.8 Primjer instalacije AngularJS-a . . . 27
4.1 Primjer strukture prijevoda . . . 29
4.2 Primjer strukture objekta prekidača . . . 30
4.3 Rute za aplikaciju definirane u TypeScriptu . . . 34
4.4 Osnovni HTML dokument . . . 35
4.5 Sadržaj datoteke main.controller.ts . . . 36
4.6 Sadržaj datoteke app.module.ts . . . 38
4.7 Podmodul naslovne strane . . . 38
4.8 Kontroler za naslovnu stranu . . . 39
4.9 HTML predložak predočaja početne strane . . . 41
4.10 Objekt za pohranu rezultata . . . 42
4.11 Metoda za provjeru rezultata . . . 42
4.12 Metode za konstrukciju parametara . . . 43
4.13 HTML predložak predočaja kviza . . . 43
4.14 Direktiva pitanja . . . 44
4.15 HTML predložak direktive pitanja . . . 45
4.16 Registracija podmodula za prekidače . . . 45
4.17 Tvornica prekidača (1/2) . . . 46
4.18 Tvornica prekidača (2/2) . . . 47
4.19 Upravljalo predočajem rezultata . . . 48
4.20 Servis prekidača . . . 49
4.21 HTML predložak prekidača . . . 50
4.22 Servis za prijevod . . . 51
4.23 FireBase DAO objekt . . . 52

9. Popis literature

[1] Joe Marini. *Document Object Model*. McGraw-Hill, Inc., 2002. ISBN: 0072224363.

[2] Chandermani. *AngularJS by Example*. Packt Publishing, 2015. ISBN: 978-1-78355-381-5.

[3] Mohamed Fayad i Douglas C Schmidt. „Object-oriented application frameworks". *Communications of the ACM* 40.10 (1997), str. 32–38.

[4] Dennis Vera. *Software Architecture: MVC Design Pattern*. URL: https://medium.com/@dennisvera.z/software-architecture-mvc-design-pattern-ceae5d5083d7 (pogledano 30. 6. 2017).

[5] Wikipedia. *Model-View-Controller*. URL: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller (pogledano 22. 8. 2017).

[6] Wikipedia. *Trygve Reenskaug*. URL: https://en.wikipedia.org/wiki/Trygve_Reenskaug (pogledano 30. 6. 2017).

[7] Matt Gallagher. *Looking at Model-View-Controller in Cocoa*. URL: https://www.cocoawithlove.com/blog/mvc-and-cocoa.html#smalltalk-80 (pogledano 30. 6. 2017).

[8] Anonymous. *Separation of Concerns*. URL: http://deviq.com/separation-of-concerns/ (pogledano 19. 8. 2017).

[9] Bulcsú László, Zdenko Škiljan i Damir Boras. *Englezko-Hrvatski i Hrvatsko-Englezki rječnik obavjestnîčkôga nazivlja*. URL: http://govori.tripod.com/laszlo.htm (pogledano 19. 8. 2017).

[10] Wikipedia. *History of Typescript*. URL: https://en.wikipedia.org/wiki/TypeScript#History (pogledano 20. 8. 2017).

[11] Wikipedia. *Miguel de Icaza*. URL: https://en.wikipedia.org/wiki/Miguel_de_Icaza (pogledano 20. 8. 2017).

[12] Više autora. *What's the difference between using "let" and "var" to declare a variable?* URL: https://stackoverflow.com/questions/762011/whats-the-difference-between-using-let-and-var-to-declare-a-variable (pogledano 22. 8. 2017).

[13] Mozilla Developer Network. *The let statement*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let (pogledano 22. 8. 2017).

[14] ECMA International. *ECMAScript® 2015 Language Specification*. URL: http://www.ecma-international.org/ecma-262/6.0/#sec-let-and-const-declarations (pogledano 22. 8. 2017).

[15] Chris Bednarski. *What are namespaces?* URL: https://stackoverflow.com/a/3384384/5552688 (pogledano 23. 8. 2017).

[16] Wikipedia. *Loose coupling*. URL: https://en.wikipedia.org/wiki/Loose_coupling (pogledano 23. 8. 2017).

[17] Anonymous. *Declarative programming*. URL: http://wiki.c2.com/?DeclarativeProgramming (pogledano 24. 8. 2017).

[18] Wikipedia. *AngularJS*. URL: https://en.wikipedia.org/wiki/AngularJS#Development_history (pogledano 24. 8. 2017).

[19] AngularJS team. *Directives*. URL: https://docs.angularjs.org/guide/directive (pogledano 24. 8. 2017).

[20] AngularJS team. *Directive definition object*. URL: https://docs.angularjs.org/api/ng/service/%24compile#directive-definition-object (pogledano 25. 8. 2017).

[21] The Node.js Team. *Node.js*. URL: https://nodejs.org/en/ (pogledano 19. 9. 2017).

[22] Jeremy Ashkenas. *CoffeeScript*. URL: http://coffeescript.org/#top/ (pogledano 20. 9. 2017).