

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET

ZAVRŠNI RAD

IZRADA INTERAKTIVNOG TESTA U
ANGULAR OKRUŽENJU

Student:

Krešimir Sudar

Mentor:

doc.dr.sc. Ani Grubišić

Sadržaj

1.	Uvod	1
2.	<i>Single page</i> aplikacija	2
3.	Resursi potrebni za oblikovanje sadržaja.....	3
3.1	Microsoft Visual Studio – razvojno okruženje.....	3
3.1.1.	C#.....	3
3.1.2.	ASP.NET	5
3.1.3.	Entity Framework	5
3.1.4.	Web API.....	7
3.2	Alati.....	Error! Bookmark not defined.
3.2.2.	Cascade Style Sheets	10
3.2.3.	AngularJS	12
4.	Upitnik o Pythonu	14
4.1	Postavljanje aplikacije.....	14
4.2	Oblikovanje elemenata stranice	15
4.3	Datoteke	15
4.3.1.	index.cshtml	16
4.3.2.	mvcZavrshi.js	17
4.3.3.	KvizController.js.....	17
4.3.4.	Kviz.cshtml.....	19
4.3.5.	OdgovoriController.cs	21
4.3.6.	AuthHttpResponseInterceptor.js	22
4.3.7.	Style.less	22
4.4	Sučelje aplikacije.....	24
4.5	Sadržaj upitnika	26
5.	Zaključak.....	28
6.	Literatura.....	29

1. Uvod

Danas, u 21. stoljeću nalazimo se u vremenu kada je prosječnom državljaninu naprednijih zemalja nezamislivo živjeti bez interneta, bez nemogućnosti jednostavnog i brzog dohvaćanja informacija, provjeravanja istih i sl. Isto tako, od samih početaka do danas, web tehnologije su znatno napredovale. Počevši od statičnih i iz današnjeg gledišta, jako ružnih web stranica, bez mnogo sadržaja, danas imamo dostupne mnoge tehnologije kojima je moguće izraditi mnoštvo raznovrsnih aplikacija, različitih performansi. Osim što bi svaki vlasnik web stranice trebao mariti da ista bude ugodna oku, potrebno je voditi računa i o brzini iste. Tu nastupaju single page aplikacije koje nam omogućuju sav sadržaj desktop aplikacija u internet pregledniku. Iako je moguće izrađivati web aplikacije samo uz pomoć HTML-a, JavaScripta i CSS-a, postoje brojni alati koji nam omogućuju da te izrade budu jednostavnije i automatiziranije. Uz pomoć tih alata naša aplikacija može imati pristup bazi podataka, a izrada može biti olakšana zbog raznih programskih knjižnica (eng. *library*). U ovome radu je glavni zadatak kako napraviti pouzdanu aplikaciju koja će imati sve kvalitete *single page* aplikacije. Glavni problemi koje ćemo probati riješiti su spajanje aplikacije s bazom podataka, te brz i responzivan rad. Za razliku od klasičnih web stranica, cilj je da korištenjem ove aplikacije dobijete dojam rada na desktop aplikaciji.

U ovom radu biti će prikazana izrada *single page* aplikacije pomoću koje se može riješavati kviz o Pythonu koji uključuje gradivo kolegija Programiranje 1. Pri izradi ćemo koristiti Microsoft Visual Studio razvojno okruženje, AngularJS kao okruženje za JavaScript programski jezik te HTML i CSS za izgled same aplikacije. Prije izrade aplikacije prikazani su alati i okruženja koji su korišteni pri izradi. Sami rad se može podijeliti na dvije sadržajne cjeline. Prva cjelina će govoriti o alatima i okruženjima koje ćemo koristiti pri izradi. Druga cjelina će prikazati kako je točno kviz izrađen te će prikazati određene (bitnije) segmenta koda i objašnjenje istih.

2. *Single page* aplikacija

Single page (eng. *jedna stranica*) struktura web aplikacija je relativno nova pojava ali se zbog određenih prednosti vrlo brzo proširila među web programerima. Postoji nekoliko razvojnih okruženja *single page* aplikacija (kasnije u tekstu SPA) i zajednica koja praktički svakodnevno stvara nove biblioteke. Za razliku od tradicionalnih web aplikacija gdje je korisnik stvarao zahtjeve i slao ih serveru, kod SPA se ta komunikacija korisnik-server svodi na minimum jer pri prvom slanju zhtijeva, korisniku se isporučuje cijela aplikacija. [1] Dakle prvo učitavanje stranice učita sve resurse (HTML, JavaScript, CSS...) ili ih, ako je potrebno tijekom korištenja aplikacije pomoću AJAX poziva, učita i ugradi u aplikaciju. Single page aplikacije se dakle pokreću u web pregledniku na klijentu i korisniku daje osjećaj glatkosti i dinamičnosti. [2] Zbog toga nas pri korištenju SPA podsijeca na standardne desktop aplikacije. HTML se generira na klijentu pa je tranzicija glatka jer se jednom generiran HTML ne mora ponovno stvarati sve dok se sam ne promijeni. [3] Server je i dalje bitna stavka ovakvih aplikacija ali mu je olakšan posao jer je kod ovakvih aplikacija skoro sav posao usmjeren radu s podacima. SPA arhitektura je odlična za aplikacije koje zahtijevaju visoku interaktivnost i imaju mnogo manjih stanja između kojih je potreban gladak prijelaz. Vrlo dobar primjer ovakve aplikacije je Gmail. Kod SPA aplikacije imamo dvije odvojene aplikacije: klijentsku i serversku. Klijentska obavlja svu komunikaciju s korisnikom dok serverska služi uglavnom za pohranu podataka, implementaciju sigurnosti i slanje statičnih elemenata kao što su HTML, CSS, JavaScript. [21] Današnji uređaji koje koristimo za pregledavanje interneta imaju sve više RAM-a i procesorskih jezgri tako da neke poslove koje je prije morao obavljati sam server, sada rade uređaji klijenata tako da korisnici praktički sami sebi budu server za određene radnje. Ovo olakšava rad servera jer se ovdje renderiranje HTML-a prebacuje sa servera na klijenta što znači da je rad aplikacije distribuiran. Serveri samo pošalju klijentu datoteku koja se kod njega renderira dok je to u prošlosti sve radio server. Na internetu postoji dvije vrste klijenata. Mršavi (eng. *thin*) i debeli (eng. *fat/thick*). Mršavi su oni koji se u potpunosti oslanjaju na rad servera koji za njih obavlja sve aktivnosti dok su debeli klijenti SPA. Debeli klijenti su oni koji mogu obavljati radnje koje je u prošlosti obavljao server. [20]

3. Resursi potrebni za oblikovanje sadržaja

Za izradu upitnika o Pythonu izabrana je izrada single page aplikacije koja će biti realizirana u Microsoft Visual Studio okruženju koristeći C# programski jezik za back-end dio aplikacije. S front-end strane koristiti će se HTML, CSS za kreiranje i dizajniranje elemenata stranice te angular za dinamični dio.

3.1 Microsoft Visual Studio – razvojno okruženje

Visual studio (kasnije u tekstu VS) je Microsoftovo integrirano razvojno okruženje (eng. *IDE*) koje služi za razvijanje računalnih programa, web stranica, web aplikacija i web servisa. U Visual Studiu možemo imamo podršku za brojne programske jezike. Ona standardne poput C, C++/CLI (Visual C), VB.NET (Visual Basic i .NET), C# (Visual C#) i F# (od Visual Studio 2010). Podrška za dodatne jezike se može naknadno instalirati npr. Python, Ruby, Node.js i M. Također pruža podršku i za XML/XSLT, HTML/XHTML, JavaScript i CSS. Visual studio sam po sebi ne podržava niti jedan programski jezik, *solution* ili alat, nego dopušta dodavanje funkcionalnosti kao VSPaket (eng. *VSPackage*) koji kada se instalira daje funkcionalnost kao Servis (eng. *Service*) Za dodavanje podrške za programski jezik moramo koristiti poseban VSPaket znan kao jezični servis (eng. *language service*). [4]

3.1.1. C#

C# je suvremeni programski jezik stvoren od strane Microsofta kao dio njihove platforme jezika za komunikaciju s .NET-om. .NET su slojevi programa koji omogućuju lakšu izradu programa koji komuniciraju sa operacijskim sustavom (Windows u ovom primjeru). Iako C# ima korijene u C++-u, kroz vrijeme je napredovao dovoljno da razvije svoju tehniku i elemente preko kojih je razlika postala više nego očita. U današnjici većina objektno orijentiranih programskih jezika su u principu nalik jedan drugome, ali ono što čini C# posebnim je pozadina koju mu omogućuje .NET. C# sam za sebe je vrlo jednostavan jezik što ga čini savršenim za početnike i uz to ako niste sasvim sigurni u sintaksu, daje vam prijedloge što želite napisati u kodu (*intellisense*). [13] Također C# vam daje punu podršku za objektno orijentirano programiranje što je i za očekivati kod suvremenog jezika baziranog na C++-u i Javi. C# je u potpunosti opremljen za izrade velikih projekata dok je istovremeno lagan za savladati za početnike. Stvorio ga je mali tim pod vodstvom dva Microsoftova

programera, Andersa Hejlsberga i Scotta Wiltamutha. Hejlsberg je poznat po tome što je razvio TurboPascal, također popularni programski jezik. Osim intellisense koji olakšava posao početnicima postoji i razni mehanizmi koji omogućuju lakše pronalaženje grešaka. Greške sintakse se odmah prikažu, dok se programske greške prikažu pri pokretanju ili pucanju programa. Ovo omogućuje izradu pouzdanih programa koje je lako za održavati. [12]

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Primjer 1 - jednostavan C# program

Main metoda je posebna metoda u C#-u koja postoji u svakoj C# aplikaciji. To je metoda u kojoj program počinje. *Console* je objekt koji predstavlja prozor na vašem ekranu. *Console* klasa posjeduje statičnu metodu *WriteLine* kojoj možemo pristupiti instancom *konzole*, ali kroz klasu *Console*. Metodi pristupamo koristeći znak točke (*Console.WriteLine*). *WriteLine* je metoda koja prima jedan znakovni parametar, a to je ono što želimo ispisati na ekranu (u ovom slučaju "Hello World"). Primjer koji sam dao služi samo kako bih prikazao osnovnu sintaksu korištenja klasa i njihovih metoda.

U C#-u imamo mogućnosti izrade konzolnih aplikacija (eng. *console applications*), Windows aplikacija (eng. *Windows applications*), web aplikacija (eng. *web applications*) i web servisa (eng. *web services*).

3.1.2. ASP.NET

ASP.NET nam donosi novu eru stvaranja web aplikacija jer nam on dopušta korištenje u potpunosti opremljenih programskih jezika kao što su C# ili VB.NET za jednostavno stvaranje web aplikacija. Nažalost, internet je i dalje ograničen brzinom i ne koriste sve osobe isti preglednik pa ASP.NET na serveru procesira sav kod i korisniku šalje HTML. To znači da web aplikacija neće izgledati idealno kao aplikacije koje imamo na našim računalima, ali sa malo znanja i umijeća i dalje može napraviti odlične aplikacije. Iako ograničen HTML-om, ASP.NET uspijeva korisniku omogućiti OOP (Objektno Orijentirano Programiranje) na internetu. [18] ASP.NET mnogi netočno nazivaju programskim jezikom kada je on zapravo platforma napravljena na .NET-ovom (čitaj *dotnet*) CLR-u. CLR (eng *Common Language Runtime*) je osnovna komponenta Microsoftova .NET okruženja (eng. *framework*) zaslužna za izvođenje .NET aplikacija. Kako bi koristili CLR, programeri moraju napisati kod u nekom programskom jeziku (npr. C#, VisualBasic) koji pri kompajliranju .NET pretvara u CIL (eng. *Common Intermediate Language*) kojeg tada razumije operacijski sustav. Zbog CLR-a .NET developeri mogu koristiti bilo koji programski jezik (koji dopušta .NET) kako bi napravili svoju ASP.NET aplikaciju. To bi značilo da mogu koristiti C#, VisualBasic, Python, PHP, Perl kao i brojne druge dokle god postoji .NET kompajler za odabrani jezik. [19] Zbog vlastitog subjektivnog mišljenja, bogatih materijala koji se mogu pronaći na internetu i Microsoftova “nepotizma“, odabrao sam C#. C# je posebno stvoren za .NET okruženje i zbog toga je najprirodniji programski jezik za korištenje .NET-ovih značajki (eng. *features*). I zato nam često .NET djeluje zastrašujuće, ne samo da moramo poznavati okruženje, već također moramo znati i programski jezik.

3.1.3. Entity Framework

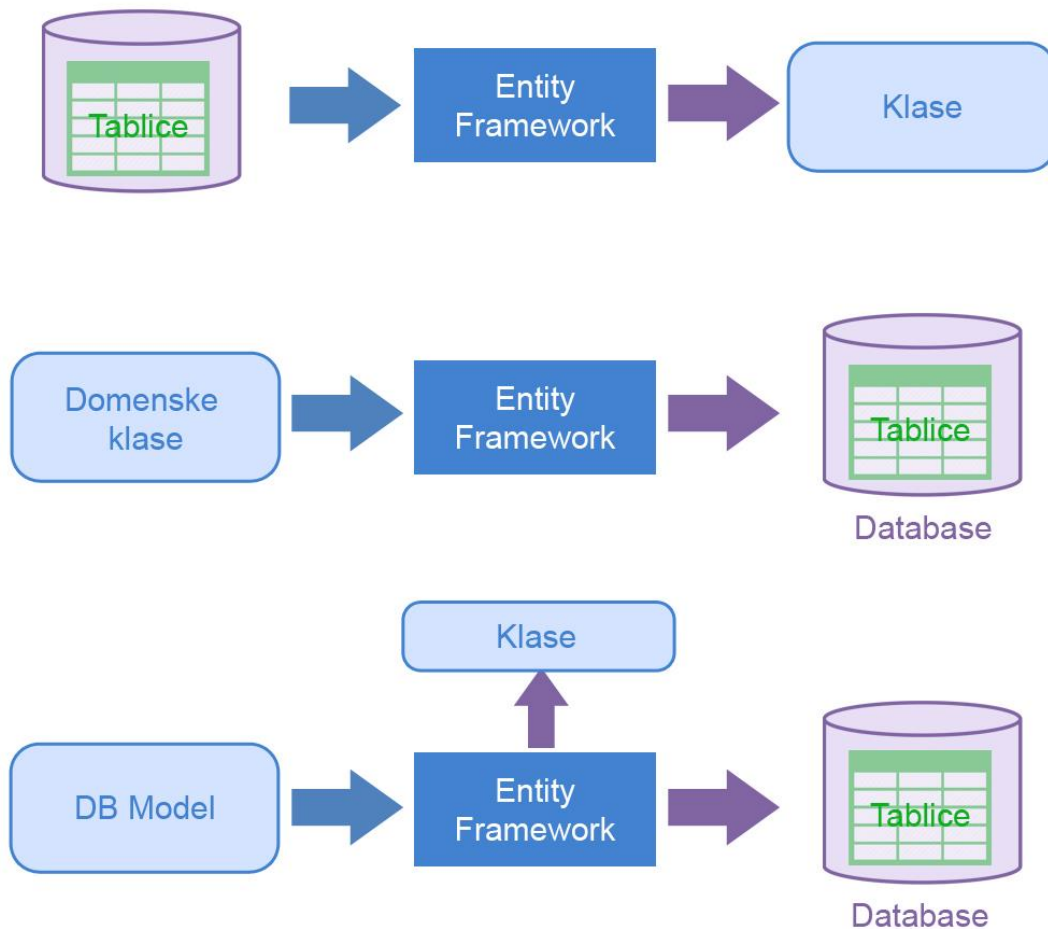
Entity framework omogućuje programeru rad s tablicama i bazama podataka preko klasa koristeći sintaksu C#-a. Objektno-relacijsko mapiranje je dosta monoton posao i zato je Microsoft razvio *entity framework* kako bi automatizirao aktivnosti vezane za baze podataka vaše aplikacije. Objektno-relacijsko mapiranje (kasnije u tekstu ORM) je alat za automatizacijsku pohranu podataka objekta domene u relacijsku bazu podataka bez pretjeranog programiranja. ORM se sastoji od tri osnovne komponente. Objekata klasa domene, relacijske baze podataka i mapiranja informacija o uputama kako se objekt domene mapira u relacijsku bazu podataka (tablice, pogleda i pohranjene procedure). ORM nam dopušta odvajanje dizajna baze podataka od dizajna klase domene što čini program održivim i

proširivim (Slika 1). Također automatizira CRUD (stvori, učitaj, izmjeni, obriši eng. *create, read, update, delete*) operacije kako developer ne treba sav kod pisati ručno.



Slika 1 - Grafički prikaz ORM-a

Entity framework je koristan u tri scenarija (Slika 2).



Slika 2 - Scenariji Entity Framework-a

Prvi je ako već imate spremnu bazu podataka ili je želite izraditi prije ostalih komponenti aplikacije, drugi je ako se želimo fokusirati na klase domene i zatim izraditi bazu podataka iz

klasa domene i na kraju treći scenarij je ako želimo izraditi bazu podataka sa vizualnim dizajnerom (eng. *visual designer*) te zatim stvoriti bazu podataka i klase. [23]

Klasa domene je klasa koja modelira podatke. Najčešće se koristi za mapiranje podataka iz baze u objekt u memoriji. *Entity framework* je okruženje za ADO.NET (eng. *ActiveX Data Objects*), Microsoftovo sučelje namjenskih programa koje omogućuje programeru Windows aplikacija pristup relacijskim i ne-relacijskim bazama podataka što Microsoftovih, što nekih drugih. Ukoliko želimo napisati program koji bi korisniku vaše web aplikacije dostavio podatke iz baze podataka, napišem ADO programske naredbe u HTML dokument koji postavimo kao aktivnu stranicu servera. Tada kada korisnik pošalje upit za stranicu s naše web aplikacije, stranica odgovora bi uključila podatke iz baze dobivene unesenim ADO kodom. [22]

3.1.4. Web API

Aplikacijsko programersko sučelje (eng. *application programming interface*, dalje u tekstu API) je skup funkcija, definicija, protkola i alata koji olakšavaju programiranje jer se možemo poslužiti radom drugih programera koji su se pridržavali istih standarda. Web API je API za web server ili za web preglednik. Ovo je koncept razvijanja weba najčešće ograničen na web aplikacije. Postoji dvije vrste, a to su web Api serverske strane (eng. *server-side*) i klijentske strane (eng. *client-side*). Serverska strana je programersko sučelje koje se sadrži od jedne ili više krajnjih točaka definiranog zahtjev-odgovor sustava poruka. Najčešće se izražava kao JSON ili XML. Klijentska strana je programersko sučelje koje proširuje funkcionalnost web preglednika ili nekog drugog HTTP klijenta. Originalno su korišteni kao ekstenzije za preglednik dok noviji ciljaju standardizirano JavaScript vezivanje. [24]

Web API podržava osnovne CRUD operacije (eng. *Create, Read, Upade, Delete*) s obzirom da radi s http radnjama kao što su uzmi, postaj, stavi, obriši (eng. *GET, POST, PUT, DELETE*).

ASP.NET web API je sučelje koje olakšava izradu HTTP servisa koji dosežu širok spektar klijenata uključujući pregledniku i mobilne uređaje. ASP.NET web API je idealan za izradu REST aplikacija u .NET okruženju.[25]

REST (eng. *Representational state transfer*) aplikacije je stil arhitekture korišten za izradu web aplikacija. Aplikacijama izrađenim korištenjem ove arhitekture je cilj brzo

izvođenje, odnosno dobre performanse, pouzdanost i sposobnost širenja, odnosno mogućnost rasta i podržavanje povećanja broja korisnika bez dodatnih poteškoća. Da bi zadovoljili ove uvijete, programeri koriste već korištene komponente kojima se može upravljati i ažurirati ih bez utjecaja na sustav koji se koristi. [26]

3.2 Dodatni alati

Kao što je već navedeno, za fronted dio upitnika biti će korišteni HTML, CSS i Angular. U sljedećim podpoglavljima će biti riječ o svakom alatu zasebno, kao i osnovama kodiranja u istima.

3.2.1. HTML

HTML (*HyperText Markup Language*) jer računalni jezik osmišljen da omogućí stvaranje web stranica. Relativno je lagan za naučiti sa osnovama koje može se mogu vrlo brzo shvatiti i vrlo je moćan s obzirom na sve što vam omogućuje da stvorite. Stalno prolazi kroz promijene kako bi zadovoljio zahtjeve korisnika interneta i W3C-a (organizacije zadužene za stvaranje i održavanje jezika). *HyperText* je metoda kojom se krećemo po internetu tako što kliknemo na poseban tekst koji zovemo hyperlink koji nas vodi na drugu stranicu. Hyper znači da se ne krećemo linearno i da u bilo kojem trenutku možemo pristupiti bilo kojoj lokaciji na internetu. *Markup* je ono što HTML tagovi čine tekstu unutar njih. Oni ga označe (eng. *Mark*) kao određenu vrstu teksta. HTML se sastoji od serije kratkih kodova (*tagova*) unesenih u tekstualni dokument od strane autora stranice. Dokument se tada spremi kao HTML tip dokumenta i otvara se u internet pregledniku. Preglednik pročita tekst i pretvori ga u vidljivu formu predstavljajući stranicu točno onako kako je autor zamislio. [5]

Glavni element HTML-a je oznaka (eng. *tag*) kojom se definira kako će se stranica prikazivati u pregledniku. Svaka oznaka može imati određene attribute pomoću kojih možemo oblikovati HTML elemente dodane od strane programera preko CSS-a (npr. ID, class). Oznake se pišu između znakova “<“ i “>“. Većina oznaka dolazi u paru te ih možemo podijeliti na početne (eng. *start tag*) i završne (eng. *end tag*). Početna se što možemo i sami zaključiti nalazi ispred sadržaja koji se nalazi u njoj, dok se završna nalazi iza. Razlikujemo ih također i po tome što završna oznaka ima znak “/“ unutar zagrada ali ispred naziva oznake. Primjer 2 prikazuje kako izgleda struktura koda za najjednostavniju web stranicu napisanu samo u HTML-u. Unutar oznake glave (eng. *head*) ide oznaka naslova (eng. *title*) i neke dodatne oznake za metapodatke koje nisu važne kod naše jednostavne html stranice, ali kod

naprednijih stranica tu se nalaze podaci koje mi kao korisnici ne vidimo, ali su bitne za rad stranice. U oznaku tijela (eng. *body*) ide sadržaj stranice što je zapravo ono što korisnik vidi. Svaki HTML dokument bi u pravilu na početku trebao imati oznaku dokumenta. U našem primjeru to je oznaka `<!DOCTYPE html>` koja nam govori da se radi o HTML5 verziji HTML-a. Kod našeg primjera oznaka dokumenta nije potrebna, ali kod naprednijih stranica ova oznaka bi imala puno veću važnost. HTML elemente možemo podijeliti na dvije osnovne vrste, block i inline.

```
<!DOCTYPE html>
<html>
<head>
  <title>Naslov stranice</title>
</head>
<body>
</body>
</html>
```

Primjer 2 - Osnovna struktura web stranice

Glavne oznake HTML-a su `html`, `head` i `body`. HTML oznaka označava početak HTML koda.

Block elementi će se stvoriti u novom redu i popuniti čitavu širinu stranice dok inline elemente možemo slagati u isti red sve dok se cijela širina stranice ne popuni. Pri izradi web stranice u HTML-u također moramo obratiti pozornost na otvaranja i zatvaranja oznaka. Ukoliko otvaramo oznaku unutar druge, prvo moramo zatvoriti oznaku koju smo zadnju otvorili.

```
<body>
<p>Sadržaj jednostavne stranice</p>
</body>
```

Primjer 3 - Ugnježdavanje elemenata

Primjetimo u primjeru 3 kako smo element paragrafa (`<p>`) otvorili unutar *body* elementa, ali smo ga zatvorili prije zatvaranja *body*-a. U početku su stranice bile u potpunosti rađene u čistom HTML-u, ali kasnije se počeo koristiti CSS radi lakšeg uređivanja izgleda stranice. Zadnja verzija CSS-a je CSS3.

3.2.2. Cascade Style Sheets

Cascading Style Sheets (kasnije u tekstu CSS) se koriste kako bi kontrolirali kako se naša web stranica prikazuje i kako bi stranice bile dostupnije. CSS je uobičajen način definiranja prezentacije vaše HTML stranice. Od fonta i boja pa do potpunog izgleda stranice. Puno je efikasnije koristiti CSS i definirati određene stvari jednom već na svakoj HTML stranici definirati izgled. [6] To znači da trebamo pisati manje koda pri stvaranju stranice, izgled stranice je isti kroz sve HTML stranice koje koriste isti CSS, unaprijeđivanje naše web stranice je lakše jer moramo napraviti samo promjenu u CSS-u i tako smanjujemo šansu za greške koje se mogu dogoditi mjenjanjem svakog HTML-a posebno. Dobro napisani CSS poboljšava pristupačnost web sadržaju dopuštajući prikaz na različitim uređajima. CSS dokumenti su kaskadni (eng. *cascading*) iz dva razloga. Jedan je taj da jedan CSS dokument može utjecati na nekoliko HTML stranica, a drugi je taj da na jednu HTML stranicu može utjecati više CSS dokumenata. Postoje tri načina za implementaciju CSS dokumenta na vašu stranicu. Koristite jedan CSS za sve vaše stranice, integrirajte CSS naredbe u *head tag* svakog HTML dokumenta i koristite style atribut da stavite CSS kod direktno u HTML element. CSS dopušta da koristite sva tri načina u isto vrijeme [7]. U CSS-u koristimo selektore (eng. *selectors*) kako bi odabrali element ili elemente koje želimo stilizirati.

Postoji više vrsta selektora. Univerzalan selektor (eng. *universal*) se koristi za uređivanje svih elemenata HTML dokumenta što znači da sve linije koda koje se upisane unutar njega, utječu na sve elemente stranice. Deklarira se znakom zvijezdice (*). Selektor vrste elementa (eng. *element type selector*) se koristi za uređivanje jednog ili više HTML elemenata istog naziva što znači da pri deklaraciji ovog selektora samo napišemo ime HTML oznake (npr. p). Identifikacijski selektor (eng. *ID selector*) se koristi za uređivanje svih elemenata koji su definirani nizom znakova koje odabire sam programer. Deklariramo ga tako da napišemo znak ljestve (#) i niz znakova. Selektor klase (eng. *class selector*) je najkorisniji među selektorima u CSS-u. Deklariramo ga tako da napišemo točku (.) i niz znakova. Koristi se za uređivanje svih elemenata koji su definirani tim nizom znakova. [16] Sigurno bi pomislili kako su selektor klase i identifikacijski selektor isti, ali tu dolazimo do onog bitnog. Selektor klase je najkorisniji zato što za razliku od identifikacijskog možemo jedan selektor koristiti na više elemenata, isto tako na jedan element možemo koristiti više različitih selektora klasa. [15]

Spomenut ćemo još dva relativno nova *style sheet* jezika koji olakšavaju pisanje CSS koda, a to su LESS i SASS. To su jezici koji koriste svoju sintaksu, ali se kompajliraju u CSS (eng. *precompilers*). Jedan od važnijih okruženja za izradu web stranica je Bootstrap koji posjeduje razne predloške, forme, gumbе i brojne druge komponente sučelja. Bootstrap je jedan od rijetkih front-end sučelja koji imaju ekstenziju za JavaScript. Iako Bootstrap nudi brojne mogućnosti, neće biti korišten u ovom projektu.

```
//Univerzalni selektor:
* {
    color: blue;
    font-size: 14px;
    line-height: 32px;
}

//Selektor vrste elementa:
ul {
    list-style: none;
    border: solid 2px #ccc;
}

//Identifikacijski selektor:
#container {
    width: 840px;
    margin: 1 auto;
}

//Selektor klase:
.box {
    padding: 20px;
    margin: 10px;
    width: 240px;
}
```

Primjer 4 - primjeri selektora

Pri izradi ovog projekta koristiti će se jedan definirani način dodijeljivanja imena (eng. *naming convention*) klasama HTML-a i CSS-a. Taj način je blok, element, modifikator (eng. *block, element, modifier*, kasnije u tekstu BEM). Svrha BEM-a je omogućiti programeru bolje razumijevanje HTML-a i CSS-a određenog projekta. Kod ove metodologije, blok je apstrakcija najviše razine novog elementa. Naprimjer gumb (eng. *button*, `.btn { }`). Na ovaj blok se može gledati kao da je roditelj čija djecu su elementi koje stavljamo unutar njega. Blok djeteta ili element je označen sa dvije povlake (`.btn__oznaka { }`). I na kraju

dolazimo do modifikatora koji mogu mijenjati izgled i dizajn određene komponente bez da utječu na neke ne povezane module. Modifikator označavamo sa dva minusa (`.btn--orange`). [14]

3.2.3. AngularJS

AngularJS je JavaScript okruženje (eng. *framework*) za dinamične web aplikacije kojeg je napravio Google u svrhu izrade održivih web aplikacije odgovarajuće arhitekture. Postoje dvije verzije Angulara, prva je AngularJS koji je okruženje za JavaScript, dok je druga Angular 2 koji je okruženje za TypeScript. Za izradu upitnika ću koristiti prvu verziju. Angular dopušta korištenje HTML-a kao jezik predloška (eng. *template*) i proširenje HTML sintakse kako bi jasnije i jezgrovitije izrazili komponente aplikacije. Angularovo vezivanje podataka i injektor zavisnosti (eng. *dependency injection*) vam pomažu da se riješite velikog dijela koda koji bi u protivnom morali pisati. Sve se to odvija u pregledniku što Angulara čini savršenim partnerom za bilo koju tehnologiju servera. Angular je zapravo sve ono što bi HTML bio kada bi bio dizajniran za aplikacije. [8] Angular je okruženje MVC (eng. *Model, View, Controller*) arhitekture koji definira brojne koncepte kako bi pravilno organizirali vašu web aplikaciju. Model je sloj gdje se nalaze varijable na koje utječe sam korisnik. Pogled (eng. *View*) je ono što korisnik vidi na svom ekranu i preko čega mijenja vrijednosti u modelu. Kontroler (eng. *controller*) je sloj koji upravlja korisničkim zahtjevima i modelu šalje naloge kojima ažurira njegovo stanje. Aplikacija je definirana modulima koji zavise jedan o drugome. Moduli su kontejneri (eng. *container*) za određene dijelove aplikacije. U njih spremamo kontrolere, servise, direktive i filtere. Osnovnu sintaksu HTML-a proširujemo dodavanjem direktiva (eng. *directives*) u kojima definiramo nove attribute (eng. *attributes*), oznake (eng. *tags*) i izraze (eng. *expressions*). U pozadini, radom aplikacije upravljaju kontroleri (eng. *controllers*) koje instanciramo koristeći ranije spomenute injektore zavisnosti Angular aplikacije se uglavnom oslanjaju na kontrolere kako bi kontrolirali tok podataka aplikacije. [17] Kontroler definiramo koristeći `ng-controller` direktivu. Kontroleri su JavaScript objekti koji sadrže svojstva, attribute i funkcije. Svi kontroleri prihvaćaju `$scope` kao parametar koji referira na aplikaciju ili modul kojim kontroler upravlja. Preko direktiva elementi HTML-a se uče novim načinima ponašanja. Praktički sve što koristimo kod Angulara su direktive. Iako je angular opremljen direktivama koje dobijemo uz njega, često ćemo napisati vlastite direktive koje odgovaraju našim potrebama. Direktive su u biti funkcije koje se pokreću kada se DOM (*document object model*) kompajlira. [10] Koristeći ovaj

koncept, Angular nam omogućuje jednostavnu manipulaciju DOM-om što znači da možemo stvoriti direktive koje će promijeniti ili čak stvoriti potpuno novo ponašanje u HTML-u. DOM je ono što se stvori u pregledniku kada se stranica učita. Direktive pišemo tako da prvo lociramo modul na koji želimo da direktiva utječe, zatim pozovemo funkciju `directive()`. Ova funkcija prima naziv direktive i funkciju. Direktive mogu biti atributi, klase, elementi i komentari. [11] Većina direktiva pri korištenju možemo prepoznati po slovima `ng` ispred određenog izraza. Naprimjer `ng-class` je direktiva koja nam dopušta da dinamički postavimo CSS klasu na HTML element povezujući izraz podacim klasa koje će biti dodane. Izrazi (eng. *expressions*) su JavaScript predlošci koji nam dopuštaju čitanje iz varijabli i pisanje u iste. Njih u kodu pišemo u dvostruke vitičaste zagrade (`{{izraz}}`). [9]

4. Upitnik o Pythonu

Upitnik o Pythonu je *single page* aplikacija namijenjena učenicima, studentima ili osobama koje žele provjeriti koliko dobro poznaju osnove programskog jezika Python. Ovoj aplikaciji se može pristupiti na dva načina, kao administrator te kao korisnik. Pristup administracijskom sučelju omogućava korisniku dodavanje, pregled i brisanje pitanja te uvid u statistiku najčešće netočno odgovorenih pitanja i sl.

Svakom prijavom na stranicu, korisniku se nasumično prikazuju pitanja. Koncipirana su na način da se korisniku pruža mogućnost odabira jednog od ponuđenih odgovora. Točan odgovor na svako pitanje iznosi dva boda, dok se netočnim odgovorom oduzimaju dva boda od ukupnog rezultata. Isto tako, ne odgovaranjem na pitanje korisniku se ne oduzimaju niti dodavaju bodovi. Ova web aplikacija ima dva načina pregleda, kao korisnik te kao administrator. Običan korisnik ima mogućnost odgovaranja na pitanja te uvid u rezultate nakon rješavanja testa. S druge strane, administrator ima neke dodatne mogućnosti, a to su dodavanje i brisanje pitanja.

Pitanja za ovaj upitnik pronašao sam u knjizi Rješavanje problema programiranjem u Pythonu od autora Budin, Brođanac, Markučić te Perić, a možemo ih podijeliti prema temama.

4.1 Postavljanje aplikacije

Za opciju logiranja i registriranja korištene su zadane ASP.NET-ove metode i tablice. Iz kojih ću kasnije dohvaćati potrebne podatke i tako puniti svoju bazu podataka. Kako bi imali mogućnost korištenjai Angulara, u index.cshtml dokumentu se koristi oznaka script (<script>) i dodaje se link kojivodi na skriptu (Primjer 6).

```
<script
src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.2.20/angular.js
">
</script>
<script
src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.2.20/angular-
route.js">
</script>
```

Primjer 6 - Ugrađivanje angular skripti

Također se u dokumentu bundle dodaje novi script bundle u koji se uključuju Angular kontroler dokumenti koji su potrebni za rad aplikacije. (Primjer 7) Zatim u index.cshtml se dodaje linija koda koja renderira stvoreni bundle i tako povezuje stranicu s potrebnim angular kontrolerima.(Primjer 8)

```
bundles.Add(new ScriptBundle("~/bundles/mvcZavrzni")
    .Include("~/Scripts/mvcZavrzni.js")
    .IncludeDirectory("~/Scripts/Controllers", "*.js")
    .IncludeDirectory("~/Scripts/Factories", "*.js")
);
```

Primjer 7 - Dodavanje angular kontrolera

```
@Scripts.Render("~/bundles/mvcZavrzni")
```

Primjer 8 - Linija za renderiranje bundle-a

4.2 Oblikovanje elemenata stranice

Ulaskom u aplikaciju otvara se forma za prijavu (eng. *login form*) u koju se unosi korisničko ime (eng. *user name*) „admin@admin.hr“ ukoliko se aplikaciji želi pristupiti kao administrator ili „user@user.hr“ ukoliko se želi pristupiti kao korisnik. Lozinka za obe vrste korisnika je „12345Aa!“. Nakon klika na „login“ gumb (eng. *button*) otvara se sadržaj stranice, ovisno o odabranom tipu korisnika. Dizajn stranice je čist i jednostavan. Za pozadine su korištene svijetle boje dok je tekst taman kako prilikom izrade upitnika ili rješavanja istih ne bi dolazilo do nepotrebnog zamaranja kod korisnika. U sljedećim poglavljima biti će detaljnije opisani i prikazani elementi svakog tipa stranice.

4.3 Datoteke

Upitnik o Pythonu je sačinjen od niza datoteka koje imaju ekstenzije .cshtml, .less, .js, te .cs. Aplikaciju se naravno može podijeliti na *frontend* i *backend* dio. Kada se govori o frontendu, misli se na onaj dio aplikacije koji korisnik vidi, bio on administrator ili običan korisnik, dok se pod *backend* dio podrazumijeva unos i ispis podataka iz baze. Komunikacija *backenda* i *frontenda* u *single page* aplikaciji događa se preko JavaScript Object Notation-a ili skraćeno json-a. Znači, backend dio aplikacije u json formatu šalje objekte *frontendu* koji zatim s istima upravlja i prema potrebi ih prikazuje korisnicima. Prije nego se opiše logika

rada programa biti će rečeno par riječi o dijelu *frontenda* u kojem se nalaze naši elementi stranice te način na koji su isti posloženi. U nastavku će biti riječ o nekim važnijim dokumentima projekta. Neki od njih su vezani za prikaz i izgled aplikacije (HTML, CSS), drugi imaju funkciju upravljanja rada same aplikacije (*Controller.js), dok treći tip dokumenata služi za pristupanje podacima u bazi podataka (*Controller.cs).

4.3.1. **index.cshtml**

Ovaj dokument je kostur aplikacije. Osim osnovne strukture HTML-a unutar head oznake poziva vanjske datoteke koje koristimo unutar aplikacije. Također, unutar oznake body se nalazi div unutar kojeg se koristi `ng-view` direktiva koja preko tražene rute prikazuje traženi *view*. (Primjer 9)

```
<div ng-view></div>
```

Primjer 9 - Pozivanje ng-view direktive

Znači, unutar tijela ovog html dokumenta se ne nalazi niti jedan drugi html element osim navedenog diva. Pomoću njega pozivamo sve ostale html elemente koji se ugnježđuju unutar `index.cshtml`-a, te se mijenja prema željama korisnika. Bitno je napomenuti da svi ostali `.cshtml` dokumenti ne sadrže osnovnu strukturu html-a (Primjer 2) jer se navedene datoteke ne koriste kao samostalne već su uvijek dio `index.cshtml`-a. Navedeno ugnježđivanje dokumenata napravljeno je pomoću angulara u datoteci `MvcZavršni.js`, što će biti detaljnije objašnjeno u narednom poglavlju.

4.3.2. mvcZavrzni.js

Ovaj dokument djeluje kao svojvrnsni putokaz aplikacije i za svaki kliknuti link na aplikaciji aktivira odgovarajući kontroler i vraća potrebni view.

```
$routeProvider.  
  when('/', {  
    templateUrl: '/Pages/kviz',  
    controller: 'KvizController'  
  })  
  .when('/login', {  
    templateUrl: '/Account/Login',  
    controller: 'LoginController'  
  })  
  .when('/register', {  
    templateUrl: '/Account/Register',  
    controller: 'RegisterController'  
  })  
  .when('/logout', {  
    templateUrl: '/Account/LogOut',  
    controller: 'LogOutController'  
  })  
  .when('/unos', {  
    templateUrl: '/pages/Unos',  
    controller: 'UnosController'  
  });
```

Primjer 10 - Putokaz aplikacije

Primjerice, ako se ulazom u aplikaciju otvara adresa localhost:5353, to znači da će se onda otvoriti index.cshtml dokument te će se unutar njega pozvati *template* koji je namijenjen za otvaranje početne stranice. U slučaju ove aplikacije to bi bio .cshtml dokument čiji je url **template**-a na /Account/Login (Primjer 10). Isto tako, kada je url adresa naše aplikacije localhost:5353/#/kviz tada će se unutar index.cshtml-a pozvati *template* koji sadrži html elemente unutar kojih se nalaze pitanja na koje korisnik može odgovarati.

4.3.3. KvizController.js

Možebitno i najbitniji JavaScript kontroler projekta. Barem što se tiče samog rada aplikacije. U njemu se dohvaćaju pitanja iz baze podataka uz pomoć PitanjaController.cs klase preko Web API-ja korištenjem \$http.get servisa (Primjer 11).

Također se određuje prikaz određenih elemenata stranice zadavanjem uvijeta preko definiranih funkcija koje se pozivaju u kviz.cshtml dokumentu.

Korištenjem `$scope` servisa aplikacija provjerava nalazi li se korisnik na zadnjem pitanju u kvizu i tako odlučuje hoće li prikazati gumb za iduće pitanje ili za završetak kviza. Isto tako provjerava nalazi li se korisnik na prvom pitanju kviza te odlučuje hoće li se prikazati gumb za prethodno pitanje ili ne. Točan kod koji provjerava ove uvijete biti će prikazan u poglavlju o `kviz.cshtml` dokumentu. `$scope` je objekt koji se definira u kontroleru i njemu se može pristupiti iz `.cshtml` dokumenta.

```
$http.get("/api/pitanja").then(function (response) {
    $scope.pitanja = response.data;
    $scope.pitanje = $scope.pitanja[$scope.index];
})
```

Primjer 11 - Dohvaćanje podataka iz baze podataka

U aplikaciji je `$scope` glavna veza između `.cshtml` dokumenta i njegovog kontrolera. (Primjer 12)

```
$scope.index = 0;
$scope.goNext = function () {
    $scope.index = $scope.index + 1;
    $scope.pitanje = $scope.pitanja[$scope.index];
}
$scope.goPrevious = function () {
    $scope.index = $scope.index - 1;
    $scope.pitanje = $scope.pitanja[$scope.index];
}
```

Primjer 12 - Korištenje `$scope` servisa

Unutar ovog kontrolera također se izračunava rezultat korisnika koji je riješio test te se taj rezultat dodjeljuje vrijednosti koja se korisniku ispisuje na ekranu. (Primjer 13)

```
$scope.finish = function ()
{
    for (var j = 0; j < $scope.pitanja.length; j++)
    {
        var pitanje = $scope.pitanja[j];
        for (var i = 0; i < pitanje.Odgovori.length; i++)
        {
            odgovor = pitanje.Odgovori[i];
            if (odgovor.IsChecked && odgovor.Tocan.trim() == 1)
            {
                $scope.korisnik.rezultat =
                    $scope.korisnik.rezultat + 2;
            }
            else if (odgovor.IsChecked &&
                odgovor.Tocan.trim() == 0)
            {
                $scope.korisnik.rezultat =
                    $scope.korisnik.rezultat - 2;
            }
        }
    }
    $scope.isGotov = true;
}
```

Primjer 13 - Računanje rezultata

4.3.4. Kviz.cshtml

Ovo je glavni view element aplikacije na kojem će korisnik provesti najviše vremena. Na njemu se prikazuje kviz odnosno pitanje i ponuđeni odgovor (Primjer 14) kao i gumbi za prethodno pitanje, iduće pitanje ili ukoliko se korisnik nalazi na zadnjem pitanju kviza, gumb za završetak kviza. Preko ruta u dokumentu `mvcZavršni.js` smo odredili da je za ovaj view, kontroler `KvizController.js`.

```
<div class="card__question card--distance">
    {{pitanje.Id}}. pitanje: {{ pitanje.Pitanje}}
    [{{pitanje.Bodovi}}]
</div>

<div class="card__answer card--distance" ng-repeat="odgovor in
pitanje.Odgovori">
    <input type="checkbox" ng-model="odgovor.IsChecked">
    {{odgovor.Tekst}}</input>
</div>
```

Primjer 14 - Prikaz pitanja i odgovora

Ranije spomenuti uvjeti koji određuju koji će se i hoće li se gumbi prikazivati su definirani s `ng-if` direktivama kojima je uvijet definiran preko objekta `index` kojem se pristupa kao što bi se u nekom drugom programskom jeziku prisupalo deklariranoj varijabli. U kontroleru je `index` definiran preko `$scope` servisa i sada ga se uspoređuje sa nulom i duljinom niza kako bi se znalo koji se gumbi prikazuju. Ukoliko je `index` nula, tada se ne prikazuje gumb za prethodno pitanje, ukoliko je `index` jednak duljini niza pitanja koji je također definiran u kontroleru, tada se ne prikazuje gumb za iduće pitanje, ali se tada zato prikazuje gumb za završetak kviza. (Primjer 15)

```
<div class="table__cell" ng-if="index != 0" ng-
click="goPrevious()">
  <button class="btn">Prethodno pitanje</button>
</div>

<div class="table__cell text-right" ng-if="index !=
pitanja.length-1" ng-click="goNext()">
  <button class="btn">Iduće pitanje</button>
</div>

<div class="table__cell text-right" ng-if="index ==
pitanja.length-1" ng-click="finish()">
  <button class="btn">Završi kviz</button>
</div>
```

Primjer 15 - Gumbi, pozivanje funkcija i uvijeti

Otvaranjem ovog viewa se pokreće kontroler koji je zapravo JavaScript dokument vezan za ovaj `.cshtml` dokument. I obavlja određene funkcije. Neke od funkcija se ne obavljaju pri pokretanju ove skripte nego se pozivaju u `.cshtml` dokumentu. Jedan od primjera je korištenje direktive `ng-click` u kojoj se poziva funkcija koja je definirana u kontroleru i tada se ona izvršava. (Primjer 15)

Cijeli ovaj dio koda s pitanjima, odgovorima i gumbima se nalazi unutar div elementa u koji je postavljen if uvijet koji provjerava je li korisnik gotov s rješavanjem kviza. Ukoliko nije odnosno ukoliko je vrijednost varijable `isGotov` jednaka laži (eng. *false*) tada se prikazuju pitanja i sve što ide s njima. Ukoliko je `isGotov` jednak istini (eng. *true*) tada se prikazuje dio programa koji ispisuje korisniku njegov rezultat i daje mu gumb za odjavu iz aplikacije. (Primjer 16)

```
<div ng-if="isGotov == true" class="section_card">
  <div class="card_answer card--distance">
    <label>Vaš rezultat je: {{korisnik.rezultat}}
  </div>

  <div class="table">
    <div class="table__cell text-right" ng-
click="logout()">
      <button class="btn">Odjava</button>
    </div>
  </div>
</div>
```

Primjer 16 - Prikaz rezultata

4.3.5. **OdgovoriController.cs**

Ovaj kontroler služi za dohvaćanje svih podataka iz tablice pitanja iz entiteta baze podataka. Odgovori kontroler je klasa koja nasljeđuje klasu `ApiController` i tako stvara link odnosno putanju kojom se pristupa podacima u bazi podataka. Isti takav kontroler se može napraviti za sve tablice klase, ali za potrebe aplikacije napravljeni su kontroleri samo za neke. Ovu se putanju nije direktno koristilo u kontroleru, ali se zato unutar ove klase direktno povezuje tablica pitanja i tablica odgovora tako što će tablica vraćati samo odgovore čiji `pitanje_Id` element odgovara broju koji šaljemo u metodu, tako da aplikacija dohvaćajući pitanja, dohvati i za svako pitanje njegove ponuđene odgovore. (Primjer 17)

```
// GET: api/Odgovori/5

public IQueryable<Odgovori> GetOdgovori(int id)
{
    return db.Odgovori.Where(odgovor => odgovor.Pitanje_Id ==
id);
}
```

Primjer 17 - Dodjeljivanje vrijednosti

4.3.6. AuthHttpResponseInterceptor.js

Ovaj kontroler odnosno *factory* se aktivira jedino ako se pokušava pristupiti određenom elementu aplikacije za koji se nema potrebna autorizacija. Ukoliko se to dogodi, zadani kod greške je 401 i tada ovaj kontroler reagira i šalje nas na stranicu za logiranje. (Primjer 18)

```
if (rejection.status === 401) {  
    console.log("Response Error 401", rejection);  
    $location.path('/login');  
}
```

Primjer 18 - Provjera autorizacije

4.3.7. Style.less

style.less je ime datoteke koju se koristi za definiranje svih stilova na aplikaciji. Kao što je već spomenuto na početku rada, za stiliziranje je korišten less pre-processor koji omogućava lakše, organiziranije i naprednije pisanje koda koji se prilikom spremanja kompajlira u css dokument. Na samom početku ovog dokumenta je korišten * selektor kojim se postavljaju padding i margine na 0 kako bi ih se resetiralo u svim preglednicima tj. da paddinzi i margine u svakom pregledniku ne bi drugačije izgledali. Nakon toga je korišten body tag da bi se definirao font koji će se koristiti na cijeloj aplikaciji, veličina fonta te pozadina. Nakon body taga, koristi se još jedan globalni selektor, a to je onaj za input polja, u ovom slučaju samo ako se unosi tekst. Za ostatak stilova koriste se klase koje su upotrebljene na određenim elementima. Često klase imaju ugnježdene elemente pa se stilovi koriste i za sve one stilizirane elemente koji su unutar glavne klase (Primjer 19). Kao što se može vidjeti iz primjera, uzet je dio koda koji se odnosi na stilove elemenata vezanih za unos pitanja i odgovora na pitanje. Section je glavni element koji se koristi na više elemenata stranice. Primjerice, koristi se i u zaglavlju stranice kao i u tijelu. Oznake koje se nalaze unutar elementa s klasom section mogu i ne moraju imati svoje klase. Prema BEM-u korišten je element section koji definira stilove koji se koriste na svakom elementu s istoimenom klasom, a klasa section__card se nadodaje uz navedenu klasu jedino ukoliko se radi o kartici u kojoj se nalaze pitanja. Također, unutar tog elementa ugnježdene su elementi koji se odnose na pitanja i odgovore te svaki ima svoje stilove, što se može vidjeti iz primjera. No, klasa card-distance se dodaje istim tim elementima kako bi se sljedeći element ispod njih prikazao tek 10px nakon, tj. da elementi ne budu priljubljeni jedan uz drugi. Na taj način

omogućavamo bilo kojem sljedećem card elementu da iskoristi istu klasu te sprječavamo nepotrebno ponavljanje istih stilova.

Za izradu kviza o Pythonu su korištene boje prilagođene logotipu istoimenog programskog jezika, a isto tako vodilo se računa da aplikacija bude intuitivna, jednostavna te da prevladavaju kontrasti koji ne umaraju čovjeka prilikom korištenja iste.

```
.section {
  max-width: 1000px;
  width: 90%;
  margin: 0 auto;
  overflow: hidden;

  &.section__card {
    padding: 40px;
    margin-bottom: 20px;
    background-color: white;
    border-radius: @border-radius;
    border: solid 1px @gray;

    .card__question {
      border-bottom: solid 1px @gray;
      padding-bottom: 10px;
      font-size: 16px;
      font-weight: bold;
    }

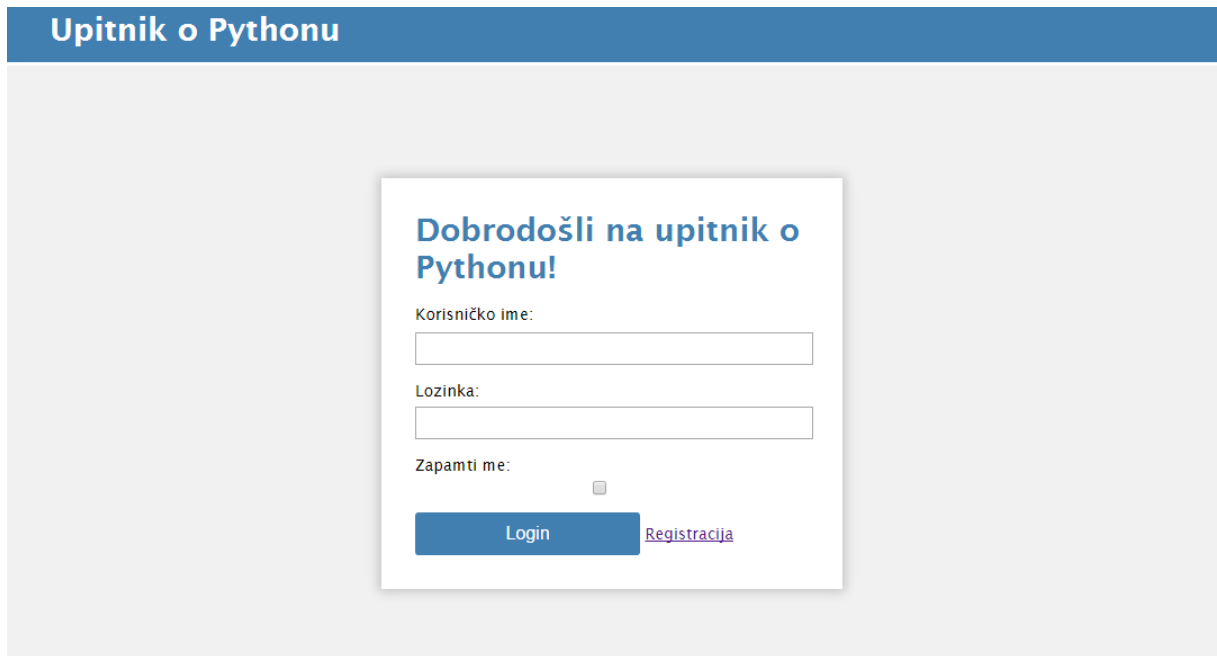
    .card__answer {
      padding: 10px;
      border: solid 1px @gray;
    }

    .card--distance {
      margin-bottom: 10px;
    }
  }
}
```

Primjer 19 - Stilizirnje elemenata unutar glavne klase

4.4 Sučelje aplikacije

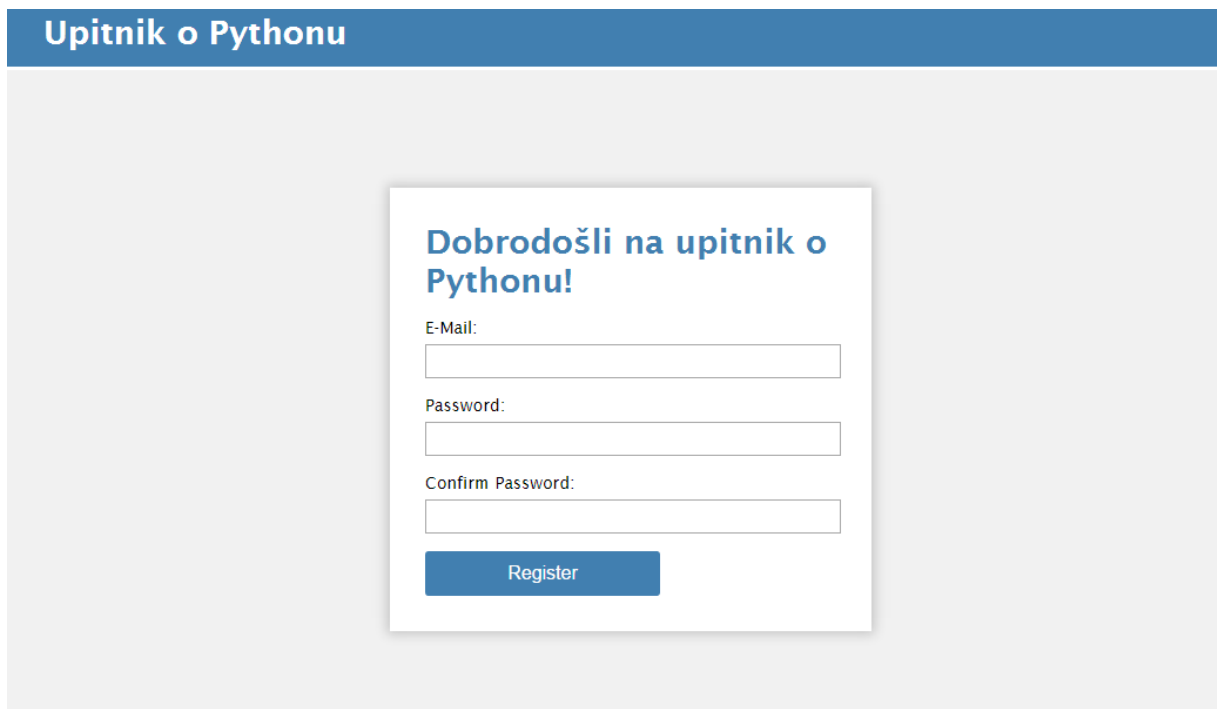
Početna stranica aplikacije je forma za logiranje s elementima za unos podataka, gumbom za logiranje (Slika 3) te linkom za registraciju (Slika 4) ukoliko korisnik nema već izrađen račun.



The screenshot shows a web page with a blue header containing the text "Upitnik o Pythonu". Below the header is a white login form with a blue border. The form contains the following elements:

- A heading: "Dobrodošli na upitnik o Pythonu!"
- A label "Korisničko ime:" followed by a text input field.
- A label "Lozinka:" followed by a text input field.
- A label "Zapamti me:" followed by a small square checkbox.
- A blue button labeled "Login".
- A purple link labeled "Registracija".

Slika 3 - Forma za logiranje



The screenshot shows a web page with a blue header containing the text "Upitnik o Pythonu". Below the header is a white registration form with a blue border. The form contains the following elements:

- A heading: "Dobrodošli na upitnik o Pythonu!"
- A label "E-Mail:" followed by a text input field.
- A label "Password:" followed by a text input field.
- A label "Confirm Password:" followed by a text input field.
- A blue button labeled "Register".

Slika 4 - Forma za registraciju

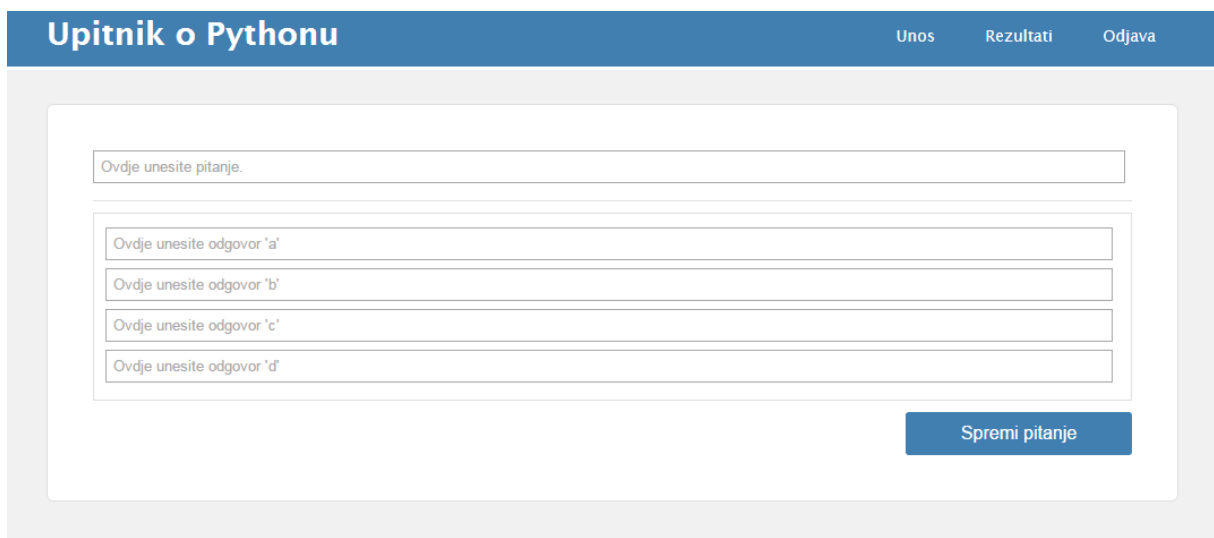
Nakon logiranja, aplikacija korisnika prosljeđuje na kviz gdje na ekranu prikazuje jedno pitanje i njegove ponuđene odgovore. (Slika 5)



The screenshot shows a quiz interface with a blue header containing the text "Upitnik o Pythonu" and two links: "Rezultati" and "Odjava". The main content area displays a question: "4. pitanje: Koji od ponuđenih izraza je ISTINIT? { a=2; b=5; c=4; } [2]". Below the question are four radio button options: "(a<b && b<c)", "(a<b && b>c)", "(a>b || b<c)", and "(a>b || a>c)". At the bottom of the question area are two blue buttons: "Prethodno pitanje" on the left and "Završi kviz" on the right.

Slika 5 - Izgled kviza

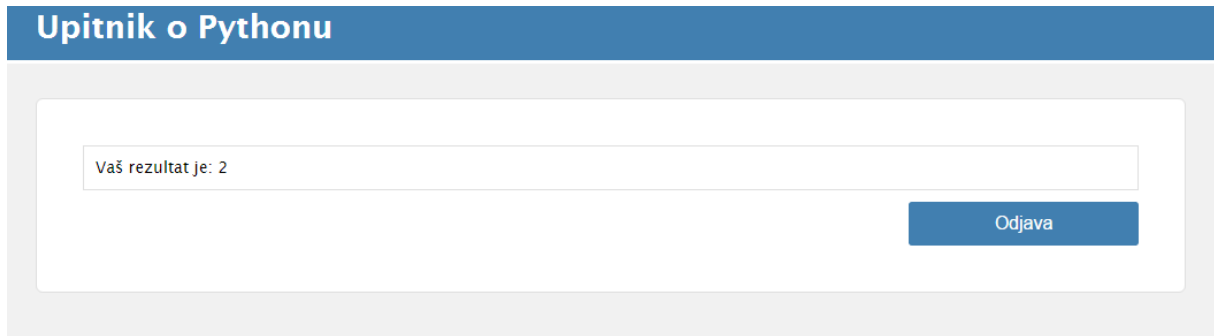
Korisnik pritiscima na gumbе navigira kroz pitanja. Odgovore koje smatra točnim korisnik će označiti klikom na gumb za označavanje (eng. *checkbox*). Administrator također ima opciju unosa pitanja u kviz. (Slika 6)



The screenshot shows a form for adding a new question. The header is blue and contains "Upitnik o Pythonu" and three links: "Unos", "Rezultati", and "Odjava". The form area has a text input field labeled "Ovdje unesite pitanje." Below it are four text input fields for answers, each labeled "Ovdje unesite odgovor 'a'", "Ovdje unesite odgovor 'b'", "Ovdje unesite odgovor 'c'", and "Ovdje unesite odgovor 'd'". A blue button labeled "Spremi pitanje" is located at the bottom right of the form area.

Slika 6 - Forma unosa pitanja

Korisnik po završetku rješavanja kviza pritišće gumb za završetak kviza koji mu na ekranu prikazuje ostvareni rezultat te gumb za odjavu. (Slika 7)



Slika 7 - Ispit rezultata

4.5 Sadržaj upitnika

Kao što je već navedeno, pitanja u ovom upitniku biti će izvučena iz knjige Rješavanje problema programiranjem u Pythonu (Budin et al), a koncipirana su na način da obuhvaćaju znanja naučena na kolegiju Programiranje 1 na Prirodoslovno – matematičkom fakultetu u Splitu. Pitanja se mogu podijeliti na nekoliko područja:

- Tipovi podataka
- Varijable
- Aritmetički izrazi
- Osnove pisanja programa
- Donošenje odluka i grananja
- Ponavljanje blokova naredbi
- Funkcije

Pitanja u ovom kvizu su napravljena kao abc pitalice. Svako pitanje može imati više točnih odgovora te kao što je već spomenuto, svaki točan odgovor nosi dva boda, dok svaki netočan nosi dva negativna boda. Kviz počinje sa jednostavnim pitanjima vezanim za tipove podataka. Cilj je provjeriti može li korisnik razlikovati tipove podatak i može li ih sa sigurnošću prepoznati. Poprilično je bitno da učenik ili student nema nedoumica oko tipova podataka kako bi ostatak gradiva bolje shvaćao. Nakon toga, slijede pitanja vezana za varijable, načinu pridruživanja vrijednosti određenom nizu znakova koje naknadno možemo upotrebljavati. Potrebno je ispitati shvaćaju li da se korištenjem varijabli rješava problem

pamćenja vrijednosti. Također, u upitniku će biti i pitanja vezana za posebne znakove za oblikovanje teksta, biti će potrebno poznavati razlike između istih što je također bitno za nastavak bavljenja programiranjem. Nakon što završe pitanja vezana za tipove podataka i varijable na red će doći ona vezana za aritmetičke izraze. Među prvim pitanjima ispitanicima su bila postavljena pitanja vezana za osnovna svojstva različitih tipova podataka te načine upisivanja i ispisa. No ulazeći u temu aritmetički izrazi dolazi trenutak za ponavljanje i osnova matematike, razlike između operanda i operatora i sl. Isto tako, pitanja su postavljena na način da provjere da li ispitanik zna razliku među operatorima i način na koji funkcioniraju. Biti će im ponuđene neki aritmetički izrazi koje će morati izračunati onako kako bi to računalo učinilo da unesemo isti niz znakova. Nakon svega navedenog, potrebno je ispitati znanja vezana za osnovna pravila pisanja programa. Pisanje imena, naredbe pridruživanja, model pohranjivanja vrijednosti u spremniku, naredbe višestrukog pridruživanja, zamjena vrijednosti varijabli su teme koje su korištene za pitanja ovog poglavlja. Zatim slijede pitanja vezana za donošenje odluka i grananja. Potrebno je provjeriti znanja vezana za if uvjete. Uz navedene uvjete postoje i pitanja vezana za relacijske operatore koje je neophodno znati za postavljanje uvjeta tijekom programiranja. Isto tako ispituje se i znanje vezano za logičke operatore te redoslijed izvođenja operacija. I na kraju su postavljena pitanja vezana za ponavljanje blokova naredbi i funkcije. Kod blokova koda dani su primjeri koda iz čega korisnik mora znati što program ispisuje. U pitanjima vezanim za funkcije će biti pitanja vezana što za sintaksu definiranja funkcije i pozivanja, isto tako će biti pitanja koja će provjeravati može li korisnik gledanjem u kod reći koju će vrijednost funkcija vratiti ili ispisati.

5. Zaključak

Najveći izazov pri izradi web aplikacije na ovaj način je taj što je potrebno poznavanje rada u nekoliko okruženja što je ovaj zadatak činilo zahtjevnijim nego što bi bio kada bi se izrađivala desktop aplikacija u jednom programskom jeziku. Pri izradi teme smo korišten je HTML, CSS, JavaScript, Angular, C# i osnovno poznavanje rada ASP.NET bazama podataka i Entity Frameworkom. Iako više od jednog okruženja otežava izradu projekata, rastavljanje aplikacije na segmente olakšava organizaciju direktorija i dokumenata aplikacije. To je upravo ono po čemu je MVC arhitektura prepoznatljiva.

Angular omogućuje da u HTML dokumentu koristimo varijable s vrijednostima koje smo definirali u skriptama povezanim s istim dokumentom. To nam uvelike olakšava rad aplikacije i čini je dinamičnom jer za razliku od statičnih web stranica ove skripte reagiraju na korisnikove unose i tako mijenjaju sadržaj stranice.

Aplikaciji koju smo napravili treba neko vrijeme da se učita ali nakon toga se sve odvija glatko i bez ikakvog zastajkivanja. To je omogućeno zbog glavne prepoznatljivost *single page* aplikacija, a to je mogućnost izmjene bilo kojeg dijela korisničkog sučelja bez da server mora slati novi HTML.

6. Literatura

- [1] M. S. Mikowski i J. C. Powell, Single Page Web Applications: JavaScript End-to-End, Shelter Island: Manning Publications, 2013
- [2] J. Papa, SPA and the single page myth, 2013
Dostupno: <http://www.johnpapa.net/pageinspa/>.
[Posljednji pristup 21. 6. 2016.]
- [3] M Watson, Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET, 2013
Dostupno: <https://msdn.microsoft.com/en-us/magazine/dn463786.aspx>
[Posljednji pristup 24.6.2016]
- [4] Anonymous, Microsoft Visual Studio
Dostupno: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio
[Posljednji pristup 25.7.2016.]
- [5] R. Shannon, What is HTML?, 2012
Dostupno: <http://www.yourhtmlsource.com/starthere/whatishtml.html>
[Posljednji pristup 27.6.2016.]
- [6] Anonymous, Stylesheets
Dostupno: <http://www.yourhtmlsource.com/stylesheets/>
[Posljednji pristup 27.6. 2016.]
- [7] R. Shannon, Introduction to CSS, 2012
Dostupno: <http://www.yourhtmlsource.com/stylesheets/introduction.html>
[Posljednji pristup 27.6.2016.]
- [8] Anonymous, What is Angular?
Dostupno: <https://docs.angularjs.org/guide/introduction>
[Posljednji pristup 22.7.2016.]
- [9] Anonymous, ngClass
Dostupno: <https://docs.angularjs.org/api/ng/directive/ngClass>
[Posljednji pristup 7.8.2016.]
- [10] Dan Wahlin, Creating Custom AngularJS Directives Part I – The Fundamentals, 2014
Dostupno: <http://weblogs.asp.net/dwahlin/creating-custom-angularjs-directives-part-i-the-fundamentals>
[Posljednji pristup 9.8.2016.]
- [11] Anonymous, Build custom directives with AngularJS
Dostupno: <http://www.ng-newsletter.com/posts/directives.html>
[Posljednji pristup 9.8.2016.]

- [12] Anonymous, Introduction to the C# Language and the .NET Framework
Dostupno: <https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx?f=255&MSPPEror=-2147217396>
[Posljednji pristup 9.8.2016.]
- [13] J. Liberty i B. McDonald, Learning C# 3.0
Dostupno: <https://msdn.microsoft.com/en-us/library/orm-9780596521066-01-01.aspx?f=255&MSPPEror=-2147217396>
[Posljednji pristup 9.8.2016.]
- [14] R. Rendle, BEM 101, 2015
Dostupno: <https://css-tricks.com/bem-101/>
[Posljednji pristup 5.8.2016.]
- [15] C. Coyier, The difference between ID and class, 2008
Dostupno: <https://css-tricks.com/the-difference-between-id-and-class/>
[Posljednji pristup 5.8.2016.]
- [16] A. Roberts, CSS Selectors, 2014
Dostupno: <https://www.sitepoint.com/web-foundations/css-selectors/>
[Posljednji pristup 5.8.2016.]
- [17] S. Begaudeau, Everything you need to understand to start with AngularJS, 2013
Dostupno: <http://stephanebegaudeau.tumblr.com/post/48776908163/everything-you-need-to-understand-to-start-with>
[Posljednji pristup 6.8.2016.]
- [18] M. Harper, What is ASP.NET?
Dostupno: <http://www.javascriptkit.com/howto/aspnet.shtml>
[Posljednji pristup 30.7.2016.]
- [19] J. McPeak, The best way to learn ASP.NET, 2011
Dostupno: <http://code.tutsplus.com/tutorials/the-best-way-to-learn-aspnet--net-22404>
[Posljednji pristup 30.7.2016.]
- [20] V. Beal, The Differences Between Thick, Thin & Smart Clients, 2006
Dostupno:
http://www.webopedia.com/DidYouKnow/Hardware_Software/thin_client_applications.asp
[Posljednji pristup: 2.9.2016.]
- [21] M. Klimushyn, Web Application Architecture from 10,000 Feet, Part 1 – Client-Side vs. Server-Side, 2015
Dostupno: <https://spin.atomicobject.com/2015/04/06/web-app-client-side-server-side/>
[Posljednji pristup: 2.9.2016.]

- [22] M. Rouse, ActiveX Data Objects (ADO), 2006
Dostupno: <http://searchsqlserver.techtarget.com/definition/ActiveX-Data-Objects>
[Posljednji pristup: 2.9.2016.]
- [23] Anonymous, What is entity framework?
Dostupno: <http://www.entityframeworktutorial.net/what-is-entityframework.aspx>
[Posljednji pristup: 2.9.2016.]
- [24] Anonymous, Web API
Dostupno: https://en.wikipedia.org/w/index.php?title=Web_API&action=history
[Posljednji pristup: 2.9.2016.]
- [25] S. Chauhan, What is web API and why to use it?, 2014
Dostupno: <http://www.dotnet-tricks.com/Tutorial/webapi/VG9K040413-What-is-Web-API-and-why-to-use-it-?.html>
[Posljednji pristup: 2.9.2016.]
- [26] Anonymous, Representational state transfer
Dostupno: https://en.wikipedia.org/wiki/Representational_state_transfer
[Posljednji pristup: 2.9.2016.]