Angular web aplikacija matematički kviz

Lončar, Max

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

ANGULAR WEB APLIKACIJA MATEMATIČKI KVIZ

Završni rad

Max Lončar

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 04.09.2020.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Max Lončar
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4088, 25.09.2019.
OIB studenta:	14433163535
Mentor:	Izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor:	Dr. sc. Tomislav Galba
Sumentor iz tvrtke:	
Naslov završnog rada:	Angular web aplikacija matematički kviz
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	04.09.2020.
Datum potvrde ocjene Odbora:	09.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 14.09.2020.

Ime i prezime studenta:	Max Lončar
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4088, 25.09.2019.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Angular web aplikacija matematički kviz**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Alfonzo Baumgartner

i sumentora Dr. sc. Tomislav Galba

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Tema ovog završnog rada je izrada Angular web aplikacije matematički kviz. Glavni zadatak ove aplikacije je omogućiti korisniku da nakon registracije, dobije pristup odabiru težine i rješavanju samog kviza. Za uspješno izvođenje aplikacije potrebno je prikupiti određena matematička pitanja različitih težina te ih pohraniti unutar baze podataka. Zatim iz te iste baze podataka se pitanja dohvaćaju i prikazuju u aplikaciji zajedno sa svim ponuđenim odgovorima od kojih je samo jedan odgovor točan.

Pri izradi ove web aplikacije mnogo pomažu znanja stečena na fakultetu iz raznih kolegija kao što su „Objektno orijentirano programiranje“, „Algoritmi i strukture podataka“, „Razvoj programske podrške objektno orijentiranim načelima“, „Baze podataka“ te kolegij „Osnove razvoja web i mobilnih aplikacija“. Aplikacija je namijenjena svim korisnicima koji žele testirati svoje matematičko znanje uz tri prisutne težine kviza. Za kvalitetnu izvedbu web aplikacije korištene su brojne tehnologije kao što su web aplikacijski okvir Angular, Microsoft SQL Server baza podataka te web proširivi okvir ASP.NET Web API s ulogom poslužitelja. Od opisnih i stilskih jezika, korišteni su HTML (engl. *Hypertext Markup Language*) i CSS (engl. *Cascading Style Sheets*).

Što se tiče strukture završnog rada, nakon uvoda dolazi teorijska podloga, tj. kratak opis tehnologije korištene za izvedbu web aplikacije. Nakon teorijske podloge, slijedi opis nastanka i postupak izrade web aplikacije počevši od baze podataka, zatim Web API-a kreiranog u Microsoft Visual Studio-u pa sve do konačnog uređenja aplikacije u Visual Studio Code-u. Na kraju prije samog zaključka, slijedi prikaz postignutih rezultata, odnosno izgled same aplikacije i njegovih komponenti.

1.1. Zadatak završnog rada

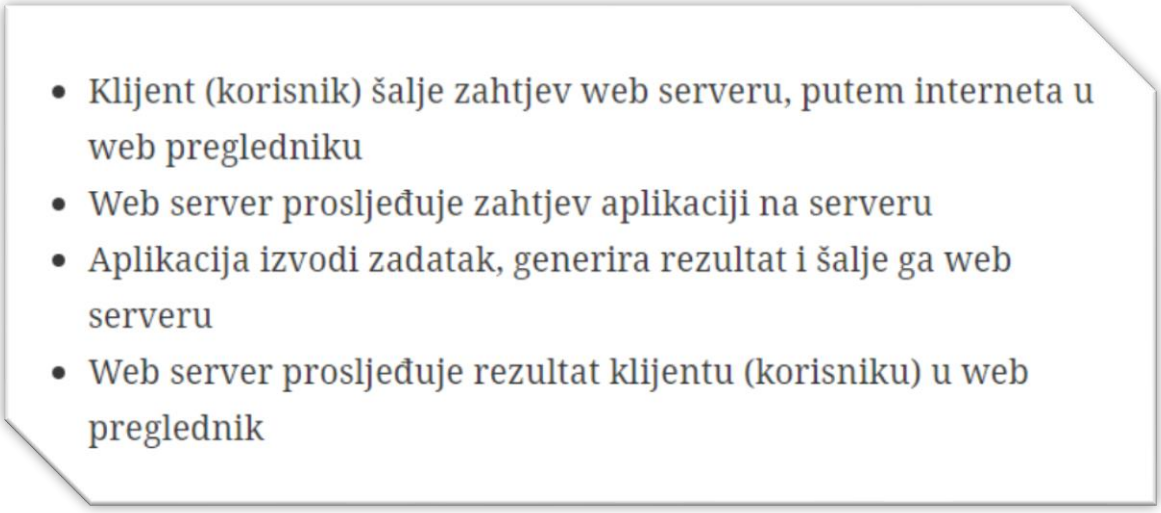
Zadatak završnog rada je napraviti Angular web aplikaciju matematički kviz s mogućnošću registracije korisnika unoseći vlastito ime i e-mail. Odabirom težine kviza, korisnik započinje rješavanje kviza odgovarajući na deset nasumično postavljenih pitanja iz baze podataka dok brojač (engl. *timer*) započinje brojati vrijeme potrebno za rješavanje samog kviza. Nakon završetka kviza, na zaslonu se prikazuje ime korisnika, njegov ostvareni rezultat, proteklo vrijeme rješavanja te točni odgovori na pitanja.

2. TEORIJSKA PODLOGA

2.1. Usporedba web i desktop aplikacije

Jedna od najraširenijih vrsta aplikacija jest web aplikacija. U usporedbi s drugim vrstama aplikacija, poput desktop aplikacija koji su pohranjeni lokalno na operacijskom sustavu (engl. *Operating System – OS*) uređaja, web aplikacija se izvodi na web serveru preko web preglednika. „Za razliku od desktop aplikacije, web aplikacija (engl. *web app*) je kompjuterski program koji se izvodi u internet pregledniku.“ [1]

Osim desktop aplikacija, tu su i mobilne aplikacije koje rade na pametnim telefonima (engl. *smartphone*) i tabletima. Sve te vrste aplikacija se mogu usporediti s web aplikacijama, no ono što ih razdvaja jest upravo princip rada. Web aplikacije rade na principu klijent-server koji je u par koraka opisan na slici **2.1.**. [2]



Sl. 2.1. *Princip rada web aplikacije* [1]

Mnoge tehnologije mogu imati ulogu servera (poslužitelja) od kojih su najpoznatiji ColdFusion, PHP, JSP, ASP te ASP.NET koji će upravo biti korišten u izradi ove web aplikacije.

„**Web aplikacija** radi na serveru, što znači da server upravlja zahtjevima i upitima samog koda aplikacije.“ [1]

Da bi radile na navedenom principu, web aplikacije moraju biti napisane/programirane u nekim od programskih jezika koje web preglednici podržavaju. Najčešće su to jezici poput JavaScript programskog jezika i HTML opisnog jezika koje web preglednik može *pročitati*. [1]

Za razliku od web aplikacija, desktop aplikacije moraju biti instalirane na samom računalu da bi mogle biti korištene. Pokretanjem same desktop aplikacije, ona koristi resurse računala. „Izvođenjem, aplikacija koristi resurse računala, prvenstveno snagu procesora, količinu radne memorije i prostor na tvrdom disku.“ [2]

Dok s druge strane, web aplikacije balansiraju upotrebu resursa tako što jedan dio uzimaju s lokalnog računala, a drugi dio sa servera s kojim u tom trenutku komuniciraju. [2]

Unatoč tome, web aplikacija može raditi i s bazom podataka. „Baza se nalazi na serveru, i svi upiti se izvode po liniji aplikacija-baza, a krajnji rezultat se isporučuje opet putem web preglednika.“ [1]

Sama web aplikacija može biti jednostavna, ali i kompleksna. Neke od kompleksnijih aplikacija su primjerice svima vrlo dobro poznati Gmail te Microsoft 365. [1]

Usporedbom web aplikacije i desktop aplikacije, može se uvidjeti da obje strane imaju svoje prednosti, ali i nedostatke. „Web aplikaciju možemo pokrenuti bez obzira koji OS koristimo (MS Windows, Linux, macOS, UNIX). Važno je imati instaliran preglednik (Edge, Internet Explorer, Google Chrome, Firefox, Opera) koji podržava izvođenje aplikacije.“ [1]

Za pokretanje web aplikacije nije bitno koji operacijski sustav se koristi. Bitan je web preglednik koji može podržati takvu aplikaciju, a velika većina korisnika koristi upravo takve preglednike. Jedna od prednosti je i kompatibilnost web aplikacija. Svi korisnici koriste istu verziju aplikacije te stoga nije moguća nekompatibilnost. Također jedna od prednosti je ta što sama web aplikacija ne zauzima prostor na disku lokalnog računala te je i samo održavanje aplikacije lakše. „Održavanje Web aplikacije lakše je, uvjetno rečeno, jer se održavanje i nadogradnje izvode na samom serveru, bez potrebe da se rade intervencije na lokalnim računalima.“ [1]

Međutim imaju web aplikacije i svojih nedostataka. Neki od nedostataka su primjerice potrebna stabilna i brza internetska povezanost, potreban dobro konfiguriran i dostupan server tijekom izvođenja aplikacije te zasigurno sigurnosni problemi aplikacije. „Sa sigurnosnog aspekta, mogli bi kazati da je server na kojem radi aplikacija izložen mogućim napadima. Nismo u

potpunosti zaštićeni ni lokalno, kada koristimo desktop aplikaciju, ali je mogućnost napada na server ipak više realna i ne ovisi samo o korisniku aplikacije.“ [1]

Iako ni u desktop aplikacijama korisnici nisu potpuno sigurni, ipak se one smatraju nešto sigurnije od web aplikacija. Ne zahtijevaju internetsku povezanost te zbog toga znaju biti brže jer ne ovise o brzini samog interneta. No također imaju i svoje nedostatke poput potrebe da se instaliraju na lokalnom računalu. Ako se radi o komercijalnoj desktop aplikaciji, moguća je potreba za aktiviranjem licence u obliku unosa ključa ili nekog drugog oblika aktivacije. [2]

2.2. Angular

Angular je web aplikacijski okvir (engl. *framework*), otvorenog koda (engl. *open source*), koji je zasnovan na TypeScript programskom jeziku. Razvijen je od strane Google-a te se i dan danas održava i nadograđuje uz pomoć pojedinih programera i tvrtki koje surađuju s Angular timom. Služi kao aplikacijski okvir i platforma za kreiranje učinkovitih jednostraničnih aplikacija uz pomoć opisnog jezika HTML te programskog jezika TypeScript. [3]

AngularJS je prva verzija Angular-a te je napisan u višem programskom jeziku JavaScript. Angular je nastao po uzoru na AngularJS s iznimkom da je pisan u TypeScript-u te službeno započinje verzijom 2.0. Od tada svaka nadolazeća verzija je bila nadogradnja na Angular 2 te se jednostavno nazivala Angular što jasno daje do znanja da su AngularJS i Angular različiti i nekompatibilni. Time se uklonila svaka zbunjenost koja je često bila prisutna kod korisnika i programera. [3]

Angular je jedan od web aplikacijskih okvira koji kreiraju jednostranične aplikacije. To znači da kreiraju web aplikacije ili web stranice koje održavaju interakciju s web preglednikom tako što dinamički izmjenjuju trenutnu web stranicu s novim podacima preuzetim sa web servera. Takav postupak omogućava bržu tranziciju za razliku od klasične metode kada preglednik učitava novu cjelokupnu web stranicu. [4]

Što se tiče programskog jezika na kojem je zasnovan Angular, TypeScript je zapravo programski jezik otvorenog koda koji je razvijen od strane Microsoft-a. Može se reći da je zapravo TypeScript nadskup JavaScript programskog jezika jer nudi više značajki poput klasa i sučelja te „statičko tipkanje“ (engl. *static typing*). To znači da programer sam može odrediti je li neka određena varijabla broj, niz (engl. *string*) ili nešto sasvim drugo. Međutim, TypeScript se ne može pokrenuti u pregledniku već se prevodi (engl. *compile*) u JavaScript. [5]

Svaka Angular aplikacija se sastoji od barem jedne komponente (engl. *component*). Glavna komponenta se naziva korijenska komponenta (engl. *root component*) koja povezuje ostale kreirane komponente. Svaka kreirana komponenta definira klasu koja sadrži podatke i logiku aplikacije te je povezana s HTML predloškom koji definira pogled same aplikacije. Također svaka komponenta sadrži i dekorater (engl. *decorator*) naziva @Component unutar kojega se nalaze meta podaci (engl. *metadata*). Meta podaci predstavljaju podatke koji pružaju informacije Angular-u što da učini s klasom unutar same komponente, odnosno informacije koje su Angular-u potrebne za stvaranje i prikaz komponenti. [6]

U primjeru na slici **2.2.** prikazan je dekorater @Component koji sadrži određene informacije:

- *selector* – HTML oznaka ili tag pomoću kojega se koristi navedena komponenta u predlošcima drugih komponenata,
- *templateUrl* – predstavlja relativnu adresu predloška komponente,
- *styleUrls* – predstavlja relativnu adresu stilskog uređenja komponente. [6]

```
@Component({
  selector: 'app-easyresult',
  templateUrl: './easyresult.component.html',
  styleUrls: ['./easyresult.component.css']
})
export class EasyresultComponent implements OnInit {
```

Sl. 2.2. *Dekorater @Component*

Arhitektura Angular aplikacije se oslanja na određene ključne pojmove. Ti osnovni pojmovi su zapravo moduli koji se jednostavno nazivaju **NgModules**. Svaka Angular aplikacija također mora sadržavati barem jedan modul, a to je najčešće korijenski modul (engl. *root module*). Taj modul se može vidjeti u primjeru na slici **2.3.** te se naziva **AppModule**. Funkcija modula je povezati različite dijelove aplikacije i funkcionalnosti u pakete, a ti dijelovi su najčešće zapravo komponente aplikacije. Moduli također sadrže dekoratere s nazivom @NgModule u kojem se nalaze svojstva koja opisuju same module. Neka od takvih svojstava su:

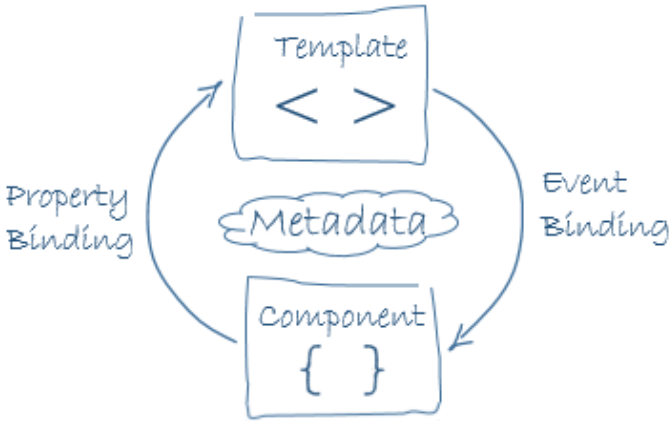
- *declarations* – svojstvo pomoću kojega se javlja Angular-u za prisutnost komponenti (engl. *components*), direktiva (engl. *directives*) i *pipes*-ova,

- *imports* – moduli čije su izvozne klase potrebne predlošcima komponenata deklariranim u NgModule-u,
- *providers* – globalna skupina usluga (engl. *services*) koje NgModule doprinosi. Sve te usluge postaju dostupne svim dijelovima aplikacije,
- *bootstrap* – predstavlja glavni pogled aplikacije nazvan korijenska komponenta (engl. *root component*), koji sadrži sve druge poglede aplikacije. Samo korijenski NgModule može biti postavljen kao *bootstrap* svojstvo. [7]

```
@NgModule({
  declarations: [
    AppComponent,
    RegisterComponent,
    NavbarComponent,
    EasyquizComponent,
    RegularquizComponent,
    HardquizComponent,
    EasyresultComponent,
    RegularresultComponent,
    HardresultComponent,
    DifficultyComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    RouterModule.forRoot(appRoutes),
    FormsModule,
    HttpClientModule
  ],
  providers: [QuizService, AuthGuard],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

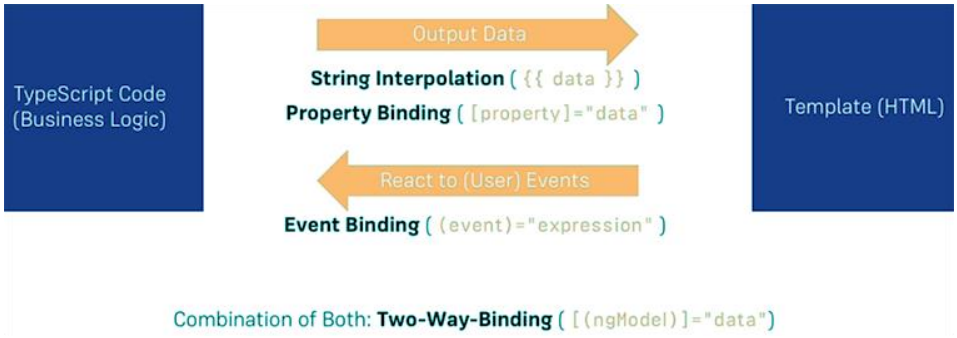
Sl. 2.3. *Dekorater @NgModule*

Angular također ima i mogućnost vezanja podataka (engl. *data binding*). Vezanje podataka je zapravo komunikacija između komponenti i njihovih HTML predložaka (ono što korisnik vidi). Postoje različiti oblici vezanja podataka koji se koriste u Angular-u. Za prikaz izlaznih podataka komponente na predlošku, koriste se oblici poput interpolacije niza (engl. *String interpolation*) i vezanja svojstvom (engl. *Property binding*). No ponekad je potrebno i obratno, tj. omogućiti korisniku da pri interakciji s predloškom napravi izmjenu na komponenti. To je moguće pomoću oblika vezanja zvanog vezanje događajem (engl. *Event binding*). Primjerice ako se poveže jedan klik događaj (engl. *click event*) na način da kada se klikne na gumb, da se tada pozove navedena metoda. [6, 8]



Sl. 2.4. *Vezanje svojstvom i vezanje događajem* [6]

No također postoji i dodatan oblik vezanja podataka s nazivom dvosmjerno vezanje podataka (engl. *Two-way data binding*). Ovim oblikom vezanja podataka, aplikacija može dijeliti i koordinirati podatke između komponente i samog predloška u oba smjera. Po sintaksi se vidi da je dvosmjerno vezanje podataka zapravo kombinacija vezanja svojstvom i vezanja događajem. Takvu i ostale primjere sintakse može se vidjeti na slici **2.5.**. [9]



Sl. 2.5. *Vezanje podataka* [10]

Jedno od bitnih svojstava Angular-a je to da su Angular predlošci dinamični (engl. *dynamic*). To znači da kada ih Angular iscrtava, tada mijenja DOM (engl. *Document Object Model*) prema naredbama dobivenim od direktiva. Direktiva je zapravo klasa s @Directive dekoraterom. Postoje tri vrste direktiva u Angular-u, a to su već spomenute komponente (engl. *components*), strukturne direktive (engl. *structural directives*) i atributne direktive (engl. *attribute directives*). [6]

Komponenta je, tehnički gledano, direktiva s predloškom te je najučestalija od svih triju direktiva. Budući da su komponente toliko karakteristične za Angular aplikacije, definiran je dekorater @Component koji se nastavlja na @Directive dekorater. [6]

Strukturne direktive su direktive koje su odgovorne za izgled HTML-a i koje mijenjaju strukturu pogleda, odnosno mijenjaju raspored i strukturu DOM-a dodavajući, brišući i mijenjajući elemente. Lagano se prepoznaju po oznaci zvjezdice koja stoji ispred imena direktive. Dva najčešća primjera su **ngFor** i **ngIf**. *ngFor* je iterativna direktiva koja predstavlja listu stavki dok je *ngIf* uvjetovana direktiva koja dodaje ili briše određenu stavku ako je uvjet zadovoljen (*boolean* izraz). Od ostalih primjera izdvaja se još i direktiva **ngSwitch** koja izdvaja jedan element između ostalih ovisno o postavljenom uvjetu. [11, 12]

S druge strane, atributne direktive mijenjaju izgled ili ponašanje elementa, komponente ili neke druge direktive. U predlošcima izgledaju poput običnih HTML atributa zbog čega su i dobile naziv atributne direktive. Najpoznatiji primjer atributnih direktiva je upravo **ngModel** direktiva koja implementira dvosmjerno vezanje podataka. ngModel mijenja ponašanje postojećeg elementa tako što prikazuje vrijednost svojstva i odgovara na događaje. Od ostalih primjera, ističu se **ngStyle** i **ngClass**. ngStyle direktiva omogućuje istovremeno i dinamičko izmjenjivanje stilova, dok ngClass omogućuje istovremeno dodavanje ili brisanje nekoliko CSS klasa. [6, 12]

Osim toga, Angular sadrži i usluge (engl. *services*). Usluga se može shvatiti kao jedna velika kategorija koja obuhvaća bilo koje vrijednosti, funkcije ili svojstva potrebne za neku aplikaciju. Ukratko rečeno, to je klasa s vrlo dobro definiranom svrhom da učini nešto specifično. Da bi učinio komponente učinkovitijim i jednostavnijim, Angular je razdvojio komponente od usluga te uslugama prepustio složenije zadatke poput dohvaćanja podataka sa servera i druge. Komponente bi trebale prezentirati svojstva i metode za vezanje podataka te biti posrednik između pogleda i logike aplikacije. Kako bi klasa bila definirana kao usluga u Angular-u, koristi se dekorater naziva @Injectable za pružanje meta podataka koji Angular-u omogućuju da ga ubrizgaju u komponentu kao ovisnost. Taj postupak se naziva ubrizgavanje ovisnosti (engl. *dependency injection*). [13]

Što se tiče usmjeravanja unutar Angular aplikacije, ono se obavlja na način da se korisnik pomiče između različitih pogleda koji su definirani u aplikaciji. Kako bi to pomicanje između različitih pogleda bilo moguće, koristi se Angular usmjerivač (engl. *Angular Router*). Taj usmjerivač omogućava navigaciju interpretirajući URL (engl. *Uniform Resource Locator*) preglednika kao instrukciju za mijenjanje pogleda. Da bi se Angular usmjerivač uopće mogao

koristiti, aplikacija mora sadržavati barem dvije komponente između kojih bi se moglo usmjerivati. Sve kreirane komponente treba zapisati unutar dekoratera @NgModule pod svojstvom *decorations*, što se nalazi u datoteci *app.module.ts*. Također treba kreirati niz ruta (engl. *routes*) između kojih je moguće usmjerivati. Zatim je taj niz ruta potrebno proslijediti metodi *RouterModule.forRoot* da bi se mogao konfigurirati usmjerivač. [14]

2.3. HTML

HTML je opisni jezik koji se koristi za kreiranje HTML dokumenata te web stranica. Kod web aplikacija se koristi u svrhe kreiranja predložaka za same komponente aplikacije. Pomoću njega opisuje se struktura web stranice, odnosno struktura predloška aplikacije. HTML za svoj opisni jezik koristi posebne oznake (engl. *tags*) pomoću kojih stvara elemente koji čine građevne blokove HTML dokumenta. Funkcionira na način da HTML pomoću oznaka daje upute web pregledniku kako prikazati određeni sadržaj. [15]

Sama struktura HTML dokumenta je poprilično jednostavna. Strukturu čine pojedini elementi HTML dokumenta, kao što su:

- *<!DOCTYPE html>* – označava da je riječ o HTML5 dokumentu te ga je potrebno navesti samo jedanput unutar HTML dokumenta,
- *<html></html>* – korijenski element stranice koji okružuje sav sadržaj na cijeloj stranici,
- *<head></head>* – element koji sadrži meta podatke o stranici,
- *<meta charset="utf-8">* – element koji uključuje većinu znakova koja se koriste u ljudskim jezicima,
- *<title></title>* – element koji postavlja ime stranice,
- *<body></body>* – element koji sadrži sav sadržaj koji se želi prikazati korisniku. [15]

Da bi stvorili elemente web stranice, potrebno je koristiti određene oznake. „HTML elementi se najčešće sastoje od početnih i završnih oznaka, gdje se sadržaj nalazi između navedenih oznaka. Pod HTML elementom smatra se sve od početne do završne oznake i sadržajem između njih." [15]

Svakom elementu je moguće dodati jedan ili više atributa. S atributima se dobiva više informacija o samom elementu. Ono što je potrebno atributu je to da je odvojen praznim prostorom

između sebe i elementa (ili atributa ako ih postoji više). Također je potrebno nakon naziva atributa staviti znak jednakosti te unutar dvostrukih navodnika staviti vrijednost atributa. [15]

U HTML-u postoji podjela na dvije vrste elemenata, a to su blok i linijski elementi. Blok elementi uvijek započinju u novoj liniji i zauzimaju cijelo područje od lijeve strane do desne, dok linijski elementi ne započinju u novoj liniji i zauzimaju samo onoliko koliko im je potrebno širine. Primjer blok elementa je <*div*> element, a primjer linijskog elementa je <*span*> element. [15]

Što se tiče praznog prostora, nije bitno koliko se praznog prostora koristi jer se u HTML-u svi ti razmaci smanjuju na jedan razmak. Stoga se korištenje više razmaka koristi radi bolje preglednosti u kodu. Također u HTML-u postoji i mehanizam za pisanje komentara. Komentari su ignorirani od strane preglednika i nisu vidljivi samom korisniku. [15]

HTML ima i mogućnost formatiranja teksta. HTML sadrži šest razina naslova, koji se kreću od najvažnijeg (<*h1*>) do najmanje važnog (<*h6*>). Tekst se piše u odlomcima, odnosno u kontejnerima s nazivom paragraf (<*p*>). Također postoji element horizontalni prekid (engl. *horizontal rule*) te element za prekid linije. Element za prekid linije (<*br*>) se koristi za prijelaz u novi red unutar paragrafa, dok se element horizontalni prekid (<*hr*>) koristi za tematsko odvajanje nekog sadržaja. [15]

Postoji i mogućnost naglašavanja dijelova teksta. Dio teksta koji se želi naglasiti, naglasi se <*em*> elementom naglašavanja koji ga stilizira kosim slovima. Ako postoji neki dio teksta koji je od iznimne važnosti, taj dio teksta se naglasi <*strong*> elementom koji ga stilizira podebljanim, odnosno „masnim“ slovima. Također postoje elementi poput <*i*>, <*b*> i <*u*> koji su se koristili za stiliziranje teksta prije nego što je CSS bio podržan. Osim ovih klasičnih elemenata za stiliziranje, koriste se i elementi za stiliziranje matematičkih jednadžbi te kemijskih formula. Za to su korisni elementi *superscript* (<*sup*>) za mala slova iznad teksta i *subscript* (<*sub*>) za mala slova ispod teksta. [15]

Osim stiliziranja teksta, moguće je koristiti element za obilježavanje kratica te element za označavanje detalja o kontaktu. Element za obilježavanje kratica/akronima (<*abbr*>) se koristi s atributom *title* unutar kojega se napiše puni naziv same kratice. Elementom <*address*> označuju se detalji i informacije o kontaktu. [15]

Budući da liste čine veliki dio web stranica, u HTML-u imaju svoje zasebne elemente. Za kreiranje nenumeriranih lista koristi se oznaka <*ul*>, dok se za svaki element liste koristi oznaka

<*li*>. S druge strane, numerirane liste se kreiraju s oznakom <*ol*>, dok se svaki element liste također kreira s oznakom <*li*>. Atributom *type* moguće je promijeniti vrstu numeriranja kod numeriranih lista, a atributom *start* početni broj od kojega se broji. No također postoji i treći tip lista koje se rijetko koriste. Nazivaju se opisne liste te se kreiraju oznakom <*dl*>, dok im se elementi kreiraju oznakama <*dt*> i <*dd*>. Oznakom <*dt*> se označavaju termini, dok se oznakom <*dd*> označavaju značenja tih termina. [15]

Što se tiče povezivanja HTML dokumenata i određenih resursa, to je moguće preko hiperveza. Hiperveza može biti bilo koji drugi HTML element. Da bi kreirali hipervezu, koristi se element <*a*> što označava sidro (engl. *anchor*). Nakon njega, koristi se atribut *href* (engl. *Hypertext reference*) koji kao vrijednost prima web adresu. Također je moguće i koristiti atribut *title* kojim se nadopunjuju informacije o samoj hipervezi. Atributom *target* može se omogućiti način otvaranja dokumenta kod povezivanja veza na resurs koji će se preuzeti (npr. pdf i word dokumenti). [15, 16]

Kako bi uljepšali dizajn i izgled web stranice, koristi se element <*img*>. Atributom *src* se odredi URL slike, dok atributom *alt* alternativni tekst za korisnike koji ne mogu vidjeti ili učitati sliku iz nekog razloga. Iako se to ne preporuča, veličinu slike je moguće promijeniti pomoću HTML atributa *width* i *height*. Moguće je i dodijeliti naslov slici pomoću već spomenutog atributa *title*. No da bi postojala semantička veza između slike i naslova, preporuča se korištenje dodatnih elemenata <*figure*> i <*figcaption*>. [15, 16]

HTML ima i mogućnost kreiranja tablica. Tablice se kreiraju koristeći element <*table*>, a naslov tablice elementom <*caption*>. Red tablice se kreira pomoću elementa <*tr*>. Zaglavlje tablice se definira pomoću elementa <*th*> unutar elementa <*tr*>, čiji tekst bude zadebljan i centriran. Podatkovna ćelija se kreira pomoću elementa <*td*>, također unutar elementa <*tr*>. Kod kompleksnijih tablica je korisno definirati strukturu tablice. To se postiže elementima <*thead*>, <*tfoot*> i <*tbody*>. Elementom <*thead*> okružuje se područje tablice koje predstavlja zaglavlje tablice, elementom <*tfoot*> područje tablice koje predstavlja podnožje tablice, a elementom <*tbody*> područje između zaglavlja i podnožja. U slučaju da je potrebno da se neki sadržaj proteže kroz više redova ili stupaca, koriste se elementi *rowspan* i *colspan*. [15, 16]

Već mnogo puta spomenuti URL je zapravo tekst koji definira gdje se nešto nalazi na internetu. Ono je ujedno i drugi naziv za web adresu, a može se sastojati od IP adrese (engl. *Internet Protocol*

address) ili običnih riječi. Za pronalazak datoteka, URL koristi putanje. „Putanja definira gdje se na datotečnom sustavu nalazi datoteka od važnosti.“ [16]

Postoje dvije vrste putanja – apsolutna i relativna putanja. Razlika između ovih dviju vrsta putanja je ta što „apsolutna putanja pokazuje na lokaciju koja je definirana sa svojom apsolutnom putanjom na web-u“ te će uvijek pokazivati na tu istu lokaciju, dok „relativna putanja će uvijek pokazivati na lokaciju koja je relativna od datoteke s koje se povezuje.“ Preporuča se koristiti relativne putanje ukoliko je to moguće zbog toga što su efikasnije te je lakše pregledavati izvorni kod jer su URL adrese kraće. [16]

Što se tiče semantike, HTML sadrži pojedine elemente koji opisuju dijelove dokumenta:

- zaglavlje: <*header*>,
- navigacijska traka: <*nav*>,
- glavni sadržaj: <*main*> s elementima <*article*>, <*section*> i <*div*>,
- bočna traka: <*aside*> često postavljena unutar <*main*>,
- podnožje: <*footer*>. [16]

Elementi <*article*> i <*section*> često izazivaju zbunjenost zbog nedovoljno razlika između njih. Razlika između njih je ta da je „element <*section*> definiran kao blok srodnih elemenata, a element <*article*> kao kompletan, samodostatan blok srodnih elemenata.“ [16]

Sl. 2.6. *Primjer semantičkog rasporeda* [16]

2.4. Microsoft alati

U web aplikaciji su korišteni mnogi Microsoft alati kao što su Microsoft Visual Studio 2019, Microsoft Visual Studio Code i Microsoft SQL Server Management Studio 18 koji je upravljao već kreiranim serverom koji je kreiran od strane alata Microsoft SQL Server 2017.

2.4.1. Microsoft SQL Server Management Studio 18

„Microsoft SQL Server Management Studio 18 (SSMS) je softverska aplikacija koju je razvio Microsoft. Koristi se za upravljanje, konfiguriranje i pružanje svih komponenti unutar Microsoft SQL Servera.“ [17]

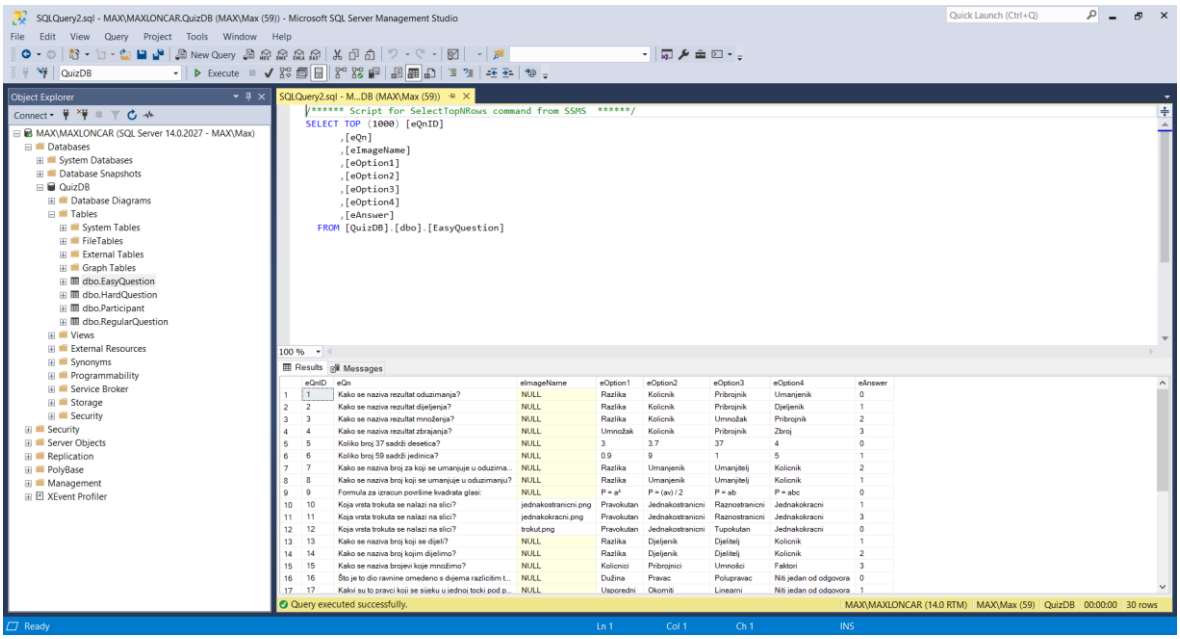
„Microsoft SQL Server je relacijska baza podataka također razvijena od strane Microsoft-a. Microsoft SQL Server kao server (poslužitelj) baze podataka ima primarnu funkciju pohranjivanja i dohvaćanja podataka na zahtjev drugih aplikacija.“ [18]

„Baza podataka pohranjuje podatke kao kolekciju tablica s različitim tipovima podataka. Microsoft SQL Server podržava više različitih tipova podataka kao što su *Integer* (cijeli brojevi), *Float* i *Decimal* (decimalni brojevi), *Char* (niz znakova), *Varchar* (niz znakova promjenjive duljine), *binary* (binarni podaci) te *Text* (tekstualni podaci).“ [18]

Svaka tablica s podacima treba sadržavati jedinstveni podatak (nekakav id) kojeg može označiti kao primarni ključ. Ako ima potrebe, u svakoj tablici mogu se dodati brojna ograničenja kao i strani ključ.

„Počevši od SQL Server Management Studio verzije 11, aplikacija se počela temeljiti na ljusci Visual Studio-a 2010. Od verzije 18 i nadalje, Microsoft SQL Server Management Studio se počeo temeljiti na izoliranoj ljusci Visual Studio-a 2018.“ [17]

Upravo ta verzija je i korištena za izradu ove web aplikacije.



Sl. 2.7. *Primjer Microsoft SQL Server Management Studio-a 18 koji prikazuje upit i rezultate upita*

2.4.2. Microsoft Visual Studio 2019

„Microsoft Visual Studio je integrirano razvojno okruženje (IDE) koje je razvio Microsoft. Koristi se u mnoge svrhe kao što su razvoj računalnih igrica, web stranica, web aplikacija, web usluga te razvoj mobilnih aplikacija.“ [19]

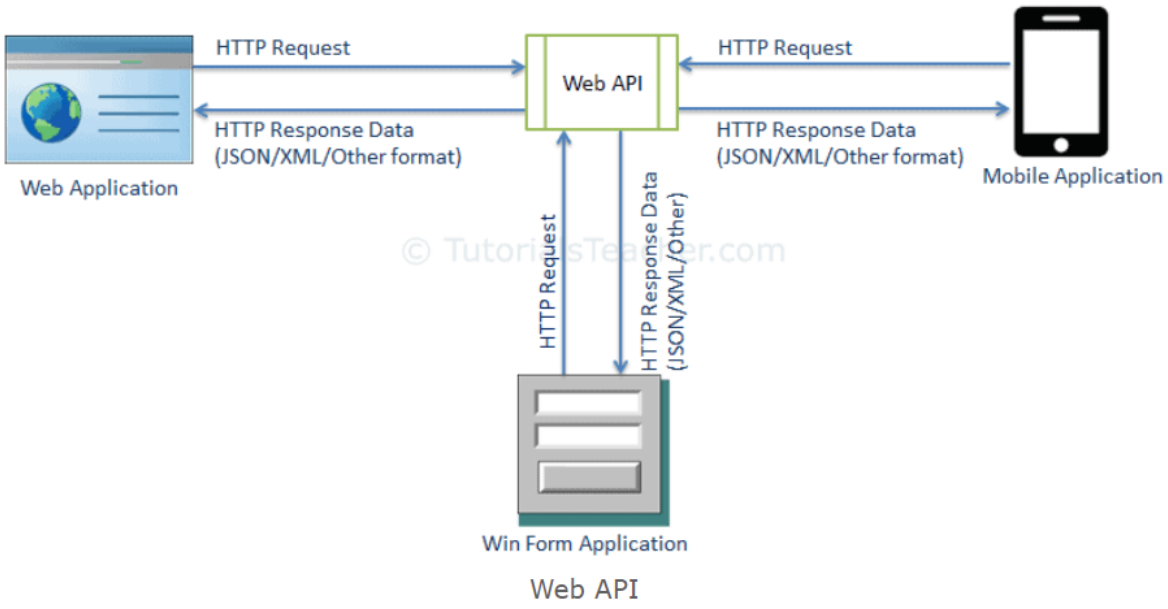
Što se tiče podrške programskih jezika, „Visual Studio podržava 36 različitih programskih jezika i dopušta uređivaču koda (engl. *code editor*) i programu za uklanjanje pogrešaka (engl. *debugger*) da podržavaju skoro svaki programski jezik. U ugrađene programske jezike se ubrajaju jezici od kojih su najpopularniji C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML i CSS. Podrška za druge jezike kao što su Python, Ruby, Node.js i M je moguća preko dodataka (engl. *plug-ins*) kojima se može dodati željeni programski jezik ili neko specifično svojstvo.“ [19]

Za kreiranje ASP.NET Web API-a (engl. *Active Server Pages Web Application Programming Interface*), korišten je upravo Microsoft Visual Studio 2019. API je zapravo sučelje koje sadrži skupine funkcija, protokola i alata za izgradnju softvera i aplikacija. Budući da se API nalazi na internetu, može mu se pristupiti pomoću HTTP (engl. *Hypertext Transfer Protocol*) protokola. Web API se može kreirati koristeći razne tehnologije kao što su Java, .NET i druge. [20]

„ASP.NET je poslužiteljski web aplikacijski okvir, otvorenog koda, dizajniran za web razvoj dinamičnih web stranica. Razvio ga je Microsoft kako bi omogućio programerima da kreiraju dinamične web stranice, aplikacije i usluge.“ [21]

ASP.NET Web API zapravo služi kao proširivi okvir za izgradnju usluga na temelju HTTP protokola. Uslugama koje su kreirane na temelju HTTP protokola je moguće pristupiti preko različitih aplikacija na različitim platformama. Platforme mogu biti web platforme, *windows* platforme, mobilne platforme i druge. [20]

Web API radi na način da aplikacija, koja može biti s bilo koje platforme, šalje HTTP zahtjev samom Web API-u. Web API odgovara aplikaciji tako što šalje HTTP podatkovni odgovor u obliku JSON-a, XML-a ili nekog drugog formata. Sama aplikacija može biti web aplikacija, mobilna aplikacija ili *windows form* aplikacija. Primjer rada Web API-a se nalazi na slici **2.8.**. [20]

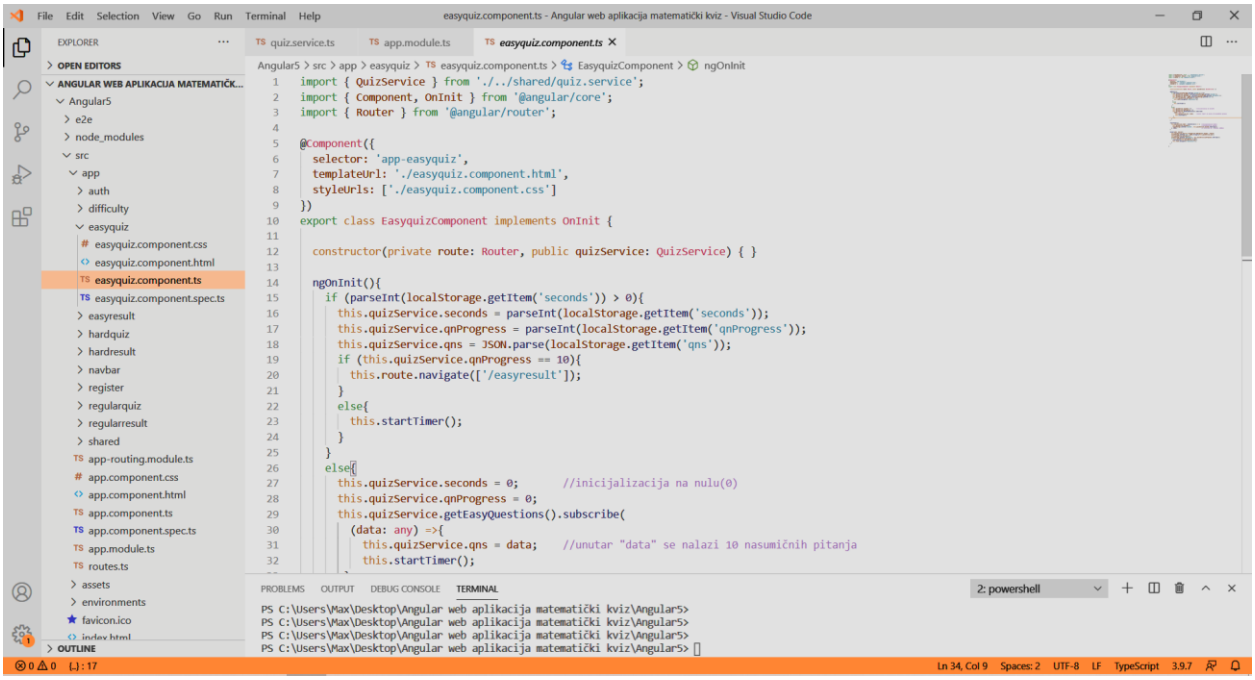


Sl. 2.8. *Primjer rada Web API-a* [20]

2.4.3. Microsoft Visual Studio Code

„Microsoft Visual Studio Code je besplatni uređivač izvornog koda kojeg je razvio Microsoft za Windows, Linux i macOS. Sadrži podršku za uklanjanje pogrešaka (engl. *debugging*), naglašavanje sintakse (engl. *syntax highlighting*), inteligentno dovršavanje koda (engl. *intelligent code completion*), isječke (engl. *snippets*), ispravljanje koda (engl. *code refactoring*) i ugrađeni Git (engl. *embedded Git*). Korisnici mogu promijeniti temu, prečace na tipkovnici, postavke i instalirati proširenja (engl. *extensions*) koja dodavaju dodatnu funkcionalnost.“ [22]

Visual Studio Code se koristio za interpretaciju web aplikacijskog okvira Angular. U njemu se pisala sva logika i funkcionalnosti web aplikacije, pretežito na TypeScript jeziku. Također su se u njemu kreirali predlošci, koji predstavljaju strukturu web aplikacije, za svaku komponentu. Predlošci su se uređivali i stilizirali pomoću CSS-a, također u Visual Studio Code-u.



Sl. 2.9. *Visual Studio Code s otvorenom EasyQuiz komponentom*

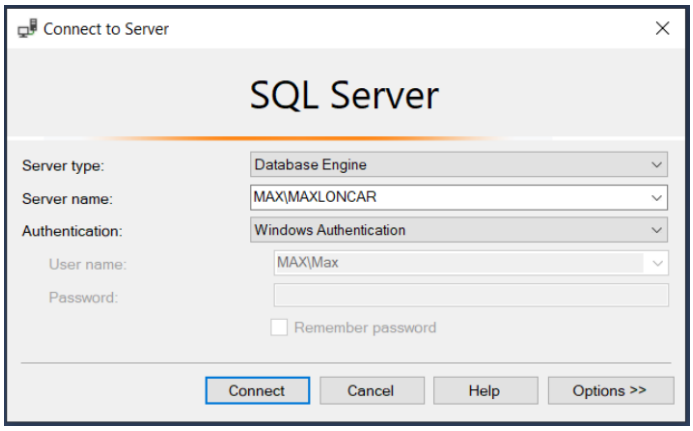
3. IZRADA ANGULAR WEB APLIKACIJE MATEMATIČKI KVIZ

Za izradu Angular web aplikacije, potrebno je bilo razmisliti koje programske alate koristiti. Odlučeno je da će se koristiti Microsoft SQL Server Management Studio 18 za izradu baze podataka te Microsoft SQL Server 2017 za izradu vlastitog servera. Podaci iz baze podataka će se dohvaćati preko ASP.NET Web API-a. Web API će poslužiti kao proširivi *okvir* za web aplikaciju te će se kreirati u Microsoft Visual Studio-u. Unutar Web API-a će se nalaziti model entiteta baze podataka s prisutnim podacima svih kreiranih tablica. Također će se nalaziti i kontroleri (engl. *controllers*) svih tablica te metode i njihove rute. Za svu logiku i funkcionalnosti te stilsko uređenje web aplikacije će se koristiti Microsoft Visual Studio Code. U njemu će se preko usluga (engl. *services*) pristupati kreiranim Web API metodama.

3.1. Baza podataka

Za početak je potrebno preuzeti Microsoft SQL Server i Microsoft SQL Server Management Studio. Nakon preuzimanja, potrebno je pokrenuti instalaciju „SQL Server Setup“ kako bi se kreirao vlastiti server. Pri instalaciji samog servera, potrebno je odabrati vrstu servera, vrstu autentifikacije te unijeti naziv i ID servera (ako se ne unese, generira se pomoću naziva). Ti podaci će kasnije biti potrebni za povezivanje na SQL Server pri pokretanju Microsoft SQL Server Management Studio programa.

Nakon kreiranja servera, potrebno je pokrenuti već spomenuti Microsoft SQL Server Management Studio. Ulaskom u SQL Server Management Studio, otvara se prozor za spajanje na server. U tom prozoru je potrebno unijeti podatke unesene pri kreiranju servera te se konačno povezati na server. Primjer se nalazi na slici **3.1.**.



Sl. 3.1. *Povezivanje na SQL Server*

Nakon povezivanja na server, potrebno je kreirati bazu podataka. U ovom slučaju, baza podataka se zove **QuizDB**. Zatim je potrebno kreirati četiri tablice s nazivima *EasyQuestion, RegularQuestion, HardQuestion* i *Participant*. Za svaku tablicu je potrebno unijeti njezine podatke, uključujući i jedinstveni ID kojeg treba učiniti primarnim ključem.

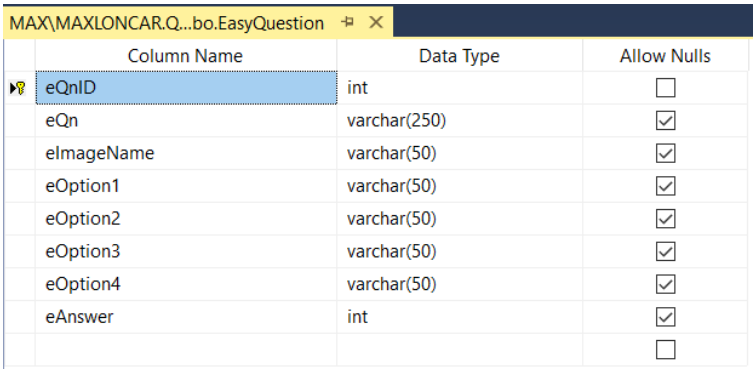
Tri tablice s nazivima *EasyQuestion*, *RegularQuestion* i *HardQuestion*, predstavljaju tablice u kojima se nalaze pitanja s tri različite težine, tj. svaka tablica sadrži pitanja sa svojom težinom (npr. tablica *EasyQuestion* sadrži samo lagana pitanja). Sve tri tablice sadrže iste podatke sa sličnim nazivima. Jedina razlika je u prvom slovu koje predstavlja određenu tablicu (npr. *rQnID* pripada tablici *RegularQuestion*, *eQnID* pripada tablici *EasyQuestion*, *hQnId* pripada tablici *HardQuestion*).

Sve tri tablice sadrže osam stupaca. Prvi stupac sadrži jedinstveni identifikator pitanja (*QnID*), dok drugi stupac sadrži tekst pitanja (*Qn*). Treći stupac sadrži naziv slike koja će biti prikazana samo ako ga to određeno pitanje sadrži (*ImageName*). Slike s navedenim nazivima će biti spremljene unutar Web API programa pod mapom *Images*. Četvrti, peti, šesti i sedmi stupac sadrže četiri ponuđene opcije za odgovor (*Option1*, *Option2*, *Option3*, *Option4*). Sedmi stupac sadrži numeričku vrijednost točnog odgovora (*Answer*) koja djeluje poput indeksa unutar polja.

Numerička vrijednost se nalazi u rasponu od 0 do 3:

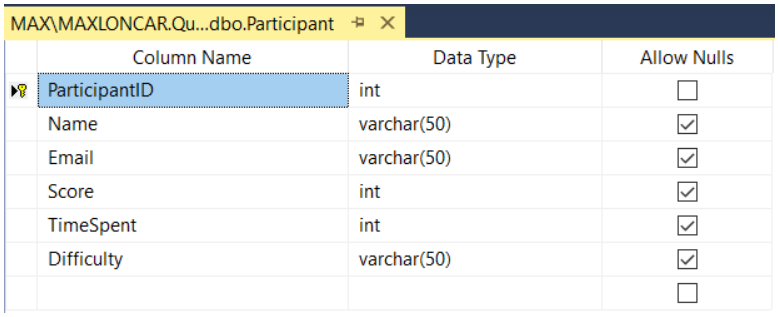
- 0 – prva opcija je točan odgovor,
- 1 – druga opcija je točan odgovor,
- 2 – treća opcija je točan odgovor,
- 3 – četvrta opcija je točan odgovor.

Primjer ovakvog unosa podataka se nalazi na slici **3.2.**.



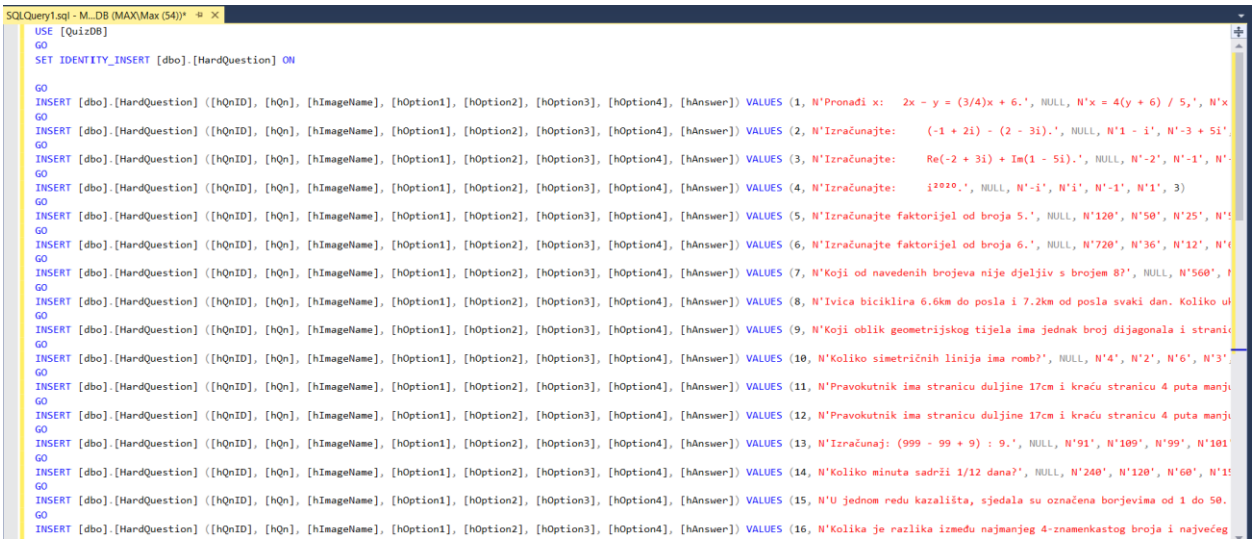
Sl. 3.2. *Unos podataka unutar tablice EasyQuestion*

Također istu stvar treba učiniti s tablicom *Participant*. Tablica *Participant* sadrži šest stupaca. Prvi stupac također sadrži jedinstveni identifikator (*ParticipantID*). Drugi stupac sadrži ime korisnika (*Name*), a treći stupac e-mail korisnika (*Email*). Četvrti stupac sadrži rezultat korisnika kojeg je ostvario (*Score*), dok peti stupac sadrži proteklo vrijeme (*TimeSpent*) rješavajući kviz. Posljednji, šesti stupac sadrži težinu kviza (*Difficulty*) koju je korisnik odabrao prije početka rješavanja kviza. Primjer unosa tablice *Participant* se nalazi na slici **3.3.**.



Sl. 3.3. *Unos podataka unutar tablice Participant*

Nakon kreiranja tablica, potrebno je umetnuti pitanja unutar tablice *EasyQuestion*, *RegularQuestion* i *HardQuestion*. Pitanja se umeću pomoću napisane SQL skripte koja je prikazana na slici **3.4.** (SQL skripte svih triju tablica su priložene na CD-u) . Nakon što se napiše skripta, potrebno je pritisnuti naredbu *Execute*.



Sl. 3.4. *Primjer umetanja pitanja unutar tablice HardQuestion*

3.2. Web API

Nakon baze podataka, na red dolazi kreiranje Web API-a. Za početak je potrebno pokrenuti Microsoft Visual Studio program te kreirati novi projekt. Pri kreiranju projekta je potrebno odabrati vrstu samog projekta. Za kreiranje Web API-a, potrebno je pod kategorijom *Web* odabrati *ASP.NET Web Application*. Nakon toga je potrebno dodijeliti ime projektu te odabrati lokaciju projekta koji se u ovom slučaju naziva **WebAPI**. Na kraju nakon dodijele naziva, potrebno je odabrati predložak Web API. Time se kreirao Web API projekt koji se može pokrenuti u pregledniku kombinacijom tipki CTRL + F5.

Unutar Web API-a je potrebno kreirati određene metode. No može se pojaviti problem u kojem web preglednik zbog sigurnosti sprječava web stranicu da pošalje zahtjev drugačijoj domeni od one koja ju poslužuje. Stoga je potrebno instalirati paket CORS (engl. *Cross Origin Resource Sharing*) kako bi osiguranje malo popustilo te dopustilo pristup web stranicama s drugačijim domenama. Instalacija se obavlja na način da se naredba *Install-Package Microsoft.AspNet.WebApi.Cors* unese unutar *Package Manager Console*. Nakon uspješne instalacije je potrebno obaviti *Build Solution*. Više informacija o CORS paketu se može pronaći na [23].

Moguće je da nakon instalacije CORS paketa dođe do pogreške (engl. *Error*). Razlog tomu je nekompatibilnost verzija između paketa *System.Web.Http* i *System.Web.Cors*. Da bi se ovaj problem riješio, potrebno je ažurirati paket *System.Web.Http*. To se obavlja na način da se prvo ažurira *WebApi* paket pomoću naredbe *Update-Package Microsoft.AspNet.WebApi*, a zatim da se ponovno instalira *Core* paket pomoću naredbe *Install-Package Microsoft.AspNet.WebApi.Core*.

Da bi započeli rad na Web API-u, potrebno je konfigurirati Web API projekt kako bi dopustili pristup zahtjevu web aplikacije s brojem porta „4200“. Zato je potrebno unutar prozora *Solution Explorer* pod mapom *App_Start* ući u konfiguracijsku datoteku naziva *WebApiConfig.cs* te dodati liniju koda koja se nalazi na slici **3.5.**.

```
config.EnableCors(new EnableCorsAttribute("http://localhost:4200", headers: "*", methods: "*"));
```

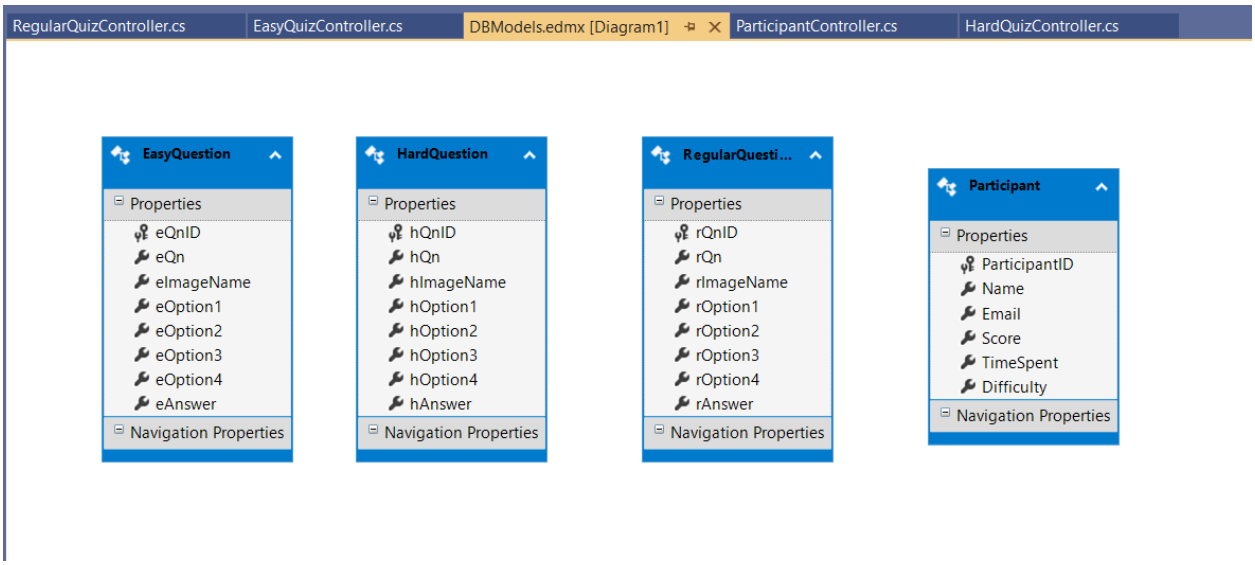
Sl. 3.5. *Dopuštanje pristupa zahtjevima web aplikacije*

Zatim je potrebno formatirati podatkovni odgovor (engl. *Data Response*) od strane Web API-a. Potrebno je proširiti datoteku *Global.asax* i ući u datoteku *Global.asax.cs*. Unutar te datoteke je potrebno dodati dvije linije koda kojima se dodaje *JsonFormatter* i briše *XmlFormatter*. Dvije linije koda se mogu vidjeti na slici **3.6.**.

```
GlobalConfiguration.Configuration.Formatters.JsonFormatter.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore;
GlobalConfiguration.Configuration.Formatters.Remove(GlobalConfiguration.Configuration.Formatters.XmlFormatter);
```

Sl. 3.6. *Formiranje odgovora podataka*

Nakon toga je potrebno dodati model datoteke tipa *ADO.NET Entity Data Model* unutar datoteke *Models* te mu odrediti naziv. U ovom slučaju se model datoteke naziva **DBModels**. Pri upitu što će model datoteke sadržavati, potrebno je odabrati opciju *Generate from database* te odabrati vlastitu konekciju koja je već ranije kreirana pomoću Microsoft SQL Server-a. Također je potrebno odabrati i već kreiranu bazu podataka koja se u ovom slučaju naziva QuizDB. Nakon toga je potrebno unijeti naziv pod kojim će se spremiti postavke modela entiteta, a taj naziv je u ovom slučaju **DBModel**. Na kraju je potrebno odabrati željene objekte koje se žele uključiti u model entiteta, a to bi bile sve četiri tablice. Model entiteta baze podataka je prikazan na slici **3.7.**.



Sl. 3.7. *Model entiteta baze podataka DBModels*

Nakon kreiranja modela entiteta baze podataka, potrebno je kreirati četiri kontrolera (engl. *Controllers*). Unutar mape *Controllers* je potrebno dodati kontrolere tipa *Web API 2 Controller – Empty* te ih nazvati imenom tablice nakon koje slijedi naziv „Controller“ (npr. *ParticipantController*). Nakon kreiranja svih četiri kontrolera, unutar svakoga je potrebno napisati ključne Web API metode.

Unutar kontrolera *ParticipantController*, prvo je potrebno kreirati Web API metodu naziva *Insert()* za umetanje novih sudionika kviza. Za tu metodu se koristi HTTP metoda POST za slanje podataka prema poslužitelju/serveru. Također je potrebno deklarirati rutu metode koja je u ovom slučaju „api/InsertParticipant“. Sama metoda kao parametar sadrži objekt klase *Participant*. Osnovna funkcija te metode je da preko objekta modela entiteta *DBModel* umetne nove sudionike nakon čega okvir entiteta (engl. *Entity Framework*) ažurira nove ID-ove unutar klase *Participant*. Nakon umetanja se spremaju promjene te se vraća objekt klase Participant.

Nakon metode s nazivom *Insert()*, potrebno je dodati još jednu Web API metodu naziva *UpdateScoreTime()*. Navedena metoda također koristi HTTP metodu POST te joj je također potrebno deklarirati rutu („api/UpdateScoreTime“). Metoda kao parametar također sadrži objekt klase *Participant*. Pomoću ove metode se ažurira trenutno stanje rezultata, vremena potrebnog za rješavanje kviza te težinu kviza. To se obavlja pomoću stanja entiteta (engl. *Entity State*) koje također koristi objekt modela entiteta. Stanje entiteta se postavlja u modificirano (engl. *modified*), što znači da je entitet praćen kontekstom i da postoji u bazi podataka te da su neke ili sve vrijednosti modificirane. Nakon toga se spremaju promjene. Obje Web API metode se nalaze na slici **3.8.**.

```
namespace WebAPI.Controllers
{
    0 references
    public class ParticipantController : ApiController
    {
        [HttpPost]
        [Route("api/InsertParticipant")]
        0 references
        public Participant Insert(Participant model) {
            using (DBModel db = new DBModel())
            {
                db.Participants.Add(model);
                db.SaveChanges();
                return model;
            }
        }

        [HttpPost]
        [Route("api/UpdateScoreTime")]
        0 references
        public void UpdateScoreTime(Participant model)
        {
            using (DBModel db = new DBModel())
            {
                db.Entry(model).State = System.Data.EntityState.Modified;
                db.SaveChanges();
            }
        }
    }
}
```

Sl. 3.8. *Metode unutar ParticipantController-a*

Budući da su sva tri kviz kontrolera slična, odnosno imaju jednake Web API metode s različitim nazivima (*EasyQuizController*, *RegularQuizController*, *HardQuizController*), kao primjer će biti prikazan samo jedan kontroler sa svojim metodama. Kontroler koji će biti prikazan se naziva *EasyQuizController*.

Za početak je potrebno kreirati Web API metodu s nazivom *GetEasyQuestions()* kojom će se dohvaćati pitanja iz baze podataka. Za tu metodu se koristi HTTP metoda GET koja dohvaća podatke sa poslužitelja/servera. Definirana ruta metode je „api/EasyQuestions“. Metoda ne sadrži parametre, no također koristi objekt modela entiteta za pristup podacima iz tablice. Unutar varijable *eQns* je izvučeno deset nasumičnih pitanja od mogućih trideset iz tablice *EasyQuestion* te su odabrani svi stupci iz tablice osim jednog, stupca *eAnswer* koji će se iskoristiti unutar Angular programa. Nasumičnost se postigla pomoću *Guid.NewGuid()* metode te se varijabla konvertirala u polje pomoću metode *ToArray()*. Zbog lakšeg dizajna Angular aplikacije, četiri ponuđene opcije odgovora će biti smještene u zasebno polje *Options*. Stoga je kreirana nova varijabla naziva *updated* kojoj se pridružuju vrijednosti varijable *eQns* konvertirane u listu s novim dodatkom polja nizova („stringova“) *Options* u kojoj se nalaze sve četiri ponuđene opcije odgovora. Sama metoda *GetEasyQuestions()*, budući da je tipa *HttpResponseMessage*, stvara i vraća objekt klase *HttpResponseMessage* kojem se predaje varijabla *updated*. Sama metoda se nalazi na slici **3.9.**.

```
public class EasyQuizController : ApiController
{
    [HttpGet]
    [Route("api/EasyQuestions")]
    0 references
    public HttpResponseMessage GetEasyQuestions()
    {
        using (DBModel db = new DBModel())
        {
            var eQns = db.EasyQuestions
                .Select(x => new { easyQnID = x.eQnID, easyQn = x.eQn, easyImageName = x.eImageName, x.eOption1, x.eOption2, x.eOption3, x.eOption4 })
                .OrderBy(y => Guid.NewGuid())
                .Take(10)
                .ToArray();
            var updated = eQns.AsEnumerable()
                .Select(x => new
                {
                    eQnID = x.easyQnID,
                    eQn = x.easyQn,
                    eImageName = x.easyImageName,
                    Options = new string[] { x.eOption1, x.eOption2, x.eOption3, x.eOption4 }
                }).ToList();
            return this.Request.CreateResponse(HttpStatusCode.OK, updated);
        }
    }
```

Sl. 3.9. *Metoda GetEasyQuestions() unutar EasyQuizController-a*

Druga Web API metoda se naziva *GetEasyAnswers()* te je njezina osnovna funkcija vraćanje odgovora na deset nasumično izvučenih pitanja. Za tu metodu se koristi HTTP metoda POST te joj je ruta „api/EasyAnswers“. Metoda kao parametar prima cjelobrojni niz *eqIDs* unutar kojega će biti ID-ovi od deset nasumičnih pitanja vraćenih od metode *GetEasyQuestions()*. Metoda također koristi objekt modela entiteta za pristup podacima iz tablice. Unutar varijable *result* su odabrani odgovori za svaki ID izvučenog pitanja. Metoda *GetEasyAnswers()* je također tipa *HttpResponseMessage* te također stvara i vraća objekt klase *HttpResponseMessage* kojem se predaje varijabla *result*. Unutar vraćenog cjelobrojnog polja odgovora je zadržan redoslijed ID-ova pitanja koja se nalaze unutar cjelobrojnog niza *eqIDs*. Što znači da je unutar vraćenog cjelobrojnog niza, prvi element stupca eAnswer zapravo odgovor na prvo pitanje unutar cjelobrojnog niza *eqIDs*. Taj redoslijed je održan pomoću metode *OrderBy()* u kojoj pretražujemo polje pomoću naredbe *IndexOf(eqIDs, x.eQnID)* u kojoj prvu vrijednost (*eqIDs*) pretražujemo počevši od druge vrijednosti (*x.eQnID*) i redom. Metoda se nalazi na slici **3.10.**.

```
[HttpPost]
[Route("api/EasyAnswers")]
0 references
public HttpResponseMessage GetEasyAnswers(int[] eqIDs)
{
    using (DBModel db = new DBModel())
    {
        var result = db.EasyQuestions
            .AsEnumerable()
            .Where(y => eqIDs.Contains(y.eQnID))
            .OrderBy(x => { return Array.IndexOf(eqIDs, x.eQnID); })
            .Select(z => z.eAnswer)
            .ToArray();
        return this.Request.CreateResponse(HttpStatusCode.OK, result);
    }
}
```

Sl. 3.10. *Metoda GetEasyAnswers() unutar EasyQuizController-a*

Što se tiče naziva slika, sve slike su spremljene unutar Web API-a pod mapom *Images*. Pitanje koje sadrži sliku će imati napisani puni naziv slike zajedno s ekstenzijama, dok pitanje koje nema sliku će pod tom vrijednosti imati napisanu vrijednost „NULL“.

Nakon što su sve metode i svi kontroleri završeni, potrebno je spremiti sve promjene i kliknuti na *Build Solution*.

3.3. Angular web aplikacija

Nakon završetka kreiranja Web API-a, na red dolazi Angular aplikacija. Angular će se obrađivati unutar Microsoft Visual Studio Code uređivača. Prije kreiranja Angular projekta, potrebno je preuzeti i instalirati *Node.js* i *npm package manager*. Tijekom programiranja, kod se može unositi preko integralnog terminala (engl. *Integrated Terminal*) ili preko Angular CLI-a. Angular CLI se može instalirati unošenjem naredbe ***npm install -g @angular/cli*** unutar integralnog terminala. Kreiranje Angular projekta se obavlja unošenjem naredbe *ng new ime_aplikacije* (Angular5). Kreiranje projekta se potvrđuje pritiskom na tipku ENTER nakon čega dolazi do instalacije potrebnih paketa.

Što se tiče same strukture web aplikacije, potrebno je kreirati devet komponenti (*difficulty, easyquiz, easyresult, hardquiz, hardresult, navbar, register, regularquiz, regularresult*). Unutar *register* komponente, korisnik se registrira unoseći svoje ime i e-mail. Nakon uspješne registracije, dolazi *difficulty* komponenta unutar koje korisnik odabire težinu od mogućih triju težina. Komponenta *navbar* predstavlja navigacijsku traku na kojoj se nalazi ime kviza te gumb za odjavu korisnika. Komponente *easyquiz, regularquiz* i *hardquiz* prikazuju deset nasumičnih pitanja dohvaćenih iz baze podataka, svaka iz svoje težine kviza. Dohvaćena pitanja se prikazuju jedno po jedno sa mogućim opcijama za odgovoriti. Unutar komponenti *easyresult, regularresult* i *hardresult* su prikazani konačni rezultati zajedno sa svim pitanjima i točnim odgovorima, također i ime korisnika te vrijeme proteklo rješavajući kviz.

Osim ovih devet komponenata, postoji i uslužna klasa (engl. *service class*) naziva *QuizService* unutar datoteke *quiz.service.ts* koja je dio mape *shared*. Unutar ove klase se nalaze uobičajene funkcije i funkcije za konzumiranje Web API metoda. Također, projekt sadrži i datoteku imena *routes.ts* unutar koje su konfigurirane rute za usmjeravanje unutar web aplikacije. Osim nje, kreirana je i mapa s nazivom *auth.* Unutar nje se nalazi *auth.guard.ts* datoteka koja se bavi autorizacijom i autentifikacijom korisnika.

Kako bi kreiranje aplikacije moglo započeti, potrebno je promijeniti radni direktorij da pokazuje unutar kreirane projektne mape koja se u ovom slučaju naziva **Angular5**. To se obavlja pomoću naredbe *cd ime_aplikacije*. Za kreiranje komponenti se koristi naredba *ng g c ime_komponente* (npr. *ng g c register)*. Oznaka *g* je skraćeno od *generate,* dok je oznaka *c* skraćeno od *component*. Stoga tu naredbu treba koristiti za izradu svih devet komponenti. Za izradu uslužne klase, kreirana je mapa *shared*. Da bi se kreirala uslužna klasa, prvo je potrebno promijeniti radni

direktorij da pokazuje na mapu *shared*, a zatim kreirat uslužnu klasu pomoću naredbe *ng g s ime_uslužne_klase*. Oznaka *s* je skraćeno od *service* te je u ovom slučaju kreirana datoteka uslužne klase datoteka s nazivom *quiz.service.ts*.

Nakon kreiranja osnovnih komponenti, potrebno je ponovno radni direktorij usmjeriti prema projektu Angular5 te pokrenuti server pomoću naredbe *ng serve --open*. Ta naredba provjerava datoteke i ponovno gradi aplikaciju svaki puta kada programer napravi neku promjenu na navedenim datotekama. Opcija *--open* ili skraćeno *--o* automatski otvara preglednik na adresi *http://localhost:4200/*.

Nakon pokretanja aplikacije, sama aplikacija je spremna za uređivanje i prilagođavanje. Budući da će se koristiti *front-end* okvir Materialize CSS za pripomoć pri uređivanju obrazaca, tipaka i ikona, prvo ga je potrebno uključiti unutar globalne *index.html* datoteke. Materialize CSS se uključuje na način da se sa službene stranice [24] uzme *stylesheet* i zalijepi u područje meta podataka (unutar *<head>* sekcije). Nakon toga je potrebno uzeti JavaScript referencu na okvir i zalijepiti je unutar *<body>* sekcije. No da bi JavaScript referenca radila, potrebno je također dodati i *jQuery script* referencu. Uz *front-end* okvir, korišteni su također i *materialized* ikone kreirane od strane Google-a. Način uključivanja je vidljiv na slici **3.11.**.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Matematički kviz</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css">
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
</head>
<body>
  <app-root></app-root>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js"></script>
</body>
</html>
```

Sl. 3.11. *Uključivanje Materialize CSS front-end okvira i ikona*

Unutar predloška zadane komponente *app.component.html*, potrebno je zamijeniti zadani HTML kod s *placeholder*-om *<router-outlet></router-outlet>* kojim se povezuju usmjerivač i rute s predloškom. Kako bi to funkcioniralo, dodaje se nova datoteka s nazivom *routes.ts* unutar mape *app*. Unutar datoteke *routes.ts* se dodaje konstanta *appRoutes* unutar koje se dodavaju rute za svaku kreiranu komponentu. Zadanom rutom se proglašava ruta na *register* komponenti na način da se *path* postavi kao prazan niz te opcija preusmjeravanja postavi na *register* komponentu.

```
import { AuthGuard } from './auth/auth.guard';
import { Routes } from '@angular/router';
import { RegisterComponent } from './register/register.component';
import { DifficultyComponent } from './difficulty/difficulty.component';
import { EasyquizComponent } from './easyquiz/easyquiz.component';
import { RegularquizComponent } from './regularquiz/regularquiz.component';
import { HardquizComponent } from './hardquiz/hardquiz.component';
import { EasyresultComponent } from './easyresult/easyresult.component';
import { RegularresultComponent } from './regularresult/regularresult.component';
import { HardresultComponent } from './hardresult/hardresult.component';
import { from } from 'rxjs';

export const appRoutes : Routes =[
    { path:'register', component:RegisterComponent },
    { path:'difficulty', component:DifficultyComponent, canActivate :  [AuthGuard]},
    { path:'easyquiz', component:EasyquizComponent, canActivate :  [AuthGuard] },
    { path:'regularquiz', component:RegularquizComponent, canActivate :  [AuthGuard] },
    { path:'hardquiz', component:HardquizComponent, canActivate :  [AuthGuard] },
    { path:'easyresult', component:EasyresultComponent, canActivate :  [AuthGuard] },
    { path:'regularresult', component:RegularresultComponent, canActivate :  [AuthGuard] },
    { path:'hardresult', component:HardresultComponent, canActivate :  [AuthGuard] },
    { path:'', redirectTo:'/register', pathMatch:'full' }
];
```

Sl. 3.12. *Datoteka routes.ts*

Na ovakav način se konfigurira usmjeravanje za ovu aplikaciju, no aplikaciji je također potrebno reći da slijedi ove rute. Stoga je potrebno unutar datoteke *app.module.ts* uvesti (engl. *import*) *RouterModule* te ga također postaviti unutar polja *imports* sa funkcijom *forRoot()* kojoj se predaje konstanta *appRoutes* kao što je prikazano na slici **3.13.**.

```
imports: [
  BrowserModule,
  AppRoutingModule,
  RouterModule.forRoot(appRoutes),
```

Sl. 3.13. *RouterModule sa funkcijom forRoot()*

Budući da je *navbar* komponenta prisutna kod drugih komponenti, najlakše ja započeti od nje. Komponenta *navbar* je dizajnirana tako da na njoj piše naslov kviza te se na njoj nalazi gumb kojim se korisnik može odjaviti. Da bi to bilo moguće izvesti, deklariran je jedan klik događaj kojem je pridružena funkcija *SignOut()*. Funkcija *SignOut()* čisti *localStorage* (o njemu detaljnije malo kasnije) i interval pomoću metoda *clear()* i *clearInterval()*. Također funkcija usmjerava korisnika natrag na *register* komponentu. No da bi to bilo moguće, potrebno je uvesti i deklarirati objekt klase *Router* s nazivom *route* te objekt klase *QuizService* s nazivom *quizService*. Izgled funkcije *SignOut()* se nalazi na slici **3.14.**.

```
SignOut(){
  localStorage.clear();
  clearInterval(this.quizService.timer);
  this.route.navigate(['/register']);
}
```

Sl. 3.14. *Funkcija SignOut()*

Budući da se *navbar* komponenta treba nalaziti unutar drugih komponenti, potrebno je njezin selektor *app-navbar* zalijepiti unutar predložaka tih komponenti na način prikazan na slici **3.15.**.

```
<app-navbar></app-navbar>
```

Sl. 3.15. *Placeholder komponente navbar*

Od glavnih komponenti, za početak je najbolje krenuti s *register* komponentom. Unutar predloška *register.component.html* se uređuje struktura obrasca za prijavu. Ono što je bitno za *register* komponentu jest to da ima dva prostora za unos imena i e-maila. Korištena je HTML oznaka <*form*>, te je u njoj navedena lokalna referenca *#registerForm* kojoj je pridružena direktiva *ngForm*. Ta direktiva se veže za obrazac te ona prati skup vrijednosti i status potvrde. Stoga je uz direktivu naveden i *ngSubmit* događaj, koji će biti obaviješten kada korisnik podnese obrazac. *ngSubmit* događaju je pridružena metoda *OnSubmit()* koja kao parametre prima vrijednosti imena i vrijednosti e-maila dodane preko lokalnih referenci *#Name* i *#Email*. Za podnošenje obrasca, kreiran je gumb. Pored ovih dviju lokalnih referenci je navedena i direktiva *ngModel*. Ovakav pristup se naziva obrazac na temelju predloška (engl. *template-driven form*). *ngModel* reagira na korisnikov unos s potvrdom unosa i rukovanjem pogreškama. Da bi to funkcioniralo, potrebno je unutar datoteke *app.module.ts* uvesti *FormsModule* te ga dodati unutar polja *imports*. Također ako je predlošku potrebna neka slika, ona se može primjerice ubaciti u kreiranu mapu *img* koja je kreirana unutar mape *assets*. Putanju zajedno sa punim nazivom slike je potrebno navesti unutar atributa *src*.

```
<form #registerForm="ngForm" (ngSubmit)="OnSubmit(Name.value, Email.value)">
    <div class="row">
        <div class="input-field col s12">
            <i class="material-icons prefix">account_circle</i>
            <input type="text" name="Name" #Name ngModel required>
            <label>Ime i prezime</label>
        </div>
    </div>
    <div class="row">
        <div class="input-field col s12">
            <i class="material-icons prefix">email</i>
            <input type="text" name="Email" #Email ngModel required [pattern]="emailPattern">
            <label>Email</label>
        </div>
    </div>
    <div class="row">
        <div class="input-field col s12">
            <button class="waves-effect waves-light btn-small" type="submit" [disabled]="!registerForm.valid">Prijavi se</button>
        </div>
    </div>
</form>
```

Sl. 3.16. *Obrazac za registraciju unutar register predloška*

Pri registraciji korisnika, obavezan je unos imena i e-maila. Stoga se stavlja atribut *required* pored unosa imena i e-maila, odnosno pored lokalnih referenci i direktive *ngModel*. Kako bi potvrdili format e-maila, potrebno je unutar datoteke *register.component.ts* dodati uzorak e-maila pod nazivom *emailPattern*. Zatim je potrebno unutar predloška, pored unosa e-maila, pomoću vezanja svojstvom (engl. *Property Binding)* navesti uzorak e-maila što je prikazano na slici **3.17.**. Time se onemogućava prijava ako su unosi prazni te ako e-mail nije u pravilnom obliku kakav je naveden unutar uzorka *emailPattern*. Ako je prijava onemogućena, gumb (engl. *button*) također treba biti onemogućen. To se obavlja na način da se ponovno koristi vezanje svojstvom u kojem se gumb onemogućava ako je prijava onemogućena. Takav primjer se nalazi na slici **3.18.**.

```
[pattern]="emailPattern"
```

Sl. 3.17. *Potvrđivanje uzorka e-maila*

```
[disabled]="!registerForm.valid"
```

Sl. 3.18. *Onemogućavanje gumba ako je prijava onemogućena*

Sada je potrebno omogućiti unos novog korisnika preko njegovog imena i e-maila tako da se pozove Web API metoda *Insert()*. Pomoću *QuizService* klase će se konzumirati Web API metode. Za početak je potrebno definirati novo svojstvo unutar klase *QuizService* s nazivom *rootUrl*.

Svojstvu *rootUrl* se pridružuje zadani Web API URL (*https://localhost:44375/*). Kako bi konzumirali Web API metode, uvodi se *HttpClient* tako da ga se definira unutar konstruktora. Nakon toga, definira se nova funkcija *insertParticipant()* koja kao parametre također prima ime i e-mail. Unutar te funkcije se nalazi varijabla *body* kojoj se pridružuju ime i e-mail sudionika. Funkcija zove POST funkciju od HTTP objekta u kojoj je prvi parametar zadani *rootUrl* na kojeg se dodaje ruta Web API metode *Insert()* („/api/InsertParticipant“). Drugi parametar joj je definirana varijabla *body* koja sadrži ime i e-mail sudionika kviza. *insertParticipant()* funkcija se nalazi na slici **3.19.**.

```
insertParticipant(name: string, email: string){
  var body = {
    Name: name,
    Email: email
  }
  return this.http.post(this.rootUrl + '/api/InsertParticipant', body);
}
```

Sl. 3.19. *insertParticipant() funkcija*

Kako bi se funkcija *OnSubmit()* mogla izvršiti, potrebno ju je definirati. Definira se unutar datoteke *register.component.ts* te mora pozvati funkciju iz klase *QuizService*. Kako bi to bilo moguće, potrebno je deklarirati objekt klase *QuizService* naziva *quizService* unutar konstruktora zajedno sa objektom klase *Router* s nazivom *route* koji je potreban za usmjeravanje. Preko objekta *quizService* se može pristupiti funkciji *insertParticipant()* na kojoj se poziva metoda *subscribe()*. Unutar metode *subscribe()* se deklarira *arrow* funkcija s jednim parametrom *data*. Nakon uspješne registracije, potrebno je korisnika usmjeriti prema *difficulty* komponenti. Usmjeravanje na drugu komponentu se obavlja preko funkcije *navigate()*. Za potrebe autorizacije i autentifikacije, korišten je *localStorage*. Ako sve funkcionira, ime i e-mail sudionika bi se trebali pojaviti unutar baze podataka u tablici *Participant*. Nakon uspješne registracije je spremljen i sličan JSON *string* objekt unutar *localStorage*-a s ključem *participant*. No naravno, prije toga je bilo potrebno očistiti taj *localStorage*. Budući da je za ove potrebe korišten *HttpClient*, potrebno je dodati *HttpClientModule* unutar datoteke *app.module.ts*. Potrebno ga je prvo uvesti, a zatim dodati u polje *imports*. Također je u istoj datoteci potrebno dodati i klasu *QuizService* unutar polja *providers*. Funkcija *OnSubmit()* se nalazi na slici **3.20.**.

```
OnSubmit(name: string, email:string){
  this.quizService.insertParticipant(name, email).subscribe(
    (data: any) =>{
      localStorage.clear();
      localStorage.setItem('participant', JSON.stringify(data));  //zbog provjere autorizacije
      this.route.navigate(['/difficulty']);
    }
  );
}
```

Sl. 3.20. *OnSubmit() funkcija*

Sada na red dolazi provjera autorizacije i autentifikacije preko ključa *participant*. Implementirati će se klijentska strana autentifikacije koristeći ključ *participant* iz *localStorage*-a. Autentifikaciju je potrebno implementirati jer bez nje bio svatko mogao pristupiti kvizu pod tuđim imenom te registracija ne bi uopće imala smisla. Zbog toga je potrebno kreirati mapu *auth* te u njoj kreirati datoteku *auth.guard.ts* pomoću naredbe *ng g g ime_datoteke* (*auth*). *g* je skraćeno od *generate*, te je drugo *g* skraćeno od *guard*. Za izvedbu autorizacije, korištena je funkcija *canActivate()* u kojoj se provjerava je li korisnik registriran. Ako korisnik nije registriran, usmjerivač usmjerava korisnika natrag na komponentu *register*. Kako bi usmjeravanje bilo moguće, potrebno je unutar konstruktora deklarirati objekt klase *Router* s nazivom *route*. Da bi aplikacija znala da postoji klasa *AuthGuard*, potrebno ju je uvesti unutar datoteke *app.module.ts* i postaviti ju unutar polja *providers*. Također je potrebno unutar datoteke *routes.ts* naznačiti autentifikaciju na svim komponentama koje dolaze poslije komponente *register*. Primjer autentifikacije komponente se nalazi na slici **3.21.**, dok se primjer funkcije *canActivate()* nalazi na slici **3.22.**.

```
path:'difficulty', component:DifficultyComponent, canActivate :  [AuthGuard]},
```

Sl. 3.21. *Autentifikacija komponente difficulty*

```
export class AuthGuard implements CanActivate {
  constructor(private route : Router){}
  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): boolean {
      if (localStorage.getItem('participant') != null){
        return true;
      }
      else{
        this.route.navigate(['/register']);
        return false;
      }
  }

}
```

Sl. 3.22. *Funkcija canActivate()*

Time je završena komponenta *register*. Sada na red dolazi komponenta *difficulty* u kojoj dolazi do odabira težine kviza. U predlošku *difficulty.component.html* se nalazi selektor komponente *navbar* te je kreirana jednostavna struktura obrasca koji sadrži tri gumba. Na svakom gumbu je definiran klik događaj, tj. klikom na određeni gumb se poziva definirana funkcija. Sve tri funkcije imaju ulogu usmjeravanja na pitanja kviza određene težine, odnosno na kviz komponente. Sve tri funkcije su prikazane na slici **3.23.**, dok je izgled predloška na slici **3.24.**.

```
OnClickEasy(){
  this.route.navigate(['/easyquiz']);
}

OnClickRegular(){
  this.route.navigate(['/regularquiz']);
}

OnClickHard(){
  this.route.navigate(['/hardquiz']);
}
```

Sl. 3.23. *Funkcije za usmjeravanje unutar difficulty komponente*

```
<form>
    <div class="row">
        <div class="input-field col s12" (click)="OnClickEasy()">
            <button class="btn-large btn-submit" type="submit">Lagana</button>
        </div>
        <div class="input-field col s12" (click)="OnClickRegular()">
            <button class="btn-large btn-submit" type="submit">Normalna</button>
        </div>
        <div class="input-field col s12" (click)="OnClickHard()">
            <button class="btn-large btn-submit" type="submit">Teška</button>
        </div>
    </div>
</form>
```

Sl. 3.24. *Predložak difficulty komponente*

Nakon *difficulty* komponente, na red dolazi kviz komponenta (*EasyQuiz*, *RegularQuiz*, *HardQuiz*) u kojoj će biti prikazano jedno po jedno pitanje. Budući da su sve težine kviza prikazane na isti način, za ogledni primjer će se ponovno uzeti komponenta *EasyQuiz*.

Za prikaz pitanja je potrebno konzumirati Web API metodu *GetEasyQuestions()* iz *EasyQuizController*-a. Stoga je za početak potrebno definirati funkciju za konzumiranje te metode unutar uslužne klase *QuizService*. Ta funkcija se naziva *getEasyQuestions()* i ona zove GET funkciju od HTTP objekta kojoj je parametar zadani *rootUrl* na kojeg se dodaje ruta Web API metode *GetEasyQuestions()* („/api/EasyQuestions“). Definiranu funkciju *getEasyQuestions()* je moguće pozvati unutar datoteke *easyquiz.component.ts*, točnije unutar *ngOnInit() kuke* životnog

ciklusa programa. No prije toga, potrebno je također uvesti i deklarirati objekt klase *Router* s nazivom *route* te objekt klase *QuizService* s nazivom *quizService*. Funkciju *getEasyQuestions()* je moguće pozvati preko objekta *quizService* te na nju također pozvati metodu *subscribe()*. Unutar metode *subscribe()* se također deklarira *arrow* funkcija s jednim parametrom *data*. Budući da Web API metoda *GetEasyQuestions()* vraća deset nasumičnih pitanja, ta pitanja je potrebno spremiti unutar Angular projekta. Zato je unutar uslužne klase *QuizService* potrebno definirati par svojstava. Prvo takvo svojstvo je polje *qns* unutar kojeg će se pohraniti deset nasumično izvučenih pitanja. Drugo svojstvo s nazivom *seconds* zapravo predstavlja ukupno vrijeme koje će sudionik provesti rješavajući kviz. Pomoću trećeg svojstva imena *timer* će se brojiti proteklo vrijeme rješavanja. Posljednje svojstvo imena *qnProgress* predstavlja broj pitanja kojeg korisnik rješava. Pitanja koja metoda *GetEasyQuestions()* vraća je potrebno pohraniti unutar svojstva *qns* kojem se pristupa pomoću objekta klase *quizService*. To se radi na način da se svojstvu *qns* pridruži parametar *arrow* funkcije s nazivom *data* unutar tijela funkcije.

```
getEasyQuestions(){
  return this.http.get(this.rootUrl + '/api/EasyQuestions');
}
```

Sl. 3.25. *Funkcija getEasyQuestions()*

Za brojanje vremena je također potrebno definirati funkciju. Takva funkcija je s*tartTimer()* funkcija koja inicijalizira svojstvo *timer* pomoću funkcije *setInterval()* unutar koje se povećava svojstvo *seconds* za 1 u intervalu od jedne sekunde. Izgled s*tartTimer()* funkcije se može vidjeti na slici **3.26.**.

```
startTimer(){
  this.quizService.timer = setInterval(() => {  //inicijalizacija timera
    this.quizService.seconds++;                 //inkrementacija sekundi
    localStorage.setItem('seconds', this.quizService.seconds.toString());
  }, 1000);                                     //u intervalu od 1 sekunde (1000ms)
}
```

Sl. 3.26. *startTimer() funkcija*

Nakon pridruživanja parametra *data* svojstvu *qns*, potrebno je pozvati tek kreiranu funkciju s*tartTimer()*. No prije toga je potrebno inicijalizirati svojstva *seconds* i *qnProgress* na vrijednost 0. Ovo je sve u redu što se tiče logičke strane aplikacije. No postoji veliki problem u smislu ustrajnosti podataka. To znači da ako primjerice usred rješavanja kviza korisnik osvježi web stranicu, svi podaci, odgovorena pitanja i rezultati će biti izgubljeni. Stoga se koristi *localStorage* kojim se ažuriraju i spremaju svi podaci i svojstva. Primjer spremanja podataka unutar *localStorage*-a je vidljiv na slikama **3.26.** i **3.27.**.

```
ngOnInit(){
  if (parseInt(localStorage.getItem('seconds')) > 0){
    this.quizService.seconds = parseInt(localStorage.getItem('seconds'));
    this.quizService.qnProgress = parseInt(localStorage.getItem('qnProgress'));
    this.quizService.qns = JSON.parse(localStorage.getItem('qns'));
    if (this.quizService.qnProgress == 10){
      this.route.navigate(['/easyresult']);
    }
    else{
      this.startTimer();
    }
  }
  else{
    this.quizService.seconds = 0;       //inicijalizacija na nulu(0)
    this.quizService.qnProgress = 0;
    this.quizService.getEasyQuestions().subscribe(
      (data: any) =>{
        this.quizService.qns = data;    //unutar "data" se nalazi 10 nasumičnih pitanja
        this.startTimer();
      }
    );
  }
```

Sl. 3.27. *Pozivanje funkcije getEasyQuestions() unutar datoteke easyquiz.component.ts*

Vidljivo je korištenje *localStorage*-a za ispitivanje je li kviz već pokrenut. Ako je svojstvo *seconds* veće od 0, ažurirat će se *quizService* svojstva *seconds*, *qnProgress* te svojstvo *qns*. Ako je svojstvo *qnProgress* jednako 10, odnosno ako je korisnik odgovorio na deset pitanja, usmjerivač će korisnika usmjeriti na komponentu *easyresult*. Ako korisnik nije odgovorio na svih deset pitanja, preglednik će jednostavno započeti gdje je stao i ponovno pokrenuti brojanje vremena. Zatim ako korisnik nije započeo rješavanje kviza, jednostavno će se sva svojstva inicijalizirati na 0 i ponovno pokrenuti proces dohvaćanja pitanja.

Za dizajn izgleda komponente *easyquiz*, potrebno je ući u datoteku *easyquiz.component.html*. Za početak je potrebno selektor komponente *navbar* zalijepiti unutar predloška. Ispod toga je potrebno prikazati deset nasumičnih pitanja, jedno po jedno. Koristi se strukturna direktiva *ngIf* kao uvjet za prikaz pitanja samo onda kada su nasumična pitanja dohvaćena i kada nisu riješena sva pitanja. Redni broj je prikazan preko vezanja podataka, točnije interpolacije niza:

{{quizService.qnProgress+1}}. Jedinica je dodana iz razloga što *qnProgress* započinje od nule (0). Ispod rednog broja pitanja je prikazan opis pitanja također interpolacijom niza: *{{quizService.qns[quizService.qnProgress].eQn}}*. Svojstvo *eQn* predstavlja sam tekst pitanja koji se nalazi u tablici *EasyQuestion*. Ispod teksta pitanja je potrebno prikazati sliku pitanja ako ga pitanje sadrži. Također se koristi strukturna direktiva *ngIf* kod prikaza slike pitanja. (**Slika 3.28.**) Funkcionira na način da ako postoji slika, ona će biti prikazana, u suprotnom neće. Pod atribut *src* je postavljena adresa slike određenog pitanja unutar mape *Images* koja se nalazi u Web API-u. Zatim je potrebno prikazati opcije za odgovor na pitanje. Za opcije su korištene strukturne direktive *ngFor* i *ngIf*. Pomoću direktive *ngFor* iterativno se lista kroz opcije (*string* polje *Options* unutar EasyQuizController-*a*). Svakom iteracijom se svaka opcija pohranjuje unutar varijable *option*, a svaki indeks unutar varijable *i*. Direktivom *ngIf* se uvjetuje da ako postoji opcija, da ju se prikaže, u suprotnom ne. Opcije se prikazuju u listama te je također na opcijama definiran klik događaj na čiji klik se poziva funkcija *Answer()*. Za prikaz opcija je korištena interpolacija niza: *{{option}}*. (**Slika 3.29.**)

```
<span class="card-title">{{quizService.qnProgress+1}}</span>
<p>{{quizService.qns[quizService.qnProgress].eQn}}</p>
<div *ngIf="quizService.qns[quizService.qnProgress].eImageName != null">
    <img class="center" [src]="quizService.rootUrl + '/Images/' + quizService.qns[quizService.qnProgress].eImageName"
</div>
```

Sl. 3.28. *Primjer interpolacije niza i strukturna direktiva ngIf kod prikaza slike*

```
<ng-container *ngFor="let option of quizService.qns[quizService.qnProgress].Options; let i = index">
    <li *ngIf="option != null" class="collection-item" (click)="Answer(quizService.qns[quizService.qnProgress].eQnID, i)">
        {{option}}
    </li>
</ng-container>
```

Sl. 3.29. *Direktive ngFor i ngIf za prikazivanje opcija odgovora*

Kako bi se prikazalo vrijeme koje je pokrenuto funkcijom s*tartTimer()*, potrebno je unutar *QuizService* klase dodati funkciju *displayTimeElapsed()* koja pomoću funkcije *Math.floor()* vraća vrijednosti svojstva *seconds* koje su prikazane u formatu sati, minuta i sekunda. (**Slika 3.31.**) U predlošku vrijeme je prikazano pomoću interpolacije niza: *{{quizService.displayTimeElapsed()}}*. Zatim je prikazana traka za napredak (engl. *progress bar*) čiji dizajn je preuzet s *front*-end okvira Materialize CSS. Pomoću svojstva *width* mijenja se napredak trake. Svojstvo *width* je definirano na način da se pristupi broju pitanja (*quizService.qnProgress+1*) te ga se pomnoži s brojem 10 kako bi se dobio postotak. Primjer trake za napredak i prikaza vremena se nalazi na slici **3.30.**.

```
<div class="row" *ngIf="quizService.qns && this.quizService.qnProgress != 10">
    <div class="col s8 offset-s2">
        <span class="time">Proteklo vrijeme: {{quizService.displayTimeElapsed()}}</span>
        <div class="progress">
            <div class="determinate" [style.width.%]="(quizService.qnProgress+1)*10"></div>
        </div>
```

Sl. 3.30. *Prikaz vremena i trake za napredak u predlošku easyquiz komponente*

```
displayTimeElapsed(){
  return Math.floor(this.seconds / 3600) + ':' + Math.floor(this.seconds / 60) + ':' + Math.floor(this.seconds % 60);
}
```

Sl. 3.31. *displayTimeElapsed() funkcija unutar klase QuizService*

Da bi funkcija *Answer()* funkcionirala, potrebno ju je definirati unutar komponente *easyquiz.component.ts*. Funkcija *Answer()* sadrži dva parametra od kojih jedan predstavlja ID trenutnog pitanja, a drugi odabir korisnika. U funkciju je dodano novo svojstvo naziva *answer* unutar kojeg se sprema indeks odabrane opcije (ako je odabrana prva opcija, sprema se indeks 0, ako je druga onda indeks 1, itd.). Nakon toga se povećava svojstvo *qnProgress* sve dok ne dođe do 10. Tada se čisti timer i usmjerivač usmjerava korisnika prema komponenti *easyresult*. Da bi podaci ostali ustrajni, odnosno da svi podaci ne budu izgubljeni prilikom osvježenja stranice, koristi se također *localStorage* kojim se spremaju podaci i svojstva. U funkciji *Answer()* će se spremiti svojstva *qns* i *qnProgress*.

```
Answer(gID, choice){
  this.quizService.qns[this.quizService.qnProgress].answer = choice;
  localStorage.setItem('qns', JSON.stringify(this.quizService.qns));
  this.quizService.qnProgress++;
  localStorage.setItem('qnProgress', this.quizService.qnProgress.toString());
  if (this.quizService.qnProgress == 10){
    clearInterval(this.quizService.timer);
    this.route.navigate(['/easyresult']);
  }
}
```

Sl. 3.32. *Answer() funkcija*

Nakon što se riješi kviz, potrebno je prikazati ostvareni rezultat korisnika zajedno sa deset pitanja i točnim odgovorima. Također je potrebno prikazati ime korisnika te proteklo vrijeme rješavanja samog kviza. Da bi se to ostvarilo, potrebno je konzumirati Web API metodu *GetEasyAnswers()* koja kao parametar sadrži polje ID-ova. Metoda vraća polje odgovora s istim

redoslijedom kakvim su pitanja bila raspoređena. Unutar klase *QuizService* je potrebno kreirati funkciju *getEasyAnswers()* kako bi se Web API metoda mogla konzumirati. Unutar nje se deklarira varijabla *body* koja je inicijalizirana poljem koje sadrži ID-ove pitanja. Kako bi se dohvatili ID-ovi pitanja (*eQnID*), korištena je metoda *map*. Na kraju funkcija zove POST funkciju od HTTP objekta u kojoj je prvi parametar zadani *rootUrl* na kojeg se dodaje ruta Web API metode *GetEasyAnswers()* („/api/EasyAnswers“). Drugi parametar joj je definirana varijabla *body* koja sadrži polje ID-ova. Time funkcija vraća polje odgovora te pomoću te funkcije je moguće procijeniti točnost odgovora korisnika. (**Slika 3.33.**) Za procjenu točnih odgovora, potrebno je kreirati unutar klase *QuizService* novo svojstvo naziva *correctAnswerCount* te ga inicijalizirati na vrijednost 0.

```
getEasyAnswers(){
  var body = this.qns.map(x => x.eQnID);                          //u "body" stavljamo ID-eve od 10 nasumičnih pitanja
  return this.http.post(this.rootUrl + '/api/EasyAnswers', body);   //zahtjev ispisuje polje pitanja (eAnswer)
}
```

Sl. 3.33. *Funkcija getEasyAnswers()*

Što se tiče datoteke *easyresult.component.ts*, za početak je potrebno unutar datoteke uvesti i unutar konstruktora deklarirati *QuizService* objekt *quizService* i *Router* objekt *route*. Unutar *ngOnInit() kuke* životnog ciklusa je potrebno pozvati funkciju *getEasyAnswers()* preko objekta *quizService*. Također je na funkciju *getEasyAnswers()* potrebno pozvati metodu *subscribe()*. Unutar metode *subscribe()* se također deklarira *arrow* funkcija s jednim parametrom *data*. U tijelu funkcije se za početak svojstvo *correctAnswerCount* „resetira“, odnosno postavi na nulu. Zatim se prolazi kroz polje od deset nasumičnih pitanja (*qns*) pomoću metode *forEach()* koja sadrži *arrow* funkciju s dva parametra, element *e* i indeks *i*. Unutar *arrow* funkcije se uspoređuju korisnikov izbor odgovora (*e.answer*) i točan odgovor iz parametra *data* (*data[i]*). Unutar svojstva *answer* je spremljen korisnikov izbor između više opcija preko funkcije *Answer()*. Ako su korisnikov izbor i točan odgovor jednaki, uvećava se svojstvo *correctAnswerCount* za vrijednost 1 pomoću unarnog operatora inkrementiranja (++). Kako bi se prikazalo svih deset pitanja s njihovim točnim odgovorima, potrebno je dodati još jedno svojstvo unutar polja *qns* koje se naziva *correct*. I u ovom slučaju se koristi *localStorage* za spremanje rezultata i održavanja ustrajnosti podataka. Ako je korisnik odgovorio na svih deset pitanja, odnosno ako je riješio cijeli kviz, svojstva *seconds*, *qnProgress* i *qns* se pohranjuju unutar *localStorage*-a. Korisniku prilikom osvježenja web stranice ostaju postignuti rezultati, ime korisnika, proteklo vrijeme rješavanja kviza te deset pitanja i njihovi točni odgovori. Primjer korištenja metode *getEasyAnswers()* se nalazi na slici **3.34.**.

```
ngOnInit(){
  if (parseInt(localStorage.getItem('qnProgress')) == 10){
    this.quizService.seconds = parseInt(localStorage.getItem('seconds'));
    this.quizService.qnProgress = parseInt(localStorage.getItem('qnProgress'));
    this.quizService.qns = JSON.parse(localStorage.getItem('qns'));
    this.quizService.getEasyAnswers().subscribe(
      (data: any) => {
        this.quizService.correctAnswerCount = 0;
        this.quizService.qns.forEach((e, i) => {
          if(e.answer == data[i]){
            this.quizService.correctAnswerCount++;
          }
          e.correct = data[i];
        });
      }
    );
  }
  else{
    this.route.navigate(['/easyquiz']);
  }
}
```

Sl. 3.34. *Primjer korištenja getEasyAnswers() unutar easyresult.components.ts datoteke*

Nakon toga, potrebno je dizajnirati predložak *easyresult* komponente. Potrebno je otvoriti datoteku *easyresult.component.html* te za početak dodati selektor komponente *navbar* unutar predloška. Ispod toga je potrebno dodati naslov pomoću kojega se može označiti kraj kviza. Zatim je potrebno prikazati rezultat kviza, odnosno ime korisnika koje je uneseno prilikom registracije, postignuti rezultat pri rješavanju kviza te proteklo vrijeme. Kao znak uspjeha, dodana je slika trofeja čija je slika smještena unutar mape *img* koja je kreirana unutar mape *assets*. Ime korisnika je prikazano interpolacijom niza: *{{quizService.getParticipantName()}}*, kao i proteklo vrijeme: *{{quizService.displayTimeElapsed()}}*. Konačni rezultat je također prikazan interpolacijom niza kao točan broj odgovora od mogućih deset: *{{quizService.correctAnswerCount}}/10*. Budući da funkcija *getParticipantName()* nije kreirana, potrebno ju je kreirati unutar klase *QuizService*. Tijekom registracije su preko JSON *string* objekta spremljeni podaci korisnika unutar *localStorage*-a pod ključem *participant* preko funkcije *OnSubmit()*. Stoga je unutar funkcije *getParticipantName()* kreirana varijabla *participant* kojoj je pridružen JSON *string* objekt koji je ponovno prebačen u JSON. Podaci su dohvaćeni preko metode *getItem()* kojoj je kao parametar predan ključ s nazivom *participant.* Funkcija vraća ime korisnika tako što preko varijable *participant* pristupa stupcu *Name* unutar tablice *Participant*.

```
getParticipantName(){
  var participant = JSON.parse(localStorage.getItem('participant'));
  return participant.Name;
}
```

Sl. 3.35. *getParticipantName() funkcija*

Također su dodana dva gumba. Jedan gumb služi za predaju rezultata koji se upisuju unutar baze podataka pod tablicu *Participant*, dok drugi gumb služi za ponovno pokretanje kviza, odnosno ponovni odabir težine kviza. Da bi gumbi obavljali zadane funkcije, potrebno je za svaki gumb definirati klik događaj.

```
<div class="col s8 offset-s2">
    <h2 class="header">Završen kviz - lagana težina!</h2>
    <div class="card horizontal">
        <div class="card-image">
            <img src="../../assets/img/trophy.png" style="width: 250px; height: 250px;">
        </div>
        <div class="card-stacked">
            <div class="card-content data">
                <h4>{{quizService.getParticipantName()}}</h4>
                <h3>{{quizService.correctAnswerCount}}/10</h3>
                <span>Proteklo vrijeme: {{quizService.displayTimeElapsed()}}</span>
            </div>
            <div class="card-action">
                <button class="waves-effect waves-light btn-small" (click)="OnEasySubmit()">Predaj</button>
            </div>
        </div>
        <a class="btn-floating halfway-fab waves-effect waves-light red" (click)="Restart()">
            <i class="material-icons">replay</i>
        </a>
    </div>
```

Sl. 3.36. *Prikazivanje svojstava pomoću interpolacije niza te definiranje dvaju klik događaja*

Prije nego što se kreira klik događaj, potrebno je konzumirati Web API metodu naziva *UpdateScoreTime()*. Da bi to bilo moguće, potrebno je kreirati funkciju *submitEasyScoreTimeDifficulty()* unutar koje se za početak kreira varijabla *body*. Varijabli *body* se pridružuju podaci korisnika preko metode *getItem()* koja kao parametar sadrži ključ s nazivom *participant*. Preko varijable *body* se pristupa stupcima unutar tablice *Participant* kao što su *Score*, *TimeSpent* i *Difficulty* te im se pridružuju vrijednosti *correctAnswerCount* i *seconds*. Stupcu *Difficulty* se pridružuje niz znakova 'Lagana težina' koja predstavlja odabranu težinu korisnika. Navedena funkcija također koristi HTTP metodu POST kojoj je prvi parametar zadani *rootUrl* na kojeg se dodaje ruta Web API metode *UpdateScoreTime()* („api/UpdateScoreTime“), dok joj je drugi parametar varijabla *body*.

```
submitEasyScoreTimeDifficulty(){
  var body = JSON.parse(localStorage.getItem('participant'));
  body.Score = this.correctAnswerCount;
  body.TimeSpent = this.seconds;
  body.Difficulty = 'Lagana težina';
  return this.http.post(this.rootUrl + "/api/UpdateScoreTime", body);
}
```

Sl. 3.37. *submitEasyScoreTimeDifficulty() funkcija*

Klik događaju na gumbu kojim se predaju rezultati je predana funkcija *OnEasySubmit()*. Unutar komponente *easyresult* je kreirana funkcija *OnEasySubmit()* koja poziva kreiranu funkciju *submitEasyScoreTimeDifficulty()* preko objekta *quizService*. Na funkciju *submitEasyScoreTimeDifficulty()* je potrebno pozvati i metodu *subscribe()* koja u sebi poziva zasada nedefiniranu funkciju *Restart()*. Nakon uspješnog ažuriranja ostvarenih bodova, proteklog vremena i odabrane težine potrebno je ponovno pokrenuti kviz. Stoga je potrebno kreirati dodatnu funkciju naziva *Restart()* pomoću koje se sva svojstva putem *localStorage*-a postavljaju na vrijednost 0 te se polja prazne. Također funkcija usmjerava korisnika na komponentu *difficulty* pomoću metode *navigate()* kojom se pristupa pomoću objekta *route*. Na drugom gumbu koji služi za ponovno pokretanje kviza je također potrebno definirati klik događaj. Kliku događaja je potrebno predati tek kreiranu funkciju *Restart()*. Izgled drugog gumba je prikazan pomoću *materialize* ikone s nazivom *replay*.

```
OnEasySubmit(){
  this.quizService.submitEasyScoreTimeDifficulty().subscribe(() => {
    this.Restart();
  });
}

Restart(){
  localStorage.setItem('qnProgress', "0");
  localStorage.setItem('qns', "");
  localStorage.setItem('seconds', "0");
  this.route.navigate(['/difficulty']);
}
```

Sl. 3.38. *Funkcije OnEasySubmit() i Restart()*

Nakon završetka usavršavanja klik događaja, na red dolazi prikaz svih deset pitanja sa točnim odgovorima koji se prikazuje slično kao i u predlošku *easyquiz* komponente. Za početak je potrebno dodati naslov kako bi se označio početak gdje započinje prikaz pitanja. Zatim se pomoću strukturne direktive *ngFor* provodi iteracija kroz polje *qns* gdje se svako pitanje pohranjuje unutar varijable *qn*, a svaki indeks unutar varijable *i*. Samom tekstu pitanja i broju pitanja se pristupa preko interpolacije niza: *{{qn.eQn}}* i *{{i+1}}*. Također se koristi i strukturna direktiva *ngIf* koja prikazuje sliku pitanja ako ona zapravo postoji. Ispod svih pitanja je potrebno prikazati određene opcije za odgovoriti. Za prikaz opcija su korištene dvije strukturne direktive *ngFor* i *ngIf*. Pomoću direktive *ngFor* se provodi iteracija kroz polje *Options* te se svaka opcija pohranjuje unutar varijable *option*, dok se indeks pohranjuje unutar varijable *j*. Pomoću direktive *ngIf* se prikazuju opcije samo ako postoje pomoću interpolacije niza: *{{option}}*. Za detekciju je li odabrana opcija

zapravo točan odgovor, koristi se još jedna strukturna direktiva *ngIf* pomoću koje se provjerava je li odabrana opcija pod indeksom *j* jednaka točnom odgovoru koji je spremljen unutar svojstva *correct*. Točan odgovor će se označiti sa zelenom strelicom koja je također prikazana *materialize* ikonom s nazivom *check_circle*. Posljednjom direktivom *ngIf* će se provjeriti je li korisnik točno odgovorio. Ako nije točno odgovorio, na njegovom odabiru će pisati „Netočno“.

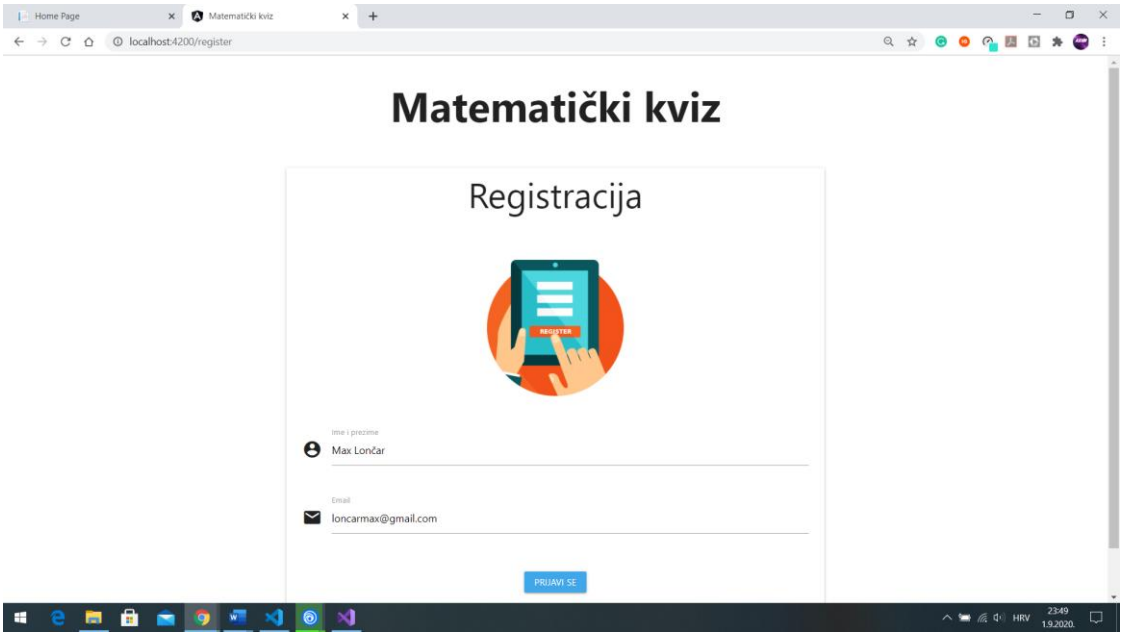
```
<h3>Pitanja s točnim odgovorima</h3>
<ng-container *ngFor="let qn of quizService.qns; let i = index;">
    <div class="card blue">
        <div class="card-content white-text">
            <span class="card-title">{{i+1}}</span>
            <p>{{qn.eQn}}</p>
            <div *ngIf="qn.eImageName != null">
                <img class="center" [src]="quizService.rootUrl + '/Images/' + qn.eImageName" style="width: 350px; height: 400px;">
            </div>
        </div>

        <div class="card-action">
            <ul class="collection">
                <ng-container *ngFor="let option of qn.Options; let j = index;">
                    <li *ngIf="option != null" class="collection-item">
                        {{option}}
                        <span class="badge" *ngIf="qn.correct == j">
                            <i class="material-icons green-text">check_circle</i>
                        </span>
                        <span class="badge red-text" *ngIf="qn.answer == j && qn.correct != j">Netočno</span>
                    </li>
                </ng-container>
            </ul>
        </div>
```

Sl. 3.39. *Prikaz pitanja s točnim odgovorima*

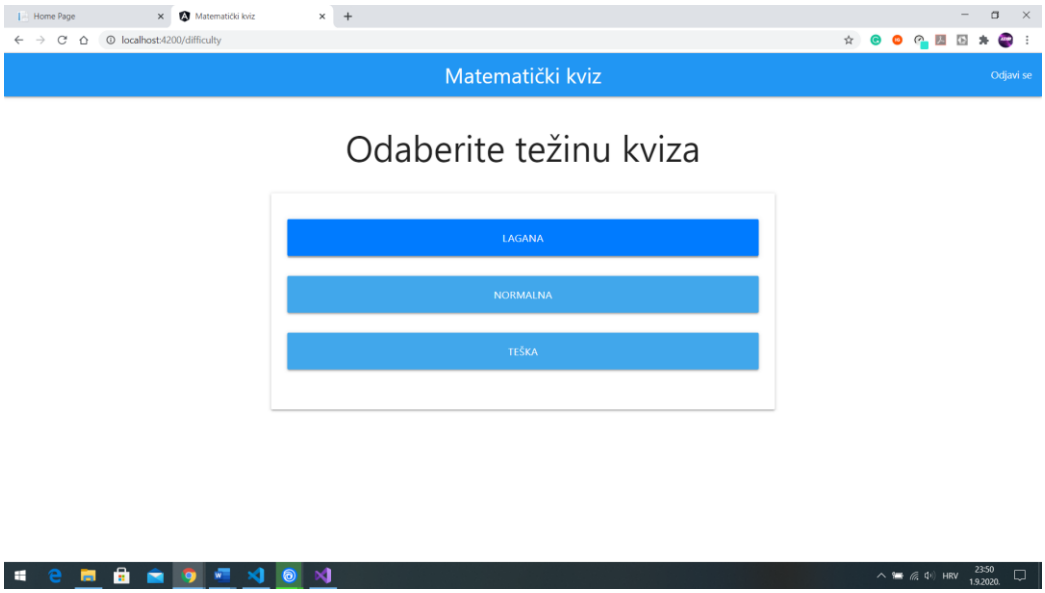
4. IZGLED ANGULAR WEB APLIKACIJE MATEMATIČKI KVIZ

Izgled Angular web aplikacije je poprilično jednostavan. Pokretanjem web aplikacije, otvara se zadana *register* komponenta unutar koje korisnik unosi vlastito ime (i prezime) i e-mail. Klikom na gumb „Prijavi se“ korisnik se usmjeruje na komponentu *difficulty* te se registrira na način da se njegovo ime i e-mail upisuju unutar baze podataka pod tablicom *Participant*.



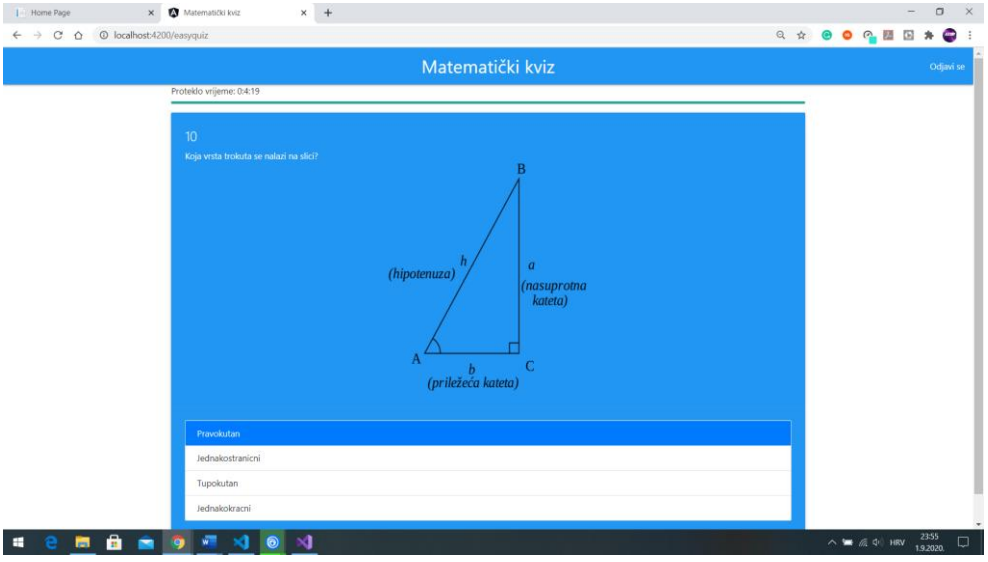
Sl. 4.1. *Registracija korisnika*

Dolaskom na komponentu *difficulty*, korisnik bira težinu kviza. Komponenta *difficulty* sadrži navigacijsku traku koja ima svoju zasebnu komponentu *navbar* na kojoj se nalazi naziv kviza te gumb za odjavu korisnika. Pritiskom na gumb „Odjavi se“, korisnik se usmjeruje prema *register* komponenti. Komponenta *difficulty* također sadrži tri gumba za odabir težine o čemu ovisi daljnja putanja kviza. Svaka težina ima svoje komponente. „Lagana težina“ sadrži komponente *easyquiz* i *easyresult*, „Normalna težina“ sadrži komponente *regularquiz* i *regularresult* dok „Teška težina“ sadrži komponente *hardquiz* i *hardresult*. Sve kviz komponente su slične kao i sve komponente rezultata. Daljnji izgled kviza će biti prikazan iz perspektive odabira „Lagane težine“.



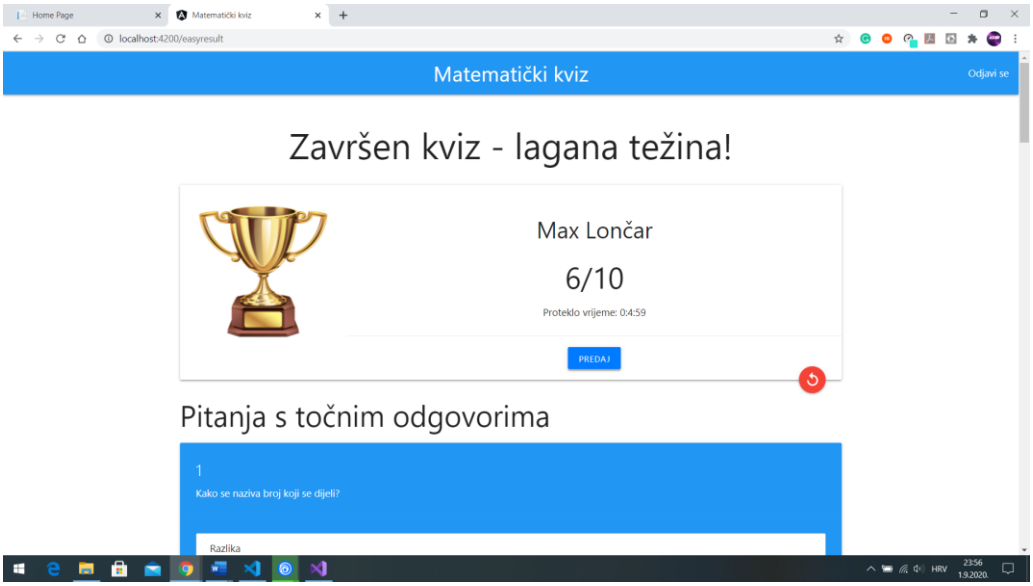
Sl. 4.2. *Odabir težine kviza*

Odabirom „Lagane težine“, otvara se komponenta *easyquiz* u kojoj se prikazuje jedno po jedno pitanje. Unutar komponente *easyquiz* je također prikazana navigacijska traka, trenutno vrijeme rješavanja, traka za napredak, redni broj i tekst pitanja, slika (ako je pitanje sadrži) te četiri (ili manje) opcije od kojih treba odabrati jednu.

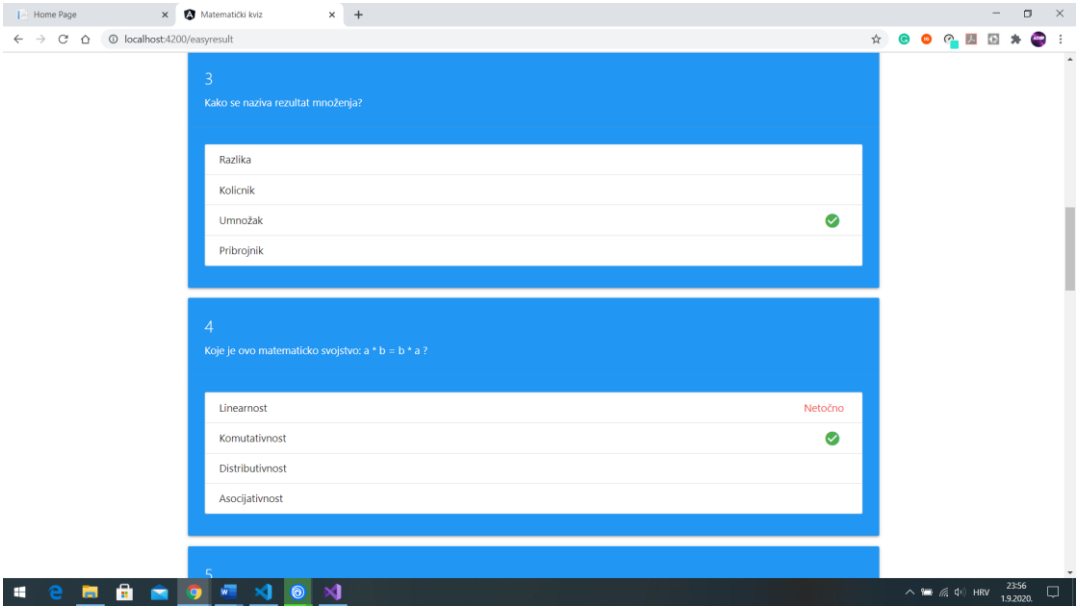


Sl. 4.3. *Rješavanje kviza*

Nakon što se riješi kviz, otvara se komponenta *easyresult* koja također sadrži navigacijsku traku, naslov da je kviz završen i naziv kategorije, ime korisnika, proteklo vrijeme rješavanja te ostvareni rezultat. Također se nalaze i dva gumba. Na jednom gumbu piše „Predaj“, čijim pritiskom se svi podaci zapisuju unutar baze podataka pod tablicom *Participant*. Drugi gumb ima izgled *restart* gumba čijim pritiskom se korisnik usmjeruje na odabir težine kviza kako bi ponovno krenuo rješavati kviz. Ispod toga se nalaze svih deset pitanja i točni odgovori na njih.



Sl. 4.4. *Rezultati kviza*



Sl. 4.5. *Točni odgovori na pitanja*

5. ZAKLJUČAK

Cilj postavljen u zadatku završnog rada je izrada Angular web aplikacije matematički kviz koja korisnicima omogućuje rješavanje matematičkih pitanja različitih težina. Za izradu aplikacije je bilo potrebno smisliti pitanja iz područja matematike te ih rasporediti po težinama i pohraniti unutar baze podataka, dobro proučiti Angular i postupak izrade Web API-a.

Baza podataka je kreirana na jednostavan način s četiri tablice *EasyQuestion, RegularQuestion, HardQuestion* i *Participant*. U tablici *Participant* se nalaze podaci o korisniku koji je pristupio rješavanju kviza (ime i e-mail korisnika, ostvareni rezultat, proteklo vrijeme rješavanja kviza i težina kviza). U tablicama *EasyQuestion, RegularQuestion* i *HardQuestion* se nalaze pitanja kviza. Najveći posao kod baze podataka predstavlja unos pitanja za svaku tablicu.

Izrada ASP.NET Web API-a je zapravo najteži zadatak web aplikacije. Za početak je potrebno kreirati model datoteke te model entiteta baze podataka koja uključuje sve četiri kreirane tablice. Za svaku tablicu je potrebno kreirati kontroler (*EasyQuizController, RegularQuizController, HardQuizController, ParticipantController*) unutar kojega se kreiraju Web API metode. Za svaku metodu je bilo potrebno odabrati tip HTTP metode te napisati rutu. Problem je zapravo predstavljao postupak izrade Web API-a i sintaksa koju je bilo potrebno dobro proučiti i istražiti kako bi se uspješno kreirala Web API metoda.

Angular web aplikacija se većinom uređivala unutar Microsoft Visual Studio Code uređivačkog programa. Najvažniji dijelovi aplikacije su zapravo komponente u kojima se programirala sva logika aplikacije te predlošci u kojima se uređivala struktura aplikacije. Vrlo bitan dio aplikacije je također i uslužna klasa preko koje se konzumiralo Web API metode. Da bi bilo moguće se prebacivati između komponenti, definirane su rute za svaku komponentu unutar jedne datoteke. Što se tiče sintakse okvira Angular, potrebno ju je dosta proučavati i testirati kako bi se njome ovladalo, pogotovo za nekoga tko se nije susretao sa sličnim programskim jezikom kao što je to TypeScript. Pored toga, svaki zadatak je obavljen bez nekih većih poteškoća.

Aplikacija je uspješno napravljena i potpuno funkcionalna. Registracijom korisnik dobiva pristup odabira težine kviza, nakon čega može krenuti rješavati kviz. U tijeku rješavanja se računa vrijeme koje je korisniku potrebno da riješi kviz. Nakon što riješi kviz, korisniku se prikažu ostvareni rezultati zajedno s njegovim imenom i proteklim vremenom. Kviz se može unaprijediti na način da se ne računa vrijeme potrebno za rješavanje kviza, već da se kviz ograniči na određeno vrijeme.

LITERATURA

[1] Što je WEB aplikacija? [online], dostupno na: https://geek.hr/pojmovnik/sto-je-web-aplikacija/ [pristupljeno 2. rujna 2020.]

[2] Što je desktop aplikacija? [online], dostupno na: https://geek.hr/pojmovnik/sto-je-desktop-aplikacija/ [pristupljeno 2. rujna 2020.]

[3] Angular (web framework) [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Angular_(web_framework) [pristupljeno 2. rujna 2020.]

[4] Single-page application [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Single-page_application [pristupljeno 2. rujna 2020.]

[5] TypeScript [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/TypeScript [pristupljeno 2. rujna 2020.]

[6] Introduction to components and templates [online], Angular, dostupno na: https://angular.io/guide/architecture-components [pristupljeno 2. rujna 2020.]

[7] Introduction to modules: NgModule metadata [online], Angular, dostupno na: https://angular.io/guide/architecture-modules [pristupljeno 2. rujna 2020.]

[8] Event binding [online], Angular, dostupno na: https://angular.io/guide/event-binding [pristupljeno 2. rujna 2020.]

[9] Two-way binding: Basics of two-way binding [online], Angular, dostupno na: https://angular.io/guide/two-way-binding [pristupljeno 2. rujna 2020.]

[10] Angular Dana Binding: What is Dana Binding In Angular [online], Vegibit.com, dostupno na: https://vegibit.com/angular-data-binding/ [pristupljeno 2. rujna 2020.]

[11] Structural directives: What are structural directives? [online], Angular, dostupno na: https://angular.io/guide/structural-directives [pristupljeno 2. rujna 2020.]

[12] Built-in directives [online], Angular, dostupno na: https://angular.io/guide/built-in-directives [pristupljeno 2. rujna 2020.]

[13] Introduction to services and dependency injection [online], Angular, dostupno na: https://angular.io/guide/architecture-services [pristupljeno 2. rujna 2020.]

[14] In-app navigation: routing to views [online], Angular, dostupno na: https://angular.io/guide/router [pristupljeno 2. rujna 2020.]

[15] Materijali s Loomen stranice kolegija „Osnove razvoja web i mobilnih aplikacija“: LV1 Uvod u HTML [online], Loomen, dostupno na: https://loomen.carnet.hr/mod/folder/view.php?id=349095 [pristupljeno 2. rujna 2020.]

[16] Materijali s Loomen stranice kolegija „Osnove razvoja web i mobilnih aplikacija“: LV2 Uvod u HTML [online], Loomen, dostupno na: https://loomen.carnet.hr/mod/folder/view.php?id=350899 [pristupljeno 2. rujna 2020.]

[17] SQL Server Management Studio [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/SQL_Server_Management_Studio [pristupljeno 2. rujna 2020.]

[18] Microsoft SQL Server [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Microsoft_SQL_Server [pristupljeno 2. rujna 2020.]

[19] Microsoft Visual Studio [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio [pristupljeno 2. rujna 2020.]

[20] What is Web API? [online], TutorialsTeacher, dostupno na: https://www.tutorialsteacher.com/webapi/what-is-web-api [pristupljeno 2. rujna 2020.]

[21] ASP.NET [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/ASP.NET [pristupljeno 2. rujna 2020.]

[22] Visual Studio Code, Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Visual_Studio_Code [pristupljeno 2. rujna 2020.]

[23] Enable Cross-Origin Requests (CORS) in ASP.NET Core [online], Microsoft, dostupno na: https://docs.microsoft.com/en-us/aspnet/core/security/cors?view=aspnetcore-3.1 [pristupljeno 2. rujna 2020.]

[24] Materialize [online], Google, dostupno na: https://materializecss.com/ [pristupljeno 2. rujna 2020.]

SAŽETAK

Glavni zadatak ovog završnog rada je napraviti Angular web aplikaciju koja će korisniku omogućiti izbor između nekoliko razina težine matematičkog kviza te omogućiti korisniku odgovaranje na postavljena pitanja zatim ispisati njegov rezultat. Baza podataka, u kojoj su pohranjena pitanja, je kreirana pomoću programa Microsoft SQL Server Management Studio zajedno s četiri tablice. Kao poslužitelj se koristio ASP.NET Web API. Poslužitelj ASP.NET Web API je kreiran unutar programa Microsoft Visual Studio te se pomoću njega riješio problem dohvaćanja pitanja iz baze podataka preko kreiranih Web API metoda. Angular se koristio kao okvir pomoću kojega se izradila web aplikacija unutar uređivača Microsoft Visual Studio Code. Angular se sastoji od komponenata i predložaka, dok preko uslužne klase konzumira Web API metode. Od opisnih jezika korišteni su HTML za strukturu predložaka te CSS za stilsko uređenje aplikacije. Pokretanjem i testiranjem web aplikacije ustanovljeno je da web aplikacija potpuno funkcionalna i da radi bez ikakvih problema i poteškoća.

Ključne riječi: Angular, ASP.NET Web API, Microsoft, predlošci, web aplikacija

SUMMARY

ANGULAR WEB APPLICATION MATH QUIZ

The main task of this final paper is to create an Angular web application that will allow the user to choose between several levels of difficulty of the math quiz and allow the user to answer the questions then print his result. The database, in which the questions are stored, was created using Microsoft SQL Server Management Studio along with four tables. The ASP.NET Web API was used as the server. The ASP.NET Web API server was created within Microsoft Visual Studio to solve the problem of retrieving questions from database via created Web API methods. Angular was used as a framework to create a web application within the Microsoft Visual Studio Code editor. Angular consists of components and templates, while consuming Web API methods over the service class. Among the descriptive languages, HTML was used for the structure of the templates and CSS for the stylish editing of the application. By launching and testing web application, it was established that the web application is fully functional and works without any problems and difficulties.

Keywords: Angular, ASP.NET Web API, Microsoft, templates, web application

ŽIVOTOPIS

Max Lončar je rođen 25.02.1998. godine u Vinkovcima, Hrvatska. Živi u Ivankovu gdje je pohađao Osnovnu školu August Cesarec Ivankovo u razdoblju od 2005. do 2013. godine. Nakon završetka osnovne škole upisuje Tehničku školu Ruđera Boškovića Vinkovci, smjer Tehnička gimnazija. Tijekom srednjoškolskog školovanja, 2016. godine je sudjelovao na Međužupanijskom natjecanju učenika u obrazovnom sektoru Elektrotehnika i računalstvo u disciplini Osnove elektrotehnike i mjerenja u elektrotehnici. Nakon polaganja mature 2017. godine, upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, koji trenutno pohađa. Solidno se služi Microsoft alatima te poznaje razne programske i opisne jezike kao što su HTML, CSS, C, C++, C#. Što se tiče Android aplikacija, upoznat je s Android Studio-m te s radom baza podataka. Također posjeduje znanje engleskog jezika u smislu čitanja, pisanja i govora te vozačku dozvolu B kategorije.

Max Lončar

PRILOZI

Na CD-u priloženom uz završni rad nalaze se:

- Završni rad u .docx i .pdf formatu
- *Angular5* projekt
- *WebAPI* projekt u Microsoft Visual Studio-u
- Mapa *Images* u kojoj se nalaze slike pitanja
- Mapa *Slike* unutar koje se nalaze slike iz završnog rada
- Mapa *SQL skripta* u kojoj se nalaze skripte za ubacivanje pitanja unutar baze podataka