Web aplikacija za ocjenjivanje ispita s višeslojnom arhitekturom

Paradžik, Ivan

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:595982

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

WEB APLIKACIJA ZA OCJENJIVANJE ISPITA S VIŠESLOJNOM ARHITEKTUROM

Diplomski rad

Ivan Paradžik

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 16.09.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Ivan Paradžik
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1149R, 13.10.2020.
OIB studenta:	82559835055
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	,
Sumentor iz tvrtke:	Jelena Tufeković
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 1:	Prof.dr.sc. Goran Martinović
Član Povjerenstva 2:	Izv. prof. dr. sc. Ivan Aleksi
Naslov diplomskog rada:	Web aplikacija za ocjenjivanje ispita s višeslojnom arhitekturom
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U teorijskom dijelu diplomskog rada treba opisati probleme i izazove pri ocjenjivanju ispita učenika/studenata s osvrtom na postojeća slična rješenja i postupke ocjenjivanja usklađene s nastavnom metodologijom. Također, treba analizirate višeslojne arhitekture programskih rješenja za web, njima odgovarajuće predloške programskih arhitektura, metodologiju razvoja, te izbor programskih tehnologija, jezika i razvojne okoline. Nadalje, potrebno je definirati funkcionalne i nefunkcionalne zahtjeve na web sustav za ocjenjivanje ispita, predložiti model i arhitekturu web sustava na strani korisnika i
Prijedlog ocjene pismenog dijela ispita (diplomskog 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 od strane mentora:	16.09.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 22.09.2022.

Ime i prezime studenta:	Ivan Paradžik
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1149R, 13.10.2020.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za ocjenjivanje ispita s višeslojnom arhitekturom**

izrađen pod vodstvom mentora Prof.dr.sc. Goran Martinović

i sumentorice Jelena Tufeković ,

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 1

2. IZAZOVI U PROVEDBI ON-LINE ISPITA I STANJE U PODRUČJU 2

2.1. Izazovi u provedbi on-line ispita 2

2.2. Stanje u području on-line izvođenja nastave i provedbe ispita 2

2.3. Postojeća programska rješenja za on-line rješavanje i ocjenjivanje ispita 5

2.4. Idejno rješenje web aplikacije za provedbu on-line ispita 5

3. ARHITEKTURA, MODEL I PRISTUPI U IZRADI PREDLOŽENE WEB APLIKACIJE 7

3.1. Višeslojna programska arhitektura 7

3.1.1. Zatvorenost i otvorenost sloja 9

3.2. Zahtjevi na sustav 11

3.2.1. Funkcionalni zahtjevi na sustav 11

3.2.2. Nefunkcionalni zahtjevi na sustav 12

3.3. Model web aplikacije 12

3.3.1. Arhitektura web aplikacije 12

3.3.2. Dijagram korištenja web aplikacije 13

4. PROGRAMSKO RJEŠENJE WEB APLIKACIJE 15

4.1. Korištene tehnologije 15

4.1.1. ASP.NET Core 15

4.1.2. Entity Framework Core 16

4.1.3. JavaScript 16

4.1.4. ASP.NET Core Identity 20

4.1.5. React 20

4.1.6. Redux state managment 21

4.1.7. PostgreSQL 22

4.2. Programsko rješenje na strani poslužitelja 23

4.2.1. Baza podataka 23

4.2.2. Struktura projekta ASP.NET Core 24

4.2.3. Prikaz multilayer strukture nad entitetom Exam 25

4.2.4. Administracija korisnika 30

4.3. Programsko rješenje na strani klijenta 34

4.3.1. Struktura direktorija aplikacije 35

4.3.2. Komunikacija s API-jem 36
4.3.3. Prikaz korištenja React hook-ova 37
4.3.4. Primjer rada s formama na komponenti *Register* 38

5. KORIŠTENJE APLIKACIJE 40

5.1. Registracija i prijava korisnika 40

5.2. Stranica s predmetima 41

5.3. Uporaba aplikacije u ulozi nastavnika 42

5.3.1. Obavijesti 42
5.3.2. Upravljanje korisnicima 42
5.3.3. Postavke predmeta 43
5.3.4. Kreiranje i baza pitanja 43
5.3.5. Kriterij ocjenjivanja ispita 45
5.3.6. Kreiranje i baza ispita 46
5.3.7. Vrednovanje ispita 48

5.4. Uporaba aplikacije u ulozi učenika 49

5.5. Osvrt na rad aplikacije 51

6. ZAKLJUČAK 52

LITERATURA 53

SAŽETAK 55

ABSTRACT 56

ŽIVOTOPIS 57

PRILOZI 58

1. UVOD

Ispitivanje znanja u obliku on-line ispita posljednjih se godina populariziralo. On-line ispiti pružaju moderno rješenje za potrebe ispitivanja znanja učenika ili studenata. Onima koji održavaju provjere znanja je znatno olakšana organizacija i provedba ispita. Za razliku od tradicionalnih ispita, nakon provedbe ispita, rezultati ispita su odmah dostupni. Zbog naglog povećanja učestalosti provedbe on-line ispita u svim obrazovnim ustanovama, od škole pa do fakulteta, prednosti i nedostatci provedbe on-line ispita aktivna su tema znanstvenih istraživanja. Jedno od najaktivnijih pitanja kod provedbe on-line ispita je pitanje sigurnosti od varanja na ispitima. Školske ustanove koriste mnoga rješenja u cilju ostvarivanja sigurnosti provedbe ispita i zaštite od varanja. Tema diplomskoga rada ne obuhvaća temu sigurnost i zaštita od varanja prilikom provedbe ispita.

Cilj ovoga rada bio je napraviti funkcionalnu web stranicu koja koristi višeslojnu arhitekturu, a glavna funkcija joj je omogućiti provedbu on-line ispita. Web stranica omogućuje registraciju korisnika. Pri registraciji korisnika odabrat će se uloga, odnosno je li korisnik učenik ili nastavnik. Nakon što se registrirani korisnici prijave, funkcije i korisničko sučelje im je definirano njihovom ulogom. Svi korisnici mogu administrirati postavkama svoga računa. Osnovne funkcije korisnika koji je nastavnik su: kreiranja predmeta i ispita te prihvaćanje učenika na predmet. Osnova funkcija korisnika koji je učenik je odabir predmeta te rješavanje ispita unutar odabranih predmeta.

U drugome poglavlju opisano je korištenje on-line oblika provođenja ispita te prikazana postojeća rješenja provedbe on-line ispita koja se koriste u školstvu. U trećem poglavlju opisana je višeslojna arhitektura te je prikazan model izvedbe programskoga rješenja. Četvro poglavlje sadrži pregled korištenih tehnologija u izradi web aplikacije i prikazuje način implementacije dijelova aplikacije. Zadnje poglavlje objašnjava korištenje web aplikacije uz prikaze pripadnih sučelja.

2. IZAZOVI U PROVEDBI ON-LINE ISPITA I STANJE U PODRUČJU

2.1. Izazovi u provedbi on-line ispita

Iako su on-line ispiti u praktičnom korištenju već duže vrijeme u različitim ispitima iz informatičke pismenosti, tijekom pandemije COVID-19 pojavila se njihova najveća potreba. Opće je poznato kako je navedena pandemija imala učinak na svaku sferu života, pa tako i na obrazovni sustav, koji se našao pred izazovom kako uspostaviti, zadovoljiti i osigurati kvalitetu on-line ispita poput tradicionalnih provjera znanja.

U ruralnim područjima pojavili su se izazovi financijskog stanja obitelji koji obuhvaćaju potrebnu opremu za on-line nastavu te on-line ispite, a to su: osobno ili stolno računalo, struja te kvalitetna internetska veza.

Međutim, jedan od najvećih izazova ovakve provedbe ispita jest vjerodostojnost istog. Forma on-line ispita bez obaveze videonadzora dovodi do najvećeg izazova, mogućnosti prepisivanja i varanja na ispitima. Provedba ispita na takav način tijekom pandemije COVID-19 dovela je do masovne pojave učenika odlikaša koji s lakoćom upisuju željenu školu, ali se onda u istoj muče budući da se on-line nastava te on-line ispiti više ne održavaju. Također se provode i on-line usmeni ispiti u smislu prezentiranja projekata te se i ovdje naišlo na izazov. Kako učenik dijeli svoj zaslon, pored sebe može imati papir s natuknicama o čemu će u kojem trenutku prezentacije govoriti. Na taj način učenici u budućnosti imaju veći rizik od straha javnog nastupa.

Sami izazovi koji su stavljeni ispred programa za održavanje on-line ispita su: dostupnost, trajanje i ispravljanje ispita. Ispit mora biti dostupan svim učenicima u isto vrijeme. Kako bi se smanjio rizik za prepisivanjem, potrebno je omogućiti nasumičan redoslijed pitanja u ispitu za svakog učenika. Također je i potrebno imati široku bazu s pitanjima koji ulaze u provjeru znanja kako bi se ista pitanja ponavljala u što manjem broju za sve učenike. Kako bi program mogao vjerodostojno ispraviti ispite potrebno je pod točne i moguće odgovore staviti svaku vrstu odgovora. Uz sve navedeno, nastavnik mora imati opciju i ručnog pregledavanja provedenog ispita kako bi mogao reagirati ukoliko je to potrebno.

2.2. Stanje u području on-line izvođenja nastave i provedbe ispita

Organizacija on-line izvođenja nastave tijekom pandemije COVID-19 nije predstavljala problem budući da su već u svijetu dobro korišteni programi za internet komunikaciju poput Skype-a, Microsoft Teams-a, Zoom-a, Webex-a, Slack-a, itd. Živeći u svijetu u kojem tehnologija brzo

napreduje, i nastavnici i učenici su s lakoćom mogli rukovati navedenim programima. Za prijenos materijala potrebnih za pojedini predmet, nastavnici iz Republike Hrvatske ponajviše su koristili CARNET-ov Loomen i Merlin, čiju uslugu pruža Centar za e-učenje Srce. Loomen kao sustav za on-line učenje daje mogućnost provedbe ispita. Primjenom višeslojne arhitekture u ovom diplomskom radu dan je prikaz izrade web aplikacije za održavanje on-line ispita koja sadržava sve potrebne koncepte i funkcionalnosti za održavanje takve forme ispita.

Prema [1], provedeno je istraživanje tijekom širenja pandemije COVID-19 kojem je primaran cilj bio utvrditi stupanj prihvaćanja on-line ispita od strane studenata na Sveučilištu Ajman u Ujedinjenim Arapskim Emiratima. Prilikom istraživanja korišten je deskriptivan pristup, a upitnik se sastojao od 27 stavki koji je bio podijeljen između 1986 studenata preddiplomskih studija. Rezultati navedenog istraživanja pokazali su umjeren stupanj prihvaćanja provedbe on-line ispita tijekom pandemije COVID-19, pritom su studentice takvu vrstu ispita smatrale prihvatljivijima od studenata. Osim spola, na rezultate su utjecali i disciplina i akademska godina te je zaključeno kako su studenti svih godina Fakulteta za farmaciju i zdravstvo te općenito studenti na trećoj godini preddiplomskih studija pokazali najviše razine prihvaćanja on-line ispita. Sveučilište Ajman u Ujedinjenim Arapskim Emiratima usvojilo je Moodle kao jedan od programa koji omogućava učenje na daljinu tijekom pandemije COVID-19. Budući da je takva vrsta nastave i odvijanja ispita bila nužna, Sveučilište je održalo obuku i radionice za nastavno osoblje kako bi im pružilo osnovne vještine i odgovarajuće znanje o osmišljavanju on-line nastave te pripreme on-line ispita. Zaključeno je da on-line ispiti odlikuju mnogim prednostima: brže povratne informacije i ocjene, ušteda vremena, ekološki su prihvatljivi, lako ih je identificirati i pristupiti neodgovorenim pitanjima, sustav on-line ispita koji je jasan i jednostavan omogućuje učenicima da polažu ispit bilo gdje i bilo kada. Rezultati ove studije mogli bi potaknuti i olakšati prijelaz na on-line učenje i on-line ispite kao proces ocjenjivanja u obrazovnom sustavu općenito, tijekom i nakon pandemije COVID-19 [1].

Prema [2], provedeno je istraživanje kojim su određene prednosti i nedostatci on-line nastave tijekom pandemije COVID-19 na jednom Sveučilištu u Indoneziji. Pandemija COVID-19 je pred predavače postavila brojne izazove u provođenju on-line procesa učenja. To znači da predavači moraju biti u stanju osigurati odgovarajuću tehnologiju aplikacije kako bi mogli izgraditi interes učenika za učenje njihovog predmeta. Podaci za navedeno istraživanje prikupljeni su promatranjem, dokumentiranjem i intervjuiranjem, a sudionici istraživanja bili su i predavači i studenti. Rezultati studije pokazali su da on-line nastava putem video prikaza ima mnoge prednosti

u on-line učenju te da se na taj način može probuditi interes učenika za učenje. Učenici su se osjećali kao da mogu učiti gradivo predmeta bilo kada kod kuće, što je jedna od velikih prednosti budući da nisu morali ići do sveučilišnih knjižnica nakon predavanja. S druge strane, pronađeni nedostatci takve nastave su ograničenja internetske mreže te nije bilo kreativnosti predavača u osmišljavanju on-line video predavanja zbog cijena programa koji predavanja mogu učiniti zanimljivijima nego što bi to bilo na tradicionalnim predavanjima koja se odvijaju uživo. Ovo istraživanje je još uvijek ograničeno na samo jedno sveučilište, stoga se rezultati ove studije mogu ponoviti naknadnim istraživanjima analizirajući druge aspekte on-line nastave te drugih sveučilišta. Uz to, rezultat ove studije također daje prijedlog da sveučilišta trebaju poboljšati digitalnu i tehnološku kompetenciju svakog predavača [2].

Istraživanje [3] bavi se tematikom budućnosti on-line visokog obrazovanja u post COVID-19 eri. Proučena su istraživanja koja su provela sveučilišta s istoka i zapada tijekom pandemije COVID-19. Pandemija je zaslužna za neočekivani rast on-line učenja te je diljem svijeta došlo do ubrzane digitalne transformacije svakog sveučilišta. Pandemija je promijenila načine na koje su studenti dobivali podršku, budući da je potpuno on-line studiranje postalo jedini izlaz i jedina opcija za većinu predmeta tijekom razdoblja karantene. Sukladno tome, sveučilišta su bila prisiljena uložiti u nove tehnologije i ažurirati postojeću IT infrastrukturu kako bi uspjela podržati takvu naglu promjenu. Studenti, ali i akademsko osoblje, bili su prisiljeni učiti nove tehnologije i brzo se prilagoditi novim postupcima učenja te polaganja ispita. Sveučilišta s istoka, poput onih u kontinentalnoj Kini i Hong Kongu, tvrde da su studenti bili motivirani za on-line učenje te da su cijenili sve mogućnosti takvog učenja, ali da oni ipak preferiraju mješovito učenje budući da im podučavanje licem u lice omogućuje odgovarajući kontakt i društvene interakcije s predavačima, ali i s kolegama. Sveučilišta sa zapada, poput sveučilišta u Kanadi, daju ipak negativnije dojmove. Podaci se temelje na negativnim učincima, točnije na poteškoćama u učenju tijekom on-line nastave jer učenicima nedostaje motivacija i angažman oko predmeta. Nagli prelazak na on-line nastavu stvorio je mnoge nepredvidive tehničke i pedagoške poteškoće u tim područjima, ali i probleme s mentalnim zdravljem akademskog osoblja i studenata. Unatoč tome, on-line učenje nametnuto pandemijom pružilo je i neke pozitivne stvari poput mogućnosti fleksibilnosti za međunarodnu suradnju studenata između sveučilišta diljem svijeta [3].

Prema [4], objavljen je rad koji pokušava evaluirati on-line ispite na Sveučilištu Dr. Babasaheb Ambedkar Marathwada u Indiji na Odsjeku za engleski jezik između studenata. Cilj ovog rada bio je otkriti valjanost on-line ispita koji se provode na tom Odsjeku. Istražitelji su slijedili kriterij

valjanosti koji se obično koristi u obrazovnim i psihološkim mjerenjima te su proveli kvalitativnu studiju. Prikupljeni su podaci korištenjem zatvorenog upitnika koji je ispunilo 57 studenata koji polažu magisterij na Odsjeku za engleski jezik. Korišteni su Google obrasci i distribuirani su on-line. Rezultati ove studije pokazali su da provedeni on-line ispit ima visoku valjanost u pogledu vremena ispita, uputa za ispit te pokrivenosti nastavnog plana i programa. Rezultati također pokazuju i da provedeni on-line ispit ima umjerenu valjanost u pogledu jednostavnosti pitanja. Međutim, višestruke prijave i odjave, tehnički problemi i nesavjesnost sudionika smanjili su valjanost on-line ispita. Neetičko ponašanje i nesavjesno postupanje uzrokovano je ograničenim nadzorom prilikom odvijanja on-line ispita. Nedostatak on-line ispita je nesavjesnost pojedinih ispitanika što je dovelo do pojave globalnog problema, stoga se preporučivanju daljnja istraživanja kako bi se usporedila valjanost on-line ispita s tradicionalnim ispitima [4].

2.3. Postojeća programska rješenja za on-line rješavanje i ocjenjivanje ispita

CARNET Loomen vrsta je on-line platforme koja omogućava učenje na daljinu. Pritom se misli na mogućnost pohađanja i otvaranja tečajeva, provjere dobivenih znanja, predaje i kontrole zadaća, ali i evidentiranja prisutnosti i komunikacije. Navedena se platforma bazira na Moodle-u. To je najkorišteniji sustav za on-line učenje, program je otvorenog koda za kojeg je licenca besplatna. Njegove vrline su širok izbor alata, dodataka i modula potrebnih prilikom izrade elektroničnih sadržaja za nastavu te održavanje nastave na daljinu [5].

Merlin, sustav za e-učenje, također je temeljen na sustavu otvorenog koda Moodle. Razvijen je od strane Centra za e-učenje Srce te je prilagođen potrebama korisnika što ga čini najmodernijim sustavom za e-učenje. Nastavnici, studenti i ustanove u sustavu visokog obrazovanja imaju mogućnost izvođenja kolegija pojedine ustanove koristeći tehnologiju e-učenja. Merlin-ovo virtualno okruženje za e-učenje čini sustav za e-učenje Merlin, sustav za webinare i e-portfolio sustav [6].

2.4. Idejno rješenje web aplikacije za provedbu on-line ispita

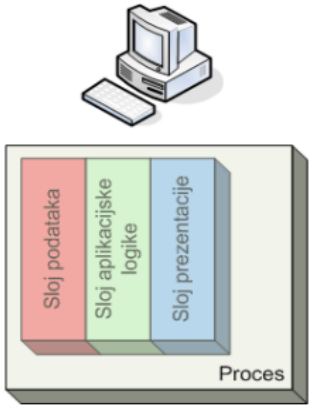
Idejno rješenje web aplikacije definira programsku podršku kojoj je zadaća evaluacija znanja učenika. U izvedbi rješenja naglasak je na korištenju višeslojne arhitekture. Izvedba zasnovana na višeslojnoj arhitekturi omogućuje lakše održavanje i proširivost sustava. Cilj je zadovoljiti sve funkcionalne i nefunkcionalne zahtjeve na sustav. Nakon izvedbe rješenja aplikacije, potrebno je prikazati korištene tehnologije u izradi te objasniti ključne dijelove implementacije opisane u

kontekstu korištenja aplikacije. Nakon pregleda dijelova implementacije, prikazati ostvareno rješenje u načinu korištenja od strane krajnjeg korisnika.

3. ARHITEKTURA, MODEL I PRISTUPI U IZRADI PREDLOŽENE WEB APLIKACIJE

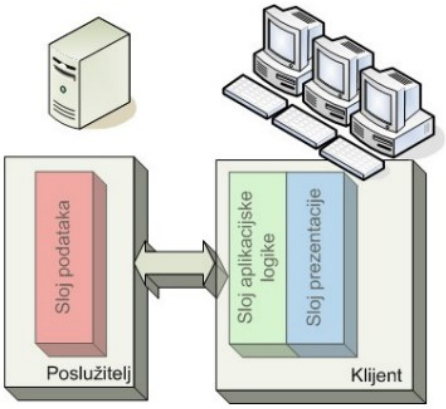
3.1. Višeslojna programska arhitektura

Kako bi se mogao razumjeti princip višeslojne programske arhitekture, prvo je potrebno definirati monolitnu i dvoslojnu arhitekturu. Monolitna arhitektura (slika 3.1) naziv je koji obuhvaća sve funkcionalne slojeve aplikacije unutar procesa koji se izvodi na jednom računalu [7].



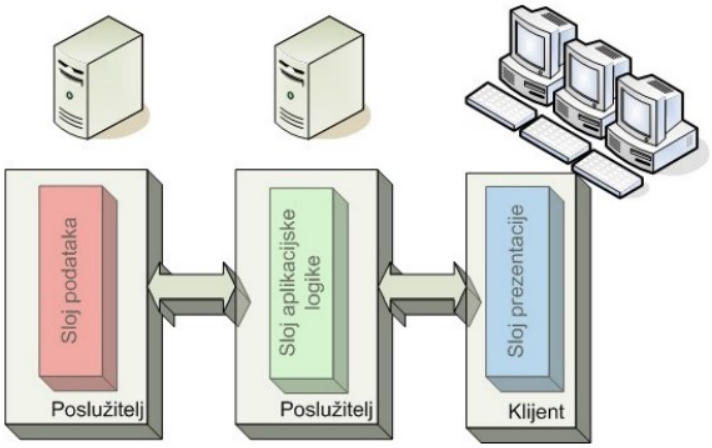
Slika 3.1. Shematski prikaz monolitne arhitekture [7].

S druge strane, dvoslojna arhitektura (slika 3.2) obuhvaća naziv za funkcionalne slojeve grupirane u dva zasebna sloja arhitekture, odnosno u dva zasebna procesa. Aplikacija klijenta mora sadržavati funkcionalne slojeve prezentacije i sloj aplikacijske logike, ukoliko ta logika postoji. Aplikacija poslužitelja mora sadržavati sloj podataka te može pružiti usluge jednoj ili više aplikacija klijenata u isto vrijeme [7].



Slika 3.2. Shematski prikaz dvoslojne arhitekture [7].

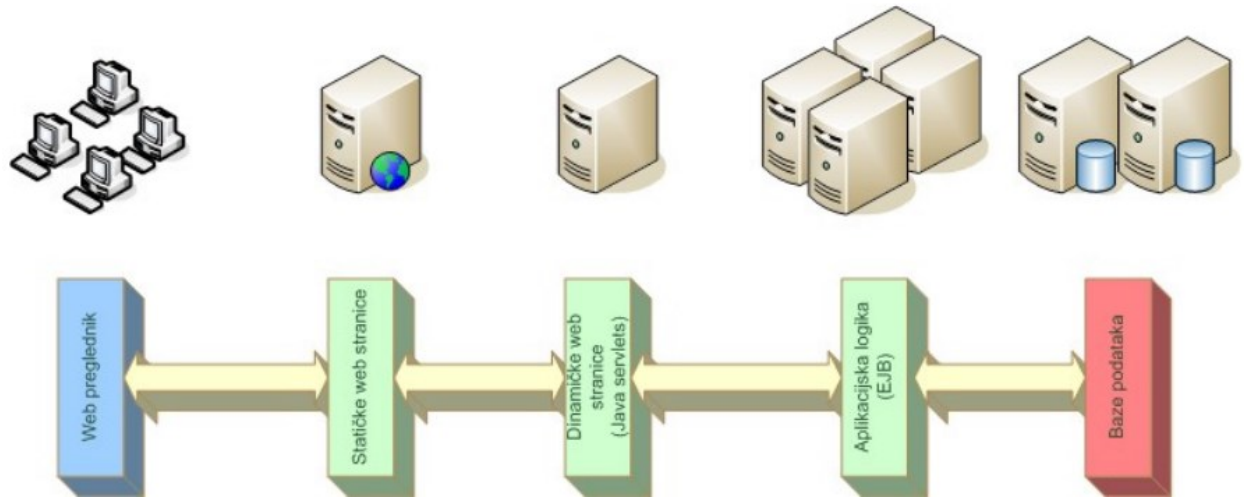
Troslojna programska arhitektura predstavlja jednu od najzastupljenijih arhitektura korištenih pri razvoju programske podrške. Kao što i sam naziv kaže, definira ju skup triju različitih slojeva od kojih svaki posjeduje svojstvenu ulogu u radu aplikacije. Slojevi međusobno ne ovise jedni o drugima što omogućava lakšu izmjenu na sloju kojeg je potrebno izmijeniti nego što bi bilo napraviti izmjenu na cijeloj arhitekturi. Slika 3.3 prikazuje shematski prikaz slojevite troslojne arhitekture iz koje se može zaključiti da su njezini glavni slojevi: prezentacijski sloj, aplikacijski sloj i podatkovni sloj. Prvi i najviši sloj u aplikaciji kojem je funkcija prezentacija sadržaja korisniku koristeći grafičko sučelje naziva se prezentacijski sloj. Za prezentaciju sadržaja, nužna je komunikacija navedenog sloja s ostalim slojevima aplikacije. Srednji sloj višeslojne programske arhitekture koji je odgovoran za logiku poslovanja aplikacije naziva se aplikacijski sloj. Poslovnu logiku čine sva pravila koja se koriste za implementaciju funkcionalnih zahtjeva aplikacije. Sastavnice ove razine mogu se izvoditi na jednom ili više poslužitelja. Zadnji, ali ujedno najznačajniji sloj ove vrste arhitekture je podatkovni sloj. Njegova je uloga pohranjivanje i pronalazak podataka aplikacije. Uobičajeno je da će se podaci aplikacije pohraniti na poslužitelju pripadajuće baze podataka, poslužitelju datoteka ili na bilo kojem drugom uređaju s istom logikom pristupanju podacima. Poslužitelj to omogućava pružanjem aplikacijskog programskog sučelja (engl. *application programming interface*, API). Spomenuti API će osigurati cjelovitu transparentnost akcija podataka koje se odrađuju na ovoj razini, ne imajući utjecaj na razinu aplikacije [8].



Slika 3.3. Shematski prikaz troslojne programske arhitekture [7].

Elementi višeslojnog modela su slojevi i konektori. Slojevi su razine apstrakcije kojima je zadaća riješiti neovisne zadatke, dok su konektori protokoli interakcija koji se odvijaju oko susjednih slojeva [7]. Međutim, važno je razumjeti da višeslojna aplikacija ne sadržava samo navedena tri

sloja. Odnosno, višeslojna aplikacija uvijek posjeduje prezentacijski i podatkovni sloj na koje se mogu dograditi i ostali slojevi što je i prikazano na slici 3.4 Karakteristike takve arhitekture su ravnomjerna raspodjela opterećenja između slojeva, ali je potrebna veća propusnost infrastrukture komunikacija između slojeva [7].



Slika 3.4. Shematski prikaz višeslojne programske arhitekture s više od tri sloja [7].

Slojevita arhitekruta ima brojne prednosti, ali i nedostatke. Prednosti višeslojne programske arhitekture su:

- svaki sloj ima točno definiranu ulogu,
- slojevi su „slabo“ povezani konektorima,
- slojevi su neovisni o implementaciji što ih čini lako zamjenjivima i
- protokoli interakcije slojeva moraju biti strogo poštivani.

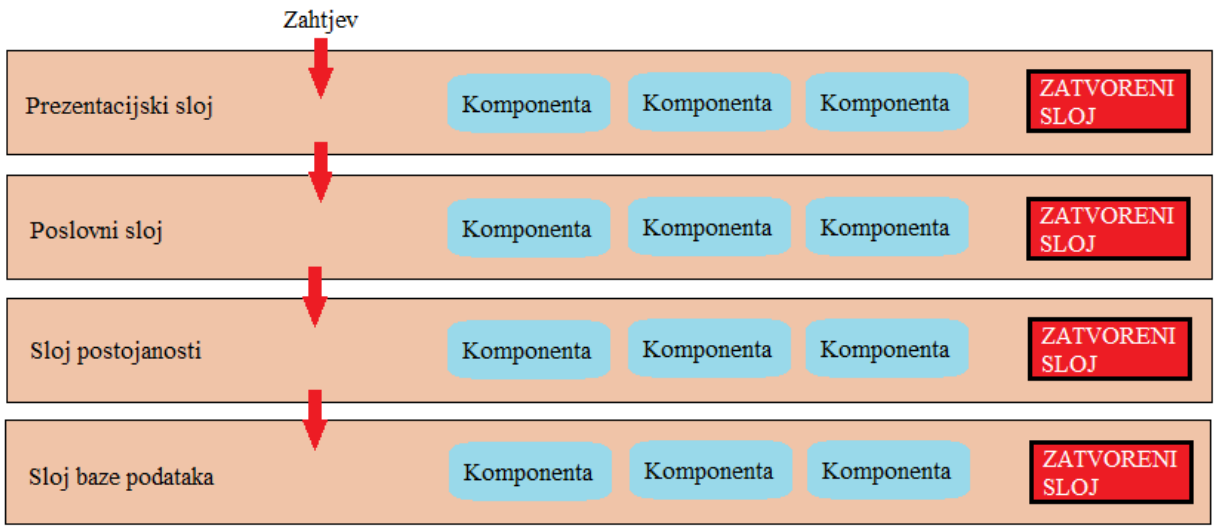
S druge strane, nedostatci višeslojne programske arhitekture su:

- smanjene performanse sustava,
- visoka cijena promjene protokola interakcija i
- ponekad teška identifikacija jasno odijeljenih slojeva [7].

3.1.1. Zatvorenost i otvorenost sloja

Vrlo važan koncept višeslojne arhitekture jest zatvorenost, odnosno otvorenost, njezinih slojeva. Ukoliko su svi slojevi zatvoreni, to znači da će prilikom nekog zahtjeva taj zahtjev proći kroz svaki sloj arhitekture. Na primjeru slike 3.5 vidljiv je zahtjev koji počinje na prezentacijskome sloju. Taj

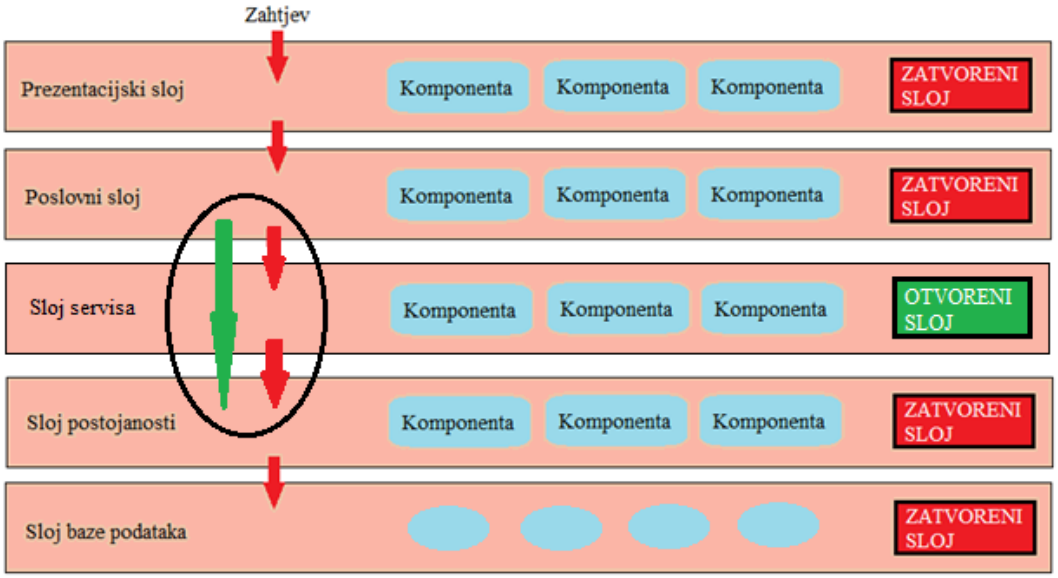
zahtjev prvo prolazi kroz poslovni sloj, a zatim kroz sloj postojanosti prije nego što konačno dostigne do sloja baze podataka [9].



Slika 3.5. Primjer višeslojne arhitekture sa zatvorenim slojevima [9].

Postavlja se pitanje zašto prezentacijski sloj nema izravan pristup sloju postojanosti ili sloju baze podataka, a odgovor se pronalazi u ključnom konceptu poznatom pod nazivom sloj izolacije. Koncept izolacije definiran je tako da promjene nastale u jednome sloju ne utječu na komponente drugih slojeva. Drugim riječima, promjena sloja ili neke njegove komponente je izolirana, a može obuhvaćati i male promjene na povezanome sloju. Primjer izmjena povezanog sloja je vidljiv na primjeru sloja pristupa podacima i sloju baze podataka. Ukoliko se pojave izmjene na sloju baze podataka, očekivane su i male prilagodbe na sloju pristupa podacima. Koncept slojeva izolacije označava i neovisnost o preostalim slojevima što dovodi do nedovoljnog znanja o unutarnjem radu drugih slojeva u arhitekturi [9].

Iako zatvoreni slojevi olakšavaju slojeve izolacije te kao takvi pomažu u izoliranju promjene unutar arhitekture, postoje i slučajevi kada se javlja potreba da neki slojevi arhitekture budu otvoreni. Slika 3.6 prikazuje primjer višeslojne arhitekture s otvorenim slojem te se može zaključiti kako sloj servisa omogućava zahtjevu da ga zaobiđe i direktno pristupi sljedećem sloju [9].



Slika 3.6. Primjer višeslojne arhitekture s otvorenim slojem [9].

Korištenje koncepata otvorenosti i zatvorenosti slojeva omogućava definiranje odnosa slojeva arhitekture i tijeka zahtjeva kroz te slojeve. Uz to dizajneri i programeri dobivaju sve potrebne informacije nužne za razumijevanje različitih ograničenja pristupa slojevima unutar arhitekture. Međutim, neuspjeh prilikom dokumentiranja ili pravilne komunikacije koji je sloj arhitekture otvoren ili zatvoren dovodi do usko povezanih i krhkih arhitektura koje se teško testiraju, održavaju i implementiraju [9].

3.2. Zahtjevi na sustav

U procesu planiranja izrade aplikacije definiraju se funkcionalni i nefunkcionalni zahtjevi koje konačan proizvod mora ispunjavati. Takav pristup štiti proces izrade aplikacije od dodavanja nepotrebnih funkcionalnosti, štedi vrijeme i olakšava organizaciju posla tijekom izrade [10].

3.2.1. Funkcionalni zahtjevi na sustav

Funkcionalni zahtjevi definiraju funkcionalnosti koje će aplikacija omogućavati svojim korisnicima. U nastavku su nabrojani svi funkcionalni zahtjevi:

- mogućnost registracije u ulozi „Nastavnik“ ili „Učenik“ i prijava,
- potvrda korisničkog računa pomoću adrese e-pošte,
- korištenje tokena prilikom autorizacije korisnika,

- omogućavanje korištenja refresh tokena ukoliko je korisnik prethodno ulogiran,
- odjava korisnika ukoliko je sesija istekla,
- dohvaćanje koda za ponovno postavljanje zaboravljene lozinke,
- promjena lozinke ulogiranog korisnika,
- razlike u korisničkome sučelju ovisne o ulozi,
- kreiranje i administriranje predmeta, obavijesti predmeta, pitanja, kriterija ocjenjivanja, ispita, korisnika pojedinih predmeta,
- obavještavanje korisnika o kreiranim ispitima putem e-pošte,
- automatsko i ručno ocjenjivanje riješenih ispita i
- prikaz uspješnosti riješenosti ispita.

3.2.2. Nefunkcionalni zahtjevi na sustav

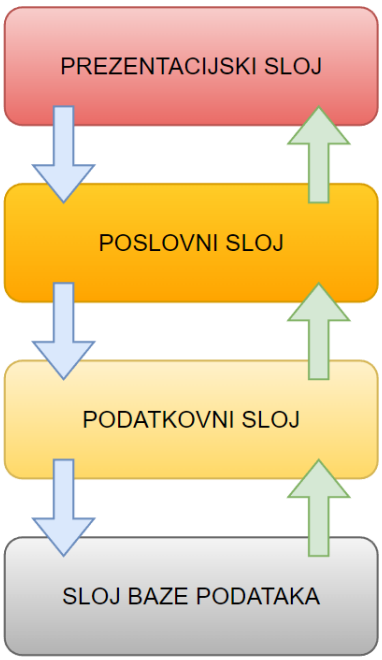
Nefunkcionalni zahtjevi (engl. *Non-Functional Requirements*, NFRs) ne predstavljaju stvari koje će program raditi, nego način na koji će taj program to obavljati uz naglasak na kvalitetu samog programa. Neki od NFRs su: sigurnost podataka, prilagodljivost sustava, modularnost, održavanje itd. Zadovoljavanje nefunkcionalnih zahtjeva od ključne je važnosti u izradi programskog rješenja. Ukoliko NFRs nije zadovoljen u potrebnoj mjeri, konačan proizvod neće opravdati očekivanja. Neki od problema koji mogu nastati su nekonzistentnost programa i, u konačnici, njegova loša kvaliteta [10].

3.3. Model web aplikacije

3.3.1. Arhitektura web aplikacije

Arhitektura web aplikacije sadrži četiri sloja koji su prikazani na slici 3.7. Svaki sloj je zatvorenog tipa. Prezentacijski dio aplikacije se odnosi na dio aplikacije vidljiv korisniku te omogućava interakciju korisnika sa sustavom. Načinjen je u JavaScript-u, korištenjem biblioteke React. Budući da je arhitektura kroz sve slojeve zatvorenog tipa, interakcija sa sustavom jedino je moguća pomoću prezentacijskog sloja. Poslovni sloj predstavlja API. On je izrađen pomoću razvojnog okruženja .NET Core 6. Neke od uloga poslovnoga sloja u aplikaciji su: primanje podataka od strane korisnika, interakcija s bazom pomoću podatkovnog sloja, obrada podataka, itd. Podatkovni sloj izrađen je pomoću alata Entity Framework Core. On služi kao spona između poslovnog sloja

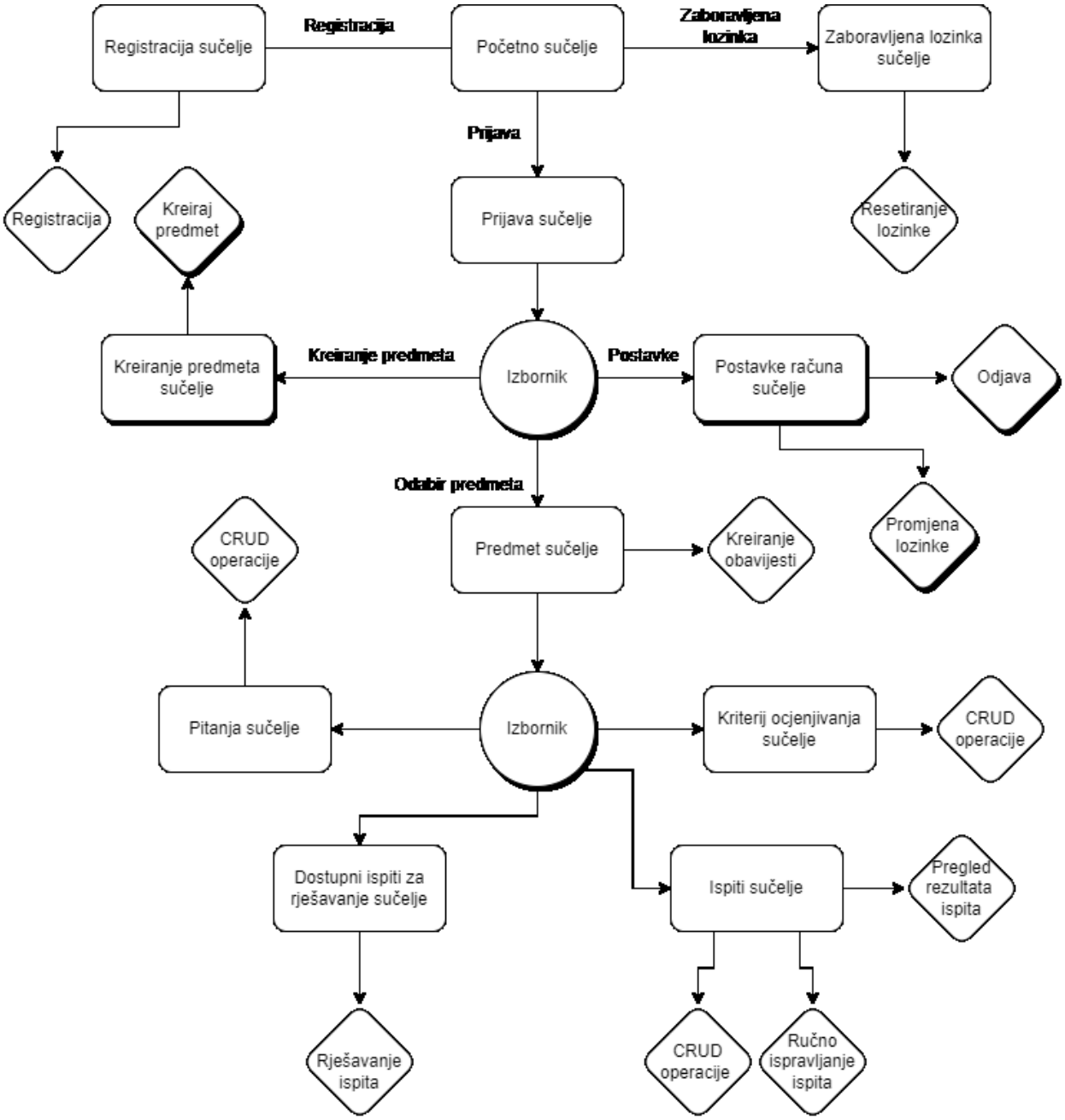
i sloja baze podataka te olakšava njihovu interakciju. Za izradu sloja baze podataka korištena je PostgreSQL baza podataka.



Slika 3.7. Prikaz arhitekture web aplikacije.

3.3.2. Dijagram korištenja web aplikacije

Način na koji aplikacija omogućuje korisnicima okruženje zasnovano na interakciji nastavnika i učenika prikazan je na slici 3.8 pomoću dijagrama korištenja. Svim korisnicima je omogućeno korisničko početno sučelje, pomoću kojeg dobivaju pristup funkcijama aplikacije. Neke od prikazanih funkcionalnosti nisu dostupne svim korisnicima, nego ovise o ulozi korisnika u sustavu.



Slika 3.8. Dijagram korištenja aplikacije.

4. PROGRAMSKO RJEŠENJE WEB APLIKACIJE

4.1. Korištene tehnologije

Za dio aplikacije na strani korisnika (engl. *front-end*) koristi se programski jezik JavaScript i biblioteka React. U izradi dijela aplikacije na strani poslužitelja (engl. *back-end*) koristi se C# programski jezik, odnosno razvojno okruženje ASP.NET Core. Za bazu podataka korištena je PostgresSQL baza podataka.

4.1.1. ASP.NET Core

ASP.NET Core (engl. *Active Server Pages*) je višeplatformsko (engl. *cross platform*) razvojno okruženje otvorenog koda (engl. *open source*). Svojstvo višeplatfomnosti omogućuje da se aplikacije koje koriste ASP.NET Core mogu razvijati i pokretati na raznim platformama kao što su Windows, Mac i Linux, dok svojstvo otvorenog koda omogućava da je izvorni kod dostupan svakome na uvid, korištenje te izmjene poštujući Apache licence, odnosno ALv2. Servisi koji se koriste u aplikaciji definiraju se metodom *ConfigureServices*, a neki od najučestalijih servisa su ASP.NET MVC Core framework, Entity Framework Core i Identitiy. ASP.NET Core omogućava uporabu NuGet paketa potrebnih za aplikaciju. .NET Core brzo napreduje zato što se može implementirati uz aplikaciju, ali se može i često mijenjati, a napravljene promjene nemaju utjecaja na druge .NET Core aplikacije na istom računalu. Iako ih Microsoft pokušava poboljšati, većina Microsoft-ovih poboljšanja .NET Core-a i modernog .NET-a ne mogu se lako dodati u .NET Framework [11].

Za izradu ovog diplomskog rada korišten je ASP.NET Core 6 koji je objavljen u studenom 2021. godine s fokusom na poboljšanje produktivnosti. Neke novine su smanjivanje koda za implementiranje osnovnih web stranica i usluga, .NET Hot Reload i nove opcije hostinga za Blazor kao što su hibridne aplikacije koje koriste .NET MAUI [11].

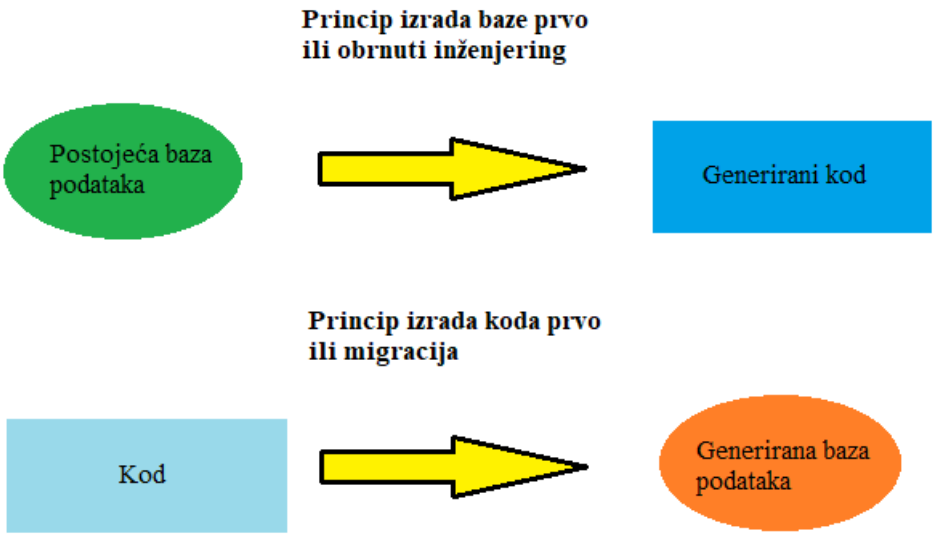
.NET 6 uključuje poboljšanja koja su širokog spektra:

- olakšava novim programerima početak rada s .NET-om,
- proširuje podršku prilikom izrade nativnih klijentskih aplikacija za više platformi,
- osigurava da .NET ima sve potrebno za izvorno pokretanje u oblaku,
- osigurava da poduzeća koja se oslanjaju na LTS imaju nesmetan put nadogradnje,
- jača .NET ekosustav te
- čini razvoj s .NET-om bržim i produktivnijim [12].

4.1.2. Entity Framework Core

Entity Framework Core objavljen je 2016. godine od strane tvrtke Microsoft, ali je to alat koji se može koristiti na više platformama jer ga podržavaju i Windows, Linux i MacOS. U samom imenu riječ „Core“ označava dio .NET inicijative kojoj je cilj bio prilagodba razvoja ASP.NET aplikacija uz razvijanje na više različitih platforma. Nadalje, Entity Framework Core je razvojno okruženje koji olakšava interakciju aplikacije s bazom podataka. Definira se kao alat za objektno relacijsko mapiranje (engl. *Object-Relational Mapper*, ORM). EF Core kao takav povezuje svijet objekata i svijet programskog koda, a glavna funkcija mu je ubrzanje i olakšavanje razvoja programa [13].

EF Core omogućuje dva pristupa rada (slika 4.1): princip izrada baze prvo (engl. *database first*) i princip izrada koda prvo (engl. *code first*). *Database first* pristup znači da već postoji baza i da će se modeli u programskome kodu praviti prema postojećim entitetima u bazi. Ovakav pristup koristi se kada je baza podataka definirana i preporučljiv je u manjim projektima. *Code first* pristup je fleksibilniji način rada s EF Core-om. On omogućava programeru razvoj orijentiran na principe objektno-orijentiranoga programiranja (OOP) i da se na taj način olakšava razvoj. Nakon što je aplikacijski dio gotov, pomoću domenskih modela se generira baza podataka sa svim potrebnim relacijama koje su rezultat programskoga koda [13].



Slika 4.1. Shematski prikaz principa izrada baze prvo i izrade koda prvo u EF Core-u [14].

4.1.3. JavaScript

JavaScript programski je jezik koji je primjenu pronašao kao jezik za brze popravke. Međutim, u posljednjem se desetljeću pronašao i usko integrirao s webom. JavaScript je vrlo rasprostranjen

jezik i zbog toga je i brzo prihvaćen kao jezik opće namjene za rad u programiranju na strani klijenta i poslužitelja. On je popularan programski jezik te je jednostavan za korištenje kako za početnike, tako i za iskusne korisnike. Brendan Eich kreirao je JavaScript 1995. godine s interaktivnim ciljem stvaranja weba pomoću web preglednika NetScape Navigatora. Godine 1995. World Wide Web dobivao je na popularnosti što je omogućilo da NetScape Navigator bude jedan od najpopularnijih web preglednika toga vremena. Prezentacijski jezik za izradu web stranica (engl. *HyperText Markup Language*, HTML) se prvenstveno koristio za stvaranje statičnog sadržaja koji je bio prikazan na web stranicama. Upoznavanjem s JavaScript-om te su web-stranice postale također interaktivne što je pridodalo ukupnoj vrijednosti i iskustvu posljednjeg korisnika koji istražuje web. Do 2005. godine upotreba JavaScript-a od strane developera znatno se povećala. JQuery, Dojo i Prototype objavljeni su iste godine i dobro su se integrirali s JavaScript-om što je dovelo do dodatnog povećanja u usvajanju i korištenju JavaScript-a. U početku se JavaScript pojavio kao skriptni jezik, ali uz prethodno spomenuti razvoj i podršku, JavaScript došao je do potpune transformacije jezika. Razvio se u robustan, siguran, dinamičan jezik opće namjene koji je vrlo prikladan za *front-end*, *back-end* i izradu složenih programskih rješenja. Nove značajke, tehnologije, razvojno okruženje i dodaci se stalno integriraju u JavaScript [15]. Prema [16], 97,7 % web stranica koristi JavaScript kao programski jezik na strani klijenta.

Većina web-stranica koristi JavaScript kako bi web bio dinamičniji i prilagođen korisnicima. JavaScript je programski jezik visoke razine, dinamičan je te se temelji na prototipu s podrškom za objektno-orijentirano programiranje. Ključne značajke JavaScript-a koje su ga učinile toliko popularnim među programerima su:

- JavaScript je programski jezik neovisan o platformi što znači da se može pokrenuti na bilo kojoj platformi koja ga proizvodi, svestran je jezik koji je jednostavan za korištenje, a podržava ga većina preglednika koji imaju ugrađene tumače za JavaScript.
- Objektni model dokumenta (engl. *Document Object Model*, DOM) omogućava pristup HTML komponentama unutar web-stranice za manipulaciju.
- JavaScript nudi ugrađene metode za operacije koje trebaju pristup uslužnim programima za datum i vrijeme što omogućava korisniku da ima prilagođen sadržaj na svojem web-pregledniku.
- JavaScript je u početku pružala skriptiranje na strani klijenata, dok danas pruža mogućnost pokretanja aplikacija i na strani klijenta i na strani poslužitelja što je učinilo JavaScript još popularnijim jer programer može razviti potpunu aplikaciju koristeći samo JavaScript.

- JavaScript se koristi za provjeru valjanosti korisničkih ulaznih podataka na strani klijenta i izostavlja potrebu slanja na poslužitelj. Na taj način pruža veću kontrolu nad preglednikom nego web-poslužitelj dok se kod izvršava na strani klijenta. Također, omogućuje web-poslužitelju da provjeri vrstu preglednika, OS, veličinu zaslona i sl., a sve navedeno pomaže u stvaranju smislenije skripte s manjim opterećenjima na strani poslužitelja i većom upotrebom resursa na strani klijenta.
- Funkcije u JavaScript-u tretiraju se kao objekti koji mogu imati i svojstva i metode. Funkcije mogu biti proslijeđene kao argumenti drugim funkcijama, mogu se ugnijezditi, mogu biti izuzetci i slično. Navedeno čini JavaScript svestranim za korištenje funkcija, a samim time jezik postaje lakšim za rukovanje i prikladnijim za rješavanje problema u stvarnom svijetu.
- Streličaste (engl. *arrow*) funkcije uključene su u jezik za izradu jednostavne sintakse za uporabu i pisanje. To su lake funkcije koje se mogu koristiti prilikom pisanja anonimnih funkcija.
- *Template literal* omogućava developeru pohranu varijabilnih vrijednosti u tokove (engl. *string*) te se zbog toga programer može bolje usredotočiti na logiku i razvoj aplikacije, a manje na sintaksu.
- Rukovanje događajima značajka je JavaScript-a koja kaže da kada god se dogodi događaj, može se izvršiti odgovarajuća rutina za pojavu tog događaja. Navedenu značajku JavaScript intenzivno koristi za odgovor aktivnosti korisnika na web-stranici kao što su *onClick*, *onLoad* i *onSelect* ponašanje [15].

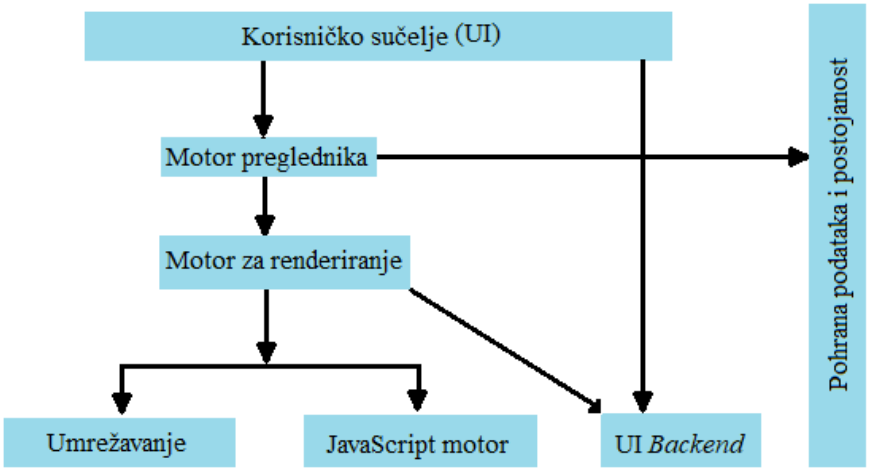
Kao i svaki drugi programski jezik, postoje pozitivne i negativne značajke uporabe JavaScript-a koje su prikazane u tablici 4.1 [15].

Tablica 4.1. Prednosti i nedostatci prilikom uporabe programskog jezika JavaScript [15].

PREDNOSTI	NEDOSTATCI
brzina izvršenja zahtjeva	nemogućnost čitanja/pisanja u datoteke
lakoća za učenje, jednostavnost	nemogućnost višenitnosti (engl. *multithreading*)
interoperabilnost	sigurnost na strani klijenta
izvršenje i provjera valjanosti na strani klijenta	nije ujednačena podrška za preglednike
bogata sučelja i proširena funkcionalnost	otklanjanje pogrešaka (engl. *debugging*)
	JavaScript pohranjuje brojeve kao 64-bitne, a po bitovima operatori rade s 32-bitnim brojevima

Web preglednik je programska aplikacija kojoj je uloga prikazati web sadržaj korisniku. To je potpuno funkcionalan program koji obavlja niz zadataka poput traženja, dohvaćanja, obrade i prikaza web sadržaja krajnjem korisniku. Sadrži grafičko korisničko sučelje kojim se tumači web sadržaj uključujući statični sadržaj na stranici umotan u HTML, CSS i JavaScript baziran interaktivan kod. Na klijentskoj strani sustava se funkcionalnosti web preglednika mogu proširiti pomoću raznih dodataka (engl. *plugin*). Danas su na tržištu dostupni razni web preglednici, a to su: Mozilla Firefox, Internet Explorer, Safari, Chrome, Netscape Navigator, itd.

Web-preglednik sastoji se od zbira komponenata koje zajedno obavljaju skup konkretnih zadataka. Shematski prikaz strukture web-preglednika prikazan je na slici 4.2. Prva komponenta naziva se korisničko sučelje (engl. *User Interface*, UI). To je prostor unutar kojeg korisnik komunicira s preglednikom i šalje zahtjev web poslužitelju. Motor preglednika (engl. *browser engine*) druga je komponenta kojoj je funkcija primiti unos i izvršiti primljeni upit iz korisničkog sučelja. Dobiveni podaci se šalju u mehanizam za prikaz za daljnju obradu što čini motor preglednika poveznicom korisničkog sučelja s mehanizmom za prikaz. Motor za renderiranje (engl. *rendering engine*) treća je komponenta koja tumači i izvodi kod za prikaz web-stranice korisniku. On tumači HTML, CSS kod koji je poslan od poslužitelja kao odgovor i prikazuje ga na korisničkom sučelju. Za izvođenje JavaScript koda koristi se posebna komponenta, JavaScript motor, koja šalje svoj odgovor na motor za renderiranje, a zatim ga prikazuje na korisničkom sučelju. Iduća komponenta je umrežavanje (engl. *networking*). Mrežna komponenta upravlja dohvaćanjem ujednačenog lokatora sadržaja (engl. *Uniform Resource Locator*, URL) preko internetskih protokola poput hipertekstualnog transfernog protokola (engl. *Hypertext Transfer Protocol*, HTTP) ili protokola za prijenos podataka (engl. *File Transfer Protocol*, FTP). Umrežavanje je uglavnom odgovorno za održavanje komunikacije i sigurnost koda, ali također može i implementirati podatke predmemorije kako bi se smanjilo zagušenje mreže. Pohrana podataka i postojanost (engl. *data persistance*) je posljednja komponenta web preglednika kojoj je funkcija održati postojanost preko mreže, preglednika i izvornog koda. Radi se o bazi podataka koja je prisutna u pohrani gdje je instaliran preglednik. Komponenta za pohranu također pohranjuje predmemoriju, korisničke podatke, oznake, postavke i kolačiće [15].



Slika 4.2. Struktura web preglednika [15].

4.1.4. ASP.NET Core Identity

ASP.NET Core Identity je sustav identiteta koji se isporučuje s ASP.NET Core-om. Poput i svega drugog u ekosustavu ASP.NET Core-a, to je skup NuGet paketa koji se mogu instalirati u bilo koji projekt, ali mogu biti i već uključeni u projekt ako se koristi zadani predložak. Funkcija ASP.NET Core Identity-ja jest briga o pohranjivanju računa korisnika, haširanje (engl. *hashing*) i pohranjivanje enkriptirane lozinke te upravljanje ulogama korisnika. Podržava prijavu e-poštom s pripadajućom lozinkom, provjeru autentičnosti s više faktora, prijavu putem društvenih mreža te povezivanje s drugim uslugama koje koriste protokole poput OAuth 2.0 te OpenID Connect. ASP.NET Core Identity pomaže pri dodavanju značajki sigurnosti i identiteta kao što su prijava i registracija na aplikaciju. Novi predlošci .NET-a unaprijed daju izgrađene prikaze i kontrolore koji omogućuju brzo pokretanje prijave ili registracije [17].

4.1.5. React

React je vrsta JavaScript *front-end* biblioteke otvorenog koda kojoj je funkcija olakšati razvijanje korisničkog sučelja kreiranjem komponenti koje su međusobno nezavisne, ali se mogu upotrijebiti neograničen broj puta. Te se komponente mogu definirati kao JavaScript klase ili JavaScript funkcije. Njihova glavna zadaća im je vraćanje HTML elemenata. Svaka React komponenta ima svoj životni vijek koji se po potrebi može prepisati te na taj način pružiti mnoge mogućnosti programeru. Uz to, React olakšava otklanjanje pogrešaka u kodu jer ih dijeli na komponente. React je odgovoran za dizajniranje korisničkog sučelja aplikacije [18].

React biblioteka pomaže programerima sa sljedećim značajkama:

- daje mogućnost ponovne upotrebe koda,
- jednostavna je za korištenje,
- kontrola je primarno područje u kojem se biblioteka i razvojno okruženje razlikuju što čini React u odnosu na Angular vrlo prilagodljivim, dakle korisnik ima kontrolu i može uključiti potrebne biblioteke u rad,
- React koristi virtualni DOM koji renderira čvorove kada je to potrebno kako bi se povećala učinkovitost,
- renderanje na strani poslužitelja za neke implementacije poput aplikacija usmjerenih na sadržaj,
- omogućava grupiranje kako bi se smanjilo opterećenje resursa korisnika,
- čist i jednostavan kod React-a čini vrijeme učitavanja web stranice gotovo zanemarivim,
- React funkcionira na način da rendera sadržaj u DOM i učinkovito ga kontrolira te
- React pruža sve potrebne alate za određivanje što treba prikazati na koji način i pod kojim uvjetima [19].

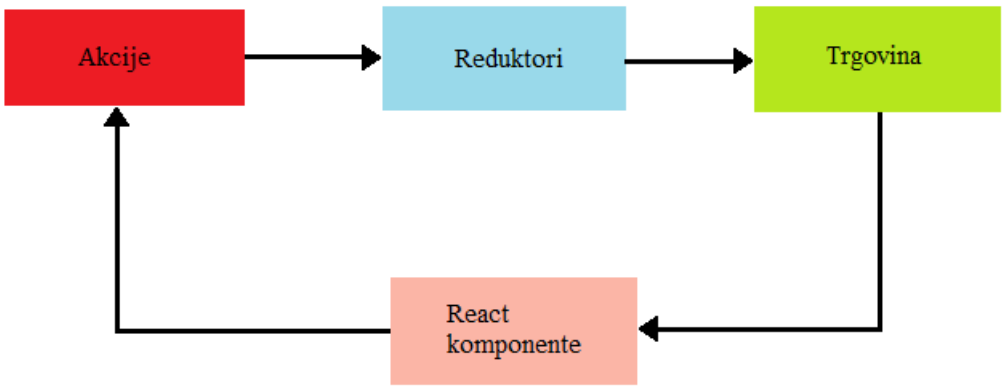
Međutim, React nema ugrađenu funkciju za provjeru obrazaca. Ne isporučuje HTTP klijent niti dolazi s usmjerivačem (engl. *router*), na primjer za renderanje različitih komponenata kao odgovor na promjene URL-a. React ima ugrađenu podršku za neka stanja, ali osim toga nema nikakvih dodatnih jedinstvenih značajki što ga čini „tanjim“ od Angulara [19].

Angular, Vue i React tri su razvojna okruženja koja su proučavana zasebno i u usporedbi jedni s drugima. Rezultat tog istraživanja pokazao je da je React najbolje razvojno okruženje te se preporuča kao jedan od najboljih izbora za razvoj *front-end* aplikacija budući da nema vidljivih slabosti, a ima mnogo prednosti među kojima su najznačajnije virtualni DOM, širok izbor alata i skalabilnost [20].

4.1.6. Redux state managment

Redux je biblioteka treće strane koja se koristi za upravljanje stanjem u *front-end* aplikacijama. Glavni koncept iza ove biblioteke jest da se cijelo stanje aplikacije pohranjuje na jednom središnjem mjestu. Svaka komponenta aplikacije može imati izravan pristup stanju aplikacije. Globalno stanje aplikacije naziva se *store* i služi samo za čitanje, a da bi se ovdje napravile promjene, potrebno je poslati odgovarajući zahtjev. Promjene u globalnom stanju odvijaju se kroz čiste funkcije što znači da te funkcije ne bi trebale imati nikakve nuspojave. Izgradnja arhitekture za složenu aplikaciju s Reduxom nije pretjerano težak zadatak jer se glavni koncepti već odvajaju

u različite odgovornosti. Akcije (engl. *actions*) u Reduxu su obični objekti koji imaju tip i nosivost koje su modificirane tako da se navedene akcije pretvaraju u asinkrone funkcije. Reduktori (engl. *reducers*) su čiste funkcije koje prihvaćaju trenutno stanje i poslanu radnju, a zatim to stanje obrađuju i vraćaju sljedeće stanje. Svi reduktori, kao i dodatni međuprogrami, kombinirani su u trgovinu (engl. *store*) koja se prosljeđuje React aplikaciji od strane dobavljača posebne React komponente (engl. *special component-provider*). Korištenje Reduxa uključuje stvaranje svih struktura koje implementiraju pristup prikazan na slici 4.3. [20].



Slika 4.3. Redux protok podataka [20].

Redux je započeo kao pratnja React-u, no ubrzo je počeo skupljati velik broj korisnika s drugim razvojnim okruženjem poput Angulara. U svojoj osnovi je Redux u potpunosti nevezan za razvojno okruženje te se može s lakoćom koristiti s bilo kojim JavaScript razvojnim okruženjem za obradu stanja i promjena. Povezivanje s različitim razvojnim okruženjima odvija se pomoću biblioteka trećih strana koje mogu pružiti skup praktičnih funkcija za svako razvojno okruženje kako bi se neprimjetno povezale s Reduxom. Redux je korišten među programerima zbog njegove jednostavnosti, odnosno toliko je jednostavan za korištenje da se većina toga može implementirati u već samo nekoliko linija koda [21].

4.1.7. PostgreSQL

PostgreSQL je objektno-relacijska baza podataka otvorenoga koda. U razvoju je više od 30 godina i besplatna je za korištenje. POSTGRES bio je pionir u mnogim objektno-relacijskim konceptima koji su dostupni u nekim komercijalnim bazama podataka. Tradicionalni sustavi upravljanja relacijskim bazama podataka podržavaju podatkovni model koji se sastoji od zbirke imenovanih relacija koje sadrže atribute otvorenog tipa. U trenutnim komercijalnim sustavima ti su atributi decimalni i cijeli brojevi, nizovi znakova, novac i datumi. Općenito se priznaje da je ovaj model

neadekvatan za buduće aplikacije za obradu podataka. Relacijski model uspješno je zamijenio prethodne modele zbog jednostavnosti. Međutim, navedena jednostavnost može otežati implementaciju nekih aplikacija. PostgreSQL nudi dodatne značajke ugradnjom dodatnih koncepata na način da korisnici mogu lako proširiti sustav, a te značajke su: nasljedstvo, vrste podataka i funkcije. Druge značajke koje pružaju dodatnu snagu i fleksibilnost su: ograničenja, okidači, pravila te transakcijski integritet. Zbog svih ovih značajki se PostgreSQL svrstava u kategoriju baza podataka koje se nazivaju objektno-relacijske baze podataka [22].

Tijekom godina PostgreSQL dobio je smanjenu reputaciju i korištenje među ostalim bazama podataka. Usprkos opsežnom popisu podrške i funkcionalnih značajki nad preostalim konkurencijama, PostgreSQL još uvijek nije vidljivo poznat [23].

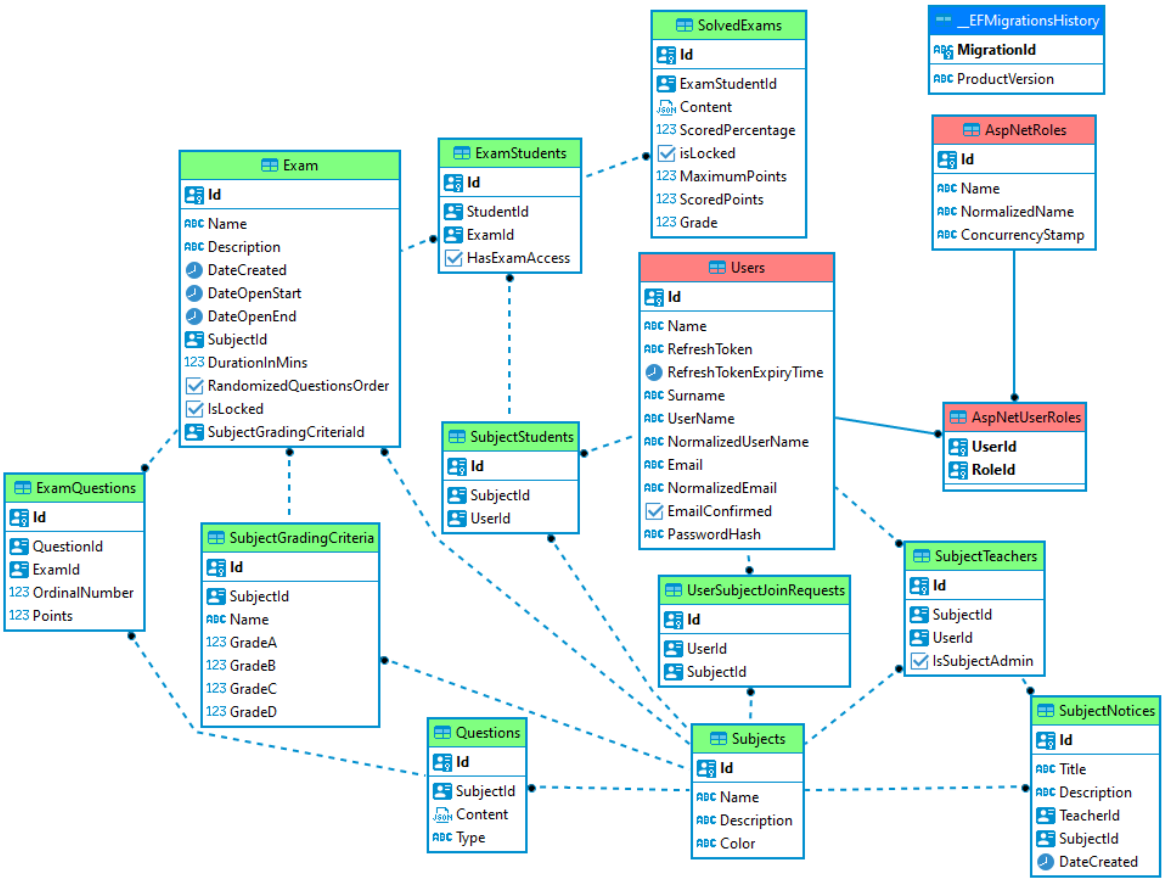
4.2. Programsko rješenje na strani poslužitelja

Poslužiteljska strana definira krajnje točke (engl. *endpoints*) kojima se pristupa pomoću klijentske strane. Svaka krajnja točka ima svoju zadaću koja se strogo definirana. Neke od funkcionalnosti koje poslužiteljska strana pruža su: interakcija s bazom podataka, obrada podataka, osiguravanje sigurnosti pristupa krajnjim točkama, itd.

4.2.1. Baza podataka

Nakon što su određeni funkcionalni zahtjevi programskoga rješenja, prvi korak bio je organizacija sloja baze podataka. Definirani su potrebi entiteti u bazi i odnosi među njima te sveobuhvatni model baze podataka. Implementacija baze napravljena je principom *code first* korištenjem EF Core NuGet paketa. Model baze podataka prikazan je na slici 4.4. Crvenom bojom su označene tablice kreirane pomoću .NET Core Identity te se brinu o podacima administracije korisnika na razini aplikacije. U *Users* tablici nalaze se podaci o svim korisničkim računima na razini aplikacije. *Roles* tablica sadrži uloge koje je moguće imati u aplikaciji, a to su *Student* i *Teacher*. Korisnicima je pridružena uloga u tablici *UserRoles*. Zelenom bojom označene su tablice koje se brinu o podacima logičkog dijela aplikacije. Tablica *Subjects* služi za spremanje kreiranih predmeta. *SubjectTeachers* i *SubjectStudents* predstavljaju korisnike koji se nalaze u određenome predmetu. Tablica *UserSubjectJoinRequests* označava korisnike koji su zatražili pristup određenome predmetu. *SubjectNotices* se nalazi između tablica *Subject* i *SubjectTeachers* te sadržava obavijesti kreirane od nastavnika u nekome predmetu. Svaki predmet ima svoja pitanja koja su spremljena u tablicu *Questions*, a pitanja se koriste pri kreiranju ispita u tablici *Exams* pomoću pomoćne tablice

ExamQuestions. Ispiti obavezno imaju barem jedan od kreiranih kriterija ocjenjivanja u nekome predmetu, spremljenih u tablicu *SubjectGradingCriteria*. Učenici nekog predmeta su za svaki kreirani ispit smješteni u tablicu *ExamStudents*, a nakon što je ispit riješen, spremljen je u tablici *SolvedExams*. Budući da je korišten izrada koda prvo pristup u izradi sloja baze podataka, sve potrebne izmjene baze su obavljene u obliku migracija. Plavom bojom je označena tablica *_EFMigrationsHistory* u kojoj se nalaze podaci o migracijama baze.

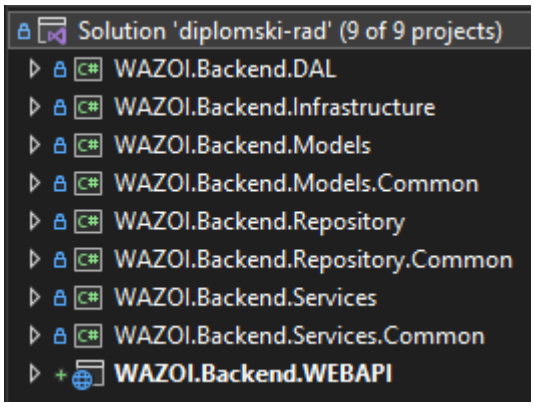


Slika 4.4. Korišteni model baze podataka.

4.2.2. Struktura projekta ASP.NET Core

Programsko rješenje implementacije poslovnog sloja i sloja pristupa podacima odrađeno je u ASP.NET Core-u projektu. Projekt je organiziran u strogo definiranu strukturu koja omogućava lako dodavanje novih funkcionalnosti i izmjenu postojećih rješenja. Slika 4.5 prikazuje rješenje (engl. *solution*) projekta. *Solution* sadrži devet projekata. Nazivi projekata su sukladni njihovoj funkciji. *DAL* projekt označava sloj pristupa podacima i u njemu je definirana sva logika sloja za pristup podacima. *Infrastructure* projekt sadrži definiciju nekih od globalnih postavki aplikacije i omogućava njihovo korištenje kroz cijeli *solution*. *Models.Common* projekt definira sučelja za

klase koje služe prijenosu podataka (engl. *Data Transfer Object*, DTO). Za svaki entitet iz *DAL* projekta definirano je pripadno sučelje u *Models.Common* projektu, a implementacija pripadnih sučelja nalazi su u projektu *Models.* Odnos između *Models.Common* i *Models* projekata poštuje princip *inversion of control.* Takav pristup korišten je između projekata *Services.Common* i *Services* te između projekata *Repository.Common* i *Repository*. Projekt *Repository.Common* definira sučelja za repozitorije čija implementacija se nalazi u *Repository* projektu. On služi za dohvaćanje i spremanje podataka, a to obavlja pomoću direktne komunikacije s *DAL* dijelom. *Services.Common* i *Services* slijede isti princip definiranja sučelja. *Services* projekt služi kao spona između *API* i *Repository* projekata. U *Services* se odvija sva logika aplikacije te obrada podataka. *API* projekt sadrži kontrolere koji služe kao pristupna točka programa, odnosno definiraju sve *endpointe* koje poslužitelj omogućava. Unutar *API* projekta nalazi se *Program.cs* datoteka (engl. *file*) koja se prilikom izgradnje aplikacije pokreće. U *Program.cs* su definirane sve postavke aplikacije.



Slika 4.5. Struktura ASP.NET Core projekta.

4.2.3. Prikaz multilayer strukture nad entitetom Exam

Na primjeru entiteta *Exam* u nastavku je objašnjen *flow* kroz *multilayer* strukturu ASP.NET Core projekta. Slika 4.6 prikazuje *ExamController* i njegove *field*-ove. Svi *fieldovi* inicijaliziraju se pomoću *dependency injection*-a kroz konstruktor kontrolera. Kontroler u sebi sadrži *field _mapper* koji je tipa *IMapper.* On služi za mapiranje primljenih podataka kroz HTTP zahtjev u *DTO* modele aplikacije. Ostali *field*-ovi su servisi koji se u kontroler dodaju ovisno o zadaći i potrebama kontrolera.

```
14      [Route("api/[controller]")]
15      [ApiController]
        1 reference
16      public class ExamController : ControllerBase
17      {
18          #region Fields
19
20          protected readonly IMapper _mapper;
21          private readonly IExamService _service;
22          private readonly IExamStudentService _examStudentService;
23          private ISubjectStudentService _subjectStudentService;
24
25          #endregion Fields
26
27          #region Constructors
28
            0 references
29          public ExamController(IExamService service,
30              IExamStudentService examStudentService,
31              IMapper mapper,
32              ISubjectStudentService subjectStudentService)
33          {
34              _service = service;
35              _examStudentService = examStudentService;
36              _mapper = mapper;
37              _subjectStudentService = subjectStudentService;
38          }
39
40          #endregion Constructors
41
42          Methods
198     }
199
```

Slika 4.6. *ExamController.*

Jedan od *endpointa* na *ExamController*-u služi za predaju ispita tijekom rješavanja od strane učenika. Slika 4.7 prikazuje navedeni *endpoint*. Vrsta *endpointa* je *HTTP POST*, a podaci koji mu se prosljeđuju definirani su klasom, odnosni *REST* modelom *SolvedExamSubmitRest.* Prilikom ulaska u metodu, primljeni podaci u *REST* modelu mapiranju se u *DTO* model. Nakon što su podaci mapirani u *DTO* model, poziva se metoda *CalculateAndSaveSolvedExamAsync()* definirana u *ExamService.*

```
106         [HttpPost]
107         [Route("solved-exam-submit")]
            0 references
108         public async Task<IActionResult> PostSolvedExam(SolvedExamSubmitRest solvedExamSubmit)
109         {
110             var solvedExamSubmitDomain = _mapper.Map<ISolvedExamSubmitDto>(solvedExamSubmit);
111
112             var result = await _service.CalculateAndSaveSolvedExamAsync(solvedExamSubmitDomain);
113
114             if (result)
115             {
116                 return Ok
117                     (result);
118             }
119             else {
120                 return BadRequest();
121             }
122         }
123
```

Slika 4.7. *Endpoint* za prihvaćanje riješenih ispita.

Slika 4.8 prikazuje klase *SolvedExamSubmitRest* i *SolvedExamAnswerReast* koje se mapiraju u klase koje implementiraju sučelja *IsolvedExamSubmitDto* i *ISolvedExamAnswerDto*.

```
234 public class SolvedExamSubmitRest
235 {
236     #region Properties
237
        0 references
238     public Guid ExamId { get; set; }
        0 references
239     public Guid StudentExamId { get; set; }
        0 references
240     public List<SolvedExamAnswerRest>? Answers { get; set; }
241
242     #endregion Properties
243 }
```

```
9  public interface ISolvedExamSubmitDto
10 {
11     #region Properties
12
       3 references
13     Guid ExamId { get; set; }
       2 references
14     Guid StudentExamId { get; set; }
       2 references
15     List<ISolvedExamAnswerDto> Answers { get; set; }
16
17     #endregion Properties
18 }
```

```
245 public class SolvedExamAnswerRest
246 {
247     #region Properties
248
        0 references
249     public Guid Id { get; set; }
250
        0 references
251     public string Answer { get; set; }
252
253     #endregion Properties
254 }
```

```
9  public interface ISolvedExamAnswerDto
10 {
11     #region Properties
12
       2 references
13     Guid Id { get; set; }
       5 references
14     string Answer { get; set; }
15
16     #endregion Properties
17 }
```

Slika 4.8. Primjeri *REST* modela i sučelja *DTO* modela.

Metoda *CalculateAndSaveSolvedExamAsync* prikazana je na slici 4.9. Linija broj 91 prikazuje dohvaćanje svih pitanja iz baze za predani ispit, odnosno za predani primarni ključ entiteta *Exam*. Nakon što su dohvaćena sva pitanja iz baze, dohvaćaju se podaci o ispitu, ali ujedno i podaci iz pripadnog entiteta *ExamCriteria*. Nakon kreiranja objekta tipa *SolvedExamDto*, dostupni su svi podaci za evaluaciju riješenosti ispita. Pomoću *foreach* petlje prolazi se kroz sva predana pitanja iz *SolvedExamSubmitRest* modela. Pri svakoj iteraciji petlje uspoređuje se predani odgovor za pojedino pitanje s točnim odgovorom dohvaćenim iz baze.

```
89      public async Task<bool> CalculateAndSaveSolvedExamAsync(ISolvedExamSubmitDto solvedExamSubmitDomain)
90      {
91          var examQuestions = await _examQuestionService.GetExamQuestionsAsync(solvedExamSubmitDomain.ExamId);
92          var exam = await _repository.GetExamWithCriteriaAsync(solvedExamSubmitDomain.ExamId);
93
94          var solvedExam = new SolvedExamDto();
95          solvedExam.ExamStudentId = solvedExamSubmitDomain.StudentExamId;
96          solvedExam.isLocked = false;
97          solvedExam.ScoredPoints = 0;
98          solvedExam.MaximumPoints = 0;
99
100         JObject solvedExamContent = new JObject();
101         List<JObject> solvedExamContentList = new List<JObject>();
102
103         foreach (var submittedAnswer in solvedExamSubmitDomain.Answers)
104         {
105             var question = examQuestions.FirstOrDefault(x => x.Id == submittedAnswer.Id);
106
107             solvedExam.MaximumPoints += question.Points;
108
109             var answerJson = JObject.Parse(question.Question.Content);
110
111             if (question.Question.Type.Equals(DefaultApplicationValues.TrueFalse))...
134             else if (question.Question.Type.Equals(DefaultApplicationValues.SingleChoice))...
168             else if (question.Question.Type.Equals(DefaultApplicationValues.SingleWord))...
191             else if (question.Question.Type.Equals(DefaultApplicationValues.MultiChoice))...
248         }
249
250         solvedExam.ScoredPercentage = (float)Math.Round((double)solvedExam.ScoredPoints
251             / (double)solvedExam.MaximumPoints * 100, 2);
252
253         if (solvedExam.ScoredPercentage >= exam.GradingCriterion.GradeA)...
257         else if (solvedExam.ScoredPercentage >= exam.GradingCriterion.GradeB)...
261         else if (solvedExam.ScoredPercentage >= exam.GradingCriterion.GradeC)...
265         else if (solvedExam.ScoredPercentage >= exam.GradingCriterion.GradeD)...
269         else...
273
274         solvedExamContent["items"] = JArray.FromObject(solvedExamContentList);
275         solvedExam.Content = solvedExamContent.ToString(Newtonsoft.Json.Formatting.None);
276         var result = await _solvedExamService.CreateSolvedExamAsync(solvedExam);
277         return result > 0;
278     }
```

Slika 4.9. Implementacija metode za računanje postignutog broja bodova za predani ispit.

U aplikaciji su dostupne četiri vrste pitanja. Slika 4.10 prikazuje dio metode koji se odnosi na evaluaciju odgovora ukoliko je pitanje tipa točno/netočno. Budući da su pitanja u bazi spremljena u *JSON* formatu, vrijednost točnog odgovora se izvlači iz *JSON* objekta. Nakon toga se pravi *JSON* zapis podataka o pitanju i točnog odgovora. Taj zapis koristi se na *front-end* dijelu aplikacije pri *renderanju* ispita prilikom dodatnog ocjenjivanja ili pregleda ispita. Pri kraju iteracije, ukoliko je dani odgovor točan, dodjeljuju se ostvareni bodovi za pojedino pitanje.

```
111                 if (question.Question.Type.Equals(DefaultApplicationValues.TrueFalse))
112                 {
113                     var submittedAnswerValue = submittedAnswer.Answer;
114                     var correctAnswerValue = answerJson.Value<string>("correctAnswer");
115
116                     JObject solvedExamContentItem = new JObject
117                     {
118                         new JProperty("type", question.Question.Type),
119                         new JProperty("stem", answerJson.Value<string>("stem")),
120                         new JProperty("correctAnswer", correctAnswerValue),
121                         new JProperty("answered", submittedAnswerValue),
122                         new JProperty("maximumPoints", question.Points),
123                     };
124
125                     if (correctAnswerValue.Equals(submittedAnswerValue, StringComparison.InvariantCultureIgnoreCase))
126                     {
127                         solvedExam.ScoredPoints += question.Points;
128                         solvedExamContentItem["scoredPoints"] = question.Points;
129                     }
130                     else
131                     {
132                         solvedExamContentItem["scoredPoints"] = 0;
133                     }
134
135                     solvedExamContentList.Add(solvedExamContentItem);
136                 }
```

Slika 4.10. Implementacija metode za računanje postignutog broja bodova za predani ispit.

Nakon prolaska kroz *foreach* petlju računa se ostvareni postotak riješenosti ispita te se ovisno o kriteriju ocjenjivanja koji je pridružen ispitu, odlučuje konačna ocjena. Kada je odrađena obrada podataka, poziva se *CreateSolvedExamAsync* metoda iz *SolvedExamService.* Njena implementacija prikazana je slikom 4.11. Ona poziva *Insert* metodu na svome pripadnome repozitoriju (*SolvedExamRepository*) te pomoću *UnitOfWork* principa (transakcija) završava spremanje podataka u bazu.

```
35          public async Task<int> CreateSolvedExamAsync(ISolvedExamDto solvedExam)
36          {
37              _repository.Insert(solvedExam);
38              return await _uow.SaveAsync();
39          }
40
```

Slika 4.11. Implementacija metode za spremanje SolvedExam entiteta u bazu.

Insert metoda repozitorija implementirana je na razini generičkog repozitorija koji služi kao baza svim repozitorijima u aplikaciji. Slika 4.12 prikazuje *SolvedExamRepository* kako nasljeđuje generički repozitorij (*GenericRepository*) te *Insert* metodu koja se nalazi unutar generičkog repozitorija.

```
15  public class SolvedExamRepository
16      : GenericRepository<ISolvedExamDto, SolvedExam>, ISolvedExamRepository
17  {
18      #region Constructors
19
        1 reference
20      public SolvedExamRepository(AppDataContext context, IMapper mapper)
21          : base(context, mapper)
22      {
23      }
24
25      #endregion Constructors
26
27      Methods
35  }
```

```
32      public void Insert(TDto dto)
33      {
34          var entity = _mapper.Map<TEntity>(dto);
35          _table.Add(entity);
36      }
```

Slika 4.12. Prikaz nasljeđivanja generičkog repozitorija i njegova *insert* metoda.

Suradnja s bazom podataka na način spremanja ili ažuriranja njenih entiteta predstavlja opasnost od gubitka podataka. Zbog toga se promjene nad bazom ne odvijaju sve dok se ne pozove metoda *SaveAsync* koja se nalazi u klasi *UnitOfWork*. Nakon poziva metode za izvršenje transakcije, endpointu, odnosno početnoj metodi prosljeđuje se povratna informacija o izvođenju svih operacija. Ukoliko je evaluacija ispita i njegovo spremanju u bazu uspješno izvršeno, rezultat *POST* zahtjeva je *response* sa status kodom 200 koji označava uspješno obavljen zahtjev. Nasuprot tome, ukoliko je došlo do problema u izvršenju zahtjeva, rezultat je *response* sa statusom 400 koji signalizira da je došlo do problema pri izvođenju zahtjeva.

4.2.4. Administracija korisnika

Administracija korisnika na poslužitelju obuhvaća registraciju, prijavu, potvrdu računa, promjenu lozinke, dodavanje uloge korisniku i upravljanje tokenima. Sve radnje administracije korisnika omogućene su korištenjem .NET Core Identity NuGet paketa. U *Program.cs* datoteci se nalazi registriranje *Identity* paketa i omogućavanje njegovog korištenja kroz aplikaciju. Na slici 4.13 prikazana je registracija *Identity* paketa. Važno je napomenuti da programski kod prikazuje da je podešena obavezna potvrda adrese e-pošte svakog korisničkog računa. Pri generiranju tokena za resetiranje lozinke dodana je e-pošta kao uobičajeni (engl. *default*) pružatelj usluga (engl. *provider*). Nakon podešavanja opcija dodan je *defaultni* token *provider* te je u *Entity Framework Store* dodan AppDataContext koji predstavlja *IdentityDatabaseContext.*

```
117 builder.Services.AddIdentity<User, IdentityRole<Guid>>(opt =>
118 {
119     opt.SignIn.RequireConfirmedAccount = true;
120     opt.Tokens.PasswordResetTokenProvider = TokenOptions.DefaultEmailProvider;
121 })
122     .AddEntityFrameworkStores<AppDataContext>()
123     .AddDefaultTokenProviders();
```

Slika 4.13. Prikaz registracije *.NET Core Identity* servisa u *Program.cs* datoteci.

Nakon što je podešeno korištenje *Identity* servisa, podešena je autentifikacija i dodana autorizacija. Slika 4.14 prikazuje dodavanje autentifikacije. Za sve potrebe autentifikacije korištena je *JWT Bearer defaultna* shema. U *extension* metodi *AddJwtBearer()* podešene su opcije *JWT* tokena. Pri slanju zahtjeva JWT token slat će se u *Authorizaton header* dijelu zahtjeva te će biti ekstrahirana njegova vrijednost. Na liniji 150 prikazano je dodavanje autorizacije.

```
129 builder.Services.AddAuthentication(options =>
130 {
131     options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
132     options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
133     options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
134 })
135     .AddJwtBearer(options =>
136     {
137         options.SaveToken = true;
138         options.RequireHttpsMetadata = false;
139         options.TokenValidationParameters = new TokenValidationParameters()
140         {
141             ValidateLifetime = true,
142             ValidateIssuerSigningKey = true,
143             ClockSkew = TimeSpan.Zero
144         };
145     });
146
147 builder.Services.AddAuthorization();
```

Slika 4.14. Prikaz dodavanje autorizacije i autentifikacije u *Program.cs* datoteci.

Za konfiguriranja globalnih vrijednosti koje se koriste kroz aplikaciju služi datoteka *appsettings.json*. U njoj se definiraju *connection stringovi,* globalne varijable i sve druge potrebne informacije i podaci. Na slici 4.15 prikazane su definirane za korištenje *JWT* tokena. Definirana je trajanje valjanosti tokena i *refresh* tokena.

```
8  "JWT": {
9    "TokenValidityInMinutes": 120,
10   "RefreshTokenValidityInDays": 7
11 },
```

Slika 4.15. Prikaz postavki *JWT* tokena u *appsettings.json* datoteci.

Svi *endpointi* za administraciju korisnika nalaze se u datoteci *AuthenticateController.cs*. Slika 4.16 prikazuje *POST* metodu za registraciju korisnika. Prilikom ulaska u metodu, vrši se provjera postoji li već registrirani korisnik s predanom adresom e-pošte. Ako korisnik s pripadnom adresom e-pošte već postoji, prekida se proces registracije uz *status code 409* koji označava konflikt u zahtjevu. Ukoliko korisnik s predanom adresom e-pošte ne postoji, nastavlja se proces registracije. Kreira se objekt tipa *User* te se pomoću servisa *UserManager* kreira novi korisnički račun. Korisniku se nakon toga dodaje *role* koji je pri registraciji odabrao. Budući da je u potvrda adrese e-pošte nakon registracije postavljena kao obavezna, na e-poštu se šalje obavijest o registraciji te je korisnički račun potrebno potvrditi pomoću primljene poveznice.

```
137 [HttpPost]
138 [Route("register")]
    0 references
139 public async Task<IActionResult> Register([FromBody] RegisterDataRest registerData)
140 {
141     var userExists = await _userManager.FindByEmailAsync(registerData.Email);
142     if (userExists != null)
143         return StatusCode(StatusCodes.Status409Conflict,
144             new Response { Status = "Error", Message = "User already exists!" });
145
146     User user = new()
147     {
148         UserName = registerData.Username,
149         Email = registerData.Email,
150         Name = registerData.Name,
151         Surname = registerData.Surname,
152         SecurityStamp = Guid.NewGuid().ToString(),
153     };
154     var result = await _userManager.CreateAsync(user, registerData.Password);
155     if (!result.Succeeded)
156         return StatusCode(StatusCodes.Status500InternalServerError,
157             new Response { Status = "Error", Message = "User creation failed!" });
158
159     if (!await _roleManager.RoleExistsAsync(UserRoles.Student))
160         await _roleManager.CreateAsync(new IdentityRole<Guid>(UserRoles.Student));
161     if (!await _roleManager.RoleExistsAsync(UserRoles.Teacher))
162         await _roleManager.CreateAsync(new IdentityRole<Guid>(UserRoles.Teacher));
163
164     if (registerData.Role == UserRoles.Student)
165     {
166         await _userManager.AddToRoleAsync(user, UserRoles.Student);
167     }
168     if (registerData.Role == UserRoles.Teacher)
169     {
170         await _userManager.AddToRoleAsync(user, UserRoles.Teacher);
171     }
172
173     var token = await _userManager.GenerateEmailConfirmationTokenAsync(user);
174
175     await _emailSender.SendRegistrationEmailAsync(user.Email, token);
176
177     return Ok(new Response { Status = "Success", Message = "User created successfully!" });
178 }
```

Slika 4.16. *Endpoint* za registraciju korisnika.

Nakon registracije, potvrda korisničkog računa se odvija pomoću *POST endpointa* na slici 4.17. Pozivom metode *ConfirmEmailAsync()* na *UserManager* servisu obavlja se provjera s primljenim tokenom i pronađenim korisnikom iz baze. Ukoliko je rezultat poziva metode uspješan, korisnički račun je potvrđen.

```
180     [HttpPost]
181     [Route("confirm-email")]
        0 references
182     public async Task<IActionResult> ConfirmEmail(string token, string email)
183     {
184         var user = await _userManager.FindByEmailAsync(email);
185         var test = Uri.UnescapeDataString(token);
186         var result = await _userManager.ConfirmEmailAsync(user, test);
187
188         if (!result.Succeeded)
189         {
190             return BadRequest();
191         }
192         else
193         {
194             return Ok(new
195             {
196                 Status = true,
197                 Message = "Your email is confirmed succesfully",
198                 StatusCode = System.Net.HttpStatusCode.OK,
199             });
200         }
201     }
```

Slika 4.17. *Endpoint* za potvrdu korisničkog računa.

Nakon što je korisnički račun registriran te je adresa e-pošte potvrđena, korisnik se može prijaviti pomoću *login endpointa*. *Login* metoda prikazana je na slici 4.18. Obavlja se provjera predane adrese e-pošte te ukoliko je adresa validna, pronađen je korisnik. Nakon toga provjerava se točnost predane lozinke za račun. Ako je predana lozinka valjana, korisniku se u *response* vraćaju podaci o korisničkome računu. Među tim nalaze se token za pristup (engl. *access token*) i *refresh token*. Svi podaci se spremaju u lokalnu memoriju aplikacije. *Access* token se koristi pri slanju zahtjeva na *endpointe* za koje je potrebno biti autorizirani korisnik. Generirani token vrijedi 2 sata, a nakon toga je za nastavak korištenja aplikacije potreban novi token.

```
53  [HttpPost]
54  [Route("login")]
    0 references
55  public async Task<IActionResult> Login([FromBody] LoginDataRest loginData)
56  {
57      var user = await _userManager.FindByEmailAsync(loginData.Email);
58      if (user != null && await _userManager.CheckPasswordAsync(user, loginData.Password))
59      {
60          var userRoles = await _userManager.GetRolesAsync(user);
61
62          var authClaims = new List<Claim>
63          {
64              new Claim(ClaimTypes.Name, user.UserName),
65              new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
66          };
67
68          foreach (var userRole in userRoles)
69          {
70              authClaims.Add(new Claim(ClaimTypes.Role, userRole));
71          }
72
73          var token = CreateToken(authClaims);
74          var refreshToken = GenerateRefreshToken();
75
76          _ = int.TryParse(_configuration["JWT:RefreshTokenValidityInDays"], out int refreshTokenValidityInDays);
77
78          user.RefreshToken = refreshToken;
79          user.RefreshTokenExpiryTime = DateTime.UtcNow.AddDays(refreshTokenValidityInDays);
80
81          await _userManager.UpdateAsync(user);
82
83          return Ok(new
84          {
85              Email = loginData.Email,
86              Token = new JwtSecurityTokenHandler().WriteToken(token),
87              RefreshToken = refreshToken,
88              Expiration = token.ValidTo,
89              Role = userRoles.First(),
90              Name = user.Name,
91              Surname = user.Surname
92          });
93      }
94      return Unauthorized();
95  }
```

Slika 4.18. *Endpoint* za prijavu korisnika.

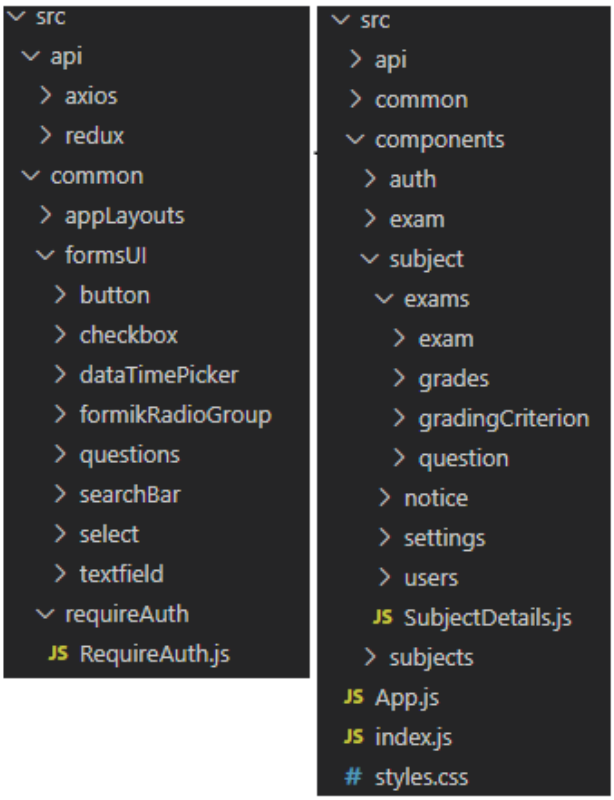
Nakon što korisniku istekne *access token*, za nastavak korištenja aplikacije, potreban mu je novi token. U tom slučaju, korisnik pomoću svoga isteklog tokena i aktivnog *refresh tokena* odlaskom na *endpoint* za *refresh token* dobiva novi *access token.* Ovakav *flow* omogućava korisniku neometan rad, bez potrebe da se korisnik pri isteku *access tokena* mora ponovno prijavljivati na sustav.

4.3. Programsko rješenje na strani klijenta

Programsko rješenje na strani klijenta je aplikacija koja predstavlja prezentacijski sloj sustava. Aplikacija je izrađena u JavaScript jeziku, odnosno biblioteci React. Neke od značajnijih dodatno korištenih biblioteka su: MaterialUI, Axios, React-Router, Redux itd.

4.3.1. Struktura direktorija aplikacije

Struktura direktorija aplikacije podijeljena je u smislene cjeline koje su prikazane na slici 4.19. Unutar direktorija *src* nalaze se sve kreirane datoteke aplikacije. Direktorij *api* sadrži sve potrebno za ostvarivanje komunikacije s API-jem te lokalno spremanje podataka o ulogiranome korisniku. Direktorij *Common* sadrži React komponente koje se koriste kroz cijeli projekt. Unutar *appLayouts* se nalaze dva različita *layouta*, jedan od njih se koristi ukoliko je korisnik prijavljen, a drugo ukoliko korisnik nije prijavljen. Direktorij *formsUI* sadrži React komponente koje se koriste unutar formi pri komunikaciji s *API*-jem. Datoteka *RequireAuth.js* predstavlja React komponentu koja se koristi kao *wrapper* komponenta oko onih dijelova aplikacije kojima nije dopušten neautorizirani pristup. Sve preostale React komponente aplikacije nalaze se u direktoriju *components* te su organizirane u funkcionalne komponente. Direktorij *auth* sadrži sve komponente koje se odnose na autentifikaciju korisnika. Unutar direktorija *subjects* nalaze se sve komponente koje pružaju pregled svih kreiranih predmeta na aplikaciji. Unutar *subject* direktorija nalaze se sve komponente kojima se pristupa nakon što je odabran pojedini predmet. Direktorij *exam* sadrži komponente za rješavanje ispita i njegovo ručno ispravljanje.



Slika 4.19. Struktura direktorija klijentske aplikacije.

4.3.2. Komunikacija s API-jem

Za komunikaciju s *API*-jem korištena je biblioteka *axios.* U zahtjevima kao što su *register* i *login*, korištena je bazna instanca *axiosa*. Primjer korištenja *axios* instance dan je slikom 4.20. Prikazani zahtjev odlazi na *endpoint* koji je svima dostupan, odnosno ne zahtjeva token za uspješan pristup.

```
41  let response = await axios.post(baseURL + "/Authenticate/login", {
42      ...values,
43  });
```

Slika 4.20. Primjer korištenja *axios* instance za POST zahtjev.

Za korištenje aplikacije potrebna je prijava te korištenje dobivenih tokena s *API*-ja za uspješno komuniciranje s ostalim slojevima sustava. Za korištenje dobivenih tokena oni se pri slanju zahtjeva dodaju u *header* dijelu zahtjeva. Zbog toga se u svim onim zahtjevima gdje je potreban token za potrebe autentifikacije koristi *custom* kreirani *hook* prikazan na slici 4.21. Ukoliko neki zahtjev vrati *response* sa statusom 401, to govori da je token istekao. Nakon što se dobije *response* 401, *axios middleware* šalje novi zahtjev s ciljem da se dohvate novi token i *refresh* token. Ukoliko su uspješno dohvaćeni novi tokeni, prvotni zahtjev se ponavlja, ali s novim važećim tokenom u *headeru.*

```
17  axiosInstance.interceptors.response.use(
18      (res) => {
19          return res;
20      },
21      async (err) => {
22          const originalConfig = err.config;
23          if (err.response.status === 401 && !originalConfig._retry) {
24              originalConfig._retry = true;
25              try {
26                  const response = await axios.post(
27                      `${baseURL}/Authenticate/refresh-token/`,
28                      {
29                          accessToken: token,
30                          refreshToken: refreshToken,
31                      }
32                  );
33
34                  dispatch(setCredentials(...response.data));
35                  return axiosInstance(originalConfig);
36              } catch (_error) {
37                  return Promise.reject(_error);
38              }
39          }
40
41          return Promise.reject(err);
42      }
43  );
```

Slika 4.21. *Axios middleware* za dohvaćanje novog tokena.

4.3.3. Prikaz korištenja React hook-ova

Za upravljanje podacima i renderanju komponenti ovisno o podacima, korišten je React *hook useState().* Slika 4.22 prikazuje dio komponente *SubjectList* gdje su definirani *state*-ovi u ovisnosti o kojima će se renderati stavke na stranici.

```
19  export const SubjectList = ({ userOnly }) => {
20
21      let [page, setPage] = useState(1);
22      let [pageCount, setPageCount] = useState(1);
23      let [searchQuery, setSearchQuery] = useState("");
24      let [searchQueryFromClick, setSearchQueryFromClick] = useState("");
25      let [predmeti, setPredmeti] = useState([]);
26      let [isLoading, setIsLoading] = useState(true);
```

Slika 4.22. *State*-ovi korišteni u komponenti *SubjectList*.

Za potrebe dohvaćanja podataka prilikom renderanja *SubjectList* komponente, korišten je React *useEffect() hook.* Slika 4.23 prikazuje funkciju *getSubjects* koja služi za dohvaćanje predmeta prema predanim parametrima u *axios* zahtjev-u. Prilikom izvršenja zahtjeva *state isLoading* se postavlja na *true* te se na stranici rendera *loader* kao pokazivač da je učitavanje predmeta u tijeku. Nakon što se obavi zahtjev, *isLoading* se postavlja na *false* vrijednost te se *loader* više ne prikazuje na stranici. Osim što poziva funkciju *GetSubjects*, *useEffect* u svojoj *dependency* listi prihvaća *state*-ove. Prihvaćanjem *state*-ova u *dependency* listu, *useEffect hooku*-u se naređuje ponovno pozivanje svaki puta kada neki od *state*-ova u toj listi promjeni vrijednost. U ovome slučaju ukoliko se promjeni jedan od *state*-ova *searchQueryFromClick* ili *userOnly.*

```
63      const getSubjects = async () => {
64          try {
65              setIsLoading(true);
66              let result;
67              if (userOnly || role === "Teacher") {
68                  result = await api.get(
69                      `/Subject/find-subjects-by-user?page=${page}&rpp=6&searchQuery=${searchQuery}`
70                  );
71              } else {
72                  result = await api.get(
73                      `/Subject/find-subjects-with-joinStatus?page=${page}&rpp=6&searchQuery=${searchQuery}`
74                  );
75              }
76
77              setSubjects(result.data.items);
78              if (result.data.totalCount % 6 > 0) {
79                  setPageCount(parseInt(result.data.totalCount / 6 + 1));
80              } else {
81                  setPageCount(parseInt(result.data.totalCount / 6));
82              }
83              setIsLoading(false);
84          } catch (error) {
85              setIsLoading(false);
86          }
87      };
88
89      useEffect(() => {
90          getSubjects();
91      }, [searchQueryFromClick, userOnly]);
```

Slika 4.23. Prikaz funkcije za dohvaćanje predmeta i *useEffect hook*-a koji ju poziva.

4.3.4. Primjer rada s formama na komponenti *Register*

U aplikaciji se nalaze mnoge formi koje služe za slanje podataka na API. Primjer koda koji definira formu za registraciju prikazan je slikom 4.24. Konstanta *INITIAL_FORM_STATE* definira polja dostupna kroz formu. Druga vidljiva konstanta, *ValidationSchema,* definira pravila koja se moraju poštovati prije nego li se forma može predati. Prikazana *ValidationSchema* govori da su sva polja tipa *string* potrebna te da su obavezna. Dodatno, polje *role* mora imati vrijednost *Student* ili *Teacher.* Ukoliko sva definirana pravila forme nisu poštovana, forma se neće moći predati.

```
20  const INITIAL_FORM_STATE = {
21      name: "",
22      surname: "",
23      role: "",
24      email: "",
25      password: "",
26  };
27
28  const validationSchema = yup.object({
29      name: yup.string("Unesite ime").required("Niste unijeli ime"),
30      surname: yup.string("Unesite prezime").required("Niste unijeli prezime"),
31      role: yup
32          .string()
33          .required("Odaberite ulogu")
34          .oneOf(["Student", "Teacher"], "Odaberite ulogu"),
35      email: yup.string().email("Unesite email").required("Niste unijeli email"),
36      password: yup
37          .string()
38          .required("Niste unijeli lozinku")
39          .matches(
40              "^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-,.]).{8,}$",
41              "Minimalno 8 znakova i po jedno: veliko slovo, malo slovo, broj i poseban znak."
42          ),
43  });
44
```

Slika 4.24. Primjer koda forme za registraciju.

Slika 4.25 prikazuje korištenje *Formik* komponente unutar koje se smješta forma. Vidljivo je da su *Formik* komponenti kao svojstva predani *initialValues*, *validationSchema* i *onSubmit* funkcija. Linije 102 i 105 prikazuju korištenje definiranih polja u formi prosljeđivanjem svojstva *name* komponentama *Textfield* koje podržavaju rad s formama. Na liniji 108 nalazi se komponenta *Select* koja služi za odabir uloge korisnika pri registraciji. Ona upravlja *fieldom role.*

```
84                  <Formik
85                      initialValues={{ ...INITIAL_FORM_STATE }}
86                      validationSchema={validationSchema}
87                      onSubmit={handleSubmit}
88                  >
89                      <Form>
90                          <Grid
91                              sx={{ pb: 2 }}
92                              container
93                              spacing={3}
94                              justifyContent="center"
95                          >
96                              <Grid item xs={12}> ···
100                             </Grid>
101                             <Grid item xs={8}>
102                                 <Textfield name="name" label="Ime" />
103                             </Grid>
104                             <Grid item xs={8}>
105                                 <Textfield name="surname" label="Prezime" />
106                             </Grid>
107                             <Grid item xs={8}>
108                                 <Select
109                                     name="role"
110                                     label="profesija"
111                                     options={{
112                                         Student: "Učenik",
113                                         Teacher: "Nastavnik",
114                                     }}
115                                 />
116                             </Grid>
```

Slika 4.25. Primjer korištenja komponente *Formik*.

Za predaju forme koristi se funkcija predana kao *property onSubmit*. Slika 4.26 prikazuje dio funkcije koja podatke iz forme šalje na *API* u obliku POST zahtjeva. Podaci iz forme su funkciji predani kroz podatak *values.* Pri predaji podataka POST metodi korišteno je destruktuiranje vrijednosti. U argumentima *post* metode predaje se *baseURL* koji predstavlja putanju API-ja koja je zajednička svim *endpoint*-ima. Dio argumenta „*Authenticate*“ govori da metoda odlazi na kontroler imena *AuthenticateController,* a „*register*“ dio govori koji *endpoint* s tog kontrolera se koristi za prihvat zahtjeva.

```
52      const handleSubmit = async (values) => {
53          try {
54              setisLoading(true);
55              let response = await axios.post(
56                  baseURL + "/Authenticate/register",
57                  {
58                      ...values,
59                      username: values.email,
60                  }
61              );
```

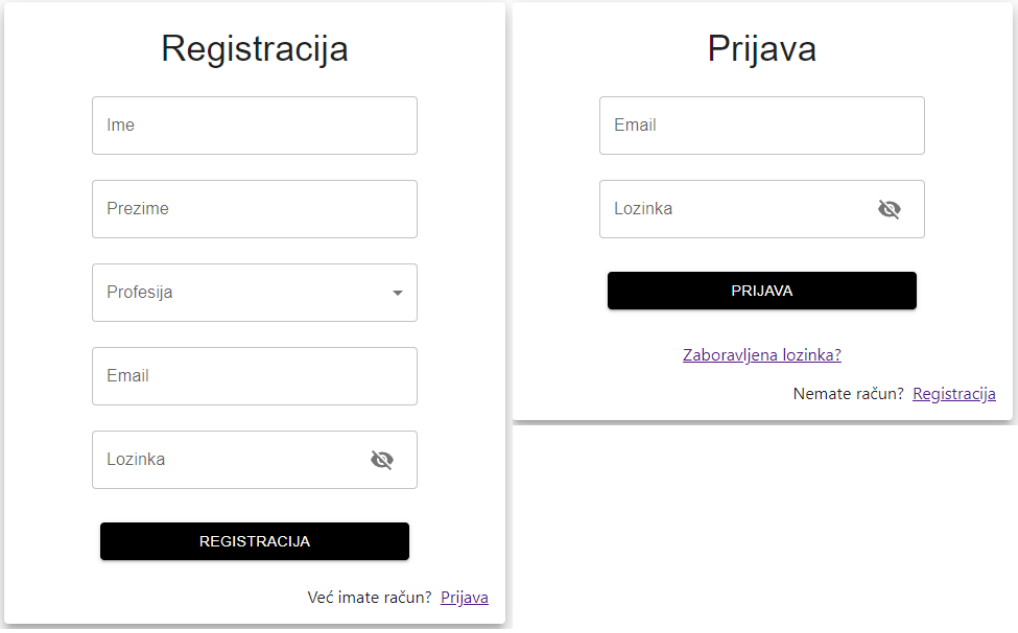
Slika 4.26. Primjer funkcije za predaju forme pomoću POST zahtjeva.

5. KORIŠTENJE APLIKACIJE

U nastavku je objašnjeno korištenje aplikacije uz prikazane dijelove korisničkoga sučelja. Objašnjen je proces korištenja. Za korištenje aplikacije je potreban korisnički račun. Prvenstveno su objašnjene registracija i prijava. Nakon toga nastavnik kreira predmet na koji mu se mogu pridružiti učenici. Na određenom predmetu prikazano je kreiranje kriterija ocjenjivanja, pitanja te ispita. Učenici imaju mogućnost rješavanja kreiranih ispita, a nakon toga slijedi prikaz dodatnog ocjenjivanja ispita te zaključavanje rezultata ispita. Dostupan je i grafički prikaz uspješnost učenika na predmetu.

5.1. Registracija i prijava korisnika

Slika 5.1 prikazuje forme za registraciju i prijavu korisnika. Forma za registraciju sadržava ime, prezime, profesiju, adresu e-pošte i lozinku. Forme se ne mogu predati ukoliko sva polja nisu ispunjena. Osim što polja moraju biti ispunjena, trebaju zadovoljiti i pravila svakog pojedinačnog polja. Polje e-pošte zahtjeva unos pravilne adrese, a lozinka mora sadržavati minimalno osam znakova, od kojih su barem po jedno veliko slovo, malo slovo, broj i poseban znak.



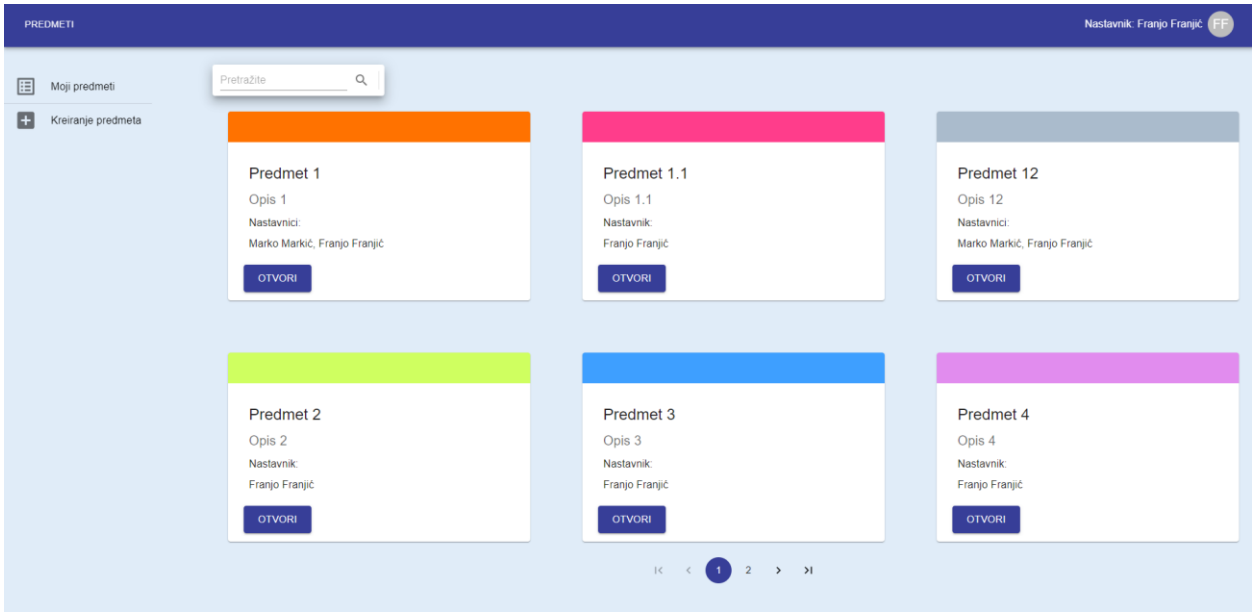
Slika 5.1 Forme za registraciju i prijavu.

Nakon registracije, na adresu e-pošte šalje se generirana poveznica za potvrdu korisničkoga računa. Otvaranjem poveznice potvrđuje se korisnički račun te se korisnik može uspješno prijaviti.

Ukoliko je korisnik zaboravio lozinku, može zatražiti novu lozinku te će mu se na adresu e-pošte poslati poveznicu za ponovno postavljanje lozinke.

5.2. Stranica s predmetima

Nakon što se korisnik prijavi, odlazi na stranicu koja sadržava pregled svih postojećih predmeta. Slika 5.2 prikazuje stranicu *Predmeti*. Na vrhu stranice se nalazi alatna traka koja je prisutna kroz cijelu aplikaciju, osim tijekom rješavanja ispita. Alatna traka sadrži tipku koja vodi na stranicu predmeti te ikonu koja pruža mogućnost odjave korisnika i promjenu njegove lozinke. Lijevo na stranici nalazi se izbornik navigacije. Izbornik korisniku u ulozi *nastavnik* omogućava prikaz svih njegovih predmeta te nudi mogućnost kreiranja predmeta. Ukoliko je korisnik u ulozi *učenik*, on u izborniku nema mogućnost kreiranja predmeta, ali ima dodatnu mogućnost prikaza svih postojećih predmeta na stranici. Nastavnik može pristupiti svim svojim predmetima, a učenik samo onima u koje je prijavljen. Ukoliko učenik želi imati pristup određenome predmetu, mora zatražiti pristup. Nakon toga ga jedan od nastavnika predmeta može prihvatiti.



Slika 5.2. Prikaz stranice predmeti.

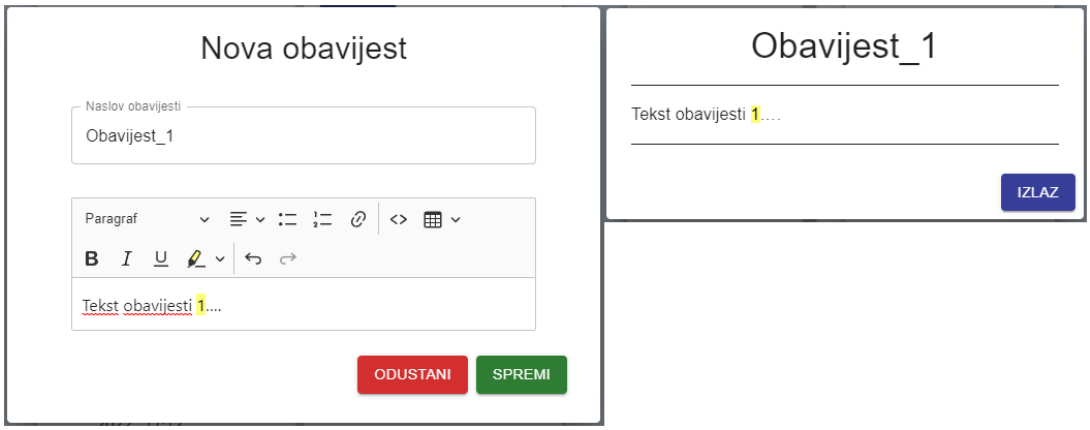
Nakon što korisnik otvori predmet, prikazuje mu se sučelje predmeta. Mogućnosti prikazane na sučelju razlikuju se ovisno o ulozi korisnika. U nastavku su objašnjene mogućnosti unutar predmeta ovisno o ulozi korisnika.

5.3. Uporaba aplikacije u ulozi nastavnika

Unutar predmeta se može nalaziti jedan ili više nastavnika te proizvoljan broj učenika. Korisnik, odnosno nastavnik, koji je kreirao predmet dobiva administratorske ovlasti nad tim predmetom. Administratorske ovlasti dodatno omogućavaju dodavanje drugih profesora na predmet, izmjenu podataka o predmetu i mogućnost brisanja predmeta. Ostale mogućnosti unutar predmeta dostupne su svim nastavnicima tog predmeta.

5.3.1. Obavijesti

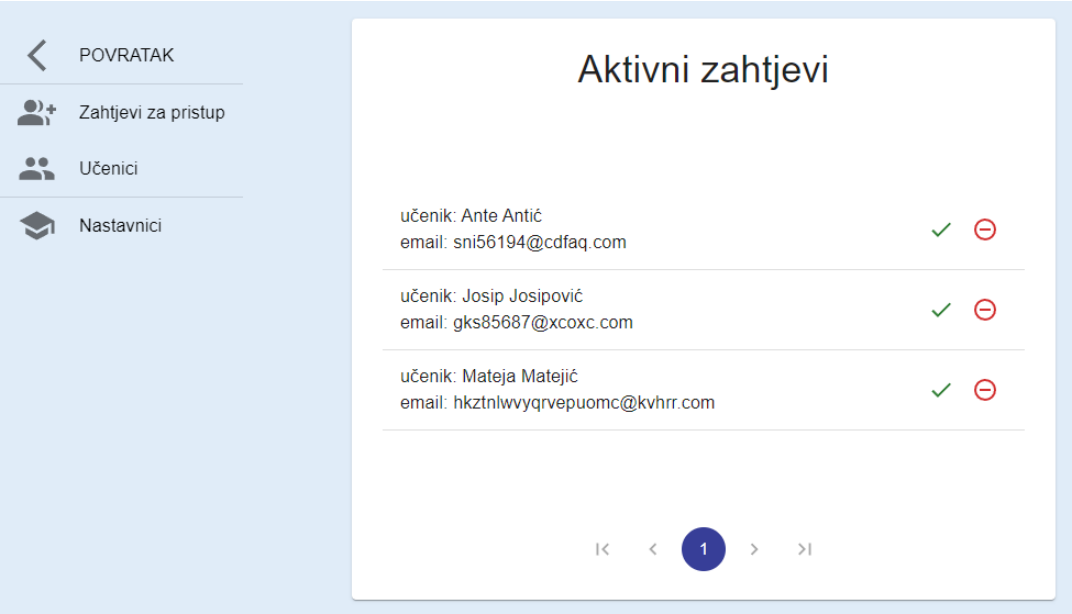
Svaki nastavak ima mogućnost kreiranja obavijesti. Slika 5.3 prikazuje sučelje pregleda i kreiranja obavijesti. Obavijesti se kreiraju u uređivaču obogaćenog teksta (engl. *rich text editor*). Kreirane obavijesti su dostupne svim učenicima i nastavnicima na pregled.



Slika 5.3. Kreiranje i prikaz kreirane obavijesti.

5.3.2. Upravljanje korisnicima

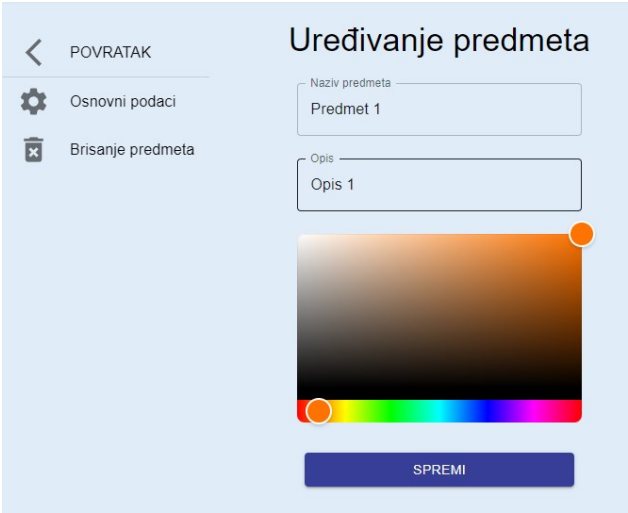
Nastavnicima je unutar predmeta omogućena stranica za upravljanje korisnicima toga predmeta te je njen izgled prikazan na slici 5.4. U izborniku postoje tri stavke. Trenutno odabrana stavka je *Zahtjevi za pristup* te ona prikazuje aktivne zahtjeve učenika za pristup predmetu. U izborniku su još dostupne dvije kontrole. Kontrola *Učenici* je za pregled svih učenika koji se nalaze u predmetu, a omogućeno je i njihovo uklanjanje. Preostala kontrola, *Nastavnici,* vidljiva je jedino nastavniku koji je administrator predmeta, a ona omogućava dodavanje i uklanjanje drugih nastavnika u predmetu.



Slika 5.4. Stranica za upravljanje korisnicima predmeta.

5.3.3. Postavke predmeta

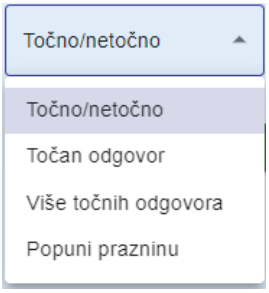
Na slici 5.5 prikazana je stranica postavki predmeta. U njoj je moguće ažurirati podatke predmeta. U izborniku se nalazi i kontrola za brisanje predmeta, ali ona je dostupna jedino administratoru predmeta.



Slika 5.5. Stranica za postavke predmeta.

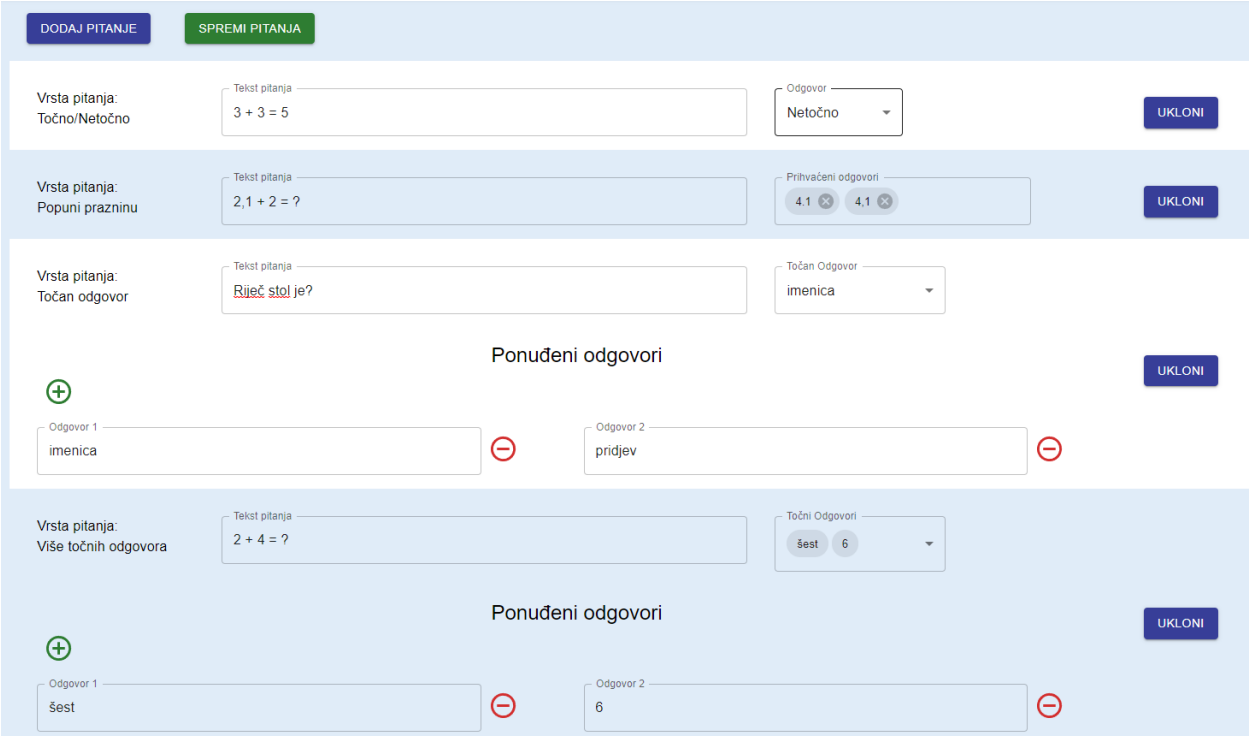
5.3.4. Kreiranje i baza pitanja

Aplikacija omogućava kreiranje četiri vrsta pitanja. Kreiranje pitanja se odvija pomoću dinamične forme. Prvo se odabire vrsta pitanja, a nakon toga se pitanje dodaje u formu. Slika 5.6 prikazuje dostupne vrste pitanja pri kreiranju.



Slika 5.6. Odabir vrste pitanja za dodavanje u formu.

Na slici 5.7 prikazana je forma koja sadrži dodanu svaku vrstu pitanja. Dinamična forma omogućava brzi unos velika količine pitanja. Svaka vrsta forme ima polje *tekst pitanja* u koje se unosi pitanje. Ovisno o formi, odabire se točan odgovor. U vrsti pitanja *točno/netočno* odabire se je li odgovor točan ili nije. Vrsta pitanja *popuni prazninu* omogućava proizvoljan broj prihvaćenih odgovora. Ukoliko je prilikom rješavanja ispita kao odgovor upisan jedan od odgovora iz polja *prihvaćeni odgovori,* odgovor je točan. Vrsta pitanja *točan odgovor* omogućava unos proizvoljnog broja ponuđenih odgovora, a jedan od njih se odabire za točan odgovor. Preostala vrsta pitanja je *više točnih odgovora.* Ta vrsta pitanja isto omogućava proizvoljan broj dostupnih odgovora, ali jednako tako i proizvoljan broj točnih odgovora.



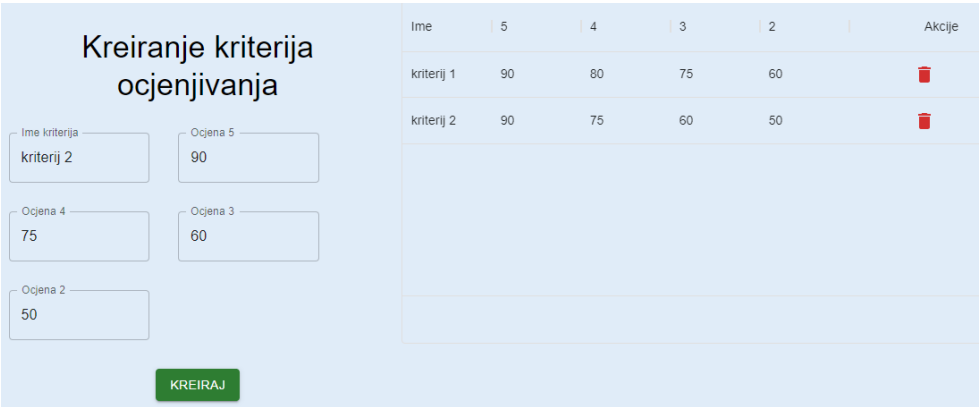
Slika 5.7. Prikaz forme za svaku vrstu pitanja.

Kreirana pitanja dostupna su u bazi pitanja. Pitanja se mogu sortirati po vrsti, a omogućeno je i njihovo uklanjanje te pregled. Slika 5.8 prikazuje izgled opisane baze pitanja.

Slika 5.8. Baza kreiranih pitanja.

5.3.5. Kriterij ocjenjivanja ispita

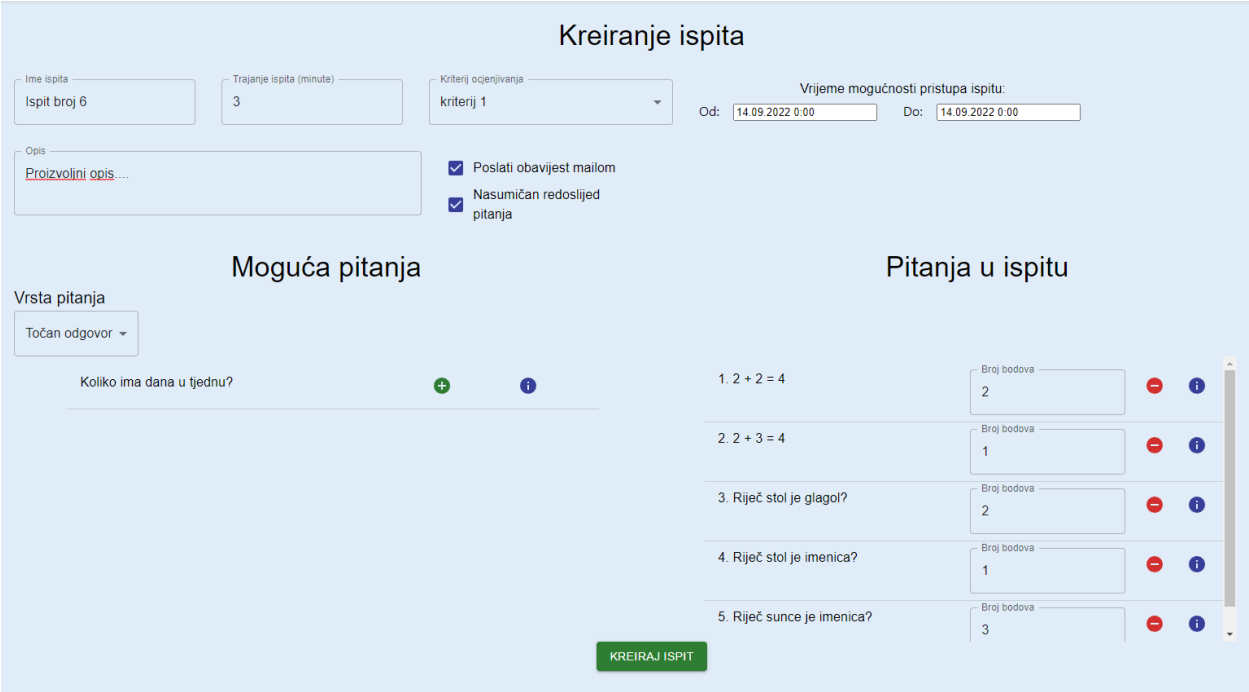
Prije kreiranja ispita, osim što je potrebno imati kreirana pitanja, potrebno je imati kreiran i kriterij ocjenjivanja. Na predmetu je moguće imati proizvoljan broj kriterija ocjenjivanja. Slika 5.9 prikazuje formu za kreiranje kriterija ocjenjivanja. Ona dozvoljava unos postotka bodova koji je potrebno ostvariti za određenu ocjenu. Granice koje se definiraju za svaku ocjenu su uključivog karaktera, odnosno ukoliko je ostvareni postotak bodova na ispitu jednak jednoj od granica, postotak je unutar granice.



Slika 5.9. Prikaz forme za svaku vrstu pitanja.

5.3.6. Kreiranje i baza ispita

Na slici 5.10 prikazana je forma za kreiranje ispita. Od tekstualnih polja potrebno je unijeti *ime ispita*, odrediti *trajanje ispita* u minutama te *opis*. U polju *kriterij ocjenjivanja* dostupni su svi kriteriji ocjenjivanja kreirani u predmetu. Konačna ocjena iz ispita određuje se ovisno o odabranome kriteriju. Postoje dva polja za odabir datuma. Prvo polje imena *od* označava datum i vrijeme u koje ispit postaje dostupan za rješavanje, a polje *do* označava datum i vrijeme kada ispit prestaje biti dostupan za rješavanje. Ispiti se kreiraju uz mogućnost odabira dvije opcije. Prva opcija upravlja time hoće li se poslati svim učenicima iz predmeta e-pošta koja ih obavještava o kreiranome ispitu. Druga opcija određuje hoće li redoslijed pitanja prilikom rješavanja biti nasumičan ili će se koristiti poredak kojim su pitanja poredana pri kreiranju ispita. Za mogućnost kreiranja ispita preostalo je dodati minimalno jedno pitanje. Sva dostupna pitanja prikazana su u lijevoj listi, a klikom na zeleni gumb pitanje se dodaju u desnu listu, odnosno u set pitanja za ispit. Za svako dodano pitanje moguće je odrediti koliko bodova to pitanje vrijedi na ispitu. Dodatno, moguće je manipulirati redoslijedom pitanja na *drag and drop* način korištenja miša u listi *pitanja u ispitu.*



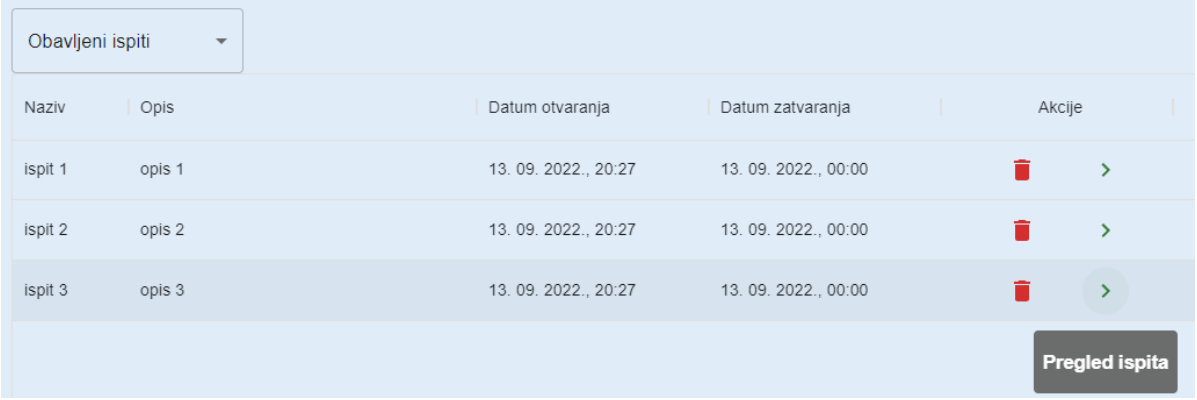
Slika 5.10. Forma za kreiranje ispita.

Kreirani ispiti dostupni su učenicima za rješavanje u vremenu koje je odabrano prilikom kreiranja ispita. Nakon što istekne rok za rješavanje ispita, više nije moguće pristupiti pisanju. Riješeni ispiti ocjenjuju se na poslužiteljskoj strani aplikacije. Nakon što su ispiti ocjenjeni, nastavnicima je

omogućeno dodatno ocjenjivanje svakog riješenog ispita. Ukoliko je nastavnik dodatno ispravio potrebne ispite ili nema potrebe za time, nastavnik zaključava riješene ispite. Zaključani ispiti više nisu dostupni za dodatno ocjenjivanje. Na taj način se sprječavaju moguće manipulacije rezultatima ispita.

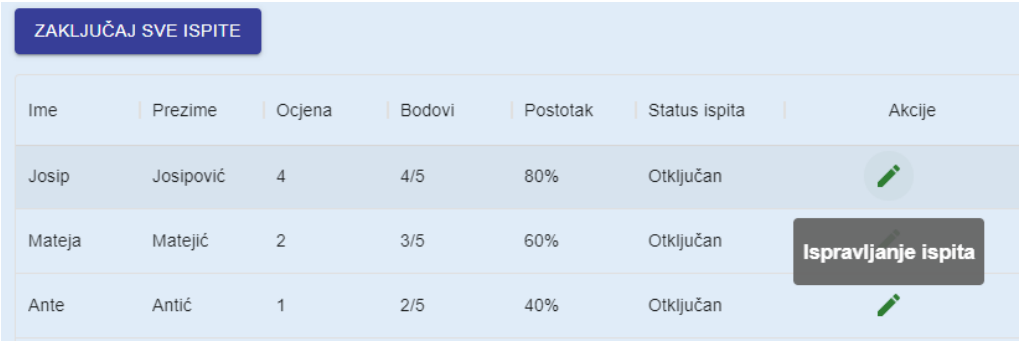
Na stranici za pregled ispita moguće je filtrirati ispite po njihovome statusu. Definirana su tri statusa, a to su: budući ispiti, otvoreni ispiti i riješeni ispiti. Budući ispiti su oni koji će tek biti dostupni za rješavanje. Otvoreni ispiti su trenutno dostupni za rješavanje. Riješeni ispiti predstavljaju one ispite koji više nisu dostupni za rješavanje. Ispiti u sva tri statusa se mogu obrisati.

Na slici 5.11 prikazan je pogled na obavljene ispite. Kod obavljenih ispita dodatno je dostupna akcija predstavljena strjelicom koja omogućava pregled ispita.



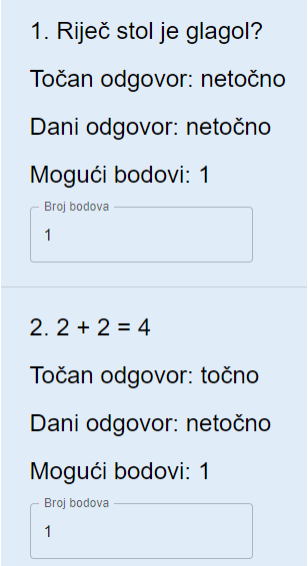
Slika 5.11. Prikaz obavljenih ispita.

Primjer stranice pregleda određenog ispita prikazan je na slici 5.12. U tablici su prikazani svi učenici koji su pristupili ispitu. Svaki ispit moguće je dodatno pregledati te mijenjati ostvarene bodove za svako pitanje. Na stranici je moguće i zaključati sve ispite.



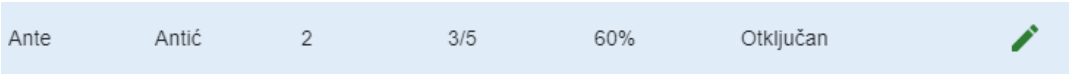
Slika 5.12. Stranica za pregled ispita.

Odabirom akcije za ispravljanje ispita odlazi se na stranicu za ispravak ispita. Slika 5.13 prikazuje ispravljanje ispita sa slike 5.12, a odabrani učenik je Ante Antić. Na slici su vidljiva dva pitanja sa ispita. Prvo pitanje je točno odgovoreno, a drugo netočno. Za potrebe testiranja, promijenjen je broj ostvarenih bodova na drugome pitanju. Nakon ispravljanja, spremljene su izmjene.



Slika 5.13. Primjer dodatnog ispravljanja ispita.

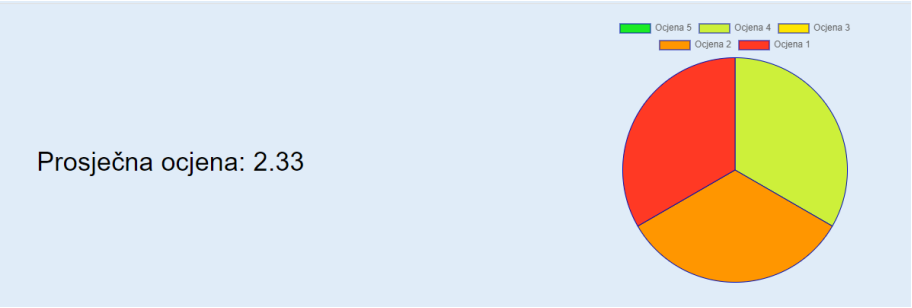
Nakon što su u prošlome ispitu izmijenjeni ostvareni bodovi na drugome pitanju, slika 5.14 prikazuje promjenu ostvarenih bodova učenika Ante Antića. Učeniku su se ostvareni bodovi povećali za jedan bod te mu je ocjena promijenjena. Nakon što je nastavnik siguran da su svi ispiti pravilno ispravljeni, potrebno je zaključati ispite i time se sprječava daljnje manipuliranje rezultatima ispita.



Slika 5.14. Primjer ažuriranja rezultata ispita.

5.3.7. Vrednovanje ispita

Nastavnicima su omogućena dva grafička prikaza rezultata ispita. Prvi grafički prikaz (slika 5.15) je kružni graf. On prikazuje udio svih ostvarenih ocjena odabranog ispita. Dodatno je prikazana i prosječna ocjena ispita.



Slika 5.15. Prikaz udjela ocjena nekog ispita.

Druga vrsta grafa je stupčasti graf. On omogućuje pregled ocjena nekog učenika po ispitima. Slika 5.16 prikazuje ocjene odabranog učenika na svim ispitima koje je pisao. Iznad grafa nalazi se učenikova prosječna ocjena ostvarena na predmetu.

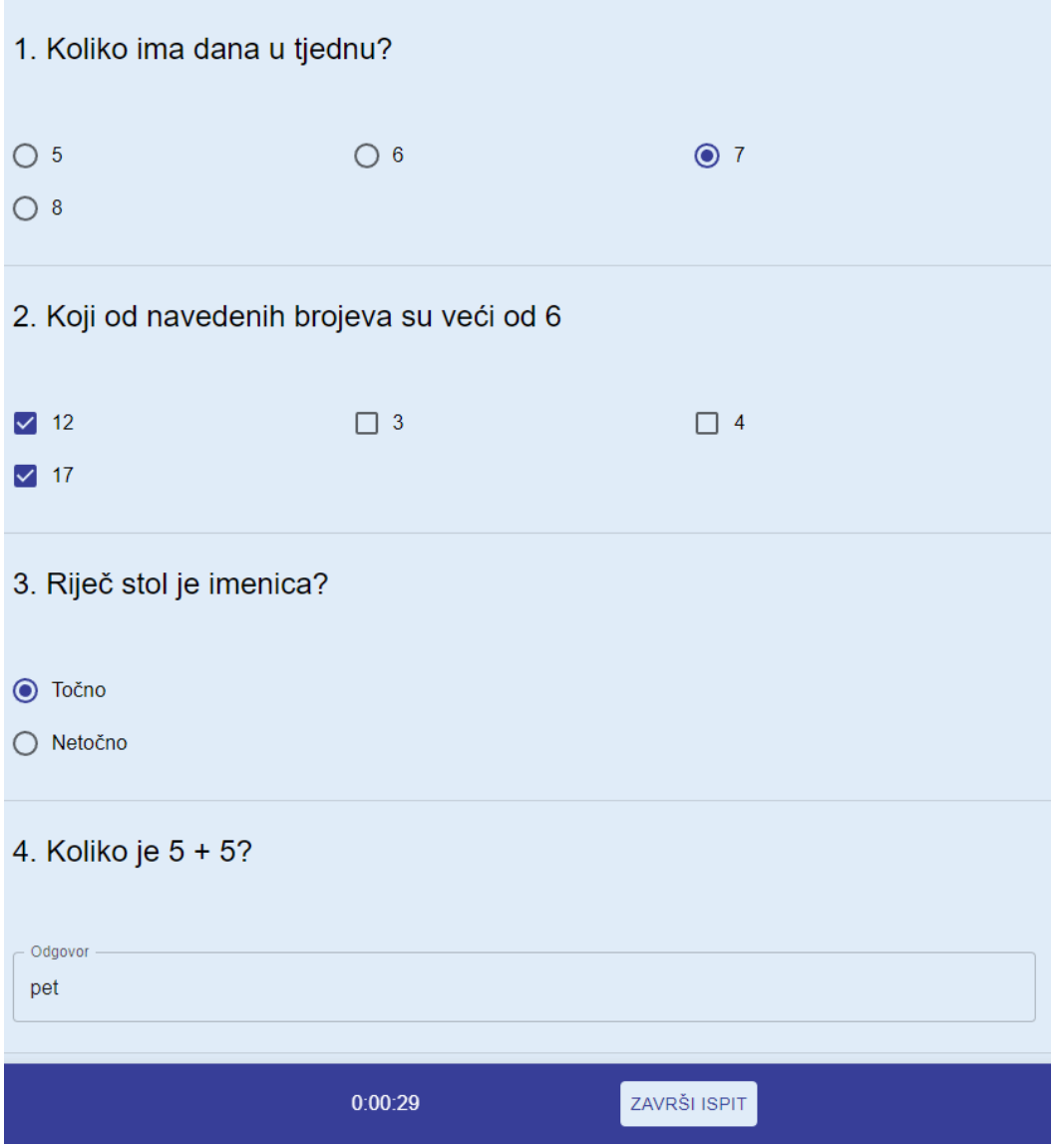
Slika 5.16. Prikaz ocjena odabranog učenika.

5.4. Uporaba aplikacije u ulozi učenika

Nakon registracije, učenik mora zatražiti pristup na potrebni predmet. Nakon što je učenik zatražio pristup, nastavnik toga predmeta mora ga prihvatiti u predmet. Učenik unutar predmeta ima mogućnosti pregleda obavijesti, rješavanja ispita, pregleda svojih ocjena te opciju napuštanja predmeta. Od tih mogućnosti, glavni cilj učenika je rješavanje ispita. Slika 5.17 prikazuje sučelje na kojemu učenik ima informacije o budućim i otvorenim ispitima. Učenik dostupni ispit započinje rješavati pritiskom na gumb ispod informacija o ispitu.

Slika 5.17. Učenikov prikaz sučelja ispita.

Nakon što je učenik pokrenuo rješavanje ispita, otvara mu se prozor za rješavanje. Slika 5.18 prikazuje primjer rješavanja ispita. Vrijeme pisanja ispita jednako je onome vremenu koje je nastavnik odredio pri kreiranju ispita. Pri dnu stranice prikazano je preostalo vrijeme pisanja, a pokraj se nalazi i tipka za predaju ispita. Ispit se može predati pritiskom na gumb ili automatskom predajom nakon isteka vremena. Na korisničkome sučelja svaka vrsta pitanja prikazana je u prikladnome formatu za rješavanje. Nakon što učenik riješi ispit, ostvarena ocjena vidljiva je na pregledu svih ocjena toga predmeta, gdje mu je prikazana i prosječna ocjena svih ispita.



Slika 5.18. Primjer rješavanja ispita od strane učenika.

5.5. Osvrt na rad aplikacije

Predstavljeno rješenje web aplikacije zadovoljava sve definirane funkcionalne zahtjeve. Budući da je aplikacija izrađena principom korištenja višeslojne arhitekture, lako je proširiva te se postojeće funkcionalnosti mogu prilagoditi. U kontekstu on-line rješavanja ispita, predstavljeno rješenje zadovoljava potrebe uspješne provjere znanja. Bitna stavka on-line provjere znanja je sigurnost tijekom pisanja, odnosno reguliranje varanja tijekom provjere znanja. Postoji mnogo predstavljenih i predloženih rješenja te problematike. Ukoliko se kreirano rješenje web aplikacije želi koristiti uz zadovoljavanje aspekta sigurnosti provedne on-line ispita, moguća je laka proširivost njenih funkcionalnost.

6. ZAKLJUČAK

U ovome diplomskome radu ostvareno je programsko rješenje web aplikacije zasnovane na višeslojnoj arhitekturi. Aplikacija predstavlja sustav za on-line rješavanje i ocjenjivanje ispita. Prilikom istraživanja teme, analizirana su slična postojeća rješenja. Uz pregled postojećih rješenja na razini programske podrške, istraženi su trendovi on-line učenja i evaluacije znanja. Analizirana su novija istraživanja koja se bave tom temom te je dan pregled prednosti i mana različitih pristupa. Podrobno je istražena tema višeslojne arhitekture u svrhu izrade modela prikladnog rješenja programske podrške. Nakon definiranja funkcionalnih i nefunkcionalnih zahtjeva na sustav izrađen je model programske podrške zasnovan na višeslojnoj arhitekturi.

Web aplikacija sastoji se od četiri sloja. Sloj baze podataka predstavlja *PostgreSQL* baza podataka. Sloj pristupa podacima izrađen je pomoću alata *Entity Framework Core.* Poslovni sloj aplikacije izrađen je u *ASP.NET Core* projektu. Poslovni sloj definira *API* koji nudi različite opcije korištenja sustava. Korisnici suradnju s *API*-jem ostvaruju korištenjem korisničkog sučelja aplikacije, odnosno prezentacijskog sloja. Prezentacijski sloj implementiran je u JavaScript programskome jeziku uz korištenje dostupnih biblioteka, od kojih je najznačajnija React biblioteka. Prikazan je tijek korištenja aplikacije kroz arhitekturne slojeve te je objašnjena njihova suradanja. Nakon prolaska kroz implementacijski dio, prikazano je korištenje programske podrške od strane korisnika. Objašnjene su njihove mogućnosti prilikom uporabe aplikacije. Analizirana je okolina u slučaju korištenja korisnika u različitim ulogama, odnosno u ulozi nastavnika te učenika.

LITERATURA

[1] M.E. Eltahir, N.R. Alsalhi, S.S. Al-Qatawneh, Implementation of E-exams during the COVID-19 pandemic: A quantitative study in higher education, PLOS ONE, 17(5), svibanj 2022.

[2] Y. Citriadin, Pros and Cons of Video for Learning Interest in Online Learning Amid Pandemic Covid-19 in University, Al-Ishlah: Jurnal Pendidikan, 14(1), pp. 335-342, travanj 2022.

[3] X.C. O'Dea, J. Stern, Virtually the same?: Online higher education in the post Covid-19 era, Br J Educ Technol., 53(3), svibanj 2022.

[4] S. Altam, Dr. G.F. Kokane, Validity of Online Exams in Indian Universities During COVID-19 Pandemic: A Case Study of M.A Students at Bamu University, UIJRT, 3(4), veljača 2022.

[5] Loomen, https://www.carnet.hr/usluga/loomen/ (5.9.2022.)

[6] Moodle, https://moodle.srce.hr/2021-2022/ (5.9.2022.)

[7] I. Bosanić, I. Čavrak, Raspodjeljeni sustavi 1, Otvoreno računarstvo, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2021./2022. (creative commons licence)

[8] Packt Editorial Staff, What is a Multi Layered Software Architecture?, Packt Publishing, svibanj 2018.

[9] M. Richards, Software Architecture Patterns, O'Reilly Media, Inc., Sebastopol, 2015.

[10] L. Chung, B.A. Nixon, E.Yu, J. Mylopoulos, Non-Functional Requirements in Software Engineering, Springer US, New York, 2000.

[11] M.J. Price, C# 10 and .NET 6 – Modern Cross-Platform Development, Packt Publishing, Birmingham, 2021.

[12] D. Roth, What's New in ASP.NET Core in .NET 6, CODE Focus Magazine, 18(1), studeni 2021.

[13] R. Peres, Entity Framework Core Cookbook, Packt Publishing, Birmingham, 2016.

[14] S. Saini, Entity Framework Code First vs Database First vs Model First Approach, Pro Code Guide, siječanj 2022.

[15] R. Tomar, S. Dangi, JAVASCRIPT Syntax and Practices, CRC Press, Boca Raton, 2022.

[16] A. Alazab, A. Khraisat, M. Alazab, S. Singh, Detection of Obfuscated Malicious JavaScript Code, Future Internet, 14(217), srpanj 2022.

[17] N. Barbettini, The little ASP.NET Core book, Creative Commons license, 2018.

[18] H. Talari, What are the features of ReactJS?, GeeksfoorGeeks, listopad, 2021.

[19] R. Vyas, Comparative Analysis on Front-End Frameworks for Web Applications, IJRASET, 10(7), srpanj, 2022.

[20] D. Pronina, I. Kyrychenko, Comparison of Redux and React Hooks Methods in Terms of Performance, COLINS-2022: 6th International Conference on Computational Linguistics and Intelligent Systems, 3171, svibanj, 2022.

[21] B. Dinkevich, I. Gelman, The Complete Redux Book - Everything You Need To Build Real Projects With Redux, Lean Publishing, New York, 2017.

[22] The PostgreSQL Global Development Group, PostgreSQL 7.3.2 Tutorial, University of California, Berkeley, Computer Science Department, Berkeley, 2002.

[23] M.N. Jeyarajm S. Sucharitharathna, C. Senarath, Y. Kanagaraj, I. Udayakumara, Cognitive Visual-learning Environment for PostgreSQL, arXiv, svibanj, 2022.

SAŽETAK

Teorijski dio rada obuhvaća izazove u održavanju on-line nastave te on-line ispita. Prikazuje i postojeća programska rješenja za navedenu tematiku, sukladno čemu je razvijena web aplikacija za održavanje on-line ispita kao zadatak ovog diplomskog rada. Model izrađene aplikacije izrađen je principom višeslojne arhitekture. Aplikacija je implementirana korištenjem PostgreSQL baze podataka, razvojnog okruženja ASP.NET Core te JavaScript biblioteke React. Na primjerima korištenja objašnjeni su ključni dijelovi implementacije. Nakon prolaska kroz implementaciju, prikazan je i analiziran rad aplikacije na strani krajnjeg korisnika.

Ključne riječi: ASP.NET Core, React, višeslojna arhitektura, web aplikacija.

ABSTRACT

Web application for online exam grading using multilayered architecture.

The theoretical part of the thesis explains challenges in online teaching and examination. It explains existing software solutions for the said topic. This thesis proposes a solution in a form of the web application made accordingly to the covered topics. The application is made on the priciples of multilayered architecture. It is implemented by using PostgreSQL database, ASP.NET Core framework and React JavaScript library. The key parts of the implementation are explained with use cases. Ultimately, the application is shown and analyized in the hands of the end user.

Ključne riječi: ASP.NET Core, React, multilayered architecture, web application.

ŽIVOTOPIS

Ivan Paradžik rođen je u Đakovu 23. siječnja 1999. godine. Završio je Gimnaziju Antuna Gustava Matoša Đakovo, smjer opća gimnazija, nakon koje je upisao sveučilišni preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku 2017. godine. Godine 2020. upisuje sveučilišni diplomski studij računarstva, smjer informacijske i podatkovne znanosti, na istom fakultetu, kojeg završava 2022. godine. Stručnu praksu odradio je u osječkoj IT tvrtki MONO te je tijekom studiranja na diplomskom studiju bio zaposlen kao web developer u navedenoj tvrtki.

PRILOZI

Prilog 1. Diplomski rad u datoteci .docx

Prilog 2. Diplomski rad u datoteci .pdf

Prilog 3. Programski kod aplikacije