EduCoder - aplikacija za evaluaciju studentskih vještina u programiranju

Žužić, Alesandro

SVEUČILIŠTE JURJA DOBRILE U PULI
FAKULTET INFORMATIKE

Alesandro Žužić

EDUCODER - APLIKACIJA ZA EVALUACIJU STUDENTSKIH VJEŠTINA U PROGRAMIRANJU

DIPLOMSKI RAD

Pula, srpanj, 2024. godine

SVEUČILIŠTE JURJA DOBRILE U PULI
FAKULTET INFORMATIKE

Alesandro Žužić

EDUCODER - APLIKACIJA ZA EVALUACIJU STUDENTSKIH VJEŠTINA U PROGRAMIRANJU

DIPLOMSKI RAD

JMBAG: 0303088161, redoviti student
Studijski smjer: Informatika

Kolegij: Razvoj IT rješenja
Znanstveno područje : Društvene znanosti
Znanstveno polje : Informacijske i komunikacijske znanosti
Znanstvena grana : Informacijski sustavi i informatologija

Mentor: Nikola Tanković

Pula, srpanj, 2024. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Alesandro Žužić, kandidat za magistra Informatike ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

U Puli, 09.07.2024.

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, ALESANDRO ŽUŽIĆ dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom EDUCODER - APLIKACIJA ZA EVALUACIJU STUDENTSKIH VJEŠTINA U PROGRAMIRANJU koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 09.07.2024.

Potpis

AŽ

Sadržaj

1 Uvod **1**

2 Motivacija **2**

2.1 Općenito . . . 2

2.2 SWOT analiza . . . 2

3 Model sustava **4**

3.1 Model slučaja . . . 4

3.1.1 Model slučaja za korisnika . . . 6

3.1.2 Model slučaja za administratora . . . 7

3.2 Model slijeda . . . 7

3.2.1 Model slijeda za korisnika . . . 9

3.2.2 Model slijeda za administratora . . . 14

3.3 Klasni model . . . 23

3.3.1 User . . . 26

3.3.2 Data . . . 27

3.3.3 Exam . . . 29

3.3.4 Rješenja . . . 33

4 Implementacija **35**

4.1 Korištene metode . . . 35

4.1.1 Vue.js . . . 35

4.1.2 Vite.js . . . 38

4.1.3 Node.js . . . 39

4.1.4 Electron . . . 41

4.1.5 Firebase . . . 42

4.2 Arhitektura sustava . . . 44

4.2.1 Struktura aplikacije . . . 44

4.2.2 Korišteni npm paketi . . . 47

4.2.3 Web aplikacija . . . 48

4.2.4 Desktop aplikacija . . . 51

4.3 Sučelje aplikacije . . . 54

4.3.1 Korisničko sučelje . . . 57

4.3.2 Administrativno sučelje . . . 62

4.4 Implementacija ključnih funkcionalnosti . . . 71

4.4.1 Evaluacija kôda . . . 71

4.4.2 Započinjanje i vraćanje u ispit . . . 72

4.4.3 Sinkronizacija ažuriranja podataka . . . 73

4.4.4 Optimiziranje dohvata iz baze podataka 73

5 Primjena u praksi **75**
5.1 Studija slučaja: Primjena EduCoder aplikacije u nastavi 75
5.2 Analiza rezultata i korisničkog iskustva 76
5.2.1 Anketa 77
5.2.2 Analiza rezultata ankete 80

6 Zaključak **85**
6.1 Daljnja istraživanja 86
6.2 Zaključne misli 87

Literatura **88**

Popis slika **93**

7 Sažetak **95**

8 Abstract **96**

9 Prilog **97**

1 Uvod

S razvojem tehnologije i digitalnih alata, sposobnost programiranja postala je jedna od ključnih vještina potrebnih za uspjeh u raznim profesionalnim područjima. Bez obzira na to radi li se o inženjeringu, znanosti, financijama ili kreativnim industrijama, programiranje se sve više pojavljuje kao osnovna vještina. Učinkovita evaluacija programskih vještina postala je stoga neophodna kako bi se osigurala adekvatna priprema studenata za izazove suvremenog tržišta rada i njihov daljnji napredak.

Trenutno obrazovne institucije traže načine kako poboljšati proces učenja i evaluacije programiranja. Tradicionalni načini testiranja često ne mogu u potpunosti obuhvatiti praktične aspekte programiranja niti pružiti dovoljno povratnih informacija studentima. U tom kontekstu, aplikacija EduCoder razvijena je s ciljem unapređenja procesa evaluacije studentskih vještina u programiranju.

EduCoder pruža interaktivno okruženje za vježbanje i polaganje ispita iz programiranja, koristeći HTML, CSS i JavaScript jezike. Aplikacija omogućuje studentima da pišu i testiraju svoj kôd u stvarnom vremenu, dok istovremeno pruža mogućnost vježbanja predefiniranih zadataka i pregleda skripti. Ova platforma ne samo da poboljšava iskustvo učenja, nego i omogućava nastavnicima da učinkovitije prate napredak svojih studenata.

U ovom radu analizirat ćemo implementaciju aplikacije EduCoder, njenu arhitekturu i ključne funkcionalnosti koje omogućuju njezinu primjenu u obrazovnom kontekstu. Također ćemo istražiti kako je aplikacija integrirana u nastavni proces na Fakultetu informatike u Puli, te ćemo prikazati rezultate i povratne informacije prikupljene od korisnika putem anketa. Na temelju prikupljenih podataka, identificirat ćemo prednosti i nedostatke aplikacije te predložiti moguća poboljšanja za buduće verzije.

2 Motivacija

2.1 Općenito

U današnjem digitalnom dobu, gdje tehnološke inovacije oblikuju način na koji učimo i radimo, evaluacija vještina u programiranju postaje ključna za pripremu studenata za izazove suvremenog tržišta rada. Stoga je važno osigurati da naši studenti ne samo steknu teorijsko znanje, već i praktične vještine u programiranju koje su ključne za uspješnu karijeru u informacijskoj tehnologiji.

Međutim, trenutni pristupi evaluaciji vještina u programiranju suočavaju se s brojnim izazovima. Postojeći alati poput Verifikatora, iako su korisni, često su ograničeni svojom funkcionalnošću, posebno u pogledu podržanih programskih jezika i operativnih sustava. Primjerice, Verifikator [1], koji nije dostupan na Linuxu ili macOS operativnim sustavima, često se percipira kao neugodan, problematičan i nepouzdan alat. Ova ograničenja stvaraju prepreke za učenje i ocjenjivanje programerskih vještina, posebno za one koji preferiraju ili su prisiljeni koristiti alternativne operativne sustave.

Osim toga, s porastom dostupnosti tehnologija umjetne inteligencije (AI), suočavamo se s novim izazovima u sprečavanju prepisivanja i osiguravanju integriteta evaluacije. Online nastava, koja je postala norma u današnjem svijetu, dodatno otežava nadzor studenata tijekom ispita, čime se otvara prostor za nepravilnosti poput prepisivanja s pomoću AI alata.

Da bi se riješili ovi izazovi i osiguralo kvalitetno ocjenjivanje vještina u programiranju, potrebno je razviti nove alate i pristupe evaluaciji. Ti alati trebaju ići dalje od jednostavne provjere ispravnosti kôda i pružiti sveobuhvatno iskustvo učenja i ocjenjivanja. Osim toga, važno je osigurati dostupnost materijala za vježbu i samoevaluaciju, poput skripti i zadataka za vježbu, kako bi se studentima pružila prilika za kontinuirani razvoj svojih vještina. Kroz ovaj pristup možemo osigurati da naši studenti ne samo steknu, već i održe i unaprijede svoje vještine u programiranju, pripremajući ih za uspješnu karijeru u digitalnom dobu.

2.2 SWOT analiza

Uz pomoć SWOT analize prikazane u tablici 1 moguće je analizirati snage, slabosti, prilike i prijetnje aplikacije.

Snage EduCoder aplikacije obuhvaćaju intuitivno korisničko sučelje, što olakšava korištenje i navigaciju, te raznolikost zadataka koja omogućuje studentima razvoj vještina u različitim programskim jezicima i konceptima. Također, aplikacija je prilagođena online okruženju i nudi fleksibilnost platforme, s mogućnošću pristupa putem weba ili desktop aplikacije.

Slabosti EduCoder aplikacije uključuju ovisnost o internetskoj vezi, što može ograničiti njenu funkcionalnost u uvjetima nestabilne ili spore internetske veze. Nedostatak naprednijih značajki poput automatskog ocjenjivanja ili personaliziranih preporuka također se može smatrati slabošću.

Prilike za EduCoder aplikaciju uključuju povećanu potražnju za online obrazovanjem, što može omogućiti širenje korisničke baze i pružanje dodatnih usluga. Partnerstva s obrazovnim institucijama mogu pružiti nove mogućnosti za integraciju te poboljšanje funkcionalnosti aplikacije, uključujući podršku za nove jezike i značajke, što predstavlja priliku za rast.

Prijetnje za EduCoder aplikaciju uključuju brze tehnološke promjene i evoluciju programskih jezika koje mogu zahtijevati kontinuirano prilagođavanje aplikacije. Također, sigurnosni rizici poput hakiranja ili krađe podataka predstavljaju prijetnju povjerenju korisnika i reputaciji aplikacije.

Kategorija	**Opis**
Snage	• Intuitivno korisničko sučelje • Prilagođeno online okruženje • Fleksibilnost platforme
Slabosti	• Ovisnost o internetskoj vezi • Nedostatak naprednijih značajki
Prilike	• Povećana potražnja za online obrazovanjem • Partnerstva s obrazovnim institucijama • Poboljšanje funkcionalnosti
Prijetnje	• Tehnološke promjene • Sigurnosni rizici

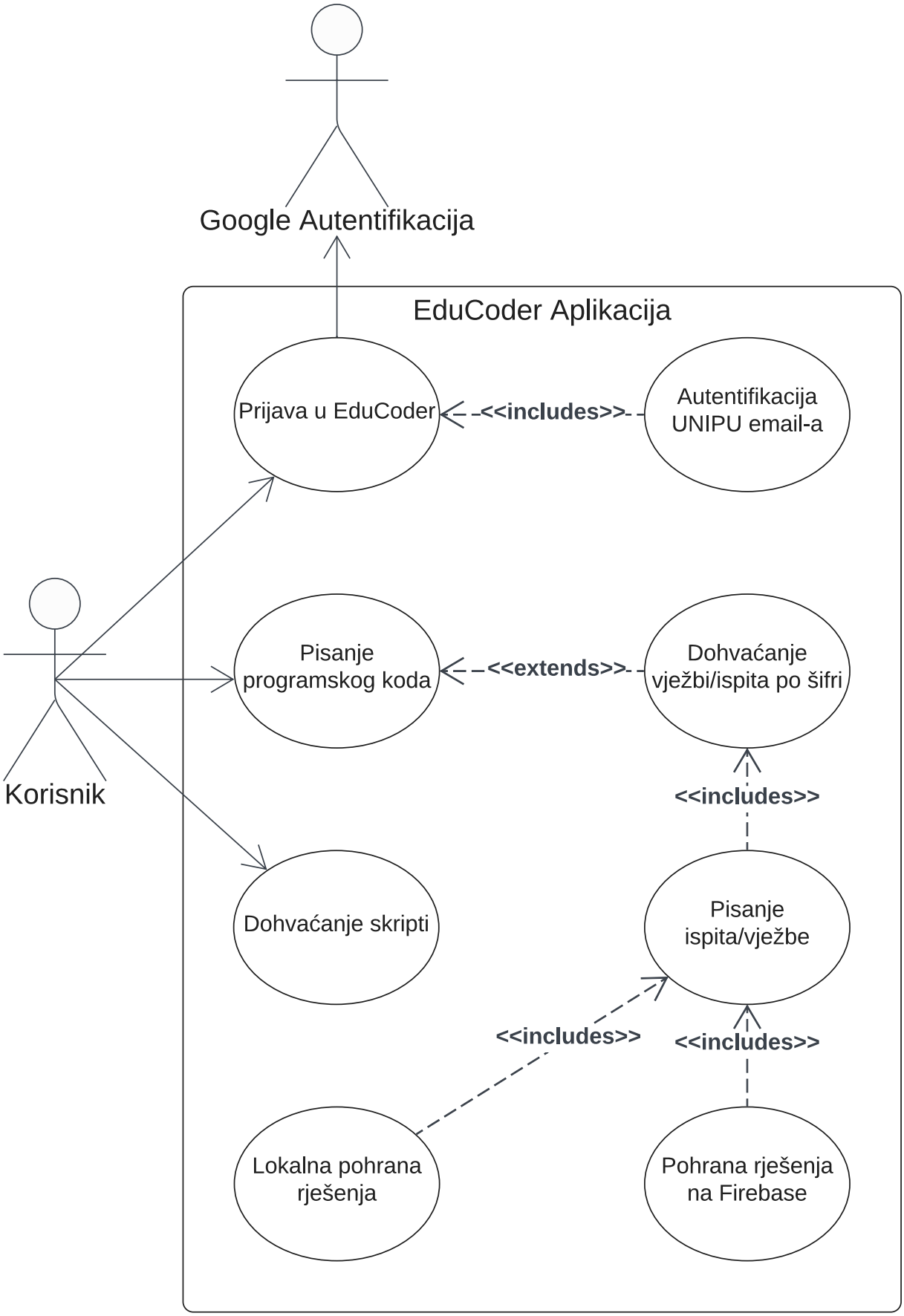
Tablica 1: SWOT analiza za EduCoder aplikaciju

3 Model sustava

U ovoj sekciji detaljno je opisan model sustava EduCoder aplikacije. Model sustava pruža strukturirani prikaz funkcionalnosti i arhitekture aplikacije, što omogućuje jasno razumijevanje njenog dizajna i implementacije. Glavni dijelovi modela obuhvaćaju model slučaja, koji identificira korisničke interakcije s aplikacijom, model slijeda, koji prikazuje redoslijed operacija unutar sustava, te klasni model, koji opisuje strukturu i odnose između različitih klasa unutar aplikacije.

3.1 Model slučaja

Model slučaja (*eng. use case*) pruža vizualni prikaz funkcionalnosti sustava i interakcije između korisnika (*eng. actor*) i sustava. Koristi se za modeliranje dinamičke strukture aplikacije i prikazuje različite funkcionalnosti koje korisnici mogu izvoditi unutar sustava.

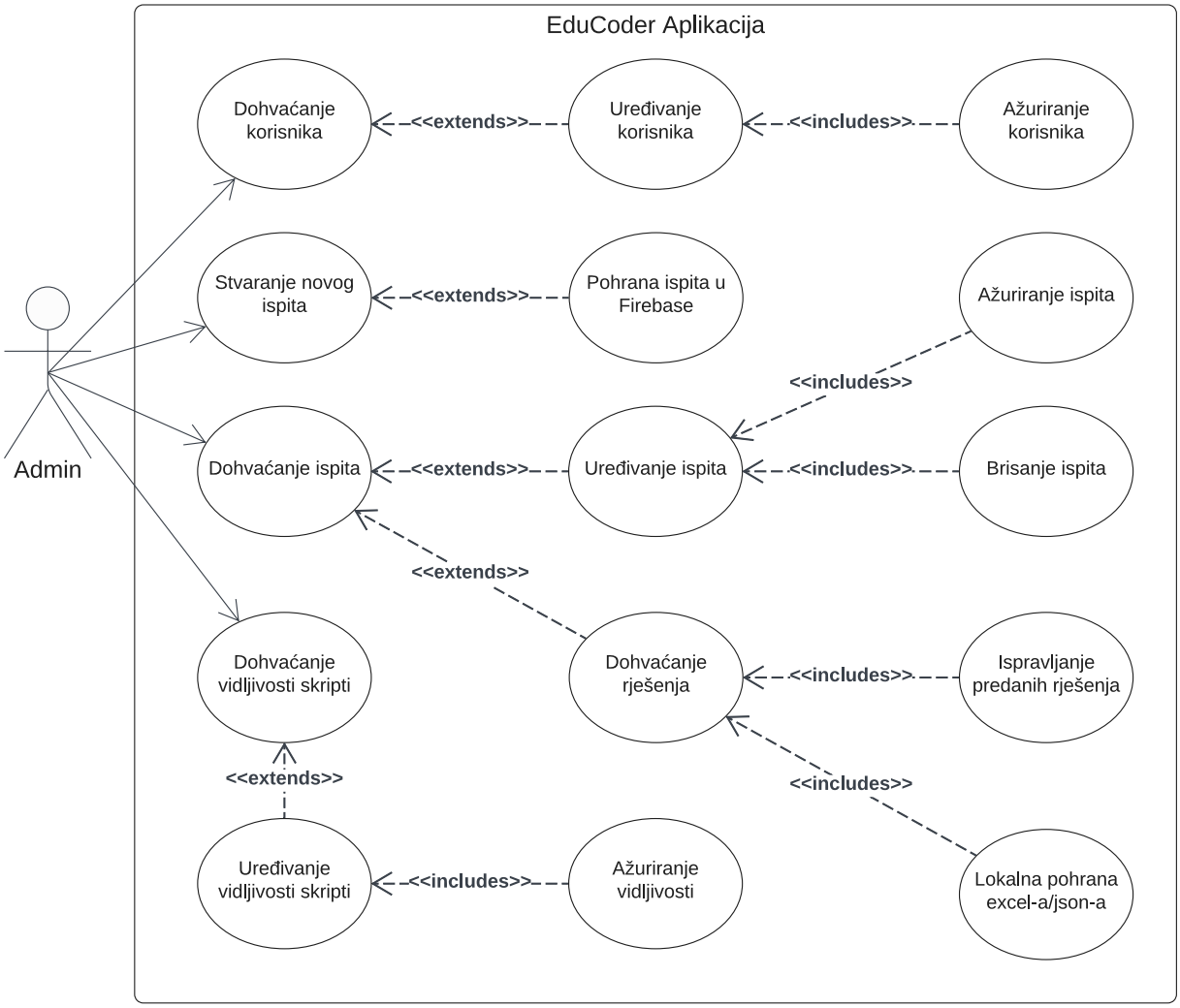


Slika 1: *Model slučaja za korisnike unutar EduCoder aplikacije (izvor: Autor)*

3.1.1 Model slučaja za korisnika

Model slučaja za korisnika na slici 1 prikazuje funkcionalnosti dostupne korisniku unutar EduCoder aplikacije. Korisnik može koristiti aplikaciju nakon autentifikacije putem Google računa koristeći samo UNIPU email. Nakon uspješne prijave, korisnik ima sljedeće funkcionalnosti:

- **Pisanje programskog kôda**
  Korisnik može pisati programski kôd unutar aplikacije. Ova funkcionalnost se proširuje na dohvaćanje vježbi ili ispita prema šifri što onda uključuje pisanje ispita/vježbi te lokalnu pohranu i pohranu rješenja na Firebase.
- **Dohvaćanje skripti**
  Korisnik može dohvaćati skripte koje su dostupne unutar aplikacije.



Slika 2: *Model slučaja za administratore unutar EduCoder aplikacije (izvor: Autor)*

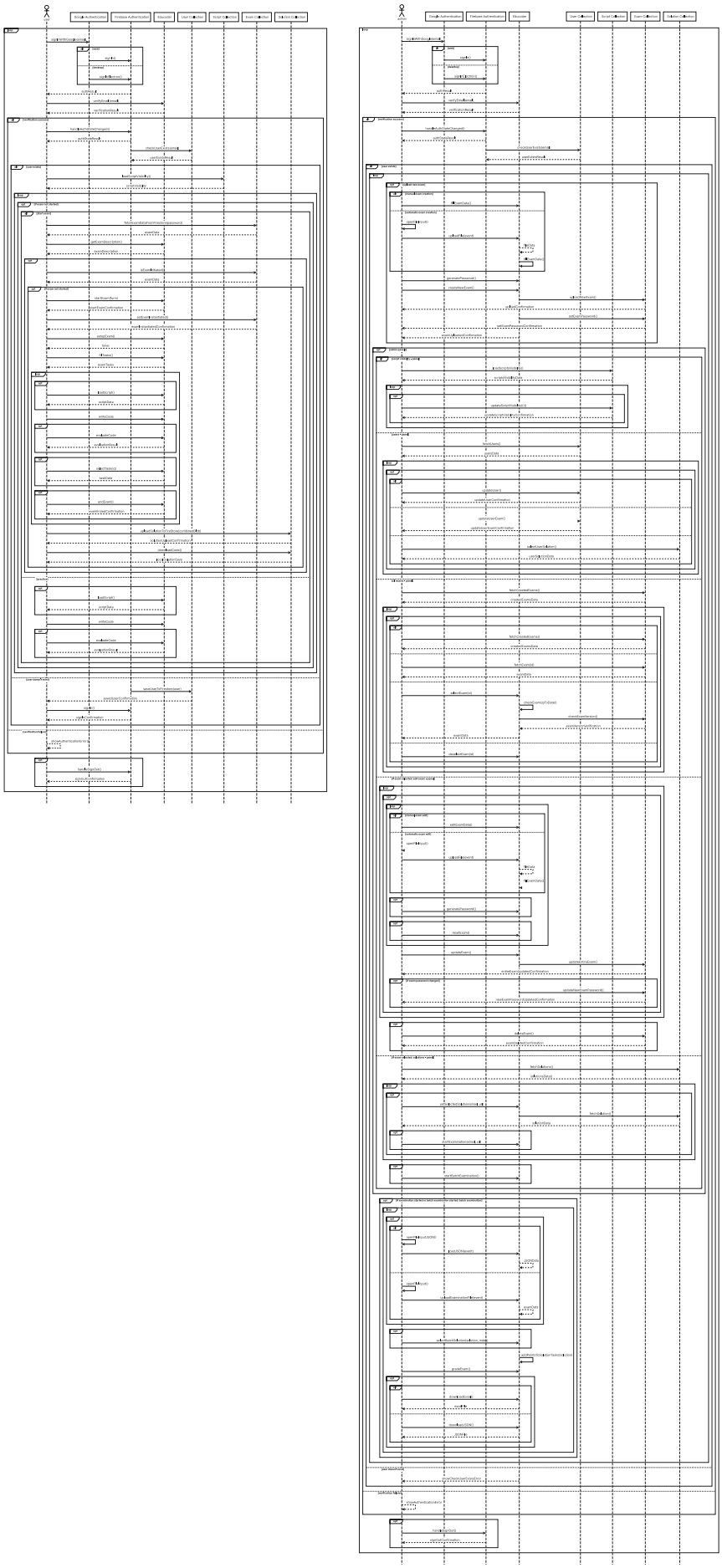
3.1.2 Model slučaja za administratora

Model slučaja za administratora na slici 2 prikazuje funkcionalnosti dostupne administratoru unutar EduCoder aplikacije. Administrator može koristiti aplikaciju za upravljanje korisnicima, ispitima i skriptama. Sljedeće funkcionalnosti su dostupne administratoru:

- **Dohvaćanje korisnika**
  Administrator može dohvaćati informacije o korisnicima. Ova funkcionalnost se proširuje na uređivanje korisnika i ažuriranje korisnika.
- **Stvaranje novog ispita**
  Administrator može kreirati nove ispite. Ova funkcionalnost se proširuje na pohranu ispita u Firebase i uređivanje ispita.
- **Dohvaćanje ispita**
  Administrator može dohvaćati informacije o ispitima. Ova funkcionalnost se proširuje na uređivanje ispita, ažuriranje ispita i brisanje ispita.
- **Dohvaćanje vidljivosti skripti**
  Administrator može dohvaćati vidljivost skripti. Ova funkcionalnost uključuje uređivanje vidljivosti skripti i ažuriranje vidljivosti.
- **Dohvaćanje rješenja**
  Administrator može dohvaćati predana rješenja. Ova funkcionalnost se proširuje na ispravljanje predanih rješenja i lokalno pohranjivanje podataka u Excel/JSON formatu.

3.2 Model slijeda

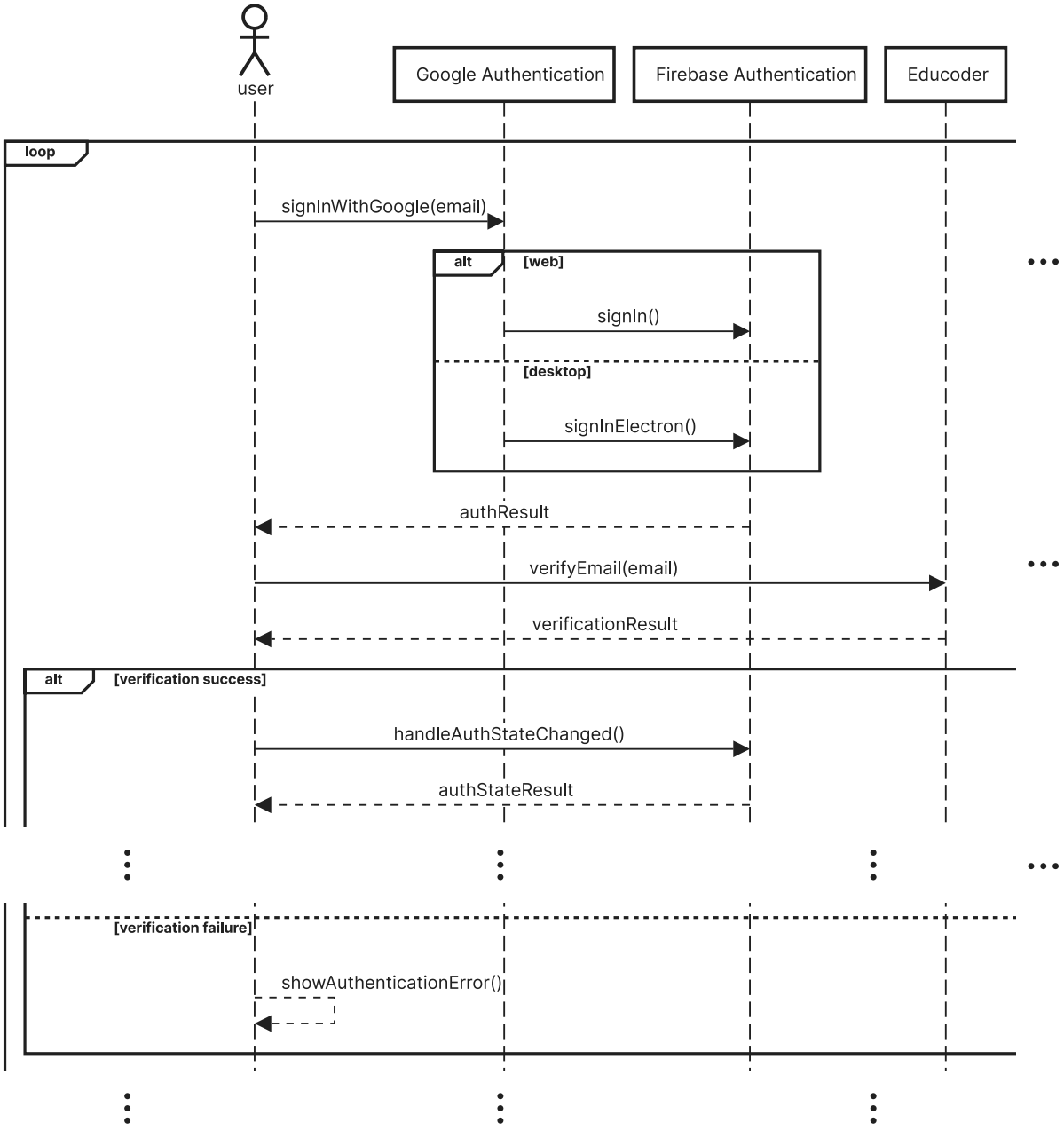
Model slijeda (*eng. sequence dijagram*) pruža detaljan prikaz redoslijeda operacija i komunikacije između različitih dijelova sustava. Koristi se za ilustraciju dinamičkog ponašanja sustava u određenim scenarijima, čime se olakšava razumijevanje toka izvršavanja operacija unutar aplikacije. Na slici 3 prikazani su cijeli modeli slijeda za korisnika i administratora.



Slika 3: *Modeli slijeda korisnika i administratora unutar EduCoder aplikacije (izvor: Autor)*

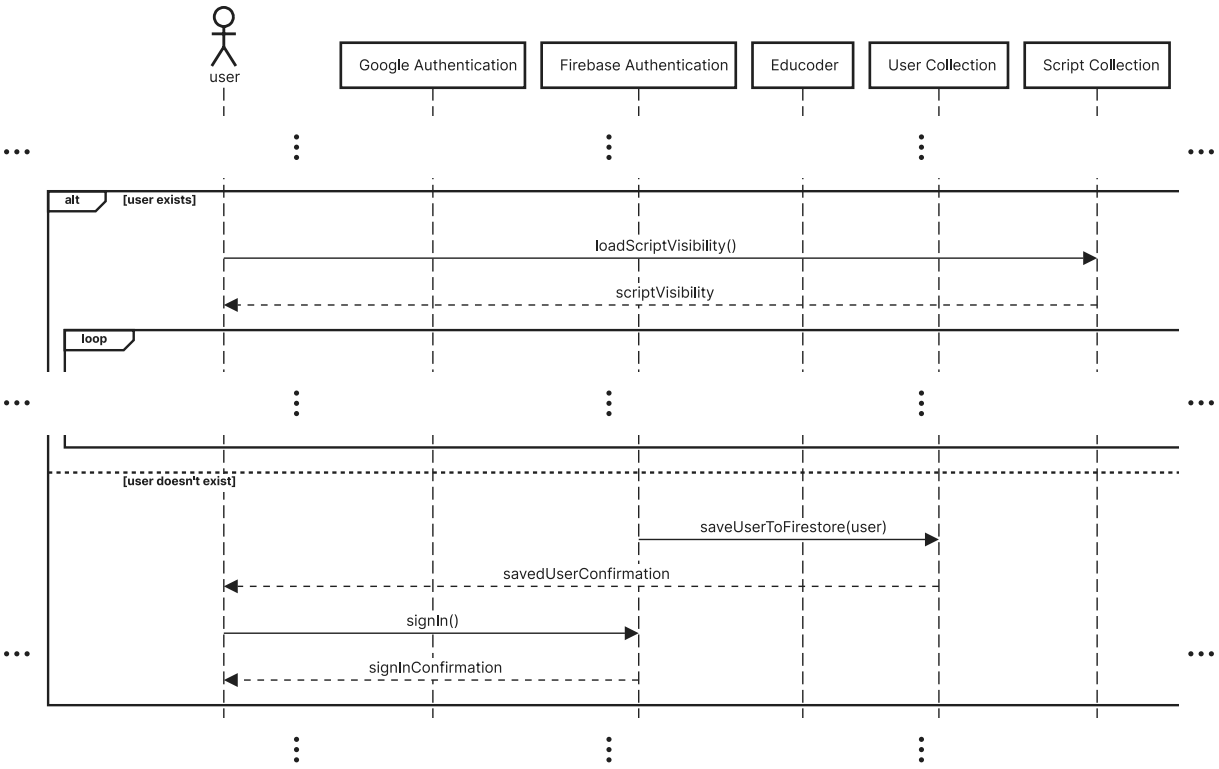
3.2.1 Model slijeda za korisnika

Model slijeda za korisnika u EduCoder aplikaciji započinje kada korisnik želi pristupiti sustavu i prijaviti se putem Google autentifikacije. Nakon odabira svog Google računa, korisnikov zahtjev šalje se Firebase autentifikacijskom sustavu, ovisno o tome pokušava li se korisnik prijaviti putem web ili desktop aplikacije. Nakon prijave, EduCoder provjerava identitet korisnika, odnosno valjanost email adrese u sklopu fakulteta, pri čemu email mora imati nastavak ”student.unipu.hr” ili ”unipu.hr”. Nakon uspješne validacije email adrese, korisnik prima potvrdu o validaciji; u suprotnom je obaviješten o neuspjeloj autentifikaciji (vidi sliku 4).



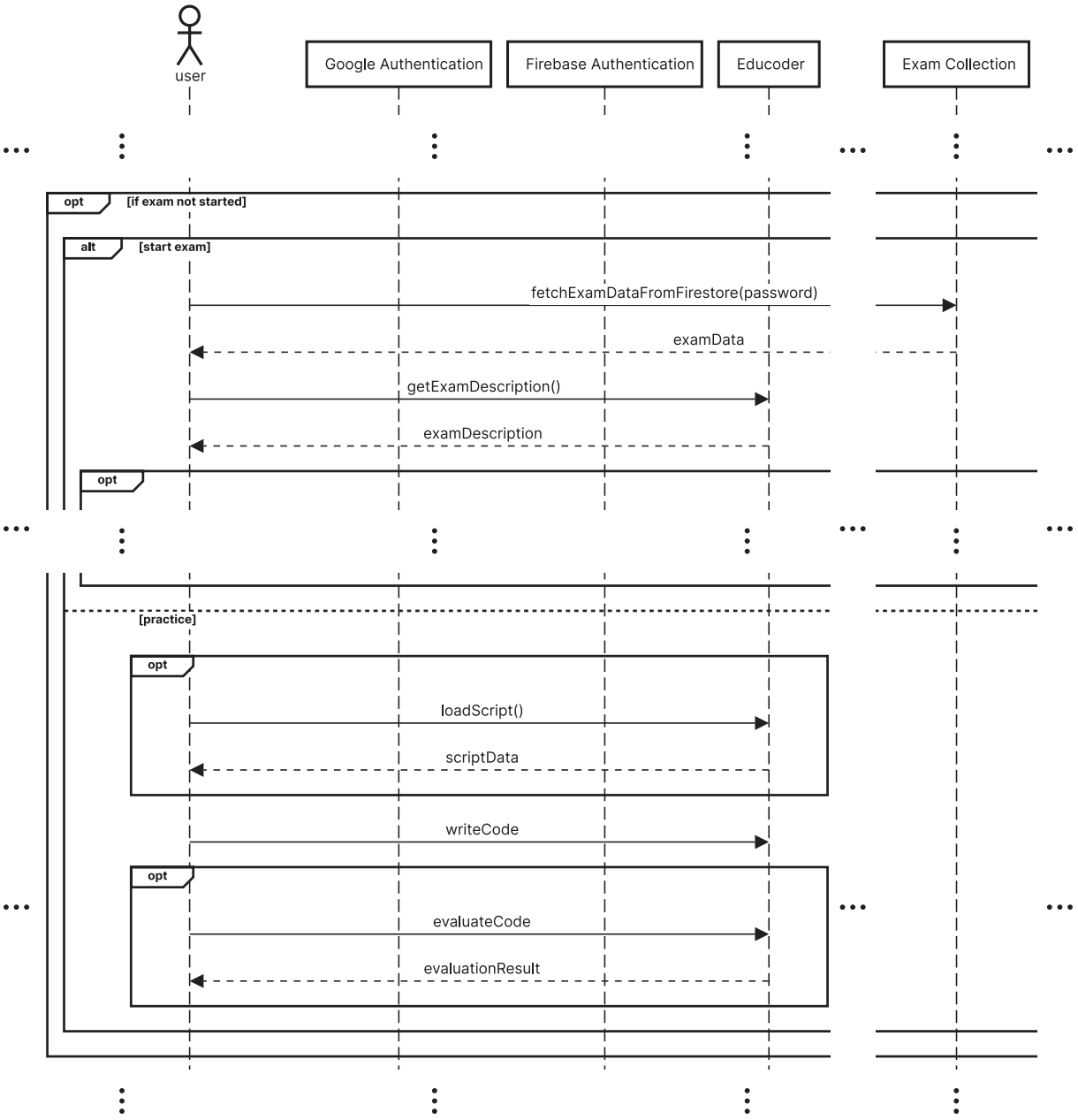
Slika 4: *Model slijeda korisnika - prijava i autentifikacija korisnika (izvor: Autor)*

Ako je verifikacija uspješna, provjerava se nalazi li se korisnikov email u kolekciji korisnika. Ako email nije pronađen, korisnik se sprema u kolekciju korisnika te prima potvrdu o uspješnom spremanju. Nakon toga, korisnik se ponovno prijavljuje u aplikaciju te mu se vraća obavijest o uspješnoj prijavi. Ako korisnik već postoji u bazi, ulazi u aplikaciju gdje se prvo učitava vidljivost skripti, čime se korisniku prikazuje vidljivost skripte (vidi sliku 5).



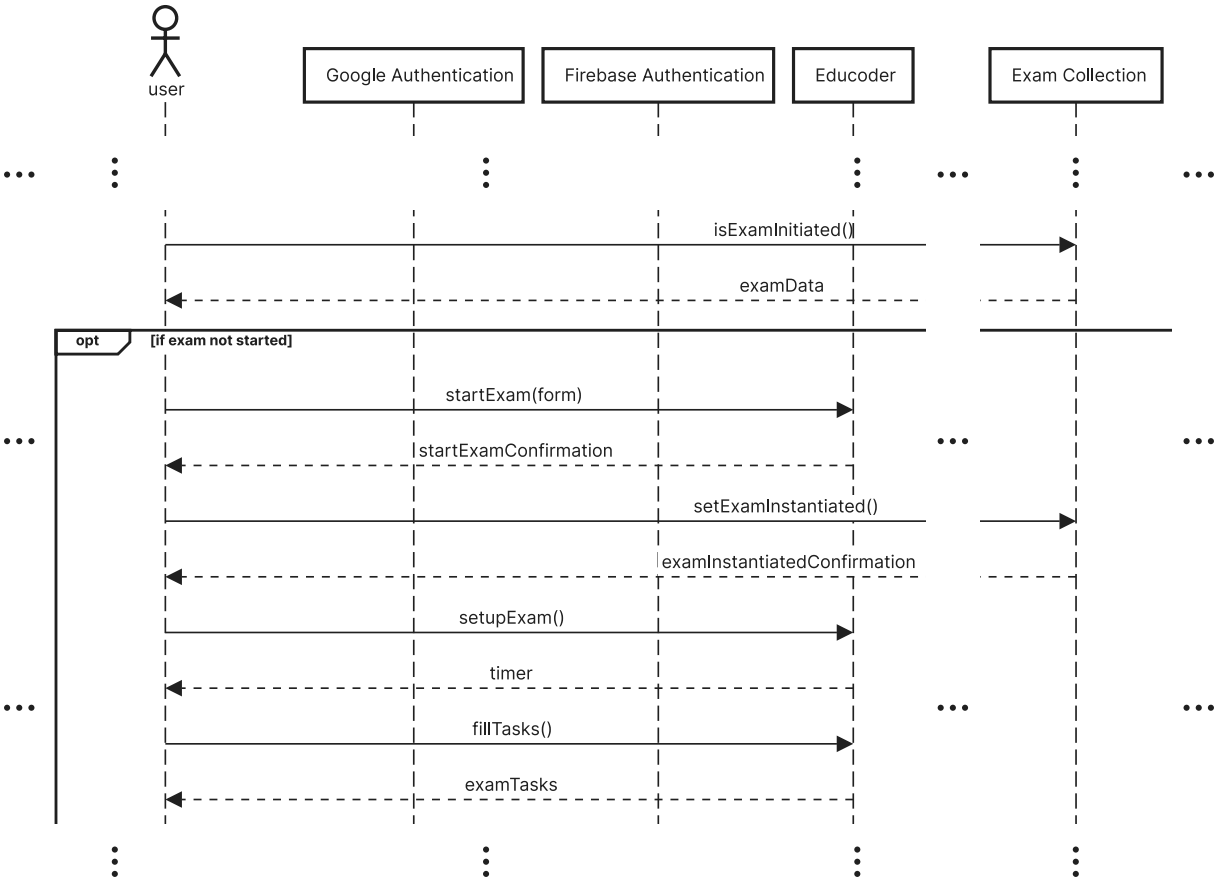
Slika 5: *Model slijeda korisnika - spremanje korisnika i dohvaćanje vidljivosti skripti (izvor: Autor)*

Ovisno o tome je li ispit započet ili nije, korisnik ima sljedeće opcije. Ako ispit nije započet, može vježbati u Sandbox okruženju, gdje može pregledavati vidljive skripte iz aplikacije, pisati kôd te isti evaluirati, čime dobiva rezultat izvršenja kôda. Također, student može umjesto vježbanja započeti ispit tako da unese odgovarajuću šifru i prima natrag podatke o ispitu. Aplikacija iz tih podataka izvlači opis ispita poput naziva, trajanja, početka, kraja, vrste i kratkog opisa ispita (vidi sliku 6).



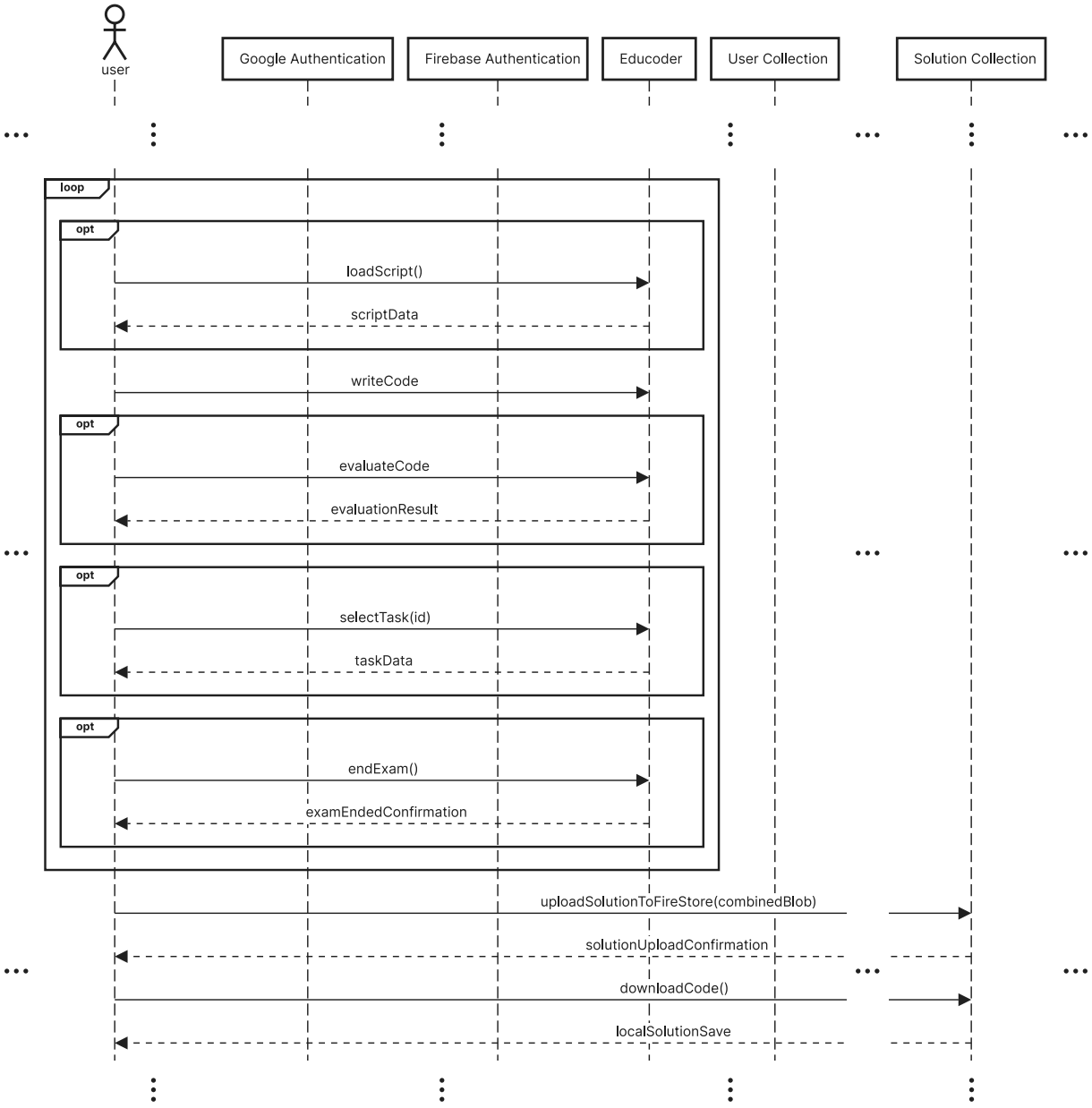
Slika 6: *Model slijeda korisnika - sandbox i dohvaćanje ispita (izvor: Autor)*

Prilikom započinjanja ispita, provjerava se je li ispit već započet. Ako je, korisnika se na to upozorava. U suprotnom, ispit započinje, šalje se popunjena korisnička forma, a korisnik dobiva obavijest o započetom ispitu. Ispit se istovremeno ažurira i postavlja mu se zastavica da je započet, što onemogućava istovremeno započinjanje dva ispita ili ponovno započinjanje ispita. Aplikacija namješta ispit, inicijalizira mjerač vremena te popunjava zadatke ovisno o nasumično dodijeljenoj grupi ispita (vidi sliku 7).



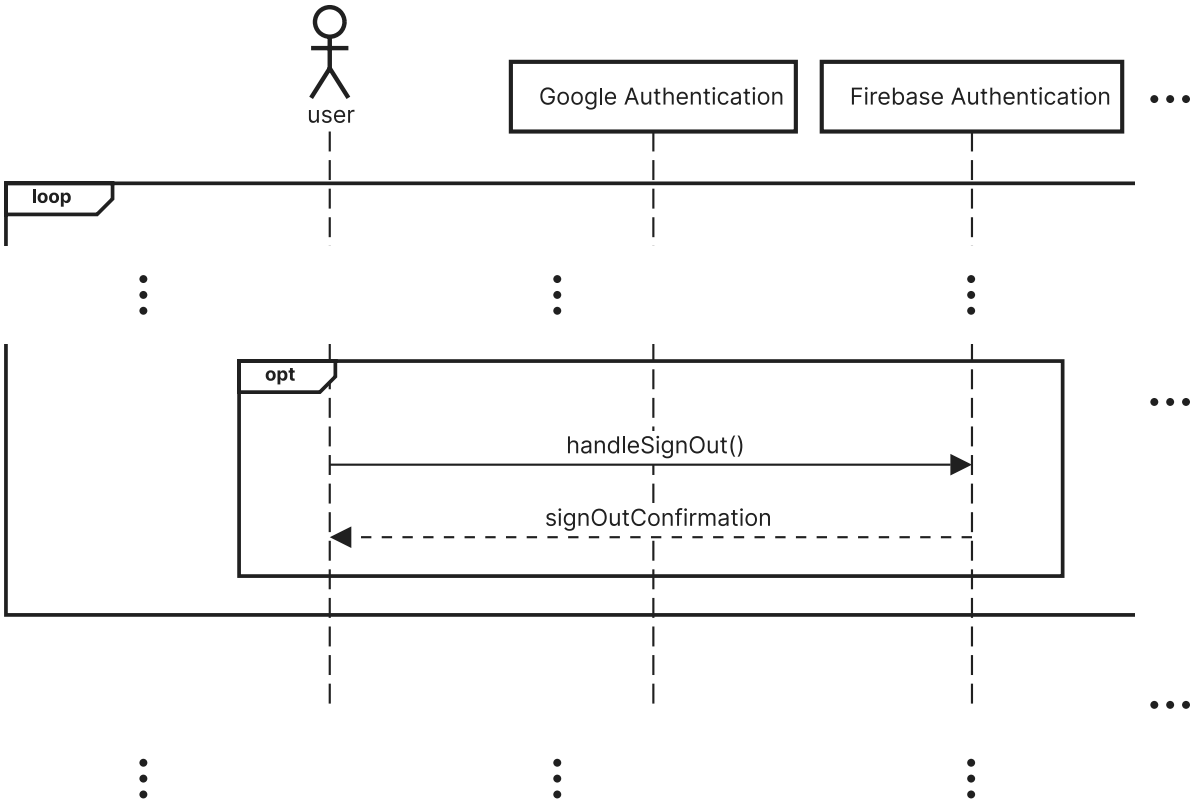
Slika 7: *Model slijeda korisnika - započinjanje i postavljanje ispita (izvor: Autor)*

Kada je ispit namješten i spreman, korisnik može pregledavati vidljive skripte dok rješava ispit, pisati kôd, evaluirati ga te pregledavati povratnu informaciju o evaluiranom kôdu. Kada vrijeme istekne ili korisnik završi ispit, rješenja se pohranjuju u Firebase Storage, o čemu je korisnik obaviješten, te se također lokalno spremaju ista rješenja (vidi sliku 8).



Slika 8: *Model slijeda korisnika - rješavanje i spremanje ispita (izvor: Autor)*

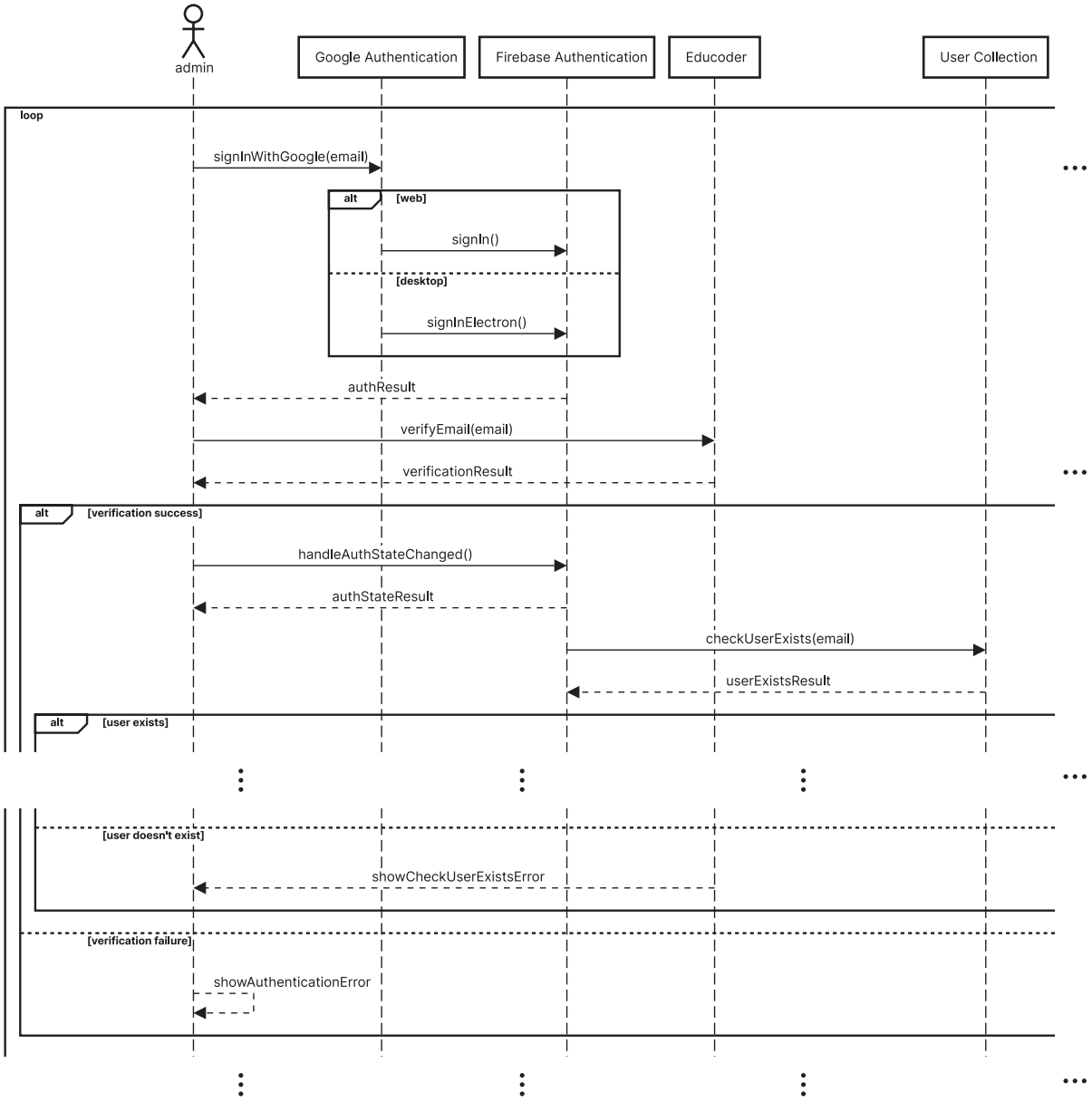
Na kraju svake sesije, korisnik ima mogućnost odjave iz sustava putem funkcije odjave. Time se završava sekvencijski ciklus koji korisnik može ponavljati neograničen broj puta (vidi sliku 9).



Slika 9: *Model slijeda korisnika - odjava (izvor: Autor)*

3.2.2 Model slijeda za administratora

Dijagram slijeda za administratora u EduCoder aplikaciji počinje kada administrator želi pristupiti sustavu i prijaviti se putem Google autentifikacije. Nakon odabira svog Google računa, administratorov zahtjev šalje se Firebase autentifikacijskom sustavu, ovisno o tome koristi li administrator web ili desktop aplikaciju. Nakon prijave, EduCoder provjerava identitet administratora, odnosno validnost email adrese, koja mora imati nastavak ”student.unipu.hr” ili ”unipu.hr”. Ako je email adresa uspješno validirana, administrator prima potvrdu o validaciji; u suprotnom je obaviješten o neuspjeloj autentifikaciji. U slučaju uspješne verifikacije, sustav provjerava nalazi li se administratorov email u kolekciji korisnika. Ako email nije pronađen, administrator se na to upozorava (vidi sliku 10).



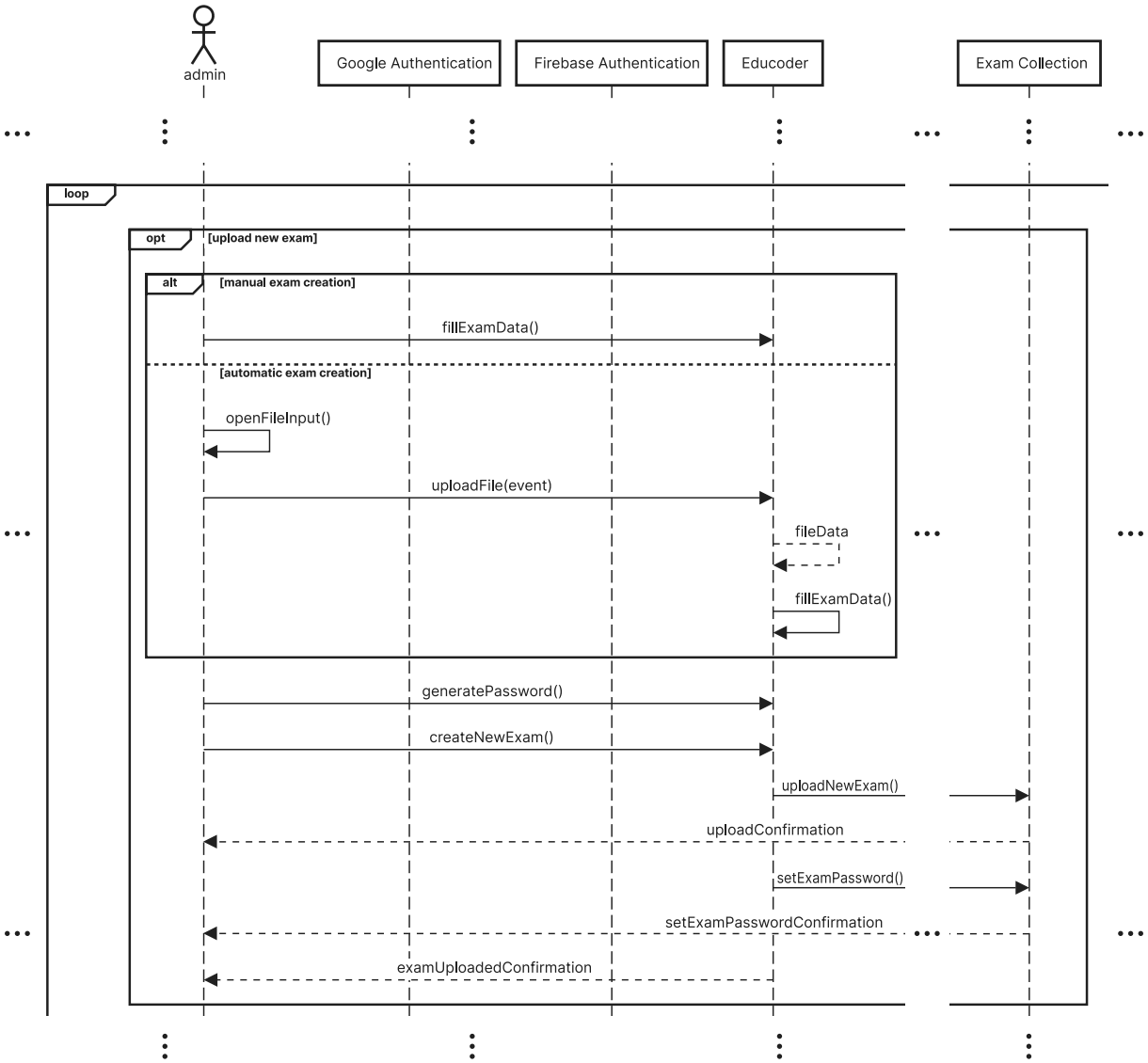
Slika 10: *Model slijeda administratora - prijava i autentifikacija administratora (izvor: Autor)*

Administrator ima tri glavne opcije:

- **učitaj novi ispit** (*eng. upload new exam*)
- **administratorski paneli** (*eng. admin panels*)
- **skupno ocjenjivanje ispita** (*eng. batch examination*)

Kod opcije ”učitaj novi ispit”, administrator može popuniti podatke za ispit na dva načina. Prvi je ručni unos, gdje administrator popunjava svako polje za unos ručno. Drugi je automatski unos, gdje administrator učitava markdown datoteku u EduCoder. EduCoder iz datoteke izdvaja podatke i automatski popunjava sva polja za unos. Nakon popunjavanja podataka, administrator generira novu šifru ispita i sprema ispit

u bazu podataka. Podaci ispita i šifra ispita pohranjuju se u kolekciju ispita. Nakon uspješnog spremanja ispita, administrator prima obavijest (vidi sliku 11).

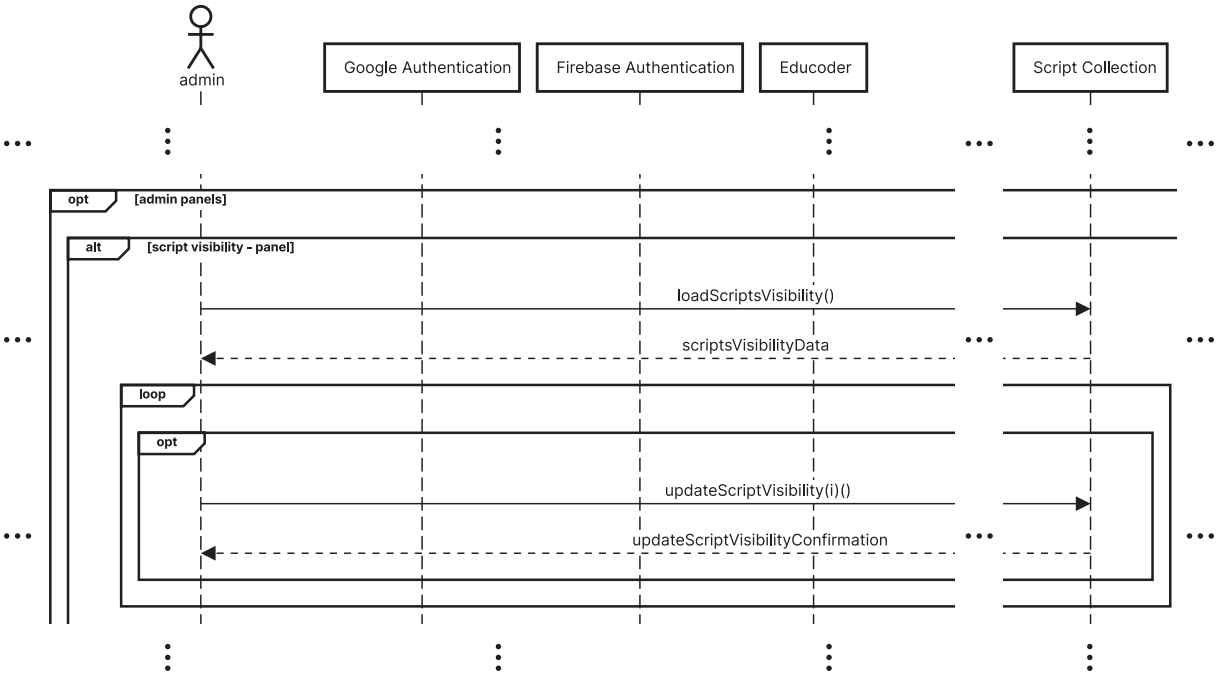


Slika 11: *Model slijeda administratora - učitavanje novog ispita (izvor: Autor)*

Opcija ”administratorski paneli” dalje se dijeli na pet panela koje administrator može koristiti:

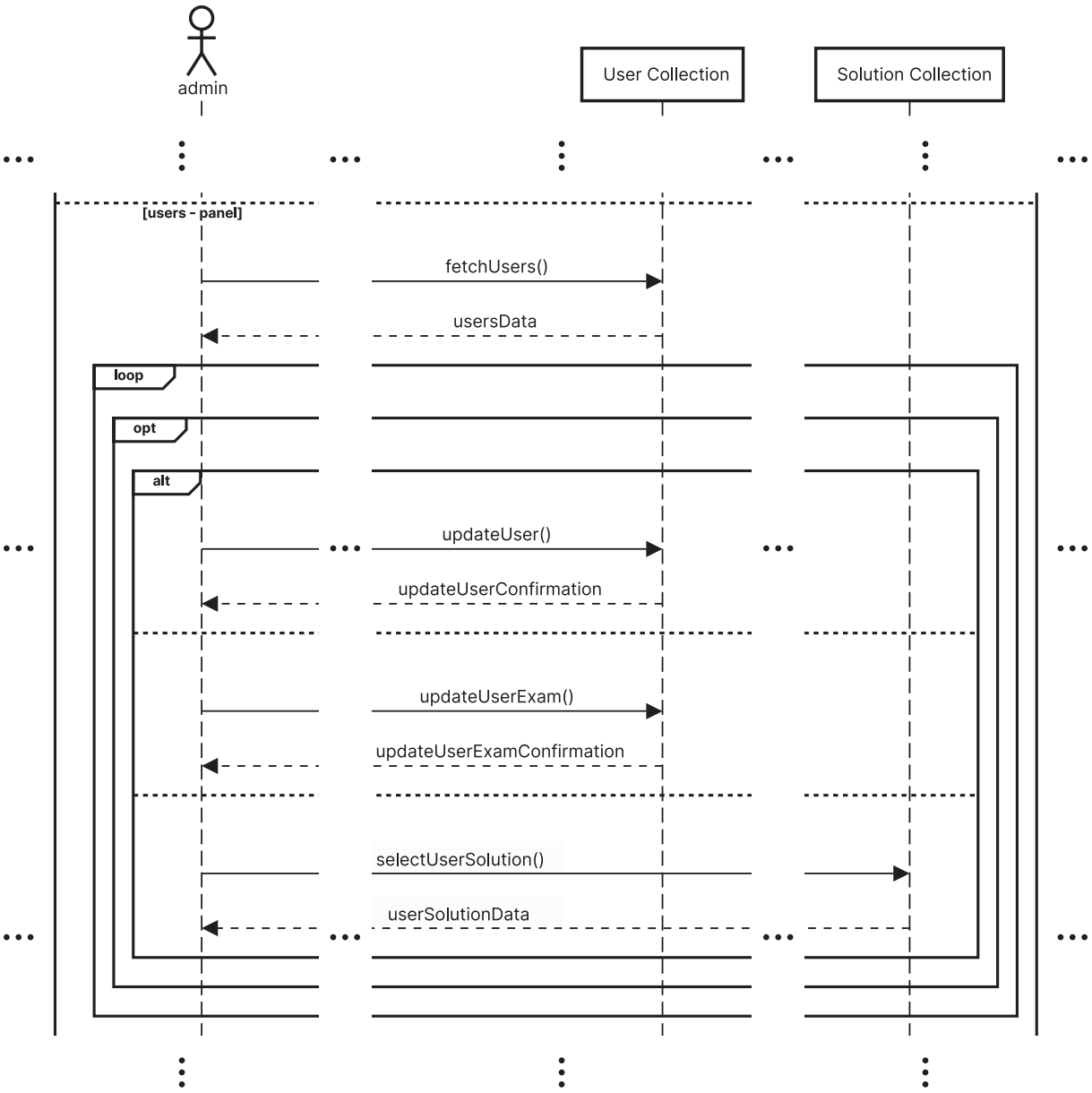
- **vidljivost skripti** (*eng. script visibility*)
- **korisnici** (*eng. users*)
- **svi ispiti** (*eng. all exams*)
- **uredi ispit** (*eng. edit exam*)
- **rješenja** (*eng. solutions*)

U panelu ”vidljivost skripti”, administrator može dohvatiti i promijeniti vidljivost svih skripti. Nakon uspješne promjene, prima se obavijest (vidi sliku 12).



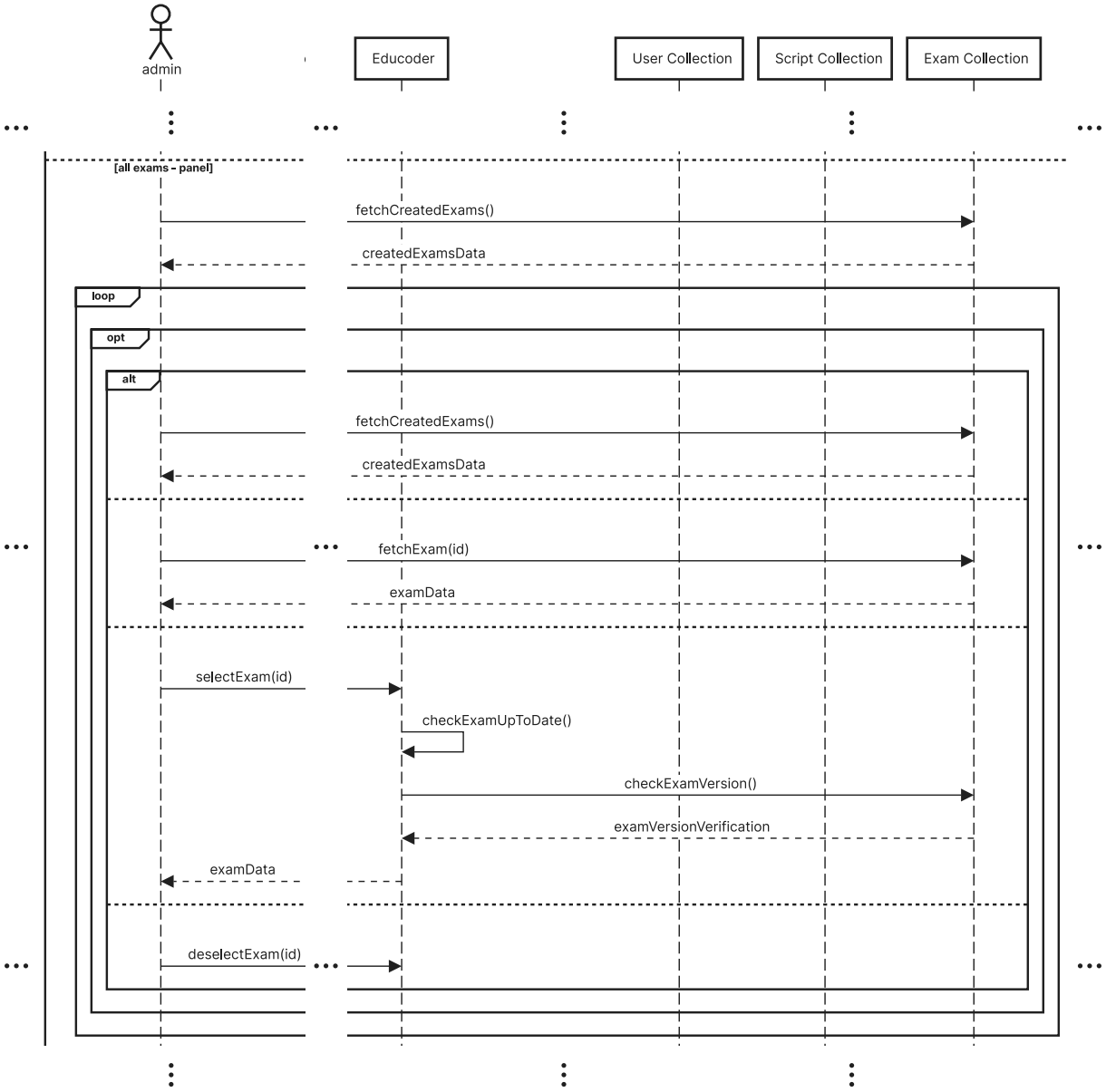
Slika 12: *Model slijeda administratora - učitavanje i uređivanje vidljivosti skripti (izvor: Autor)*

U panelu ”korisnici”, administrator može dohvatiti korisnike prema zadanim filtrima. Nakon dohvaćanja podataka, moguće je ažurirati korisničke podatke, omogućiti pristup pojedinom ispitu ili pregledati predana rješenja (vidi sliku 13).



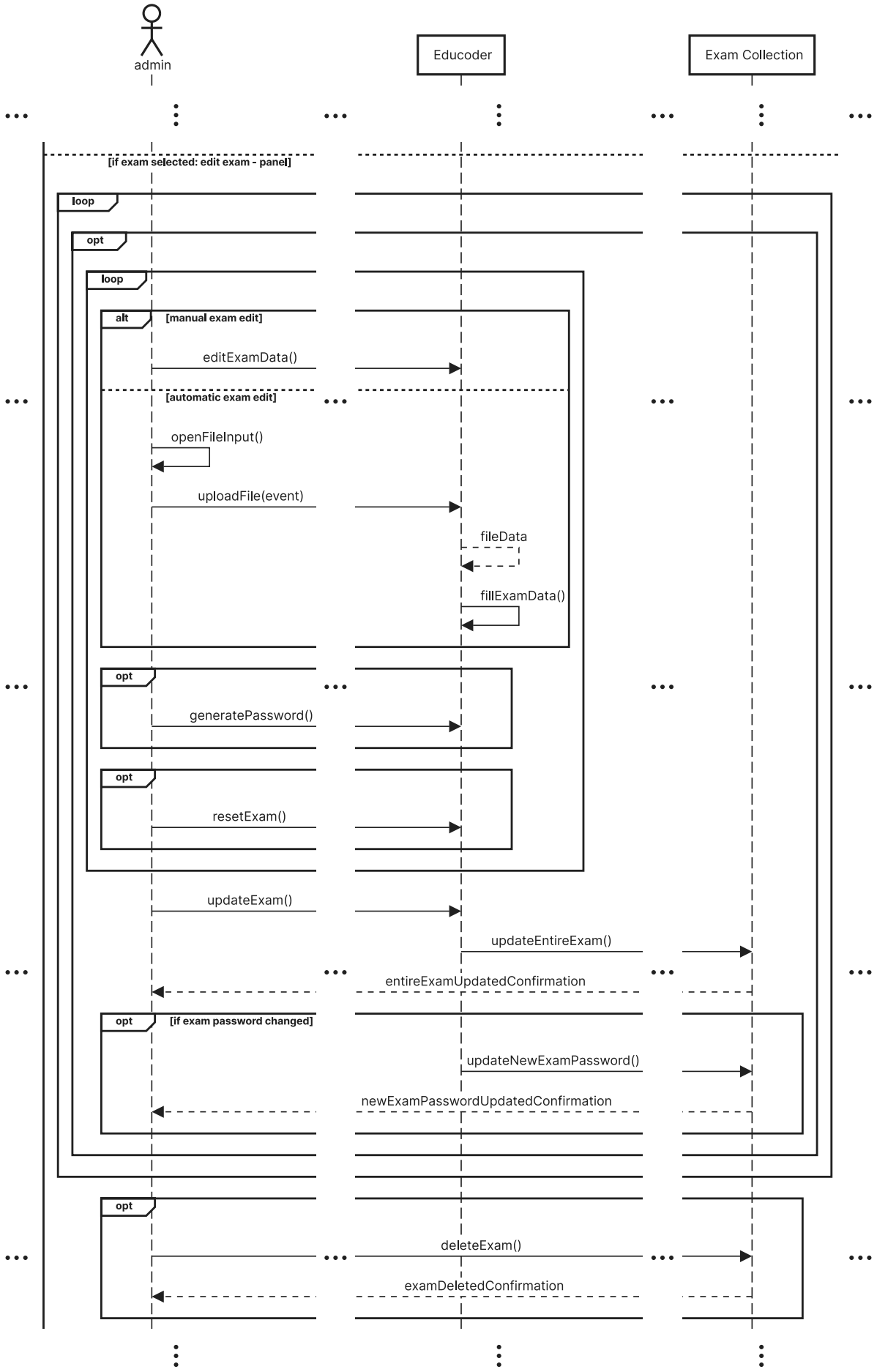
Slika 13: *Model slijeda administratora - učitavanje i uređivanje korisnika (izvor: Autor)*

U panelu ”svi ispiti”, administrator može dohvatiti sve ispite i odabrati pojedini ispit. Provjerava se je li trenutna verzija ispita u skladu s onom u Firebase bazi. Ako verifikacija ne uspije, administrator mora dohvatiti najnoviju verziju ispita; u suprotnom, moguće je uređivati ispit u panelu ”uredi ispit” ili pregledati predana rješenja u panelu ”rješenja”. Također se može poništiti trenutni odabir ispita (vidi sliku 14).



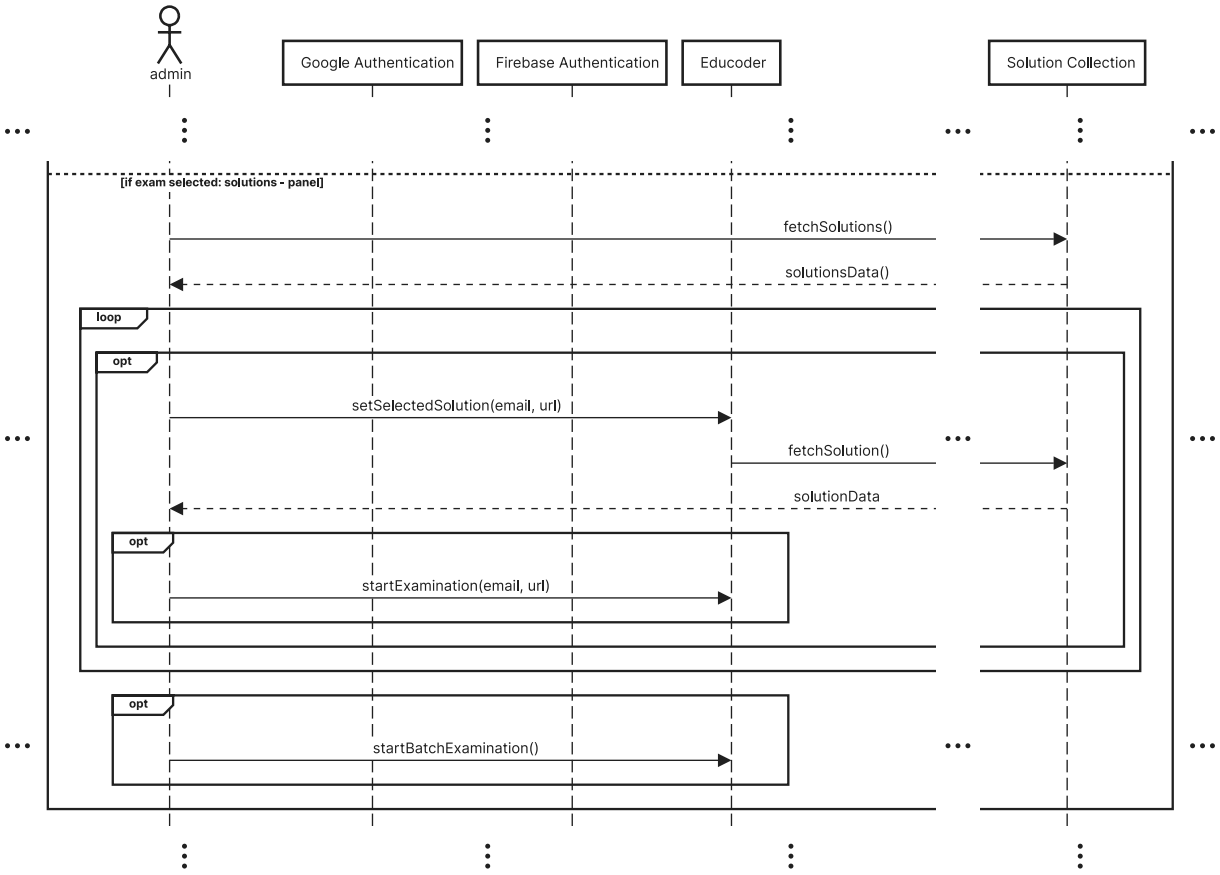
Slika 14: *Model slijeda administratora - učitavanje i pregled svih ispita (izvor: Autor)*

U panelu ”Uredi ispit”, administrator može uređivati i brisati ispite. Uređivanje ispita može se obaviti ručno, popunjavanjem svakog polja, ili automatski, učitavanjem markdown datoteke u EduCoder. Iz datoteke se izdvajaju podaci koji automatski popunjavaju sva polja za unos. Nakon uređivanja podataka, administrator može generirati novu šifru ispita i spremiti promjene u bazu podataka. Nakon uspješnog uređivanja ispita, administrator prima obavijest. Ako se promijeni šifra, ta promjena se također sprema i administrator prima obavijest o uspješnoj promjeni. Kod brisanja ispita, ispit i njegove šifre brišu se iz baze, a administrator je obaviješten o uspješnom brisanju (vidi sliku 15).



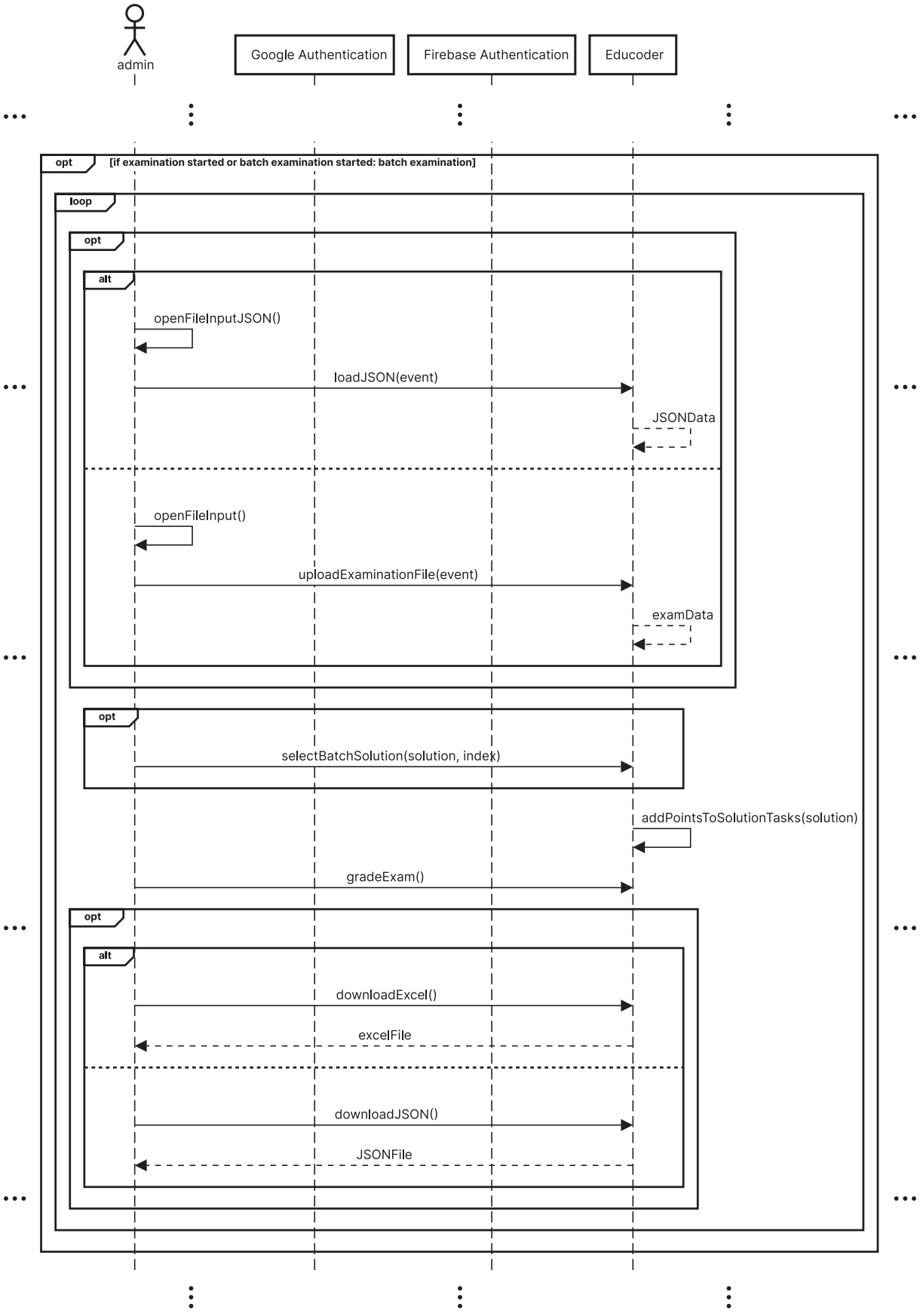
Slika 15: *Model slijeda administratora - uređivanje ispita (izvor: Autor)*

U panelu ”rješenja”, administrator može dohvatiti predana rješenja prema odabranoj šifri ispita. Administrator može odabrati pojedino rješenje korisnika, pregledati ga te započeti ocjenjivanje ili započeti skupno ocjenjivanje ispita (vidi sliku 16).



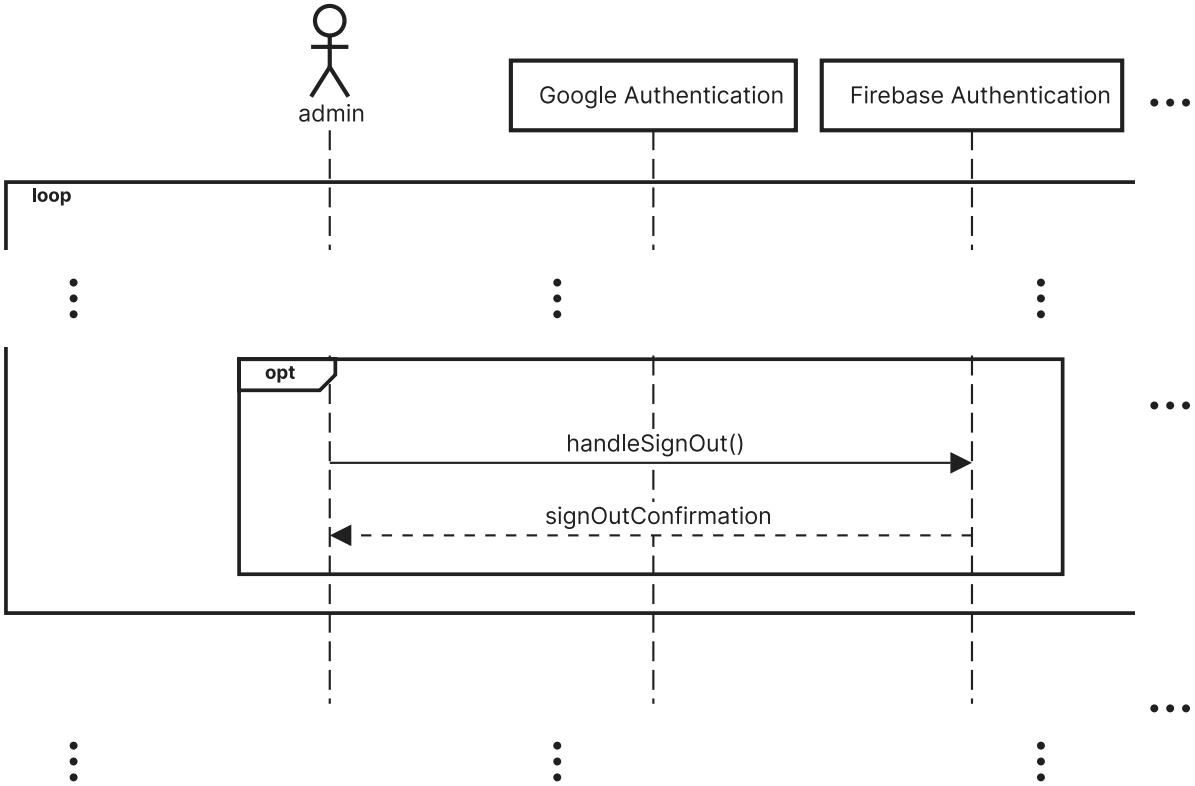
Slika 16: *Model slijeda administratora - pregled predanih rješenja (izvor: Autor)*

Kod opcije ”skupno ocjenjivanje ispita”, administrator ocjenjuje svako predano rješenje. Administrator može učitati dvije datoteke: JSON datoteku koja sadrži stanje ocjenjivanja (ako je prethodno spremljeno) i markdown datoteku ispita koja uključuje upute za bodovanje svakog zadatka. Administrator može odabrati rješenje koje se potom evaluira, pri čemu se zbrajaju bodovi. Nakon završetka ocjenjivanja, moguće je preuzeti trenutno stanje ocjenjivanja u JSON formatu ili analizu bodova u Excel formatu (vidi sliku 17).



Slika 17: *Model slijeda administratora - skupno ocjenjivanje ispita (izvor: Autor)*

Na kraju svake sesije, administrator se može odjaviti iz sustava putem funkcije odjave, čime se završava sekvencijski ciklus koji administrator može ponavljati neograničen broj puta (vidi sliku 18).



Slika 18: *Model slijeda administratora - odjava (izvor: Autor)*

3.3 Klasni model

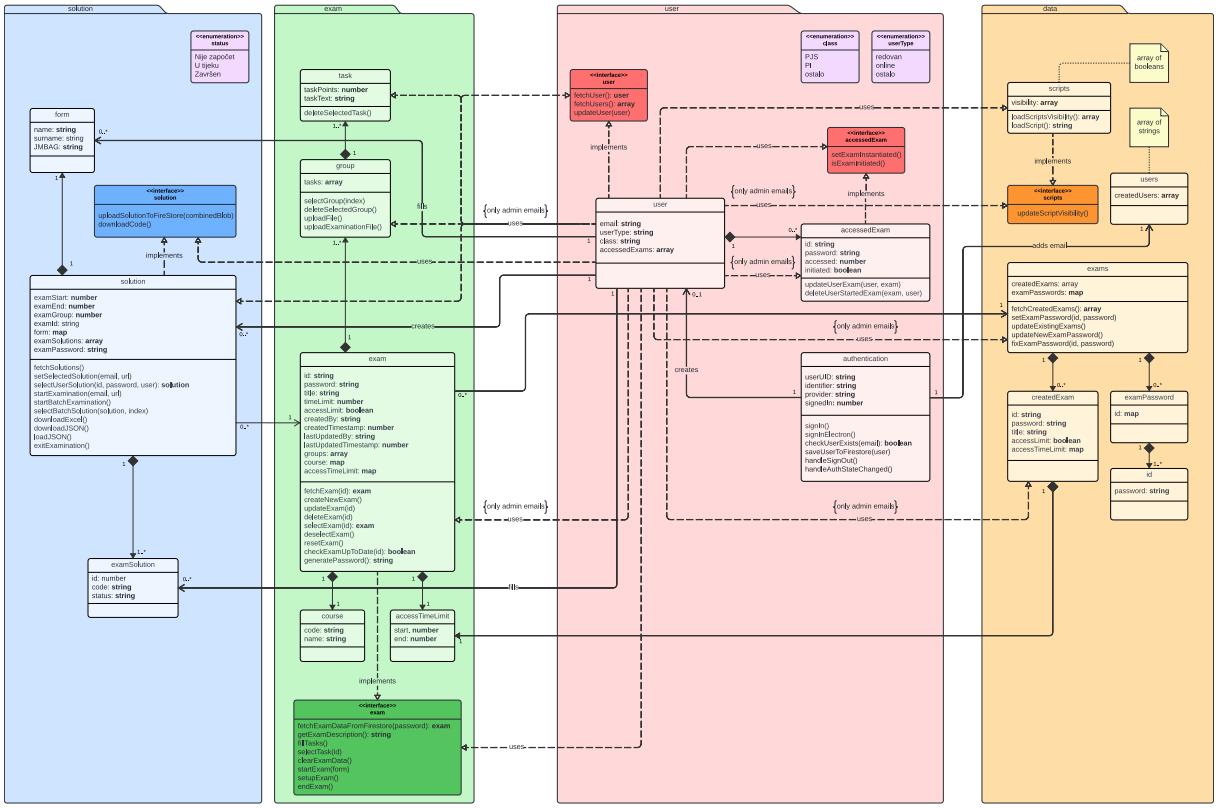
Klasni model (*eng. class diagram*) pruža strukturirani prikaz klasa, atributa i veza između njih unutar sustava. Koristi se za modeliranje statičke strukture aplikacije i odnosa među različitim entitetima. Klasni model omogućuje jasno razumijevanje organizacije kôda i odnosa između različitih komponenti aplikacije. Na slici 19 prikazan je klasni model napravljen prema Firebase strukturi.

Firebase baze podataka, bilo da se radi o stvarnom vremenskom sustavu za bazu podataka ili cloud Firestore, koriste NoSQL pristup. U ovim bazama nema tablica, već kolekcije dokumenata, pri čemu svaki dokument predstavlja strukturu sličnu JSON-u, a dokumenti mogu sadržavati ugrađene kolekcije ili dokumente.

Teorijski, svaki dokument u kolekciji može biti potpuno drugačiji od ostalih. U praksi, dokumenti u kolekciji obično su skup vrlo sličnih objekata, ili barem povezanih objekata, gdje se male razlike u strukturi između dokumenata iste vrste prikazuju s ”opcionalnim” svojstvima unutar klasnog modela. Značajnije razlike između dokumenata obično bi se modelirale s više specijaliziranih klasa. Izazov u modeliranju je razumijevanje

ugrađenih dokumenata i potkolekcija, koje često označavaju druge povezane klase ili kompozicijske klase.

U osnovi, klasni model se ne bi samo koristio za bazu podataka, već bi se koristio za dizajn objektnog modela aplikacije neovisno o bazi podataka. Model bi mogao pomoći u dizajniranju baze podataka na temelju vrste objekata potrebnih za zadovoljavanje potreba korisnika.



Slika 19: *Klasni model (izvor: Autor)*

Klasni model je podijeljen na četiri grupe, gdje tri reprezentiraju Firestore kolekcije (*eng. collection*) i jedna bazu podataka (*eng. storage*):

- **data** - collection (narančasta boja)
- **exam** - collection (zelena boja)
- **user** - collection (crvena boja)
- **solution** - storage (plava boja)

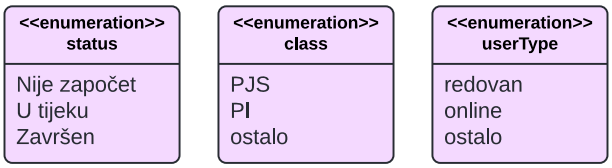
Sve klase u modelu su grupirane unutar jedne od navedenih grupa. Rozom bojom su predstavljena nabrajanja (*eng. enumeration*), tri su u cijelom modelu (vidi sliku 20):

- **class**:
  - vrijednosti: *PJS, PI, ostalo*

  - Predstavljaju skraćene nazive kolegija fakulteta

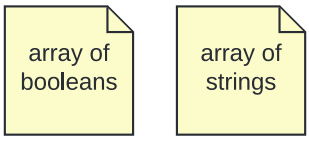
- **userType**:
  - vrijednosti: *redovan, online, ostalo*
  - Predstavljaju vrstu korisnika na fakultetu

- **status**:
  - vrijednosti: *Nije započet, U tijeku, Završen*
  - Predstavlja izvršenost zadatka kod predanog rješenja



Slika 20: *Klasni model - nabrajanja (izvor: Autor)*

Tamnijim bojama prethodno nabrajanih kategorija naznačena su sučelja (*eng. interface*) koja u ovom slučaju predstavljaju dodatne funkcionalnosti klasa te služe za razdvajanje pristupa funkcionalnosti prema ograničenjima (*eng. constraints*). U modelu je samo jedna vrsta ograničenja, *samo administrativni email-ovi* (*eng. only admin emails*), koja naglašava da pojedinom vezom samo korisnici čiji su email-ovi označeni kao ”administrator” unutar Firebase projekta imaju pristup funkcionalnostima krajem veze. Žutom bojom su označene dvije bilješke (*eng. notes*) koje prikazuju dodatne informacije za povezanu klasu (vidi sliku 21).



Slika 21: *Klasni model - bilješke (izvor: Autor)*

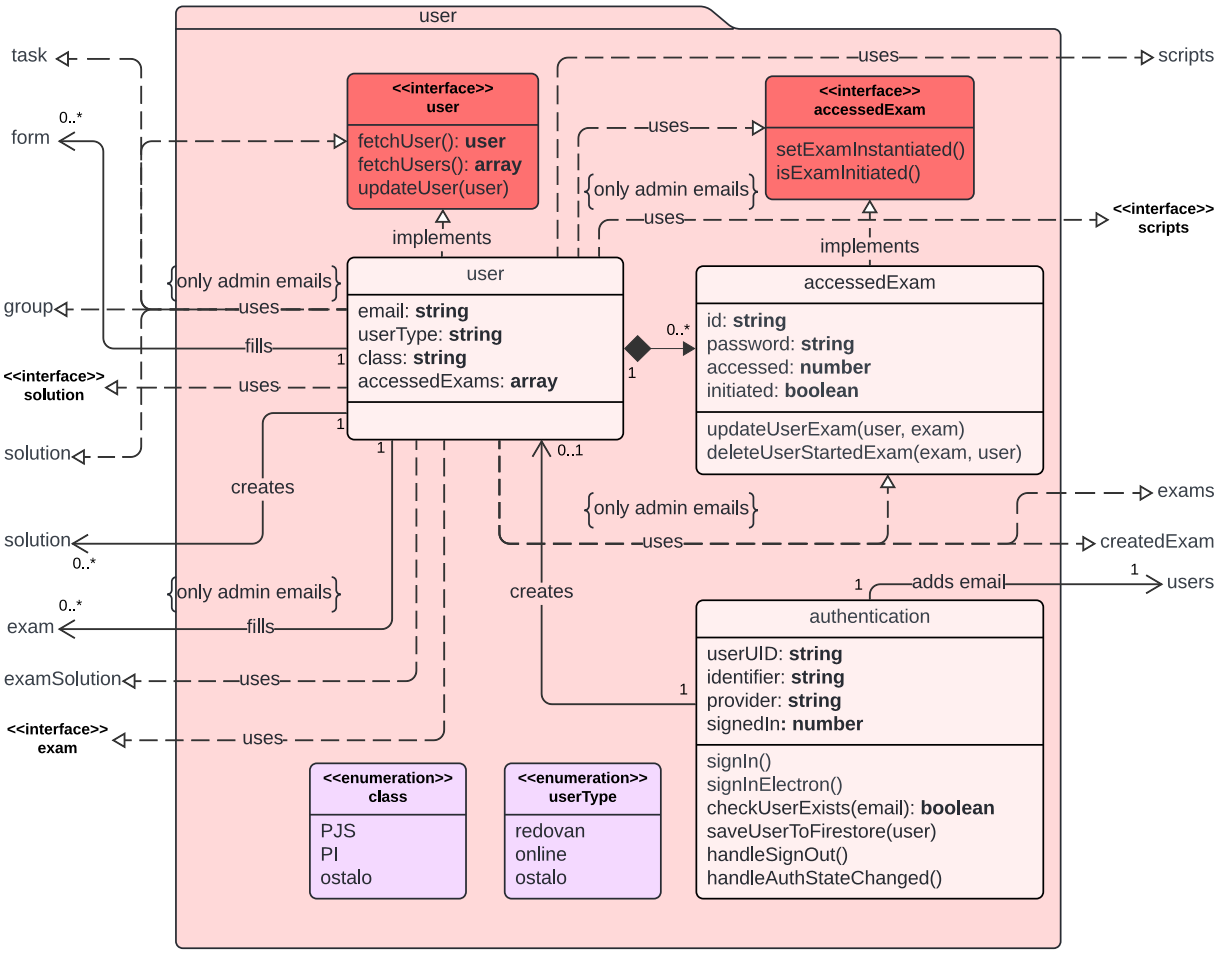
Također, unutar modela je prikazano nekoliko specifičnih veza s nazivima:

- **uses** - označava klasu koja koristi operacije druge klase, u ovom slučaju označava da klasa *user* koristi metode drugih klasa
- **implements** - označava klasu koja implementira operacije sučelja, u ovom slučaju se koristi da se pojedine operacije izdvoje te time omogući razdvajanje administrativnih metoda
- **creates** - označava stvaranje klase na koju je povezana klasa

- **fills** - označava popunjavanje klase na koju je povezana klasa
- **adds email** - označava popunjavanje polja email-ovima unutar druge klase

3.3.1 User

Prvo se kreće od klase *authentication*, koja nije unutar kolekcije *user* već je to Firebase Authentication značajka preko koje se korisnici registriraju koristeći operaciju *signIn()* za web ili *signInElectron()* za desktop. Time se stvara klasa *user* te dodaje email u polje *createdUsers* kod klase *users* koristeći operaciju *saveUserToFirestore(user)*, prije stvaranja i dodavanja provjerava prisutnost korisnika u bazi operacijom *checkUserExists(email)*. Dodatno, klasa ima operaciju *handleSignOut()* koja omogućuje odjavu korisnika iz aplikacije, te *handleAuthStateChanged()* operaciju koja provjerava i automatski prijavljuje korisnika pri ponovnom pristupu aplikaciji (vidi sliku 22).



Slika 22: *Klasni model - user collection (izvor: Autor)*

Dalje je klasa *user* koja je najpovezanija klasa u modelu s obzirom na to da je korisnik taj koji vrši većinu operacija u aplikaciji. Unutar Firebase projekta se definiraju koji su email-ovi administratori. To je vrlo bitno naglasiti jer većina veza zahtijeva da

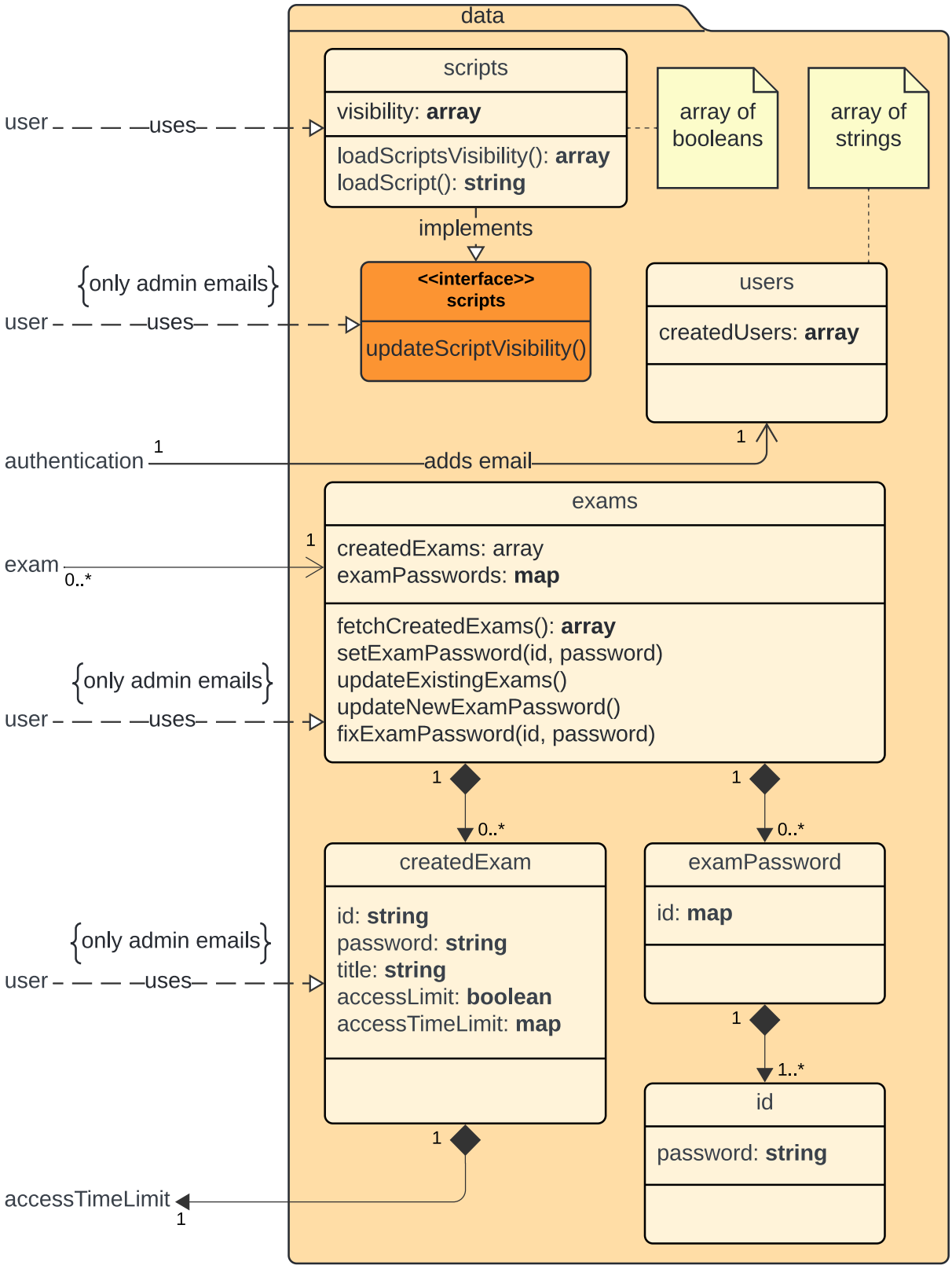
je ograničenje *only admin emails* zadovoljeno. Klasa sama nema operacije nego implementira sučelje koje samo administrator može koristiti. Sučelje proširuje klasu s operacijama:

- **fetchUser()** - vraća pojedinog korisnika
- **fetchUsers()** - vraća listu filtriranih korisnika
- **updateUser(user)** - ažurira atribute korisnika

Klasa sadrži polje *accessedExams*, koje zapravo sadržava instance klase *AccessedExam*, a služi za praćenje vježbi/ispita kojima su korisnici pristupili. Operacije *updateUserExam(user, exam)* i *deleteUserStartedExam(exam, user)* omogućuju administratoru ažuriranje i brisanje pojedinih instanci, dok korisnici mogu koristiti operacije implementiranog sučelja za postavljanje, odnosno nadodavanje nove instance te provjeru *initiated* atributa instance operacijama *setExamInstantiated()* i *isExamInitiated()* respektivno. Nadalje, korisnik koristi operacije klase *scripts* iz grupe *data*, operacije sučelja *exam* iz grupe *exam*, te operacije sučelja *solution* iz grupe *solution*. Administrator dodatno koristi operacije klasa *exams*, *createdExams* i *scripts* iz grupe *data*. Također, koristi operacije klasa *task*, *group* i *exam* iz grupe *exam*, te operacije iz sučelja *solution* iz grupe *solution*. Korisnik može stvarati klase *solution* te popunjavati klase *form* i *examSolution*.

3.3.2 Data

Unutar grupe *data* nalaze se klase koje optimiziraju korištenje Firebase baze tako da smanjuju broj dohvaćanja dokumenata pod cijenom prostora i kompleksnosti (vidi sliku 23), više o tome u poglavlju *baza podataka*.



Slika 23: *Klasni model - data collection (izvor: Autor)*

Unutar klase *scripts* sprema se polje vidljivosti (*eng. visibility*) koje se koristi za lokalno učitavanje određenih skripti. Implementirano je sučelje za dodatnu operaciju *updateScriptVisibility()*, gdje administrator može pojedini element u polju ažurirati, dok

druge dvije operacije dostupne korisnicima su:

- **loadScriptVisibility()** - vraća *visibility* polje
- **loadScript()** - učitava pojedinu skriptu

Klasa *users*, slična prethodnoj klasi, sadrži polje *createdUsers*, koje služi kao brza i efikasna provjera postojanosti korisnika. Zatim, najveća klasa u grupi je klasa *exams*, koja ima dva atributa: *createdExams* polje i *examPasswords* rječnik (*eng. map*). Polje sadržava klase *createdExam*, dok rječnik sadržava klase *examPassword*, koji ima atribut *id*, koji je također rječnik te dalje sadržava klase *password*, koji ima atribut šifru (*eng. password*). Time vidimo da je *examPasswords* atribut zapravo stablasta struktura podataka koja sadrži id-jeve te svaki id sadrži šifre. Služi za brzu provjeru šifri i pristupanje ispitima, dok atribut *createdExams* služi za efikasnije pretraživanje i dohvaćanje ispita. Klasa *exams* sadrži sljedeće operacije dostupne samo administratoru:

- **fetchCreatedExams()** - dohvaća polje *createdExams*
- **setExamPassword(id, password)** - ažurira klasu *examPassword* unutar rječnika *examPasswords*
- **updateExistingExams()** - ažurira polje *createdExams*
- **updateNewExamPassword()** - dodaje novu klasu *examPassword* u rječnik *examPasswords*
- **fixExamPassword(id, password)** - popravlja rječnik *examPasswords* u rijetkom slučaju nepotpunog stvaranja ispita

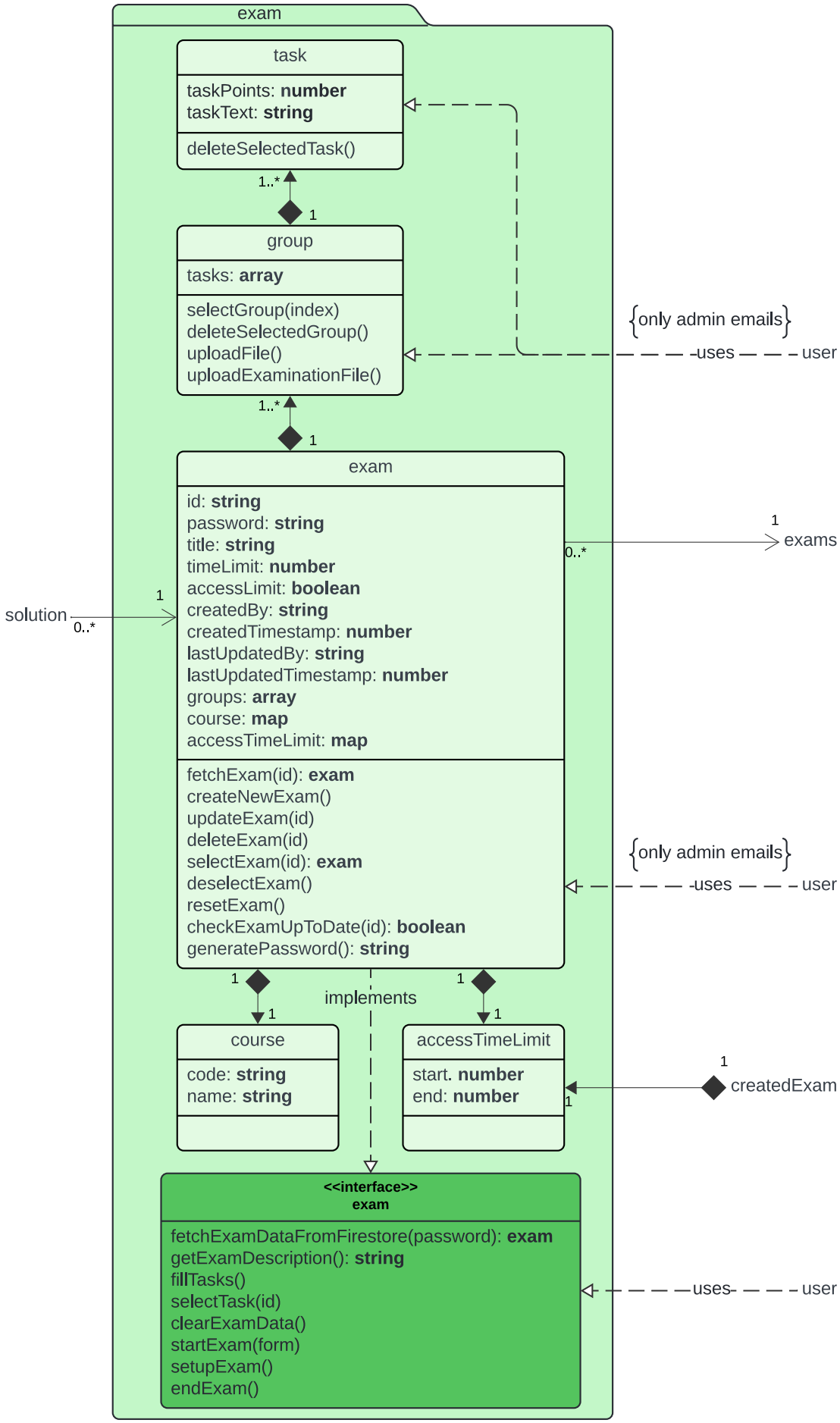
3.3.3 Exam

Ovdje se nalazi najveća klasa u cijelom modelu, klasa *exam* koja sadrži atribute:

- **id** - jedinstvena oznaka ispita koja se nikad ne mijenja
- **password** - jedinstvena šifra ispita po kojoj se pristupa ispitu, može se mijenjati
- **title** - naziv ispita
- **timeLimit** - vremensko ograničenje ispita/trajanje ispita
- **accessLimit** - je li omogućen ponovni pristup ispitu nakon predaje rješenja
- **createdBy** - korisnik koji je stvorio ispit
- **createdTimestamp** - Unix vrijeme kada je ispit stvoren

- **lastUpdatedBy** - korisnik koji je ažurirao ispit
- **lastUpdatedByTimestamp** - Unix vrijeme kada je ispit ažuriran
- **groups** - polje klase *group*
- **course** - rječnik klase *course*
- **accessTimeLimit** - rječnik klase *accessTimeLimit*

Svaki ispit sadrži jednu do više grupa. Svaka grupa sadrži jedan do više zadataka, pri čemu svaki zadatak ima *taskText* i *taskPoints* atribute koji predstavljaju tekst i bodove zadatka. Klasa *Course* posjeduje atribute *code* i *name* koji se koriste za dodatnu kategorizaciju ispita, dok klasa *AccessTimeLimit* posjeduje atribute *start* i *end* za određivanje početka i kraja pristupa ispitu (vidi sliku 24).



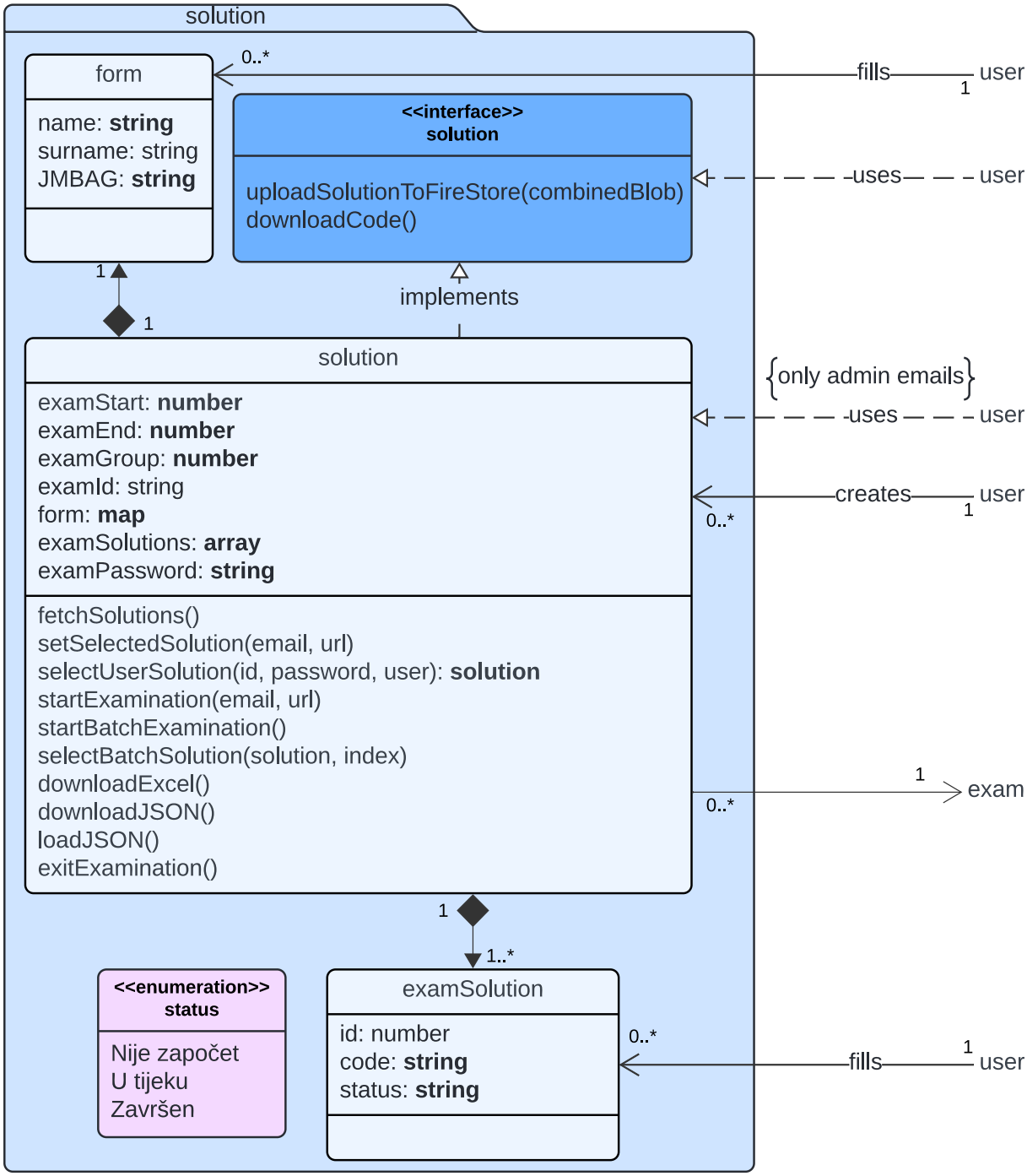
Slika 24: *Klasni model - exam collection (izvor: Autor)*

Samo admin može koristiti operacije klasa *task*, *group* i *exam*, dok korisnik koristi operacije implementiranog sučelja *exam*. Sljedeće operacije su navedene redom:

- klasa *task*
  - **deleteSelectedTask()** - briše se označeni zadatak
- klasa *group*
  - **selectGroup(index)** - označuje se grupa prema proslijeđenom indeksu
  - **deleteSelectedGroup()** - briše se označena grupa
  - **uploadFile()** - učitava se datoteka prema kojoj se automatski generiraju grupe i zadaci
  - **uploadExaminationFile()** - učitava se datoteka prema kojoj se automatski generiraju grupe i zadaci s objašnjenjem bodovanja
- klasa *exam*
  - **fetchExam(id)** - dohvaćanje ispita prema proslijeđenom id-ju
  - **createNewExam()** - izrada novog ispita
  - **updateExam(id)** - ažuriranje ispita prema proslijeđenom id-ju
  - **deleteExam(id)** - brisanje ispita prema proslijeđenom id-ju
  - **selectExam(id)** - označavanje ispita prema proslijeđenom id-ju
  - **deselectExam()** - odznačuje se ispit
  - **resetExam()** - sve trenutne promjene ispita vraćaju se na prijašnje stanje
  - **checkExamUpToDate(id)** - provjerava ažurnost ispita prema proslijeđenom id-ju
  - **generatePassword()** - generiranje nove šifre za ispit
- sučelje *exam*
  - **fetchExamDataFromFirestore(password)** - dohvaćanje podataka ispita prema proslijeđenoj šifri
  - **getExamDescription()** - dohvaćanje opisa ispita
  - **fillTasks()** - popunjavanje zadataka ispita
  - **selectTask(id)** - odabir zadataka ispita prema proslijeđenom indeksu
  - **clearExamData()** - brisanje podataka lokalno dohvaćenog ispita
  - **startExam(form)** - početak rješavanja ispita
  - **setupExam()** - namještanje ispita za rješavanje
  - **endExam()** - završetak rješavanja ispita

3.3.4 Rješenja

Posljednja grupa, *solution*, koja je zapravo *storage*, sadrži spremljene klase *solutions* koje korisnici izrađuju (vidi sliku 25).



Slika 25: *Klasni model - solution storage (izvor: Autor)*

Klasa sadrži atribute *examStart* i *examEnd* koji predstavljaju početak i kraj pisanja ispita, *examGroup*, *examId* i *examPassword* koji predstavljaju grupu, ID i šifru ispita, te polje *examSolutions* i rječnik *form*. U polju se nalaze klase *examSolution*, pri čemu svaka ima *id*, *code* i *status*. U rječniku se nalazi klasa *form*, koja sadrži dodatne infor-

macije (ime, prezime, JMBAG) o korisniku koji rješava ispit. Administrator može koristiti operacije klase *solution*:

- **fetchSolutions()** - dohvaćanje svih rješenja na temelju odabranog ispita
- **setSelectedSolution(email, url)** - dohvaćanje specifičnog rješenja prema emailu i URL-u
- **selectUserSolution(id, password, user)** - dohvaćanje specifičnog rješenja prema ID-u, šifri i korisniku
- **startExamination(email, url)** - otvaranje rješenja prema emailu i URL-u za ocjenjivanje
- **startBatchExamination()** - otvaranje svih rješenja određenog ispita s odabranom šifrom
- **selectBatchSolution(solution, index)** - označavanje otvorenog rješenja prema rješenju i indeksu
- **downloadExcel()** - preuzimanje Excela s ispravljenim ispitima
- **downloadJSON()** - preuzimanje JSON-a trenutnog stanja ispravljanja
- **loadJSON()** - učitavanje JSON-a u trenutno stanje ispravljanja
- **exitExamination()** - izlazak iz ocjenjivanja

Klasa implementira sučelje *solution* koje sadrži operacije koje korisnik može koristiti:

- **uploadSolutionToFireStore(combinedBlob)** - spremanje rješenja na Firebase
- **downloadCode()** - lokalno spremanje rješenja

4 Implementacija

U ovoj sekciji opisana je implementacija EduCoder aplikacije. Implementacija pruža pregled korištenih tehnologija i arhitekture sustava, omogućujući jasno razumijevanje razvoja i funkcioniranja aplikacije. Glavni dijelovi implementacije uključuju korištene metode za razvoj *frontend* i *backend* dijelova, arhitekturu sustava za web i desktop aplikacije, korisničko i administratorsko sučelje, te ključne funkcionalnosti i bazu podataka koja podržava aplikaciju.

4.1 Korištene metode

Implementacija EduCoder aplikacije uključuje korištenje modernih tehnologija i pristupa za izradu interaktivnog i funkcionalnog sustava. Razvoj aplikacije zasnovan je na nekoliko ključnih tehnologija: Vue.js za frontend razvoj, Node.js za backend logiku, Vite za brzu izgradnju i razvoj, Electron za izradu desktop aplikacije te Firebase za autentifikaciju i pohranu podataka.

4.1.1 Vue.js

Vue.js je progresivni JavaScript framework koji se koristi za izradu korisničkih sučelja. Njegova glavna prednost je reaktivna i komponentno bazirana arhitektura koja omogućuje modularni razvoj aplikacija [2].

Glavne karakteristike Vue.js:

- **Reaktivnost**: Vue.js koristi reaktivni sustav podataka koji automatski ažurira DOM kada se promijeni stanje aplikacije. Ovaj sustav omogućuje jednostavno praćenje i upravljanje promjenama u podacima aplikacije bez potrebe za ručnim manipuliranjem DOM-om. To znači da kada se promijeni vrijednost podatka, sve povezane komponente automatski će se ažurirati kako bi prikazale te promjene [3].
- **Komponente**: Komponente su osnovni gradivni blokovi Vue.js aplikacija. One omogućuju enkapsulaciju HTML, CSS i JavaScript kôda u samostalne i ponovno iskoristive module. Svaka komponenta može imati vlastiti skup podataka, metoda i stilova. Komponente također podržavaju hijerarhijsku strukturu, gdje roditeljske komponente mogu prenositi podatke i metode na svoje podređene komponente [4].
- **Direktive**: Vue.js koristi direktive, kao što su `v-if`, `v-for` i `v-bind`, koje omogućuju deklarativno vezivanje podataka i manipuliranje DOM-om. Direktive se koriste za

uvjetno prikazivanje elemenata, iteraciju preko skupa podataka i vezivanje atributa ili stilova na temelju podataka aplikacije. Osim toga, Vue.js omogućuje rukovanje događajima putem direktive `v-on`, koja pojednostavljuje dodavanje slušatelja događaja kao što su klikovi, unosi i ostale korisničke interakcije [5].

- **Virtualni DOM**: Vue.js koristi virtualni DOM za optimizaciju performansi. Virtualni DOM je lagana kopija stvarnog DOM-a koja se koristi za minimiziranje broja promjena koje se moraju napraviti u stvarnom DOM-u. Kada se podaci promijene, Vue.js prvo ažurira virtualni DOM, a zatim uspoređuje razlike između virtualnog i stvarnog DOM-a te primjenjuje samo neophodne promjene. Ovaj pristup značajno poboljšava performanse aplikacija, posebno kod kompleksnih i dinamičnih korisničkih sučelja [6].
- **Options API i Composition API**: Vue.js podržava dva glavna pristupa u razvoju komponenti: Options API i Composition API (vidi sliku 26). Options API koristi objektno orijentirani pristup gdje su sve metode, podaci i lifecycle hooks definirani unutar objekta komponente. Composition API, s druge strane, omogućuje funkcionalni pristup koji koristi reaktivne reference i funkcije za definiciju stanja i ponašanja komponenti [7].

Slika 26: *Usporedba Options API i Composition API (izvor: Vue.js Team)*

Prednosti Vue.js:

- **Jednostavnost**: Vue.js je jednostavan za učenje i korištenje, što ga čini pristupačnim za developere svih razina. Intuitivna sintaksa, opsežna i jasna dokumentacija omogućuju brzi početak rada s frameworkom.
- **Modularnost**: Komponentno bazirana arhitektura Vue.js omogućuje modularni razvoj, što olakšava održavanje i skaliranje aplikacije. Komponente omogućuju enkapsulaciju funkcionalnosti u samostalne i ponovno iskoristive module. Ovo poboljšava čitljivost i održavanje kôda te olakšava suradnju među razvojnim timovima jer se različiti dijelovi aplikacije mogu razvijati i testirati neovisno.

- **Visoke performanse**: Vue.js koristi optimizacije kao što su lazy loading komponenti, virtualni DOM i memorijsko učinkovite metode za smanjenje opterećenja i ubrzanje renderiranja korisničkog sučelja [8].
- **Fleksibilnost i integracija**: Vue.js se može lako integrirati s postojećim projektima i aplikacijama. Ova fleksibilnost omogućuje korištenje Vue.js za različite projekte, bilo da se radi o jednostavnim komponentama za interaktivnost ili kompleksnim aplikacijama za jednu stranicu. Podrška za TypeScript dodatno povećava prilagodljivost i upotrebljivost u različitim razvojnim okruženjima, omogućujući razvojnim timovima da koriste najnovije tehnike i alate za razvoj softvera [2].

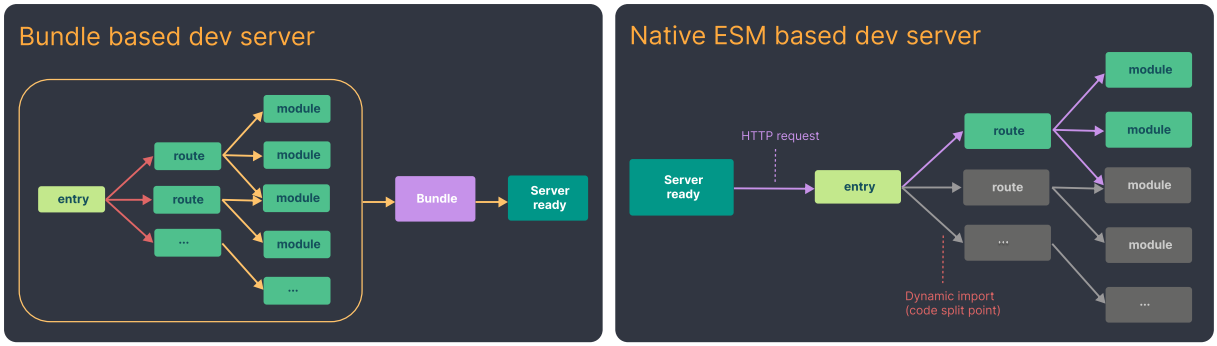
Nedostaci Vue.js:

- **Problemi s kompatibilnošću**: Budući da Vue.js često ažurira svoje verzije (Vue 1, Vue 2, Vue 3), ponekad može doći do problema s kompatibilnošću između različitih verzija. To može uzrokovati poteškoće pri održavanju i nadogradnji postojećih projekata, zahtijevajući dodatno vrijeme i resurse za prilagodbu novim verzijama.
- **Manja zajednica i manje trećih alata/knjižnica**: Iako zajednica Vue.js raste, ona je manja u usporedbi s Reactom ili Angularom, što može utjecati na dostupnost resursa i podrške. No, manja zajednica ne znači mala; ona je i dalje prilično velika, ali nije toliko opsežna kao ekosustav oko Reacta.
- **Ograničena podrška za TypeScript**: Vue.js podržava TypeScript, ali ta podrška može biti nedovoljna u usporedbi s drugim frameworkovima. Često je potrebno ponovno pokretati TypeScript server, a Vue.js zahtijeva Volar ekstenziju za pravilno funkcioniranje, što uvodi dodatni sloj apstrakcije koji može biti sklon pogreškama. Iako se props mogu definirati kao sučelja, postoji ograničena podrška za kompleksna sučelja jer Vue interno transformira props u objekt prop, što može biti zbunjujuće [9].
- **Složenost kompozicije i reaktivnosti**: Korištenje Vue Composition API-a može biti nezgrapno, posebno kada je riječ o očuvanju reaktivnosti. Prosljeđivanje props-a u kompozicijske funkcije i zadržavanje reaktivnosti može biti frustrirajuće jer je potrebno omotati props u `computed` ili `toRef` kako bi se zadržala reaktivnost [9].

4.1.2 Vite.js

Vite je moderni alat za izgradnju frontend aplikacija koji značajno ubrzava i poboljšava procese razvoja i produkcije. Razvijen je kao rješenje za ograničenja i sporost tradicionalnih alata poput Webpacka i Parcela. Vite koristi inovativne tehnike, uključujući

nativne ES module (vidi sliku 27) i hot module replacement (HMR), što omogućuje ažuriranje tijekom razvoja, čime se značajno smanjuje vrijeme potrebno za kompilaciju i poboljšava učinkovitost razvoja [10].



Slika 27: *Bundle based dev server and native ESM based dev server (izvor: Vite.js Team)*

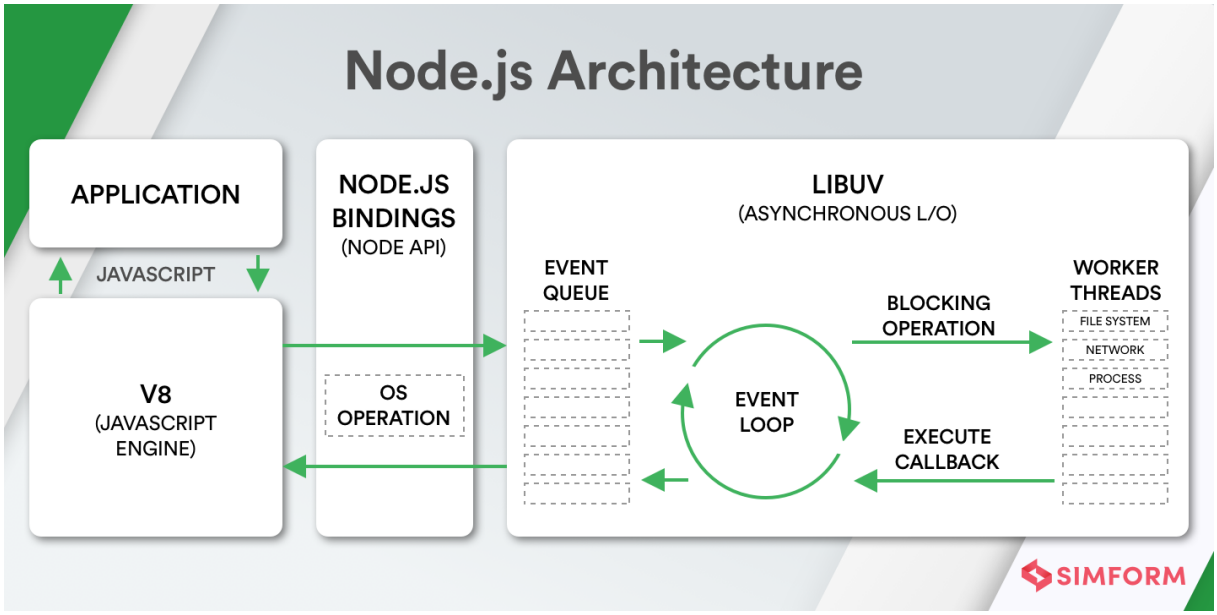
Glavne karakteristike Vitea:

- **Brzina**: Vite se ističe po svojoj brzini, posebno pri pokretanju razvoja. Zahvaljujući korištenju nativnih ES modula, Vite može odmah služiti aplikaciju bez potrebe za prethodnim procesiranjem svih datoteka. Ovo značajno smanjuje vrijeme potrebno za pokretanje projekta [10].
- **Hot Module Replacement (HMR)**: Vite omogućuje trenutačna ažuriranja modula tijekom razvoja bez potrebe za ponovnim učitavanjem cijele stranice. HMR povećava produktivnost programera omogućavajući brze iteracije i trenutne povratne informacije o promjenama u kôdu [11].
- **Optimizacija za produkciju**: Za produkciju, Vite koristi Rollup kao bundler. Rollupova modularna arhitektura omogućuje Viteu da iskoristi napredne tehnike optimizacije za minimaliziranje veličine i poboljšanje performansi aplikacija [12].
- **Plugin ekosustav**: Pluginovi se mogu koristiti za dodavanje podrške za različite tehnologije kao što su TypeScript, Vue, React i mnoge druge, čineći Vite vrlo prilagodljivim za različite projekte [13].
- **Jednostavnost konfiguracije**: Vite dolazi s jednostavnom i minimalističkom konfiguracijom koja omogućava brz početak rada. Vite koristi praktične zadane postavke koje rade iz kutije, ali omogućuju i jednostavno prilagođavanje prema potrebama projekta [10].

4.1.3 Node.js

Node.js je okruženje otvorenog kôda (*eng. open-source*) koje omogućuje izvršavanje JavaScript kôda na poslužiteljskoj strani. Ova višeplatformska (*eng. cross-platform*)

tehnologija temelji se na V8 JavaScript engine-u, kojeg koristi i Google Chrome. Node.js pruža visokokvalitetan i učinkovit način za razvoj skalabilnih mrežnih aplikacija [14]. Njegova glavna prednost leži u sposobnosti asinkronog programiranja, što omogućuje aplikacijama izvršavanje više funkcija istovremeno, bez blokiranja procesa (vidi sliku 28) [15].



Slika 28: *Node.js arhitektura (izvor: Simform)*

Node.js koristi opsežan ekosustav modula i biblioteka dostupnih putem NPM-a (Node Package Manager), omogućujući brz i jednostavan razvoj aplikacija [16]. Posebno je koristan za razvoj aplikacija koje zahtijevaju brzu obradu podataka i visoku propusnost, poput RESTful API-ja, streaming aplikacija i sustava za razmjenu podataka [17].

Glavne karakteristike Node.js:

- **Asinkronost i non-blocking I/O**: Node.js koristi asinkrone operacije i non-blocking I/O, omogućujući visoku skalabilnost i učinkovitost. Umjesto čekanja na završetak operacije, Node.js nastavlja s izvršavanjem kôda, čime se optimizira korištenje resursa [18].
- **Jedinstveni jezični ekosustav**: Korištenjem JavaScripta za backend razvoj, Node.js omogućuje developerima upotrebu istog jezika na klijentskoj i serverskoj strani. Ova jedinstvena jezična konzistencija pojednostavljuje razvojne procese i olakšava prelazak između frontend i backend zadataka [19].
- **Arhitektura temeljena na događajima**: Node.js se temelji na event-driven arhitekturi, gdje se operacije izvršavaju kao odgovori na događaje. Ova arhitektura

smanjuje opterećenje na serveru i omogućuje učinkovito rukovanje velikim brojem istovremenih korisnika [15].

- **NPM (Node Package Manager)**: NPM je ključni dio Node.js ekosustava, pružajući pristup velikom broju paketa i modula. Ova bogata zbirka resursa omogućuje developerima brzo pronalaženje i integraciju potrebnih funkcionalnosti u aplikacije [16].
- **Visoke performanse**: Zahvaljujući V8 JavaScript engineu, Node.js postiže visoke performanse i brzinu izvršavanja kôda. V8 kompajlira JavaScript u nativni strojni kôd, što rezultira bržim izvršavanjem i boljom optimizacijom aplikacija [20].

Nedostaci Node.js-a:

- **Jedinstveni thread**: Node.js koristi jedinstveni thread za izvršavanje kôda, što može biti ograničavajuće za CPU-intenzivne zadatke. CPU-intenzivne operacije mogu značajno usporiti rad cijelog sustava [21].
- **Složenost asinkronog programiranja**: Asinkroni model programiranja, iako efikasan, može biti složen za razumijevanje i implementaciju, posebno za početnike. Ovo može otežati debugiranje i održavanje aplikacija [22].
- **Nedostatak kvalitetnih paketa**: Iako NPM nudi veliki broj paketa, kvaliteta mnogih od njih može biti upitna. Također, postavljanje Node.js servera može zahtijevati više vremena i truda zbog različitih konfiguracija i problema koji se mogu pojaviti [23].

4.1.4 Electron

Electron je open-source framework koji omogućuje izradu desktop aplikacija korištenjem web tehnologija kao što su HTML, CSS i JavaScript. Razvijen od strane GitHub-a, Electron kombinira Chromium renderiranje i Node.js runtime, što omogućuje korištenje web tehnologija za izradu aplikacija koje se mogu pokretati na različitim operativnim sustavima, uključujući Windows, macOS i Linux [24].

Glavne karakteristike Electrona:

- **Cross-platform podrška**: Electron omogućuje izradu aplikacija koje se mogu pokretati na različitim operativnim sustavima bez potrebe za promjenama u kôdu. To značajno smanjuje vrijeme i resurse potrebne za razvoj i održavanje aplikacija [25].
- **Web tehnologije**: Korištenjem poznatih web tehnologija poput HTML-a, CSS-a i JavaScripta, Electron omogućuje web developerima da koriste svoje postojeće vještine za izradu desktop aplikacija. Ovo pojednostavljuje proces razvoja,

ubrzava uvođenje novih značajki te omogućuje lak prijelaz s weba na desktop okruženje [26].

- **Integracija s Node.js**: Electron koristi Node.js za backend operacije, što omogućuje pristup velikom ekosustavu Node.js modula i paketa, što omogućava jednostavnu integraciju s različitim servisima i bazama podataka [27].
- **Automatsko ažuriranje**: Electron podržava automatska ažuriranja aplikacija, što omogućuje jednostavnu distribuciju novih verzija i poboljšanja korisnicima bez potrebe za ručnim ažuriranjem [28].

Nedostaci Electrona:

- **Velika veličina aplikacija**: Electron aplikacije mogu biti prilično velike zbog uključivanja Chromiuma i Node.js runtime-a.
- **Potrošnja resursa**: Electron aplikacije često troše više memorije i CPU resursa u usporedbi s nativnim aplikacijama, što može biti problematično za korisnike sa starijim računalima [29].
- **Sigurnosni izazovi**: Budući da Electron aplikacije koriste web tehnologije, mogu biti podložne istim sigurnosnim prijetnjama kao i web aplikacije, poput XSS napada, zbog čega je potrebna dodatna pažnja pri osiguravanju aplikacija [30].

4.1.5 Firebase

Firebase je sveobuhvatna platforma razvijena od strane Googlea koja olakšava razvoj mobilnih i web aplikacija pružajući niz moćnih alata i infrastrukturu. Platforma omogućuje jednostavnu izradu, unaprjeđivanje i skaliranje aplikacija nudeći razne backend usluge kao što su autentifikacija korisnika, pohrana i sinkronizacija podataka, hosting, izvršavanje funkcija u oblaku, te analitiku i praćenje performansi aplikacija [31].

Glavne karakteristike Firebasea:

- **Firebase Authentication**: Firebase Authentication pruža jednostavne i sigurnosne metode za autentifikaciju korisnika koristeći različite načine prijave, uključujući email i lozinku, telefonski broj, kao i autentifikaciju putem popularnih društvenih mreža poput Googlea, Facebooka i Twittera [32].
- **Firebase Realtime Database**: Firebase Realtime Database je NoSQL baza podataka koja omogućuje pohranu i sinkronizaciju podataka u realnom vremenu. Podaci se pohranjuju u JSON formatu i automatski se sinkroniziraju s klijentima u stvarnom vremenu [33].

- **Cloud Firestore**: Cloud Firestore je fleksibilna, skalabilna baza podataka za razvoj mobilnih, web i server aplikacija. Podržava strukture podataka kao što su kolekcije i dokumenti, te omogućuje napredne upite i indeksiranje [34].
- **Firebase Storage**: Firebase Storage omogućuje pohranu i dohvaćanje korisničkih datoteka kao što su slike, videozapisi i drugi dokumenti. Omogućuje sigurnu i skalabilnu pohranu s integriranim Firebase Security Rules za zaštitu podataka [35].
- **Firebase Hosting**: Firebase Hosting pruža brzi hosting za web aplikacije i statične sadržaje. Omogućuje jednostavno implementiranje i upravljanje web aplikacijama te podržava SSL certifikate za sigurnost [36].
- **Cloud Functions**: Cloud Functions omogućuju pokretanje backend kôda kao odgovora na događaje pokrenute putem Firebase značajki i HTTPS zahtjeva [37].
- **Firebase Analytics**: Firebase Analytics pruža besplatne i neograničene analitičke usluge za praćenje ponašanja korisnika unutar aplikacije [38].

Prednosti Firebasea:

- **Brza implementacija**: Firebase omogućuje brzu implementaciju backend funkcionalnosti bez potrebe za upravljanjem vlastitim serverima.
- **Skalabilnost**: Firebase usluge su dizajnirane da se skaliraju prema potrebama aplikacije, omogućujući rast i prilagodbu povećanom broju korisnika.
- **Integracija s drugim Google uslugama**: Firebase se jednostavno integrira s drugim Google uslugama kao što su Google Ads, Google Analytics i BigQuery, omogućujući sveobuhvatan pristup podacima i analitici.
- **Sigurnost**: Firebase pruža robustan sigurnosni model s detaljnim pravilima pristupa i autentifikacije.

Nedostaci Firebasea:

- **Ovisnost o Google ekosustavu**: Korištenje Firebasea može značiti veliku ovisnost o Googleovom ekosustavu, što može biti problematično u slučaju promjena politika ili cijena.
- **Troškovi**: Iako Firebase nudi besplatni sloj, troškovi mogu brzo rasti s povećanjem broja korisnika i količine pohranjenih podataka.
- **Ograničena fleksibilnost**: Firebase usluge su dizajnirane da budu jednostavne za korištenje, što može ograničiti fleksibilnost i prilagodbu u specifičnim slučajevima.

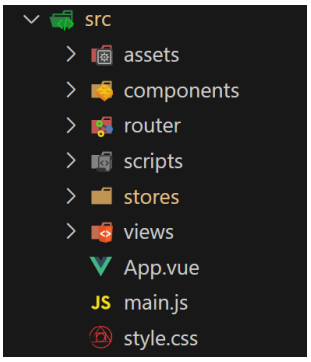
4.2 Arhitektura sustava

U ovom poglavlju detaljno se opisuje arhitektura sustava EduCoder aplikacije. Arhitektura sustava pruža jasan i strukturiran pregled komponenti aplikacije, njihovih međusobnih odnosa i načina na koji surađuju kako bi osigurale funkcionalnost i performanse aplikacije. Glavne komponente arhitekture uključuju web i desktop aplikacije te njihove ključne module i servise.

S obzirom na to da je desktop verzija aplikacije bazirana na Electron tehnologiji, struktura je u velikoj mjeri preuzeta iz web verzije aplikacije. Stoga će se prvo opisati općenita struktura aplikacije i korišteni npm paketi za obje verzije, nakon čega slijede potpoglavlja koja detaljno opisuju specifičnosti arhitekture za web i desktop aplikacije.

4.2.1 Struktura aplikacije

Struktura aplikacije organizirana je na način koji omogućuje jednostavno upravljanje kôdom i resursima.



Slika 29: *Struktura glavnog direktorija (izvor: Autor)*

Slika 29 prikazuje direktorijsku strukturu unutar glavnog direktorija **src**:

- **assets**: Sadrži statičke resurse kao što su slike, fontovi i drugi medijski sadržaji koji se koriste u aplikaciji. Resursi smješteni ovdje često se referenciraju u komponentama i stilovima.
- **components**: Sadrži pojedinačne Vue komponente koje se koriste unutar aplikacije. Komponente su modularni dijelovi aplikacije koji mogu sadržavati vlastiti HTML, CSS i JavaScript kôd. Razbijanje aplikacije na manje komponente poboljšava čitljivost i održavanje kôda.
- **router**: Sadrži datoteke vezane uz rutiranje aplikacije. U Vue.js aplikacijama, `router` upravlja navigacijom i omogućuje definiranje različitih ruta koje korisnik može posjetiti. Najčešće se koristi `vue-router` za ovu svrhu.

- **scripts**: Sadrži različite skripte koje se koriste za aplikaciju, kao što su pomoćne funkcije, konfiguracije i drugi JavaScript kôd koji nije specifično vezan za komponente.
- **stores**: Sadrži Pinia store datoteke koje upravljaju globalnim stanjem aplikacije. Pinia je state management library za Vue.js aplikacije, koji omogućuje centralizirano upravljanje stanjem.
- **views**: Sadrži Vue komponente koje predstavljaju različite prikaze ili stranice unutar aplikacije. Svaki prikaz odgovara jednoj ruti u aplikaciji.
- **App.vue**: Glavna Vue komponenta aplikacije koja služi kao korijenska komponenta. Unutar ove datoteke definiran je osnovni layout i struktura aplikacije.
- **main.js**: Služi kao ulazna točka aplikacije. Ovdje se inicijalizira glavna *Vue instance* i definiraju osnovne postavke aplikacije. U `main.js` datoteci se uvoze potrebni moduli i komponente, kreira i konfigurira glavna Vue instance te montira na određeni DOM element u HTML dokumentu.
- **style.css**: Sadrži globalne stilove koji se primjenjuju na cijelu aplikaciju. CSS definiran ovdje utječe na izgled svih komponenti unutar aplikacije.

Unutar **assets** direktorija nalaze se dva dodatna direktorija: *slike* i *skripte*. Unutar direktorija *slike* smješteni su logotip i ikone za skripte, dok su u direktoriju *skripte* smještene markdown datoteke koje aplikacija učitava i parsira.

Direktorij **components** sadrži dodatne direktorije za daljnju kategorizaciju komponenti s obzirom na to da ih ima pedeset i jedna (51): *Admin*, *App*, *Exam* i *Global*. Direktorij *Admin* sadrži komponente vezane za administrativno sučelje, dok direktorij *Exam* sadrži sve komponente vezane za rješavanje ispita. U direktoriju *App* nalaze se sve komponente koje se nalaze u datoteci `App.vue`, dok se u direktoriju *Global* nalaze sve komponente koje se koriste po cijeloj aplikaciji. Svi ovi direktoriji sadrže dodatne poddirektorije za još precizniju kategorizaciju.

Router direktorij sadrži datoteku `index.js`. Datoteka `index.js` unutar `router` direktorija u Vue.js aplikaciji služi za definiranje ruta aplikacije. Ovdje se konfigurira `vue-router`, koji omogućuje navigaciju između različitih prikaza ili komponenti aplikacije.

U direktoriju **scripts** nalaze se dodatne skripte koje se koriste unutar aplikacije:

- `firebase.js`: Sadrži konfiguraciju za Firebase, Firestore, autentifikaciju te sve uvoze iz Firebasea koji se koriste u aplikaciji.
- `helpers.js`: Sadrži pomoćne funkcije poput `codeCopy()`, `formatDateTime()`, `formatTime()` i `extractFromFile()`.

- `images.js`: Definirani su svi uvozi za slike za lakši rad sa slikama.
- `imports.js`: Definirane su sve komponente unutar aplikacije kako bi datoteka `main.js` bila manja i preglednija.
- `messages.js`: Definirane su sve poruke za aplikaciju za debugiranje i obavještavanje.
- `stores.js`: Definirane su sve store datoteke unutar aplikacije kako bi datoteka `main.js` bila manja i preglednija.
- `unitTesting.js`: Sadrži uvoze za sve datoteke koje služe za testiranje evaluacije kôda unutar aplikacije.
- `validators.js`: Sadrži validacijske funkcije za inpute poput: `croatianAlpha`, `containsAlpha`, `containsNumeric`, `isUnipuEmail`, `getFirstErrorForField`, `exactLength`.

Direktorij **stores** sadrži sve Pinia store datoteke:

- `adminStore.js`: Samo administrator može koristiti ovaj store. Sadrži varijable i funkcije za stvaranje, ažuriranje i brisanje ispita, ažuriranje vidljivosti skripti, dohvaćanje i ažuriranje korisnika te dohvaćanje i ocjenjivanje predanih rješenja.
- `examStore.js`: Sadrži varijable i funkcije za dohvaćanje, započinjanje i rješavanje ispita.
- `fileStore.js`: Sadrži varijable i funkcije za učitavanje skripti te preuzimanje i učitavanje datoteka.
- `globalStore.js`: Sadrži varijable i funkcije za upravljanje izgledom aplikacije i obavještavanje korisnika.
- `userStore.js`: Sadrži varijable i funkcije za prijavu, autentifikaciju i odjavu korisnika.

Unutar **views** direktorija nalaze se svi prikazi koji odgovaraju jednoj ruti u aplikaciji:

- `EC_AdminView.vue`: Stranica za administratora gdje se vrši ažuriranje i brisanje ispita, ažuriranje vidljivosti skripti, dohvaćanje i ažuriranje korisnika te dohvaćanje i pregled predanih rješenja.
- `EC_BatchExaminationView.vue`: Stranica za administratora gdje se vrši skupno ocjenjivanje ispita.
- `EC_ExamInProgressView.vue`: Stranica u koju administrator može ući dok studenti pišu ispit.

- `EC_ExamView.vue`: Stranica u kojoj korisnici dohvaćaju i pišu ispite ili vježbaju u sandbox okruženju.
- `EC_HomeView.vue`: Stranica za administratora koja sadrži sve poveznice na druge stranice.
- `EC_IntroductionView.vue`: Početna stranica aplikacije koja sadrži opis i upute za aplikaciju.
- `EC_SignIn.vue`: Stranica za prijavu u desktop verziju aplikacije.
- `EC_UnderMaintenanceView.vue`: Stranica koja je prikazana kada je u tijeku održavanje aplikacije (*eng. under maintenance*).
- `EC_UploadExamView.vue`: Stranica za administratora za učitavanje

  novog ispita.

Sve `.vue` datoteke osim `App.vue` imaju prefiks `EC_` radi lakšeg raspoznavanja i korištenja vlastitih datoteka.

4.2.2 Korišteni npm paketi

Aplikacija koristi niz npm paketa kako bi osigurala različite funkcionalnosti i optimizirala razvojni proces. Sljedeći paketi su instalirani i koriste se unutar aplikacije:

- **@headlessui/tailwindcss**: Ovaj paket omogućuje brzu izradu responzivnih i pristupačnih korisničkih sučelja koristeći komponente bez zadanih stilova, što daje slobodu prilagodbe dizajna prema specifičnim potrebama aplikacije [39].
- **@vueform/slider**: Vue komponenta za prilagodljive klizače, koja se koristi za unos numeričkih vrijednosti [40].
- **@vuelidate/core**: Jezgra Vuelidate paketa za validaciju podataka unutar Vue komponenti, omogućuje dodavanje validacija formi, osiguravajući da korisnički unosi budu ispravni i u skladu s postavljenim pravilima [41].
- **@vuelidate/validators**: Kolekcija predefiniranih validatora za Vuelidate, sadrži niz gotovih validacijskih funkcija koje pokrivaju najčešće potrebe, poput provjere email adresa, minimalne duljine teksta, numeričkih vrijednosti i drugih [42].
- **crypto-random-string**: Pruža funkcionalnost za generiranje kriptografski sigurnih slučajnih nizova. Ova funkcija je korisna za generiranje sigurnih tokena, lozinki i drugih slučajnih podataka koji zahtijevaju visoku razinu sigurnosti [43].

- **firebase**: Integracija s Firebaseom, koja omogućuje autentifikaciju, pohranu podataka, hosting i druge usluge [44].
- **pinia**: State management library za Vue.js, koji omogućuje centralizirano upravljanje stanjem aplikacije. Pinia je moderna zamjena za Vuex, pružajući jednostavniji API i bolje performanse [45].
- **pinia-plugin-persistedstate**: Pinia plugin za perzistentno pohranjivanje stanja. Ovaj plugin omogućuje automatsko spremanje i učitavanje stanja aplikacije iz lokalne pohrane, osiguravajući da se stanje zadrži između osvježavanja stranice i ponovnog pokretanja aplikacije [46].
- **vue-json-pretty**: Komponenta za lijepo prikazivanje JSON podataka, formatira i prikazuje JSON podatke na čitljiv i strukturiran način, olakšavajući pregled i analizu podataka [47].
- **vue-router**: Routing library za Vue.js aplikacije, omogućuje navigaciju između različitih prikaza, upravljanje rutama, dinamične rute, čuvanje stanja i zaštitu ruta [48].
- **vue-showdown**: Komponenta za pretvaranje Markdowna u HTML koristeći Showdown parser, omogućuje pretvaranje Markdown sadržaja u HTML unutar Vue aplikacije, podržavajući različite ekstenzije i prilagodbe [49].
- **ace-builds**: Ugrađena podrška za Ace Editor, koja se koristi za pisanje kôda unutar aplikacije s podrškom za isticanje sintakse, auto-complete i druge napredne značajke [50].
- **vue3-ace-editor**: Komponenta za integraciju Ace Editora s Vue 3 aplikacijama [51].
- **vue3-colorpicker**: Komponenta za odabir boja unutar Vue 3 aplikacija s podrškom za različite formate boja i prilagodbe [52].
- **@vuepic/vue-datepicker**: Vue komponenta za odabir datuma, s podrškom za različite formate i lokalizaciju [53].
- **vuedraggable**: Komponenta za drag-and-drop funkcionalnost unutar aplikacije [54].

4.2.3 Web aplikacija

Kod razvoja web aplikacije potrebno je uzeti u obzir ograničenja i probleme na koje se može naići. Jedan od njih je neovlašten pristup specifičnim web stranicama.

Web aplikacija EduCoder koristi *vue-router* za prikaz *single page* stranica. S obzirom na to da sve web aplikacije imaju URL linkove, korisnici mogu direktno upisati specifičan link stranice aplikacije i pristupiti joj. U tom slučaju potrebno je uvesti zaštitu ruta.

Aplikaciju koriste samo profesori i studenti sa specifičnim email nastavcima "@student.unipu.hr" i "@unipu.hr". Samo korisnici s tim email adresama mogu se prijaviti i koristiti aplikaciju, dok su ostali korisnici ograničeni na početnu stranicu aplikacije. Ako neregistrirani korisnici pokušaju pristupiti drugim stranicama, aplikacija će ih preusmjeriti natrag na početnu stranicu.

Isto vrijedi i za korisnike i administratore. Samo administratori mogu pristupiti administrativnim stranicama i funkcionalnostima aplikacije, dok ostali korisnici budu preusmjereni natrag na početnu stranicu te nemaju pristup administrativnim opcijama i funkcionalnostima. Svaki korisnik sa studentskim računom može se prijaviti kao običan korisnik, dok se administratore ručno postavlja unutar baze podataka. Samo vlasnik Firebase baze može postavljati i uklanjati administratore.

Jedna od glavnih funkcionalnosti web aplikacije je omogućavanje korisnicima pisanja i evaluacije kôda tijekom vježbanja ili ispita. Web aplikacija je povezana samo na Firebase, koji koristi isključivo kao bazu podataka, bez zasebnog backend sustava. Evaluacija JavaScript kôda kojeg korisnik napiše provodi se pomoću JavaScript `eval()` funkcije.

`eval()` funkcija u JavaScriptu izvršava kôd unutar stringa, omogućujući dinamičku evaluaciju izraza. Iako je moćna, `eval()` se općenito ne preporučuje zbog sigurnosnih rizika i potencijalnog utjecaja na performanse. Korištenje `eval()` može omogućiti izvršavanje zlonamjernog kôda ako se ulazni podaci ne sanitiziraju pažljivo.

Kako bi se smanjili rizici povezani s korištenjem `eval()` funkcije, evaluacija kôda u EduCoder aplikaciji provodi se unutar *iframe* elementa. *iframe* pruža izolirano okruženje za izvršavanje kôda, čime se sprječava da potencijalno zlonamjerni ili pogrešni kôd utječe na ostatak aplikacije [55]. Ovaj pristup omogućuje sigurno i učinkovito izvršavanje korisničkog kôda, čime se osigurava da aplikacija ostaje stabilna i sigurna.

Studenti mogu varati na mnogo načina, no najvažnije je spriječiti otvaranje drugih web stranica poput ChatGPT-a radi generiranja odgovora, kao i spriječiti kopiranje kôda izvan aplikacije. Za rješavanje ovih problema koristi se JavaScript *blur* event listener.

Blur event listener detektira kada prozor aplikacije izgubi fokus, što može značiti da je korisnik otvorio novu karticu ili prozor. Kada se to dogodi, aplikacija može poduzeti mjere kao što su obavještavanje korisnika ili zaključavanje ispita. Iako je *blur* event koristan, nije uvijek učinkovit jer korisnici mogu pronaći načine da zaobiđu ovu zaštitu, na primjer, putem virtualnih strojeva ili promjenom fokusiranih prozora na načine koje *blur* event ne detektira [56].

Kako bi se dodatno osiguralo da studenti ne izlaze slučajno iz aplikacije i ne aktivi-

raju *blur()* metodu putem različitih tipkovnih kratica, koristi se *keydown* event listener. *Keydown* event listener detektira pritisak tipki na tipkovnici i može spriječiti neželjene kombinacije tipki korištenjem *preventDefault()* metode, dopuštajući samo određene kombinacije tipki koje su potrebne za ispit [57].

Iako *keydown* event može spriječiti mnoge tipkovne prečace koji bi mogli ometati fokus aplikacije, također nije uvijek potpuno učinkovit. Napredni korisnici mogu pronaći načine da zaobiđu ove mjere, koristeći prilagođene skripte ili promjene u postavkama sustava. Također, previše restriktivna implementacija može ometati normalno korištenje aplikacije i frustrirati korisnike.

Unatoč ovim mjerama, ovi pristupi imaju svoje nedostatke. *Blur* event može biti zaobiđen, a *keydown* event može frustrirati korisnike ograničavanjem njihovog normalnog rada. Zbog toga je važno balansirati sigurnosne mjere s upotrebljivošću aplikacije i kontinuirano tražiti poboljšanja kako bi se osigurao integritet ispita.

```
1      window.addEventListener("blur", antiCheatDetection);
2
3      window.addEventListener('keydown', function(event) {
4          if (event.ctrlKey && (event.key === 'A' || event.key === 'C'
               )) {
5              event.preventDefault();
6          }
7      });
```

Slika 30: *Primjer implementacije JavaScript blur i keydown event listenera (izvor: Autor)*

Jedan od dodatnih problema koji može utjecati na sigurnost web aplikacije su ekstenzije preglednika. Ekstenzije preglednika mogu imati značajan utjecaj na ponašanje aplikacije i mogu zaobići sigurnosne mjere implementirane unutar aplikacije [58]. Neke od mogućnosti koje ekstenzije pružaju uključuju:

- **Promjena ponašanja preglednika**: Ekstenzije mogu mijenjati način na koji preglednik rukuje određenim događajima, kao što su *blur* i *keydown* eventi, čime mogu zaobići sigurnosne mjere aplikacije.
- **Pristup podacima**: Ekstenzije mogu imati pristup podacima koji se unose u aplikaciju, omogućujući korisnicima kopiranje kôda ili drugih informacija koje nisu dopuštene.
- **Izmjena DOM-a**: Neke ekstenzije mogu mijenjati DOM strukturu aplikacije, što može uzrokovati nepredviđene probleme u radu aplikacije.
- **Automatizacija zadataka**: Ekstenzije mogu automatizirati određene zadatke ili radnje unutar aplikacije, što može uključivati izvršavanje skripti ili preuzimanje sadržaja bez dozvole.

Kada se studenti prijave u aplikaciju, njihove informacije o prijavi spremaju se u kolačiće (*eng. cookies*). Aplikacija koristi kolačiće za spremanje cijelog stanja aplikacije za trenutnog korisnika, omogućujući im da pri izlasku i ponovnom ulasku nastave rad gdje su stali. Međutim, ovakav pristup ima svoje probleme.

Jedan od problema je sigurnost kolačića. Kolačići su podložni napadima poput krađe identiteta (*eng. session hijacking*) i cross-site scripting (XSS) napada, što može ugroziti sigurnost korisničkih podataka. Također, prevelika ovisnost o kolačićima može dovesti do problema s performansama, posebno kod velikih količina podataka. Konačno, kolačići mogu biti izbrisani ili blokirani od strane korisnika, što može rezultirati gubitkom podataka ili prekidom rada aplikacije. Implementacija sigurnosnih mjera, poput enkripcije kolačića i korištenja sigurnih komunikacijskih kanala (HTTPS), može pomoći u smanjenju rizika povezanih s upotrebom kolačića [59].

Kao što je ranije spomenuto, jedina moguća evaluacija kôda unutar aplikacije je evaluacija JavaScript kôda. Evaluacija kôda pisanog u Pythonu, C++ i drugim programskim jezicima nije podržana. Ovo ograničenje, s ostalim navedenim nedostatcima i problemima, motiviralo je razvoj desktop aplikacije.

4.2.4 Desktop aplikacija

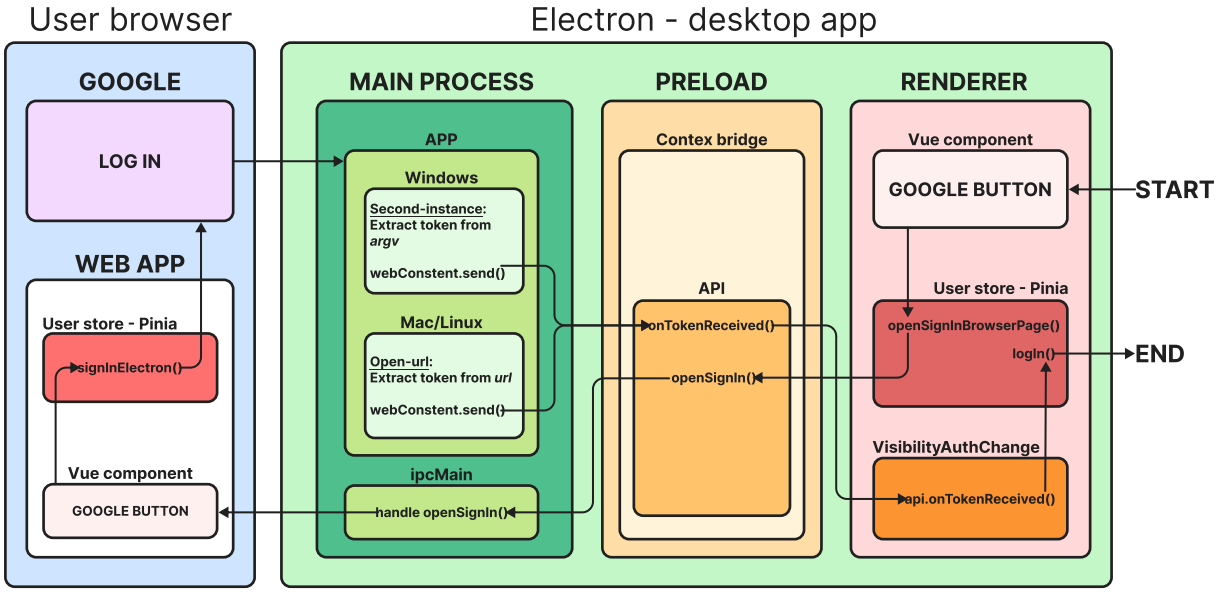
Razvoj desktop aplikacije je općenito složeniji od razvoja web aplikacije. Međutim, s obzirom na to da se za razvoj ove aplikacije koristi Electron, veći dio arhitekture preuzet je iz web aplikacije. Electron omogućuje izradu desktop aplikacija pomoću web tehnologija kao što su HTML, CSS i JavaScript, ali s dodatnim mogućnostima koje web aplikacije nemaju, kao što je komunikacija s operativnim sustavom preko Node.js-a.

Desktop verzija aplikacije koristi Vite kao alat za razvoj i izgradnju aplikacije, prema vodiču dostupnom na [60]. Projektna struktura inspirirana je primjerima iz [61] te slijedi standardnu arhitekturu Electron aplikacije koja uključuje tri glavne komponente: *main*, *preload* i *renderer* procese.

- **Main proces**: Ovaj proces djeluje kao glavni proces koji upravlja cijelom aplikacijom. Main proces odgovoran je za kreiranje i upravljanje prozorima aplikacije. Također omogućuje komunikaciju između različitih dijelova aplikacije i rukovanje sistemskim događajima kao što su otvaranje i zatvaranje aplikacije. U main procesu obično se nalaze Node.js moduli, što omogućuje pristupanje sistemskim resursima i izvođenje operacija na razini operativnog sustava.
- **Preload proces**: Preload skripte se učitavaju prije nego što renderer proces dobije kontrolu. To omogućuje da se određeni kôd izvršava s višim privilegijama prije nego što korisničko sučelje postane aktivno. Preload skripte koriste se za izlaganje ograničenog broja sigurnih API-ja renderer procesu, čime se povećava sigurnost aplikacije.

- **Renderer proces**: Renderer procesi odgovorni su za prikazivanje korisničkog sučelja aplikacije. Oni učitavaju HTML, CSS i JavaScript te prikazuju sadržaj prozora aplikacije. Renderer procesi su izolirani i nemaju izravni pristup operativnom sustavu, što povećava sigurnost aplikacije. Komunikacija između renderer i main procesa ostvaruje se putem *Inter-Process Communication* (IPC) modula.

Kada se korisnik prijavljuje u web aplikaciju, postupak je vrlo jednostavan i temelji se na pozivu Firebase metode *signInWithPopup()*. Međutim, nijedna Firebase funkcija za prijavu putem Google računa ne radi u Electron desktop verziji aplikacije. Firebase Auth službeno ne podržava Electron. Budući da je Electron kombinacija preglednika i poslužiteljskog okruženja Node.js, neke značajke možda neće raditi kako se očekuje, poput prethodno spomenute *signInWithPopup()* metode. Da bi se omogućila prijava putem Google računa, potrebno je nekoliko dodatnih koraka, koji su prikazani na slici 31.



Slika 31: *Proces prijave putem Google računa u Electron aplikaciji (izvor: Autor)*

Kada se pokrene Electron aplikacija, grafičko sučelje koje se prikazuje dio je *renderer* procesa aplikacije. Za pokretanje procesa prijave potrebno je kliknuti na Vue komponentu *Google button* koja iz *user store* Pinia poziva metodu *openSignInBrowserPage()*. Ta metoda zatim preko *window API* poziva metodu *openSignIn()* iz *preload* dijela aplikacije, koja potom poziva *handleOpenSignIn()* metodu iz glavnog procesa *ipcMain* (*eng. main process*). Ova metoda koristi *shell* za otvaranje web stranice specifične za prijavu u desktop Electron aplikaciju.

U web aplikaciji nastavljamo s prijavom, ponovno klikom na Vue komponentu *Google button* koja iz *user store* Pinia ovaj put poziva metodu *signInElectron()*. Budući da se sada poziv ove funkcije odvija u web aplikaciji, možemo koristiti *signInWithPopup()*, međutim, ovaj put dohvaćamo *oauthIdToken*. Kada se taj token dohvati, korisnik

se preusmjerava na stranicu koja ponovno poziva Electron desktop aplikaciju, ali ovog puta s proslijeđenim Google autentifikacijskim tokenom.

Tehnički se može preskočiti početni korak s Electronom i izravno pristupiti web aplikaciji za prijavu, jer ako desktop aplikacija nije pokrenuta, otvorit će se pri prijavi. Glavni proces desktop Electron aplikacije čeka na token. Ovisno o operativnom sustavu, koristi različite metode za dohvat tokena. Za Windows koristi *app.on('second-instance')*, dok za Linux i Mac koristi *app.on('open-url')*; obje metode dohvaćaju i izvlače proslijeđeni token.

Nakon toga, token se prosljeđuje u *preload* dio aplikacije, gdje *ipcRenderer.on* sluša *sendToken* poziv metode i hvata token koji se zatim prosljeđuje u *renderer* dio aplikacije u metodu *onTokenReceived()*. Ta metoda iz *user store* Pinia poziva funkciju *logIn()* s proslijeđenim tokenom, čime se korisnik uspješno prijavljuje u aplikaciju.

Evaluacija korisničkog kôda unutar aplikacije ne vrši se pomoću funkcije *eval*, već se izvršava u zasebnom procesu. Taj proces omogućava hvatanje ispisa, rezultata i grešaka bez utjecaja na glavnu aplikaciju. Izvođenje kôda u zasebnom procesu osigurava veću sigurnost i stabilnost, jer sprječava potencijalne probleme koji bi mogli nastati izravnim izvršavanjem korisničkog kôda u glavnom tijeku aplikacije.

U Electron aplikaciji, evaluacija kôda se vrši na računalu korisnika, koristeći izolirane procese. To znači da se korisnički kôd izvršava u odvojenom okruženju koje ima pristup resursima računala, ali je odvojeno od glavne aplikacije. Ovaj pristup omogućava pokretanje kôda napisanog u različitim programskim jezicima kao što su Python i C++, koristeći odgovarajuće sustavske pozive i biblioteke.

Primjerice, kôd napisan u Pythonu može se evaluirati pokretanjem Python interpretatora unutar zasebnog procesa, dok se kôd napisan u C++ može kompilirati i izvršiti pomoću odgovarajućih alata za taj programski jezik. Ovo omogućava fleksibilnost u podršci različitih programskih jezika unutar iste aplikacije.

Izolirani procesi za evaluaciju kôda pružaju nekoliko ključnih prednosti:

1. **Sigurnost**: Izvođenje kôda u izoliranom procesu smanjuje rizik od zlonamjernih napada koji bi mogli iskoristiti ranjivosti u funkciji *eval*. Budući da se korisnički kôd ne izvršava izravno u glavnom tijeku aplikacije, smanjuje se mogućnost neovlaštenog pristupa ili oštećenja podataka.

2. **Stabilnost**: Izolirani procesi osiguravaju da eventualne greške ili padovi korisničkog kôda ne utječu na rad glavne aplikacije. Ako dođe do pogreške unutar izoliranog procesa, samo taj proces će biti pogođen, dok će glavna aplikacija nastaviti raditi bez prekida.

3. **Fleksibilnost**: Korištenje izoliranih procesa omogućava podršku za različite programske jezike. Tako aplikacija može evaluirati kôd napisan u više jezika, čime se povećava njena svestranost i korisnost.

4. **Pristup resursima računala**: Izolirani procesi imaju mogućnost pristupa resursima računala, što omogućava izvršavanje složenijih zadataka koji zahtijevaju direktan pristup hardverskim resursima ili specifičnim sustavskim uslugama.

4.3 Sučelje aplikacije

Sučelje aplikacije predstavlja vizualni i funkcionalni aspekt koji korisnici koriste za interakciju s EduCoder aplikacijom. U ovom poglavlju opisat ćemo glavne elemente sučelja aplikacije, uključujući korisničko sučelje za korisnike i administratore te ključne funkcionalnosti sučelja koje im omogućuju obavljanje zadataka.

Dizajn sučelja aplikacije trebao bi pratiti Nielsenove heuristike (Nielsen's Heuristics) kako bi se osigurala visoka upotrebljivost i intuitivno korisničko iskustvo [62]. Nielsenove heuristike uključuju sljedećih deset (10) principa:

1. **Vidljivost sustava**: Korisnici bi trebali biti informirani o trenutnom stanju sustava kroz odgovarajuće povratne informacije u razumnom vremenskom roku.

    - *Primjer*: Svaka radnja koja vrši neki poziv prema bazi (prijava, započinjanje ispita, dohvaćanje i ažuriranje podataka) obavještava korisnika s vizualnim povratnim informacijama.

2. **Podudaranje između sustava i stvarnog svijeta**: Sučelje bi trebalo koristiti jezik i koncepte poznate korisnicima, umjesto tehničkog žargona.

    - *Primjer*: Pravilno korištenje boja (crvena za prekid/izlaz/grešku, narančasta za upozorenje ili trenutačno u tijeku, zelena za započinjanje ili preuzimanje, plava za obavijesti i funkcije) te ikone koje predstavljaju funkcionalnosti koje obavljaju.

3. **Korisnička kontrola i sloboda**: Korisnici često griješe te im treba omogućiti jednostavan način za poništavanje radnji ili izlazak iz neželjenih stanja.

    - *Primjer*: Sprječavanje korisnika da slučajno izađe iz aplikacije s pomoću dodatnog upozorenja, mogućnost povratka u aplikaciju i poništavanje/vraćanje pisanja kôda.

4. **Konzistentnost i standardi**: Korisnici ne bi trebali biti prisiljeni pitati se znače li različite riječi, situacije ili radnje isto. Konzistentnost u terminologiji, bojama i ikonama je ključna.

    - *Primjer*: Radi konzistentnosti aplikacija je limitirana na četiri boje koje se uvijek koriste za slične funkcionalnosti, korištenje istih ikona za dohvat, brisanje i preuzimanje, konzistentnost veličine teksta i fonta.

5. **Prevencija pogrešaka**: Dizajn bi trebao spriječiti pojavljivanje pogrešaka koliko god je to moguće, umjesto oslanjanja na obavijesti o greškama.

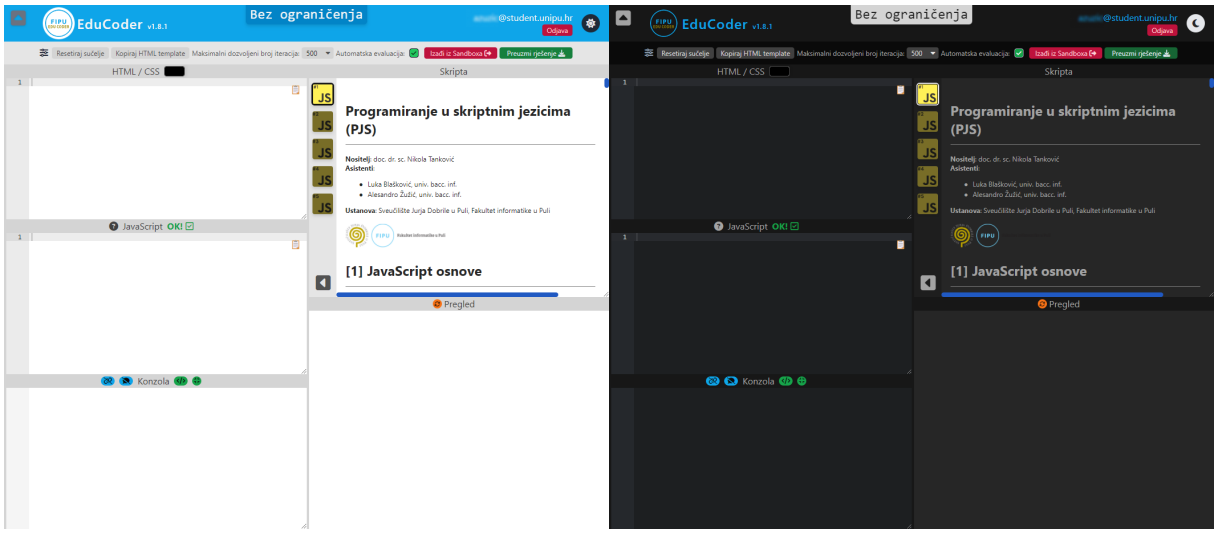
    - *Primjer*: Sprječavanje korisnika da ne izađe iz aplikacije preko tipkovnih kratica i sprječavanje blokiranja aplikacije kada korisnik napiše kôd koji se neograničeno izvršava.

6. **Prepoznavanje, a ne prisjećanje**: Minimiziranje opterećenja korisničke memorije prikazivanjem objekata, radnji i opcija. Korisnici ne bi trebali pamtiti informacije iz jednog dijela dijaloga do drugog.

    - *Primjer*: Većina funkcionalnosti označena je labelama ili prelaskom miša dodatno naglašava svoju funkcionalnost.

7. **Fleksibilnost i učinkovitost korištenja**: Sučelje bi trebalo omogućiti brži rad za iskusne korisnike, npr. kroz upotrebu tipkovnih prečaca.

    - *Primjer*: Korisnici si mogu sami podešavati sučelje, povećavati svaki prozor i tekst unutar aplikacije, mijenjati teme uređivača, birati između svijetle i tamne teme aplikacije (vidi sliku 32), povećavati font.



Slika 32: *Svijetla i tamna tema aplikacije (izvor: Autor)*

8. **Estetski i minimalistički dizajn**: Sučelje bi trebalo biti jednostavno, bez suvišnih informacija koje mogu smanjiti jasnoću.

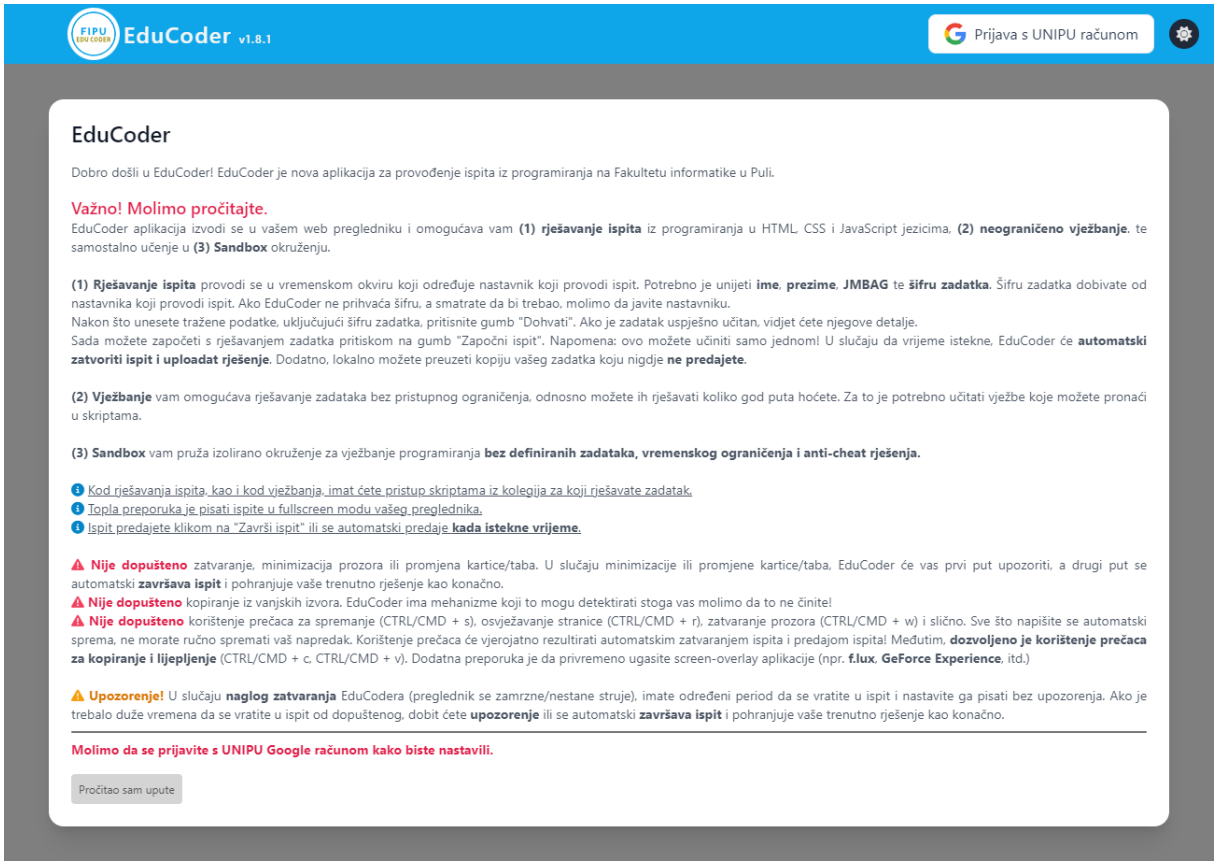
    - *Primjer*: Samo ključne opcije nalaze se na sučelju koje korisnik često koristi poput evaluacije kôda, kopiranja kôda i osvježavanja prozora, dok su rjeđe korištene funkcionalnosti poput tema uređivača, prikaza, micanja i uklanjanja prozora sakrivene unutar dodatnog izbornika opcija.

9. **Pomozite korisnicima prepoznati, dijagnosticirati i oporaviti se od pogrešaka**: Poruke o pogreškama bi trebale biti izražene u običnom jeziku, precizno naznačujući problem i predlažući konstruktivno rješenje.

    - *Primjer*: Prilikom pisanja kôda, korisnici su obaviješteni o greškama u pisanju po utvrđenim standardima s jasno definiranim bojama i ikonama.

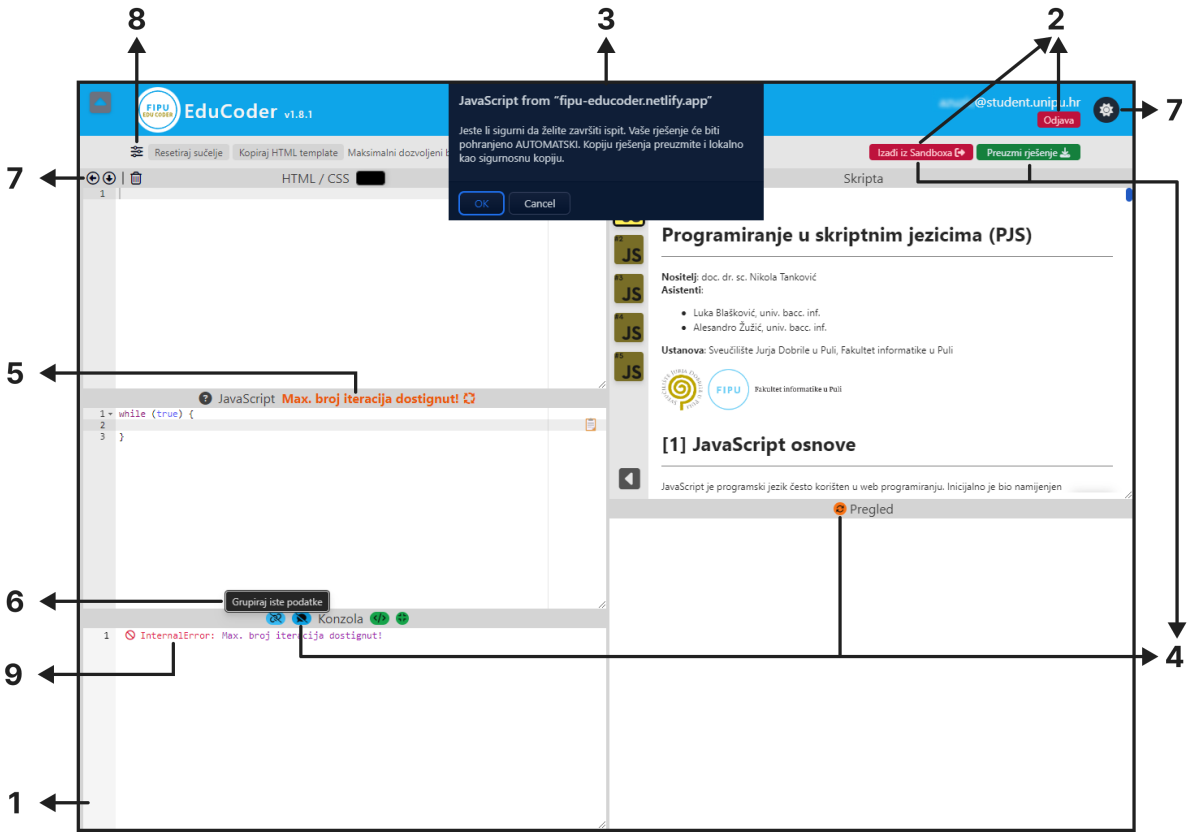
10. **Pomoć i dokumentacija**: Iako je najbolje ako sustav može biti korišten bez dokumentacije, možda će biti potrebno pružiti pomoć i dokumentaciju. Takva informacija bi trebala biti jednostavna za pretraživanje, fokusirana na korisničke zadatke i osigurati konkretne korake za izvršenje tih zadataka.

    - *Primjer*: Pri ulasku u aplikaciju, na početnoj stranici nalaze se sve upute i objašnjenja korištenja aplikacije (vidi sliku 33).



Slika 33: *Educoder upute (izvor: Autor)*

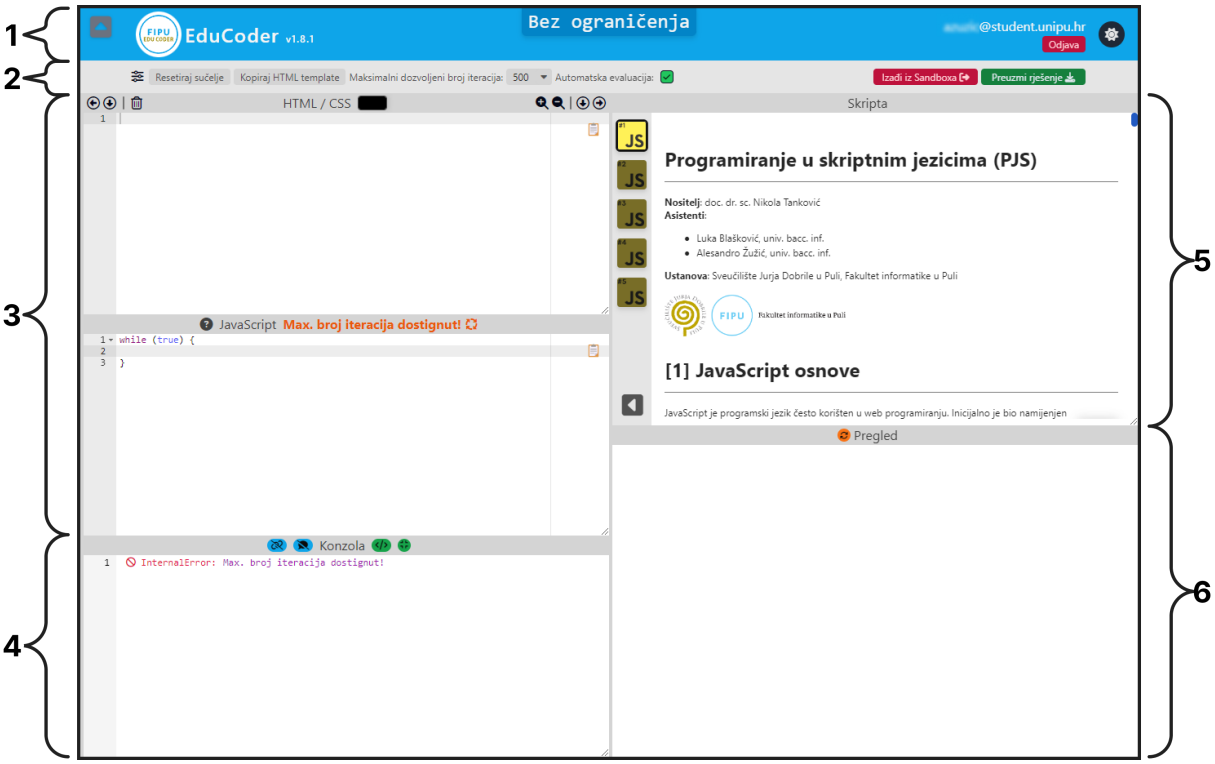
Primjeri Nielsenovih heuristika u EduCoderu prikazani su na slici 34.



Slika 34: *Primjeri Nielsenovih heuristika u EduCoderu (izvor: Autor)*

4.3.1 Korisničko sučelje

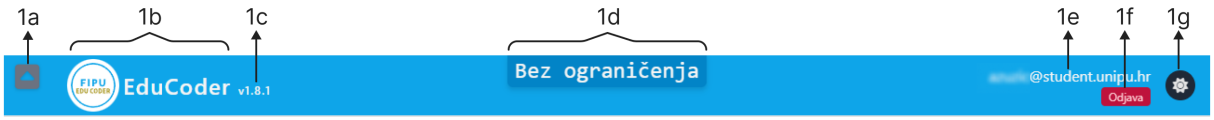
Korisnici prvo pristupaju stranici *Introduction* (vidi sliku 33), gdje mogu pročitati upute za korištenje aplikacije i izvršiti prijavu. Nakon prijave, korisnici nastavljaju na stranicu `exam`, gdje mogu rješavati ispite ili vježbati u *Sandbox* modu.



Slika 35: *Korisničko sučelje korisnika - stranica* `exam` *(izvor: Autor)*

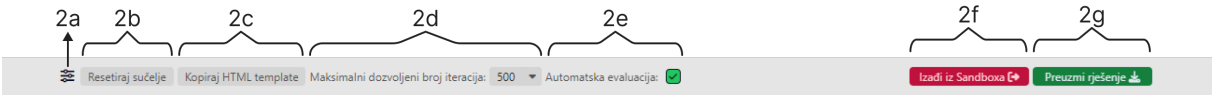
Slika 35 prikazuje glavne elemente stranice u *sandbox* modu `exam`:

1. **Zaglavlje**: Na vrhu sučelja nalazi se glavno zaglavlje aplikacije s dugmetom za sakrivanje zaglavlja `1a`, logotipom `1b` i verzijom aplikacije `1c`, prikazom vremenskog ograničenja `1d`, emailom trenutnog korisnika `1e`, gumbom za odjavu `1f` te gumbom za promjenu teme cijele aplikacije `1g` (vidi sliku 36).



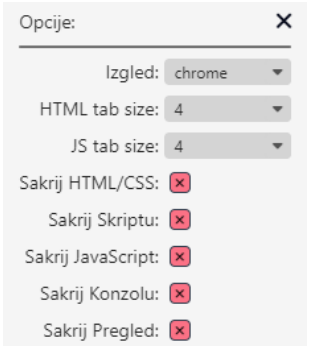
Slika 36: *Korisničko sučelje - zaglavlje (izvor: Autor)*

2. **Alatna traka**: Ispod zaglavlja se nalazi alatna traka s dugmetom za otvaranje dodatnih opcija `2a`, dugmetom za vraćanje sučelja na početni izgled `2b`, dugmetom za kopiranje HTML šablone `2c`, padajućim izbornikom za namještanje maksimalnog broja iteracija kod evaluacije korisničkog kôda `2d`, potvrdnim okvirom za uključivanje automatske evaluacije korisničkog kôda `2e`, dugmetom za izlazak iz *sandbox* okruženja `2f` te dugmetom za preuzimanje napisanog korisničkog kôda `2g` (vidi sliku 37).



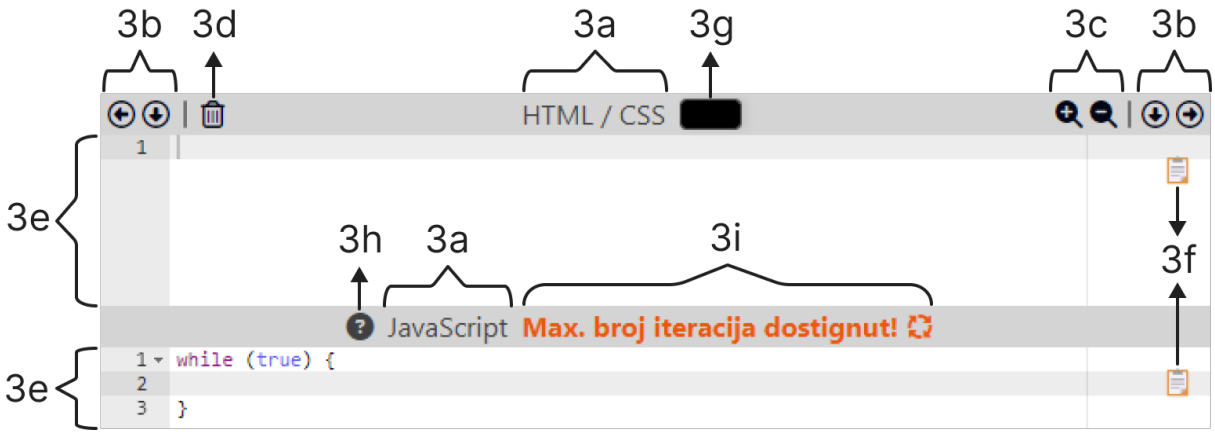
Slika 37: *Korisničko sučelje - alatna traka (izvor: Autor)*

- Dodatne opcije `2a` prikazane na slici 38 sadrže padajući izbornik izgleda uređivača, padajući izbornik za veličinu HTML i JS razmaka kod pritiska `Tab` tipke te potvrdne okvire za prikaz prozora (HTML/CSS, Skripta, JavaScript, Konzola i Pregled).



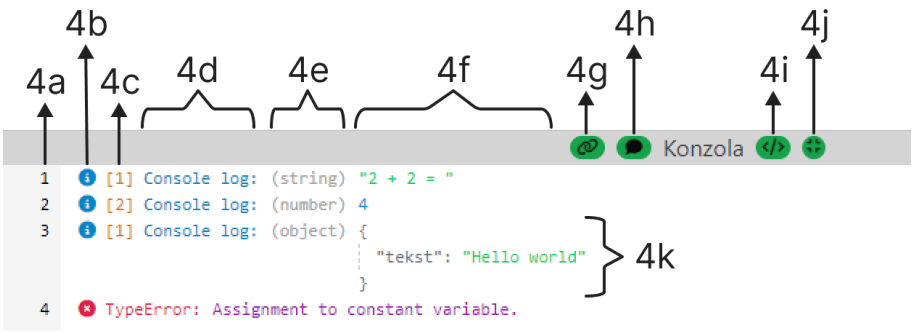
Slika 38: *Korisničko sučelje - dodatne opcije (izvor: Autor)*

3. **Prozori HTML/CSS i JavaScript**: Na lijevoj polovici sučelja nalaze se prozori *HTML/CSS* i *JavaScript* za pisanje kôda. Svi prozori sadrže naslov `3a`, dugmad za micanje (lijevo, dolje, gore, desno) ovisno o poziciji `3b`, dugmad za povećanje i smanjenje sadržaja prozora `3c` te dugme za uklanjanje prozora `3d`. Oba prozora sadrže okvir za pisanje kôda `3e` te dugme za kopiranje napisanog kôda `3f`. *HTML/CSS* prozor dodatno sadrži dugme za biranje boja `3g`, dok *JavaScript* prozor sadrži dugme za prikaz pomoćnih uputa `3h` te tekst za prikaz stanja evaluiranog kôda `3i` (vidi sliku 39).



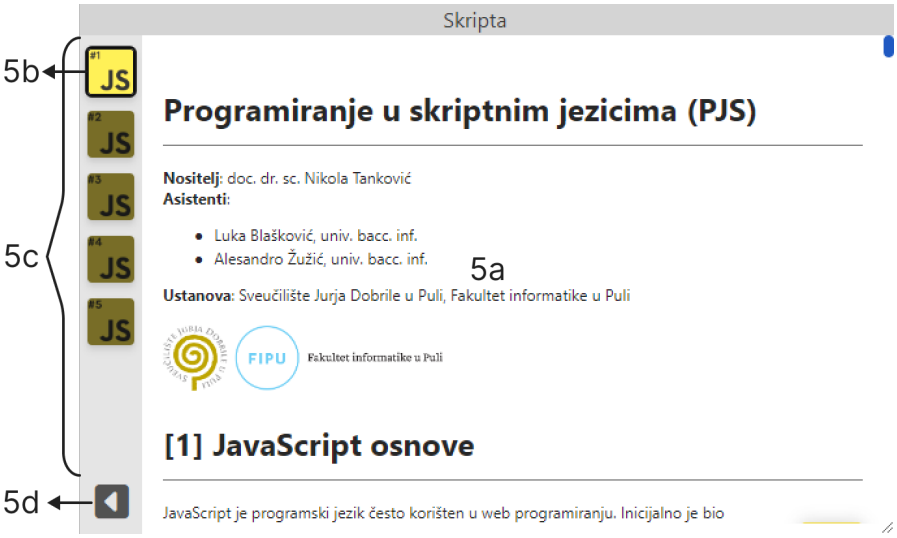
Slika 39: *Korisničko sučelje - prozori HTML/CSS i JavaScript (izvor: Autor)*

4. **Prozor konzola**: Također na lijevoj polovici sučelja nalazi se prozor *konzola* koja ispisuje sve poruke izvršavanja kôda. Prva kolona `4a` predstavlja redni broj ispisa poruke, druga kolona `4b` pokazuje ikonu vrste ispisa, treća kolona `4c` pokazuje grupirani broj istih ispisa, četvrta kolona `4d` pokazuje tekst vrste ispisa, peta kolona `4e` pokazuje tekst vrste podatka ispisa, šesta kolona `4f` pokazuje podatke ispisa. Dodatno sadrži još i dugme za grupiranje istih ispisa `4g`, dugme za prikaz tipa podataka `4h`, dugme za formatiranje ispisa `4i` te dugme za proširivanje ili smanjivanje formatiranog ispisa `4j` prikazanog kod ispisa objekta `4k` (vidi sliku 40).



Slika 40: *Korisničko sučelje - prozor konzola (izvor: Autor)*

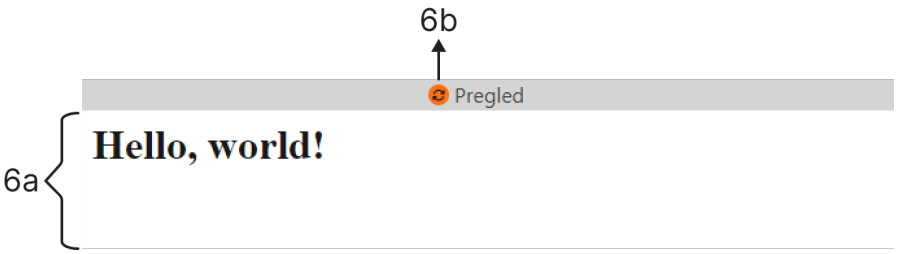
5. **Prozor skripta**: Na desnoj polovici sučelja nalazi se prozor *skripta* gdje korisnik pregledava `5a` trenutno odabranu skriptu `5b`. S desne strane prozora se nalazi izbornik vidljivih skripti koje korisnik može pregledavati `5c`. Izbornik također sadrži dugme za skrivanje izbornika `5d` (vidi sliku 41).



Slika 41: *Korisničko sučelje - prozor skripta (izvor: Autor)*

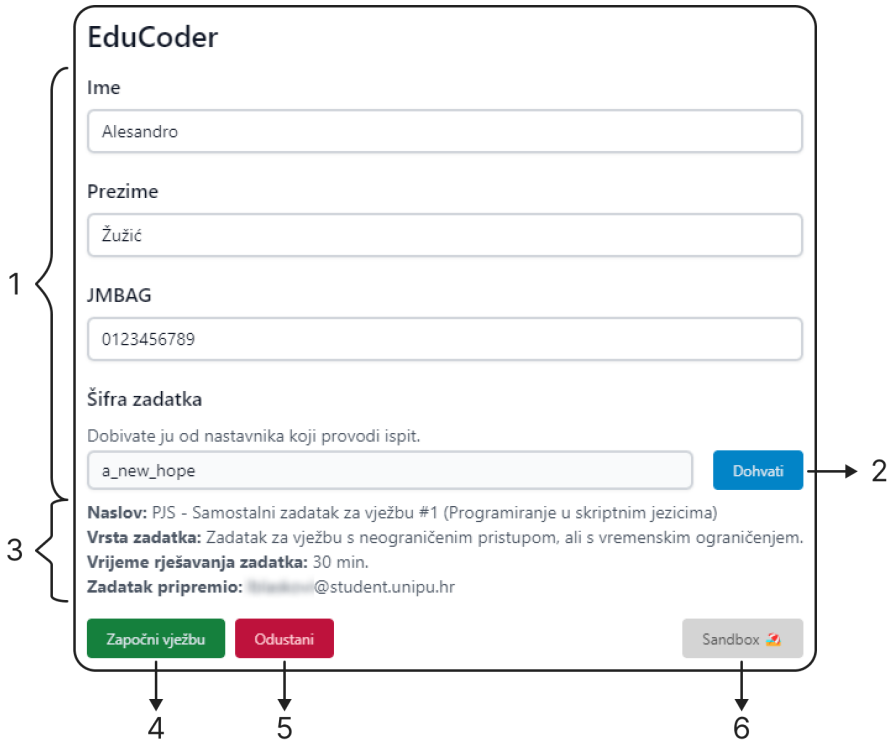
6. **Prozor pregled**: Također na desnoj polovici sučelja nalazi se prozor *pregled* gdje korisnik vidi prikaz napisanog *HTML/CSS/JavaScript* kôda `6a`. Prozor sadrži

i dugme za osvježavanje prikaza 6b (vidi sliku 42).



Slika 42: *Korisničko sučelje - prozor pregled (izvor: Autor)*

Kada korisnik izađe iz *sandbox* okruženja otvara se skočni prozor za učitavanje i započinjanje ispita (vidi sliku 43).



Slika 43: *Korisničko sučelje - skočni prozor za učitavanje i započinjanje ispita (izvor: Autor)*

Korisnik zatim može popuniti formu sa svojim podacima 1, upisati šifru ispita te dohvatiti ispit 2. Kada se ispit dohvati, prikažu se svi podaci o ispitu 3 (naziv, vrsta zadatka, vrijeme rješavanja zadatka, tko je pripremio zadatak) te korisnik može započeti s pisanjem ispita 4 ili odustati od pisanja ispita 5, u tom slučaju se može vratiti u *sandbox* okruženje 6.

Kada korisnik počne pisati ispit, ispod alatne trake se pojavi ispitna traka (vidi sliku 44), koja sadrži padajući izbornik za označavanje statusa (Nije započet - crvena, U tijeku - narančasta, Završen - zelena) zadatka 1, listu zadataka ispita 2 te dugme za završetak ispita 3.

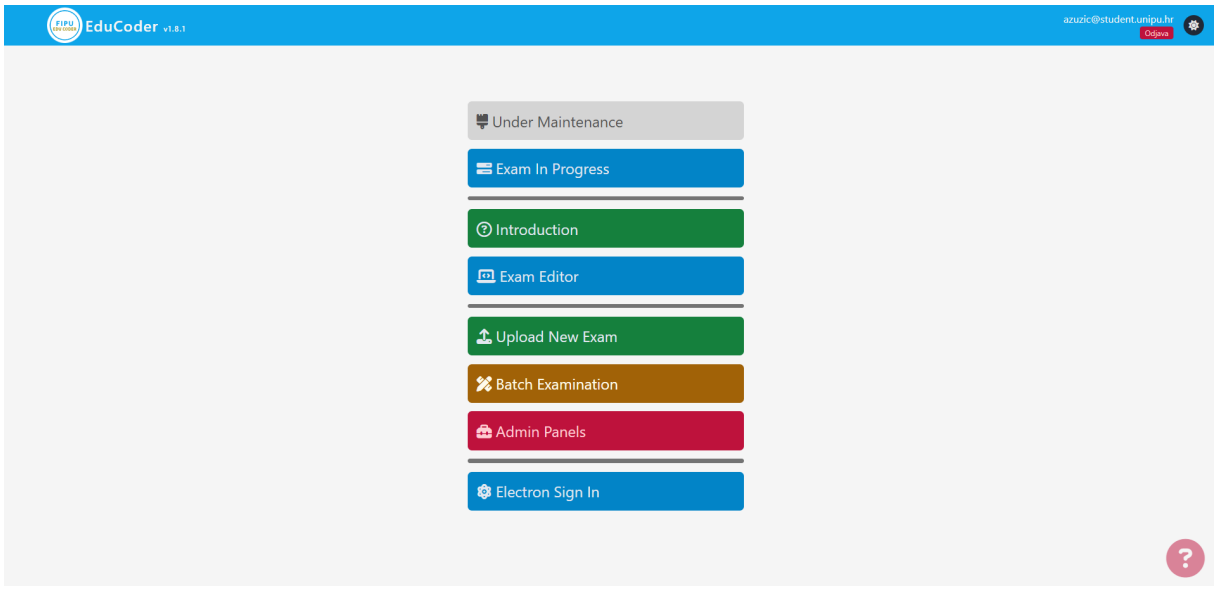
Slika 44: *Korisničko sučelje - ispitna traka (izvor: Autor)*

Kada je aplikacija u održavanju, onda su svi korisnici preusmjereni na stranicu *održavanje u tijeku* (vidi sliku 45) i nemaju pristup nijednoj drugoj stranici.

Slika 45: *Korisničko sučelje - održavanje u tijeku (izvor: Autor)*

4.3.2 Administrativno sučelje

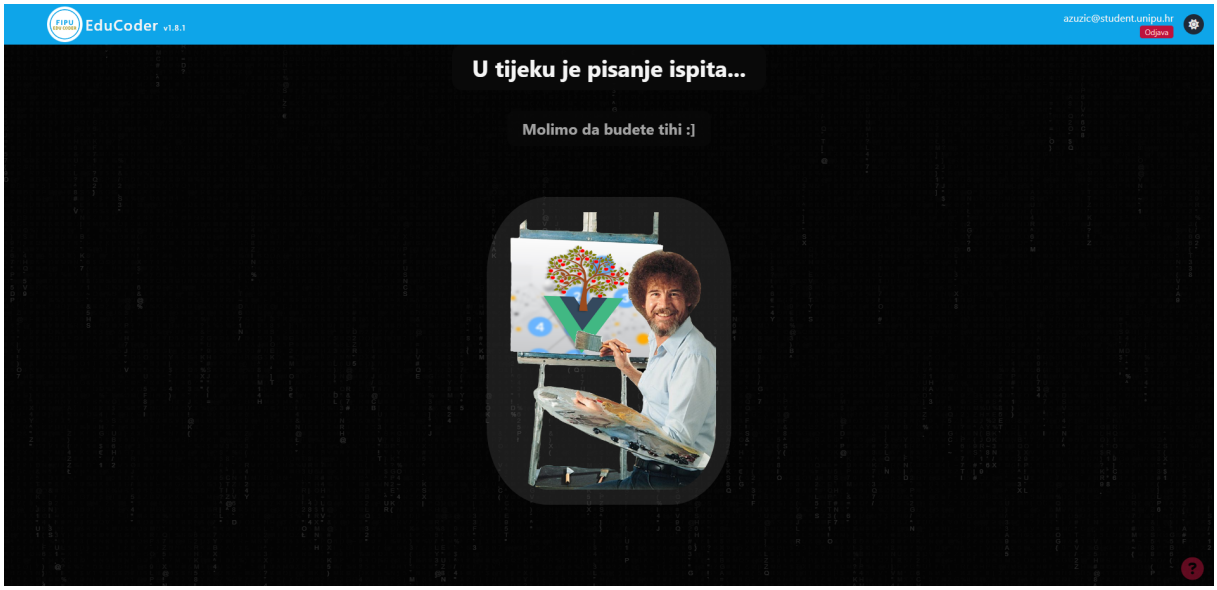
Administrator ima pristup svim stranicama kao i korisnik. Ako u zaglavlju klikne na logo EduCodera, pristupa *Home* stranici aplikacije (vidi sliku 46).



Slika 46: *Administrativno sučelje - početna stranica (izvor: Autor)*

Na ovoj stranici nalaze se dugmadi koji vode na sve druge stranice u aplikaciji:

- *Under Maintenance* - stranica koja se prikazuje svim korisnicima kada je održavanje aplikacije u tijeku (vidi sliku 45).
- *Exam In Progress* - stranica u koju administrator može ući dok studenti pišu ispit (vidi sliku 47).

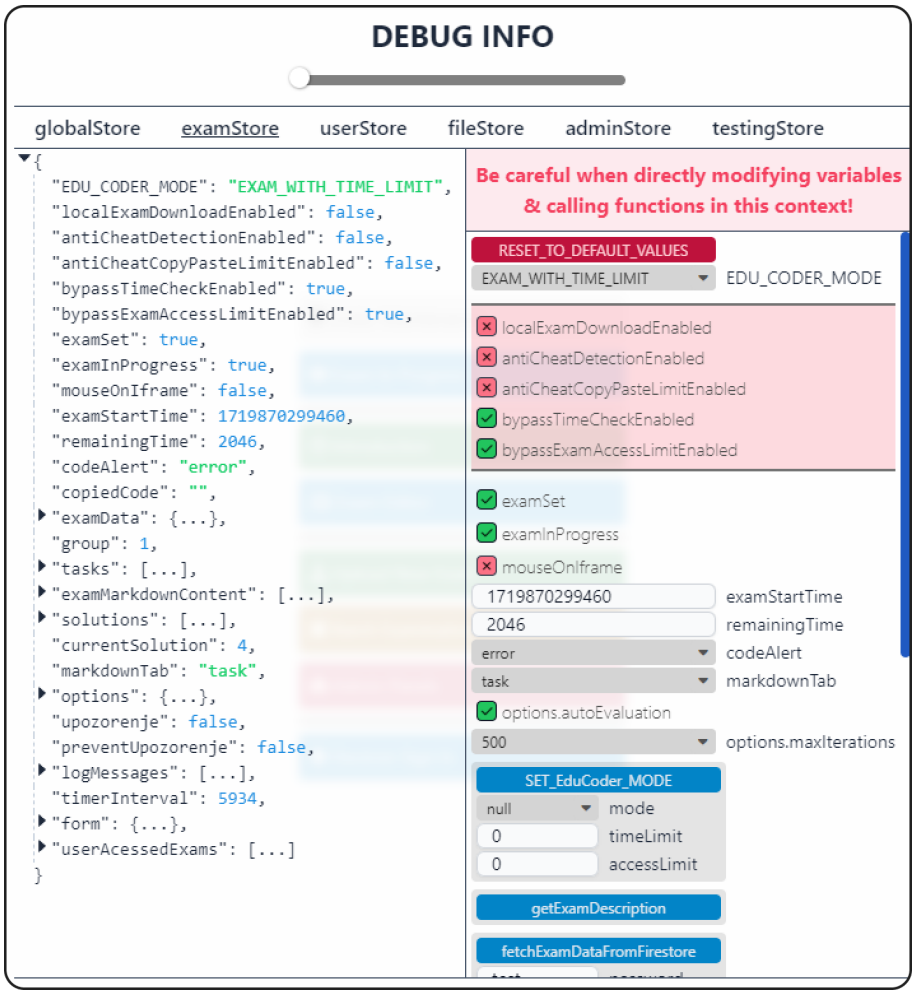


Slika 47: *Administrativno sučelje - ispit u tijeku (izvor: Autor)*

- *Introduction* - početna stranica za korisnike gdje se nalaze informacije i upute o EduCoderu.
- *Upload New Exam* - stranica na kojoj administrator stvara novi ispit.

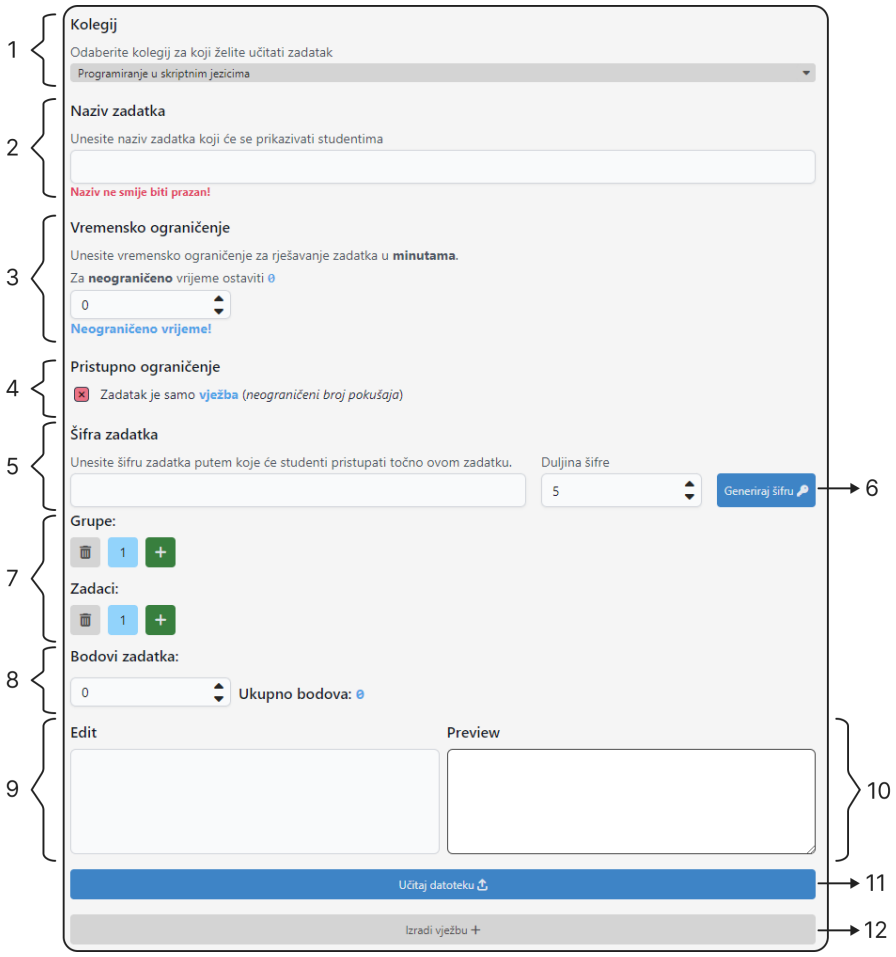
- *Batch Examination* - stranica na kojoj administrator skupno ispravlja ispite.
- *Admin Panels* - stranica na kojoj administrator pristupa i upravlja bazom aplikacije.
- *Electron Sign In* - stranica preko koje se korisnici prijavljuju u desktop verziju aplikacije.

U donjem desnom kutu slike 46 nalazi se crveni upitnik koji služi za otvaranje skočnog prozora za otklanjanje pogrešaka. Prikazuje sve Pinia storeove, njihove varijable te nudi mogućnost uređivanja varijabli i pozivanja funkcija (vidi sliku 48). Dugme za otklanjanje pogrešaka dostupno je na svakoj stranici.



Slika 48: *Administrativno sučelje - skočni prozor za otklanjanje pogrešaka (izvor: Autor)*

Na slici 49 prikazana je stranica *Upload New Exam* gdje administrator izrađuje novi ispit.



Slika 49: *Administrativno sučelje - stranica za izradu novog ispita (izvor: Autor)*

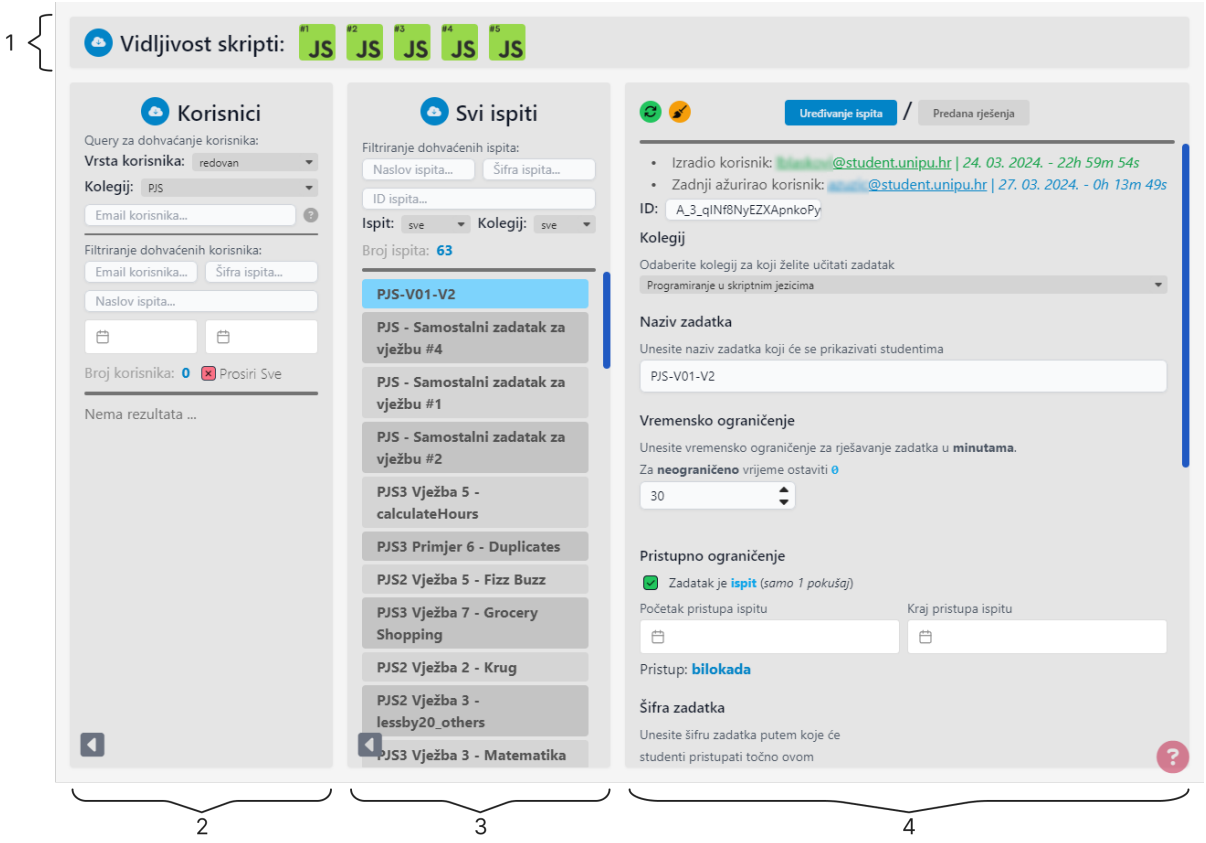
Izrada novog ispita započinje odabirom kolegija (Programiranje u skriptnim jezicima ili Programsko inženjerstvo) iz padajućeg izbornika koji služi za kategorizaciju ispita 1. Zatim se upisuje naziv ispita 2, koji će biti prikazan studentima. Naziv ne mora biti jedinstven, ali mora biti zadan. Sljedeće je postavljanje vremenskog ograničenja 3 u minutama. Vremensko ograničenje određuje koliko vremena studenti imaju za riješiti ispit; ako se postavi na nula, vrijeme rješavanja je neograničeno. Zatim se postavlja pristupno ograničenje 4 preko potvrdnog okvira. Pristupno ograničenje naznačuje vrstu ispita (vježba ili ispit). Ako je označeno kao ispit, korisnici imaju samo jedan pokušaj za riješiti ispit; u suprotnom, ako nije označeno, onda je postavljeno kao vježba te korisnici imaju neograničen broj pokušaja rješavanja vježbe.

Nadalje se postavlja šifra ispita putem koje korisnici dohvaćaju i pristupaju ispitu 5. Šifra mora biti jedinstvena tako da se pri unosu šifre provjerava postoji li već ispit s istom šifrom zadatka. Šifra se također može automatski generirati pritiskom na dugme *Generiraj šifru* koje generira šifru po postavljenoj duljini šifre 6.

Zatim se stvaraju grupe ispita i zadaci za svaku grupu. Može minimalno postojati jedna grupa i jedan zadatak po svakoj grupi 7. Za svaki zadatak se postavljaju bodovi, te aplikacija automatski računa i prikazuje ukupan broj bodova za označenu grupu 8.

Zadatak se uređuje u *Edit* prozoru u markdown obliku 9, gdje je u *Preview* prozoru prikazan stvarni izgled zadatka koji će biti prikazan korisnicima 10.

U slučaju da je administrator već napisao sve grupe i zadatke u zasebnoj datoteci na računalu prateći unaprijed definiranu strukturu pisanja ispita, može kliknuti na dugme *Učitaj datoteku* 10, preko kojeg unosi datoteku s računala u aplikaciju koja parsira tekst datoteke i automatski postavlja grupe, zadatke grupa, bodove i tekst za svaki zadatak. Kada je sve popunjeno, administrator može izraditi novi ispit 11.



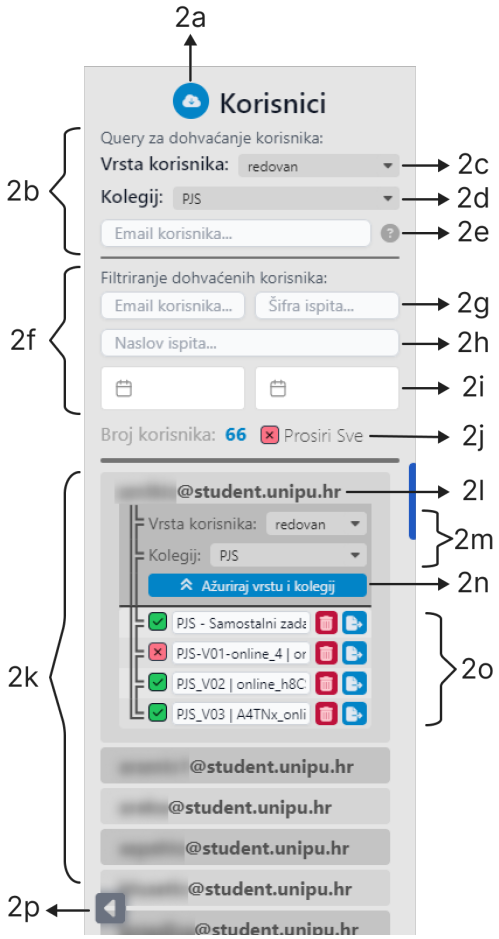
Slika 50: *Administrativno sučelje - admin panels stranica (izvor: Autor)*

Glavna stranica koju administrator koristi je *Admin Panels*, slika 50 prikazuje glavne elemente stranice:

1. **Vidljivost skripti**: Na vrhu sučelja nalazi se prozor *Vidljivost skripti* (vidi sliku 51), gdje administrator može dohvatiti vidljivost skripti iz baze 1a. Kada dohvati vidljivost, svaku skriptu može kliknuti i ažurirati njenu vidljivost. Ako je skripta zelena, znači da je vidljiva, a ako je crvena, znači da nije vidljiva korisnicima 1b.

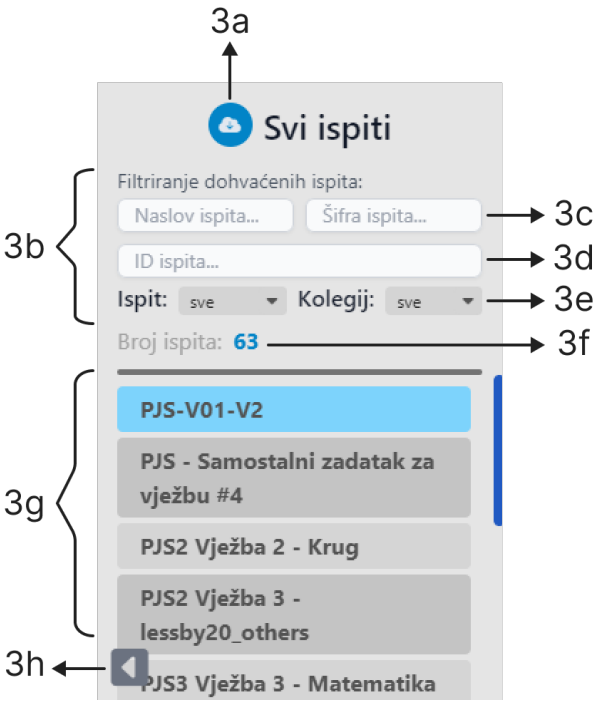
Slika 51: *Administrativno sučelje - vidljivost skripti (izvor: Autor)*

2. **Korisnici**: Na lijevoj strani sučelja nalazi se prozor *Korisnici* (vidi sliku 52), gdje administrator može dohvatiti korisnike iz baze `2a`. S obzirom na to da dohvaćanje korisnika skupa funkcija, dohvaćanje korisnika se može filtrirati preko upita (*eng. query*) `2b` koristeći padajuće izbornike *Vrsta korisnika* (online, redovan, sve, ostalo) `2c` i *Kolegij* (PJS, PI, sve, ostalo) `2d`. Ako se odabere *sve*, specifični filtar se zanemaruje. Također se mogu filtrirati po emailu, gdje se u polje za unos pišu početna slova emailova koje se želi filtrirati `2e`. Kada se korisnici dohvate, mogu se pretraživati `2f` po emailu, po šifri ispita kojima su pristupili `2g`, po naslovu ispita kojima su pristupili `2h` te po periodu pristupanja ispitu `2i`. Korisnici su prikazani u obliku stupca `2j`. Svaki se može proširiti za daljnji pregled i ažuriranje. Kod filtra je prikazan broj korisnika te se svi korisnici mogu proširiti odjednom preko potvrdnog okvira `2k`. Svakom korisniku je prikazan email `2l` te vrsta korisnika i kolegij `2m` koje administrator može promijeniti i ažurirati `2n`. Dodatno, za svakog korisnika je prikazana lista ispita kojima su pristupili. Za svaki ispit se može namjestiti pristupno ograničenje preko potvrdnog okvira, obrisati pristupljen ispit iz liste ili pregledati predano rješenje `2o`. U donjem lijevom kutu prozora nalazi se dugme za skrivanje prozora `2p`.



Slika 52: *Administrativno sučelje - korisnici (izvor: Autor)*

3. **Svi ispiti**: Na sredini sučelja nalazi se prozor *Svi ispiti* (vidi sliku 53), gdje administrator može dohvatiti sve ispite iz baze `3a`. Kada su ispiti dohvaćeni, mogu se filtrirati `3b` po naslovu, šifri `3c` i ID-ju `3d`, koristeći padajuće izbornike za vrstu ispita i kolegij `3e`. Ispiti su prikazani u obliku stupca, a odabrani ispit je označen plavom bojom `3g`. Ispod filtriranja prikazan je ukupan broj ispita u stupcu `3f`. U donjem lijevom kutu prozora nalazi se dugme za skrivanje prozora `3h`.



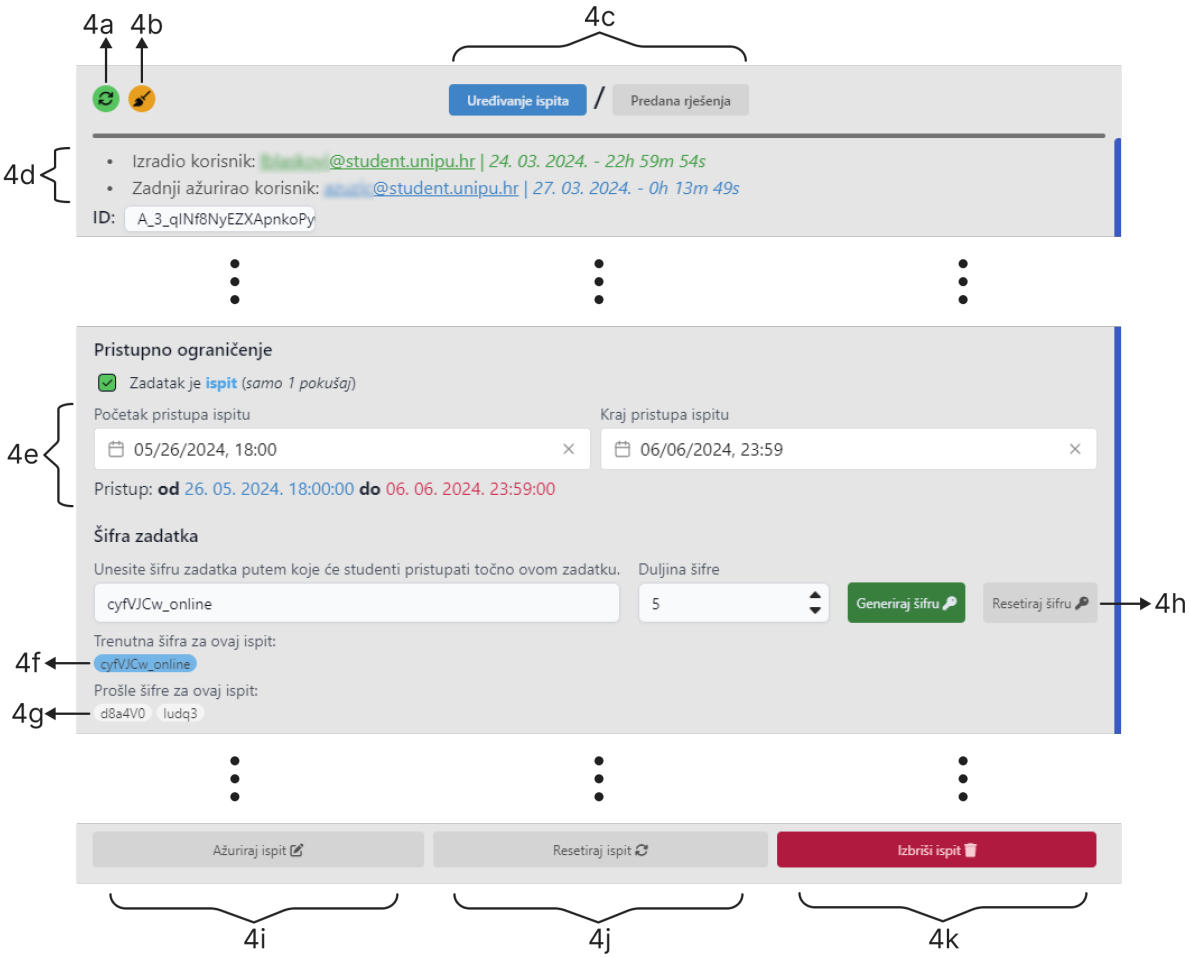
Slika 53: *Administrativno sučelje - svi ispiti (izvor: Autor)*

4. **Uređivanje ispita i predana rješenja**: Na desnoj strani sučelja nalazi se odabran prozor *Uređivanje ispita* (vidi sliku 54), gdje administrator može uređivati sve podatke ispita. U gornjem lijevom kutu nalaze se dva dugmeta: jedno za dohvaćanje najnovije verzije kako bi se spriječilo dupliciranje ispita `4a`, i drugo za odznačivanje ispita `4b`. Na vrhu prozora nalaze se dugmad *Uređivanje ispita* i *Predana rješenja* koja omogućuju prebacivanje između ta dva prozora `4c`.

   Ispit ima dodatne informacije o datumu kada je stvoren, tko ga je stvorio, datum zadnjeg ažuriranja i tko ga je zadnji ažurirao. Administrator može dodatno postaviti početak i kraj pristupa ispitu `4e`. Kada se mijenja šifra ispita, postojeća se ne briše već se sprema u polje prošlih šifri za ispit `4f`, dok je trenutna aktivna šifra naglašena `4g`. Nova šifra ne može biti jednaka staroj šifri; u tom slučaju, stara šifra postaje aktivna šifra. Šifra se može vratiti na prijašnju šifru pritiskom na dugme *Resetiraj šifru* `4h`.

   Kada su unesene željene promjene, ispit se može ažurirati u bazi podataka `4i`. Ako se želi vratiti na prijašnje stanje, moguće je pritisnuti dugme za vraćanje `4j`

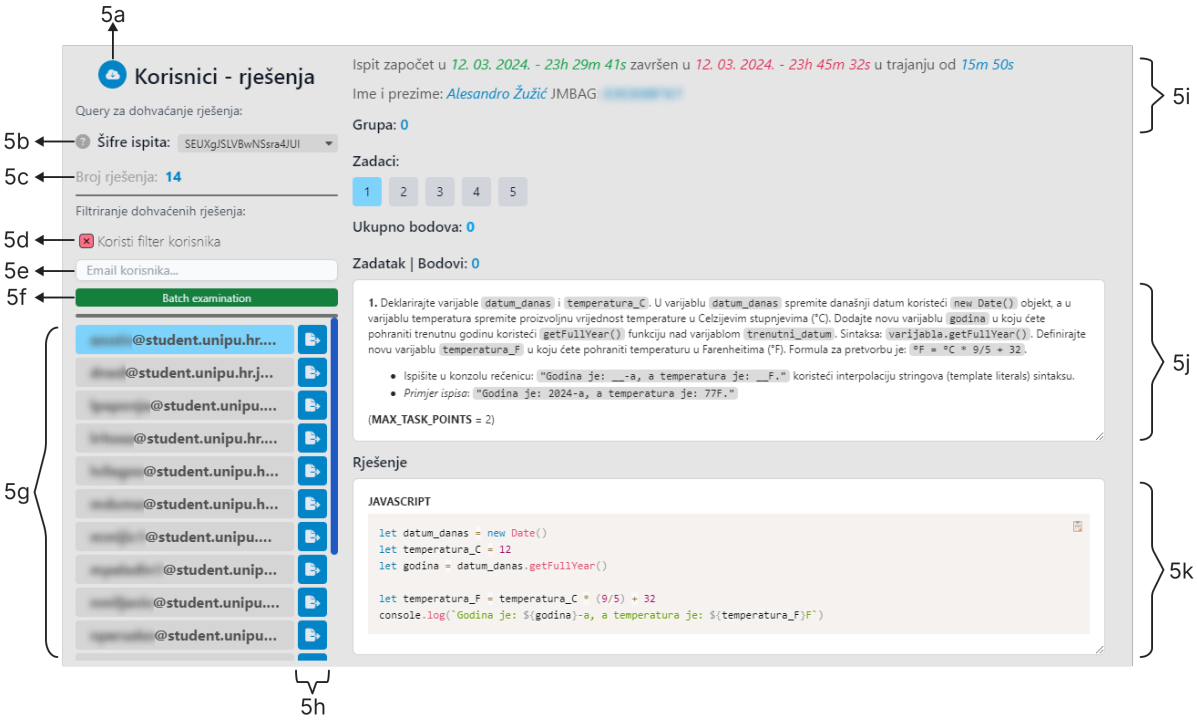
ili izbrisati ispit iz baze podataka sa svim šiframa pritiskom na dugme *Obriši* 4k.



Slika 54: *Administrativno sučelje - uređivanje ispita (izvor: Autor)*

Ako je odabran prozor *predana rješenja* (vidi sliku 55), administrator može dohvaćati 5a i pregledavati 5g sva predana rješenja po odabranoj šifri iz padajućeg izbornika 5b. Dodatno može filtrirati dohvaćena rješenja koristeći filtar za korisnike preko potvrdnog okvira 5d ili filtrirati po emailu korisnika 5e, gdje se onda prikaže broj filtriranih rješenja 5c. Administrator može odabrati i pregledavati predano rješenje što je naznačeno plavom bojom, te sva predana rješenja može skupno ocjenjivati pritiskom na dugme *Batch examination* 5f ili pojedinačno ocijeniti ispit 5h.

U desnom dijelu prozora nalaze se podaci predanog rješenja: kada je ispit započet, završen, trajanje, te ime, prezime i JMBAG korisnika predanog rješenja 5i. Također je prikazana grupa ispita te svi zadaci s tekstom zadatka 5j i predanim rješenjem za odabrani zadatak 5k.

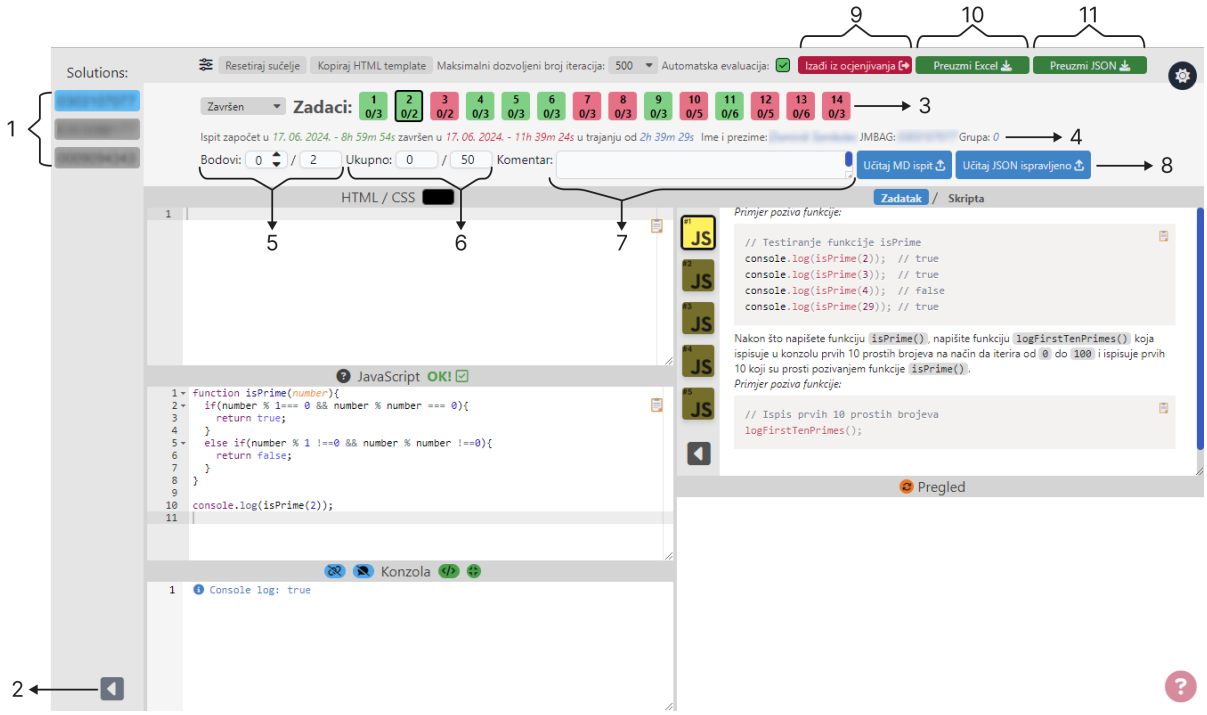


Slika 55: *Administrativno sučelje - predana rješenja (izvor: Autor)*

Zadnja stranica kojoj administrator može pristupiti je *Batch examination* (vidi sliku 56), gdje administrator skupno ispravlja sva predana rješenja ispita za odabranu šifru. Na lijevoj strani sučelja nalazi se izbornik predanih rješenja 1 koje administrator može odabrati te je označeno plavom bojom. Izbornik se može sakriti pritiskom dugmeta u donjem lijevom kutu 2.

Kada je predano rješenje odabrano, administrator vidi sve zadatke ispita 3, njihov status i broj bodova za svaki (na početku svi zadaci imaju nula bodova). Ispod zadataka nalaze se informacije o predanom rješenju 4. Administrator može unositi bodove za svaki zadatak 5, gdje svaki zadatak ima definiran maksimalni broj bodova iz ispita. Istovremeno se pri ocjenjivanju ažurira ukupan broj postignutih bodova ispita 6.

Za svaki zadatak administrator može napisati komentar 7. Također, administrator može učitati dvije datoteke 8: markdown ispit koji sadrži dodatne upute za ispravljanje ispita te JSON datoteku prethodno spremljenog stanja ocjenjivanja. Kada je administrator završio s ocjenjivanjem, može preuzeti dvije datoteke: JSON datoteku trenutnog stanja ocjenjivanja 11 i Excel datoteku ispravljenih predanih rješenja 10. Na kraju može izaći iz skupnog ocjenjivanja pritiskom na dugme 9.



Slika 56: *Administrativno sučelje - skupno ispravljanje ispita (izvor: Autor)*

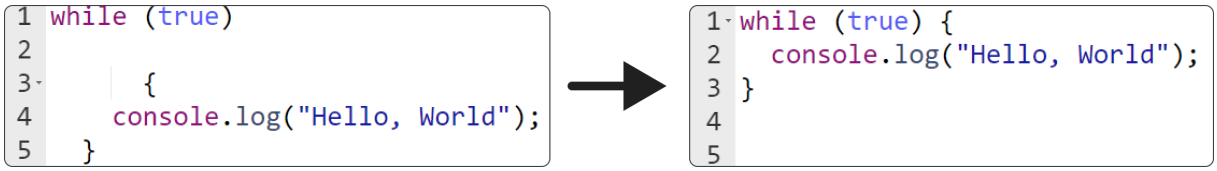
4.4 Implementacija ključnih funkcionalnosti

U ovom poglavlju opisat ćemo implementaciju ključnih funkcionalnosti EduCoder aplikacije koje osiguravaju njezinu učinkovitost i pouzdanost. Ove funkcionalnosti uključuju detaljnu evaluaciju kôda, započinjanje i vraćanje u ispit, sinkronizaciju kod ažuriranja te optimizaciju Firebasea.

4.4.1 Evaluacija kôda

Jedan od većih problema kod razvoja web verzije aplikacije je evaluacija kôda bez zamrzavanja aplikacije, s obzirom na to da su web stranice single-threaded. Način na koji korisnici mogu zamrznut aplikaciju je pisanjem beskonačnih petlji ili beskonačnih rekurzivnih funkcija. Da bi se to spriječilo, potrebno je modificirati korisnikov kôd bez utjecaja na performanse i rezultat, već spriječiti beskonačne petlje i beskonačne rekurzivne funkcije.

S obzirom na to da JavaScript nije strog oko razmaka i novih linija, kôd se može pisati u jednoj liniji ili rastegnuti na 1000 linija kôda. Stoga, prvo što treba napraviti je modificirati kôd tako da uvijek ima istu strukturu prije daljnje analize. Traže se ključne riječi: `for`, `if`, `while`, `do`, `function` te se pravilno postavljaju vitičaste zagrade za svaku, kao što je prikazano na slici 57.

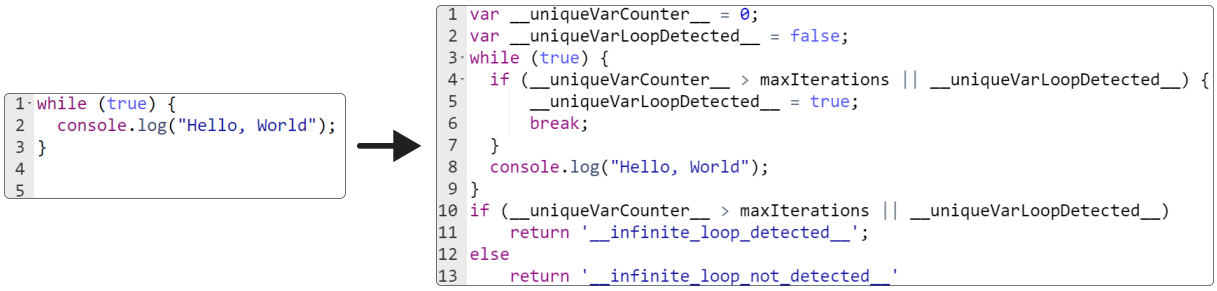


Slika 57: *Formatiranje kôda (izvor: Autor)*

JavaScript dopušta pisanje petlji `for`, `while` i `do` bez vitičastih zagrada. Stoga je potrebno provjeriti imaju li te strukture vitičaste zagrade, a ako nemaju, korisnika se na to upozorava (vidi sliku 58).

Slika 58: *Obavještavanje korisnika na nedostatak vitičaste zagrade (izvor: Autor)*

Sljedeći korak je ubacivanje dodatnog kôda u korisnikov kôd (*eng. code injection*). Na početak kôda se ubacuje brojač `f__uniqueVarCounter__` i provjera `__uniqueVarLoopDetected__`. Zatim se na početak svake petlje i funkcije ubacuju inkrementi brojača `incrementStatement` i uvjet za prekid rada petlje `exitLoopCondition` ili uvjet za izlaz iz funkcije `exitFunctionCondition`. U slučaju da brojač premaši maksimalni broj iteracija izvršavanja kôda, forsira se izlaz iz petlji i funkcija koristeći `break` i `return` respektivno, te kôd kao krajnju vrijednost vraća `__infinite_loop_detected__`. U suprotnom vraća `__infinite_not_loop_detected__` (vidi sliku 59).



Slika 59: *Ubacivanje dodatnog kôda u korisnikov kôd (izvor: Autor)*

4.4.2 Započinjanje i vraćanje u ispit

Kada korisnici žele započeti pisanje ispita, postoji mogućnost da proslijede šifru ispita nekoj drugoj osobi koja je ulogirana u aplikaciju, omogućujući joj da rješava ispit umjesto njih. Kako bi se to spriječilo, uvedena je provjera pristupa ispitu prije početka ispita, što onemogućuje istovremeno započinjanje istog ispita više puta.

S obzirom na različita računala, operacijske sustave i web preglednike koje korisnici koriste, moguća je pojava grešaka u radu aplikacije, što može dovesti do zamrzavanja aplikacije ili izbacivanja korisnika. Također, korisnici mogu ostati bez internetske veze ili struje. Zbog mogućnosti nastanka jednog od ovih slučajeva, potrebno je omogućiti korisnicima povratak u ispit. Međutim, postoji i mogućnost zlouporabe ove redundancije povratka.

Svaki pristup ispitu bilježi početak pisanja ispita i trajanje ispita, tako da se može odrediti je li korisnik pisao ispit nakon dopuštenog vremena. Cijelo vrijeme dok se piše ispit, računa se koliko je vremena prošlo i sprema se u local storage. Kada se korisnik ponovno prijavi u aplikaciju, provjerava se razlika između trenutnog vremena i onog spremljenog. Ako je razlika manja od maksimalno dozvoljene, korisnik se može normalno vratiti u aplikaciju. Ako je razlika veća od dopuštene, korisnik dobiva upozorenje ili ga se izbacuje iz ispita, te se automatski predaje rješenje ispita, ovisno o veličini vremenske razlike.

4.4.3 Sinkronizacija ažuriranja podataka

Jedan od mogućih problema koji se mogu pojaviti je kada dva administratora istovremeno ažuriraju isti ispit. U većini slučajeva to nije problem jer će novije ažuriranje pregaziti starije. Međutim, može se dogoditi da se promijeni jedan od ključnih podataka, poput šifre ispita, što može dovesti do dupliciranja ispita ili mrtvih šifri koje se mogu ukloniti samo ručno, izravno iz baze podataka.

Još jedan mogući problem nastaje kada administrator ostane prijavljen u aplikaciju i ne dohvati nove podatke, dok drugi administrator promijeni ključne podatke ispita ili izbriše ispit. U tom slučaju, može doći do dupliciranja ispita ili stvaranja dva ispita s istom šifrom. Kako bi se spriječili takvi problemi, implementirano je nekoliko mjera.

Prva mjera je dodatno spremanje vremena zadnjeg ažuriranja ispita. Svaki put pri ulasku na stranicu za uređivanje uspoređuje se trenutno vrijeme ispita s onim u bazi. Ako se ta vremena ne podudaraju, potrebno je dohvatiti novu verziju ispita, te nije moguće ažurirati ili brisati ispit dok se ne dohvati najnovija verzija. Ako je ispit ažuran i zatim ga se krene mijenjati, dok drugi administrator istovremeno mijenja ili briše ispit, aplikacija će prikazati prikladno upozorenje i onemogućiti daljnje akcije dok se ispiti ne usklade. Ova provjera dodana je prije svakog ažuriranja i brisanja ispita.

4.4.4 Optimiziranje dohvata iz baze podataka

Baza podataka se nalazi u Firebaseu koji ima određene besplatne limite, koji su sasvim dovoljni za male aplikacije koje se rijetko koriste s malim brojem korisnika. Međutim, EduCoder koristi preko 150 studenata. Jedan od problema koji se ubrzo pojavio je velika količina čitanja dokumenata iz baze (*eng. doc reads*). Mjesečni besplatni limit

je 50 000 čitanja, koji je ubrzo dosegnut. Zbog toga je bilo potrebno napraviti složeniju strukturu spremanja i dohvaćanja podataka.

S obzirom da je cilj smanjiti broj čitanja, možemo žrtvovati prostor za spremanje dodatnih struktura za lakše dohvaćanje podataka. Prije uvođenja dodatnih struktura, dodani su dodatni načini filtriranja dohvaćanja podataka (*eng. query*). Što se tiče novih struktura, jedna od njih je `createdExams` polje koje sprema osnovne podatke o stvorenim ispitima. Ovo omogućava da se prilikom dohvaćanja ispita vrši jedno čitanje umjesto onoliko koliko ima sveukupno ispita, te se dalje podaci o ispitu mogu preciznije dohvatiti s jednim čitanjem. Druga struktura je `examPasswords` rječnik gdje su spremljene sve šifre svih ispita po njihovim ID-jevima. Ovo omogućava lakše pronalaženje ispita i uspoređivanje šifri za sprječavanje stvaranja dva ispita s istom šifrom.

Osim novih struktura, stare strukture su se također ažurirale. Jedna od njih je `exam` koji prije nije imao `ID`, već samo jednu šifru, zbog čega se po jednom ispitu moglo nagomilati puno predanih rješenja. Dodatkom jedinstvenog ID-a i mogućnošću mijenjanja šifri omogućila se bolja raspodjela korisnika po grupama, što je olakšalo ocjenjivanje i poboljšalo performanse.

Sve ove promjene omogućile su smanjenje broja čitanja sa 12 000 na svega 300, što predstavlja smanjenje od 75 posto.

5 Primjena u praksi

Ovo poglavlje prikazuje konkretne primjene EduCoder aplikacije u nastavi na Fakultetu informatike u Puli. Cilj je predstaviti kako aplikacija pomaže studentima u učenju programiranja, kao i njezine prednosti i nedostatke. Također ćemo analizirati rezultate prikupljene putem ankete među korisnicima aplikacije, s fokusom na korisničko iskustvo i zadovoljstvo. Kroz studiju slučaja i analizu rezultata, pružit ćemo detaljan uvid u stvarnu upotrebu i učinkovitost EduCoder aplikacije u obrazovnom okruženju.

5.1 Studija slučaja: Primjena EduCoder aplikacije u nastavi

EduCoder web aplikacija razvijena je s ciljem unapređenja nastavnog procesa na Fakultetu informatike u Puli, posebno u kontekstu učenja programiranja. Aplikaciju su izradili studenti/asistenti Luka Blašković i Alesandro Žužić, s namjerom da se omogući studentima jednostavniji i interaktivniji način vježbanja i polaganja ispita iz programiranja.

EduCoder je integriran u nastavne programe kolegija ”Programiranje u skriptnim jezicima” i ”Programsko inženjerstvo”. Na početku semestra, studenti su upoznati s aplikacijom putem vježbi i uputa dostupnih na početnoj stranici aplikacije. EduCoder omogućuje studentima da vježbaju kodiranje u HTML, CSS i JavaScript jezicima unutar *sandbox* okruženja, koje simulira stvarno radno okruženje.

Jedna od ključnih funkcionalnosti EduCoder-a je *sandbox* okruženje koje studentima omogućava pisanje, testiranje i ispravljanje kôda u realnom vremenu. Ova funkcionalnost je posebno korisna za studente koji tek uče osnove programiranja jer im omogućava da odmah vide rezultate svog rada.

Studenti mogu pristupiti stranici *Introduction* gdje se nalaze detaljne upute za korištenje aplikacije. Nakon prijave, nastavljaju na stranicu *Exam* gdje mogu odabrati vježbu ili ispit. Vježbe su dizajnirane da budu interaktivne i progresivne, čime se studentima omogućava da postepeno savladavaju sve složenije zadatke. Također, aplikacija omogućuje studentima pristup skriptama i zadacima za vježbu, što dodatno podržava proces učenja.

Tijekom ispita, EduCoder aplikacija pruža sigurnosne mjere kako bi se spriječilo varanje. Aplikacija onemogućava kopiranje izvana, te višestruki ponovni ulazak u aplikaciju i ponovno započinjanje ispita. Studenti unose šifru ispita koju dobiju od nastavnika i pristupaju zadacima koje moraju riješiti unutar zadanog vremena.

Nastavnici i administratori imaju pristup dodatnim alatima unutar aplikacije. Oni mogu kreirati nove ispite, pregledavati i ocjenjivati predane zadatke, te upravljati korisničkim računima. Administrativno sučelje omogućava jednostavno praćenje napretka studenata i prilagodbu zadataka prema potrebama kolegija.

Korištenje EduCoder-a u nastavi donijelo je nekoliko značajnih prednosti:

- **Interaktivnost**: Studenti mogu brže vidjeti rezultate svog rada, što pomaže u bržem usvajanju gradiva.
- **Dostupnost**: Aplikacija je dostupna bilo kada i bilo gdje, omogućavajući studentima da vježbaju kodiranje izvan učionice.
- **Prilagodljivost**: Nastavnici mogu lako prilagoditi zadatke i ispite specifičnim potrebama studenata.
- **Sigurnost**: Integrirane sigurnosne mjere pomažu u očuvanju integriteta ispita.
- **Praćenje napretka**: Administrativno sučelje omogućava detaljno praćenje napretka studenata i ocjenjivanje zadataka.

Iako je EduCoder donio mnoge prednosti, tijekom njegove upotrebe identificirani su i određeni nedostaci:

- **Tehničke poteškoće**: Ponekad dolazi do tehničkih problema poput zamrzavanja aplikacije ili gubitka podataka zbog prekida internetske veze.
- **Ograničenja funkcionalnosti**: Iako je aplikacija vrlo korisna za HTML, CSS i JavaScript, web verzija nema podršku za naprednije funkcionalnosti.
- **Ograničenja preglednika**: EduCoder je podložan ograničenjima web preglednika, što može utjecati na performanse i funkcionalnost aplikacije.
- **Visoki troškovi čitanja podataka**: Zbog velikog broja studenata koji koriste aplikaciju, dolazi do visokog broja čitanja podataka iz baze, što može premašiti besplatne limite Firebasea.
- **Zavisnost o internet vezi**: Rad aplikacije je u potpunosti ovisan o stabilnoj internetskoj vezi, što može predstavljati problem u slučaju prekida ili nestabilnosti veze.

5.2 Analiza rezultata i korisničkog iskustva

Ova sekcija posvećena je analizi rezultata prikupljenih putem ankete među korisnicima EduCoder aplikacije. Cilj je dobiti uvid u korisničko iskustvo, identificirati ključne prednosti i nedostatke aplikacije te predložiti poboljšanja. Rezultati će biti prezentirani kroz različite aspekte korištenja aplikacije, kao što su jednostavnost korištenja, korisničko sučelje, tehnička podrška i utjecaj na učenje.

5.2.1 Anketa

1. **Iskustvo s EduCoderom**

   (a) Kako biste opisali svoje cjelokupno iskustvo s EduCoderom?

   - Vrlo loše
   - Loše
   - Neutralno
   - Dobro
   - Vrlo dobro

   (b) Koliko vam je korisna interna ”sandbox” okolina za vježbanje kodiranja?

   - Nisam ju koristio
   - Nije korisna
   - Neutralno
   - Korisna je
   - Vrlo je korisna

   (c) Koliko često koristite EduCoder za pripremu ili polaganje ispita?

   - Nikada
   - Rijetko, manje od jedanput tjedno
   - Ponekad, tu i tamo
   - Često, više puta tjedno
   - Gotovo svaki dan

   (d) Kako biste ocijenili jednostavnost korištenja EduCodera? (Koliko vam je teško/jednostavno započeti programirati, analizirati greške u kôdu, pratiti stanje programa)

   - Vrlo kompliciran
   - Kompliciran
   - Neutralan
   - Jednostavan
   - Vrlo jednostavan

   (e) Kako ocjenjujete korisničko sučelje EduCoder platforme? (raspored elemenata, funkcionalnosti, mogućnosti prilagođavanja)

   - Vrlo loše
   - Loše
   - Neutralno

- Dobro
- Vrlo dobro

(f) Koliko su vam jasne upute dane na početku?

- Nejasne
- Djelomično jasne
- Super jasne

(g) Kako ocjenjujete brzinu izvođenja i odaziv EduCoder aplikacije?

- Vrlo loše
- Loše
- Neutralno
- Dobro
- Vrlo dobro

(h) Smatrate li da EduCoder adekvatno podržava funkcionalnosti HTML, CSS i JavaScript jezika?

- Ne podržava
- Djelomično podržava
- Ne znam
- Uglavnom podržava
- Potpuno podržava

(i) Koristite li opciju za pregledavanje skripti i materijala tijekom pisanja ispita i/ili učenja?

- Nikada
- Rijetko
- Ponekad
- Često
- Uvijek

2. **Tehnička podrška i poboljšanja**

(a) Koliko ste zadovoljni tehničkom podrškom EduCodera? (Odgovor developera na probleme/bugove/glitcheve)

- Vrlo nezadovoljan/na
- Nezadovoljan/na
- Neutralno
- Zadovoljan/na

- Vrlo zadovoljan/na

(b) Jeste li ikada naišli na tehničke probleme tijekom korištenja EduCodera?
- Da
- Ne

(c) Ako jeste, molimo vas da navedete koji su to bili problemi:

(d) Smatrate li da EduCoder doprinosi poboljšanju vaših programerskih vještina?
- Uopće ne doprinosi
- Ne doprinosi
- Neutralno
- Doprinosi
- Doprinosi mnogo

(e) Koliko smatrate da je EduCoder prilagođen potrebama studenata FIPU-a? (uvjeti rada, predznanje, podržava funkcionalnosti u sklopu gradiva kolegija)
- Uopće nije prilagođen
- Nije prilagođen
- Neutralno
- Prilagođen
- Vrlo prilagođen

(f) Koje biste nove funkcionalnosti željeli vidjeti u budućim verzijama EduCodera?

(g) Imate li prijedloge ili komentare kako poboljšati EduCoder?

3. **Osnovni podaci**

(a) Dob

(b) Godina studija
- Prva godina prijediplomski
- Druga godina prijediplomski
- Treća godina prijediplomski
- Prva godina diplomski
- Druga godina diplomski

(c) Vrsta studija
- Redovni
- Online
- Ostalo

(d) Kolegij

- Programiranje u skriptnim jezicima (PJS)
- Programsko inženjerstvo (PI)

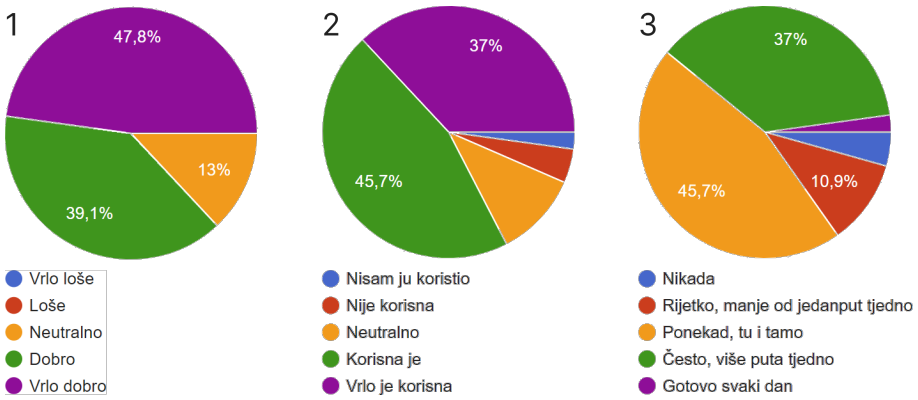
5.2.2 Analiza rezultata ankete

Rezultati ankete prikupljeni od korisnika EduCoder aplikacije pružaju uvid u različite aspekte korisničkog iskustva. Anketu je popunilo četrdeset i šest (46) studenata.

Rezultati će biti analizirani kako slijedi:

- **Iskustvo s EduCoderom**

1. *Cjelokupno iskustvo s EduCoderom*: Većina korisnika ocijenila je svoje iskustvo kao pozitivno, s velikim brojem odgovora koji su označili ocjene ”Dobro” (18) i ”Vrlo dobro” (22). Manji broj korisnika imao je neutralna iskustva (6), što ukazuje na općenito zadovoljstvo aplikacijom.
2. *Korisnost interne ”sandbox” okoline*: Veliki broj korisnika smatrao je internu ”sandbox” okolinu vrlo korisnom (21) i korisnom (17) za vježbanje kodiranja. Manji broj korisnika imao je neutralna iskustva (5) ili je smatralo da nije korisna (2) dok ju jedan korisnik nije koristio. Ovo potvrđuje važnost ove funkcionalnosti u podršci učenju kroz praktične zadatke.
3. *Učestalost korištenja EduCodera*: Odgovori pokazuju da većina korisnika koristi EduCoder više nego povremeno, s posebno velikim brojem korisnika koji koriste aplikaciju ponekad (21) i više puta tjedno (17). Manji broj korisnika koristi Educoder rijetko (5), dvoje nikada te jedna osoba gotovo svaki dan.

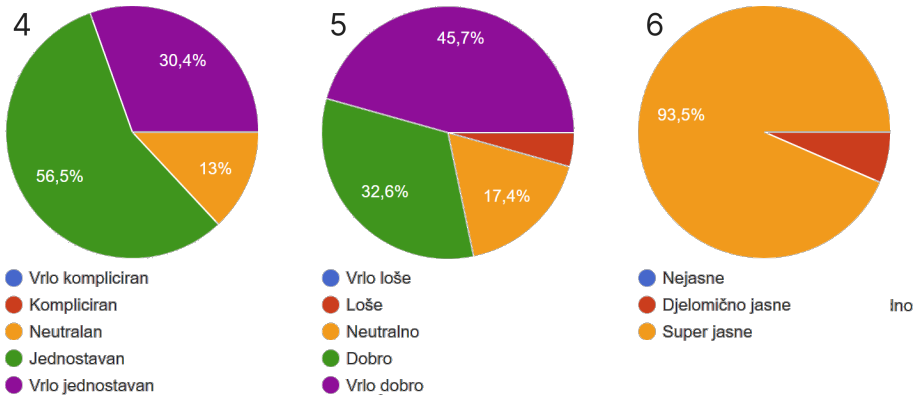


Slika 60: *Iskustvo s EduCoderom - [1,2,3] (izvor: Autor)*

4. *Jednostavnost korištenja*: EduCoder je ocijenjen kao jednostavan (26) ili vrlo jednostavan (14) za korištenje, što sugerira da je aplikacija intuitivna i lako

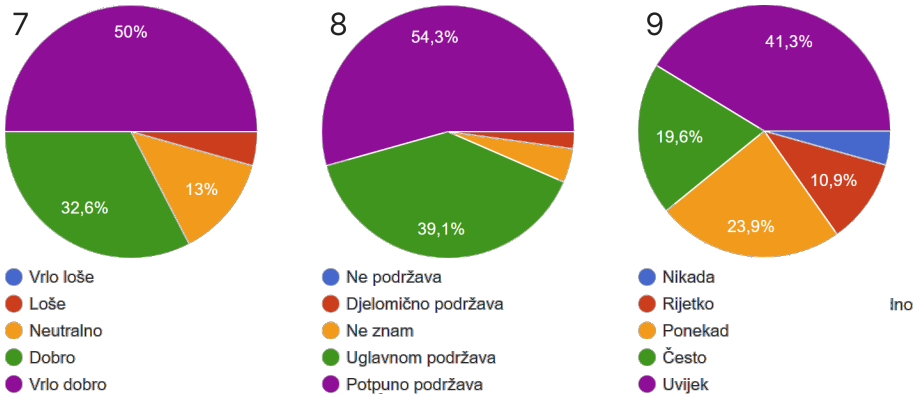
razumljiva, čak i za korisnike s manje iskustva u programiranju. Manji broj korisnika imao je neutralna iskustva (6).

5. *Korisničko sučelje*: Korisnici su općenito zadovoljni korisničkim sučeljem, veliki broj korisnika smatra sučelje vrlo dobrim (21) i dobrim (15), naglašavajući raspored elemenata i mogućnosti prilagođavanja kao pozitivne aspekte. Manji broj korisnika imao je neutralna iskustva (8) te dvoje loša iskustva, što ukazuje da se sučelje i dalje može poboljšati.
6. *Jasnoća uputa*: Većina korisnika smatra upute jasnim (3) ili super jasnim (42), što ukazuje na kvalitetne upute i vodiče unutar aplikacije unatoč nedostatku dokumentacije.



Slika 61: *Iskustvo s EduCoderom - [4,5,6] (izvor: Autor)*

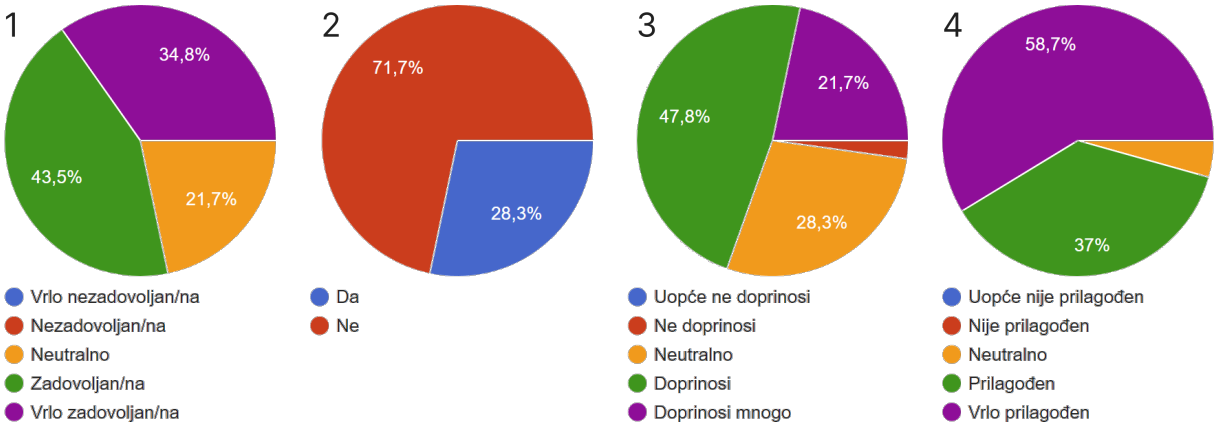
7. *Brzina izvođenja i odaziv*: Korisnici su pozitivno ocijenili brzinu izvođenja i odaziv aplikacije, gdje veliki broj korisnika smatra da je vrlo dobra (23) i dobra (15), što je ključno za efektivno korištenje tijekom ispita i vježbi. Manji broj korisnika imao je neutralna iskustva (6) te dvoje loša iskustva.
8. *Podrška za HTML, CSS i JavaScript*: Većina korisnika smatra da EduCoder uglavnom (18) ili potpuno podržava (25) funkcionalnosti HTML, CSS i JavaScript jezika, što je ključno za uspješno učenje i vježbanje. Manji broj korisnika nije mogao procijeniti (2) te jedan smatra da djelomično podržava.
9. *Korištenje opcije za pregledavanje skripti*: Mnogi korisnici uvijek (19), ponekad (11) ili često (9) koriste opciju za pregledavanje skripti i materijala tijekom pisanja ispita i učenja, što ukazuje na korisnost ove funkcionalnosti. Manji broj korisnika rijetko koristi (5) te dvoje korisnika nikada nije koristilo.



Slika 62: *Iskustvo s EduCoderom - [7,8,9] (izvor: Autor)*

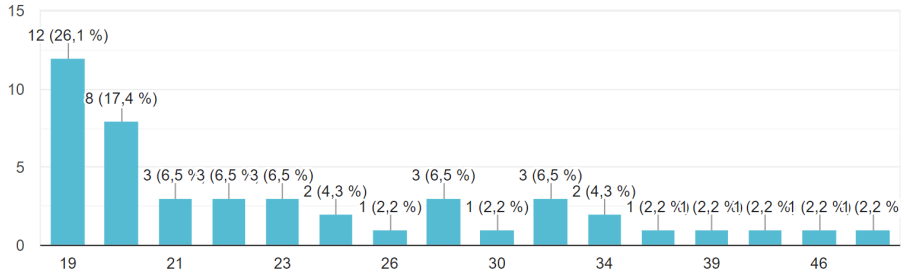
- **Tehnička podrška i poboljšanja**

  1. *Zadovoljstvo tehničkom podrškom*: Većina korisnika je zadovoljna (20) i vrlo zadovoljna (16) tehničkom podrškom, navodeći da developeri promptno reagiraju na probleme i bugove. Manji broj korisnika je neutralan (10).
  2. *Nailazak na tehničke probleme*: Neki korisnici su prijavili tehničke probleme tijekom korištenja aplikacije (13), što sugerira potrebu za daljnjim poboljšanjima i optimizacijom. Problemi uključuju povremeno zamrzavanje aplikacije, greške u konzoli, spor odaziv tipkanja, rušenje zbog *Out of Memory* greške, te ograničenja u kopiranju kôda. Ostali (33) korisnici nisu naišli na tehničke probleme.
  3. *Poboljšanje programerskih vještina*: Veliki broj korisnika smatra da EduCoder doprinosi (22) ili mnogo doprinosi (10) poboljšanju njihovih programerskih vještina, potvrđujući edukativnu vrijednost aplikacije. Popriličan broj korisnika je neutralan (13) i samo jedan korisnika smatra da ne doprinosi.
  4. *Prilagođenost potrebama studenata FIPU-a*: Većina korisnika smatra da je EduCoder prilagođen (17) ili vrlo prilagođena (27) potrebama studenata, podržavajući funkcionalnosti koje su u skladu s gradivom kolegija. Dvoje korisnika je neutralno.



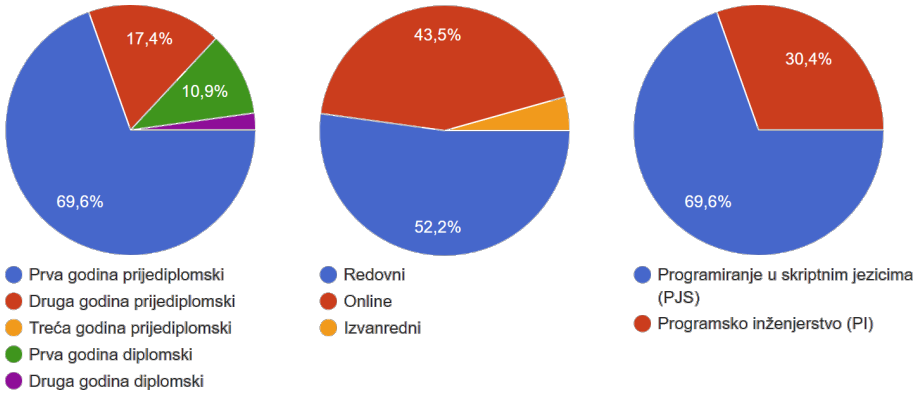
Slika 63: *Tehnička podrška i poboljšanja - [1,2,3,4] (izvor: Autor)*

- **Osnovni podaci**
  - *Dob i godina studija*: Rezultati pokazuju širok raspon godina (19-46) i različite godine studija, uključujući studente prve (32), druge (8) i treće (5) godine preddiplomskog studija te prve (5) i druge (1) godine diplomskog studija. Ova raznolikost ukazuje na to da aplikaciju koriste studenti s različitim razinama iskustva i predznanja.



Slika 64: *Osnovni podaci - raspon godina (izvor: Autor)*

  - *Vrsta studija i kolegij*: Većina korisnika su redovni (24) studenti, dok manji dio dolazi iz online (20) studija te dva korisnika koji su izvanredni studenti. Veći broj korisnika dolazi iz kolegija ”Programiranje u skriptnim jezicima” (PJS) (32) i manji iz ”Programsko inženjerstvo” (PI) (14).



Slika 65: *Osnovni podaci - godina studija, vrsta studija i kolegij (izvor: Autor)*

- **Nove funkcionalnosti i poboljšanja**: Korisnici su predložili niz novih funkcionalnosti i poboljšanja koja bi željeli vidjeti u budućim verzijama EduCodera. Među najčešće spominjanim prijedlozima su:
  - Mogućnost pretraživanja pojmova u skriptama (Ctrl+F).
  - Postavljanje vremenskog ograničenja u sandbox modu te podrška za vježbanje na više zadataka.
  - Podrška za više programskih jezika.
  - Bolje predviđanje i preporuke metoda i atributa u JavaScript editoru.
  - Pretraga i označavanje riječi unutar skripti.
  - Različite teme za estetski privlačniji izgled.
  - Pohranjivanje rezultata riješenih zadataka unutar aplikacije.
  - Mogućnost zumiranja na manjim ekranima.
  - Uklanjanje ograničenja kopiranja unutar aplikacije.
  - Implementacija AI pomoćnika.
  - Poboljšanja u code editoru, uključujući pametniji code suggestion.
  - Alat za debuggiranje kôda.
  - Omogućavanje vraćanja na isto mjesto u skripti pri ponovnom ulasku.
  - Prikaz linije kôda u kojoj se dogodila greška.

Ova analiza rezultata i korisničkog iskustva jasno pokazuje da EduCoder pruža značajne koristi studentima u učenju programiranja, unatoč nekim tehničkim izazovima i područjima za poboljšanje. Prijedlozi korisnika za nove funkcionalnosti i poboljšanja pružaju smjernice za daljnji razvoj aplikacije, s ciljem da se unaprijedi njezina funkcionalnost i prilagodba potrebama studenata. Sveukupno, povratne informacije korisnika su pozitivne i ukazuju na visoku vrijednost koju EduCoder donosi u obrazovni proces.

6 Zaključak

Kroz detaljnu analizu i primjenu EduCoder aplikacije, ovaj rad je pokazao kako moderna tehnologija može unaprijediti proces učenja i evaluacije programiranja. EduCoder je omogućio studentima Fakulteta informatike u Puli da kroz interaktivne i strukturirane vježbe poboljšaju svoje vještine programiranja, pružajući im pritom potrebnu podršku i povratne informacije koje su ključne za njihov napredak.

Implementacija EduCoder aplikacije donijela je nekoliko značajnih prednosti. Prvo, omogućila je studentima da pristupe vježbama i ispitima u bilo koje vrijeme i s bilo koje lokacije, čime je povećana fleksibilnost i dostupnost obrazovnog sadržaja. Interaktivno okruženje aplikacije potiče aktivno učenje, omogućujući studentima da odmah vide rezultate svog rada i isprave pogreške u stvarnom vremenu. Ovo je posebno korisno kod učenja programiranja, gdje su praktične vještine i iskustvo ključni za razumijevanje koncepta.

Automatska evaluacija kôda koju pruža EduCoder aplikacija značajno je smanjila opterećenje nastavnika, omogućujući im da se usmjere na pružanje dodatne podrške studentima i poboljšanje nastavnih materijala. Nastavnici su također imali koristi od detaljnih izvještaja o napretku studenata, što im je omogućilo bolje praćenje individualnih potreba i prilagodbu nastave prema tim potrebama.

Unatoč brojnim prednostima, tijekom primjene EduCoder aplikacije identificirani su i određeni nedostaci. Na primjer, tehnički problemi poput povremenog zamrzavanja aplikacije i sporog odaziva tipkanja ukazuju na potrebu za daljnjom optimizacijom softverske arhitekture. Također, korisnici su izrazili potrebu za dodatnim funkcionalnostima poput podrške za više programskih jezika, boljeg alata za debuggiranje i mogućnosti pretraživanja unutar skripti. Ovi prijedlozi korisnika pružaju vrijedne smjernice za budući razvoj i poboljšanje aplikacije.

Sigurnosne mjere implementirane u EduCoder aplikaciji, kao što su sprječavanje varanja i osiguravanje integriteta ispita, pokazale su se učinkovitima. Međutim, i dalje postoji prostor za poboljšanje u pogledu dodatnih zaštitnih mjera i prilagodbi koje bi dodatno osigurale pravovremeno otkrivanje i prevenciju mogućih pokušaja zlouporabe.

U konačnici, EduCoder je pokazao veliki potencijal kao alat za učenje i evaluaciju programiranja. Njegova upotreba na Fakultetu informatike u Puli demonstrirala je kako integracija tehnologije može unaprijediti obrazovni proces, čineći ga pristupačnijim, fleksibilnijim i učinkovitijim. S obzirom na pozitivne rezultate i korisničke povratne informacije, daljnji razvoj EduCoder aplikacije usmjeren na optimizaciju performansi i implementaciju novih funkcionalnosti može dodatno poboljšati iskustvo učenja i evaluacije za studente. Ova studija također može poslužiti kao model za druge obrazovne institucije koje razmatraju uvođenje sličnih alata u svoje nastavne programe.

6.1 Daljnja istraživanja

Buduća istraživanja i razvoj EduCoder aplikacije trebala bi se fokusirati na nekoliko ključnih područja koja mogu značajno unaprijediti njezinu funkcionalnost i korisničko iskustvo.

Prvo, integracija naprednih tehnologija poput umjetne inteligencije (AI) mogla bi revolucionirati način na koji se evaluiraju studentski zadaci. Implementacija AI-a za automatsko ocjenjivanje kôda ne samo da bi ubrzala proces evaluacije, već bi također omogućila dublju analizu studentskih rješenja. AI algoritmi mogu biti obučeni za prepoznavanje ne samo sintaktičkih, već i semantičkih grešaka, te pružanje detaljnih povratnih informacija studentima. Na primjer, AI bi mogao prepoznati neučinkovite algoritme i ponuditi optimizirane alternative, čime bi se povećala kvaliteta učenja.

Drugo, daljnja istraživanja trebala bi se usmjeriti na proširenje podrške za dodatne programske jezike. Trenutno, web verzija EduCodera podržava HTML, CSS i JavaScript, dok desktop verzija donekle podržava Python i C++. Uključivanje dodatnih jezika kao što su Java, C i R, te drugih popularnih jezika, značajno bi proširilo upotrebljivost aplikacije. Ovo bi omogućilo studentima iz različitih obrazovnih programa da koriste EduCoder za vježbanje i polaganje ispita iz raznih programskih jezika, što bi povećalo vrijednost aplikacije u širem akademskom kontekstu.

Treće, optimizacija performansi aplikacije je ključno područje koje zahtijeva pažnju. Identificirane su tehničke poteškoće poput povremenih zamrzavanja aplikacije i problema s performansama prilikom izvršavanja složenih zadataka. Buduća istraživanja trebala bi istražiti mogućnosti za poboljšanje učinkovitosti aplikacije, uključujući optimizaciju baze podataka i unaprjeđenje algoritama za izvršavanje kôda. Posebna pažnja trebala bi se posvetiti minimiziranju broja čitanja i pisanja u bazu podataka te optimizaciji mrežnih zahtjeva, što bi moglo smanjiti latenciju i poboljšati ukupnu brzinu aplikacije.

Nadalje, implementacija AI sustava za snimanje i analiziranje zaslonskih slika (screenshot) mogla bi dodatno osigurati integritet ispita. Ovi sustavi bi mogli redovito uzimati zaslonske slike tijekom ispita i analizirati ih kako bi otkrili prisutnost nedozvoljenih aplikacija ili materijala.

U konačnici, daljnja istraživanja trebala bi uključivati i redovitu evaluaciju korisničkog iskustva kroz povratne informacije i ankete, kako bi se kontinuirano prilagođavale funkcionalnosti i sučelje aplikacije potrebama korisnika. Implementacija ovih preporuka mogla bi značajno unaprijediti EduCoder i učiniti ga još korisnijim alatom za obrazovne institucije diljem svijeta.

6.2 Zaključne misli

EduCoder predstavlja značajan korak naprijed u evaluaciji studentskih vještina u programiranju. Kroz ovaj rad prikazana je detaljna analiza implementacije, primjene i korisničkog iskustva s ovom aplikacijom. Njegova primjena u nastavi omogućila je studentima da kroz praktične zadatke i strukturirane ispite unaprijede svoje vještine na način koji je teško postići tradicionalnim metodama učenja.

Jedan od ključnih aspekata EduCodera je njegova interaktivna ”sandbox” okolina, koja studentima pruža sigurno i kontrolirano okruženje za vježbanje programiranja. Ova funkcionalnost omogućava studentima da eksperimentiraju s kôdom, odmah vide rezultate svojih promjena i uče kroz praktične primjere. Povratne informacije iz ankete pokazale su da studenti visoko cijene ovu značajku, što dodatno potvrđuje njen značaj u procesu učenja.

Aplikacija je također uspjela osigurati visoku razinu sigurnosti tijekom ispita, otežavajući varanje i osiguravajući integritet procesa ocjenjivanja. Korištenje različitih tehnika poput limitiranja kopiranja, detekcije izlaska, kao i provjeravanje autentifikacije, pokazalo se učinkovitim u sprječavanju varanja.

Identificirani su i određeni izazovi koje je potrebno riješiti kako bi se aplikacija dodatno unaprijedila. Tehnički problemi kao što su povremeno zamrzavanje aplikacije, spor odaziv na tipkanje i ograničenja u kopiranju kôda ukazuju na potrebu za daljnjom optimizacijom i razvojem. Uz to, proširenje podrške za dodatne programske jezike i integracija naprednih tehnologija kao što su umjetna inteligencija za automatsko ocjenjivanje, mogli bi značajno poboljšati funkcionalnost i korisničko iskustvo.

Korisničke ankete su otkrile nekoliko područja za poboljšanje, uključujući bolje predviđanje metoda i atributa u JavaScript editoru, pretragu unutar skripti i mogućnost prilagodbe sučelja. Implementacija ovih preporuka mogla bi dodatno povećati korisnost i popularnost aplikacije među studentima.

Pozitivne povratne informacije od korisnika, posebno u pogledu korisničkog sučelja, jasnoće uputa i brzine izvođenja, ukazuju na visoku vrijednost i potencijal aplikacije. EduCoder ne samo da olakšava proces učenja programiranja, već također potiče studente na dublje razumijevanje materijala kroz praktične zadatke i interaktivne vježbe.

U konačnici, EduCoder se pokazuje kao moderna tehnologija i moćan alat u obrazovanju. Kontinuiranim razvojem i prilagođavanjem aplikacije potrebama korisnika, moguće je stvoriti još učinkovitije i inovativnije okruženje za učenje programiranja, čime se povećava kvaliteta obrazovanja i priprema studenata za daljnje obrazovanje.

Literatura

[1] Danijel Radosevic, Tihomir Orehovački, and Alen Lovrencic. Verificator: Educational tool for learning programming. *Informatics in Education*, 8:261–280, 10 2009. doi: 10.15388/infedu.2009.16.

[2] Vue.js documentation, . URL https://vuejs.org/. Accessed: 2024-06-30.

[3] Vue.js reactivity, . URL https://vuejs.org/v2/guide/reactivity.html. Accessed: 2024-06-30.

[4] Vue.js components, . URL https://vuejs.org/v2/guide/components.html. Accessed: 2024-06-30.

[5] Vue.js directives, . URL https://vuejs.org/v2/guide/syntax.html. Accessed: 2024-06-30.

[6] Vue.js virtual dom, . URL https://vuejs.org/v2/guide/render-function.html. Accessed: 2024-06-30.

[7] Vue.js Team. Vue.js options api and composition api, 2024. URL https://vuejs.org/guide/extras/composition-api-faq.html. Accessed: 2024-06-30.

[8] Vue.js Team. Performance best practices, 2024. URL https://vuejs.org/guide/best-practices/performance. Accessed: 2024-06-30.

[9] Reddit User. For anyone who have tried many frameworks, what are your takes on vue?, 2024. URL https://www.reddit.com/r/vuejs/comments/17w4s8o/for_anyone_who_have_tried_many_frameworks_what/. Accessed: 2024-06-30.

[10] Vite Team. Vite documentation, 2024. URL https://vitejs.dev/guide/. Accessed: 2024-06-30.

[11] Vite Team. Vite hot module replacement, 2024. URL https://vitejs.dev/guide/features.html#hot-module-replacement. Accessed: 2024-06-30.

[12] Vite Team. Vite rollup integration, 2024. URL https://vitejs.dev/guide/build.html#rollup. Accessed: 2024-06-30.

[13] Vite Team. Vite plugin system, 2024. URL https://vitejs.dev/guide/using-plugins.html. Accessed: 2024-06-30.

[14] Node.js. Node.js documentation, 2024. URL https://nodejs.org/docs/latest/api/. Accessed: 2024-06-30.

[15] Node.js. Event-driven architecture in node.js, 2024. URL https://nodejs.org/en/learn/asynchronous-work/event-loop-timers-and-nexttick#the-nodejs-event-loop. Accessed: 2024-06-30.

[16] NPM. Npm documentation, 2024. URL https://docs.npmjs.com/. Accessed: 2024-06-30.

[17] Node.js. Using node.js in production, 2024. URL https://nodejs.org/en/docs/guides/getting-started-guide/. Accessed: 2024-06-30.

[18] Node.js. Asynchronous programming in node.js, 2024. URL https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/. Accessed: 2024-06-30.

[19] Node.js. Introduction to node.js, 2024. URL https://nodejs.org/en/learn/getting-started/introduction-to-nodejs. Accessed: 2024-06-30.

[20] Node.js. The v8 javascript engine, 2024. URL https://nodejs.org/en/learn/getting-started/the-v8-javascript-engine. Accessed: 2024-06-30.

[21] Node.js. Single-threaded nature of node.js, 2024. URL https://nodejs.org/en/docs/guides/dont-block-the-event-loop/. Accessed: 2024-06-30.

[22] Node.js. Challenges of asynchronous programming in node.js, 2024. URL https://nodejs.org/en/learn/asynchronous-work/javascript-asynchronous-programming-and-callbacks. Accessed: 2024-06-30.

[23] NPM. Issues with npm packages, 2024. URL https://docs.npmjs.com/common-errors.

[24] Electron. What is electron?, 2024. URL https://www.electronjs.org/docs/latest/. Accessed: 2024-06-30.

[25] Electron. Electron cross-platform support, 2024. URL https://www.electronjs.org/docs/tutorial/development-environment. Accessed: 2024-06-30.

[26] Electron. Electron web technologies, 2024. URL https://www.electronjs.org/docs/tutorial/quick-start. Accessed: 2024-06-30.

[27] Electron. Electron node.js integration, 2024. URL https://www.electronjs.org/docs/tutorial/using-node-with-electron. Accessed: 2024-06-30.

[28] Electron. Electron auto update, 2024. URL https://www.electronjs.org/docs/tutorial/updates. Accessed: 2024-06-30.

[29] Electron. Electron resource usage, 2024. URL https://www.electronjs.org/docs/tutorial/performance. Accessed: 2024-06-30.

[30] Electron. Electron security, 2024. URL https://www.electronjs.org/docs/tutorial/security. Accessed: 2024-06-30.

[31] Google Firebase. Firebase overview, 2024. URL https://firebase.google.com/products. Accessed: 2024-06-30.

[32] Firebase. Firebase authentication, 2024. URL https://firebase.google.com/docs/auth. Accessed: 2024-06-30.

[33] Firebase. Firebase realtime database, 2024. URL https://firebase.google.com/docs/database. Accessed: 2024-06-30.

[34] Firebase. Cloud firestore, 2024. URL https://firebase.google.com/docs/firestore. Accessed: 2024-06-30.

[35] Firebase. Firebase storage, 2024. URL https://firebase.google.com/docs/storage. Accessed: 2024-06-30.

[36] Firebase. Firebase hosting, 2024. URL https://firebase.google.com/docs/hosting. Accessed: 2024-06-30.

[37] Firebase. Cloud functions, 2024. URL https://firebase.google.com/docs/functions. Accessed: 2024-06-30.

[38] Firebase. Firebase analytics, 2024. URL https://firebase.google.com/docs/analytics. Accessed: 2024-06-30.

[39] @headlessui/tailwindcss. URL https://github.com/tailwindlabs/headlessui. Accessed: 2024-07-01.

[40] @vueform/slider, . URL https://github.com/vueform/slider. Accessed: 2024-07-01.

[41] @vuelidate/core, . URL https://vuelidate-next.netlify.app/. Accessed: 2024-07-01.

[42] @vuelidate/validators, . URL https://vuelidate-next.netlify.app/validators.html. Accessed: 2024-07-01.

[43] crypto-random-string. URL https://github.com/sindresorhus/crypto-random-string. Accessed: 2024-07-01.

[44] Firebase. URL https://firebase.google.com/. Accessed: 2024-07-01.

[45] Pinia, . URL https://pinia.vuejs.org/. Accessed: 2024-07-01.

[46] pinia-plugin-persistedstate, . URL https://github.com/prazdevs/pinia-plugin-persistedstate. Accessed: 2024-07-01.

[47] vue-json-pretty, . URL https://github.com/leezng/vue-json-pretty. Accessed: 2024-07-01.

[48] vue-router, . URL https://router.vuejs.org/. Accessed: 2024-07-01.

[49] vue-showdown, . URL https://github.com/meteorlxy/vue-showdown. Accessed: 2024-07-01.

[50] ace-builds. URL https://github.com/ajaxorg/ace-builds. Accessed: 2024-07-01.

[51] vue3-ace-editor, . URL https://github.com/CarterLi/vue3-ace-editor. Accessed: 2024-07-01.

[52] vue3-colorpicker, . URL https://github.com/wangyi7099/vue3-colorpicker. Accessed: 2024-07-01.

[53] @vuepic/vue-datepicker, . URL https://github.com/vuepic/vue-datepicker. Accessed: 2024-07-01.

[54] vuedraggable, . URL https://github.com/SortableJS/Vue.Draggable. Accessed: 2024-07-01.

[55] Mozilla Developer Network (MDN). Using the iframe element, 2023. URL https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe. Accessed: 2024-07-01.

[56] Mozilla Developer Network. The blur event, 2023. URL https://developer.mozilla.org/en-US/docs/Web/API/Element/blur_event. Accessed: 2024-07-01.

[57] Mozilla Developer Network. The keydown event, 2023. URL https://developer.mozilla.org/en-US/docs/Web/API/Element/keydown_event. Accessed: 2024-07-01.

[58] Google Developers. Browser extensions and security, 2023. URL https://developer.chrome.com/docs/extensions/mv3/security/. Accessed: 2024-07-01.

[59] Mozilla Developer Network. Http cookies and security, 2023. URL https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. Accessed: 2024-07-01.

[60] Electron-Vite Contributors. Electron-vite guide, 2023. URL https://electron-vite.org/guide/. Accessed: 2024-07-01.

[61] Alex8088. Electron-vite quick start - vue, 2023. URL https://github.com/alex8088/quick-start/tree/master/packages/create-electron/playground/vue. Accessed: 2024-07-01.

[62] Jakob Nielsen. 10 usability heuristics for user interface design, 2023. URL https://www.nngroup.com/articles/ten-usability-heuristics/. Accessed: 2024-07-01.

Popis slika

1 *Model slučaja za korisnike unutar EduCoder aplikacije (izvor: Autor)* . . 5
2 *Model slučaja za administratore unutar EduCoder aplikacije (izvor: Autor)* 6
3 *Modeli slijeda korisnika i administratora unutar EduCoder aplikacije (izvor: Autor)* 8
4 *Model slijeda korisnika - prijava i autentifikacija korisnika (izvor: Autor)* . 9
5 *Model slijeda korisnika - spremanje korisnika i dohvaćanje vidljivosti skripti (izvor: Autor)* 10
6 *Model slijeda korisnika - sandbox i dohvaćanje ispita (izvor: Autor)* . . . 11
7 *Model slijeda korisnika - započinjanje i postavljanje ispita (izvor: Autor)* . 12
8 *Model slijeda korisnika - rješavanje i spremanje ispita (izvor: Autor)* . . . 13
9 *Model slijeda korisnika - odjava (izvor: Autor)* 14
10 *Model slijeda administratora - prijava i autentifikacija administratora (izvor: Autor)* 15
11 *Model slijeda administratora - učitavanje novog ispita (izvor: Autor)* . . . 16
12 *Model slijeda administratora - učitavanje i uređivanje vidljivosti skripti (izvor: Autor)* 17
13 *Model slijeda administratora - učitavanje i uređivanje korisnika (izvor: Autor)* 18
14 *Model slijeda administratora - učitavanje i pregled svih ispita (izvor: Autor)* 19
15 *Model slijeda administratora - uređivanje ispita (izvor: Autor)* 20
16 *Model slijeda administratora - pregled predanih rješenja (izvor: Autor)* . 21
17 *Model slijeda administratora - skupno ocjenjivanje ispita (izvor: Autor)* . 22
18 *Model slijeda administratora - odjava (izvor: Autor)* 23
19 *Klasni model (izvor: Autor)* 24
20 *Klasni model - nabrajanja (izvor: Autor)* 25
21 *Klasni model - bilješke (izvor: Autor)* 25
22 *Klasni model - user collection (izvor: Autor)* 26
23 *Klasni model - data collection (izvor: Autor)* 28
24 *Klasni model - exam collection (izvor: Autor)* 31
25 *Klasni model - solution storage (izvor: Autor)* 33
26 *Usporedba Options API i Composition API (izvor: Vue.js Team)* 37
27 *Bundle based dev server and native ESM based dev server (izvor: Vite.js Team)* 39
28 *Node.js arhitektura (izvor: Simform)* 40
29 *Struktura glavnog direktorija (izvor: Autor)* 44
30 *Primjer implementacije JavaScript blur i keydown event listenera (izvor: Autor)* 50

31 *Proces prijave putem Google računa u Electron aplikaciji (izvor: Autor)* . 52
32 *Svijetla i tamna tema aplikacije (izvor: Autor)* 55
33 *Educoder upute (izvor: Autor)* 56
34 *Primjeri Nielsenovih heuristika u EduCoderu (izvor: Autor)* 57
35 *Korisničko sučelje korisnika - stranica* `exam` *(izvor: Autor)* 58
36 *Korisničko sučelje - zaglavlje (izvor: Autor)* 58
37 *Korisničko sučelje - alatna traka (izvor: Autor)* 59
38 *Korisničko sučelje - dodatne opcije (izvor: Autor)* 59
39 *Korisničko sučelje - prozori HTML/CSS i JavaScript (izvor: Autor)* 59
40 *Korisničko sučelje - prozor konzola (izvor: Autor)* 60
41 *Korisničko sučelje - prozor skripta (izvor: Autor)* 60
42 *Korisničko sučelje - prozor pregled (izvor: Autor)* 61
43 *Korisničko sučelje - skočni prozor za učitavanje i započinjanje ispita (izvor: Autor)* 61
44 *Korisničko sučelje - ispitna traka (izvor: Autor)* 62
45 *Korisničko sučelje - održavanje u tijeku (izvor: Autor)* 62
46 *Administrativno sučelje - početna stranica (izvor: Autor)* 63
47 *Administrativno sučelje - ispit u tijeku (izvor: Autor)* 63
48 *Administrativno sučelje - skočni prozor za otklanjanje pogrešaka (izvor: Autor)* 64
49 *Administrativno sučelje - stranica za izradu novog ispita (izvor: Autor)* . 65
50 *Administrativno sučelje - admin panels stranica (izvor: Autor)* 66
51 *Administrativno sučelje - vidljivost skripti (izvor: Autor)* 66
52 *Administrativno sučelje - korisnici (izvor: Autor)* 67
53 *Administrativno sučelje - svi ispiti (izvor: Autor)* 68
54 *Administrativno sučelje - uređivanje ispita (izvor: Autor)* 69
55 *Administrativno sučelje - predana rješenja (izvor: Autor)* 70
56 *Administrativno sučelje - skupno ispravljanje ispita (izvor: Autor)* 71
57 *Formatiranje kôda (izvor: Autor)* 72
58 *Obavještavanje korisnika na nedostatak vitičaste zagrade (izvor: Autor)* 72
59 *Ubacivanje dodatnog kôda u korisnikov kôd (izvor: Autor)* 72
60 *Iskustvo s EduCoderom - [1,2,3] (izvor: Autor)* 80
61 *Iskustvo s EduCoderom - [4,5,6] (izvor: Autor)* 81
62 *Iskustvo s EduCoderom - [7,8,9] (izvor: Autor)* 82
63 *Tehnička podrška i poboljšanja - [1,2,3,4] (izvor: Autor)* 83
64 *Osnovni podaci - raspon godina (izvor: Autor)* 83
65 *Osnovni podaci - godina studija, vrsta studija i kolegij (izvor: Autor)* . . . 84

7 Sažetak

Ovaj rad istražuje implementaciju i primjenu EduCoder aplikacije, koja je razvijena za evaluaciju studentskih vještina u programiranju. Analizirani su različiti aspekti aplikacije, uključujući njenu arhitekturu, funkcionalnosti i korisničko iskustvo. Kroz anketu provedenu među studentima, identificirane su prednosti i nedostaci aplikacije te su predložena moguća poboljšanja. Rezultati pokazuju da EduCoder značajno doprinosi poboljšanju programerskih vještina studenata, unatoč tehničkim poteškoćama koje su povremeno prisutne.

Ključne riječi: EduCoder, web aplikacije, desktop aplikacija, HTML, CSS, JavaScript, Firebase, Node.js, Vue.js, Electron, evaluacija programiranja, korisničko iskustvo, obrazovna tehnologija

8 Abstract

This paper explores the implementation and application of the EduCoder application, developed to evaluate students' programming skills. Various aspects of the application, including its architecture, functionalities, and user experience, were analyzed. A survey conducted among students identified the application's strengths and weaknesses, and potential improvements were suggested. The results show that EduCoder significantly contributes to the improvement of students' programming skills, despite occasional technical difficulties.

Keywords: EduCoder, web app, desktop app, HTML, CSS, JavaScript, Firebase, Node.js, Vue.js, Electron, programming evaluation, user experience, educational technology

9 Prilog

Repozitoriji:

- Web: https://github.com/azuzic/educoder-web-public
- Desktop: https://github.com/azuzic/educoder-electron-public