Sustav za praćenje studentske aktivnosti prilikom rješavanja online testova

Grubeša, Luka

Master's thesis / Diplomski rad

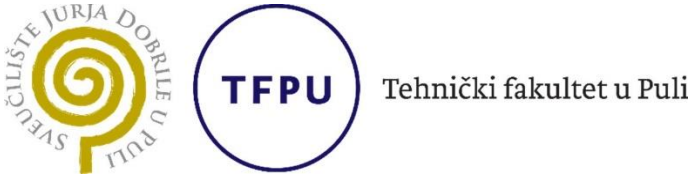
2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:726513

Sveučilište Jurja Dobrile u Puli

Tehnički fakultet u Puli



Luka Grubeša

Sustav za praćenje studentske aktivnosti prilikom rješavanja online testova

Diplomski rad

Pula, rujan, 2024.

Sveučilište Jurja Dobrile u Puli

Tehnički fakultet u Puli

Luka Grubeša

Sustav za praćenje studentske aktivnosti prilikom rješavanja online testova

Diplomski rad

JMBAG: 0303090343, redoviti student

Studijski smjer: Računarstvo

Predmet: Raspodijeljeni sustavi

Znanstveno područje: Tehničke znanosti

Znanstveno polje: Računarstvo

Znanstvena grana: Obrada informacija

Mentor: doc.dr.sc. Nikola Tanković

Pula, rujan, 2024.

Sadržaj

Sažetak

Ovaj diplomski rad prikazuje razvoj moderne web aplikacije za automatizaciju izrade, ocjenjivanja i praćenja ispita koristeći najnovije tehnologije kao što su Vue.js za frontend, FastAPI za backend, te Firebase za pohranu podataka i autentifikaciju. Aplikacija uključuje integraciju umjetne inteligencije za analizu sumnjivih aktivnosti tijekom ispita, čime se osigurava integritet ispita. Kroz detaljno istraživanje, implementaciju i testiranje, aplikacija je pokazala svoju sposobnost skaliranja i pružanja sigurnog i učinkovitog alata za obrazovne ustanove. Iako su postojali izazovi u performansama i sigurnosti, aplikacija je uspješno ostvarila postavljene ciljeve i dobila pozitivne povratne informacije od korisnika.

Ključne riječi: Full-Stack, Web App, Umjetna inteligencija, Vue.js, Javascript, HTML, CSS, FastAPI, Firebase

Summary

This thesis presents the development of a modern web application designed to automate exam creation, grading, and monitoring using cutting-edge technologies such as Vue.js for the frontend, FastAPI for the backend, and Firebase for data storage and authentication. The application integrates artificial intelligence to analyze suspicious activities during exams, ensuring the integrity of the examination process. Through comprehensive research, implementation, and testing, the application demonstrated its scalability and provided a secure and efficient tool for educational institutions. Despite challenges related to performance and security, the application successfully met its goals and received positive feedback from users.

Keywords: Full-Stack, Web App, Artificial intelligence, Vue.js, Javascript, HTML, CSS, FastAPI, Firebase

1. Uvod

U ovom diplomskom radu istražuje se cjelokupni proces izrade, testiranja i implementacije moderne web aplikacije namijenjene automatizaciji izrade ispita, njihova ocjenjivanja te kontinuiranog praćenja studenata tijekom ili nakon polaganja ispita.

Razvoj aplikacije obuhvaća korištenje najnovijih tehnologija i alata u razvoju web aplikacija, uključujući frontend izgrađen na Vue.js frameworku, backend na FastAPI frameworku te integraciju s Firebaseom za pohranu podataka i autentifikaciju korisnika. Kroz projekt, aplikacija je evoluirala od jednostavnog prototipa do robustnog rješenja koje je u mogućnosti skalirati se s obzirom na broj korisnika i složenost ispita.

1.1 Hipoteza rada

Hipoteza rada temelji se na pretpostavci da integracijom umjetne inteligencije u proces izrade i polaganja ispita možemo značajno automatizirati sve ključne procese koji se odnose na izradu, ocjenjivanje i praćenje ispita, smanjujući pritom mogućnost ljudske pogreške i poboljšavajući efikasnost cijelog sustava.

Hipoteza pretpostavlja da će integracija automatizacije stvaranja, polaganja i ocjenjivanja ispita , omogućiti pouzdanije i konzistentnije rezultate, što će rezultirati objektivnijim ocjenjivanjem i smanjenjem pristranosti. Dodatno, očekuje se da će sustav značajno smanjiti administrativno opterećenje nastavnika, omogućujući im više vremena za kvalitetno mentoriranje i interakciju sa studentima.

1.2 Predmet istraživanja

Predmet istraživanja ovog rada je izgradnja i evaluacija full-stack aplikacije pod nazivom "Bool", koja uključuje sve ključne komponente potrebne za upravljanje ispitnim procesima u digitalnom okruženju. Aplikacija obuhvaća izradu podatkovne baze, korisničkog sučelja, sučelja za polaganje ispita te administrativnih alata za pregledavanje ispita, korisnika i rezultata.

Aplikacija je koncipirana tako da podržava različite faze razvoja i testiranja, uključujući "Developer mode" za razvojne svrhe, "Quality Assurance" za testiranje funkcionalnosti te "Production" za stvarnu upotrebu od strane studenata i nastavnika.

1.3 Istraživački problem

Glavni problem istraživanja odnosi se na izazove integracije umjetne inteligencije u sustave za izradu i polaganje ispita. Iako AI modeli mogu značajno unaprijediti proces automatizacije, oni nikada nisu potpuno točni, što može dovesti do nepravilnosti u ocjenjivanju ili detekciji sumnjivog ponašanja. Pogreške u prepoznavanju lica ili analiziranju snimki mogu rezultirati nepravednim ocjenama ili propuštenim slučajevima varanja.

Također, problem predstavlja i skalabilnost sustava, s obzirom na to da dodavanje novih funkcionalnosti može povećati opterećenje na servere i produžiti vrijeme izvršavanja radnji. Osim tehničkih izazova, postavlja se i pitanje etičnosti i privatnosti, osobito kada je riječ o prikupljanju i obradi osjetljivih podataka studenata.

1.4 Ciljevi rada

Glavni ciljevi rada uključuju razvoj i implementaciju moderne web aplikacije koja koristi najnovije tehnologije za automatizaciju procesa izrade, ocjenjivanja i praćenja ispita. U specifične ciljeve uključujemo izradu stabilnog i responzivnog korisničkog sučelja koji omogćuje jednostavno i intuitivno polaganje ispita, integraciju umetne inteligencije za analizu sumljivog ponašanja, radi očuvanja integriteta ispita, osiguravanje skalabilnosti i fleksibilnosti aplikacije kako bi olakšali implementaciju novih funkcionalnosti, te provođenje i analizu rezultata kako bi se procjenila učinkovitost aplikacije.

1.5 Struktura rada

Rad je strukturiran u nekoliko ključnih poglavlja, od kojih svako detaljno obrađuje specifične aspekte razvoja i implementacije aplikacije. U Teorijskoj pozadini pruža se pregled relevantnih tehnologija, trendova i praksi u razvoju web aplikacija. Nadalje obratit će se pažnja na opisu korištenih alata, razvoju prototipa minimalne funkcionalnosti, te samo

korištenje aplikacije, uz opis odabranih ključnih funkcionalnosti, algoritama i izbora u dizajnu.

Cilj strukture ovog rada nije objašnjenje svake komponente u kodu, nego njihovih funkcionalnosti, kako bismo naznačili samo najvažnije značajke projekta.

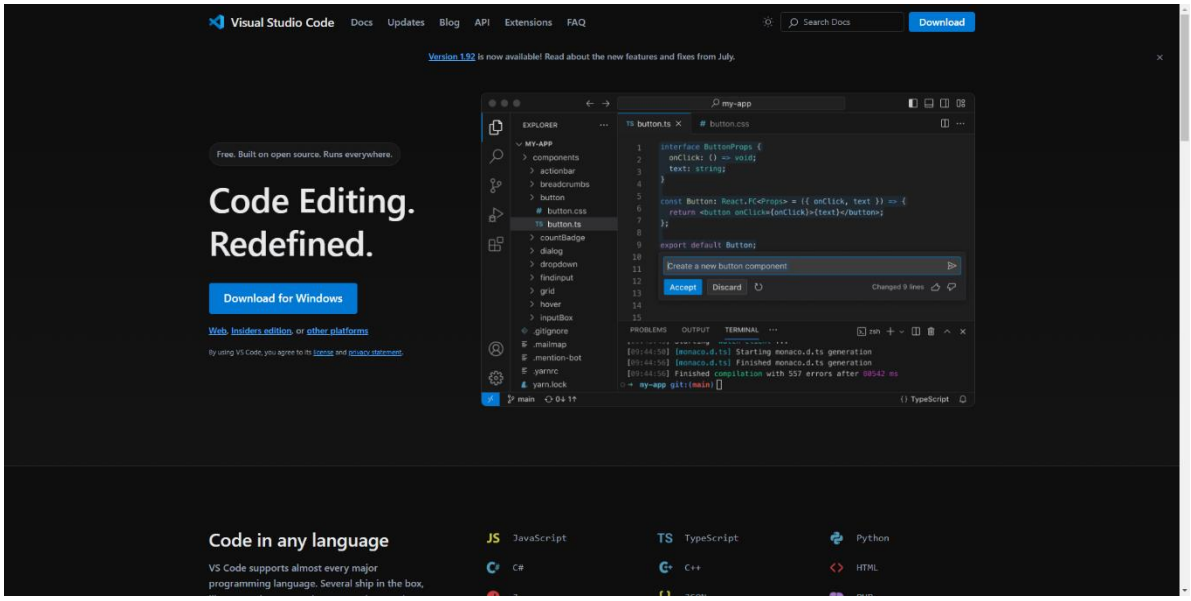
2. Korišteni alati i teorijska pozadina

2.1 Korišteni alati

2.1.1 Okruženje Visual Studio Code

Visual Studio Code (VS Code) je besplatan, „open-source“ program uređivača koda razvijen od strane Microsofta. Pokrenut je 2015. godine i brzo je postao jedan od najpopularnijih uređivača među programerima zbog svoje fleksibilnosti, performansi i širokog spektra ekstenzija. VS Code podržava mnoge jezike i tehnologije, te nudi značajke poput automatskog dovršavanja koda, debuggiranja i integracije s Gitom, što ga čini izvrsnim alatom za razvoj aplikacija.

Razlog korištenja je jednostavan; trenutačno je najpopularniji IDE na Windowsu, brz je te podržava mnogo ekstenzija koje također ubrzavaju workflow, poput Prettier, za formatiranje koda, ili git, za lakše verzioniranje. Na slici 1 vidljiva je početna stranica Visual Studio Code programa.



Slika 1. *Glavna stranica Visual Studio code-a, (izvor: https://code.visualstudio.com)*

2.1.2 *Programski okvir Vue.js*

Vite je alat za brzu izgradnju i razvoj JavaScript aplikacija, stvoren 2020. godine od strane tvorca Vue.js, Evana Youa. Vite je poznat po svom izuzetno brzom vrijeme za početak rada zahvaljujući modernom pristupu koji koristi ES Module i napredne optimizacije. Vue.js je progresivni JavaScript okvir za izgradnju korisničkih sučelja, također razvijen od Evana Youa. Kombinacija Vite i Vue.js omogućava razvijateljima brzo postavljanje projekata i efikasno upravljanje komponentama, što je idealno za razvoj dinamičkih web aplikacija.

Vue.js, prikazan na slici 2, smatran je jednim od najboljih modernih „frameworkova“ za razvoj web aplikacija, pogotovo za kompleksnije aplikacije, budući da podržava puno ekstenzija na koje se developeri oslanjaju radi boljih performansa i smanjivanje nepotrebnog koda.

Slika 2. *Glavna stranica Vue.js Frameworka (izvor: https://vuejs.org/)*

2.1.3 *Radni okvir Tailwind.css*

Tailwind.css je utilitarni CSS okvir za brzu izgradnju prilagođenih dizajna bez pisanja vlastitog CSS-a. Objavljen 2017. Godine. Glavni razlog za korištenje ovog alata jest njegova jednostavnost pisanja inline css atributa u samim komponentama, što u većinu slučajeva eliminira potrebu korištenja posebnih .css ili .scss fileova radi obavljanja iste svrhe. Tailwind također pruža intuitivnu i detaljnu listu gotovih komponenta, kao i „templates“, skupina

komponenata koje zajedno tvore veću cjelinu na stranici, primjerice dizajn moderne izborničke trake.

2.1.4 Radni okvir FastAPI

FastAPI je moderan i brz web framework za izgradnju Backend API-ja, baziran je na OpenAPI i JSON Schema standardima te je objavljen 2018. godine. Osim toga što je relativno brz za framework napisan za python programski jezik, također ima integraciju sa Swagger sučeljem za testiranje API poziva, što jako ubrzava razvojnu fazu i testing same aplikacije, te ga ćini jednim od najpristupačnijim i versatilnim alatima koji ćemo koristiti.

2.1.5 Servisi Firebase RealtimeDB i Firebase Auth

Firebase je cloud-based platforma za razvoj aplikacija koju je Google stekao 2014. godine. Pruža razne alate i usluge, od kojih ćemo koristiti „Realtime Database“, radi brze pohrane i mogućnosti mjenjanja podataka u bazi pomoću njihovog sučelja.

Također koristimo Firebase Auth, budući da je potrebno verificirati studente koji polažu ispit na sigurni i jednostavan način.

2.1.6 Platforma Vercel

Vercel je platforma za hosting web aplikacija koja se ističe jednostavnošću korištenja i iznimno brzim vremenima implementacije. Osnovana 2015. godine pod imenom ZEIT, platforma je 2020. godine preimenovana u Vercel. Danas se Vercel ubraja među vodeće platforme za hosting web aplikacija, osobito poznata po svojoj mogućnosti besplatnog hostinga i širokom spektru konfiguracijskih opcija.

Jedna od ključnih značajki Vercela je njegova podrška za automatizaciju procesa implementacije (deployment). Platforma omogućuje automatsko implementiranje promjena svaki put kada se izvrši ažuriranje na određenoj grani (branch) u Git repozitoriju. Ova funkcionalnost kontinuirane isporuke (CI/CD) značajno ubrzava razvojni proces, omogućujući timovima da brzo i učinkovito testiraju, implementiraju i održavaju svoje web aplikacije.

2.1.7 *Platforma Render*

Render, pokrenut 2018. godine, predstavlja platformu koja podržava širok spektar aplikacija, uključujući web aplikacije, statičke stranice i pozadinske procese. Render se pokazao korisnim za hosting našeg backend servera zbog svoje fleksibilnosti i jednostavnosti implementacije.

Međutim, besplatna verzija platforme ima određena ograničenja, posebno u pogledu performansi i dostupnosti. Naime, besplatna opcija nudi vrlo ograničene resurse, što može dovesti do značajno slabijih performansi. Također, besplatni planovi na Renderu uključuju automatsko gašenje i ponovno pokretanje servera ovisno o aktivnosti, što može rezultirati periodima neaktivnosti. Zbog ovih ograničenja, odlučili smo se za nadogradnju na plaćeni plan koji nudi stabilnije performanse i kontinuiranu dostupnost, što je ključno za nesmetan rad aplikacije u produkcijskom okruženju.

2.1.8 *Paketi CV2 i RetinaFace*

CV2, poznatiji kao OpenCV, je otvoreni softverski paket za računalni vid koji pruža širok spektar alata za obradu slika i videozapisa. Ovaj softver omogućuje razvoj i implementaciju složenih algoritama za analizu vizualnih podataka, čineći ga jednim od najčešće korištenih alata u području računalnog vida.

RetinaFace je moderna metoda za detekciju lica koja se temelji na dubokim konvolucijskim mrežama. Ova metoda omogućuje visoko preciznu detekciju lica i anomalija u stvarnom vremenu, pružajući superiorne performanse u usporedbi s tradicionalnim pristupima. Međutim, zbog ograničenih performansi platforme Render, osobito u pogledu raspoloživog RAM-a (512 MB), odlučeno je da će se koristiti CV2, jer troši manje resursa, što je ključno za održavanje učinkovitosti aplikacije tijekom ispita, kada do 60 korisnika može svakih pet sekundi slati zahtjeve serveru. Ova optimizacija osigurava stabilnost i pouzdanost sustava u uvjetima visokog opterećenja.

2.2 Teorijska pozadina

2.2.1 Umjetna inteligencija u obrazovanju

Umjetna inteligencija (AI) je područje računalnih znanosti koje se bavi razvojem sustava sposobnih za obavljanje zadataka koji zahtijevaju ljudsku inteligenciju. AI uključuje nekoliko ključnih komponenti, kao što su strojno učenje, obrada prirodnog jezika, računalni vid, robotski vid i druge. Ove tehnologije omogućuju stvaranje sustava koji mogu automatski učiti i prilagođavati se novim situacijama, što otvara širok spektar potencijalnih primjena u različitim industrijama.

Navedene tehnologije nalaze sve širu primjenu u raznim područjima, uključujući obrazovanje, gdje mogu preuzeti brojne uloge, od personaliziranog učenja do automatizacije administrativnih zadataka. U obrazovanju, AI može pomoći u personalizaciji nastavnih planova, omogućiti automatizirano ocjenjivanje, te olakšati analizu podataka o napretku učenika. Automatizacija administrativnih zadataka smanjuje opterećenje nastavnika, omogućujući da se više fokusiraju na kvalitetu podučavanja i interakciju s učenicima, što ovaj rad nastoji poboljšati.

Osim toga, AI sustavi u obrazovanju mogu pomoći u identificiranju potreba učenika za dodatnom podrškom, predviđanju njihovog akademskog uspjeha i personalizaciji obrazovnog iskustva prema individualnim potrebama. Ove tehnologije također omogućuju razvoj adaptivnih sustava za učenje, koji se prilagođavaju tempu i stilu učenja svakog učenika, čime se povećava učinkovitost obrazovnog procesa.

S obzirom na kontinuirani napredak u AI tehnologijama, očekuje se da će njihova primjena nastaviti rasti i širiti se na nova područja. Ovo područje nastavlja transformirati industrije i način na koji ljudi interagiraju s tehnologijom, pružajući nove mogućnosti i izazove. AI ne samo da poboljšava učinkovitost i produktivnost, već i otvara put za inovacije koje mogu značajno poboljšati kvalitetu života.

2.2.2 Sigurnost i privatnost u e-učenju

S rastućom primjenom e-učenja i digitalnih alata, sigurnost i privatnost postaju sve važniji aspekti. Prikupljanje i obrada osobnih podataka studenata, uključujući njihove rezultate,

ponašanje u učenju i druge osjetljive informacije, zahtijeva stroge mjere zaštite kako bi se osigurala privatnost korisnika.

Jedan od glavnih izazova u ovom kontekstu je zaštita podataka od neovlaštenog pristupa i cyber napada. Osiguranje sigurnosti podataka uključuje primjenu šifriranja, sigurnosnih protokola i autentifikacijskih mehanizama kako bi se spriječio pristup neovlaštenih osoba. U obrazovnim sustavima, to je posebno važno kako bi se zaštitili osobni podaci studenata i spriječilo curenje informacija.

Pored tehničkih aspekata, važno je i pridržavanje pravnih i etičkih standarda u zaštiti privatnosti. To uključuje poštivanje zakona o zaštiti podataka, kao što su Opća uredba o zaštiti podataka (GDPR) u Europskoj uniji, te informiranje korisnika o načinima na koje se njihovi podaci prikupljaju, pohranjuju i koriste. Transparentnost u vezi s korištenjem podataka ključna je za izgradnju povjerenja među korisnicima e-learning platformi.

2.2.3 Tehnologije za izgradnju web aplikacija

Razvoj web aplikacija značajno je napredovao tijekom posljednjih desetljeća, uz brojne inovacije u tehnologijama i alatima koji omogućuju izgradnju složenih i interaktivnih sustava. Današnje moderne web aplikacije često koriste tzv. full-stack razvoj, koji obuhvaća sve komponente aplikacije, od frontenda do backenda, pružajući sveobuhvatan pristup razvoju aplikacija.

Frontend tehnologije uključuju temeljne jezike kao što su HTML, CSS i JavaScript, koji služe za izradu korisničkih sučelja. Moderni JavaScript okviri poput React, Angular i Vue.js omogućuju razvoj dinamičkih i responzivnih sučelja koja se lako mogu prilagoditi različitim uređajima i veličinama zaslona. Dodatno, CSS okviri poput Tailwind.css nude unaprijed definirane stilove, što značajno olakšava dizajn i ubrzava proces razvoja, čineći ga efikasnijim i konzistentnijim.

Na backend strani, tehnologije uključuju server-side jezike i okvire kao što su Node.js, Django, Ruby on Rails i FastAPI. Ovi alati omogućuju izgradnju servera, API-ja i baza podataka koji podržavaju funkcionalnosti aplikacije. FastAPI, na primjer, pruža visoke performanse i

jednostavnu integraciju s alatima za testiranje, što ga čini popularnim izborom za izgradnju suvremenih backend sustava.

Baze podataka čine ključnu komponentu svakog sustava, omogućujući pohranu i upravljanje podacima aplikacije. Ovisno o specifičnim potrebama aplikacije i strukturi podataka, koriste se relacijske baze podataka poput MySQL i PostgreSQL, kao i NoSQL baze poput MongoDB, koje nude fleksibilnost i skalabilnost u radu s nestrukturiranim podacima.

Hosting i distribucija web aplikacija danas su značajno pojednostavljeni zahvaljujući platformama kao što su Vercel, Heroku i Render. Ove platforme omogućuju jednostavno postavljanje aplikacija, automatizirani deployment i skaliranje prema potrebama korisnika. Također, podržavaju CI/CD (Continuous Integration/Continuous Deployment) prakse, koje omogućuju kontinuirano testiranje i implementaciju novih verzija aplikacije bez prekida usluge, osiguravajući stabilnost i pouzdanost u produkciji.

3. Implementacija rješenja

3.1 Razvoj prototipa (MVP)

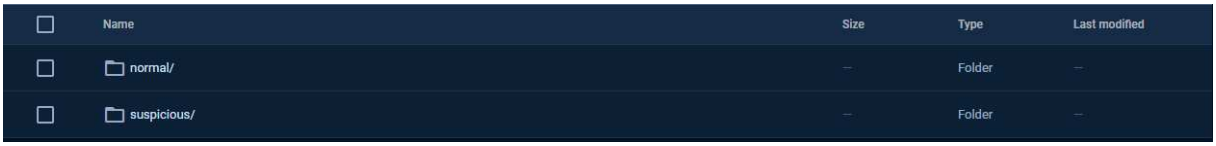
U ovom segmentu nastojimo objasniti proces razvoja minimalno prihvatljivog proizvoda (MVP), kakvim redoslijedom smo dodavali pojedine značajke, te koje promjene u korištenju aplikacije su pridonjele. Važno je napomenuti kako neki od dizajna za izgled stranica nisu krajnje inačice, nego dizajn prototipa.

3.1.1 Definiranje ciljeva i detekcija lica

Primarni cilj ovog dijela projekta bio je razviti jednostavnu aplikaciju koja će pratiti pokrete lica putem kamere i svakih x sekundi slati snimku kamere u bazu podataka, gdje je x unaprijed definirana vrijednost. Snimke se zatim kategoriziraju u jednu od dvije kategorije: sumnjivo ili normalno, vidljivo na slici 3, ovisno o tome prepoznaje li umjetna inteligencija lice na zaslonu.

Za implementaciju ove metode korišteni su alati CV2 (OpenCV) i RetinaFace, prema preporuci mentora. Kod na slici 4 prikazuje jednostavnu logiku za prepoznavanje lica pomoću biblioteke CV2. U tom procesu, slika se pretvara u "blob" (termin koji se odnosi na bilo koji komad podatka), nakon čega se vrši detekcija lica i vraća se boolean vrijednost (točno ili netočno). Ako AI prepozna lice kao sumnjivo, vraća se vrijednost "točno", dok se u suprotnom vraća

"netočno". Ovo rješenje pokazalo se adekvatnim za naše potrebe, nakon čega smo nastavili s prikazivanjem snimaka zaslona i kamere.



Slika 3: *Firebase repozitorij sa snimkama zaslona i lica, kategorizirane po sumljivosti (izvor: Autor)*

```python
def detect_face_and_flag_suspicious(image_path):
    # Load the image
    image = cv2.imread(image_path)
    if image is None:
        print("Error: Failed to load image.")
        return True  # Assume suspicious if image loading fails

    # Crop the image
    (h, w) = image.shape[:2]
    start_x, start_y = int(w//2),0
    camera_feed = image[start_y:, start_x:]

    # Prepare the image for detection
    blob = cv2.dnn.blobFromImage(camera_feed, 1.0, (500, 500), (104.0, 177.0,
    123.0))
    net.setInput(blob)

    # Perform face detection
    detections = net.forward()

    # Check if any faces are detected
    is_suspicious = True
    for i in range(detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence > 0.5:
            is_suspicious = False
            break  # Stop loop if a face is detected

    return is_suspicious
```

Slika 4: *Javascript kod za detektiranje lica u snimci (izvor: Autor)*

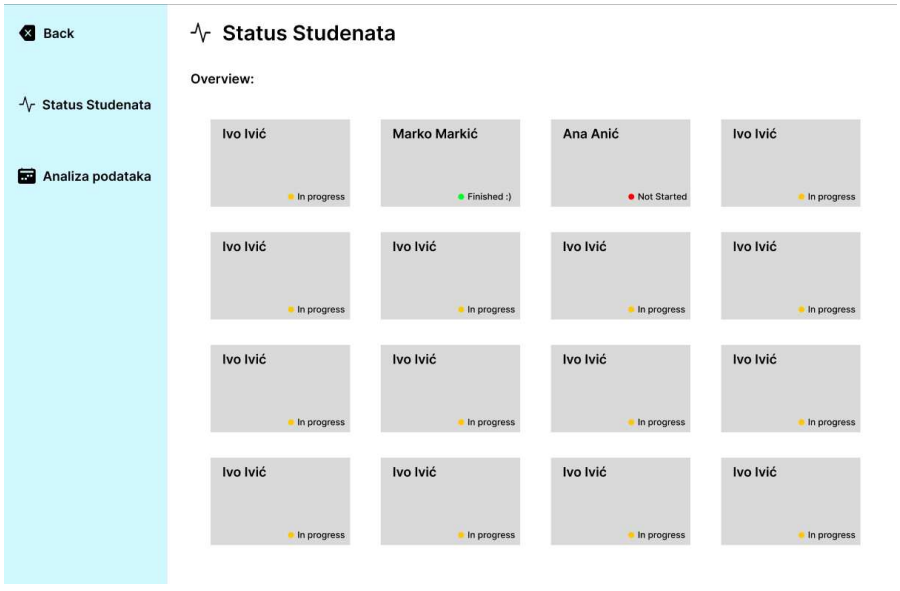
3.1.2 Sučelje za prikaz podataka

Svaki screenshot se sprema u Firebase Realtime Database, jednostavnu NoSQL bazu podataka koju možemo podešavati u Firebase sučelju. NoSQL baze podataka razlikuju se od

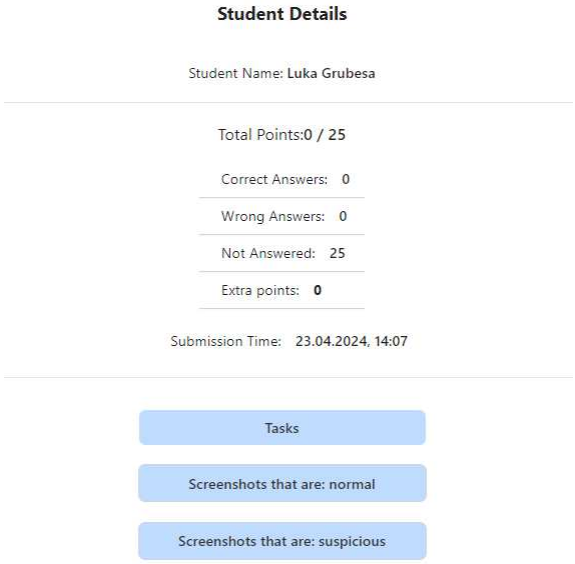
tradicionalnih relacijskih baza podataka. Dok relacijske baze koriste tablice za strukturiranje podataka, NoSQL baze koriste fleksibilnije strukture, kao što su dokumenti, parovi ključ-vrijednost, grafikoni ili široke kolone. NoSQL baze podataka mnogo su jednostavnije za implementaciju, no gubi mogućnost izvođenja kompleksnijih SQL kodova.

Firebase Realtime Database, kao NoSQL rješenje, omogućuje pohranu podataka u obliku JSON (JavaScript Object Notation) objekata, što olakšava strukturu podataka i omogućuje njihovo brzo ažuriranje i sinkronizaciju u stvarnom vremenu. Ovo je posebno korisno za aplikacije koje zahtijevaju brze promjene i ažuriranja, jer se podaci automatski osvježavaju za sve povezane korisnike kad god dođe do promjene.

Na slici 5 možemo vidjeti prototip takvog sučelja, gdje je svaki student reprezentiran kvadratom na kojemu piše njegovo ime, prezime i status ispita kojeg trenutno polaže. Klikom na bilo koji kvadrat prikazat će se pop-up menu gdje možemo vidjeti slike zaslona i kamere, kao i da li je ta slika sumljiva ili ne, kao što je prikazano na slici 6, uz detaljnije rezultate ispita za tog studenta.



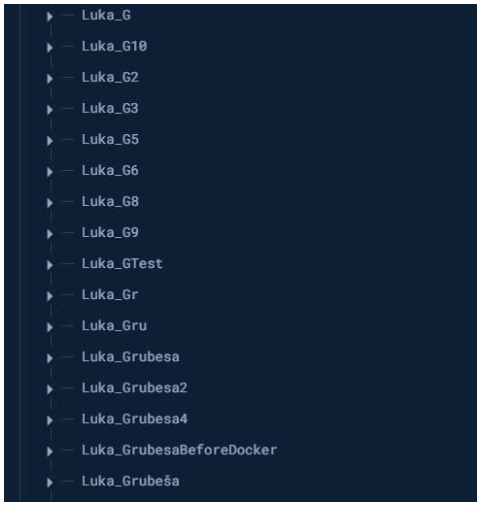
Slika 5: *Rani prototip sučelja za prikaz podataka o ispitu (izvor: Autor)*



Slika 6: *Polje sa dodatnim detaljima o ispitu za pojedinog studenta (izvor: Autor)*

3.1.3 Jednostavni Login i Register

Kako bismo znali koji student rješava koji ispit, razvili smo jednostavni Login i Register sistem, koji je koristio ime i prezime studenta, kao i lozinku testa kako bi pristupio ispitu. Taj sistem ćemo kasnije poboljšati da koristi Firebase Auth da, samo koristeći unipu email, mogu pristupiti platformi. Pomoću email-a također možemo izvući ime i prezime korisnika kako bi izbjegli manualni unos polja, te koristiti istu adresu kao identitet korisnika, a ne njegovo ime i prezime, u slučaju da dvije osobe imaju isto, kao što je prikazano na slici 7.



Slika 7: *primjer osobe sa istim imenom i prezimenom (izvor: Autor)*

3.1.4 Izrada jednostavnog Ispita

Kako bi definirali strukturu testa moramo prvo razumijeti koje komponente svaki ispit mora sadržavati

1. Header, koji sadrži identifikacijski broj testa
2. Prozor na kojoj se prikazuje preostalo vrijeme
3. Prozor na kojem se pojavljuje kod zadatka, te
4. Prozor sa pitanjima, u točno/netočno (TF) formatu

Prvi prototip, prikazan na slici 8, napravljen je u programu Figma (program za razvoj prototipa aplikacija, od dizajna do implementacije samih funkcionalnosti), te je sadržavao par UX (user experience) problema, glavni od kojih je da su rezultati, koji su prikazani na desnoj strani zaslona, bili neintuitivni tokom polaganja testa, jer bi student vidio praznu polovicu zaslona dok ne bi predao ispit, pa smo rezultate pomakli na odvojeni prozor koji se pojavljuje tek nakon što je student predao ispit. Time rezultati mogu biti opširniji, što je dovelo mogućnost dodavanja posebnog obrazloženja za svako pitanje.

Slika 8: *prototip izgleda samog ispita (izvor: Autor)*

3.1.5 Dodavanje dodatnih zadataka u ispit

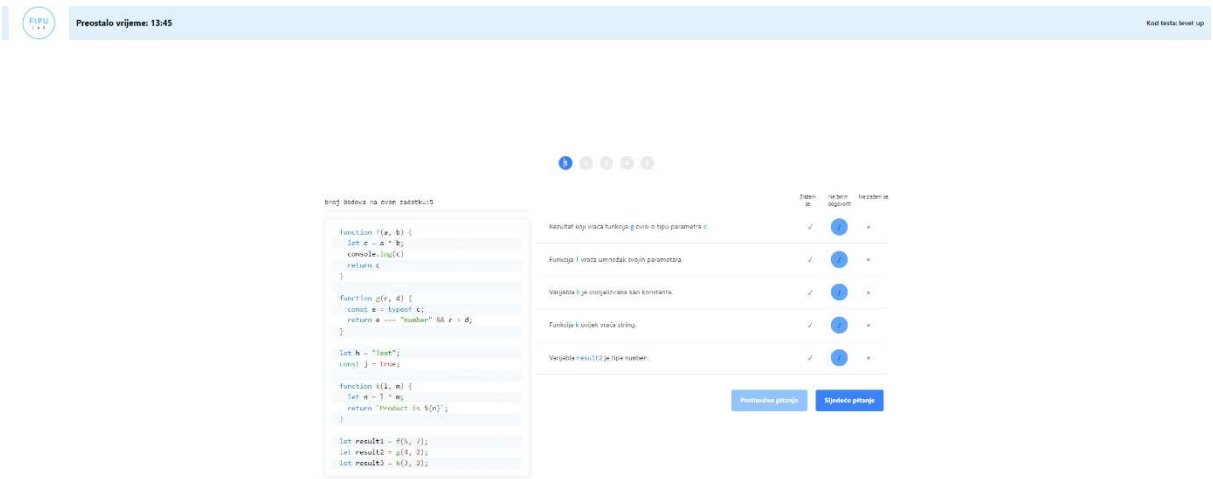
Do sada imali smo samo jedan zadatak kojeg bi rješili te dobili rezultate. Kako bi napravili MVP (Minimum Viable Product), trebamo dodati mogućnost više zadataka. Način na koji je ta

funkcionalnost implementirana jest da smo korištenjem abstrakcije napravili „Test container“, komponentu koja sadrži sve zadatke, gdje bi svaki zadatak bio instanca komponente „Test“, dok u samom ispitu imamo komponente za Navigacijsku traku, pitanja, prozor sa kodom ovisno na koji se zadatak odnosi, te druge varijable koje izvlačimo iz baze podataka. Na slici 9 vidimo primjer krajnje verzije „Test“ komponente. Na njoj se vidi primjer kako Vue.js koristi događaje, tj „events“ kako bi pokrenuli određenu funkciju, koju je izazvala komponenta u Test komponenti.

```
<Test
        v-if="tasks.length > 0"
        :containerTask="tasks[currentTaskIndex]"
        :containerTestCode="testCode"
        :remainingTime="remainingTime"
        :isLastTask="isLastTask"
        @updateTaskAnswers="handleUpdateTaskAnswers"
        @submitTest="submitTest"
        @nextTask = "nextTask"
        @previousTask = "previousTask"
        @unlock = "handleUnlock"
        class="ease-in"
      />
```

Slika 9: *komponenta Test, koja se nalazi u komponenti TestContainer (izvor: Autor)*

Nakon refaktoriranja koda i strukture frontend dijela aplikacije, na slici 10 dobivamo osnovni dizajn na kojem ćemo nadograđivati nove funkcije. Glavni fokus kod dizajniranja sučelja jest transparentnost, cilj nije bio napraviti kompleksno, već korisno i čitljivo sučelje.



Slika 10: *Finalna struktura Testa na kojem student polaže ispite (izvor: Autor)*

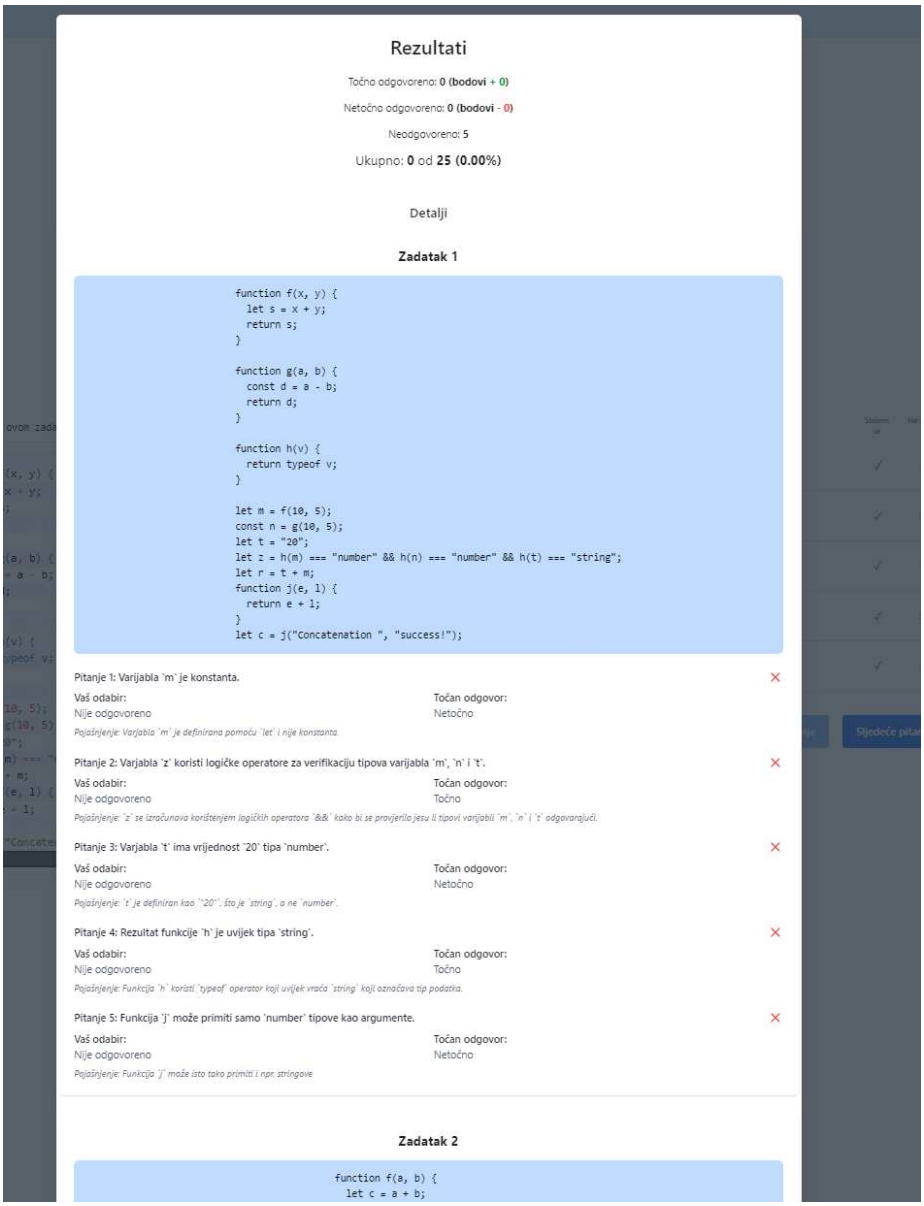
3.1.6 Označavanje sintakse

Kako bismo kod učinili čitljivijim za studente, koristimo Prism.js, library koja omogućuje definiranje programskog jezika u kojem je pisan tekst, poput JavaScript-a ili Python-a, te ga automatski pretvara u čitljiv kod s pravilnim označavanjem sintakse. Prism.js podržava širok spektar tema i varijacija boja, što omogućuje prilagodbu vizualnog prikaza prema potrebama aplikacije. Za potrebe našeg projekta odabrali smo temu prismjs-coy, koja se skladno uklapa u cjelokupni dizajn aplikacije, kao što je prikazano na slici 11.

Slika 11: *prikaz primjene prismJs na tekst koda (izvor: Autor)*

3.1.7 Zaslon sa rezultatima

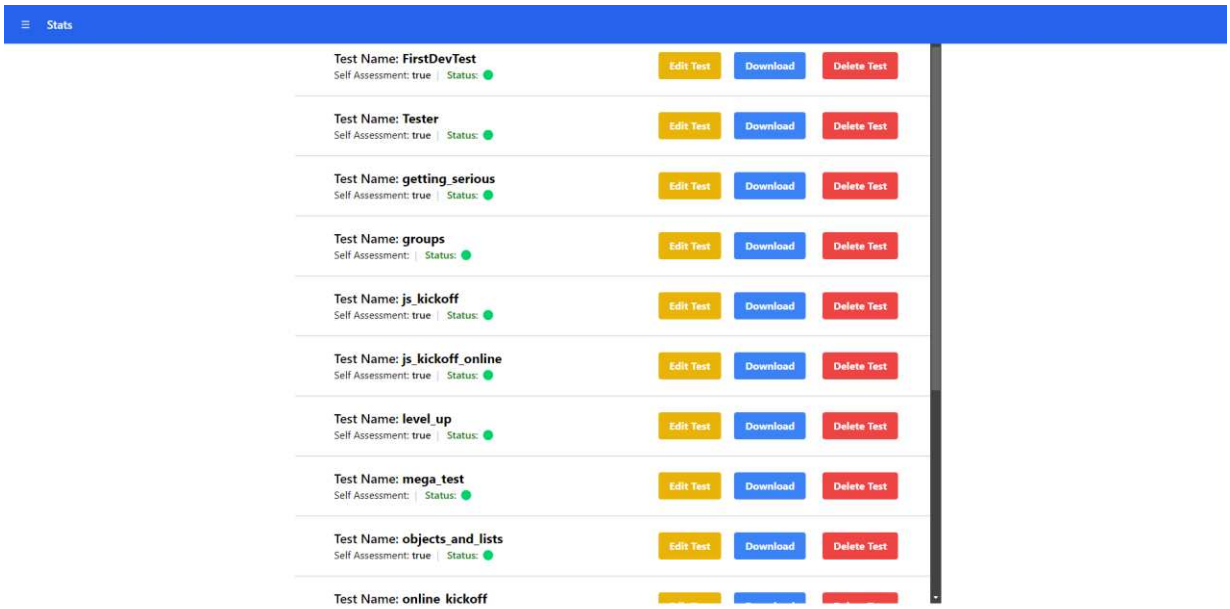
Nakon što student završi ispit, prikazuje mu se pop-up prozor s rezultatima, koji uključuje ukupan broj bodova, broj točnih i netočnih odgovora, postotak uspješnosti te, ako je opcija odabrana u testu, detalje o svakom pojedinačnom pitanju i zadatku. Finalna verzija ove komponente prikazana je na slici 12. U detaljnom pregledu rezultata prvo se prikazuje JavaScript kod ispita (implementirano prema sugestiji jednog od studenata koji su testirali aplikaciju), zatim slijedi broj i tekst svakog pitanja, te oznake "x" ili kvačica koje označavaju točnost odgovora. Također se prikazuje točan odgovor i objašnjenje za svako pitanje. Nakon pregleda rezultata, korisnik može zatvoriti prozor, što ga vraća na početnu stranicu i automatski odjavljuje iz sustava, kako bi se osigurala privatnost i spriječilo da drugi studenti dobiju pristup njegovom računu.



Slika 12: *prikaz rezultata nakon završetka ispita (izvor: Autor)*

3.1.8 Odabir ispita

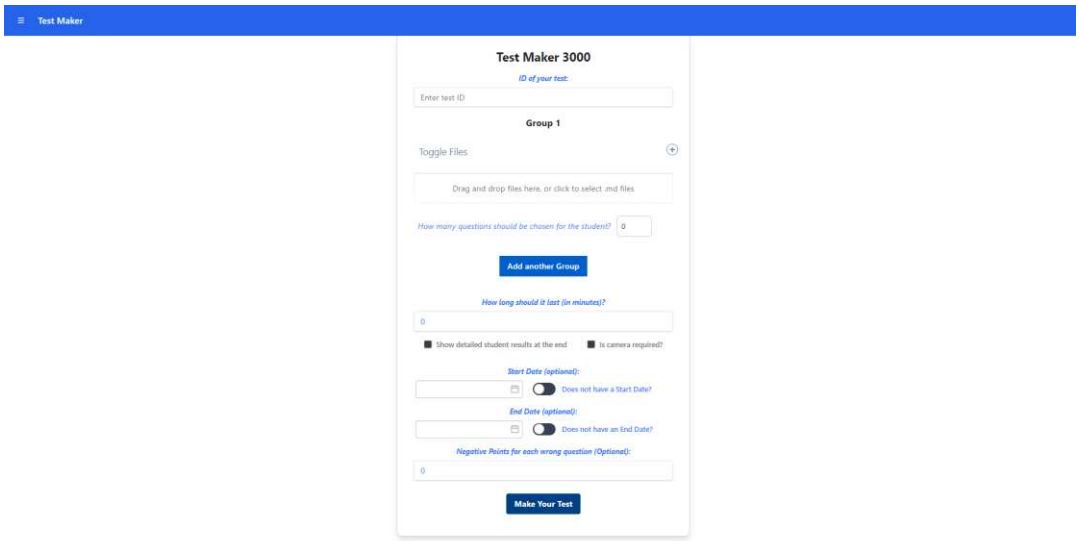
S obzirom na to da osoba koja izrađuje ispite treba imati pregledan pristup svim prethodno kreiranim ispitima, implementirana je stranica koja omogućuje pregled svih ispita koje je korisnik izradio. Ova stranica pruža ključne informacije kao što su naziv testa, njegov trenutni status (aktivnost ili neaktivnost), te opcija za pregled detaljnih rezultata nakon predaje ispita. Osim toga, korisnik ima pristup gumbima za uređivanje testa, preuzimanje podataka o bodovima u Excel formatu, kao i gumb za brisanje odabranog testa, što je prikazano na slici 13.



Slika 13: *Stranica za prikaz svih testova za pojedinog korisnika (izvor: Autor)*

3.1.9 Izrada ispita

Radi postizanja što većeg nivoa automatizacije, testove generiramo automatski u .md (MarkDown) formatu, koji nam uz same vrijednosti teksta daje i podatke o indentacijama redata, boji pojedinih slova, razmacima te drugim korisnim informacijama koje koristimo kako bi izvukli što više podataka. Sučelje mora sadržavati formu u kjojoj korisnik može unijeti identifikacijski broj ili naziv novog ispita, duljinu trajanja testa, datume od kada i do kada ispit je valjan, da li test zahtjeva kameru za prisustvovanje, da li će se studentu prikazati detaljni rezultati na kraju ispita, te .md datoteke koje šaljemo u FastAPI endpoint, pretvarajući ga u oblik čitljiv databazi (ovaj proces detaljnije ćemo objasniti u sljedećem poglavlju). Nakon stvaranja ispita on mora biti spremljen u bazu podataka, prikazan na sučelju za odabir ispita, te dostupan za polaganje. Kasnije također dodaje se opcija negativnih bodove za netočne odgovore, te opciju da možemo odabrati koliko zadataka od ukupnih za tu grupu nasumično dajemo studentu. Finalna verzija tog sučelja vidljiva je na slici 14.



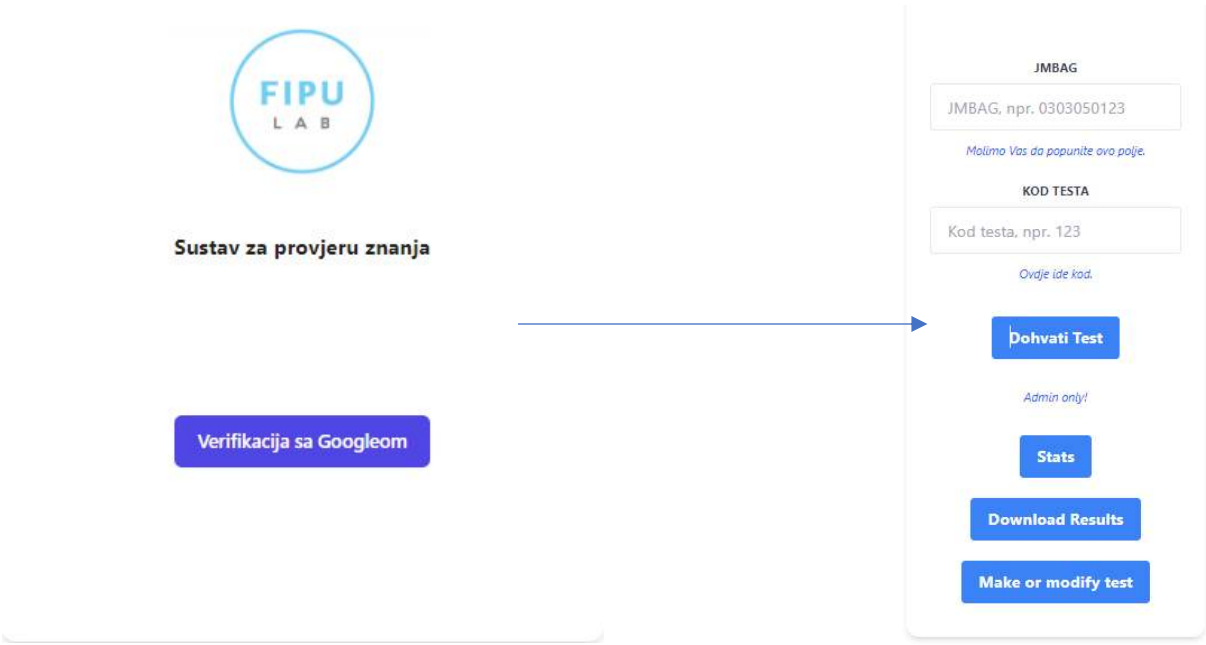
Slika 14: *Stranica za izradu ispita (izvor: Autor)*

3.2 Korištenje finalne verzije aplikacije te njezine funkcionalnosti

3.2.1 Početni zaslon

Pri ulasku u aplikaciju, korisniku je dostupna jedina opcija – verifikacija putem emaila. Nakon uspješne verifikacije, vidljivo na slici 15, korisniku se omogućuje dohvaćanje testa, pri čemu je potrebno unijeti JMBAG i kod testa. Klikom na gumb „Dohvati Test“ prelazi na stranicu za polaganje ispita. Važno je napomenuti da je unos JMBAG-a potreban samo prilikom prvog pristupa ispitu; aplikacija automatski popunjava ovo polje na temelju prethodnih unosa pri narednim posjetima.

Druge opcije koje su prisutne samo emailovima sa admin statusom uključuju prikaz svih prošlih ispita, gdje nadalje mogu manipulirati, pregledavati i brisati određene podatke o istima, stranica za izradu novog ispita, te opcija za preuzimanje rezultata bilo kojeg ispita u xlsx formatu.



Slika 15: *Prikaz početnog zaslona prije i poslije autentifikacije (izvor: Autor)*

Prva provjera, prikazana na slici 16, utvrđuje je li korisnik već pristupio određenom testu. Ako je korisnik već polagao taj test, prikazuje mu se "alert" obavijest i onemogućava mu se ponovni ulaz. Važno je napomenuti da samo korisnici s administratorskim ovlastima imaju mogućnost višestrukog polaganja istog ispita.

```
  try {
    const formData = new FormData();
    formData.append('email', userRef.value.email);
    formData.append('test_code', test_code.value);
    const response = await axios.post(`${API_BASE_URL}/hasTakenTest`,
formData, {
      headers: {
        'Content-Type': 'multipart/form-data'
      }
  });

    if (response.data.hasTakenTest === true && !isAdmin.value) {
      // uncomment for production
      alert("You have already taken this test.");
      return;
    }
```

Slika 16: *Kod za provjeru da li je korisnik već pristupio testu (izvor: Autor)*

Druga provjera, prikazana na slici 17, provjerava postoji li test s unesenim kodom. Ako test ne postoji, korisniku se onemogućava pristup.

```
  const checkTestResponse = await axios.get
  (`${API_BASE_URL}/checkTestExistence/${test_code.value}`);
    if (!checkTestResponse.data.exists) {
      alert("This test does not exist.");
      isLoading.value = false;
      return;
    }
```

Slika 17: *Prikaz provjere postojanja navedenog testa (izvor: Autor)*

Zadnja provjera na slici 18 utvrđuje je li ispit trenutno dostupan u zadanom vremenskom razdoblju. Ako trenutni datum nije unutar definiranog perioda dostupnosti ispita, korisnik će biti obaviješten da ispit trenutno nije dostupan. U slučaju da će ispit biti dostupan u budućnosti, korisniku će biti prikazano preostalo vrijeme do početka ispita, s preciznim prikazom sati i minuta, pod pretpostavkom da korisnici neće zahtjevati ispit kojim je početak predugo u budućnosti.

```
const response = await
axios.get(`${API_BASE_URL}/getTestDetails/${test_code.value}`);
  let message = "";
    if (response.data && response.data.start_date && response.data.end_date) {
      const startDate = new Date(response.data.start_date);
      const endDate = new Date(response.data.end_date);
      const currentDate = new Date();
      if (currentDate < startDate) {
      // The test has not started yet
      const timeToStart = startDate - currentDate; // This will give time in
milliseconds
      const hoursToStart = Math.floor(timeToStart / (1000 * 60 * 60));
      const minutesToStart = Math.floor((timeToStart % (1000 * 60 * 60)) /
(1000 * 60));

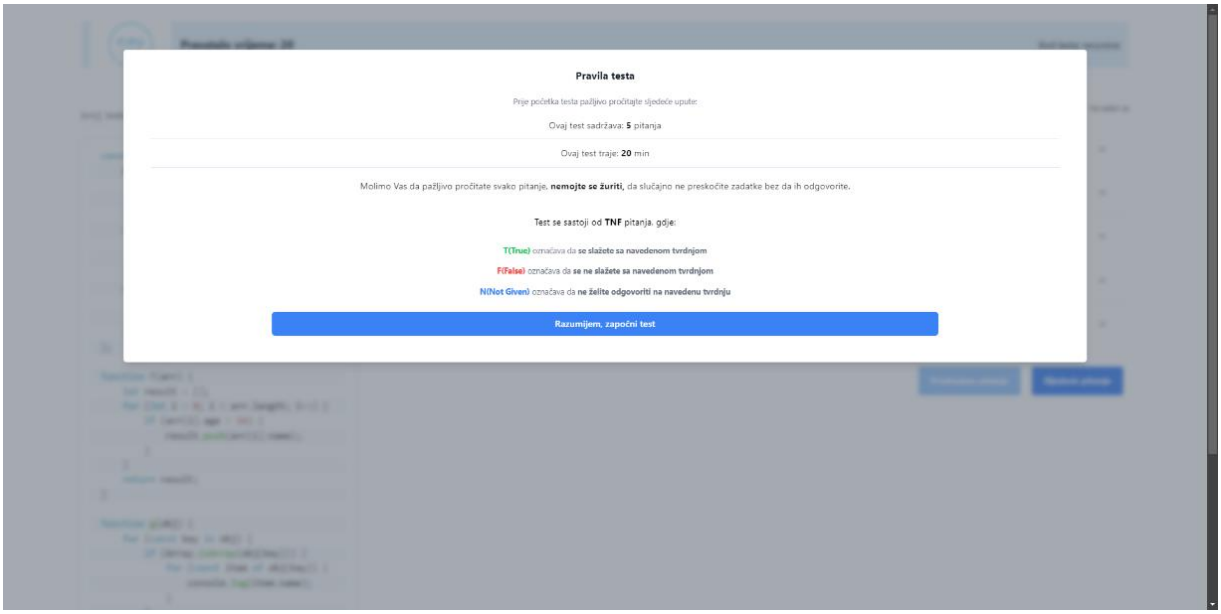
      message = `The test has not started yet. It will be available in
${hoursToStart} hours and ${minutesToStart} minutes.`;
    } else if (currentDate > endDate) {
      // The test has already ended
      message = "This test is no longer available.";
    }
```

Slika 18: *Prikaz dohvaćanja ispita, te provjere da li je test još uvijek validan (izvor: Autor)*

3.2.2 *Zaslon Ispita*

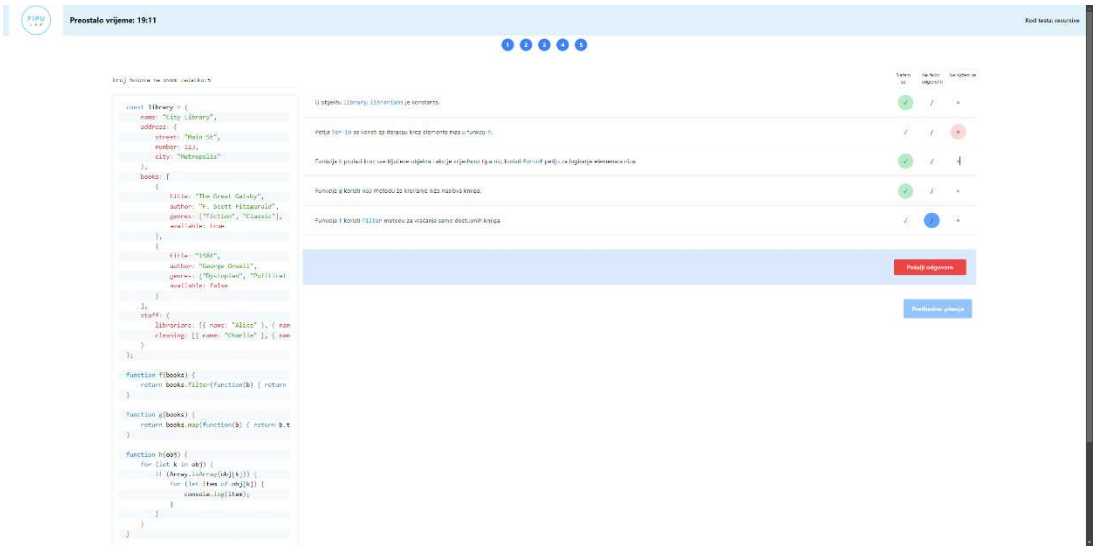
Krajnji zaslon ispita zadržava sličan dizajn kao i MVP verzija, s dodatkom treće opcije za odabir "nisam siguran". Ova opcija je uvedena radi implementacije negativnih bodova u ispitima, omogućujući korisnicima da izbjegnu gubitak bodova ako odluče ne odgovoriti na pitanje. Negativni bodovi mogu se definirati prilikom kreiranja ispita, kao i obaveznost prisustva kamere tijekom rješavanja testa. Na slici 19 prikazan je zaslon s uputama za ispit koji ne zahtijeva korištenje kamere. Važno je napomenuti da vrijeme za rješavanje ispita ne započinje sve dok korisnik ne pritisne gumb za početak ispita, a zaslon ostaje zamućen dok se ta radnja ne izvrši.

Tijekom rješavanja ispita, svakih 5 sekundi automatski se šalju podaci o kameri, uključujući trenutnu snimku kamere i zaslona, vrijeme rješavanja, te druge informacije koje su važne za određivanje potencijalnih nepravilnosti tijekom testa. Ovi podaci dostupni su na administrativnoj stranici za daljnju analizu i provjeru.

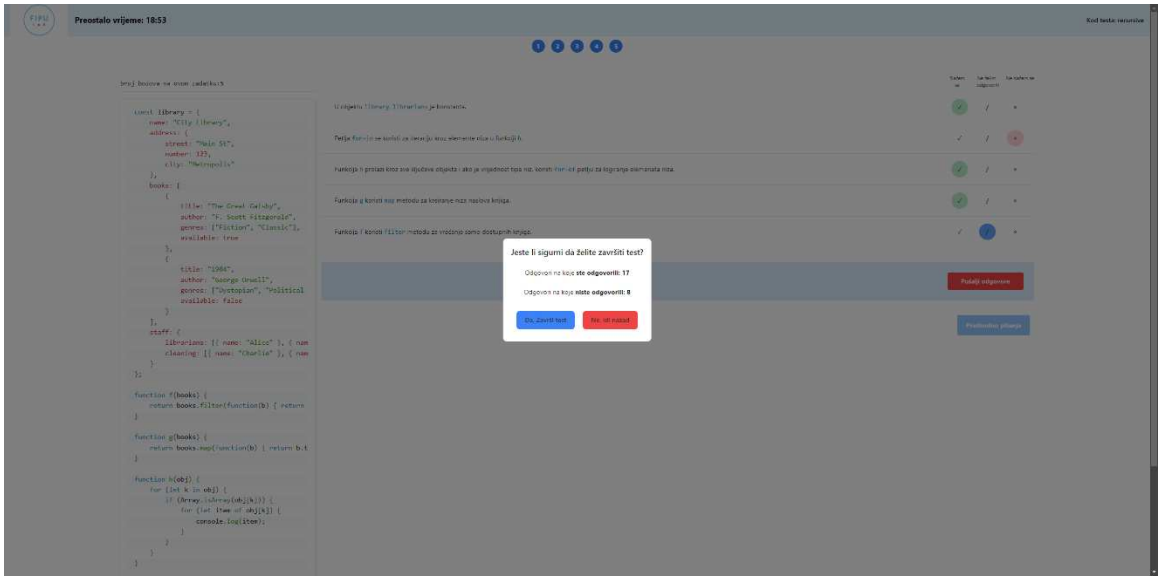


Slika 19: *Zaslon sa uputama za test koji ne zahtjeva kameru (izvor: Autor)*

Svaki prelazak na drugi zadatak popraćen je kratkom animacijom kako bi dali korisniku vizualni znak da se događa promjena, što je bio jako važni detalj u poboljšavanju UX aspekta ispita korisniku. Također, kao što se vidi na slici 20, dolaskom na zadnji zadatak pojavljuje se gumb za završetak ispita. Pri pritisku na gumb pojavi se vizualna komponenta, vidljiva na slici 21, koja traži ako je korisnik siguran da želi završiti ispit, te mu se također prikazuje broj odgovorenih i neodgovorenih pitanja.

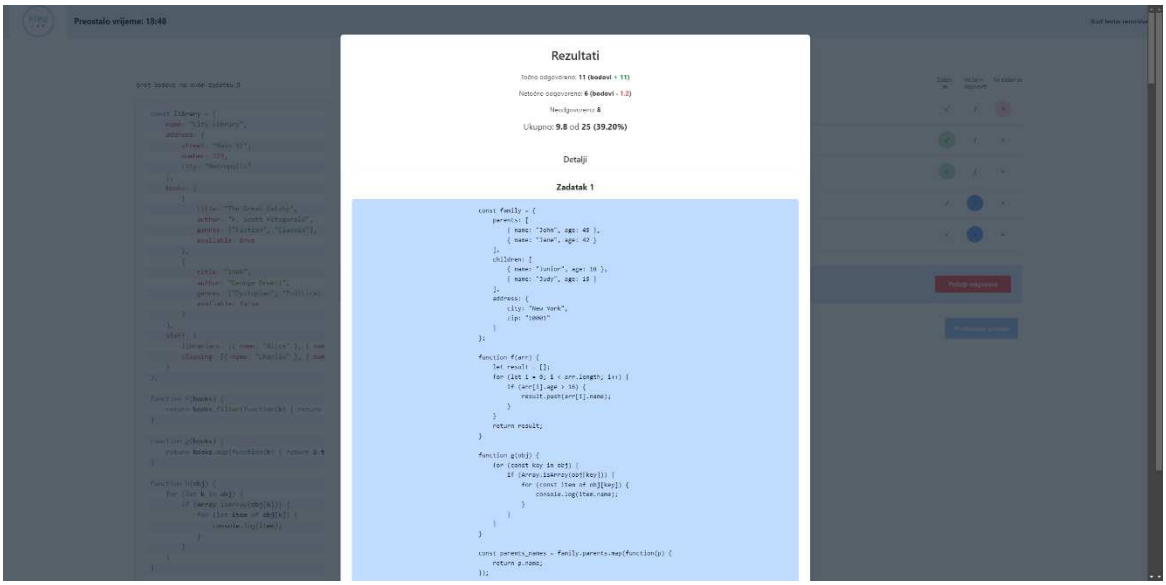


Slika 20: *Prikaz zaslona kada se korisnik nalazi na zadnjem zadatku (izvor: Autor)*



Slika 21: *Prikaz modala za potvrdu prije predaje ispita (izvor: Autor)*

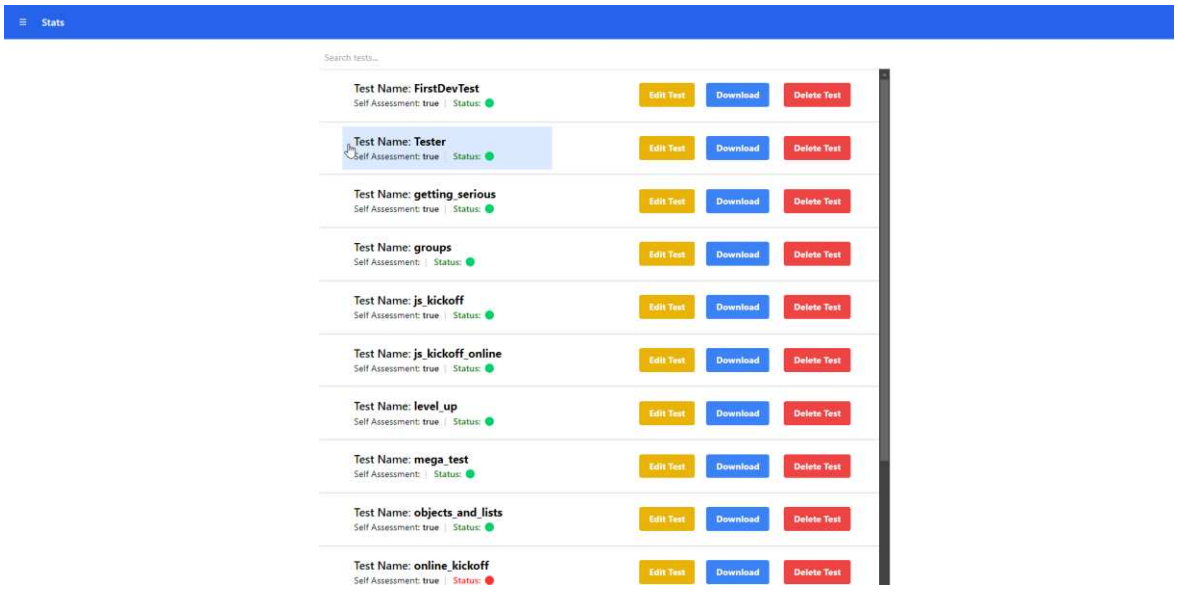
Nakon potvrde završetka ispita, korisniku je prikazana vizualna komponenta rezultata, također prikazana na slici 22, gdje se vidi ostvareni broj bodova, broj negativnih bodova te ukupan broj bodova. Ispod odjeljka nalaze se odgovori na sva pitanja, te ako je korisnik točno ili netočno odgovorio na njih. Na kraju komponente nalazi se gumb koji odjavljuje korisnika, te ga šalje na početnički zaslon.



Slika 22: *Prikaz revizioniranog zaslona sa rezultatima (izvor: Autor)*

3.2.3 Pregled dostupnih ispita

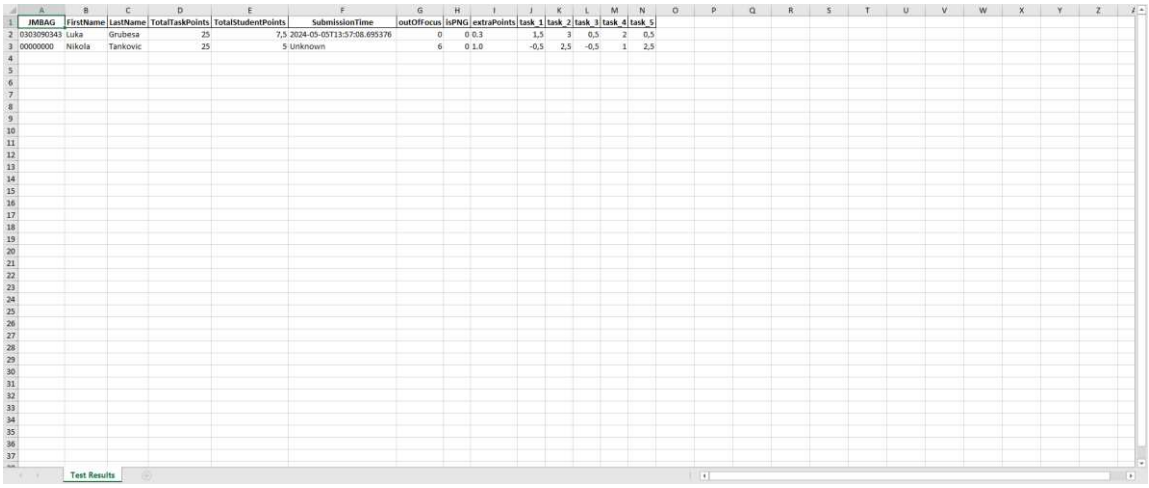
Na početnom zaslonu pritiskom na gumb „Stats“, administratora vodi na stranicu u kojoj se nalaze svi testovi koji je napravio, zajedno sa podatcima o statusu istih; crveni status označava ispit koje m je prošao rok validnosti, dok su zeleni još uvijek dostupni, kao što je vidljivo na slici 23.



Slika 23: *Pregled dostupnih ispita i izbor pojedinog testa (izvor: Autor)*

Pored svakog ispita nalaze se tri gumba koje predstavljaju moguće radnje nad tim ispitom. Gumb „Edit test“ vodi korisnika u sučelje koje emulira prikaz testa, sa mogućnošću mjenjanja teksta bilo kojeg zadatka, koda ili odgovora istog.

Gumb „Download“ preuzima podatke ispita u xlsx formatu, kao i na početnom zaslonu. Excel datoteka koju preuzimamo sadrži podatke o svakom studentu, ukupni broj bodova postignuti na ispitu, outOfFocus i isPNG vrijednosti, kao i ostvareni broj bodova na svakom zadatku. Primjer excel datoteke ispita vidljiva je na slici 24.



JMBAG	FirstName	LastName	TotalTaskPoints	TotalStudentPoints	SubmissionTime	outOfFocus	isPNG	extraPoints	task_1	task_2	task_3	task_4	task_5
0303090343	Luka	Grubesa	25	7,5	2024-05-05T13:57:08.695376	0	0	0.3	1,5	3	0,5	2	0,5
00000000	Nikola	Tankovic	25	5	Unknown	6	0	1.0	-0,5	2,5	-0,5	1	2,5

Slika 24: *Prikaz xlsx dokumenta nakon preuzimanja rezultata za pojedini ispit (izvor: Autor)*

Treći gumb, "Delete Test", omogućuje brisanje odabranog ispita iz baze podataka, kao i njegovo uklanjanje s korisničkog sučelja.

Uz brisanje ispita, također se brišu i podaci o studentima povezani s tim ispitom, kao i snimke zaslona i kamere koje su snimljene tijekom provođenja ispita. Javascript Kod na slici 25 prikazuje logiku koja se koristi za brisanje ispita, uključujući snimke zaslona i kamere. Važno je napomenuti da rezultati koje su studenti ostvarili tijekom ispita nisu obrisani, kako bi se zadržala mogućnost kasnijeg uvida u te podatke.

Ova funkcija poziva se unutar asinkronog wrappera pomoću *asyncio* biblioteke, što je neophodno za ispravno funkcioniranje. Klikom na tekstualni dio bilo kojeg ispita korisnik se preusmjerava na zaslon za pregled podataka o ispitu.

```
def delete_test(test_id: str) -> bool:
    # Reference to the test that needs to be deleted
    test_ref = db.reference(f"/tests/{test_id}")
```

```
    if not test_ref.get():
        # If the test doesn't exist, return False indicating failure to delete
        print(f"Test {test_id} does not exist.")
        return False

    # Delete the test from the tests collection

    # Attempt to delete the test from the tests collection
    test_ref.delete()
    print(f"Deleted test {test_id} from tests collection.")

    # Now, find and delete the test from all students' records
    students_tests_ref = db.reference("/studentTests")
    all_students_tests = students_tests_ref.get()

    # delete the test from all students' records
    # if all_students_tests:
    #     for student, tests in all_students_tests.items():
    #         if test_id in tests:
    #             # If the test_id is found in the student's tests, delete it
    #             student_test_ref =
students_tests_ref.child(f"{student}/{test_id}")
    #             student_test_ref.delete()
    #             print(f"Deleted test {test_id} from student {student}'s
records.")

    if not delete_screenshots_for_test(test_id):
        print(f"Failed to delete screenshots for test {test_id}.")
        return False

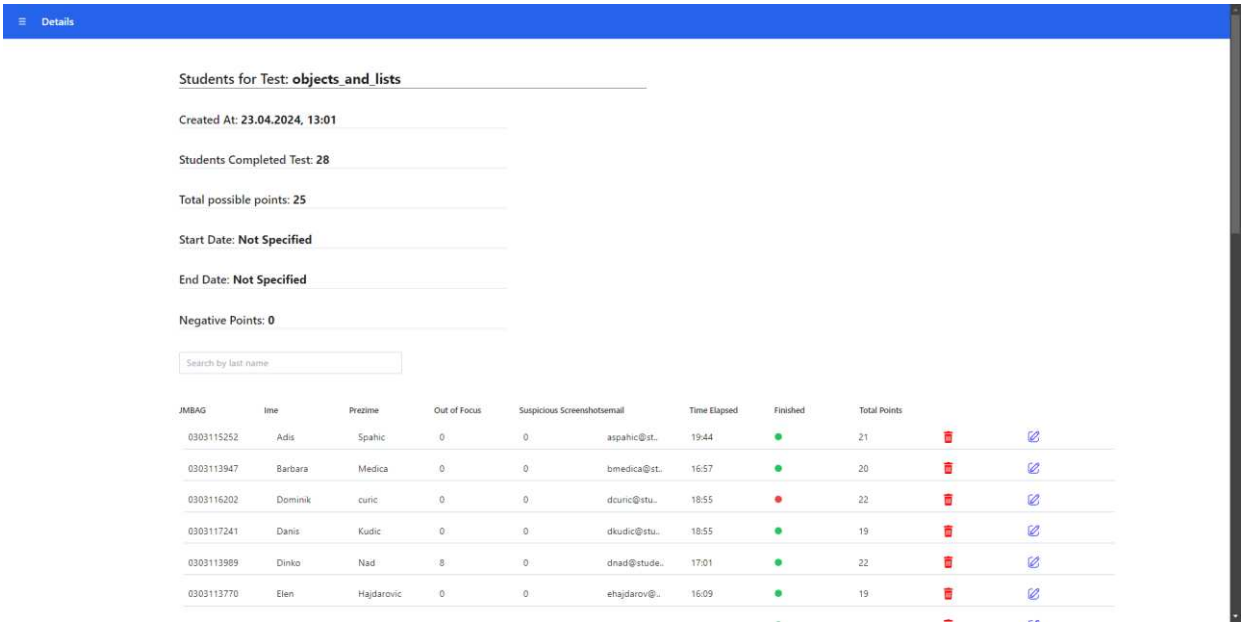
    return True
```

Slika 25: *Javascript kod za brisanje određenog ispita (izvor: Autor)*

3.2.4 Administrativno sučelje

Administrativno sučelje pruža detaljan pregled podataka o ispitu i njegovim polagateljima te nudi dodatne opcije za manipulaciju tim podacima. Kao što je prikazano na slici 26, svaki ispit uključuje informacije o vremenu stvaranja, broju studenata koji su završili ispit, vremenskom razdoblju u kojem se ispit može polagati, te broju negativnih bodova definiranih prilikom kreiranja ispita.

Podatkovno polje, u kojem su prikazani studenti, njihovi bodovi i ostali relevantni podaci, razvijeno je od nule s naglaskom na brzinu, ažuriranje u stvarnom vremenu i kontrolu nad svakim redom u tablici. Svakom studentu dodijeljene su ključne informacije, poput JMBAG-a, ukupnog broja bodova, proteklog vremena i slično. Detaljniji pregled moguće je dobiti klikom na bilo koji redak, gdje su uz prikaz bodova dostupni i bodovi za svaki zadatak, vrijeme predaje testa, te sve snimke kamere i zaslona, ako postoje.

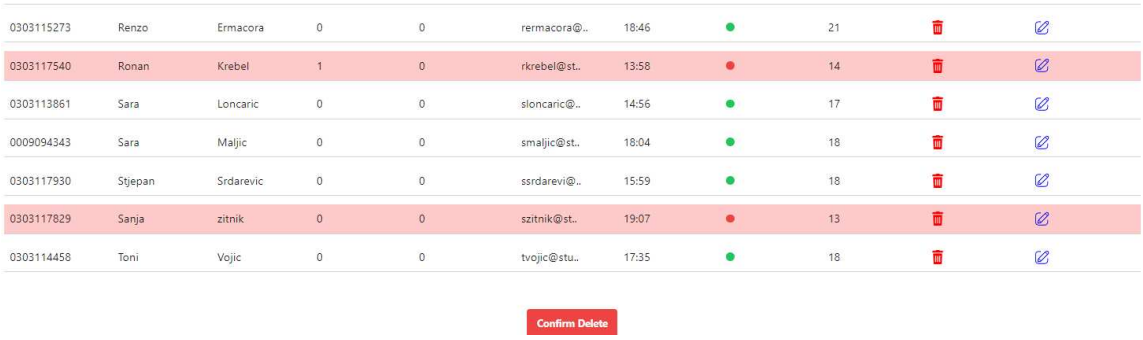


Slika 26: *Prikaz sučelja za pregled ispita (izvor: Autor)*

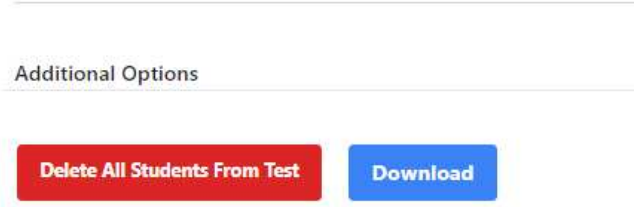
Jedna od važnih funkcionalnosti koje su dostupne u ovom sučelju je mogućnost brisanja studenata iz ispita, za što postoje dvije opcije:

Prva opcija omogućuje brisanje pojedinačnog studenta tako da se klikom na ikonu koša označi student kojeg želite ukloniti. Moguće je odabrati više studenata odjednom, a brisanje se potvrđuje klikom na gumb „Confirm Delete“, kao što je prikazano na slici 27.

Druga opcija, dostupna pod dodatnim opcijama, omogućuje brisanje svih studenata iz određenog ispita jednim klikom na gumb „Delete All Students From Test“, prikazan na slici 28. Gumb „Download“ nudi mogućnost preuzimanja podataka o ispitu u xlsx formatu, što je ista funkcionalnost kao i na stranici za pregled ispita.



Slika 27: *Biranje studenata za brisanje na određenom ispitu (izvor: Autor)*



Slika 28: *Dodatne opcije ispod podataka o ispitu (izvor: Autor)*

Dodavanjem polja za dodatne bodove, vidljivo na slici 29, studentima možemo dodati ili oduzeti bodove na pojedinom ispitu, ovisno o namjeni. Ti bodovi se spremaju pod posebno polje u bazi podataka, te su vidljivi na detaljnom pregledu studenta.

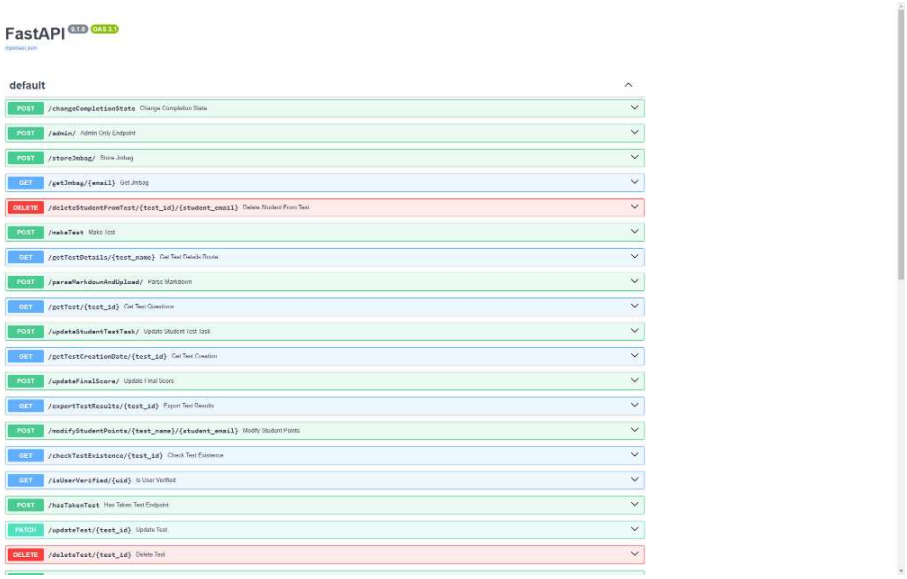
Klikom na zadnji gumb pojavit će se jednostavni pop up, vidljiv na slici x, gdje možemo dodavati ili oduzeti bodove. Primjerice unosom „-0.5“, studentu oduzeti će se 0.5 boda od ukupnog rezultata, dok bi unosom „2“ studentu nadodalo dva boda na ukupni rezultat.

Slika 29: *Prikaz prozora za manipulaciju dodatnih bodova (izvor: Autor)*

3.3 Opis odabranih sustava i algoritama

3.3.1 FastApi i Swagger sustav za testiranje endpointova

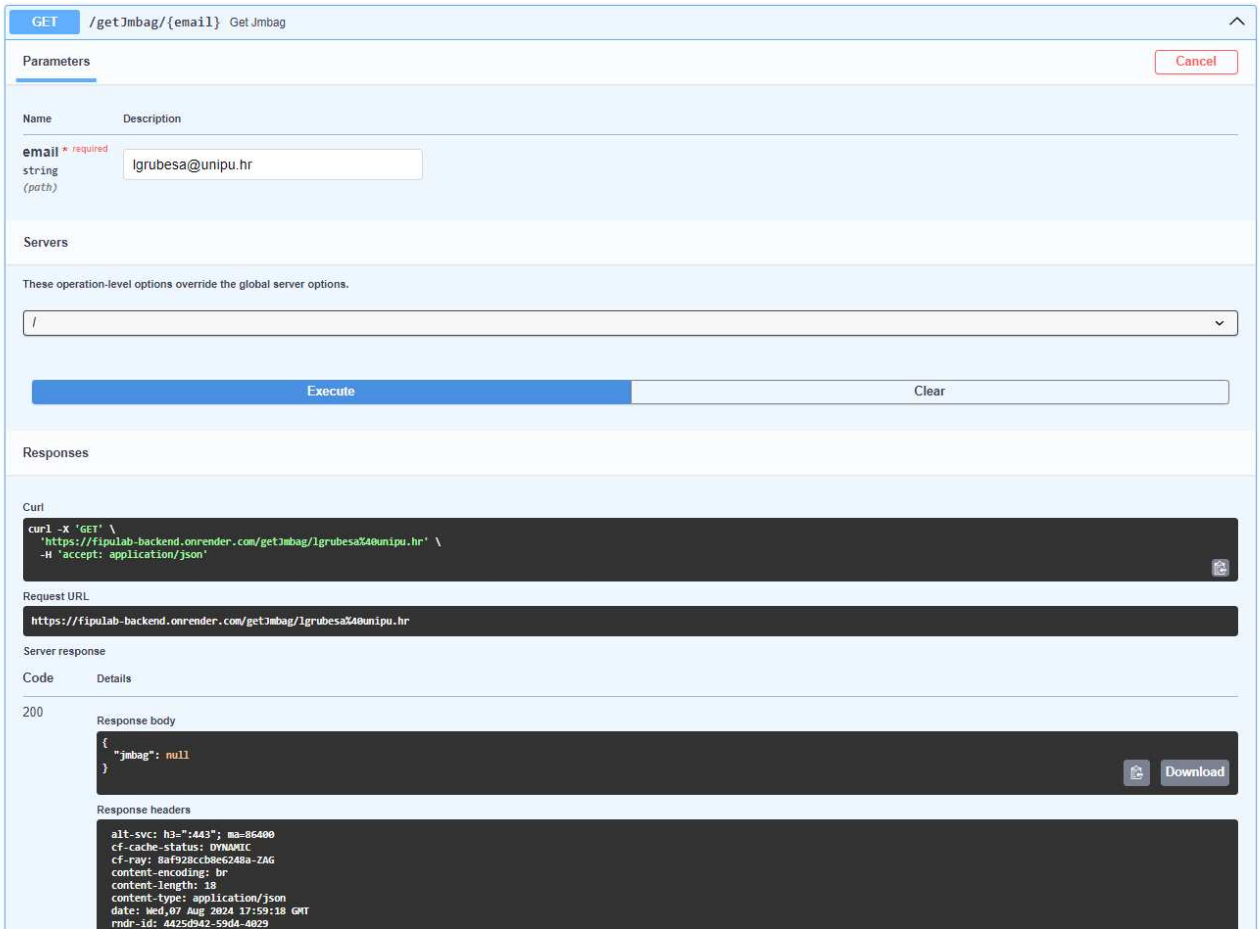
FastAPI je moderan web framework za izgradnju API-ja u Pythonu, koji nudi ugrađenu podršku za automatsko generiranje Swagger dokumentacije. Swagger je široko korišten alat za dizajniranje, izgradnju, dokumentiranje i testiranje RESTful web servisa. Na slici 30 prikazano je Swagger korisničko sučelje (UI), koje vizualno prikazuje sve endpointove koje servis prepoznaje, kategorizirajući ih prema tipu zahtjeva.



Slika 30: *Fast API Swagger sučelje (izvor: Autor)*

Klikom na bilo koji endpoint, korisnik dobiva sučelje za testiranje tog endpointa, u kojem se mogu vidjeti polja za unos podataka, polje za odgovor, kao i dodatna polja koja su dio funkcionalnosti određene rute. Na primjer, ruta koja bi trebala generirati poveznicu za preuzimanje Excel tablice prikazat će tu poveznicu u odgovoru.

U prikazanom primjeru na slici 31, demonstriramo jednostavnu GET rutu koja prima email korisnika u obliku stringa, te u odgovoru vraća JMBAG povezan s tim korisnikom, zajedno s detaljima o tipu odgovora i tijelom (body) koje sadrži JMBAG. Ovaj endpoint koristi se za automatsko popunjavanje polja za JMBAG prilikom prijavljivanja korisnika, ako je taj korisnik već ranije polagao ispit.



Slika 31: *Detaljniji pregled endpointa u Swagger sučelju (izvor: Autor)*

U servisu možemo podijeliti rute na tri kategorije;

1. Rute koje se bave radnjama nad ispitom, stvaranje i brisanje pojedinog ispita, parsiranje podataka i spremanje podataka vezanih za test
2. Rute koje dohvaćaju podatke vezane za korisnika, na primjer njegov JMBAG, ili provjeravanje da li je korisnik administrator
3. AI rute, koje se bave detekcijom lica, kamere te eksperimentalnog generativnog modela stvaranja ispita.

Na slici 32 možemo vidjeti primjer koda za stvaranje testa,gdje se „db.refrence“ odnosi na instancu Firebase baze podataka. Budući da koristimo noSQL databazu, ne moramo pisati SQL kod kako bismo spremili podatke, samo postavimo referencu na segment gdje želimo spremiti podatke, dati referencu na same podatke koje želimo spremiti, i Firebase automatski odrađuje spremanje istih.

```
def create_test(test_name, test_duration, isSelfAssessment, isCameraRequired,
```

```
start_time, end_time, negativePoints):
    test_ref = db.reference(f"/tests/{test_name}")
    if test_ref.get():
        raise Exception("Test already exists")
    test_ref.set(
        {
            "created_at": datetime.now().isoformat(),
            "test_duration": test_duration,
            "students": {},
            "isSelfAssessment": isSelfAssessment,
            "isCameraRequired": isCameraRequired,
            "start_date": start_time,
            "end_date": end_time,
            "negativePoints": negativePoints
        }
    )
```

Slika 32: *Kod za spremanje novog testa u Firebase Realtime DB (izvor: Autor)*

3.3.2 Login Sustav i firebase auth implementacija

Kako bismo osigurali da svi podaci budu šifrirani i sigurno zaštićeni, mogli smo razviti vlastiti sustav autentifikacije. Međutim, cilj ove aplikacije je osigurati stabilnost i jednostavnost u održavanju i nadogradnji. U našem slučaju, najpogodnija metoda autentifikacije je Firebase Auth, koji omogućuje prijavu putem Google računa. S obzirom na to da studenti koriste jedinstvene fakultetske email adrese, jednostavno smo prilagodili pravila tako da samo korisnici s emailom koji završava na „unipu.hr“ mogu pristupiti sustavu.

Prilikom pristupanja ispitu, ime i prezime studenta preuzimamo iz njegovog emaila, koji ujedno služi kao jedinstveni ključ za pohranu svih rezultata ispita. Ovaj pristup osigurava da ne dođe do dupliciranja korisničkih podataka, što dodatno povećava sigurnost sustava.

Slika 33 prikazuje dio koda koji upravlja prijavljivanje sa Google autentifikacijskim servisom (Google Auth). Važne metode uključuju; „getAuth“, koji vraća ili kreira novu instancu autentifikacije, „setPersistance“, koji služi da korisnika zadrži prijavljenim, a „browserSessionPersistance“ brine se da korisnika odjavi tek nakon što izađe iz Google Chrome-a.

```
async function handleGoogleSignIn() {
    const auth = getAuth();
    signInState.value = 'loading'; // UI feedback for loading state
```

```
    // Set the persistence
    setPersistence(auth, browserSessionPersistence)
        .then(() => {
            // Proceed with the sign in process
            const provider = new GoogleAuthProvider();
            return signInWithPopup(auth, provider);
        })
        .then((result) => {
            // Handle the result
            const user = result.user;
            userRef.value = user;
            displayName.value = user.displayName;

            // Additional checks and setup
            if (!user.email.endsWith('unipu.hr') &&
!user.email.endsWith('student.unipu.hr')) {
                auth.signOut();
                alert("Only unipu.hr email addresses are allowed.");
                signInState.value = 'idle';
            } else {
                signInState.value = 'success';
                isAdmin.value = checkIfUserIsAdmin(user);
                fetchJmbag(user.email);
            }
        })
        .catch((error) => {
            console.error(error);
            alert("Authentication failed: " + error.message);
            signInState.value = 'idle';
        });
```

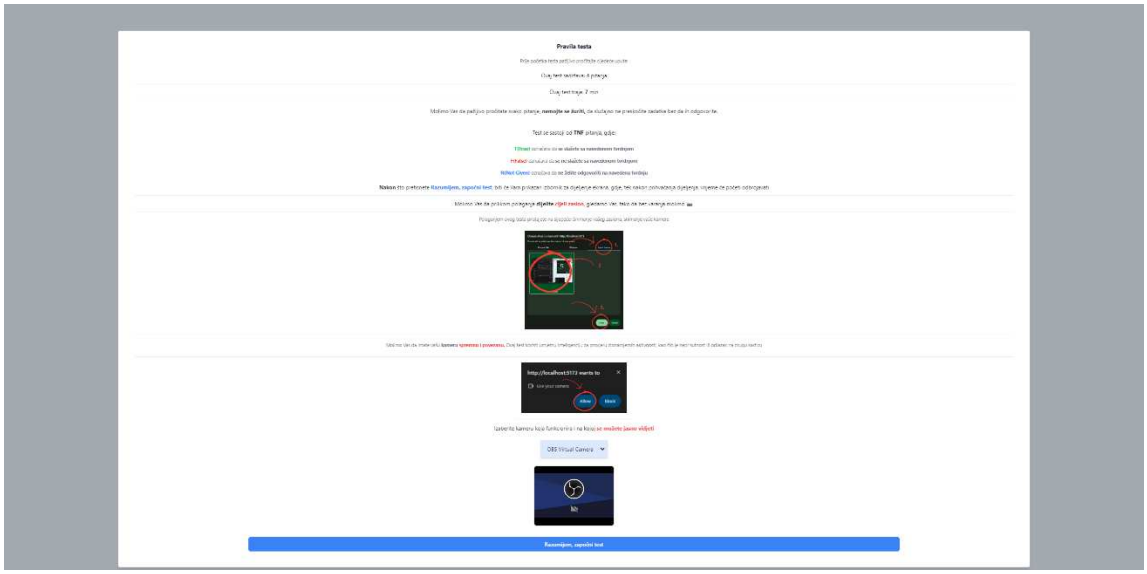
Slika 33: *Metoda za prijavljivanje pomoću Google Sign-in funkcionalnosti (izvor: Autor)*

3.3.3 AI implementacija i slanje podataka u bazu podataka tokom rješavanja ispita

Kako bismo spriječili mogućnost varanja tijekom rješavanja ispita, svaki student mora imati uključenu kameru. Prije nego što student započne ispit, prikazuje se pop-up s uputama za rješavanje ispita, kao što je prikazano na slici 34. U tom pop-up prozoru prikazuje se pregled kamere te se studentu omogućuje odabir kamere koju želi postaviti kao zadanu.

U ovom dijelu aplikacije implementirana je funkcija koja svakih 5 sekundi provjerava je li kamera previše mračna, čime osiguravamo da je slika jasna i da se mogu prepoznati eventualne nepravilnosti. Algoritam za ovu provjeru, prikazan na slici x, izračunava razinu

osvjetljenja na temelju snimke kamere i automatski obavještava studenta ako je osvjetljenje neadekvatno.



Slika 34: *Prikaz uputa za rješavanje ispita koja zahtjeva kameru (izvor: Autor)*

Funkcija „checkCameraBrightness“, prikazana na slici 35, koristi se za provjeru osvjetljenosti kamere. Prvo se stvara element „canvas“ i dobiva njegov kontekst za crtanje. Dimenzije platna postavljaju se prema dimenzijama videozapisa. Zatim se trenutni okvir videozapisa crta na platno i dobivaju se podaci o slici. Funkcija analizira piksele slike izračunavanjem svjetline svakog piksela, uzimajući prosjek crvenih, zelenih i plavih komponenti. Ako je svjetlina piksela manja od 25, taj piksel se smatra previše mračnim. Ako je više od 97% piksela previše mračno, postavlja se varijabla darknessWarning na true, što sprječava studenta da započne ispit.

```
const checkCameraBrightness = async () => {
      if(isCameraRequired) {
      const canvas = document.createElement('canvas');
      const context = canvas.getContext('2d');
      const width = videoPreview.value.videoWidth;
      const height = videoPreview.value.videoHeight;
      canvas.width = width;
      canvas.height = height;
      context.drawImage(videoPreview.value, 0, 0, width, height);
      const imageData = context.getImageData(0, 0, width, height);
      const data = imageData.data;
      let darkPixels = 0;
      let totalPixels = data.length / 4;

      for (let i = 0; i < data.length; i += 4) {
        let r = data[i];
```

```
        let g = data[i + 1];
        let b = data[i + 2];
        // Simple average for brightness calculation
        let brightness = (r + g + b) / 3;
        if (brightness < 25) darkPixels++; // Considering pixels with
brightness < 25 as "too dark"
      }
      if (darkPixels / totalPixels > 0.91) {
        console.log(darkPixels / totalPixels); // If more than 97% of pixels
are dark
        darknessWarning.value = true;

      } else {
        darknessWarning.value = false;
      }
```

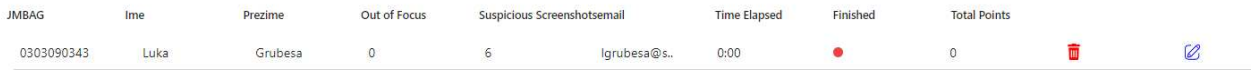
Slika 35: *Kod za provjeru svjetline kamere (izvor: Autor)*

Još jedan problem koji je trebalo riješiti jest mogućnost da student pomoću određenih metoda postavi sliku sebe kao feed s kamere i tako prevari algoritam za prepoznavanje lica. Ova se provjera vrši u dijelu koda poslužiteljskog sloja, prema sljedećem algoritmu:

1. Ako prethodna slika ne postoji, spremi trenutnu sliku kao privremeni podatak.
2. Ako prethodna slika postoji, učitavaju se obje slike te se pomoću funkcije structural_similarity iz paketa skimage.metrics provjerava njihova strukturalna sličnost. Ako je sličnost 99% ili više, povećava se vrijednost varijable isPNG za jedan, a novija slika zamjenjuje staru u privremenoj memoriji.
3. Ako slike nisu slične, novija slika zamjenjuje staru u privremenoj memoriji te se proces ponavlja.
4. Ako vrijednost varijable isPNG premaši 6, ispit se prekida, a ispitivač se obavještava.
5. Varijabla isPNG, zajedno s drugim ključnim varijablama vezanim za ispit, šalje se u bazu podataka.

Konačnu provjeru protiv varanja osigurava varijabla outOfFocus. Ona se izvršava tijekom cijelog ispita i prati fokus miša na zaslonu gdje se ispit polaže. U slučaju da student klikne na bilo koji drugi element na zaslonu (izborničku traku, polje na drugom monitoru itd.), varijabla se inkrementira. Ako vrijednost varijable outOfFocus dosegne šest ili više, ispit se zaustavlja i

rezultati se šalju u bazu podataka. Na slici 36 prikazan je primjer vizualizacije parametara koje ispitivač može pregledati, uključujući varijablu outOfFocus.



Slika 36: *Prikaz vizualizacije parametara u pregledu detalja ispita (izvor: Autor)*

3.3.4 Stvaranje ispita

Prema zahtjevima profesora, izrada ispita trebala bi primati markDown podatke, razdijeliti te podatke u čitljive dijelove, te ih spremiti u bazu podataka u obliku razumljivom programu kako bi mogao vizualizirati ispit. Za to koristimo regex ili „regular expression“. Regularni izrazi su snažan alat za pretraživanje i manipulaciju tekstualnim podacima. Omogućuju definiranje uzoraka koje tekst mora zadovoljiti kako bi bio prepoznat i obrađen na određeni način.

Sljedeći dio koda na slici 37 opisuje postupak parsiranja markDown sadržaja i njegovog spremanja u bazu podataka:

```
@router.post("/parseMarkdownAndUpload/")
async def parse_markdown(test_content: TestContent, test_id: str, task_number:
int, group_number: int):
    parsed_data = parse_markdown(test_content.markdown)
    # uploadParsedDataToDatabase(test_id, parsed_data, task_number,
group_number)
    uploadParsedDataToGroup(test_id, parsed_data, task_number, group_number,
test_content.randomSelectionCount)
    return {"message": "Test parsed and uploaded successfully"}
```

Slika 37: *Ruta za spremanje zadataka za ispit (izvor: Autor)*

Ova funkcija je asinkrona i povezana je s POST zahtjevom na ruti "/parseMarkdownAndUpload/". Prima podatke o sadržaju testa, njegovom identifikacijskom broju, broju zadatka i broju grupe. Funkcija parse_markdown, vidljiva na slici 38, koristi se za parsiranje markDown sadržaja, a potom se parsirani podaci prenose u funkciju uploadParsedDataToGroup koja ih sprema u bazu podataka. Na kraju funkcija vraća poruku o uspješnosti parsiranja i prijenosa podataka.

```
def parse_markdown(markdown: str):
    code_block_regex = re.compile(r"```javascript(.*?)```", re.DOTALL)
    question_regex = re.compile(r"1\. (.*?) \[(Točno|Netočno)\] \((.*?)\)",
re.DOTALL)
    code_match = code_block_regex.search(markdown)
    code = code_match.group(1).strip() if code_match else ""

    # Find questions
    questions = []
    for question_match in question_regex.finditer(markdown):
        question_text, answer, explanation = question_match.groups()
        questions.append(
            {
                "text": question_text.strip(),
                "answer": answer.strip() == "Točno",
                "explanation": explanation.strip(),
            }
        )

    return {"code": code, "questions": questions}
```

Slika 38: *Metoda za parsing datoteke pomoću regex notacije (izvor: Autor)*

Funkcija parse_markdown koristi dva regularna izraza za prepoznavanje specifičnih dijelova markDown sadržaja. Prvi regularni izraz, code_block_regex, traži JavaScript kod unutar markDown sadržaja. Ovaj izraz koristi se za prepoznavanje dijelova teksta unutar trostrukih navodnika s oznakom jezika javascript. Izraz koristi re.DOTALL opciju kako bi omogućio pretraživanje višelinijskog teksta.

Drugi regularni izraz, question_regex, koristi se za prepoznavanje pitanja unutar markDown sadržaja. Ovaj izraz traži linije koje započinju s brojem, zatim tekstom pitanja, te oznakama [Točno] ili [Netočno], uz dodatno objašnjenje unutar zagrada.

Unutar funkcije parse_markdown, prvo se pretražuje JavaScript kod koristeći code_block_regex, a zatim se traže pitanja koristeći question_regex. Pronađeni kod i pitanja zatim se strukturiraju u obliku „dictionary“ koji se vraća kao rezultat funkcije.

Ovaj proces omogućuje automatsko parsiranje i strukturiranje markDown sadržaja, što je ključno za daljnju obradu i vizualizaciju ispita u sustavu. Važno je napomenuti kako ova funkcija

specifično radi uz već postojeću strukturu ispitnog pitanja. U budućnosti, ako želimo mogućnost parsiranja drugih tipova datoteka, morati će se napraviti nova funkcija koja će vrijediti za taj tip dokumenta, ili promjeniti sveukupnu implementaciju parsiranja.

3.3.5 Mjenjanje testa i spremanje promjena

Jedna od važnijih implementacija koje je trebalo napraviti kako bi postigli MVP jest mjenjanje bilo kojeg dijela ispita, to jest implementirati tkzv. „Edit Mode“, jer budući da su testovi automatski generirani umjetnom inteligencijom, vrlo je velika šansa da dođe do „halucinacije“, noviji termin koji opisuje fenomen gdje umjetna inteligencija je sigurna u nekakav odgovor ili rezultat, dok taj rezultat zapravo nije točan. Zbog neizbježnosti prisustva halucinacije, moramo imati stranicu gdje imamo mogućnost pregledati ispit, uočiti pogreške, te promjeniti bilo kakve greške u pitanjima, koja će se zatim spremiti u bazu podataka.

Vizija je bila da se napravi klon pravog ispita, te da možemo mjenjati između „preview“ i „Edit“ tipa ispita, gdje je „Preview“ pregled samog ispita gdje se mogu uočiti greške, te „Edit“ mode gdje se te greške mogu popraviti, te spremiti te promjene u bazu podataka.

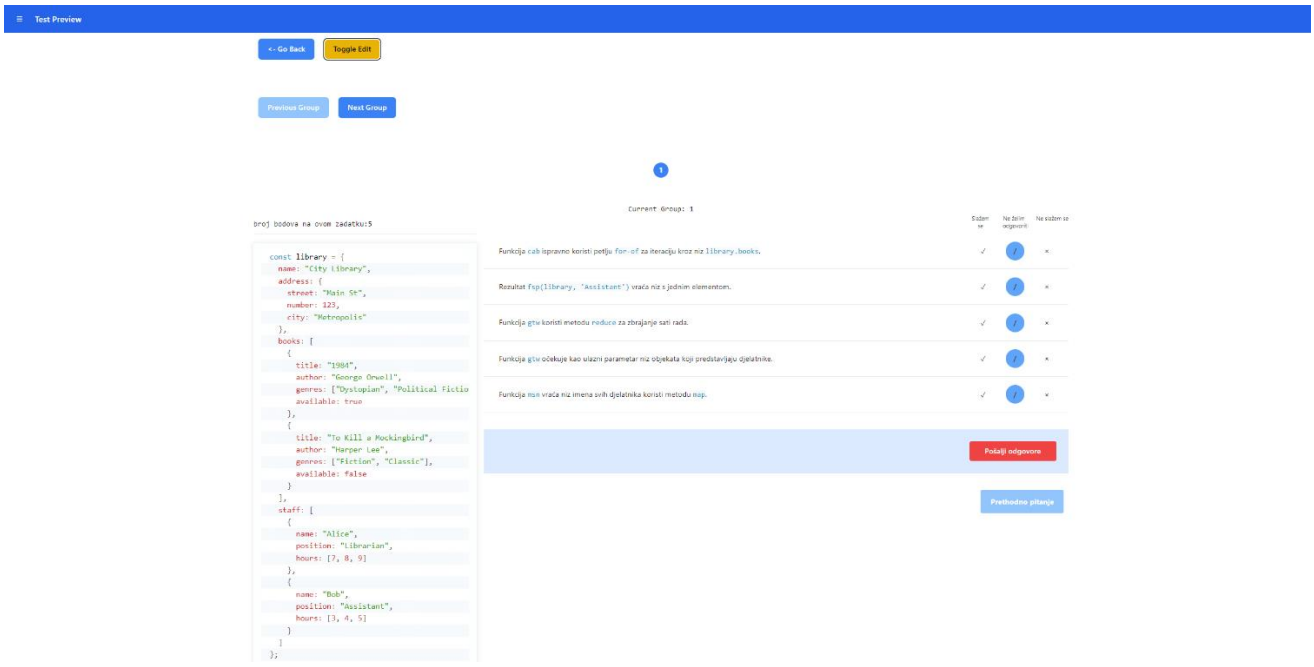
Jedan od glavnih izazova u realizaciji bio je kako upravljati ispitima koji sadrže više grupa s većim brojem zadataka nego što je prikazano na zaslonu. To je posebno važno zbog nasumičnog odabira zadataka, gdje je broj zadataka definiran tijekom izrade ispita. Prva ideja koja se nametnula bila je prikazivanje cijelog ispita sa svim grupama odjednom. Međutim, ovo bi stvorilo nekoliko problema: ne samo da bi se korisnici morali snalaziti u desecima pitanja, već bi i nedostatak vidljive razlike između grupa doveo do nepraktične implementacije i otežane navigacije kroz ispit.

Rješenje koje je implementirano, iako možda nije najpraktičnije, pokazalo se funkcionalnim. Uz prikaz zadataka na ispitu, dodane su strelice za navigaciju kroz grupe, kao i strelice za kretanje po zadacima unutar svake pojedine grupe. Ovaj pristup omogućava jasno razdvajanje grupa dok se istovremeno održava čitljivost i preglednost ispita.

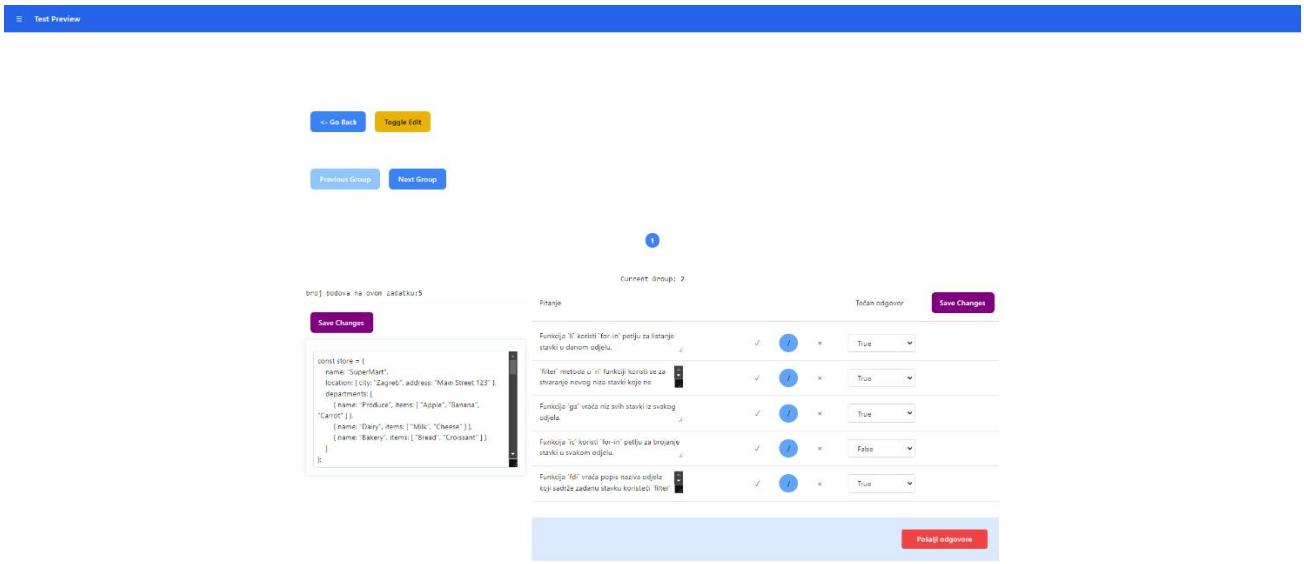
Slika 39 predstavlja „Preview mode“, koji uz sve značajke koje su prisutne na pravom testu, broji još dugme za promjenu grupe u testu, gumb za povratak na izbor testova, te dugme za

uključivanje „Edit“ moda rada, koji je prikazan na slici 40. U ovom zaslonu postoje tri kategorije koje se mogu promjeniti; javascript kod zadatka, tekst pitanja, i vrijednost odgovora. Promjene u kodu se mogu samostalno spremiti, dok pitanja i odgovori su spojeni jednim gumbom za spremanje promjena. Ova funkcionalnost je implementirana zbog brzine izvršavanja, kao i zbog minimiziranja mogućih bugova (programskih grešaka) pri spremanju promjena, jer ako imamo manje podataka koje se mjenjaju, lakše možemo uočiti gdje se pogreška stvorila. Pri prelasku u druge grupe promjene se brišu, te se jedino spremaju u bazu podataka pritiskom na gumb „Save Changes“. Ostali gumbovi, poput „Pošalji odgovore“ te ikone za T, F i N vrijednosti, nemaju funkcionalnost, nego su ostavljeni radi lakšeg snalaženja u ispitu.

Problem pri stvaranju "klona" druge komponente bio je u potrebi za prepisivanjem koda. Iako klonirana komponenta na prvi pogled izgleda slično, njene funkcionalnosti razlikuju se u dovoljnoj mjeri da nije bilo moguće jednostavno ponovno koristiti već implementiranu komponentu. Zbog tih razlika, bilo je potrebno razviti personaliziranu verziju te komponente koja zadržava većinu funkcionalnosti originala, uz dodatne mogućnosti specifične za "Edit" mod rada.



Slika 39: *Prikaz pregleda ispita u Preview Mode-u (izvor: Autor)*



Slika 40: *Prikaz ispita u Edit Mode-u. (izvor: Autor)*

3.3.6 Pinia i problem kod mjenjanja grupa

Kretanje po ispitu na početku razvoja bilo je jednostavno; studenti su imali mogućnost prelaska na sljedeće pitanje bez mogućnosti povratka na prethodno. Program bi nakon svakog rješenog zadatka u ispitu izračunao do sada stečene bodove od ukupnih, te bi na kraju prikazao i spremio rezultate. Međutim, nakon prijedloga studenata, odlučeno je da će ispiti također omogućiti povratak na prethodna pitanja. Nakon implementacije ove funkcionalnosti, ustanovilo se da trenutna implementacija dohvaćanja ispita, računanja bodova i kretanja po zadacima nije bila dovoljno fleksibilna za uvođenje navedene mogućnosti.

Zbog toga je uveden Pinia store, s kojim se prate globalne varijable, uključujući zadatke, trenutne odgovore, te omogućuje client-side implementaciju nasumičnog odabira zadataka.

Kod na slici 41 predstavlja dio implementacije defineStore metode koji Pinia nudi. Ova implementacija omogućuje upravljanje stanjem aplikacije na centraliziran način, što olakšava praćenje i manipulaciju podacima kroz različite dijelove aplikacije.

```
xport const useTestStore = defineStore('test', {
  state: () => ({
    tasks: [],
    currentTaskIndex: 0,
    userAnswersPerTask: [],
    shuffledQuestions: [],
    isSelfAssessment: false,
    originalQuestionOrder: [], // Store the original order or identifiers
    ogShuffledTasks: [],

  }),
  getters: {
    // Ensure getters account for the remapping where necessary
    currentTask: (state) => state.tasks[state.currentTaskIndex] || {},
    currentAnswers: (state) =>
state.userAnswersPerTask[state.currentTaskIndex] || [],
    currentShuffledQuestions: (state) =>
state.shuffledQuestions[state.currentTaskIndex] || [],
    questionsTotal: (state) => state.tasks.reduce((total, task) => total +
task.questions.length, 0),
 // Other part of the logic pertaining to score calculations
```

Slika 41: *Kod za korištenje Pinia Storage-a (izvor: Autor)*

U ovoj implementaciji, defineStore se koristi za definiranje stanja (state), gettera (getters) i akcija (actions) za testiranje.

Stanje pohranjuje zadatke (tasks), trenutni indeks zadatka (currentTaskIndex), korisničke odgovore po zadatku (userAnswersPerTask), nasumično odabrana pitanja (shuffledQuestions), te izvorni redoslijed pitanja (originalQuestionOrder). Ove varijable omogućuju praćenje napretka kroz ispit i prikupljanje podataka potrebnih za izračun bodova.

Getteri osiguravaju pristup trenutnim zadacima, odgovorima i nasumično odabranim pitanjima. Također omogućuju izračun ukupnog broja pitanja i ukupnog broja bodova na kraju ispita.

Iako akcije nisu definirane u prikazanom dijelu koda, one se obično koriste za manipulaciju stanjem, kao što je inicializiranje zadataka, mjenjanje trenutnog zadatka ili spremanje odgovora.

3.3.7 Poslužiteljski sloj u oblaku Firestore

U teoriji i prema najboljim praksama, svaka interakcija s bazom podataka trebala bi se izvršavati putem API poziva kako bi se osigurala sigurnost, konzistentnost i održivost sustava. Međutim, u praksi smo naišli na česta zagušenja (bottleneckovi) zbog ograničenih performansi našeg servera. Na primjer, kada bi studenti trebali započeti ispit, zbog velikog broja API poziva, dolazilo bi do čekanja na izvršenje drugih radnji prije njihovih, što bi moglo značajno produžiti vrijeme čekanja. To je posebno problematično jer svaki student svakih pet sekundi šalje svoje podatke u bazu podataka.

Kako bi poboljšali performanse sustava, odlučeno je da će se u poslužiteljski sloj slati samo ključne metode, poput metoda za AI algoritme te podatke vezane za tijek ispita, primjerice varijable koje prate gubitak fokusa na ispit (outOfFocus) ili proteklo vrijeme ispita. Ove informacije ispitivač može vidjeti u stvarnom vremenu putem administrativnog sučelja. Ostale operacije, kao što su dohvaćanje i predaja ispita, izvršavaju se direktno na frontendu i šalju se izravno u bazu podataka pomoću Firebase Realtime Database API mogućnosti.

Ova metoda nije preferirana zbog niže razine sigurnosti u odnosu na backend rješenja, ali je implementirana kako bi se postigla brža operativnost sustava. Time se omogućuje balansiranje između sigurnosti i performansi sustava.

Primjer koda koji prikazuje kako se podaci o testu šalju u bazu podataka putem API poziva prikazan je na slici 42 , te ilustrira proces slanja kombinirane snimke zaslona i kamere putem API poziva.

1. „captureFromVideoStream“ funkcija se koristi za snimanje trenutne slike s video feeda zaslona i kamere.
2. Stvara se HTML canvas element koji služi za manipulaciju slikom.
3. Nakon što su snimke zaslona i kamere prikupljene, pohranjuju se u obliku blob objekata.
4. Ti objekti se zatim dodaju u FormData zajedno s dodatnim informacijama kao što su email, kod testa, broj gubitaka fokusa (focusLossCount) i stanje završetka testa (isCompleted).
5. Također, izračunava se proteklo vrijeme testa i dodaje u FormData.
6. Konačno, FormData se šalje serveru putem fetch API-ja koristeći HTTP POST metodu, gdje se podaci dalje obrađuju i pohranjuju u Firebase DB, vidljiv na slici 43.
7.

```
const sendCombinedScreenshot = async () => {
          const screenBlob = await captureFromVideoStream(screenVideo);
          const cameraBlob = await captureFromVideoStream(cameraVideo);
          const canvas = document.createElement('canvas');
          const context = canvas.getContext('2d');
            canvas.toBlob(async (blob) => {
                const formData = new FormData();
                formData.append('screenshot', screenBlob, `screen-screenshot-
                   ${Date.now()}.jpg`);
                formData.append('camerashot', cameraBlob, `camera-screenshot-
                   ${Date.now()}.jpg`);
                formData.append('firstName', firstName);
                formData.append('lastName', lastName);
                formData.append('test_code', test_code);
                formData.append('focusLossCount',
                focusLossCount.value.toString());
                formData.append('completionState',
                   isCompleted.value.toString());
                const elapsedTime = (testDuration.value * 60) -
                   remainingTime.value;
                formData.append('elapsedTime', elapsedTime.toString()); // Add
                await fetch(`${API_BASE_URL}/upload`, {
                    method: 'POST',
                    body: formData,
                }).then(response => response.json())
                  .catch(error => console.error('Error:', error));
            }, 'image/jpeg');
```

Slika 42: *Metoda za slanje snimke zaslona i kamere u server (izvor: Autor)*

Slika 43: *Prikaz spremljenih informacija o studentu koji je polagao ispit (izvor: Autor)*

Javascript kod na slici 44 izvršava radnju predaje ispita te spremanje istog u bazu podataka direktno iz frontenda, pomoću Firebase Realtime Database API.

Prvo zaustavljamo mjerač vremena, jer se u testiranju dogodilo da bi vrijeme isteklo dok su studenti ćekali da im se preda ispit, tako da zovemo tu funkciju prije nego počnemo sa predajom. Na zaslonu se prikaže ikonica za učitavanje, te označavamo da je ispit predan, kako bi spriječili dupla izvršavanja funkcije zbog višestrukog klikanja na gumb.

Nakon kalkulacije bodova ispitnih pitanja obrađujemo svaki zadatak te ga pohranjujemo u bazu podataka kako bi kasnije mogli pregledati sve odgovore u administrativnom sučelju, kao i student na sučelju za rezultate. Nakon što je svaki odgovor spremljen, šaljemo i rezultate samog ispita u istu bazu podataka, gdje su rezultati vidljivi na slici x.

```
const handleTestSubmitDirect = async () => {
      stopTimer();
      showConfirmationModal.value = false;
      isLoading.value = true;
      isTestSubmitted.value = true;
      calculateScoresForCurrentTask(currentTaskIndex.value);
      aggregateScores();
      tasks.value.forEach((task, index) => {
        const taskSubmission = {
          task_id: `task_${index + 1}`, // Adjust if necessary
          userAnswers: store.userAnswersPerTask[index].map((answer) =>
            answer === null ? "null" : answer.toString()
          ),
          taskPoints: task.questions.length,
          userPoints: userScorePerTask.value[index] || 0,
        };
        const encodedEmail = email.value.replace(/\./g, ",");
        // Firebase path
        const path =
      `/studentTests/${encodedEmail}/${testCode.value}/tasks/${taskSubmission.
       task_id}`;
        const taskRef = dbRef(database, path);
        // Update the task in Firebase
        set(taskRef, {
          userAnswers: taskSubmission.userAnswers,
          taskPoints: taskSubmission.taskPoints,
          userPoints: taskSubmission.userPoints,
        })
          .then(() => {
            console.log("Task updated successfully");
          })
          .catch((error) => {
            console.error("Error updating task:", error);
          });
      });
      const encodedEmail2 = email.value.replace(/\./g, ",");
      const path2 = `/studentTests/${encodedEmail2}/${testCode.value}`;
      const taskRef2 = dbRef(database, path2);
      update(taskRef2, {
        correctAnswers: correctAnswers.value,
        incorrectAnswers: incorrectAnswers.value,
        nullAnswers: nullAnswers.value,
      })
```

Slika 44: *Metoda za direktno slanje snimke zaslona i kamere u bazu podataka (izvor: Autor)*

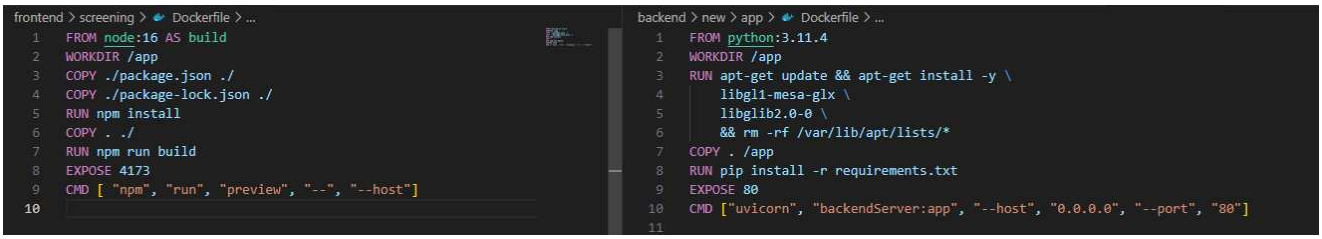
3.3.8 Dockerizacija i hosting aplikacije

Dockerizacija je proces pakiranja aplikacije i svih njenih ovisnosti u kontejner koristeći Docker. Kontejner je lagano, izolirano i prijenosno okruženje koje sadrži sve što je potrebno za pokretanje aplikacije, uključujući kod, runtime, sustavske alate, libraries i postavke konfiguracije. Docker, kao platforma za kontejnerizaciju, omogućuje razvojnim timovima da svoje aplikacije pokreću dosljedno na različitim okruženjima, od razvojnih strojeva do produkcijskih servera.

Svaku aplikaciju koju želimo dockerizirati moramo stvoriti novi „dockerfile“, koji je skripta koja definira korake potrebne za izgradnju docker slike, poput instalacije ovisnosti, kopiranja aplikacijskog koda i definiranja komandi za pokretanje aplikacije. Korisničko sučelje i poslužiteljski sloj podijelili smo u dva docker kontejnera,vidljivo na slici 45, koje smo objavili na docker hub, stranicu za kontrolu verzija docker kontejnera sa kojeg možemo skinuti stabilne dockerizirane aplikacije radi daljnje uporabe i razvoja.

Dockerizacija se koristi kako bi se riješili problemi dosljednosti i prijenosivosti aplikacija između različitih okruženja. Bez obzira na to gdje se aplikacija pokreće—na lokalnom razvojnom računalu, testnom serveru ili u produkcijskom okruženju—Docker kontejneri osiguravaju da će aplikacija uvijek raditi na isti način. Ovo je posebno važno za timove koji rade na složenim aplikacijama s mnogim ovisnostima, jer Docker omogućuje brzo postavljanje konzistentnog okruženja za razvoj, testiranje i produkciju.

Dockerizacija naše aplikacije omogućila nam je da postignemo dosljednost između različitih okruženja u kojima aplikacija radi. S obzirom na složenost naše aplikacije i broj ovisnosti, bilo je ključno osigurati da sve komponente funkcioniraju na isti način na svakom stroju, bez obzira na specifične konfiguracije sustava.



Slika 45: *DockerFile za frontend i backend aplikacije (izvor: Autor)*

3.4 Opis UX dizajna

UX (User Experience) dizajn odnosi se na proces stvaranja proizvoda koji korisnicima pružaju relevantno i smisleno iskustva. U kontekstu izrade moderne web stranice, UX dizajn igra ključnu ulogu u osiguravanju da krajnji korisnici imaju pristupačan, intuitivan i efikasan pristup funkcionalnostima koje im nudi aplikacija.

3.4.1 Korisničko iskustvo

Nastojali smo korisnicima pružiti što jednostavnije i pristupačano iskustvo, prateći linearnu progresiju događaja; Na početnom zaslonu ne nalazi se ništa što korisniku nije potrebno, a kako bi naknadno pomogli studentu također smo dodali da se pomoću URL-a može izvući kod ispita, tako da to ne moraju upisivati. Studentov JMBAG se također sprema u databazu kako ne bi ga morao ponovno upisivati kada polaže novi ispit, tako da u pojedinim slučajevima niti ne moraju unositi podatke kako bi pristupili ispitu. Na slici 46 vidimo primjer URL sa query-em za unos naziva ispita, što znatno olakšava dijeljenje istog sa studentima i smanjuje mogućnost pogreške unosa koda.

Slika 46: *URL sa query-em za dijeljenje naziva ispita (izvor: Autor)*

Na samom ispitu važno je bilo da je kod ispita čitljiv, označen pravilnim sintaktičkim oznakama, te dostupan u svojoj potpunosti studentu. Iznad svakog koda zadatka vidi se ukupni broj bodova koji student može ostvariti na trenutačnom zadatku.

Sami tekst zadataka koristi monospace obitelj fontova, radi najbolje čitljivosti, te podržava podebljavanje, podcrtavanje te bilo koje druge efekte koji su dostupni u Markdown datotekama.

Oznake True/Null/False označeni su jedinstvenim bojama, vidljivo na slici 47, kako bi korisnik dobio vizualni odziv na odabire, te lakše raspoznao iste. Mjenjanje odabira popraćeno je animacijom kako bi također se naznačila promjena.

Slika 47: *Prikaz boja za TFN zadatke (izvor: Autor)*

Oznake za sljedeći i prijašnji zadatak, vidljivi na slici 48, drugačijih su boja kako bi ih vizualno odvojili po funkciji, gjde je „sljedeći zadatak“ označen žarkom bojom, dok je „predhodni zadatak“ svijetliji i manje upada u oko. Gumb za predaju ispita je označen crvenom bojom kako bi obavjestili studenta o njezinog važnijoj funkciji.

Slika 48: *Razlika u bojama za navigacijske gumbe (izvor: Autor)*

Komponenta koja se nalazi iznad samih pitanja služi kako bi korisnik znao na kojem se zadatku nalazi, koliko zadataka još nije posjetio, te koliko zadataka se ukupno nalazi na ispitu. Također pruža mogućnost navigacije zadatcima klikom na bilo koji zadatak, te spriječava navigaciju na nepoznate zadatke, kako bi se držali principa slijednosti.

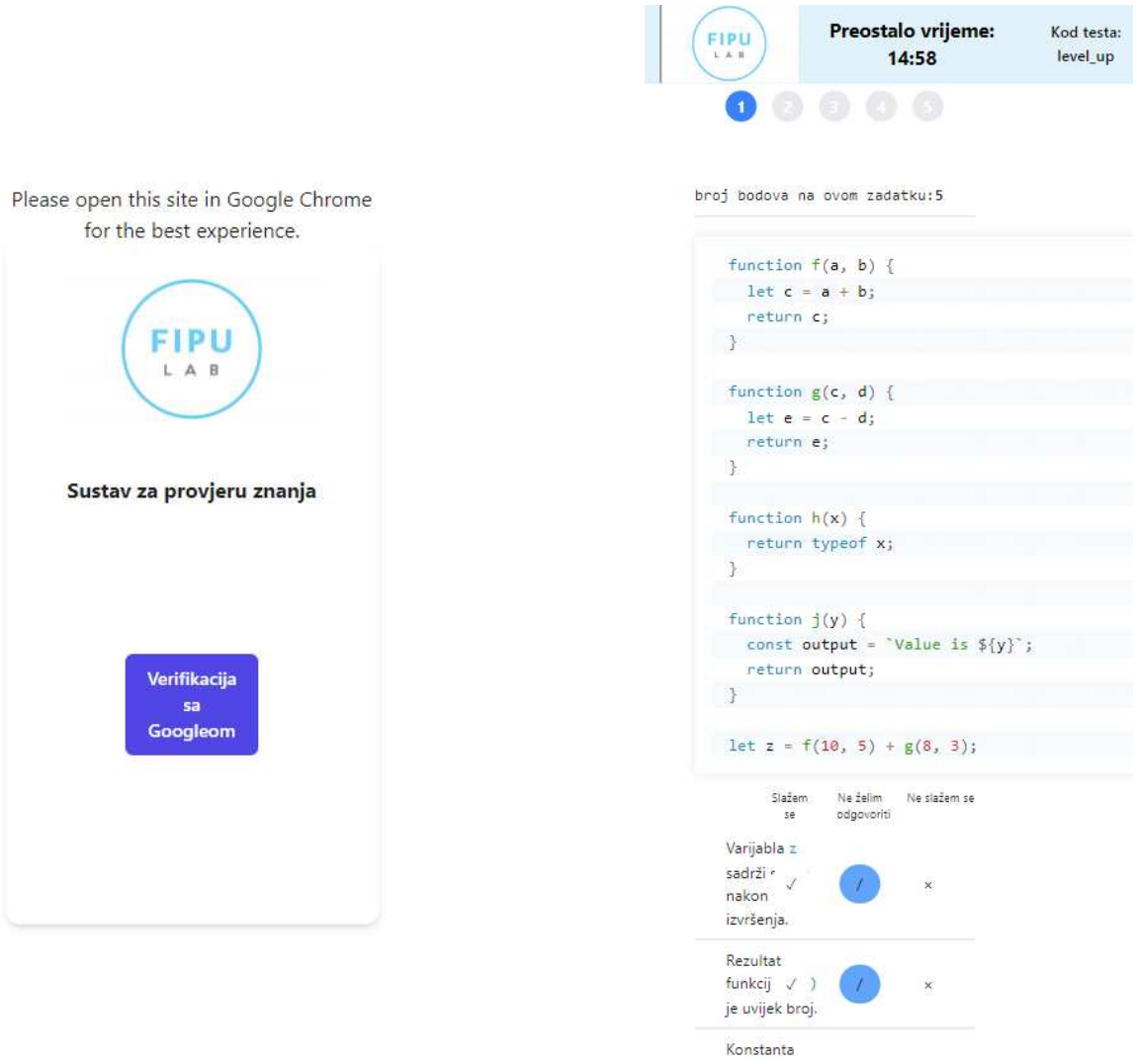
Preostalo vrijeme nalazi se na gornjem lijevom kutu navigacijske trake jer je najviše uočljivo, budući da korisnik najviše vremena provede čitajući javascript dio zadatka koji se također nalazi na lijevom dijelu stranice. Na desnoj strani stranice nalazi se identifikacijski kod ispita, on služi kako bi u slučaju pregleda snimke zaslona mogli provjeriti o kojem ispitu se radi, u slučaju nekakve greške u bazi podataka, vidljivo na slici 49.

Slika 49: *Prikaz navigacijske trake sa preostalim vremenom i kodom ispita (izvor: Autor)*

Kada korisnik preda ispit dobiva čitljive i detaljne podatke o rezultatu, broju točnih, netočnih i neodgovorenih pitanja, koliko bodova svaki od njih nosi, te ukupan broj bodova ispod tih podataka. Ako je ispitivač također označio da se korisniku prikažu detaljni podatci o rješenom ispitu, pojavit će se također javascript kod svakog zadatka, odgovora i pojašnjenje za svako pitanje. Nakon pregleda, korisnika vraćamo nazad na početnički zaslon.

3.4.2 Responzivnost

Aplikacija je optimizirana za rješavanje ispita na osobnom računalu, budući da trenutačno nije implementirana podrška za snimanje putem kamera na mobilnim uređajima. Zbog toga aplikacija nema uniformni dizajn prilagođen za mobilne uređaje, što može otežati preglednost i čitljivost zadataka na manjim zaslonima. Na slici 50 prikazano je početno sučelje aplikacije, kao i sučelje testa na uređaju iPhone Pro 12. Na početnom zaslonu također se prikazuje upozorenje koje savjetuje korisnicima da aplikaciju otvore u Google Chrome pregledniku kako bi osigurali najbolje korisničko iskustvo.



Slika 50: *Prikaz početnog sučelja i ispitnog sučelja na uređaju Iphone Pro 12 (izvor: Autor)*

3.4.3 Testiranje upotrebljivosti i Iterativni dizajn

Aplikacija je testirana kroz različite faze razvoja uz sudjelovanje studenata informatike, koji su sudjelovali u kratkim ispitima organiziranim kao dio testiranja. Osim toga, određeni studenti imali su priliku sudjelovati u dodatnim testiranjima kako bi stekli dodatne bodove na kolegiju. Testiranja su prvenstveno bila usmjerena na provjeru funkcionalnosti aplikacije, kao i na provođenje tzv. „stress testova“ radi procjene responzivnosti aplikacije u uvjetima velikog broja korisnika koji istovremeno koriste sustav.

Nakon svake iteracije testiranja, prikupljene su povratne informacije, uključujući kritike i prijedloge za daljnje poboljšanje aplikacije. Ove povratne informacije bile su predmet detaljnih diskusija s mentorom kako bi se procijenila njihova valjanost i mogućnost implementacije. U

slučajevima kada su prijedlozi bili prihvaćeni, slijedila je faza ispravljanja uočenih problema i implementacija novih funkcionalnosti.Nakon svake iteracije testiranja, prikupljene su razne kritike i ideje za daljnjim poboljšanjem aplikacije, prodiskutirane sa mentorom oko valjanosti i mogućnosti implementacije, te ako je prihvaćena, sljedila bi faza popravka uočenih problema, te implementacija novih komponenti.

3.5 Git i Github, Verzioniranje koda

Verzioniranje koda je ključni alat u modernom razvoju softvera, omogućujući timovima da učinkovito surađuju, prate promjene i održavaju povijest razvoja projekta. Verzioniranje projekta omogućuje razvojnom timu da se vrati na prethodne verzije, prati tko je i kada napravio promjene, te integrira nove funkcionalnosti bez konflikata.

Git je sustav za kontrolu verzija koji je razvijen 2005. godine od strane Linusa Torvaldsa, kreatora Linux operativnog sustava. Git je razvijen kao odgovor na potrebu za moćnim, brzom i distribuiranom verzijom kontrole nakon što je komercijalni alat BitKeeper, koji je prethodno korišten za razvoj Linux kernela, prestao biti dostupan besplatno.

Jedna od glavnih prednosti Gita u odnosu na prethodne sustave verzioniranja bila je njegova distribuirana priroda. Umjesto da postoji centralizirani server koji sadrži cijelu povijest koda, svaki Git repozitorij sadrži kompletnu povijest svih promjena. Ovo omogućuje razvojnim timovima da rade na svojim lokalnim kopijama repozitorija neovisno o mrežnim povezivanjima, a zatim sinkroniziraju svoje promjene s glavnim repozitorijem kada su spremni.

Preporučeni razvoj projekta pomoću Githuba uključuje stvaranje „branch-a“ (grana), na kojoj implementiramo novu funkcionalnost, potvrđivanje promjena u obliku „commit-a“, te pomoću „pull request-a“ dajemo mogućnost spajanje funkcionalnosti (merge) u glavni branch. Visual studio code sadrži ektenziju sa kojom možemo pratiti promjene u trenutačnom codebase-u, daje mogućnost commitanja promjena, te integracije promjena bez da moramo otvoriti sam Github.

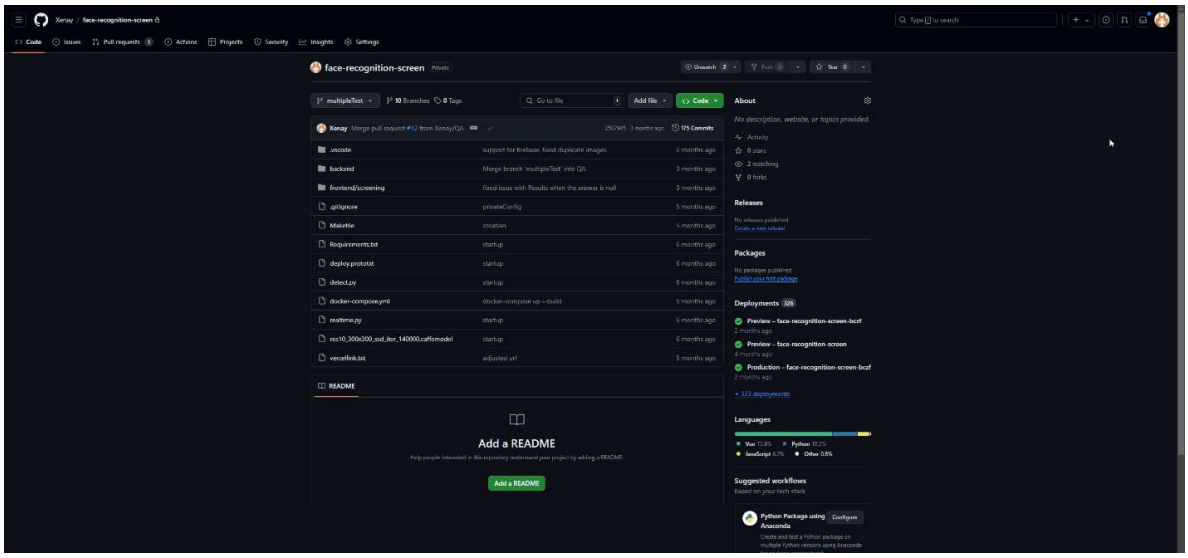
Merge conflicts (konflikti prilikom spajanja) su situacije koje nastaju kada Git ne može automatski spojiti promjene iz dvije različite grane (branches). Do merge conflicta dolazi kada dvije ili više osoba istovremeno rade na istom dijelu koda, ali naprave različite izmjene na

istom redku ili dijelu datoteke. Kada se pokuša spojiti te promjene u jednu granu, Git ne može automatski odlučiti koja verzija koda treba biti zadržana, pa traži od developera da ručno riješi konflikt.

Budući da je jedna osoba radila na projektu, nije bilo potrebno raditi nove „brancheve“ za svaku novu implementaciju, iako spada u „best practice“, no svaka verzija programa (QA, Dev i Production) ima svoji branch na kojem se prate promjene.

Na slici 51 vidimo početnu stranicu projekta na platformi Github, na kojoj možemo viditi sve detalje o trenutnom branchu projekta na kojem se nalazimo. U dosta slučajeva „frontend“ i „backend“ projekta znaju se rašćlaniti u dva git repozitorija, no budući da Render i Vercel mogu birati koji folder u repozitoriju za koristiti, odlućeno je sve staviti u isti.

Također je vidljiva stavka „Deployments“, koja prati svaki deployment koje su napravljene pomoću Vercela ili Rendera.



Slika 51: *Github sučelje za projekt (izvor: Autor)*

3.6 Uvođenje novih funkcionalnosti radi lakše navigacije ispita

3.6.1 Dodatak animacija

Tijekom rješavanja ispita, primijećeno je da su neki studenti imali poteškoća s navigacijom između zadataka. Zbog sličnosti strukture svih zadataka, često se događalo da studenti nenamjerno preskoče zadatak, misleći da je prethodni zadatak još uvijek u procesu prijelaza na sljedeći. Ovaj problem bio je posebno izražen tijekom ranih faza testiranja, kada optimizacija logike rješavanja ispita putem migracije na Firebase API još nije bila dovršena.

Kao rješenje, uvedene su animacije prijelaza između zadataka koristeći CSS stilove, vidljive na slici 52. Ove animacije olakšavaju prepoznavanje prijelaza između zadataka, smanjujući mogućnost pogrešaka. Zahvaljujući Vue.js strukturi podataka, nije bilo potrebe za izradom posebne .css datoteke za prijenos stilova; umjesto toga, stilovi su dodani izravno u označene komponente koristeći oznaku <style>. Ovaj pristup smanjuje razinu apstrakcije i povećava količinu koda unutar jedne datoteke, ali istovremeno smanjuje ukupan broj datoteka, čime se pojednostavljuje upravljanje kodom.

```
.bg-green-200:hover, .bg-red-200:hover {
  opacity: 0.9;
}
.bg-green-200 {
  background-color: #c6f6d5;
}
.bg-red-200 {
  background-color: #fed7d7;
}
.fixed-width-font {
  font-family: 'SFMono-Regular', Consolas, 'Liberation Mono', Menlo, Courier,
monospace;
}
.slide-enter-active, .slide-leave-active {
  transition: all 0.5s ease;
}
.slide-enter, .slide-leave-to {
  transform: translateX(100%);
  opacity: 0;
}
.slide-enter-to, .slide-leave {
  transform: translateX(0);
  opacity: 1;
}

.v-enter-active, .v-leave-active {
  transition: all 0.5s ease;
}
.v-enter, .v-leave-to /* for Vue versions <2.1.8, use .v-leave-active */ {
  opacity: 0;
  transform: translateY(30px);
}
.v-leave-from { opacity: 1; }

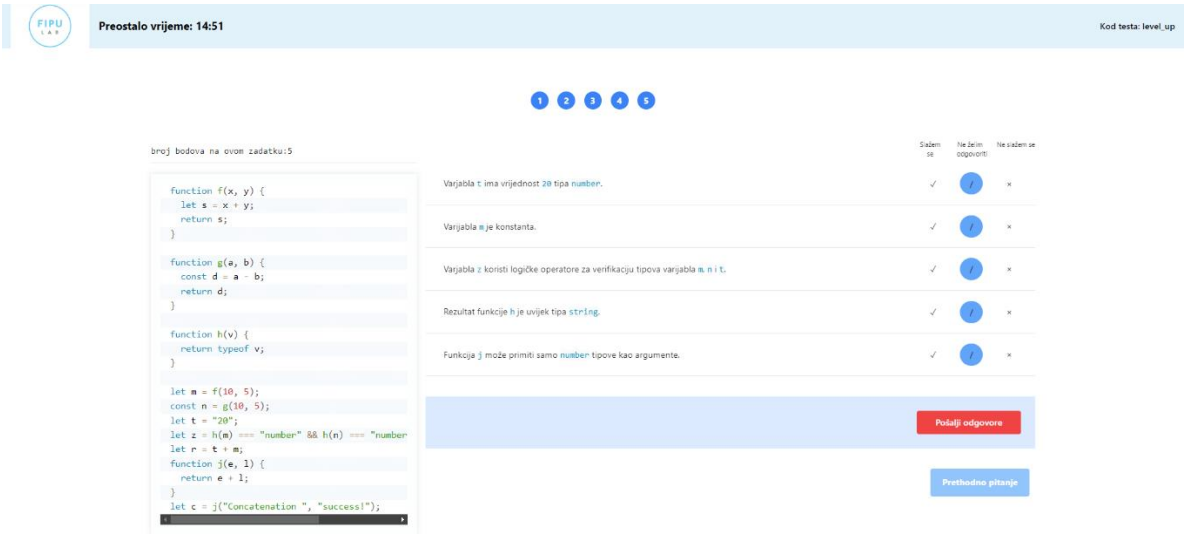
</style>
```

Slika 52: *css file za animaciju testa (izvor: Autor)*

3.6.2 Custom Modal

Prilikom navigacije kroz zadatke ispita, studenti nisu imali jasan pokazatelj da su došli do posljednjeg zadatka. Kako bi se olakšala navigacija i osiguralo bolje korisničko iskustvo, uvedeno je nekoliko ključnih indikatora:

1. Drugaćija boja gumba za predaju ispita, vidljiva na slici 53, crvene je boje te sadrži uočljivi razmak između sebe i drugih gumbova.



Slika 53: *Prikaz gumba za predaju ispita (izvor: Autor)*

2. Dialog za potvrđivanje odabira, vidljiv na slici 54, primarno služi kako bi pokazala studentu koliko zadataka još nije odgovoreno, u slučaju da su zaboravili odgovoriti, ili ne žele odgovoriti na njih radi negativnih bodova.

Slika 54: *Prikaz komponente za potvrdu predaje ispita (izvor: Autor)*

3. Komponenta koja prati napredak studenta kroz ispit, prikazujući trenutni zadatak na kojem se nalazi, vidljiva na slici 55. Klikom na bilo koji od prikazanih brojeva, student može brzo navigirati na odabrani zadatak. Dodatno, postoji varijabla maxAccessibleTask, koja sprječava korisnika da se kreće do zadataka koje još nije posjetio, čime se osigurava linearno iskustvo i izbjegavaju potencijalne pogreške prouzročene nepravilnim "skakanjem" između zadataka. Javascript kod ove komponente vidljiva je na slici 56.

Slika 55: *Komponenta za prikaz tr0enutnog indeksa zadatka na kojem se student nalazi (izvor: Autor)*

```
<template>
  <div class="flex items-center justify-center py-12">
    <div class="flex items-center m-2" v-for="index in totalTasks"
:key="index">
      <div
        :class="{
            'bg-blue-500': index <= currentTask,
            'bg-gray-300': index > currentTask,
            'cursor-pointer': index <= maxAccessibleTask,
            'cursor-not-allowed opacity-50': index > maxAccessibleTask
        }"
        class="h-8 w-8 rounded-full flex items-center justify-center text-
white font-bold"
        @click="goToTask(index)"
      >
        {{ index }}
      </div>
      <div v-if="index < totalTasks" class="flex-auto border-t-2 transition
duration-500 ease-in-out" :class="{'border-green-500': index < currentTask,
'border-gray-300': index >= currentTask}"></div>
    </div>
  </div>
</template>
```

Slika 56: *Javascript kod komponente za praćenje napretka na ispitu (izvor: Autor)*

3.6.3 Odabir kamere i poteškoće sa ulazom u test

Jedan od rijeđih problema sa kojim smo se susretali je da bi student/ica sa svojim osobnim računalom probalo/la pristupiti ispitu, ali program ne bi prepoznao kameru, stoga smo

prvobitno zabranili pristup korisnicima koji nemaju definiranu kameru kako ne bi dobivali nevažeće ispite, budući da ne znamo da li su varali na kameri ili ne. Problem bi postao prednost te bi drugi studenti probali replicirati bug, stoga smo savjetovali korisniku da ponovno provjeri kamera uređaj, te bi ga vratili na početni zaslon.

Nadalje, poboljšali smo logiku prepoznavanje kamere kako bi korisnici sa slabijim internetom mogli dobiti pregled. Na slikama 57 i 58 prikazujemo stari i novi način dohvaćanja kamere.

U starom načinu rada, funkcija getUserMedia koristi se za pristup kameri uređaja s unaprijed definiranim specifikacijama za širinu (1280px) i visinu (720px) video feeda. Nakon što se stream dobije, dodjeljuje se video elementu kako bi prikazao feed uživo. Iako je ovaj način rada jednostavan, ograničen je u smislu fleksibilnosti i mogućnosti odabira specifične kamere, posebno u situacijama kada postoji više priključenih kamera (npr. laptop i vanjska web kamera).

Novi način rada uvodi značajna poboljšanja u fleksibilnosti i pouzdanosti pristupa kameri. Funkcija sada prima camera Id kao parametar, što omogućuje odabir specifične kamere na uređaju. Ako je cameraId dostupan, koristi se za pristup upravo toj kameri, što omogućuje korisnicima da biraju između različitih priključenih kamera. Ako cameraId nije dostupan, kamera se automatski bira (obično glavna kamera). Nakon što se stream dobije, koristi se nextTick kako bi se osiguralo da je video element dostupan i spreman za dodjelu streama.

```
async startCameraold() {
      try {
        const stream = await navigator.mediaDevices.getUserMedia({
          video: { width: 1280, height: 720 }
        });

        // Assign the stream to the video element to display the feed
        this.$refs.videoPreview.srcObject = stream;
      } catch (error) {
        console.error("Error starting camera:", error);
      }
    },
```

Slika 57: *Stara metoda za detekciju kamere (izvor: Autor)*

```
const startCamera = async (cameraId) => {
    console.log("Starting camera with ID:", cameraId);
    try {
      const constraints = cameraId ? { video: { deviceId: { exact: cameraId } 
} } : { video: true };
        const stream = await navigator.mediaDevices.getUserMedia(constraints);
        console.log("Stream obtained:", stream);
        nextTick(() => {
            if (videoPreview.value) {
                videoPreview.value.srcObject = stream;
                console.log("Stream set to video element");
                videoPreview.value.play().catch(err => {
                    console.error("Error auto-playing the video:", err);
                });
            } else {
                console.error("Video element is not yet available after next
                 tick");
            }
        });
        setTimeout(() => checkCameraBrightness(), 1000);
        if (brightnessCheckInterval) clearInterval(brightnessCheckInterval);
        brightnessCheckInterval = setInterval(() => {
          checkCameraBrightness();
        }, 5000);
      } catch (error) {
        console.error("Error starting camera:", error);
      }
    };
```

Slika 58: *Nova metoda za detekciju i stanje kamere (izvor: Autor)*

Ova nadogradnja značajno poboljšava iskustvo rada s kamerama u aplikaciji, čineći je prilagodljivijom i otpornijom na različite scenarije korištenja.

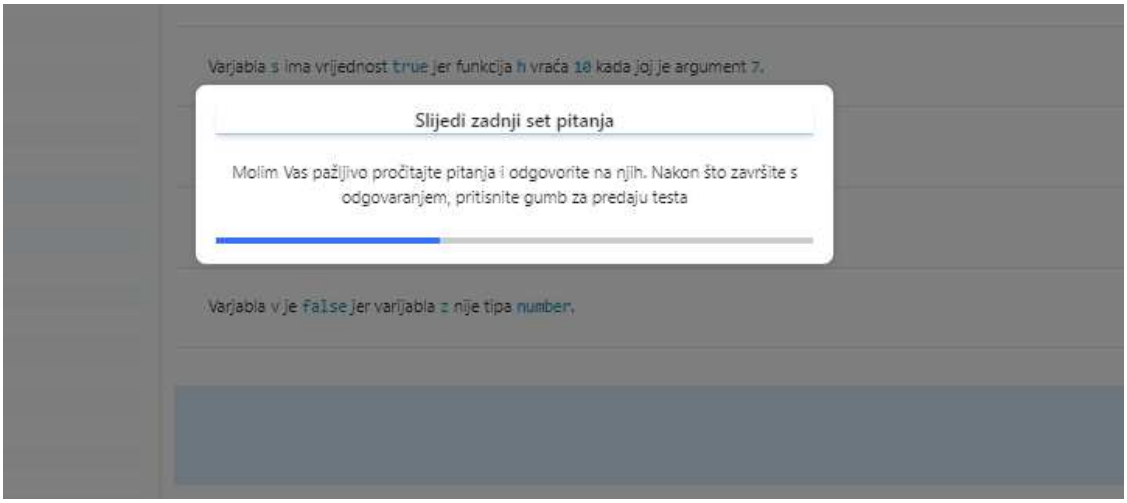
3.7 Neupotrebljene komponente i funkcionalnosti

3.7.1 Prilagodljiv prozor za notifikacije

Kako bi studentima pružili dodatno osiguranje kako se nalaze na zadnjem zadatku, implementirana je generička komponenta, vidljiva na slici 59, koja prikazuje naslov, tekst, te traku za napredak, koja vizualno odbrojava vrijeme u kojem će se komponenta prikazati, kada je traka prazna, komponenta se zatvara te student može preći na ponovno odgovaranje ispita.

Komponenta je napravljena sa modularnosti u vidu, kao što je vidljivo na slici 60, gjde zadajemo tekst naslova, tijela te trajanje u milisekundama.

Odlućeno je ne koristiti ovu komponentu u trenutnoj verziji aplikacije, no moguće je da bude korištena u novim verzijama implementacije ispita.



Slika 59: *Prikaz generične pop-up komponente (izvor: Autor)*

```
if (currentTaskIndex.value === tasks.value.length - 1) {
// Notify the user that they are about to answer the last set of questions
        showGenericPopup(
          "Slijedi zadnji set pitanja",
          "Molim Vas pažljivo pročitajte pitanja i odgovorite na njih. Nakon
što završite s odgovaranjem, pritisnite gumb za predaju testa",
          5000
        );
}
```

Slika 60: *Primjer korištenje komponente sa dodjeljenim varijablama (izvor: Autor)*

3.7.2 Google Gemini API za automatizaciju stvaranje ispita

Kako bismo postigli potpunu automatizaciju procesa izrade ispita, razmatrali smo upotrebu umjetne inteligencije za generiranje ispita putem prompt engineeringa. Ideja je bila da, koristeći AI modele, generiramo pitanja na temelju specificiranih promptova koji opisuju vrstu ispita, vrstu zadataka, broj zadataka i druge konfiguracije. Dvije najistaknutije opcije za ovu vrstu zadatka bile su Google Gemini i OpenAI.

Google Gemini je relativno nov model umjetne inteligencije razvijen od strane Googlea, dizajniran za generativne zadatke kao što su generiranje sadržaja, odgovaranje na složene upite, pa čak i stvaranje složenih struktura poput testova i zadataka. Google Gemini koristi najnovije tehnike u dubokom učenju i prirodnom jeziku, omogućujući vrlo precizno razumijevanje i generiranje teksta. Gemini je posebno koristan u situacijama gdje je potrebno zadržati visoku razinu kontekstualne točnosti i strukturiranog outputa, kao što je to slučaj kod generiranja ispita.

OpenAI je pionir u području generativne umjetne inteligencije, poznat po modelima kao što su GPT-3 i GPT-4. OpenAI modeli su iznimno sposobni za širok raspon zadataka, od generiranja prirodnog jezika do stvaranja kreativnog sadržaja, uključujući automatizirane zadatke poput stvaranja testova. Njihova prednost leži u sposobnosti prilagodbe različitim stilovima i zadacima, čime se osigurava fleksibilnost u razvoju i korištenju.

S obzirom na to da već koristimo Googleove proizvode u našem tehničkom stacku (kao što su Firebase Realtime DB i Firebase Auth), odlučili smo se proširiti tu integraciju korištenjem Google Gemini API-ja za automatizirano generiranje ispitnih zadataka. Na slici 61 prikazana je jednostavna implementacija Gemini API-ja za generiranje zadataka unutar ispita. Važno je napomenuti da je ovaj dio projekta još uvijek u eksperimentalnoj fazi, stoga implementacija nije uključena u frontend dio aplikacije, a čak i da jest, ne bi promijenila osnovnu funkcionalnost koda.

Ključan aspekt u generiranju zadataka koji su identične strukture kao prethodni zadaci jest pružanje konteksta API-ju. To uključuje specifikacije o prijašnjim generiranim zadacima, postavljanje zahtjeva za održavanje identične strukture, te definiranje drugih opcija poput tematike ispita, broja pitanja, kompleksnosti koda i slično.

Za ovaj proces koristimo model „gemini-1.5-flash“, koji je trenutno jedan od najpopularnijih modela u razvoju. Metodom „generate_content“ pružamo kontekst API-ju, zajedno s primjerom zadatka i brojem željenih zadataka. Rezultati generiranja se ispisuju u konzolu, a testiranjem smo utvrdili da, kada se generirani zadatak spremi u Markdown datoteku i unese u Test Maker, dobivamo ispravan ispit, premda formatiranje koda zadatka ponekad može biti neprecizno.

Glavni razlog zašto se ova metoda trenutačno ne koristi u potpunosti jest postojanje već uspostavljenog vanjskog pipelinea za kreiranje ispitnih zadataka. Trenutno je jednostavnije otvoriti Markdown datoteku i izvršiti promjene u tekstualnom editoru poput Notepad ili Notepad++, nego koristiti Test Editor, koji u ovom trenutku nije dovoljno intuitivan za učinkovito korištenje.

```
def generate_js_questions(code, num_questions=1):
    """Generates markdown questions about JavaScript code.
    Args:
        code (str): The JavaScript code snippet.
        num_questions (int, optional): The number of questions to generate.
Defaults to 5.
    Returns:
        str: A stringified markdown with code block and questions.
        list: A list of generated questions.
    """
    model = genai.GenerativeModel('gemini-1.5-flash')
    # Clean and format the code
    code = re.sub(r"\s+", " ", code).strip()
    # Generate questions using the model
    prompt = f"Generate {num_questions} true/false tasks that MUST be in the
same format as the following code example, MUST be the same in terms of
formatting, even the ```javascript part MUST stay the same, but the task must
be different. The code part must be newline seperated\nThe example
task:\n`javascript\n{code}\n`"
    prompt2 = f"I need a task that mirrors the following example in structure
and formatting, but with a different set of variables and expressions. Please
make sure to maintain the same format, including newlines, spacing, and type
of explanations. Here is the example task: {code}. Use different variable
names and values. Ensure there are both let and const declarations. Include
different types of expressions (arithmetic, string concatenation, logical,
type checking, modulo operation). Provide 5 true/false statements with
explanations about the variables and expressions. Each statement should
specify whether it is [Točno] or [Netočno], followed by a brief explanation in
parentheses. Also you must make the code readable like in the example, not all
in one line"
    response = model.generate_content(prompt2)
    questions = response.text
    markdown_questions = f"`javascript\n{code}\n`\n\n"
    question_list = []
    for question in questions.split("\n"):
        if question.strip():
            markdown_questions += f"1. {question.strip()}\n"
            question_list.append(question.strip())

    return markdown_questions, question_list
```

```
# Example usage
js_code = """
```javascript
let a = 5;
const b = "Hello";
let c = true;
let d = a > 3;
let e = b + " World";
let f = typeof e;
let g = c && d;
let h = 8 % 3;
let i = "Number " + a;
```
1. Varijabla `a` je tipa `number`. [Točno] (Pojašnjenje: Varijabla `a` je inicijalizirana s vrijednošću 5, što je tipa `number`.)
1. Varijabla `d` je `false`. [Točno] (Pojašnjenje: Varijabla `d` je `false` jer 5 nije veće od 3.)
1. Izraz `8 % 3` rezultira vrijednošću 2. [Točno] (Pojašnjenje: Operator modulo `%` vraća ostatak dijeljenja 8 s 3, što je 2.)
1. Varijabla `f` sadrži vrijednost `string`. [Točno] (Pojašnjenje: Varijabla `f` dobiva vrijednost `typeof e`, što je `string` jer `e` sadrži spojeni string `Hello World`.)
1. Varijabla `i` jednaka je `Number 5`. [Točno] (Pojašnjenje: Varijabla `i` je dobivena spojem stringa `Number ` i vrijednosti varijable `a` (5), što rezultira stringom `Number 5`.)
"""

markdown, questions = generate_js_questions(js_code)
print(markdown)
print(questions)
generate_js_questions(js_code)
```

Slika 61: *Prikaz metode za generiranje zadataka pomoću Google Gemini API-ja (izvor: Autor)*

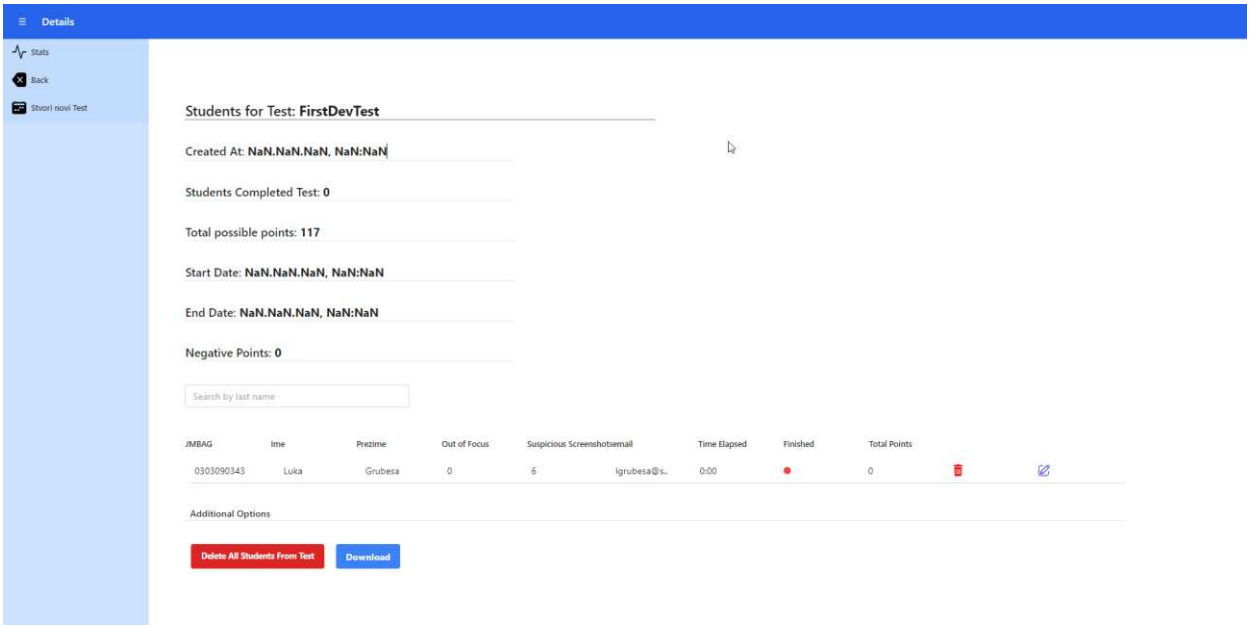
3.8 Navigacija programom

3.8.1 Bočna izbornička traka

Na stranicama za biranje, pregled i stvaranje ispita, trebalo je stvoriti komponentu koja će funkcionirati kao izbornik koji se otvara klikom na „hamburger menu“, te prikazati rute na koje se može prebaciti.

Kao što je vidljivo na slici 62, klikom na ikonicu pored „Details“ otvara animiranu izborničku traku, gdje korisnik može navigirati na prijašnju stranicu, na prikaz ispita, ili na stvaranje novog ispita.

Kod za ovu implementaciju je vrlo jednostavan, vidljivo na slici 63, gdje se „template“ sastoji od tri gumba, svaki koji pokreće novu metodu koja izvršava navigaciju.



Slika 62: *Prikaz izborničke trake (izvor: Autor)*

```
<template>
  <div class="sidebar fixed top-10 mt-12 left-0 w-24 bg-blue-50 overflow-x-
hidden divide-y divide-gray-200">
    <button @click="handleStats"
      class="shadow-sm flex items-center space-x-2 py-3 px-4 w-full hover:bg-
gradient-to-r from-blue-100 to-blue-200 rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:ring-offset-2 transition duration-150
ease-in-out"
      style="background-color: rgba(191, 219, 254, 0.8); margin-left: 0px;
border-radius: 0px; margin-bottom: 0px; border: none">
      <img class="h-6 w-6" alt="Stats"
src="https://c.animaapp.com/5MVuga3i/img/vector-1.svg" />
      <div class="text-sm font-semibold">Stats</div>
    </button>
    <button @click="back"
      class="shadow-sm flex items-center space-x-2 py-3 px-4 w-full hover:bg-
gradient-to-r from-blue-100 to-blue-200 rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:ring-offset-2 transition duration-150
ease-in-out"
      style="background-color: rgba(191, 219, 254, 0.8); margin-left: 0px;
border-radius: 0px; margin-top: 0; margin-bottom: 0px; border: none">
      <img class="h-6 w-6" alt="Back"
src="https://c.animaapp.com/5MVuga3i/img/vector-3.svg" />
      <div class="text-sm font-semibold">Back</div>
    </button>
    <button @click="makeTest"
```

```
      class="shadow-sm flex items-center space-x-2 py-3 px-4 w-full hover:bg-
gradient-to-r from-blue-100 to-blue-200 rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:ring-offset-2 transition duration-150
ease-in-out"
      style="background-color: rgba(191, 219, 254, 0.8); margin-left: 0px;
border-radius: 0px; margin-top: 0px; border: none;">
      <img class="h-6 w-6" alt="New Test"
src="https://c.animaapp.com/5MVuga3i/img/vector-2.svg" />
      <div class="text-sm font-semibold">Stvori novi Test!</div>
    </button>
  </div>
</template>
```

Slika 63: *Template dio koda za izbornička traku (izvor: Autor)*

3.8.2 Navigacijska traka

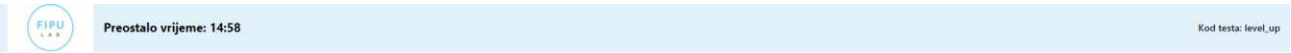
U trenutačnoj implementaciji koristimo dvije verzije Navbar komponente; statički i dinamički Navbar.

Jedina svrha statičke Navbar komponente jest da prikazuje ime trenutačne stranice, te varijablu za otvaranje SideBar komponente. Instanciramo komponentu tako da joj pružimo varijablu za manipulaciju side bar komponente, te ime stranice na koje se nalazimo, kao što je vidljivo na slici 64.

```
  <StaticNavbar @toggleSidebar="toggleSidebar" :pageName="pageName" />
  <SideBar :class="{ 'sidebar-overlay': true, 'sidebar-visible':
   sidebarVisible }" />
```

Slika 64: *Primjena statičke Navbar komponente u kodu, zajedno sa izborničkom trakom (izvor: Autor)*

Dinamička Navbar komponenta služi kako bismo prikazivali preostalo vrijeme dostupno u testu, ime testa koji se trenutačno polaže, te također drži logotip FIPULab-a, vidljivo na slici 65.



Slika 65: *Dinamička Navbar komponenta koji odbrojava preostalo vrijeme tokom ispita (izvor: Autor)*

3.9 Objavljivanje aplikacije

3.9.1 Frontend

Kada je došlo vrijeme za objavljivanje naše web aplikacije za studente, morali smo razmotriti nekoliko opcija za deployment frontenda. Ovdje su neki od popularnih izbora za frontend deployment i razlozi zašto su prikladni:

Netlify: Netlify je idealan za hosting statičnih web stranica i jednostavnih aplikacija. Nudi globalnu CDN (Content Delivery Network) za brzo isporučivanje sadržaja, besplatne SSL certifikate, automatsku kontinuiranu isporuku (CI/CD), te jednostavnu integraciju s GitHubom. Netlify također omogućava razvojne timove da koriste napredne funkcionalnosti poput A/B testiranja i uvođenja funkcionalnosti putem funkcija na rubu mreže. Ova platforma je posebno cijenjena zbog svoje jednostavnosti i fleksibilnosti u razvoju modernih web aplikacija, no budući da robustnije opcije kod isporuke, izabrali smo Vercel.

Firebase Hosting: Firebase, koji je dio Googleovog ekosustava, nudi hosting koji je duboko integriran s ostalim Firebase uslugama poput autentifikacije, baza podataka i cloud funkcija. Ova platforma je posebno korisna za projekte koji koriste Firebaseove backend usluge. Firebase Hosting nudi skalabilnost, brzu isporuku sadržaja putem globalnog CDN-a, te besplatne SSL certifikate. Također, pruža besplatni plan koji je vrlo izdašan za manje projekte, ali troškovi mogu značajno porasti s povećanjem korištenja. Zbog potrebe za hostingom koji možemo lagano skalirati, odlučili smo ne koristiti Firebase Hosting, zbog mogućnosti plaćanja naknada sa skaliranjem.

AWS Amplify: AWS Amplify je dio šireg Amazon Web Services (AWS) ekosustava i pruža moćan skup alata za razvoj i implementaciju aplikacija. Amplify je pogodan za projekte s visokim prometom i složenim zahtjevima, jer nudi besprijekornu integraciju s drugim AWS uslugama poput Lambda funkcija, S3 za pohranu, i CloudFront za isporuku sadržaja. Iako je snažan i fleksibilan, AWS Amplify može biti izazovan za početnike zbog svoje složenosti i potrebne konfiguracije. AWS servisi najbolje funkcioniraju sa drugim AWS uslugama, te se također plaća po koristnosti, stoga je odlučeno da se ne isplati u dužim rokovima.

Vercel: Krajnji izbor bio je Vercel, koji je poznat po svojoj snažnoj podršci za server-side rendering (SSR), posebno kada se koristi s Next.js frameworkom. Platforma koristi inteligentni

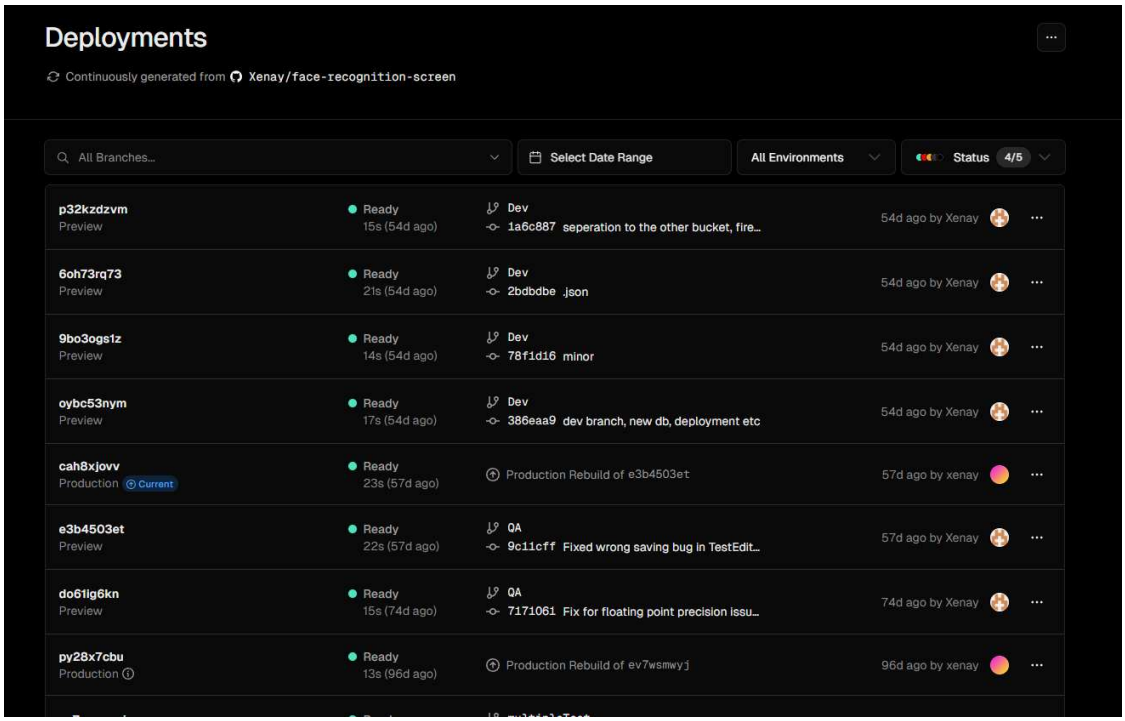
caching, omogućavajući brže izvođenje koda, čime se značajno smanjuje vrijeme učitavanja stranica. Vercel je savršen izbor za aplikacije koje zahtijevaju dinamično generiranje sadržaja u stvarnom vremenu, uz visoku performansu i stabilnost zahvaljujući svojoj globalnoj mreži. Također, Vercel nudi besplatni plan koji uključuje osnovne funkcionalnosti, dok su napredne mogućnosti dostupne kroz plaćene planove.

Pomoću Vercela stvorili smo tri verzije frontenda aplikacije, svaki koristeći drugi branch na github-u (dev, qa, prod/main). Na slici 66 vidi se „Deployments“ izbornik, koji pokazuje sve git commitove koji su napravljeni na svakoj grani (branch).

Svaki put kada se napravi commit ili promjena u repozitoriju, Vercel automatski stvara novi deployment. To znači da svaki put kada netko u timu "gurne" promjenu (push) u određenu granu repozitorija, Vercel prepoznaje tu promjenu i pokreće proces stvaranja novog deploymenta. Ovo omogućuje da se promjene brzo pregledaju i testiraju u izoliranom okruženju.

Na slici 66 također se vidi naziv svake grane koja je imala promjenu, kao i naziv commit-a koji je pokrenuo deployment. To omogućuje jednostavno praćenje tko je i na kojoj grani napravio promjenu. Primjerice, grane označene kao "Dev" ili "QA" automatski promoviraju promjene u produkcijsko okruženje nakon što su te promjene prošle potrebne provjere, dok „Main“ grana zahtjeva manualnu promociju. Ova automatizacija je korisna za kontinuiranu isporuku (Continuous Delivery) jer omogućava brzu iteraciju i testiranje novih značajki.

Ovaj sustav omogućuje visok stupanj automatizacije i osigurava da promjene budu pregledane i odobrene na odgovarajući način prije nego što postanu javno dostupne. Također, olakšava timovima da prate povijest promjena i brzo identificiraju koja grana je izvršila određeni deployment, što je ključno za održavanje stabilnosti i kontinuiteta aplikacije.



Slika 66: *Stavka „Deployments“ na stranici Vercel za projekt (izvor: Autor)*

Na slici 67, "Deployment Details" vidimo detaljan prikaz jednog specifičnog deploymenta u Vercelu. Ovaj prikaz pruža ključne informacije o statusu i karakteristikama deploymenta te omogućuje pregled logova i drugih podataka povezanih s njim.

Pod Detalje spadaju:

Status: Deployment je označen kao "Ready", što znači da je uspješno završen i aplikacija je spremna za korištenje.

Environment: Ovo pokazuje da je deployment trenutno aktivan u produkcijskom okruženju ("Production"), što znači da je ovo verzija aplikacije koja je dostupna krajnjim korisnicima.

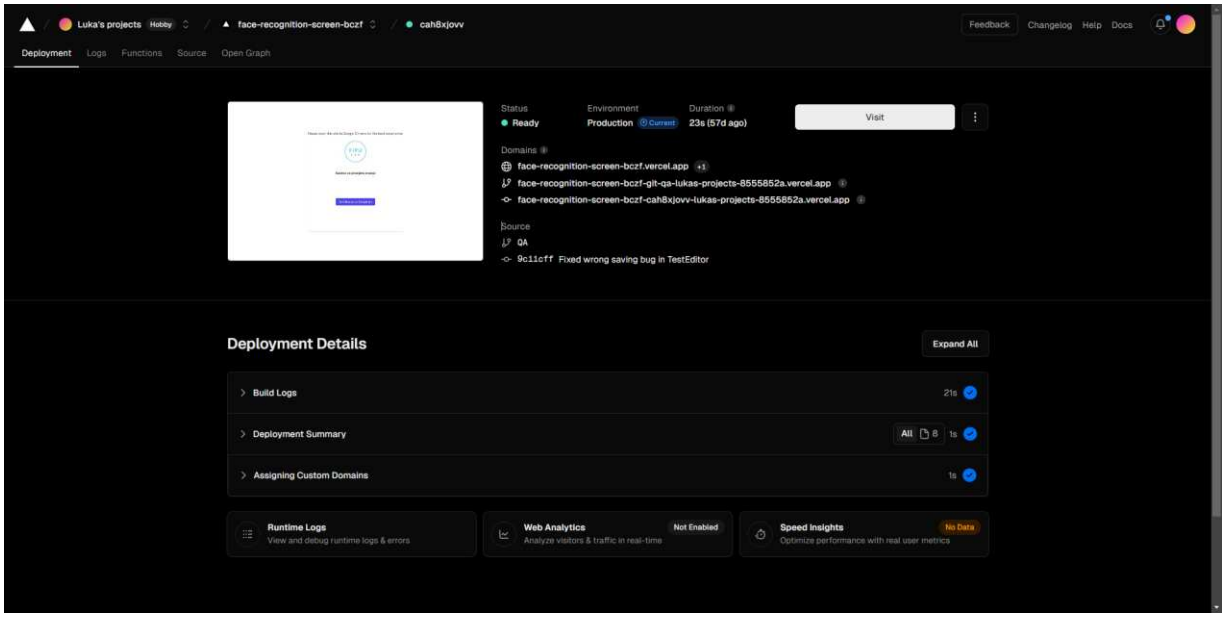
Duration: Ovaj deployment trajao je 23 sekunde, što nam daje uvid u vrijeme potrebno za izgradnju i postavljanje aplikacije.

Source: Deployment je pokrenut iz "QA" (Quality Assurance) grane, što znači da su promjene prvo testirane u QA okruženju prije nego što su promovirane u produkciju.

Domene: vidimo URL-ove na kojima je aplikacija dostupna. Ovdje su navedene različite domene koje su povezane s ovim deploymentom, uključujući zadanu Vercel domenu i eventualno prilagođene domene koje su dodane.

Build logs i Deployment summary opcije omogućavaju pregled procesa izgradnje aplikacije, koji uključuju sve korake izvršene tijekom izgradnje, kao i pregled sažetka procesa deploymenta.

Moguće je također dodati posebne domene za aplikaciju, ako ih imamo, pomoću opcije „Assign Custom Domains“, kao i pratiti promet te perfomanse aplikacije sa „Web Analytics“ i „Speed Analytics“ opcijama.



Slika 67: *prikaz detalja o pojedinom deploymentu na Vercel Dashboardu (izvor: Autor)*

3.9.2 Backend

Kada je riječ o hosting rješenjima za backend aplikacije, postoji nekoliko popularnih opcija koje nude različite prednosti i pogodnosti. Glavne značajke koje je trebalo ispuniti uključuju: automatizirani deployment aplikacije pri promjenama u repozitoriju, te da postoji besplatna verzija radi testiranja. Ovdje su neki od najčešće korištenih servisa:

Prva opcija koja je bila preporučena bila je DigitalOcean Droplet. Platforma je dizajnirana da bude developer-friendly, nudeći jednostavan UI i integracije s popularnim alatima kao što su GitHub i GitLab. DigitalOcean App Platform podržava više programskih jezika i okruženja te omogućava korisnicima da biraju između korištenja manageanih usluga ili potpunog prilagođavanja infrastrukture. Ova platforma je idealna za manje timove i startupe koji trebaju jednostavno, ali skalabilno hosting rješenje. Glavni problem ove solucije je što ne postoji

besplatna verzija, samo probna verzija, te zbog sporih performansa, odlućeno je razmotriti druge opcije.

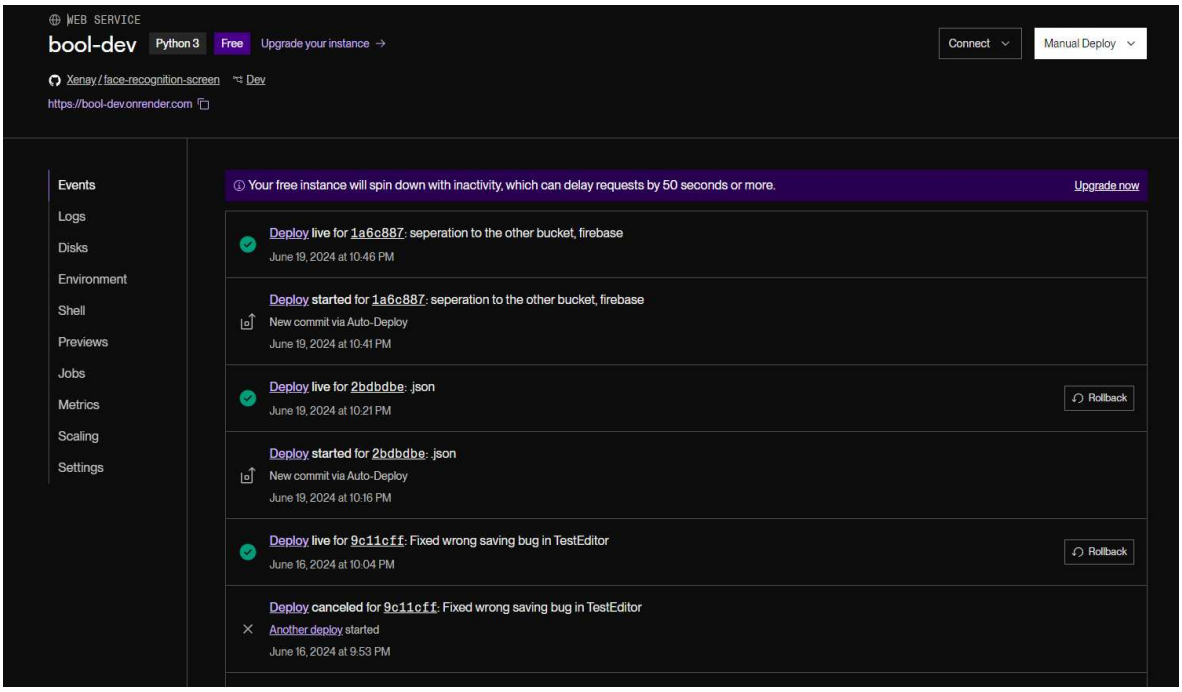
Heroku je cloud platforma koja omogućava jednostavan deploy, upravljanje i skaliranje aplikacija. Popularan je među developerima zbog svoje jednostavnosti i podrške za različite programske jezike. Heroku nudi "dyno" sustav, gdje se aplikacije pokreću na virtualnim kontejnerima. Heroku je idealan za timove koji žele brzo razvijati i implementirati aplikacije bez brige o infrastrukturi. Ova platforma bila je idealna za studente, jer je pružala besplatno okruženje za hosting backend djela aplikacije, no više ne postoji ta opcija. Također zbog već ranijeg iskustva sa aplikacijom odlučilo se na izbor novije hosting applikacije, Render.

Render je platforma koja omogućava jednostavno deployanje i hosting backend aplikacija s automatskim skaliranjem i SSL certifikatima. Također podržava razne vrste aplikacija, uključujući web servise, statične stranice, Docker kontejnerizirane aplikacije i pozadinske zadatke. Platforma nudi jednostavnu integraciju s GitHubom, omogućujući kontinuiranu isporuku (CI/CD) s automatskim deployanjem kada se pushaju promjene u repozitorij. Render je također popularan zbog svoje transparentnosti u cijeni, gdje korisnici plaćaju samo za ono što koriste, bez skrivenih troškova, te postoji besplatna verzija. Problem kod besplate verzije je što dobivamo ekstremno malo resursa, te također se backend ugasi sa neaktivnošću, pa postoji boot up period gdje se pokreće server, koji može potrajati i do deset minuta. Usprkos manama, odlučeno je da će se koristiti besplatna verzija ove aplikacije u razvoju, a kada je vrijeme za testiranja sa studentima, da će se kupiti plaćena verzija koja je online cijelo vrijeme.

Kao i kod Vercela, svaki projekt ima kategorije gdje možemo vidjeti sve opcije, metrike i zapise o stanju, performansama i mogućnost vertikalnog skaliranja.

Za ovaj projekt koriste se dvije instance baza podataka, jedna za razvoj i jedna za produkciju, kako ne bi dodavali nepotrebne ispite, opcije i podatke u glavnu bazu, te zaštitili podatke od mogućih grešaka u razvoju.

Pri svakoj promjeni u repozitoruju, aplikacija ponovno započinje deployment web servisa, kako bi automatizirali proces CI/CD. Povijest deploymenta za development server vidljiv je na slici 68.



Slika 68: *Prikaz Dashboarda projekta za stranicu Render.com (izvor: Autor)*

4. Evaluacija i buduća poboljšanja

4.1 Moguća poboljšanja u strukturi i kodu

4.1.1 Struktura koda

Iako je cilj apstrakcije koda u smislene cjeline postignut, postoji mogućnost za daljnju optimizaciju kako bi se smanjila redundantnost koda, omogućila dublja apstrakcija u komponente, te uvele „utility“ funkcije koje bi obavljale generičke operacije koje se više puta ponavljaju u kodu, poput konverzija između različitih tipova podataka.

Dodatna optimizacija koda može značajno poboljšati performanse aplikacije. Iako aplikacija trenutno radi relativno brzo, dodavanje novih funkcionalnosti može negativno utjecati na postojeći kod, ukoliko nije potpuno optimiziran.

Također, trebalo bi razmotriti minimizaciju koda u smislene cjeline. Iako struktura Vue.js-a omogućava spajanje templatinga, logike i CSS-a u jednu datoteku, prevelike datoteke mogu postati teško čitljive osobama koje će raditi na toj funkcionalnosti. Segmentiranje koda u manje cjeline značajno bi poboljšalo čitljivost i iskustvo razvijatelja (developer experience). Međutim, potrebno je pronaći ravnotežu između veličine koda i broja segmenata, jer prevelika segmentacija može također dovesti do nečitljivog koda.

4.1.2 Prebacivanje svih kritičnih radnji na backend

U trenutnoj verziji, većina podataka koji se prikazuju studentu, te nekolicina kritičnih metoda vidljive su na aplikaciji. Zbog sigurnosti, važno je sve radnje abstrahirati preko servera; odgovore na pitanja, trenutni broj bodova te druge kritične podatke. Neke od navedenih funkcionalnosti nemoguće je trenutno prebaciti na server, zbog brzine izvođenja radnji, no korištenjem VPS-a (Virtual private server) ili servera bržih performansa moglo bi se ostvariti potpuna sigurnost od manipulacije podatcima.

4.1.3 Širenje mogućnosti strukture ispita

Trenutna verzija aplikacije primarno je usmjerena na testiranje znanja programskog jezika „Javascript“, ali postoji značajan potencijal za proširenje funkcionalnosti kako bi se omogućilo testiranje šireg spektra znanja i vještina. Korištenjem modularnog pristupa, aplikacija se može prilagoditi za različite predmete, uključujući ali ne ograničavajući se na, druge programske jezike, matematiku, fiziku, kemiju, ekonomiju, te društvene znanosti.

Osim promjene sadržaja pitanja, moguće je proširiti strukturu ispita kako bi se uključili različiti tipovi zadataka, poput zadataka s višestrukim odgovorima, otvorenim pitanjima, zadacima povezivanja, rangiranja ili čak simulacija i praktičnih zadataka koji zahtijevaju rješavanje problema unutar specifičnih okruženja. Ovo bi omogućilo nastavnicima da prilagode ispite specifičnim potrebama i ciljevima svojih tečajeva, pružajući studentima raznovrsnija i dinamičnija iskustva učenja.

Na kraju, proširenje mogućnosti lokalizacije i internacionalizacije omogućilo bi korištenje aplikacije na različitim jezicima i u različitim obrazovnim sustavima, čime bi se povećala njena dostupnost i korisnost na globalnoj razini. Kroz ove nadogradnje, aplikacija bi mogla postati univerzalni alat za evaluaciju znanja i vještina u raznim disciplinama i kontekstima.

4.2 **Rezultati**

U sklopu ovog diplomskog rada razvijena je moderna web aplikacija namijenjena automatizaciji izrade, ocjenjivanja i praćenja ispita putem integracije umjetne inteligencije.

Aplikacija je postavljena kao full-stack rješenje, s korisničkim sučeljem izrađenim u Vue.js-u, backendom izgrađenim na FastAPI frameworku, te Firebaseom kao glavnim servisom za autentifikaciju i pohranu podataka. Glavni ciljevi uključivali su stvaranje stabilnog, sigurnog i prilagodljivog sustava koji omogućuje učinkovit rad u stvarnom vremenu, a sve uz minimalan pritisak na resurse sustava.

Testiranje je provedeno s grupom studenata u stvarnim uvjetima, gdje su evaluirane funkcionalnosti kao što su detekcija lica, analiza rezultata ispita, te generalna stabilnost sustava. Rezultati su pokazali da sustav zadovoljava sve osnovne zahtjeve, s minimalnim pogreškama u detekciji i ocjenjivanju. Osim toga, povratne informacije od korisnika pokazale su visoku razinu zadovoljstva aplikacijom, posebice u kontekstu jednostavnosti korištenja i jasnoće sučelja.

Među ključnim rezultatima je i uspješna integracija umjetne inteligencije u proces detekcije lica, koja je omogućila učinkovitu provjeru integriteta ispita, dok je analiza snimki zaslona i kamere pokazala visoku točnost u detekciji sumnjivog ponašanja. Implementacija sustava negativnih bodova također je polučila pozitivne rezultate, gdje su studenti imali dodatnu motivaciju za preciznost u odgovaranju.

Skalabilnost sustava testirana je na način da se aplikacija koristi na različitim uređajima, uz varijacije u brzini internetske veze. Aplikacija je pokazala visok stupanj prilagodljivosti, bez značajnih padova performansi, čak i u uvjetima opterećenja.

5. Zaključak

Ovaj diplomski rad pokazao je kako primjena moderne tehnologije i umjetne inteligencije može značajno unaprijediti proces izrade, ocjenjivanja i praćenja ispita. Razvijena aplikacija uspješno je ostvarila sve postavljene ciljeve, pružajući siguran, učinkovit i skalabilan alat za obrazovne institucije.

Korištenje Vue.js-a za frontend omogućilo je stvaranje responzivnog i intuitivnog korisničkog sučelja, dok je backend izrađen u FastAPI frameworku osigurao brzu i pouzdanu obradu podataka. Integracija Firebasea kao platforme za autentifikaciju i pohranu podataka pokazala se kao optimalan izbor, pružajući potrebnu sigurnost i jednostavnost korištenja.

Unatoč određenim tehničkim izazovima, poput problema s performansama na slabijim uređajima ili složenosti u implementaciji detekcije lica, aplikacija je uspjela postići stabilnost i pouzdanost potrebnu za svakodnevnu upotrebu. Također, iterativni pristup razvoju, uz kontinuiranu povratnu informaciju od korisnika, omogućio je stalno poboljšanje aplikacije i prilagodbu specifičnim potrebama.

Budući razvoj aplikacije mogao bi uključivati širenje funkcionalnosti na druge predmete i tipove zadataka, kao i optimizaciju postojećih rješenja za još veću učinkovitost i sigurnost. Uz to, integracija naprednijih AI modela mogla bi dodatno unaprijediti proces generiranja ispita i analize rezultata, čime bi aplikacija postala još moćniji alat u obrazovanju. U konačnici, ovaj rad predstavlja značajan doprinos razvoju tehnologija za e-učenje i potvrđuje potencijal umjetne inteligencije u obrazovnim procesima.

6. Literatura

[1] Vue.js documentation, . URL https://vuejs.org/guide/introduction.html. Accessed: 2024-08-21

[2] Tailwind CSS documentation, . URL https://tailwindui.com/documentation. Accessed: 2024-08-21

[3] ViteJS guide, . URL https://vitejs.dev/guide/ . Accessed: 2024-08-21

[4] Render documentation, . URL https://docs.render.com/ Accessed: 2024-08-22

[5] Vercel documentation, . URL https://vercel.com/docs Accessed: 2024-08-22

[6] PrismJS examples, . URL https://prismjs.com/ Accessed: 2024-08-22

[7] OpenCV documentation, . URL https://docs.opencv.org/ Accessed: 2024-08-22

[8] DavidCochard, RetinaFace: A Face Detection Model for High Resolution Images, 16.02.2024. URL

https://medium.com/axinc-ai/retinaface-a-face-detection-model-designed-for-high-resolution-6c3900771a01 Accessed: 2024-08-20

[9] Google Firebase documentation, . URL https://firebase.google.com/docs Accessed: 2024-08-22

[10] FastAPI Learn, . URL https://fastapi.tiangolo.com/learn/ Accessed: 2024-08-22

[11] Deploying Git Repositories with Vercel, . URL https://vercel.com/docs/deployments/git Accessed: 2024-08-22

[12] Production checklist for launch, . URL https://vercel.com/docs/production-checklist#production-checklist-for-launch Accessed: 2024-08-22

[13] Builds for Vercel Deployment, . URL https://vercel.com/docs/deployments/builds Accessed: 2024-08-22

[14] Best practices, Production Deployment, . URL https://vuejs.org/guide/best-practices/production-deployment.html Accessed: 2024-08-22

[15] Get Started With CV2, . URL https://opencv.org/get-started/ Accessed: 2024-08-22

[16] Env Variables and Modes, . URL https://vitejs.dev/guide/env-and-mode.html Accessed: 2024-08-22

[17] Deploy a FastAPI App, . URL https://docs.render.com/deploy-fastapi Accessed: 2024-08-22

[18] Connect GitHub, . URL https://docs.render.com/github Accessed: 2024-08-22

[19] Firebase Realtime Database, . URL https://firebase.google.com/docs/database Accessed: 2024-08-22

[20] Cloud Storage for Firebase, . URL https://firebase.google.com/docs/storage Accessed: 2024-08-22

[21] Firebase Authentication, . URL https://firebase.google.com/docs/auth Accessed: 2024-08-22

[22] Defining a Store, . URL https://pinia.vuejs.org/core-concepts/ Accessed: 2024-08-22

[23] Pinia Getters, . URL https://pinia.vuejs.org/core-concepts/getters.html Accessed: 2024-08-22

[24] Pinia Actions, . URL https://pinia.vuejs.org/core-concepts/actions.html Accessed: 2024-08-22

[25] Pinia State, . URL https://pinia.vuejs.org/core-concepts/state.html Accessed: 2024-08-22

[26] Get started with the Gemini API, . URL https://ai.google.dev/gemini-api/docs Accessed: 2024-08-22

[27] Docker Docs, . URL https://docs.docker.com/ Accessed: 2024-08-22

7. Popis slika