SVEUČILIŠTE JURJA DOBRILE U PULI
FAKULTET INFORMATIKE

Filip Bastijanić

Aplikacija za provođenje kvizova potpomognuta velikim jezičnim modelima

ZAVRSNI RAD

Pula, kolovoz, 2025. godine

SVEUČILIŠTE JURJA DOBRILE U PULI
FAKULTET INFORMATIKE

Filip Bastijanić

Aplikacija za provođenje kvizova potpomognuta velikim jezičnim modelima

ZAVRŠNI RAD

JMBAG: 0016138059, redoviti student
Studijski smjer: Informatika

Kolegij: Web aplikacije
Znanstveno područje : Društvene znanosti
Znanstveno polje : Informacijske i komunikacijske znanosti
Znanstvena grana : Informacijski sustavi i informatologija

Mentor: doc. dr. sc. Nikola Tanković

Pula, kolovoz, 2025. godine

Sažetak

U ovom radu istražuje se primjena velikih jezičnih modela u razvoju aplikacija za provođenje kvizova. Fokusira se na implementaciju sustava u Elixir programskom jeziku, koji omogućuje komunikaciju u stvarnom vremenu i dinamičko generiranje pitanja. Aplikacija nudi korisnicima mogućnost interaktivnog sudjelovanja u kvizovima, gdje se pitanja generiraju na temelju unosa korisnika i odgovori se evaluiraju u stvarnom vremenu. Na taj način rješava problem postojećih rješenja, te kreiranje i vođenje kvizova postaje jednostavnije i učinkovitije.

Ključne riječi : AI, LLM, Elixir, Phoenix, WebSocket

Abstract

This paper explores the application of large language models in the development of quiz applications. It focuses on the implementation of the system using the Elixir programming language, which enables real-time communication and dynamic question generation. The application offers users the possibility of interactive participation in quizzes, where questions are generated based on user input and answers are evaluated in real time. In this way, it solves the shortcomings of existing solutions, making quiz creation and management simpler and more efficient.

Keywords : AI, LLM, Elixir, Phoenix, WebSocket

Sadržaj

1 Uvod

U posljednjih nekoliko godina svjedočimo velikoj ekspanziji velikih jezičnih modela i umjetne inteligencije. Takvi modeli već vješto repliciraju ljudski govor, medije i funkcije koje su nekad obnašali ljudi, te su sposobni ponuditi sve od automatizirane podrške, pa do kreacije sadržaja kao što su slika i zvuk.

U kontekstu kvizova, veliki jezični modeli zauzimaju ulogu u automatizaciji i personalizaciji sadržaja. Njihova sposobnost razumijevanja ljudskog izražaja i generiranja jezika omogućuje dinamičko sastavljanje pitanja, prilagodbu težine kviza te evaluaciju odgovora u stvarnom vremenu.

Cilj ovog rada je razviti zabavnu i intuitivnu aplikaciju za provođenje kvizova, koja koristi mogućnosti velikih jezičnih modela za dinamičko generiranje pitanja i evaluaciju odgovora. Aplikacija je zamišljena kao zabavni i edukativni alat koji korisniku u stvarnom vremenu postavlja pitanja, bilježi odgovore te pruža povratnu informaciju u obliku rezultata i poretka. Rad prikazuje arhitekturu sustava, korištene tehnologije i mogućnosti daljnjeg razvoja ovakvih kviz aplikacija.

2 Postojeća rješenja

S obzirom na rapidnu evoluciju velikih jezičnih modela, mnoga postojeća rješenja koriste ove modele za različite svrhe, uključujući obrazovne aplikacije i kviz sustave. Kako bi se bolje razumjelo što treba implementirati u ovom projektu, potrebno je analizirati postojeće platforme koje koriste umjetnu inteligenciju za izradu i provođenje kvizova. U ovom poglavlju bit će prikazane platforme Quiziz, Kahoot i Quiz.com, a fokus će biti na njihovim funkcionalnostima, a ne na tehničkim detaljima implementacije. Također, bit će prikazano kako navedene platforme integriraju umjetnu inteligenciju u svoje sustave.

2.1 Kahoot

Kahoot je jedna od najpopularnijih platformi za izradu i provođenje kvizova u obrazovnom okruženju [1]. Omogućuje korisnicima stvaranje kvizova, učenje jezika te prezentiranje gradiva. Aplikacija je namijenjena upotrebi u učionicama, ali i za osobno učenje.

Kvizovi na Kahootu mogu primiti do 500 sudionika, ovisno o razini pretplate. Većina kvizova koristi pitanja višestrukog izbora, iako su dostupna i pitanja otvorenog tipa, koja se dodatno naplaćuju. Organizator kviza dijeli svoj ekran s pitanjima, a sudionici odgovaraju putem vlastitih uređaja. Sve informacije o pitanjima, točnim odgovorima i ponuđenim opcijama prikazuju se isključivo na ekranu organizatora. Tijekom kviza, rezultati se prikazuju na uređajima sudionika, uključujući poziciju u poretku.

Kahoot nudi mogućnosti generiranja pitanja pomoću velikih jezičnih modela, ali automatska evaluacija odgovora pomoću umjetne inteligencije nije dostupna. Također, platforma ne pruža mogućnost praćenja napretka korisnika u kvizovima. Sučelje Kahoota je jednostavno i intuitivno, što olakšava kreiranje i sudjelovanje u kvizovima.

2.2 Wayground

Wayground [3] je još jedna platforma za izradu kvizova koja se koristi u obrazovnom kontekstu. Platforma nudi slične funkcionalnosti kao Kahoot, ali s dodatnim mogućnostima prilagodbe i analitičkog praćenja.

Korisnici mogu stvarati kvizove s različitim tipovima pitanja, uključujući višestruki izbor, otvorena pitanja i pitanja sa slikama. Wayground koristi velike jezične modele za generiranje pitanja i evaluaciju odgovora, a teme kviza mogu se definirati pomoću prompta, PDF-a ili video materijala. Organizator može odrediti broj pitanja, temu, težinu i dodatni kontekst, te po potrebi mijenjati pitanja tijekom trajanja kviza.

Za razliku od Kahoota, Wayground prikazuje sve informacije o pitanju, uključujući točnost odgovora i bodove, izravno na sučelju sudionika. Igrači ne ovise o ekranu organizatora i ne moraju čekati da organizator pokrene sljedeće pitanje. Statistika i napredak sudionika dostupni su nakon završetka kviza. Sučelje Waygrounda

je bogato funkcijama, ali zahtijeva nešto više vremena za privikavanje zbog veće složenosti. Kvizovi na Waygroundu također mogu primiti do 500 sudionika.

2.3 Quiz.com

Quiz.com je platforma orijentirana zabavi [2], koja omogućuje korisnicima stvaranje i sudjelovanje u kvizovima različitih tema, uključujući opću kulturu, povijest, znanost i druge.

Platforma je jednostavna za korištenje i nudi mogućnost kreiranja kvizova s pitanjima višestrukog izbora, istinitosti ili lažnosti te otvorenim pitanjima. Korisnici mogu generirati kvizove pomoću AI alata, koji pruža prijedloge za pitanja i odgovore ili omogućuje automatsko generiranje cijelog kviza i popratnih slika. Kviz se može kreirati na temelju PDF-a ili pomoću prompta u kojem korisnik definira temu i broj pitanja.

Quiz.com ne pruža mogućnosti praćenja napretka igrača niti automatske evaluacije odgovora pomoću umjetne inteligencije. Sučelje je jednostavno i intuitivno, što omogućuje brzo kreiranje i sudjelovanje u kvizovima, a sve informacije o trenutnom pitanju i bodovima prikazane su i na ekranu igrača i organizatora. Također, kviz nudi čitanje pitanja na glas uz pomoću glasa generiranog umjetnom inteligencijom. Platforma podržava do 300 sudionika po kvizu.

3 Utvrđivanje zahtjeva

Kako bi uspješno implementirali aplikaciju prije početka razvoja, potrebno je definirati zahtjeve korisnika i sustava. Zahtjevi u sustavu su opis onoga što sustav mora raditi [12]. U ovom poglavlju definirati će se funkcionalni i nefunkcionalni zahtjevi koji su određeni analizom postojećih platformi, kao i `UML Use Case` dijagram.

3.1 Funkcionalni zahtjevi

Funkcionalni zahtjevi opisuju usluge i funkcionalnosti koje sustav treba pružati, način na koji se treba ponašati u određenim situacijama te reakcije na specifične ulazne podatke. Oni također mogu uključivati i ograničenja, odnosno odrediti što sustav ne smije činiti [12]. U slučaju ovog projekta definirani su sljedeći funkcionalni zahtjevi:

1. Korisnici trebaju imati mogućnost registracije i prijave u sustav kako bi mogli pohranjivati svoje rezultate i pratiti napredak.

2. Autorizirani korisnici trebaju moći kreirati nove kvizove s različitim temama.

3. Veliki jezični model na temelju zadane teme, zadane težine, dodatnog konteksta i broja pitanja treba generirati pitanja.

4. Veliki jezični model mora omogućiti automatsku evaluaciju odgovora korisnika i pružiti povratnu informaciju o točnosti.

5. Korisnici trebaju imati mogućnost samostalnog prijavljivanja na kviz.

6. Autorizirani korisnici trebaju moći započeti kviz, slati nova pitanja i ispraviti bodovanje u slučaju pogrešnog bodovanja.

7. Korisnici trebaju imati mogućnost odgovaranja na pitanja tijekom kviza.

8. Za vrijeme kviza korisnici trebaju moći vidjeti svoje trenutne rezultate i napredak.

9. Rezultati korisnika (bodovi, vrijeme rješavanja, itd.) trebaju biti pohranjeni u bazu podataka radi praćenja napretka.

10. Autorizirani korisnici trebaju imati pristup svojim prošlim kvizovima i rezultatima.

11. Mogućnost uređivanja profila, resetiranja lozinke i upravljanja postavkama računa.

3.2 Nefunkcionalni zahtjevi

Nefunkcionalni zahtjevi odnose se na svojstva i kvalitete sustava, a ne na specifične funkcionalnosti koje sustav pruža. Oni određuju uvjete pod kojima sustav treba raditi, uključujući performanse, sigurnost, pouzdanost, skalabilnost i jednostavnost korištenja. Drugim riječima, nefunkcionalni zahtjevi definiraju „kako“ sustav obavlja određene funkcionalnosti, a ne „što“ sustav radi [12]. U slučaju ovog projekta definirani su sljedeći nefunkcionalni zahtjevi:

1. Sustav mora omogućiti generiranje pitanja i evaluaciju odgovora u realnom vremenu.

2. Sustav mora osigurati zaštitu korisničkih podataka i spriječiti neovlašteni pristup.

3. Sustav treba podržavati do 500 ili više istovremenih korisnika i veliki broj kvizova bez degradacije performansi, tako da konkurira Kahoot-u i drugim sličnim platformama.

4. Sustav treba biti dizajniran za dugoročnu održivost, uključujući jednostavno ažuriranje i održavanje.

5. Sustav mora pružiti intuitivno i jednostavno korisničko sučelje radi olakšanog korištenja.

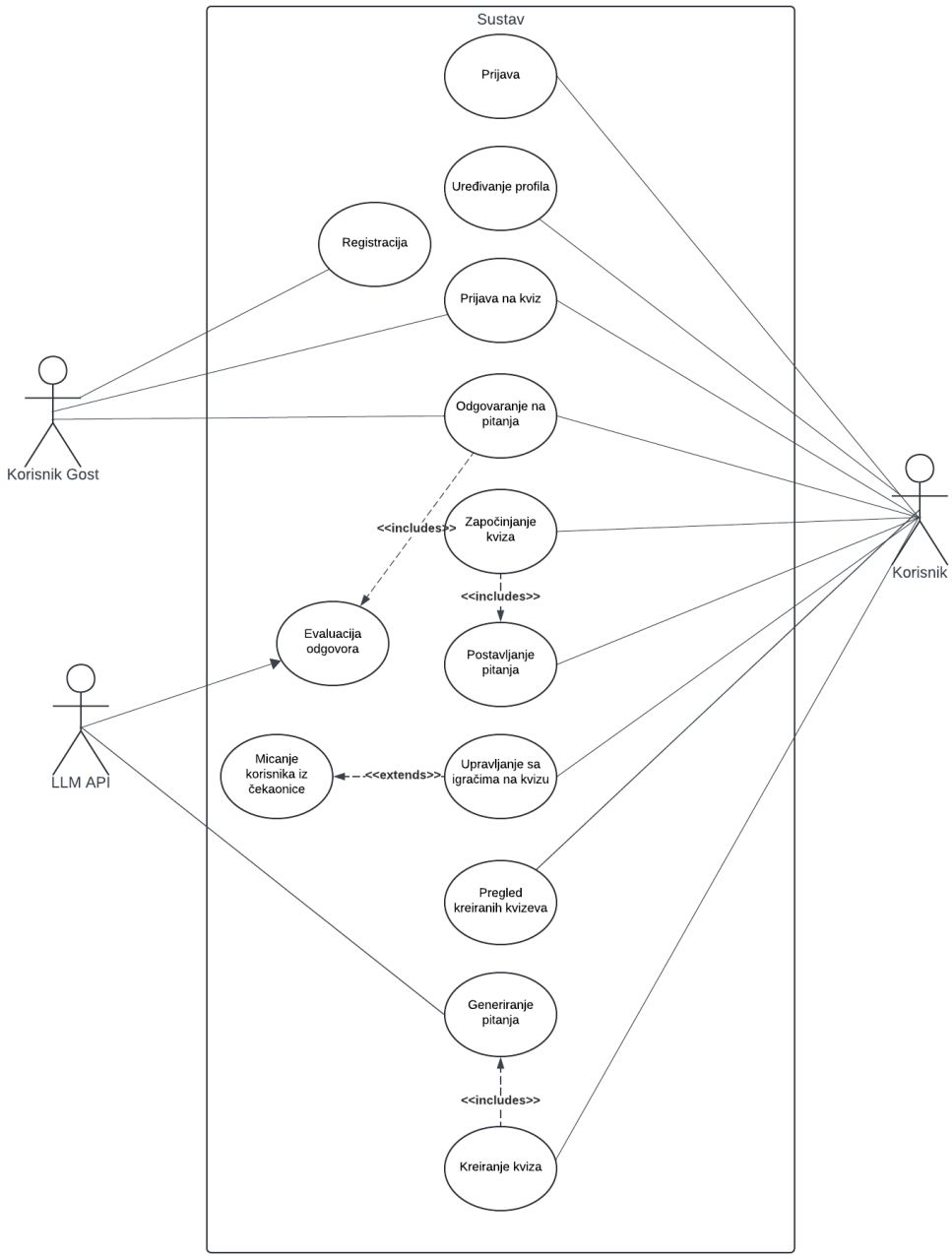
3.3 UML Use Case dijagram

Na slici 1 prikazan je `UML Use Case` dijagram koji opisuje osnovne funkcionalnosti kviz sustava i interakcije korisnika sa sustavom. Glavni akteri u sustavu su

gosti i autorizirani korisnici. LLM je prikazan kao akter i on će generirati pitanja i evaluirati odgovore korisnika. Gosti imaju mogućnost registracije i prijave, prijavljivanja na kviz i odgovaranja na pitanja.

Autorizirani korisnici osim navedenih funkcionalnosti imaju dodatne mogućnosti poput kreiranja novih kvizova, upravljanja korisničkim računima i provođenja kvizova (pokretanje kviza, dodavanje pitanja, ispravak bodovanja).

Ovaj dijagram vizualno prikazuje funkcionalne zahtjeve sustava te jasno pokazuje koje usluge sustav pruža pojedinim tipovima korisnika.



Slika 1: UML Use Case dijagram

4 Korištene tehnologije

4.1 Elixir

Elixir je alternativni jezik koji pojednostavljuje razvoj aplikacija na Erlang VM (BEAM) tehnologiji. Nudi modernu sintaksu i alate koji olakšavaju razvoj, testiranje i održavanje aplikacija. Često se koristi u web razvoju, posebno s Phoenix frameworkom, koji omogućuje brzo i učinkovito razvijanje web aplikacija.

4.1.1 Erlang

Erlang je razvojna platforma koja se koristi za izgradnju skalabilnih i pouzdanih sistema bez prekida rada. Za njegov razvoj je zaslužan Joe Armstrong, a prvi put je predstavljen 1986. godine u Ericssonu. Izmšljen je za razvoj telekomunikacijskih sistema, ali se danas koristi i u drugim područjima. Erlang je poznat po svojoj sposobnosti da podrži visoku dostupnost i otpornost na greške, što ga čini idealnim za aplikacije koje zahtijevaju stalnu dostupnost. Prema Saši Juriću [6], ovo su ključne karakteristike Erlanga:,

- **Otpornost na greške**
  Erlang procesi su potpuno izolirani i ne dijele memoriju, pa pad jednog procesa ne utječe na ostale. Greške su lokalizirane, a sustav može automatski pokrenuti novi proces umjesto onog koji je pao.

- **Skalabilnost**
  Procesi komuniciraju asinkronim porukama bez potrebe za zaključavanjem ili semaforima, što pojednostavljuje razvoj. Sustav se sastoji od velikog broja paralelnih procesa, a virtualna mašina učinkovito koristi sve CPU jezgre.

- **Distribucija**
  Komunikacija između procesa je jednaka, bez obzira nalaze li se na istom ili različitim računalima. Sustav se lako može distribuirati na više strojeva, čime se povećava skalabilnost i otpornost.

- **Responzivnost**
  Erlang koristi preemptivne raspoređivače koji ravnomjerno dijele vrijeme među procesima, pa niti jedan proces ne može blokirati sustav. I/O operacije ne blokiraju ostale procese, što osigurava visoku responzivnost aplikacije.

4.1.2 Phoenix razvojni okvir

Phoenix je web razvojni okvir napisan u Elixiru, koji koristi Erlang VM za izgradnju skalabilnih i održivih web aplikacija [9]. Ovaj razvojni okvir temeljen na MVC (Model-View-Controller) arhitekturi posjeduje razne alate za razvoj web aplikacija. Koristeći Cowboy i Plug alate, Phoenix omogućuje jednostavno upravljanje HTTP zahtjevima i izgradnju web poslužitelja, dok LiveView omogućuje izgradnju interaktivnih korisničkih sučelja bez potrebe za pisanjem JavaScript koda. Također, ovaj alat podržava WebSocket komunikaciju kroz alat "Channels", te

praćenje prisutnosti korisnika pomoću "Presence" modula. Ističe se i po integriranom ORM (Object-Relational Mapping) sustavu, Ecto, koji olakšava rad s bazama podataka.

Za potrebe ovog projekta, Phoenix je korišten za izgradnju poslužiteljske strane aplikacije. U nastavku su opisane ključne komponente Phoenix frameworka koje su korištene u razvoju poslužitelja.

4.1.3 Channels

Ovaj modul Phoenix razvojnog okvira omogućuje komunikaciju u pravom vremenu između milijuna povezanih klijenata [9]. Ovakav tip tehnologije može se koristiti za mnoge namjene kao što su chat aplikacije, notifikacije, vijesti i slično. U takvom tipu komunikacije klijenti se povezuju na određeni server putem WebSocket veze, a zatim se povezuju na određene teme (topics) unutar te veze. Unutar tih tema klijenti mogu primati poruke od servera i slati poruke serveru u stvarnom vremenu. Za razliku od HTTP zahtjeva, ovakvi kanali omogućuju dugo živeću vezu između klijenta i servera, što je podržano Erlang VM procesima. Ovakva arhitektura omogućuje skalabilnost, jer omogućava milijune istovremenih veza i slanje stotine tisuća poruka po sekundi sa razumnom latencijom [10].

4.1.4 Ecto

Ecto je ORM (Object-Relational Mapping) alat za Elixir [4]. Object–Relational Mapping je tehnika koja omogućuje rad s bazama podataka koristeći objekte i klase umjesto SQL upita. Tablice se preslikavaju kao klase, a redovi kao objekti tih klasa, time zamjenjujući potrebu za pisanjem SQL upita. Možemo reći da se ecto dijeli na četiri glavne komponente: repozitorije, sheme, upite i changesete.

Ecto repozitorij

Repozitorij je modul koji predstavlja vezu s bazom podataka, točnije odgovara na pitanje gdje se podaci nalaze. Putem repozitorija možemo stvarati, ažurirati, brisati, te dohvaćati podatke koristeći unaprijed definirane funkcije. Repozitorij zahtijeva autentifikacijske podatke za komunikaciju s bazom.

Ecto shema

Shema se koristi za mapiranje vanjskih podataka u Elixir strukture. Na taj način omogućuje rad s bazom koristeći Elixir objekte, umjesto pisanja izravnih SQL upita.

Ecto upit

Ovaj modul omogućava pisanje upita za dohvaćanje podataka iz baze u Elixir sintaksi. Ecto upiti se mogu kombinirati kao transakcije.

Ecto Changeset

Ovaj modul omogućava praćenje i validaciju promjena prije nego što se primijene na podatke. Na taj način osigurava integritet podataka i sprječava neželjene promjene.

U kontekstu projekta, Llama 4 se koristi za generiranje pitanja i evaluaciju odgovora korisnika u kvizovima. Groq API omogućuje pristup Llama 4 modelu putem jednostavnog HTTP sučelja, što olakšava integraciju s aplikacijama i uslugama.

4.2 Llama 4 (Groq API)

Llama 4 predstavlja najnoviju iteraciju velikog jezičnog modela (LLM) razvijenog od strane kompanije Meta, a objavljena je u travnju 2025. godine. Ovaj model temelji se na arhitekturi *mixture-of-experts* (MoE), što omogućuje značajno povećanje performansi i efikasnije korištenje resursa u usporedbi s prethodnim verzijama. Za razliku od ranijih generacija, Llama 4 je nativno multimodalan, što znači da osim teksta može obrađivati i slike te generirati odgovore kombiniranjem različitih modaliteta podataka [11].

Python

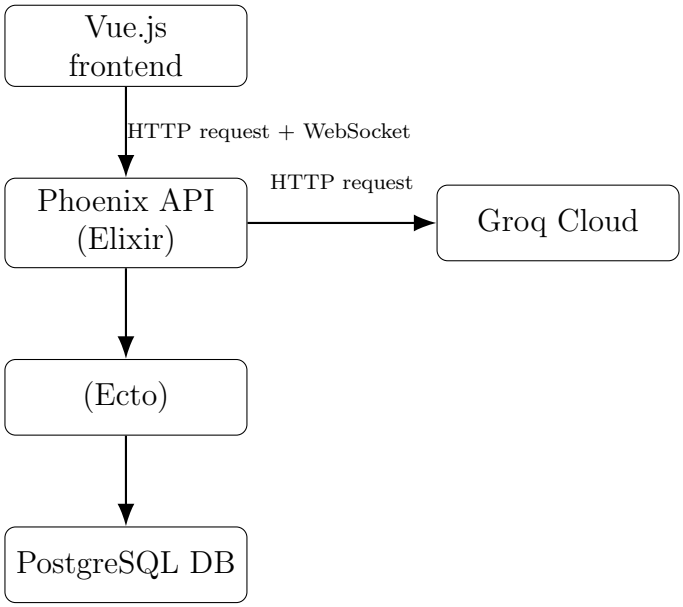
```python
import os

from groq import Groq

client = Groq(
    api_key=os.environ.get("GROQ_API_KEY"),
)

chat_completion = client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": "Explain the importance of fast language models",
        }
    ],
    model="llama-3.3-70b-versatile",
)

print(chat_completion.choices[0].message.content)
```

Slika 2: Primjer Groq API zahtjeva u Pythonu

5 Implementacija sustava

Kako bi uspješno implementirali ovaj sustav, ključno je koristiti prigodne tehnologije. U ranijem poglavlju navedene i definirane su sve tehnologije koje su ključne za glavne funkcionalnosti, no korištene su i dodatne tehnologije kako bi se ucjelovio sustav.

Za razvoj sučelja koristit će se Vue.js 3, Pinia za upravljanje stanjem aplikacije i shadcn kao UI komponentni sustav. Taj odjeljak sustava komunicira s poslužiteljom pomoću HTTP zahtjeva i WebSocket veze. Poslužitelj je izgrađen pomoću Phoenix frameworka i komunicira Http zahtjevima prema Groq cloud API-ju, od kojeg prima informacije o pitanjima i odgovorima korisnika. Ecto se koristi za interakciju sa PostgreSQL bazom podataka, stoga neće biti potrebe za pisanjem SQL upita. U praktične svrhe aplikacija će biti logički podijeljena u dvije cjeline. Kako bi isporučili aplikaciju i omogućili jednostavno postavljanje, koristi se Docker za kontenjerizaciju poslužitelja, korisničkog sučelja i baze podataka. U slučaju suradnje na programskom kodu, implementirani su CI/CD procesi na repozitoriju, uz pomoć Github akcija, koji olakšavaju razvoj i sučelja i poslužitelja.



Slika 3: Arhitektura sustava

5.1 Modeliranje podataka

Kako je prije navedeno umjesto relacija koristi se pristup modeliranja podataka temeljen na shemama. To nam omogućava Ecto kroz svoje funkcionalnosti za definiranje i upravljanje shemama. Definiranje sheme je vrlo jednostavno i može se postići korištenjem phoenix generate naredbe. U ovom slučaju, na primjer, `mix phx.gen.schema Account accounts email:string password_hash:string`. Nakon generiranja sheme, potrebno je izvršiti migraciju baze podataka kako bi se shema odrazila u bazi.

U ovom poglavlju opisat će se sve sheme koje se koriste u sustavu.

Account

Služi kako bi opisao korisnički račun u sustavu. U ovu shemu su uključeni svi potrebni atributi za autentifikaciju i autorizaciju korisnika. Na ovoj shemi definirano je da email mora imati ispravni format i ne smije biti dulja od 160 znakova. Lozinka se neće pohraniti u izvornom obliku nego će biti enkriptirana prije spremanja, a za to će se pobrinuti funkcija `put_password_hash/1`.

Naziv stupca	Tip podatka	Opis
id	uuid	Jedinstveni identifikator računa
email	string	Email adresa korisnika
password_hash	string	Hash lozinke korisnika
inserted_at	naive_datetime	Vremenska oznaka kreiranja računa
updated_at	naive_datetime	Vremenska oznaka izmjene računa

Tablica 1: Struktura tablice računa

Ograničenja:

- **email** mora biti jedinstven i u ispravnom formatu (mora sadržavati znak @, bez razmaka).
- Maksimalna duljina email adrese: 160 znakova.
- **password_hash** mora biti spremljen kao hash, ne u čistom tekstu.

Funkcije sheme:

- `put_password_hash/1` - hashira lozinku prije spremanja u bazu.

User

Predstavlja korisnika sustava. U ovu shemu su uključeni svi atributi koji opisuju korisnika kao što su biografija spol i puno ime. Ova shema također uključuje vanjske ključeve koji povezuju korisnika s njegovim računom.

Atributi	Tip podatka	Opis
id	uuid	Jedinstveni identifikator korisnika
full_name	string	Puno ime korisnika
gender	string	Spol korisnika
biography	text	Biografija korisnika
inserted_at	naive_datetime	Vremenska oznaka kreiranja korisnika
updated_at	naive_datetime	Vremenska oznaka izmjene korisnika

Tablica 2: Struktura tablice korisnika

Vanjski ključevi:

- **account_id** (`uuid`) - ID povezanog računa

Quiz

Definira kviz u sustavu. Ova shema sadrži informacije o kvizu, uključujući naziv, opis i vrijeme trajanja. Također sadrži informacije o aktivnim pitanjima odnosno koje je pitanje aktivno i kada je započelo i koliko traje. Kontekstualno ova shema je jezgra aplikacije jer povezuje direktno i indirektno sve ostale sheme u glavnu logiku aplikacije. Prije spremanja u bazu za svaki quiz generira se jedinstveni kod za pridruživanje na kviz. Direktno je povezana s shemom korisnika koji upravlja kvizom i trenutnog pitanja na kvizu.

Atributi	Tip podatka	Opis
id	uuid	Jedinstveni identifikator kviza
title	string	Naziv kviza
description	text	Opis kviza
join_code	string	Kod za pridruživanje kvizu
question_time_limit	integer	Vremensko ograničenje za pitanje
question_started_at	utc_datetime	Vrijeme početka trenutnog pitanja
inserted_at	utc_datetime	Vremenska oznaka kreiranja kviza
updated_at	utc_datetime	Vremenska oznaka izmjene kviza

Tablica 3: Struktura tablice kviza

Vanjski ključevi:

- **user_id** (`uuid`) - ID korisnika koji je kreirao kviz
- **current_question_id** (`uuid`, može biti `null`) - ID trenutnog pitanja

Funkcije sheme:

- `put_join_code/1` - automatski generira kod za pridruživanje ako nije zadan.
- `generate_unique_code/0` - generira jedinstveni kod za pridruživanje kvizu.

Ograničenja:

- **title**, **description**, **user_id** su obavezni.
- **join_code** mora biti jedinstven.

Question

Ovdje je definirana shema za koju će veliki jezični model generirati pitanja. Ona se koristi za pohranu informacija o pitanjima u kvizu, uključujući tekst pitanja, moguće odgovore i točan odgovor. Također, koristi se i zastavica za označavanje je li pitanje potrošeno. Svako pitanje pridruženo je određenom kvizu putem vanjskog ključa.

Vanjski ključevi:

- **quiz_id** (`uuid`) - ID kviza kojem pitanje pripada

Atributi	Tip podatka	Opis
id	uuid	Jedinstveni identifikator pitanja
text	text	Tekst pitanja
options	array[string]	Mogući odgovori
answer	string	Točan odgovor
used	boolean	Je li pitanje iskorišteno
inserted_at	naive_datetime	Vremenska oznaka kreiranja pitanja
updated_at	naive_datetime	Vremenska oznaka izmjene pitanja

Tablica 4: Struktura tablice pitanja

Answer

Ova shema reprezentira odgovor igrača na pitanje u kvizu. Sastoji se od teksta odgovora i zastavice koja označava je li odgovor točan ili netočan. Svaki odgovor pridružen je igraču i pitanju putem vanjskih ključeva.

Atributi	Tip podatka	Opis
id	uuid	Jedinstveni identifikator odgovora
text	string	Tekst odgovora
is_correct	boolean	Je li odgovor točan
inserted_at	naive_datetime	Vremenska oznaka kreiranja odgovora
updated_at	naive_datetime	Vremenska oznaka izmjene odgovora

Tablica 5: Struktura tablice odgovora

Vanjski ključevi:

- **question_id** (`uuid`) - ID pitanja na koje se odnosi odgovor
- **player_id** (`uuid`) - ID igrača koji je dao odgovor

Result

Definira rezultat kviza za igrača. Ova shema bilježi ukupni rezultat igrača u određenom kvizu.

Atributi	Tip podatka	Opis
id	uuid	Jedinstveni identifikator rezultata
score	integer	Broj bodova
inserted_at	naive_datetime	Vremenska oznaka kreiranja rezultata
updated_at	naive_datetime	Vremenska oznaka izmjene rezultata

Tablica 6: Struktura tablice rezultata

Vanjski ključevi:

- **player_id** (`uuid`) - ID igrača

Player

Definira sudionika kviza. Osim korisnika putem ove sheme možemo imati i goste koji se mogu pridružiti kvizu bez registracije. Stoga ova shema definira sudionika kviza, ne isključivo korisnika. Jedan sesijski kod gosta i jedan korisnik mogu se pridružiti kvizu samo jedanput.

Atributi	Tip podatka	Opis
id	uuid	Jedinstveni identifikator igrača
name	string	Ime igrača
session_id	string (null)	Identifikator sesije kviza
inserted_at	utc_datetime	Vremenska oznaka kreiranja igrača
updated_at	utc_datetime	Vremenska oznaka izmjene igrača

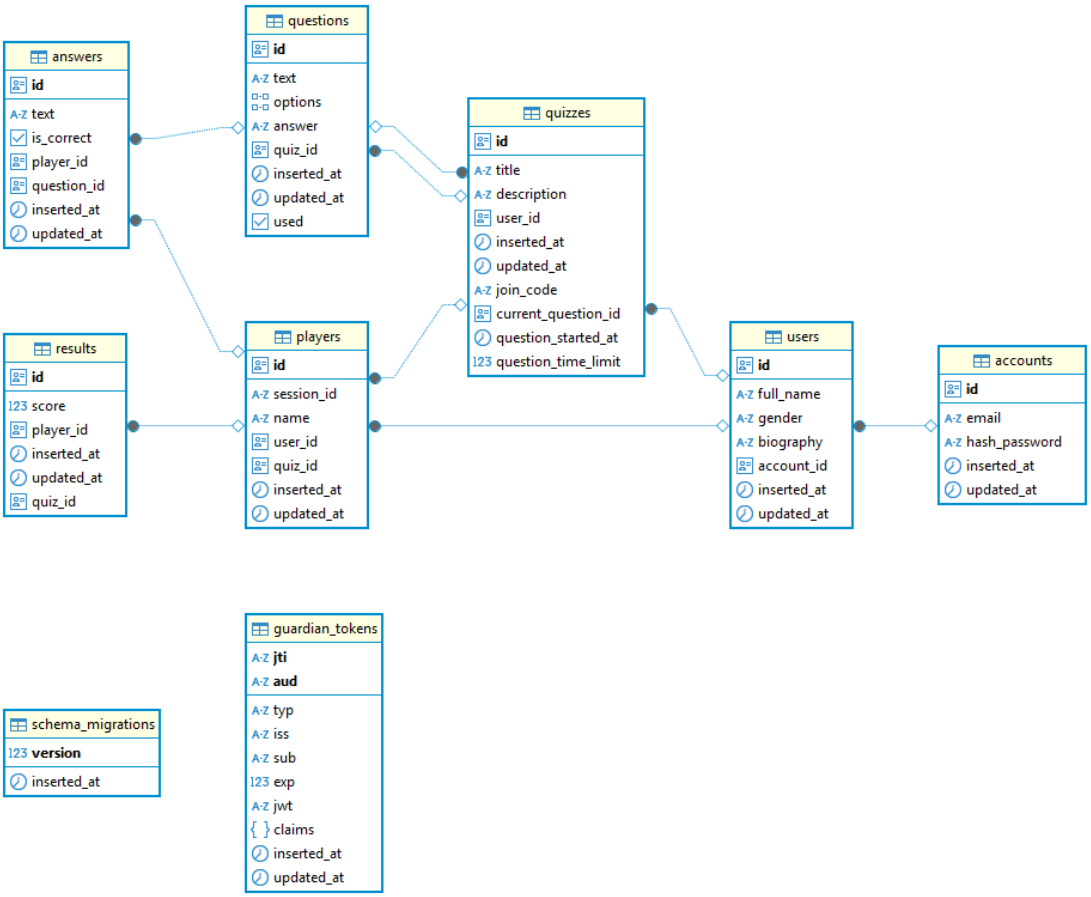
Tablica 7: Struktura tablice igrača

Vanjski ključevi:

- **user_id** (`uuid`, može biti `null`) - ID korisnika
- **quiz_id** (`uuid`) - ID kviza kojem igrač pripada

Ograničenja:

- **name**, **quiz_id** su obavezni.
- Jedna sesija može imati samo jednog igrača po kvizu.
- Jedan user može imati više igrača, ali samo jedan igrač može biti povezan s kvizom u isto vrijeme.



Slika 4: ER dijagram sustava

5.2 Postavljanje Phoenix Poslužitelja

Sustav koristi Phoenix Framework za razvoj poslužitelja. Za započeti projekt potrebno je instalirati, prije svega, Erlang i elixir, te preko hex package managera preuzeti phoenix. Projekt se može započeti naredbom mix phx.new, te uz postavke koje isključuju "LiveView" kao što su –no-live, dobit ćemo phoenix samo sa REST sučeljom. Projekt je postavljen kao Model-View-Controller (MVC) arhitektura, gdje su modeli definirani u Ecto shemama, a kontroleri upravljaju HTTP zahtjevima i odgovarajućim akcijama. Također, generator će automatski stvoriti potrebne direktorije i datoteke za početak rada, kao i odgovarajuće direktorije gdje će se nalaziti testovi. Phoenix je praktičan jer omogućuje razne generatore za brzo postavljanje strukture aplikacije, uključujući generatore za sheme, kontrolere i migracije baze podataka.

Naredbom `phx.gen.json` možemo generirati kontroler koji će generirati kontroler i potrebne rute za CRUD operacije nad modelom, a na nama je samo da ga dodamo u ruter. U direktoriju `lib/quizaar` ćemo pronaći generirane modele

i poslovnu logiku, dok će se u `lib/quizaar_web` nalaziti datoteke koje će služiti kao API sučelje. Socket se generira naredbom `mix phx.gen.socket`.

Poslužitelj u ovom projektu je podijeljen u dva dijela: WebSocket i REST API. Iako je moguće napraviti cijeli sustav kao WebSocket, funkcionalnosti koje ne trebaju biti u stvarnom vremenu su implementirane kao REST API radi jednostavnosti. WebSocket je u ovom slučaju iskorišten za upravljanje kvizevima, dok je REST api korišten za upravljanje korisnicima i početno kreiranje kanala na koje se korisnici mogu spajati.

5.2.1 REST API

Ruter

Glavna konfiguracija rutera nalazi se u datoteci `lib/quizaar_web/router.ex`. U ovom modulu definira se način na koji aplikacija obrađuje HTTP zahtjeve i greške, te se organiziraju rute i middleware. Funkcija `handle_errors/2` omogućuje vraćanje strukturiranih JSON odgovora u slučaju grešaka, uključujući nepostojeće rute ili neautorizirane zahtjeve.

Ruter definira dvije pipeline sekcije: `:api` za osnovne API zahtjeve (prihvaća JSON unos i dohvaća sesiju kao kolačić) te `:auth` za autentikaciju i postavljanje korisničkog računa. Rute su organizirane u dva scope-a: jedan za javne API rute, drugi za rute koje zahtijevaju autentikaciju. Na taj način se osigurava da su osjetljive operacije dostupne samo korisnicima sa tokenom, dok su osnovne informacije dostupne svima.

Autentikacija

REST API koristi JWT (JSON Web Tokens) za autentikaciju i autorizaciju korisnika. To je postignuto kroz Guardian i GuardianDB biblioteke koje omogućuju jednostavno upravljanje tokenima i zaštitu ruta. Guardian nam nudi mogućnost dekodiranja tokena i provjeru ispravnosti tokena, te omogućuje jednostavno upravljanje korisničkim tokenima koje, nakon izdavanja, sprema u bazu podataka. Izdane tokene je moguće opozivati, a istekli tokeni se automatski brišu. U `pipeline.ex` datoteci je definiran middleware, koji u Phoenix-u nazivamo pipeline, koji kasnije možemo postaviti na rute koje želimo zaštititi. Također, definiran je i modul `set_account.ex` koji će uz token poslati i podatke o računu u obliku kolačića (cookies) kako bi se olakšalo autoriziranje korisnika. Njega ćemo također postaviti na rute kojima je potrebna autorizacija.

Autorizacija

Autorizacija se provodi na razini kontrolera gdje će middleware provjeriti je li korisnik onaj koji ima pravo mijenjati određeni resurs. To će učiniti provjerom podataka svojeg kolačića u usporedbi s podacima koji se nalaze u tokenu.

Rješavanje grešaka autentikacije i autorizacije

U slučaju neuspjeha autentikacije ili autorizacije, sustav će vratiti odgovarajuću HTTP grešku (401 Unauthorized ili 403 Forbidden) s detaljnim opisom greške. To omogućuje klijentima da lako obrade greške i pruže korisnicima jasne informacije o problemima.

Javne rute

- `POST /api/accounts/create` - registracija korisnika
- `POST /api/sign_in` - prijava korisnika

Privatne rute

- `GET /api/accounts/current` - dohvat podataka o računu koji je trenutno prijavljen
- `GET /api/accounts/by_id/:id` - dohvat računa prema ID-u
- `PATCH /api/accounts/update` - ažuriranje podataka računa
- `PATCH /api/users/update` - ažuriranje podataka korisnika
- `POST /api/accounts/refresh_session` - osvježavanje tokena
- `POST /api/quizzes/create` - kreiranje novog kviza
- `POST /api/quizzes/:quiz_id/generate_questions` - generiranje pitanja za kviz pomoću LLaMA 4
- `GET /api/quizzes/:join_code` - dohvat kviza i pitanja prema kodu za pridruživanje
- `GET /api/quizzes/:quiz_id/questions` - dohvat svih pitanja za kviz
- `PATCH /api/questions/:id` - ažuriranje pitanja
- `POST /api/questions/create` - kreiranje novog pitanja
- `DELETE /api/questions/:id` - brisanje pitanja
- `GET /api/quizzes/list/user` - dohvat svih kvizova koje je korisnik kreirao

5.2.2 Generiranje pitanja

Generiranje pitanja za kviz provodi se putem `POST /api/quizzes/:quiz_id/generate_questions` rute. Ova ruta poseže za funkcijom `generate_questions/2` u `Quizaar.Quizzes` modulu. Funkcija koristi LLaMA 4 model za generiranje pitanja na temelju teme, težine i dodatnog konteksta. Groq API vraća isključivo JSON odgovor što je definirano u promptu koji se šalje modelu. Prompt je složen tako da jasno definira strukturu pitanja i odgovora

koje model treba generirati. U slučaju da model ne može generirati pitanja, on će pokušati ponovo generirati pitanja do petog pokušaja. U slučaju da ne uspije, znaći da nešto nije uredu sa zahtjevom prema groq serveru i vratiti će grešku 500 Internal Server Error.

5.2.3 WebSocket

WebSocket je korišten za upravljanje kvizovima u realnom vremenu. Prije spajanja na kanale korisnik mora biti spojen na `QuizaarWeb.UserSocket`.

U `lib/quizaar_web/channels/user_socket.ex` datoteci definirani su kanali koje socket koristi, a autentikacija se provodi unutar kanala.

Kanali

Ova aplikacija koristi kanal definiran u `lib/quizaar_web/channels/quiz_channel.ex` datoteci. Svaki kviz ima svoj kanal koji se koristi za komunikaciju između klijenta i poslužitelja. Korisnici se mogu pridružiti kanalu kviza koristeći kod za pridruživanje koji je generiran prilikom kreiranja kviza.

Pridruživanje kvizu

Pridruživanje kvizu omogućuje korisnicima (registriranima i gostima) da sudjeluju u kvizu putem Phoenix kanala. Svaki kviz ima svoj kanal, a korisnici se pridružuju slanjem "join" poruke na kanal s odgovarajućim kodom za pridruživanje.

Prilikom pridruživanja, sustav provjerava postoji li kviz s danim kodom. Ako kviz postoji, korisniku se dodjeljuju potrebni podaci u socket (npr. uloga, ime, session ID). Registrirani korisnici se autentificiraju putem JWT tokena, dok se gostima generira jedinstveni session ID. Nakon uspješnog pridruživanja, korisnik postaje "player" ili "organizer" (ovisno o ulozi), a u slučaju da je broj igrača dosegao maksimum, korisnik postaje "spectator".

Sustav prati prisutnost na kanalu i status svakog korisnika putem modula prisutnosti, a informacije o igračima i njihovoj spremnosti dostupne su organizatoru kviza. Pridruživanje je temelj za daljnju interakciju s kvizom, uključujući odgovaranje na pitanja, pregled rezultata i komunikaciju u stvarnom vremenu.

Prisutnost

Modul prisutnosti omogućuje praćenje korisnika koji su trenutno prisutni u kanalu kviza. Kada se korisnik pridruži kanalu, njegov status se ažurira na "online", a kada napusti kanal, status se ažurira na "offline". Organizator kviza može vidjeti popis svih prisutnih igrača, kao i njihov trenutni status (spreman/nije spreman).

Eventi i funkcionalnosti kanala

Komunikacija s kanalom kviza odvija se putem različitih događaja (eventa) koji su definirani u `lib/quizaar_web/channels/quiz_channel.ex` datoteci. Ovi događaji omogućuju

različite funkcionalnosti unutar kviza, kao što su pridruživanje kvizu, slanje pitanja, odgovaranje na pitanja i pregled rezultata.

Event	Opis funkcionalnosti
`join`	Pridruživanje kvizu putem koda; dodjeljuje ulogu (player/organizer/spectator) i session ID.
`presence_state`	Dohvat trenutnog stanja prisutnosti svih igrača u kanalu.
`ready_up`	Označavanje igrača kao spremnog za početak kviza.
`unready`	Označavanje igrača kao nespremnog.
`quiz_start`	Pokretanje kviza od strane organizatora kada su svi igrači spremni.
`serve_question`	Slanje novog pitanja svim igračima; započinje mjerenje vremena za odgovor i postavlja unutarnji timer.
`answer_question`	Slanje odgovora na trenutno pitanje od strane igrača.
`question_closed`	Obavijest o isteku vremena za odgovor; zatvara pitanje za daljnje odgovore.
`get_players`	Dohvat popisa svih igrača u kvizu.
`players_stats`	Dohvat statistike svih igrača (bodovi, status).
`player_stats`	Dohvat statistike pojedinog igrača (bodovi, rang).
`generate_questions`	Generiranje novih pitanja za kviz (organizer).
`get_current_question`	Dohvat trenutnog aktivnog pitanja i preostalog vremena.
`get_all_answers_to_current`	Dohvat svih odgovora igrača na trenutno pitanje.
`fix_answer`	Ispravak bodovanja odgovora (organizer).
`delete_player`	Brisanje igrača iz kviza (organizer).

Tablica 8: Eventi i funkcionalnosti Quiz kanala

5.2.4 Testiranje poslužitelja

Testiranje poslužitelja provedeno je korištenjem ugrađenih alata programskog okruženja Elixir/Phoenix, pri čemu su testovi organizirani u zasebne module unutar direktorija `test`. Testovi obuhvaćaju ključne komponente sustava, uključujući API rute, WebSocket komunikaciju te Ecto sheme. Za potrebe generiranja testnih podataka korištene su biblioteke `Faker` i `ExMachina`, koje omogućuju jednostavno kreiranje realističnih podataka u testnom okruženju.

Za svaku funkcionalnost razvijeni su pripadajući testovi. Primjerice, datoteke `accounts_test.exs`, `users_test.exs` i `quizzes_test.exs` provjeravaju osnovne operacije nad korisnicima, kvizovima i korisničkim računima. Testovi smješteni unutar direktorija `schema_test.exs` ispituju ispravnost Ecto shema, definirana ograničenja te odnose među podacima. Poseban naglasak stavljen je na provjeru komunikacije putem WebSocket kanala, što je realizirano u datoteci `channels/quiz_channel_test.exs`, a koja testira postupke pridruživanja kvizu,

slanja pitanja i odgovora te praćenja prisutnosti korisnika. Nadalje, direktorij `support` sadrži pomoćne module i podatke koji omogućuju učinkovitije i brže izvođenje testova.

5.3 Razvoj korisničkog sučelja

Razvoj korisničkog sučelja provodi se inicijalizacijom Vue 3 projekta, koji incijaliziran, u ovom slučaju pomoću Vite alata. U ovom projektu koristi se Vue 3 s Pinia za upravljanje stanjem aplikacije i shadcn za stilizaciju sučelja.

5.3.1 Komponente autentikacije

Komponente autentikacije uključuju prijavu i registraciju. Svaka komponenta je odgovorna za određeni dio autentikacijskog procesa i koristi Pinia store za pohranu i upravljanje stanjem korisničkog računa.

account.js

Datoteka `account.js` sadrži logiku za upravljanje korisničkim računima, uključujući funkcije za prijavu, registraciju i odjavu. Ove funkcije koriste Pinia za pohranu korisničkih podataka i upravljanje sesijama. Te funkcije pozivaju funkcije definirane u `api.js` datoteci koja sadrži funkcije za slanje HTTP zahtjeva prema REST API-ju vezane uz korisničke račune. Primjer funkcije za prijavu korisnika:

Listing 1: Funkcija za registraciju korisnika

```
async registerAccount(full_name, email, password) {
  let userDetail = {
    full_name: full_name,
    email: email,
    hash_password: password
  };
  try {
    let user = await auth.register(userDetail);
    this.accountData =  {
      email: user.email,
      id: user.id,
      token: user.token,
    };
    return user;
  } catch (error) {
    throw error;
  }
}
```

Register.vue

Komponenta `Register.vue` prikazuje formu s poljima za unos korisničkog imena, email adrese i lozinke. Povezana je s `accountStore` koji sadrži poslovnu logiku

za upravljanje korisničkim računima. Na klik gumba "Create an account" poziva se metoda `registerAccount`, koja koristi funkciju iz `accountStore` modula za registraciju korisnika . U slučaju uspješne registracije, korisnik se preusmjerava na stranicu za prijavu (`/login`), dok se u slučaju neuspjeha (npr. ako korisnik već postoji) prikazuje poruka o grešci putem `toast.error`. Komponenta koristi shadcn UI komponente za stilizaciju forme, uključujući `Card`, `Input`, `Label` i `Button`.

Login.vue

Komponenta `Login.vue` prikazuje formu s poljima za unos email adrese i lozinke. Povezana je s `accountStore` koji sadrži poslovnu logiku za autentikaciju korisnika. Na klik gumba "Login" poziva se metoda `loginAccount`, koja koristi funkciju iz `accountStore` modula za prijavu korisnika. U slučaju uspješne prijave, korisnik se preusmjerava na početnu stranicu (`/`), dok se u slučaju neuspjeha (npr. ako su podaci netočni) prikazuje poruka o grešci putem `toast.error`. Komponenta koristi shadcn UI komponente za stilizaciju forme, uključujući `Card`, `Input`, `Label` i `Button`, te omogućuje korisniku pristup opciji za resetiranje lozinke.

5.3.2 Komponente kviza

Komponente kviza omogućuju korisnicima interakciju s kvizovima, uključujući kreiranje, pregled, sudjelovanje u kvizovima, odgovaranje i upravljanje pitanjima. Ove komponente koriste Pinia za pohranu i upravljanje stanjem vezanim uz kvizove, pitanja i rezultatima kvizova.

quiz.js

Datoteka `quiz.js` definira Pinia store koji služi za upravljanje kvizovima i pitanjima unutar aplikacije. Store sadrži stanje aplikacije, poput popisa kvizova i trenutno aktivnog kviza, te metode za kreiranje, dohvat, ažuriranje i brisanje kvizova i pitanja.

Glavne funkcionalnosti uključuju kreiranje novog kviza i automatsko generiranje pitanja putem API poziva, pridruživanje kvizu pomoću koda, dohvat podataka o kvizu i pitanjima, dodavanje i brisanje pitanja, te ažuriranje postojećih pitanja. Store također omogućuje praćenje završetka kviza putem WebSocket kanala i prikaz obavijesti korisniku.

Za komunikaciju s poslužiteljem koristi se `quizApi`, dok se za navigaciju i prikaz obavijesti koriste `router` i `toast`.

Listing 2: Funkcija za kreaciju kviza

```
async createQuiz(quizData ,parameters) {
      try {
          // Simulate an API call to create a new quiz
        const response = await quizApi.createQuiz(quizData
           );
          const newQuiz = response.data;
          this.quizzes.push(newQuiz);
```

```
            this.activeQuiz = newQuiz;
            const questions = await quizApi.
               generateQuestions(newQuiz.data.id, {
            topic: parameters.topic,
            number: parameters.questions,
            difficulty: parameters.difficulty,
            description: parameters.context
          });
            newQuiz.questions = questions.data;
             router.push("/quiz/active/" +newQuiz.data.
                join_code);

            return newQuiz;
        } catch (error) {

            throw error;
        }
        },
```

score.js

Datoteka `score.js` definira Pinia store koji služi za upravljanje rezultatima i bodovima igrača u kvizu. Store sadrži stanje aplikacije, uključujući popis svih rezultata igrača, trenutni rezultat prijavljenog igrača, podatke o plasmanu te informacije o igračima s višim i nižim bodovima.

Metoda `getScore` dohvaća podatke o trenutnom igraču, njegovom plasmanu te bodovima igrača koji su bolje ili lošije rangirani. Metoda `trackResults` dohvaća rezultate svih igrača u kvizu i ažurira stanje aplikacije.

Za komunikaciju s poslužiteljem koristi se kanal koji šalje odgovarajuće evente (`player_stats` i `players_stats`) te prima podatke o rezultatima.

question.js

Datoteka `question.js` definira Pinia store koji služi za upravljanje aktivnim pitanjem u kvizu. Store sadrži stanje aplikacije, uključujući trenutno aktivno pitanje i preostalo vrijeme za odgovor.

Metoda `getCurrentQuestion` dohvaća podatke o aktivnom pitanju i vremenu za odgovor, dok metode `ActiveQuestion` i `QuestionServed` slušaju događaje s kanala i ažuriraju stanje aplikacije kada se pojavi novo pitanje. Metoda `ServeQuestion` omogućuje organizatoru slanje novog pitanja svim igračima.

Ova struktura omogućuje korisnicima praćenje aktivnog pitanja, preostalog vremena i interakciju s kvizom u stvarnom vremenu.

Listing 3: Funkcija za obradu događaja question_served

```
   async QuestionServed(channel, onServed) {
  channel.on('question_served', (response) => {
    this.currentQuestion = response.question.data;
    this.timeLeft = response.time_left || 0;
```

```
    if (onServed) onServed();
  });
   },
```

socket.js

Datoteka `socket.js` definira Pinia store koji služi za upravljanje WebSocket vezom između klijenta i Phoenix poslužitelja. Store sadrži stanje aplikacije, uključujući socket objekt, kanal za komunikaciju i modul za praćenje prisutnosti korisnika.

Store omogućuje uspostavu veze s poslužiteljem putem metode `connect`, koja koristi JWT token za autentikaciju. Metoda `joinChannel` omogućuje pridruživanje određenom kanalu i inicijalizira praćenje prisutnosti korisnika putem Phoenix Presence modula. Metode `leaveChannel` i `disconnect` omogućuju napuštanje kanala i prekid WebSocket veze.

ActiveQuiz.vue

Komponenta `ActiveQuiz.vue` predstavlja sučelje za upravljanje aktivnim kvizom i igračima tijekom kviza. Organizatoru omogućuje pregled i uređivanje pitanja, upravljanje popisom igrača, praćenje spremnosti sudionika te pokretanje kviza. Naslov kviza i kod za pridruživanje prikazuju se na vrhu sučelja, uz mogućnost kopiranja koda u međuspremnik. Popis pitanja prikazuje se i uređuje putem podkomponente `QuizQuestions`, a organizator može dodavati, ažurirati i brisati pitanja koristeći dijalog prozore iz komponente `UpdateQuestionDialog`. Popis aktivnih igrača i njihov status spremnosti prikazuju se u stvarnom vremenu, a organizator ima mogućnost uklanjanja igrača iz kviza. Prisutnost i spremnost igrača prate se pomoću WebSocket kanala i Phoenix Presence modula, dok se obavijesti o događajima, poput pridruživanja igrača, spremnosti ili grešaka, prikazuju putem `toast` poruka. Kada su svi igrači spremni, kviz se može pokrenuti, a korisnici se automatski preusmjeravaju na stranicu s trenutnim pitanjem. Komponenta koristi shadcn UI komponente (`Button`, `Input`, `Label`, `Dialog`, `Skeleton`) te ikone iz biblioteke `lucide-vue-next` za stilizaciju sučelja.

FinalResultView.vue

Komponenta `FinalResultView.vue` prikazuje završne rezultate kviza svim sudionicima. Nakon završetka kviza, komponenta dohvaća rezultate svih igrača putem WebSocket kanala i Pinia store-a za bodove (`scoreStore`). Rezultati se prikazuju u tabličnom obliku pomoću podkomponente `StatsTable`, koja omogućuje pregled bodova i poretka svih igrača.

Pri učitavanju, komponenta provjerava postoji li aktivni kanal; ako nije, automatski se pridružuje kanalu kviza koristeći kod za pridruživanje i podatke o korisniku. Nakon toga, poziva se metoda za dohvat rezultata (`trackResults`) koja ažurira stanje s rezultatima svih igrača.

Ova funkcionalnost osigurava transparentan prikaz završnih rezultata i omogućuje svim sudionicima uvid u svoj plasman nakon završetka kviza.

JoinQuiz.vue

Komponenta `JoinQuiz.vue` omogućuje korisnicima jednostavno pridruživanje postojećem kvizu. Prikazuje formu s poljima za unos korisničkog imena i koda za pridruživanje kvizu. Nakon što korisnik unese potrebne podatke i potvrdi formu, komponenta preusmjerava korisnika u čekaonicu kviza (`/quiz/lobby/`) te prenosi uneseno ime kao parametar.

Za stilizaciju sučelja koristi shadcn UI komponente (`Card`, `Input`, `Label`, `Button`).

NewQuiz.vue

Komponenta `NewQuiz.vue` omogućuje korisniku kreiranje novog kviza kroz višekorak (stepper) sučelje. Proces kreiranja kviza podijeljen je u dvije glavne faze: unos osnovnih informacija o kvizu (naslov i opis) te postavljanje parametara kviza (npr. vrijeme za rješavanje, broj pitanja, težina i kontekst).

Komponenta koristi "stepper" za prikaz napretka kroz korake, a svaki korak je zasebna podkomponenta (`GeneralQuizForm` i `QuizParameters`). Nakon unosa svih potrebnih informacija, korisnik može pokrenuti kreiranje kviza, pri čemu se poziva metoda iz Pinia store-a (`quizStore.createQuiz`) koja šalje podatke na backend i generira kviz s pripadajućim pitanjima.

Za stilizaciju i interaktivnost koristi shadcn UI komponente (`Stepper`, `Button`, `Input`, `Label`, `Textarea`).

OrgQuestionView.vue

Komponenta `OrgQuestionView.vue` omogućuje organizatoru kviza upravljanje trenutnim pitanjem, pregled odgovora igrača i statistike, te prelazak na sljedeće pitanje tijekom kviza. Sučelje je podijeljeno na dva dijela: prikaz aktivnog pitanja s preostalim vremenom za odgovor, te prikaz odgovora i statistike igrača putem tabova. Organizator može pregledati sve odgovore igrača na trenutno pitanje, ručno ispraviti bodovanje odgovora te pratiti statistiku rezultata u stvarnom vremenu. Komponenta koristi podkomponente `CurrentQuestion` za prikaz aktivnog pitanja i preostalog vremena, `UserAnswers` za prikaz odgovora igrača i mogućnost ispravka bodovanja, te `StatsTable` za prikaz statistike rezultata igrača. Organizacija prikaza odgovora i statistike ostvaruje se pomoću tabova, dok se za prelazak na sljedeće pitanje koristi gumb. Upravljanje stanjem kviza, pitanjima, odgovorima i rezultatima ostvaruje se putem Pinia store-ova (`useSocketStore`, `useQuestionStore`, `useScoreStore`, `useQuizStore`), a obavijesti korisniku prikazuju se pomoću `toast` poruka.

UserQuizListView.vue

Komponenta `UserQuizListView.vue` omogućuje korisniku pregled svih kvizova koje je kreirao. Prikazuje kvizove u obliku kartica s osnovnim informacijama, uključujući naslov, opis, datum kreiranja, status kviza (započet ili ne), te kod za pridruživanje. Navigacija kroz kvizove omogućena je putem paginacije, a korisnik može jednostavno otvoriti detalje kviza ili nastaviti s organizacijom kviza klikom

na odgovarajući gumb. Za prikaz i stilizaciju koristi shadcn UI komponente kao što su `Card`, `Button` i `Pagination`.

QuizView.vue

Komponenta `QuizView.vue` predstavlja početnu stranicu kviz modula, gdje korisnik može odabrati želi li se pridružiti postojećem kvizu ili kreirati novi. Sučelje je podijeljeno na dvije kartice: jedna za unos koda i imena radi pridruživanja kvizu, a druga za kreiranje novog kviza. Komponenta omogućuje jednostavnu navigaciju prema lobiju kviza ili formi za kreiranje kviza. Za prikaz koristi shadcn UI komponente (`Card`, `Button`, `Input`) i ikone iz biblioteke `lucide-vue-next`.

QuizLobby.vue

Komponenta `QuizLobby.vue` prikazuje lobij kviza u kojem korisnik čeka početak kviza. Prikazuje informacije o korisniku (ime i session ID) te kod za pridruživanje kvizu. Korisnik može označiti da je spreman za početak kviza klikom na gumb "Ready Up", a status spremnosti se prenosi organizatoru putem WebSocket kanala. Komponenta koristi shadcn UI komponente (`Card`, `Button`) za prikaz i interakciju, te omogućuje jednostavno upravljanje statusom sudionika prije početka kviza.

5.3.3 Promptovi za generiranje pitanja i evaluaciju odgovora

Za automatsko generiranje i evaluaciju kviz pitanja koriste se posebno dizajnirani promptovi koji se šalju LLaMA 4 modelu putem Groq API-ja. Ovi promptovi definiraju pravila i strukturu odgovora koje model mora poštovati. Promptovi su ključni za osiguranje da generirana pitanja budu relevantna, jasna i strukturirana na način koji omogućuje jednostavno korištenje u kvizu. Također, ovi promptovi prate upute koje se nalaze upute koje se nalaze u Meta Llama dokumentaciji [7], a prilagođeni su specifičnim potrebama ovog sustava.

Prompt1.txt koristi se za generiranje pitanja i odgovora za kviz. U promptu se jasno navodi da model mora odgovoriti isključivo u JSON formatu, bez dodatnih znakova za novi red. Od modela se traži da generira točno onoliko pitanja koliko je korisnik zatražio, da pitanja budu vezana uz zadanu temu i razinu težine, te da se poštuju pravila o tipu pitanja. Svako pitanje mora imati jednoznačan odgovor, a kod pitanja s izborom točan odgovor mora biti među ponuđenim opcijama. Prompt također zabranjuje generiranje nejasnih, ponavljajućih ili subjektivnih pitanja, te zahtijeva da sve opcije budu relevantne i različite.

Prompt2.txt koristi se za evaluaciju korisničkih odgovora na pitanja kviza. Ovaj prompt od modela traži da odgovori isključivo u JSON formatu s poljem `corrected_answer`, koje sadrži korisnički odgovor i informaciju je li odgovor točan. Model tolerira do 20% tipografskih pogrešaka i prihvaća odgovore koji su kontekstualno točni, čak i ako sadrže manje greške u pisanju ili formulaciji. Na taj način se omogućuje objektivna i fleksibilna evaluacija odgovora, čime se smanjuje broj nepravedno označenih netočnih odgovora.

Korištenjem ovih promptova sustav osigurava da generirana pitanja budu kvalitetna, relevantna i jasno strukturirana, dok se evaluacija odgovora provodi precizno i prilagođeno stvarnim korisničkim pogreškama.

5.3.4 Docker

Docker je open source platforma koja omogućuje jednostavno i efikasno pakiranje, distribuciju te pokretanje softvera koristeći kontejnere. Kontejneri su izolirana okruženja, slična virtualnim mašinama, ali koriste jezgru operativnog sustava hosta umjesto da pokreću cijeli operativni sustav, pa su lakši i brži za pokretanje. Služe kako bi izdvojili aplikaciju i sve njene ovisnosti u jedan paket koji se može lako premjestiti i pokrenuti na bilo kojem sustavu koji podržava Docker [8]. Ovdje će se primjeniti da bi olakšao isporučivanje aplikacije na hosting server.

5.3.5 Kontejnerizacija poslužitelja

Dockerfile za poslužitelja generira se naredbom `mix phx.gen.release --docker`. Taj Dockerfile definira kako se aplikacija gradi i pokreće unutar kontejnera, uključujući sve potrebne ovisnosti i konfiguracije.

Datoteka `docker-compose.yml` koristi se za podizanje svih potrebnih servisa, uključujući bazu podataka i poslužitelj, te omogućuje jednostavno upravljanje njima unutar kontejnera. Također, definira vanjske port-ove, povezuje servise unutar mreže, migrira baze podataka i započinje aplikaciju.

5.3.6 Kontejnerizacija korisničkog sučelja

Za frontend aplikaciju koristi se Dockerfile koji omogućuje jednostavno postavljanje i pokretanje Vue.js sučelja u produkcijskom okruženju. Dockerfile je podijeljen u dvije faze: build fazu i produkcijsku fazu.

U prvoj fazi koristi se službena Node.js slika za instalaciju ovisnosti i izgradnju aplikacije. Nakon što se aplikacija izgradi, u drugoj fazi koristi se Nginx slika za posluživanje statičkih datoteka. Izgrađeni frontend kopira se u direktorij za statičke datoteke Nginx-a, a konfiguracija servera definira se putem `nginx.conf` datoteke. Na taj način, aplikacija je dostupna na portu 80 i spremna za produkcijsko korištenje.

5.3.7 GitHub Actions

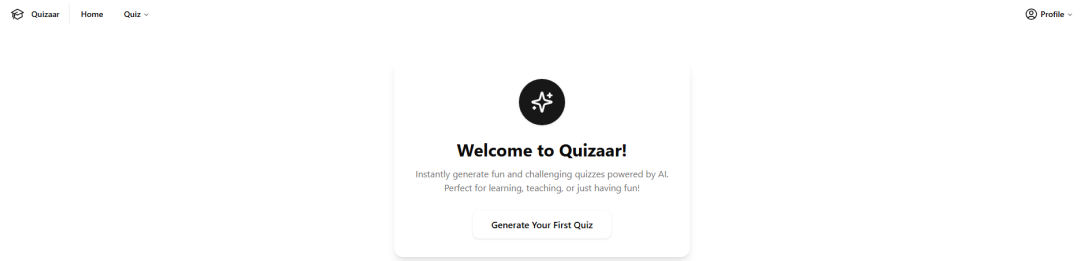
Github Actions je platforma za kontinuiranu integraciju i isporuku (CI/CD) koja omogućuje automatizaciju radnih tokova unutar GitHub repozitorija. Takvi radni tokovi mogu služiti za automatizaciju različitih zadataka, uključujući izgradnju, testiranje i implementaciju aplikacija [5]. U ovom projektu definirana su 3 workflow-a:

- CI workflow na backendu -"test"- koji služi kako bi testirao aplikaciju pri svakom push-u na glavni repozitorij

- CD workflow na backendu -"deploy"- koji automatski implementira aplikaciju na produkcijski server nakon uspješnog testiranja
- CI workflow na frontendu -"deploy"- koji automatski implementira frontend aplikaciju na produkcijski server nakon svakog push-a na glavni repozitorij

6 Funkcionalnosti sustava

U ovoj sekciji opisane su ključne funkcionalnosti sustava, uključujući upravljanje korisnicima, kvizovima i komunikaciju u pravom vremenu putem WebSocket veze. Uz pomoć shadcn biblioteke kreirano je atraktivno sučelje koje korisnicima omogućava jednostavnije kreiranje, generiranje i upravljanje kvizovima, ali i intuitivno iskustvo pri igranju samog kviza. U nastavku su opisane glavne funkcionalnosti sustava.



Slika 5: Početna stranica

6.1 Upute za korištenje

6.1.1 Prijava i registracija

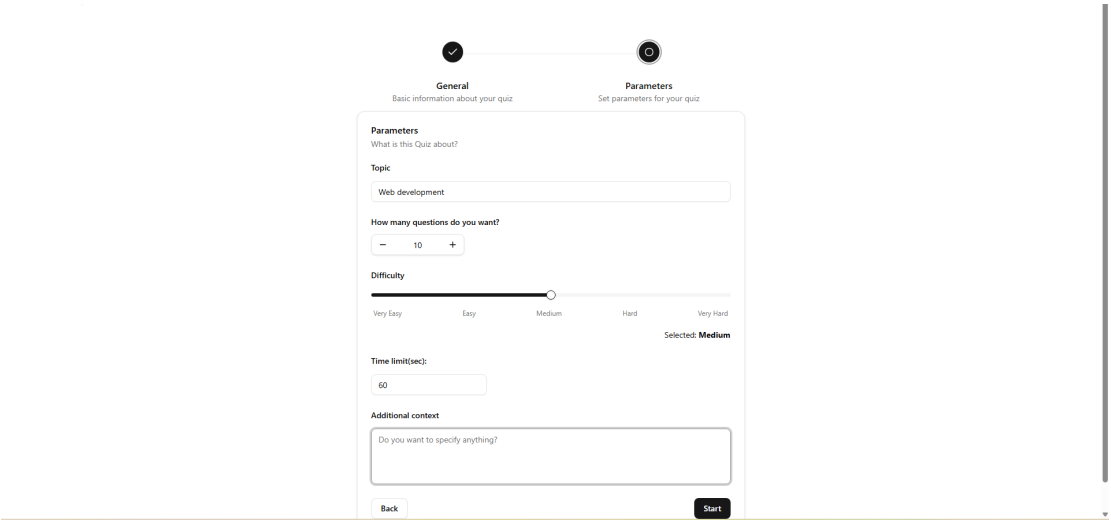
Korisnici se mogu registrirati putem obrasca za registraciju, gdje unose svoje ime, email i lozinku. Nakon uspješne registracije, korisnici se mogu prijaviti koristeći iste podatke. Te dobijaju svoj kolačić i JWT token koji se koristi za autentifikaciju prilikom slanja zahtjeva prema API-ju.

6.1.2 Upravljanje korisničkim računima

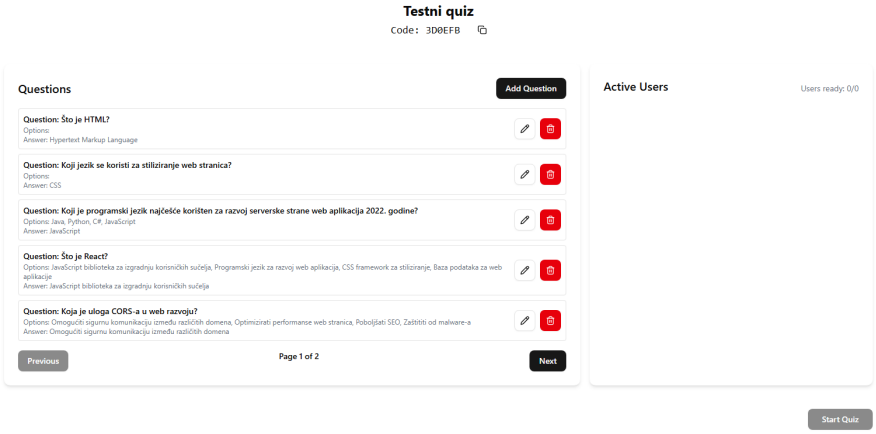
Korisnici mogu ažurirati svoje podatke, klikom na "settings" gumb u glavnom izborniku, uključujući ime, email i lozinku, putem obrazaca za ažuriranje profila.

6.1.3 Kreiranje i upravljanje kvizovima

Korisnici mogu kreirati nove kvizove putem obrasca za kreiranje kviza, koji se nalazi u glavnom izborniku pod nazivom "New quiz", gdje unose naziv, opis i postavke kviza. Nakon kreiranja, kviz se dodaje u popis dostupnih kvizova, te veliki jezični model generira pitanja na temelju unesenih parametara. Organizator kviza je preusmjeren na može dodavati, uređivati ili brisati pitanja unutar kviza, te izbacivati neželjene igrače iz kviza.



Slika 6: Forma kviza



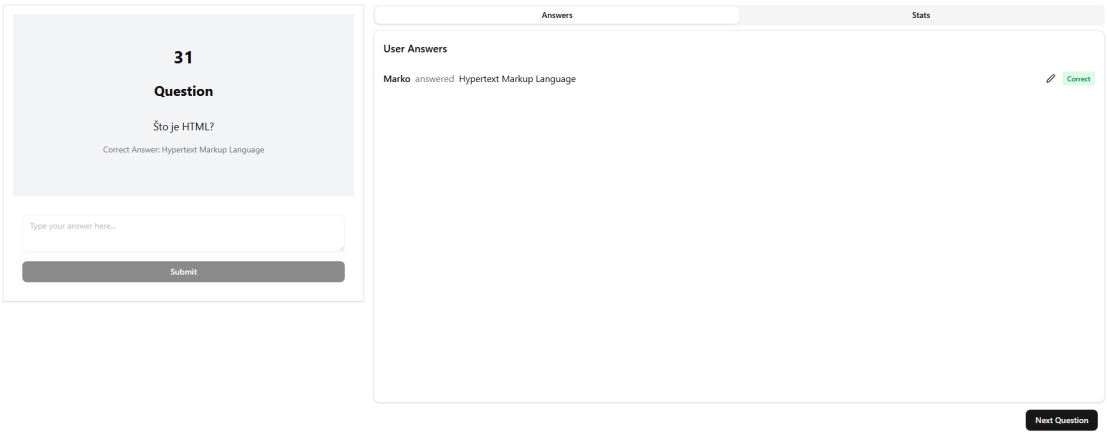
Slika 7: Konfiguracija kviza

6.1.4 Pridruživanje kvizu

Igrači se mogu pridružiti kvizu putem jedinstvenog koda koji im organizator kviza pruža. Nakon unosa koda, igrači mogu promijeniti svoj status u ”ready” .

6.1.5 Početak kviza

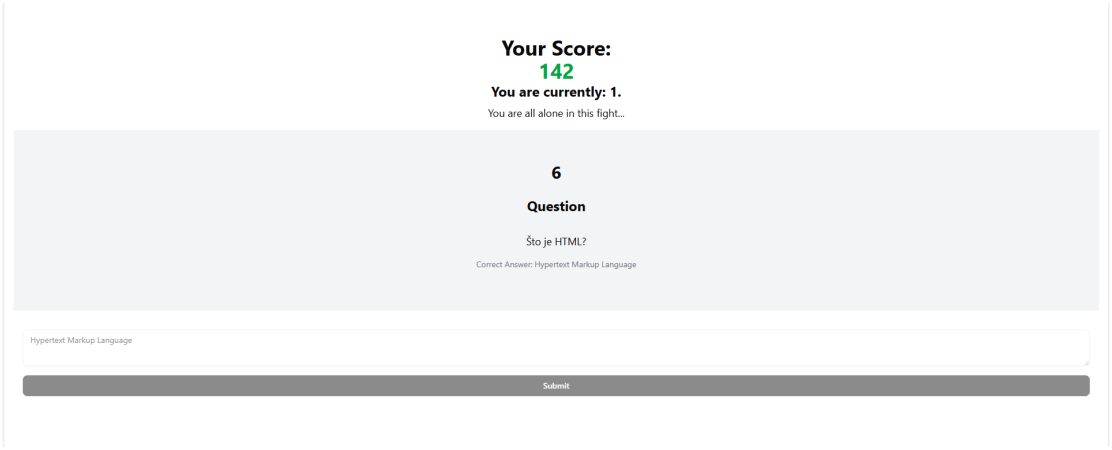
Kada svi igrači postanu ”ready”, organizator može započeti kviz. Organizator je preusmjeren na stranicu trenutnog pitanja kviza. Igrači će tada vidjeti prvo pitanje i imati mogućnost odabira odgovora. Nakon što svi igrači odgovore ili istekne vrijeme, organizator može prijeći na sljedeće pitanje. Organizator u ovom pogledu vidi tko je kako odgovorio na pitanje, te može promijeniti status točnosti odgovora u ”correct” ili ”incorrect”. Također ima uvid u ukupni poredak



Slika 8: Kontrolna ploča kviza

6.1.6 Interakcija sa kvizom

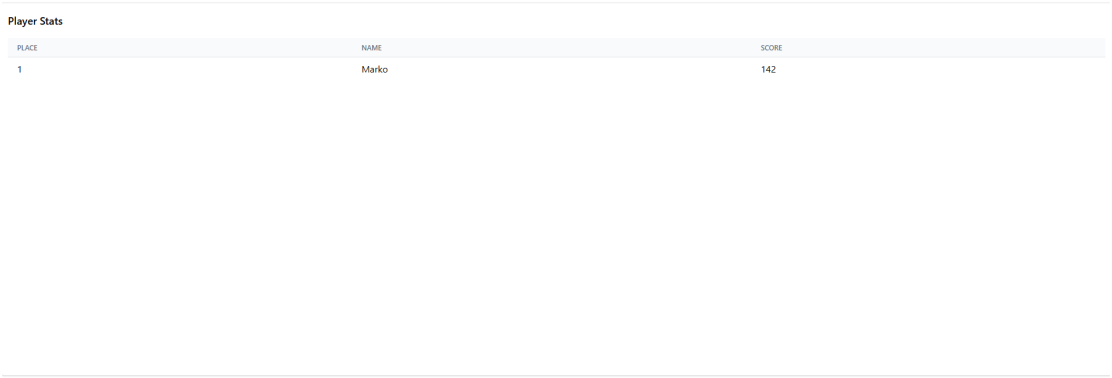
Početkom kviza igrači su preusmjereni na stranicu kviza gdje mogu vidjeti trenutno pitanje, preostalo vrijeme i svoj rezultat. Pitanja se prikazuju jedno po jedno, a igrači mogu odabrati svoj odgovor unutar zadanog vremenskog okvira. Nakon što svi igrači odgovore ili istekne vrijeme, organizator šalje novo pitanje. Igrač odgovara klikom na ponuđene odgovore ili upisom odgovora, a sustav bilježi točnost odgovora i ažurira rezultat igrača u stvarnom vremenu.



Slika 9: Sučelje sa pitanjem za igrače

6.1.7 Pregled rezultata

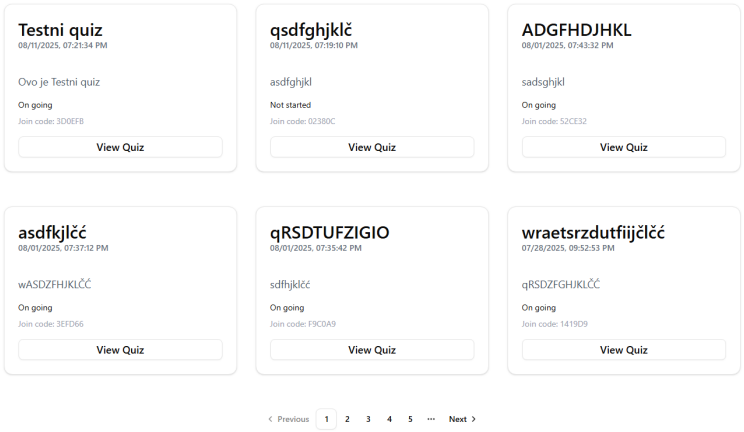
Ako kviz nema više pitanja, "next question" gumb preusmjerava igrače i organizatora na stranicu s rezultatima kviza gdje mogu vidjeti svoje ukupni poredak igrača.



Slika 10: Sučelje sa rezultatima kviza

6.1.8 Izlistaj odigranih kvizova

U glavnom izborniku na linku `See Past Quizzes` korisnici mogu pregledavati popis svojih kreiranih kvizova, uključujući naziv, opis i datum odigravanja.



Slika 11: Izlistaj kvizova sa paginacijom

7 Zaključak

Ovaj projekt demonstrira moć Elixir-a u razvoju interaktivnih web aplikacija, posebno u kontekstu komunikacije u stvarnom vremenu koja je bila potrebna za ovaj projekt. Integracijom umjetne inteligencije, dobivena je mogućnost bržeg kreiranja pitanja i evaluacije odgovora, čime se značajno poboljšava korisničko iskustvo. Ovakva aplikacija može poslužiti kao temelj za daljnju izradu sličnih alata i platformi koje koriste komunikaciju u pravom vremenu i umjetnu inteligenciju za poboljšanje interakcije s korisnicima. Poboljšanja aplikacije mogu uključivati: dodavanje ekrana za čekanje novog pitanja, dodavanje mogućnosti za pregled odigranih kvizova, te implementaciju naprednijih analitika i statistike za organizatore kvizova. Način na kojem je umjetna inteligencija u ovom radu iskorištena samo je jedan primjer od mnogo načina na koje se umjetna inteligencija može integrirati u obrazovne alate i platforme, otvarajući vrata novim mogućnostima za personalizaciju i poboljšanje iskustva učenja i zabave.

Literatura

[1] Kahoot! https://kahoot.com. Pristupljeno: 28. kolovoza 2025.

[2] Quiz.com. https://www.quiz.com. Pristupljeno: 28. kolovoza 2025.

[3] Wayground. https://www.wayground.com. Pristupljeno: 28. kolovoza 2025.

[4] Ecto. Ecto — database wrapper and query generator for elixir, 2025. Accessed: 2025-08-05. URL: https://hexdocs.pm/ecto/Ecto.html.

[5] Github. Github actions — automate your workflow, 2025. Accessed: 2025-08-05. URL: https://docs.github.com/en/actions.

[6] Saša Jurić. *Elixir in Action*, volume 2. Manning Publications Co., 2018.

[7] LLaMA. Llama — large language model, 2025. Accessed: 2025-08-05. URL: https://www.llama.com/docs/how-to-guides/prompting/.

[8] Jeff Nickoloff. *Docker in Action*. Manning Publications, Shelter Island, NY, 2016.

[9] Phoenix Framework. Phoenix — productive. reliable. fast., 2025. Accessed: 2025-08-05. URL: https://www.phoenixframework.org/.

[10] Phoenix Framework. The road to 2 million websocket connections in phoenix, 2025. Accessed: 2025-08-05. URL: https://phoenixframework.org/blog/the-road-to-2-million-websocket-connections.

[11] Ajit Singh. Meta llama 4: The future of multimodal ai. Technical report, SSRN Electronic Journal, April 2025. Available at SSRN: https://ssrn.com/abstract=5208228. doi:10.2139/ssrn.5208228.

[12] Ian Sommerville. *Software Engineering*. Addison-Wesley, 9 edition, 2011.

Popis slika

Popis tablica