Razvoj web aplikacije „Žargonetka“

Peruško, Roko

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Sveučilišni prijediplomski studij Informatika

Roko Peruško

Razvoj web aplikacije „Žargonetka“

Završni rad

Mentor: izv. prof. dr. sc. Marija Brkić Bakarić

Rijeka, rujan 2024.

(*Iza naslovne stranice, na ovome mjestu, prilikom uvezivanja umetnite original zadatka završnog rada kojeg ste preuzeli od mentora*).

Sažetak

Razvoj web aplikacija uz korištenje Django i React okvira sve je popularniji pristup u modernom softverskom inženjerstvu, zahvaljujući svojoj fleksibilnosti, modularnosti i sposobnosti da omogući stvaranje dinamičnih i responzivnih korisničkih sučelja uz stabilnu i skalabilnu poslužiteljsku podršku. Ovaj rad istražuje ključne koncepte i arhitektonska načela razvoja web aplikacija koristeći Django kao backend okvir i React kao frontend tehnologiju, pružajući sveobuhvatan pregled njihovih prednosti, izazova i najboljih praksi.

Praktična primjena ovih tehnologija ilustrirana je kroz projekt **Žargonetka**, koji predstavlja osnovni rječnik žargona s dodatnim funkcionalnostima. Kroz ovaj projekt prikazano je kako Django osigurava pouzdanu poslužiteljsku infrastrukturu, dok React omogućuje izradu dinamičnih i interaktivnih korisničkih sučelja. Projekt **Žargonetka** ističe ključne aspekte integracije i implementacije oba okvira te naglašava važnost dobre organizacije koda i upravljanja podacima za optimalno korisničko iskustvo.

U radu se također analizira niz izazova s kojima se susreću developeri tijekom razvoja web aplikacija koristeći Django i React, poput autentifikacije korisnika, upravljanja stanjem aplikacije, te integracije različitih modula i servisa. Rad pruža uvid u suvremene trendove i buduće smjerove razvoja web aplikacija.

Kroz analizu projekta **Žargonetka**, objašnjeni su ključni koraci i značajke aplikacije, pružajući praktičnu perspektivu i uvid u rad s ovim tehnologijama te njihovu ulogu u izradi suvremenih, korisnički orijentiranih web aplikacija.

Ključne riječi: Django; React; Razvoj web aplikacije; Žargonetka; Rječnik; Integracija; Autentifikacija; FIDIT

SADRŽAJ

1. UVOD

U suvremenom svijetu razvoja softvera, web aplikacije zauzimaju središnje mjesto u pružanju raznolikih digitalnih usluga i korisničkih iskustava. Razvoj web aplikacija zahtijeva korištenje učinkovitih i skalabilnih tehnologija koje omogućuju brzo i fleksibilno stvaranje dinamičnih i interaktivnih korisničkih sučelja te pouzdanu poslužiteljsku podršku [1]. Jedan od najpopularnijih pristupa u razvoju ovakvih aplikacija jest kombinacija okvira Django i React. Django, kao robustan i siguran web okvir zasnovan na Pythonu [2], omogućuje izgradnju složenih backend sustava, dok React, popularna JavaScript biblioteka za izradu korisničkih sučelja [3], omogućuje razvoj responzivnih i dinamičnih frontend aplikacija.

Django i React sve se češće koriste u tandemu zbog svoje komplementarnosti – Django osigurava skalabilan i siguran backend s visokom razinom modularnosti [2], dok React omogućuje razvoj sučelja koja su jednostavna za upotrebu i prilagodljiva korisničkim potrebama [3]. Zbog svojih prednosti, ova kombinacija predstavlja idealan izbor za brojne projekte koji zahtijevaju brz razvoj, visoke performanse i ugodno korisničko iskustvo [1][3].

Motivacija za odabir ove teme proizašla je iz želje da se stekne dublje razumijevanje razvoja modernih web aplikacija te da se primjene stečena znanja kroz praktični projekt. Rad istražuje kako se ove tehnologije međusobno nadopunjuju i kako se mogu koristiti za izradu aplikacija koje su skalabilne, jednostavne za održavanje i pružaju izvanredno korisničko iskustvo [2][3]. Poseban fokus ovog rada je na izradi aplikacije Žargonetka, koja predstavlja osnovni rječnik žargona s dodatnim funkcionalnostima, a koja služi kao konkretan primjer primjene Django i React okvira u razvoju web aplikacija.

Ciljevi rada su:

- Istražiti temeljne koncepte i arhitektonske principe Django i React tehnologija.
- Analizirati proces razvoja web aplikacija pomoću kombinacije ovih tehnologija, uključujući izazove i najbolje prakse.
- Prikazati praktičnu primjenu kroz razvoj aplikacije Žargonetka, s detaljnim opisom implementacije i ključnih značajki.
- Razmotriti sigurnosne aspekte, performanse i optimizaciju aplikacija temeljenih na Django i React tehnologijama.

2. DJANGO WEB FRAMEWORK

2.1. Uvod

Django je robustan i fleksibilan web okvir otvorenog koda napisan u Python-u, razvijen s ciljem da pojednostavi i ubrza proces izrade web aplikacija [2][4]. Kroz svoj "batteries included" pristup, Django nudi bogat skup unaprijed izrađenih funkcionalnosti, čime omogućava programerima da se usmjere na kreiranje specifičnih značajki svojih aplikacija, a ne na razvoj osnovne infrastrukture. Ovaj pristup značajno smanjuje vrijeme potrebno za razvoj i smanjuje rizik od grešaka u osnovnim funkcionalnostima, čime Django postaje privlačan izbor za širok spektar aplikacija, od manjih projekata do složenih, visoko skalabilnih sustava [2][4].

Jedna od ključnih prednosti Django okvira je njegova modularnost i visok stupanj sigurnosti. Django dolazi s nizom ugrađenih sigurnosnih značajki, kao što su zaštita od napada kao što su SQL injection, cross-site scripting (XSS) i cross-site request forgery (CSRF), što ga čini idealnim za razvoj aplikacija koje zahtijevaju visoku razinu sigurnosti, poput financijskih sustava ili platformi za elektroničku trgovinu [2][4]. Također, zbog svoje modularne strukture, Django omogućuje lako proširivanje i integraciju s drugim tehnologijama i servisima [2][4].

Django prati MVT (Model View Template) dizajn obrazac. Model predstavlja podatke, obično iz baze podataka. View obrađuje zahtjeve i vraća odgovarajući predložak i sadržaj temeljen na korisničkom zahtjevu, dok Template predstavlja tekstualnu datoteku (poput HTML datoteke) koja sadrži izgled web stranice s ugrađenom logikom za prikaz podataka [2][4]. Model omogućava rad s podacima iz baze kroz objektno-relacijsko mapiranje (engl. Object Relational Mapping – ORM), tehniku koja olakšava rad s bazama podataka bez potrebe za pisanjem složenih SQL upita. Pogledi (engl. Views) su funkcije ili metode koje prihvaćaju HTTP zahtjeve, dohvaćaju relevantne modele i vraćaju obrađeni rezultat korisniku. Predlošci (engl. Templates) su datoteke koje opisuju kako će rezultat biti prikazan, koristeći standardni HTML kod i Django oznake za dodavanje logike prikaza. Django također pruža sustav za navigaciju kroz različite stranice web stranice pomoću datoteke urls.py, gdje se definira koja će funkcija pogleda biti pozvana za određeni URL zahtjev. Kada korisnik pošalje zahtjev za određenm URL-om, Django prvo provjerava urls.py datoteku kako bi pronašao odgovarajući pogled, zatim u pogledu (views.py) provjerava relevantne modele (models.py), šalje podatke u specificirani predložak u mapi templates, a predložak potom vraća HTML sadržaj natrag pregledniku [2][4].

Izuzetno je prilagodljiv i podržava različite paradigme razvoja, od klasičnih monolitnih aplikacija do mikroservisne arhitektura, te podržava različite baze podataka i laku integraciju s frontend tehnologijama kao što su React, Vue.js ili Angular [2][4].

3. Komunikacija backenda i frontenda

U modernim web aplikacijama, učinkovita komunikacija između backend-a (poslužiteljske strane) i frontend-a (klijentske strane) od presudne je važnosti za pravilno funkcioniranje sustava i pružanje optimalnog korisničkog iskustva [1][4]. Backend i frontend međusobno komuniciraju kako bi razmjenjivali podatke, obradili korisničke zahtjeve i omogućili prikaz sadržaja na pregledniku korisnika. Ova komunikacija je ključna za realizaciju dinamičkih i interaktivnih web aplikacija te se oslanja na nekoliko ključnih tehnologija i protokola [1][4].

3.1. HTTP protokol

HTTP (Hypertext Transfer Protocol) je osnovni protokol za prijenos podataka između web preglednika (frontend) i web poslužitelja (backend) [1][6]. Svaki put kada korisnik posjeti web stranicu ili izvrši neku radnju (npr. klikne na gumb ili pošalje obrazac), preglednik šalje HTTP zahtjev poslužitelju. Poslužitelj zatim obrađuje taj zahtjev i vraća HTTP odgovor koji može sadržavati podatke poput HTML-a, CSS-a, JavaScript-a, slika ili drugih resursa potrebnih za prikaz stranice ili izvršenje zadane funkcionalnosti. Ovaj mehanizam je temelj za komunikaciju između klijenta i poslužitelja i omogućava frontend aplikaciji da dinamički zatraži i prikaže sadržaj te pruži interaktivne značajke korisnicima. HTTP protokol također podržava sigurnosne protokole kao što su HTTPS, koji osigurava šifrirani prijenos podataka i štiti osjetljive informacije od neovlaštenog pristupa [1][4][6].

3.2. REST API

REST (Representational State Transfer) je arhitektonski stil koji definira skup smjernica za stvaranje API-ja (Application Programming Interfaces) koji omogućuju komunikaciju između klijenta i poslužitelja koristeći HTTP protokol [1]. REST API-i koriste različite HTTP metode za izvođenje operacija na resursima:

- **GET**: Dohvaća podatke s poslužitelja, primjerice za preuzimanje korisničkih informacija ili popisa proizvoda.
- **POST**: Šalje podatke na poslužitelj kako bi se stvorio novi resurs, kao što je registracija novog korisnika ili dodavanje novog proizvoda.
- **PUT**: Ažurira postojeći resurs na poslužitelju, primjerice za promjenu korisničkih postavki ili ažuriranje informacija o proizvodu.
- **DELETE**: Briše postojeći resurs s poslužitelja, poput brisanja korisničkog računa ili uklanjanja proizvoda iz baze podataka.

REST API-ji često koriste JSON (JavaScript Object Notation) kao format za razmjenu podataka zbog njegove jednostavnosti i lakog korištenja u JavaScriptu, koji je najčešće korišten jezik za razvoj frontenda [1][5]. Ponekad se koristi i XML (eXtensible Markup Language), koji je stariji format s bogatijim mogućnostima za opisivanje strukture podataka. Korištenjem

REST API-ja, frontend može jednostavno komunicirati s backendom i obavljati operacije poput autentifikacije, upravljanja korisnicima i manipulacije podacima.

3.3. CORS

U slučajevima kada frontend aplikacija treba komunicirati s poslužiteljem koji se nalazi na drugoj domeni, protokolu ili portu, koristi se CORS (Cross-Origin Resource Sharing) [1][4]. CORS je sigurnosni mehanizam koji omogućava poslužiteljima da definiraju koji vanjski izvori mogu pristupiti njihovim resursima. Na primjer, kada web aplikacija koja se pokreće na https://example-frontend.com pokušava napraviti zahtjev prema API-ju na https://api-backend.com, CORS omogućava poslužitelju da specificira može li ili ne može odgovoriti na zahtjeve s druge domene. Ovaj mehanizam sprječava neovlaštene aplikacije da pristupaju podacima i resursima, te štiti osjetljive informacije korisnika. Poslužitelj može odrediti specifične domene, HTTP metode ili vrste zaglavlja koja su dopuštena, čime se osigurava stroga kontrola pristupa [1][4].

4. React

React je JavaScript biblioteka otvorenog koda koju je razvio Facebook, a koristi se za izgradnju korisničkih sučelja (UI). Primarno je namijenjen za razvoj jednostrukih stranica (*Single Page Applications* - *SPA*) i dinamičkih aplikacija koje se brzo i učinkovito ažuriraju te prikazuju velike količine podataka. Evo ključnih aspekata koje treba razumjeti i uzeti u obzir pri korištenju React-a za *web* aplikacije:

4.1. Komponentni pristup

React koristi komponentni pristup za razvoj korisničkih sučelja. Komponente su osnovni gradivni blokovi u React-u koji predstavljaju dijelove sučelja, kao što su gumbi, obrasci, izbornici, popisi, pa čak i cijele stranice. Svaka komponenta je u biti izolirani, neovisni element koji upravlja vlastitim stanjem i ponašanjem. Ovaj modularni pristup omogućuje programerima da ponovno koriste komponente u različitim dijelovima aplikacije, olakšavajući održavanje, testiranje i skaliranje aplikacije [3].

- Funkcijske komponente: Moderne komponente u Reactu definiraju se pomoću JavaScript funkcija. One su jednostavne, lakše za razumijevanje i upotrebu, posebno kada se koriste zajedno s React Hooks (funkcionalnost koja omogućava korištenje stanja i sporednih efekata unutar funkcijskih komponenti). Funkcijske komponente čine kod manje složenim i omogućuju brži razvoj aplikacija [3].

- Klasa komponente: Stariji način definiranja komponenti temelji se na JavaScript klasama. Iako su klasa komponente i dalje podržane, sve se više zamjenjuju funkcijskim komponentama zbog jednostavnosti i prednosti koje pružaju React Hooks [3].

4.2. Virtualni DOM

Jedan od ključnih razloga za korištenje React-a je njegova implementacija Virtualnog DOM-a (*Document Object Model*). DOM je struktura koja predstavlja HTML dokument u obliku stabla, a svaka grana stabla predstavlja čvor s elementima poput div, p, h1, i tako dalje. Manipulacija stvarnog DOM-a može biti skupa operacija jer zahtijeva značajne resurse za ažuriranje i ponovno renderiranje cijele stranice.

React optimizira ovaj proces kroz korištenje Virtualnog DOM-a, koji stvara virtualnu kopiju stvarnog DOM-a. Kada dođe do promjene u stanju aplikacije, React ažurira samo virtualni DOM, a zatim uspoređuje novu i staru verziju virtualnog DOM-a u postupku koji se zove *diffing*. Nakon toga, React samo minimalno i optimalno ažurira stvarni DOM, ažurirajući samo one dijelove koji su se zaista promijenili. Ovaj postupak značajno poboljšava performanse aplikacije, posebice u složenim i velikim aplikacijama koje zahtijevaju brza ažuriranja korisničkog sučelja [3].

4.3. State i props

State i Props su dva ključna koncepta koja upravljaju podacima u React aplikacijama i omogućuju stvaranje dinamičkih i interaktivnih korisničkih sučelja:

- State: Unutarnje stanje komponente koje se može mijenjati tijekom vremena. State se koristi za praćenje dinamičkih podataka koji se mogu promijeniti kao rezultat korisničke interakcije ili drugih faktora (npr. podaci iz obrasca, odgovori s API-ja). Kada se stanje promijeni, React automatski ponovno renderira komponentu kako bi prikazao ažurirane podatke, čime se omogućuje reaktivno korisničko iskustvo.

- Props: Props (skraćeno od *"properties"*) su podaci koji se prosljeđuju iz roditeljske komponente u podređene komponente. Props se koriste za prijenos informacija između komponenti i omogućuju ponovnu upotrebu komponenata s različitim podacima. Props su *"read-only"*, što znači da ih podređene komponente ne mogu mijenjati, već ih koriste samo za prikazivanje ili izvođenje logike na temelju tih podataka [3].

4.4. React hooks

React hooks su funkcije uvedene u React 16.8 koje omogućuju korištenje stanja i drugih značajki React-a unutar funkcijskih komponenti, bez potrebe za pisanjem klasa. Ovo je revolucioniralo način na koji se React koristi jer omogućuje jednostavniju i fleksibilniju izgradnju komponenti.

Neki od najvažnijih Hook-ova su:

- **useState**: Omogućuje korištenje stanja unutar funkcijske komponente. Korištenjem useState hook-a, možete definirati i upravljati stanjem unutar komponente na jednostavan i čitljiv način.

- **useEffect**: Koristi se za obavljanje sporednih efekata, kao što su dohvaćanje podataka, postavljanje pretplata ili ručno manipuliranje DOM-om unutar funkcijske komponente. Zamjenjuje metode životnog ciklusa u klasa komponentama, poput componentDidMount ili componentDidUpdate.

- **useContext**: Omogućuje korištenje konteksta za dijeljenje podataka između komponenti bez potrebe za "props drillingom", odnosno prosljeđivanjem propsa kroz svaki sloj komponenti, čime se olakšava upravljanje stanjem na višim razinama [3].

4.5. Upravljanje stanjem aplikacije

Kako aplikacija raste, upravljanje stanjem može postati izazovno. React omogućuje jednostavno upravljanje stanjem na razini komponente, ali za složenije aplikacije može se koristiti dodatne biblioteke kao što su Redux, MobX ili ugrađeni Context API.

- **Redux** je najpopularnija biblioteka za upravljanje stanjem u React aplikacijama. Omogućuje centralizirano skladište stanja koje sve komponente mogu pristupiti i ažurirati pomoću definiranih akcija i reducera. Redux pomaže u predvidljivom upravljanju stanjem aplikacije i olakšava dijeljenje podataka među različitim dijelovima aplikacije.

- **Context API** je ugrađena značajka Reacta koja omogućuje dijeljenje stanja i podataka između komponenti bez potrebe za prosljeđivanjem propsa kroz više slojeva. Ovaj pristup je koristan za manje aplikacije ili specifične podatke koji se često koriste u više komponenti [3].

4.6. React Router

React Router je biblioteka koja omogućuje upravljanje preusmjeravanjem u SPA aplikacijama. Pomoću React Router-a, programeri mogu definirati različite rute za aplikaciju, što omogućuje navigaciju između različitih dijelova aplikacije bez potrebe za ponovnim učitavanjem cijele stranice [3].

4.7. SSR i CSR

React podržava i *Server-Side Rendering* (SSR) i *Client-Side Rendering* (CSR).

- CSR je tradicionalni način rada u Reactu, gdje se kompletna aplikacija učitava na klijentu (pregledniku) i svi podaci se dinamički učitavaju i prikazuju pomoću JavaScript-a.

- SSR omogućuje inicijalno renderiranje aplikacije na poslužitelju, a zatim šalje već prikazani HTML klijentu. To poboljšava performanse i SEO (*Search Engine Optimization*) jer pretraživači mogu lakše indeksirati sadržaj. SSR je posebno koristan za aplikacije koje trebaju brzo vrijeme učitavanja prve stranice ili bolje rezultate u pretraživačima [3].

4.8. Alati i podrška

React ima pristup velikom broju alata, dodataka i biblioteka koje olakšavaju razvoj aplikacija. To uključuje razvojne alate poput React DevTools, generatora projekata kao što je Create React App, te *okvir*e kao što su Next.js za SSR ili Gatsby za statične stranice. Osim toga, React ima veliku zajednicu programera, što znači da je lako pronaći resurse, vodiče i podršku za gotovo bilo koji problem [3].

5. AWS

AWS (Amazon Web Services) je vrlo popularna platforma za udomljavanje (engl. host) i postavljanje (engl. deploy) web aplikacija. Njene brojne usluge omogućuju razvojnim programerima da lako postave, skaliraju i upravljaju svojim aplikacijama, bez potrebe za upravljanjem fizičkom infrastrukturom.

5.1. Kako AWS pomaže u implementaciji web aplikacija

1. *Hosting* aplikacije na virtualnim serverima: Jedna od najčešćih usluga AWS-a za implementaciju web aplikacija je Amazon EC2 (*Elastic Compute Cloud*). EC2 omogućava pokretanje virtualnih servera (poznatih kao instance) na kojima se može izvršavati aplikacija. Developer može konfigurirati ove servere prema potrebama aplikacije, uključujući izbor operativnog sustava, količine RAM-a, procesorske snage i prostora za pohranu. Aplikacija se može instalirati na ove servere, gdje se pokreće i obrađuje korisničke zahtjeve[7].

2. Pohrana statičkih resursa uz Amazon S3: Amazon S3 (*Simple Storage Service*) koristi se za pohranu statičkih datoteka poput slika, videozapisa, dokumenata ili drugih vrsta podataka koje aplikacija koristi. S3 je vrlo skalabilan servis za pohranu koji pruža visoku dostupnost i sigurnost. Kada se aplikacija *postave* na AWS, statički resursi se mogu pohraniti na S3 i posluživati krajnjim korisnicima s minimalnim kašnjenjem, uz mogućnost korištenja Amazon CloudFront CDN-a (*Content Delivery Network*) za još bržu isporuku [7].

3. Skalabilnost s AWS *Auto Scaling*-om: AWS omogućava automatsko skaliranje resursa pomoću AWS *Auto Scaling*-a. Ovaj servis automatski prilagođava broj pokrenutih EC2 instanci na temelju unaprijed definiranih metrika, kao što su CPU opterećenje ili broj korisničkih zahtjeva. Time se osigurava optimalna razina performansi aplikacije uz minimalne troškove, jer se resursi automatski prilagođavaju stvarnim potrebama aplikacije [7].

5.2. Koraci za implementaciju web aplikacije na AWS

Prije implementacije, aplikacija se priprema za produkcijsko okruženje. Ovo može uključivati konfiguriranje datoteka za produkciju, osiguranje baze podataka, optimiziranje kôda i resursa te provođenje sigurnosnih testova. Ovisno o potrebama aplikacije, developer bira odgovarajuće AWS servise. Na primjer, za jednostavne aplikacije može se koristiti AWS Elastic Beanstalk, dok za složenije zahtjeve i veće aplikacije može biti potreban EC2 uz dodatne servise poput RDS-a (Relational Database Service) za bazu podataka ili S3 (za pohranu).

Postavljanje aplikacije na AWS:

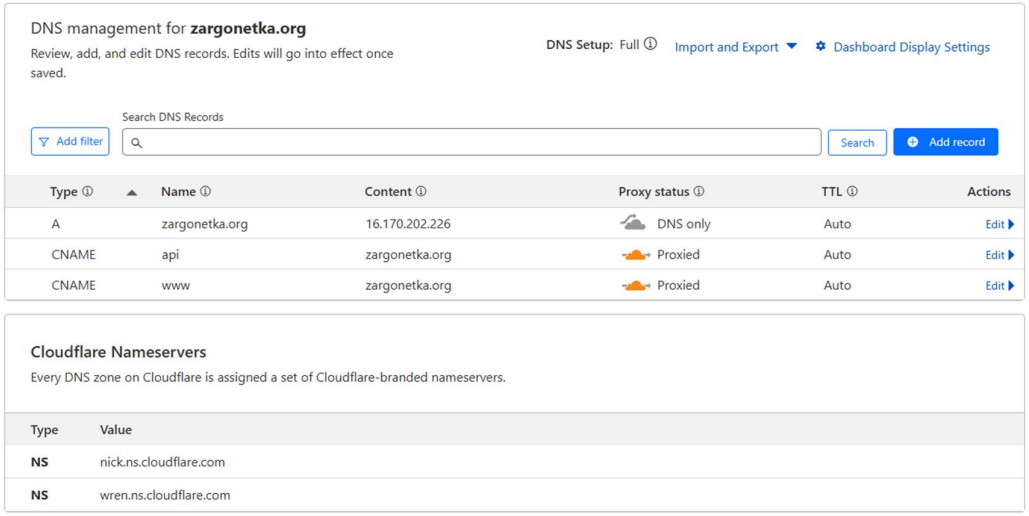
- Korištenje Amazon EC2: Ako se koristi EC2, developer će ručno kreirati EC2 instancu, instalirati potrebne komponente (kao što su *web server*, baze podataka, itd.) i implementirati aplikaciju putem SSH pristupa.
- Konfiguriranje domene i SSL certifikata: važno je postaviti prilagođenu domenu (putem AWS Route 53 ili drugog registrar servisa) i dodati SSL certifikat (npr. putem AWS Certificate Manager-a) kako bi se osigurao siguran HTTPS pristup aplikaciji.

6. CLOUDFLARE

Kada se koristiCloudflare za *postavlj*anje *web* aplikacije, koriste se njihove napredne značajke za sigurnost, brzinu i jednostavno upravljanje DNS zapisima, što je ključno za povezivanje domene s aplikacijom. U slučaju "Žargonetke", već sam kupio vlastitu domenu i konfigurirao DNS zapise koristeći Cloudflare.

Upravljanje DNS zapisima: Jedan od prvih koraka nakon kupovine domene je postavljanje DNS (*Domain Name System*) zapisa. DNS zapisi upravljaju načinom na koji internetske usluge, kao što su web preglednici, preusmjeravaju promet prema web stranici. A zapis za zargonetka.org: Postavlja glavnu (*root*) domenu zargonetka.org tako da pokazuje na IP adresu servera gdje je aplikacija udomljena.

- CNAME zapis za "www" *pod*domenu: Omogućava da se promet usmjeren prema www.zargonetka.org preusmjeri na glavnu domenu zargonetka.org, koristeći Cloudflare proxy za sigurnost i performanse.
- CNAME zapis za "api" poddomenu: Postavlja da api.zargonetka.org bude proslijeđen kroz Cloudflare, što znači da Cloudflare posreduje i osigurava promet prema API-ju, čime se omogućava brža i sigurnija isporuka podataka [8].



Slika 1. Prikaz DNS konfiguracije na Cloudflare-u

4. Sigurnost kroz Proxy i SSL: Cloudflare-ova funkcija proxy statusa omogućuje dodatnu zaštitu podataka. Kada je proxy uključen (kao u slučaju zapisa za api i www poddomene), Cloudflare preusmjerava promet kroz vlastitu mrežu, dodajući slojeve sigurnosti kao što su DDoS zaštita, automatski SSL certifikati i WAF (*Web Application Firewall*). Ova sigurnosna mjera osigurava da su svi podaci između korisnika i servera kriptirani i zaštićeni od potencijalnih napada [8].

5. Cloudflare Nameservers: Da bi Cloudflare mogao upravljati domenom, potrebno je postaviti Cloudflare-ove DNS poslužitelje za domenu zargonetka.org. U mom slučaju, nameserveri su nick.ns.cloudflare.com i wren.ns.cloudflare.com. Ovi nameserveri preusmjeravaju promet prema Cloudflare mreži i omogućuju upravljanje svim DNS zapisima i sigurnosnim postavkama putem Cloudflare sučelja.

6. Jednostavno postavljanje i skaliranje: Korištenjem Cloudflare-a za *postavlj*anje aplikacije može se lako skalirati kako bi podržala više korisnika bez potrebe za kompliciranom infrastrukturom [8].

7. Analiza aplikacije "Žargonetka"

Aplikacija Žargonetka osmišljena je kao interaktivni internetski rječnik koji korisnicima omogućuje unos, pregled i dijeljenje žargona, odnosno specifičnih riječi i fraza unutar nekog socijalnog kruga. Cilj aplikacije Žargonetka je stvoriti digitalnu platformu koja služi kao centralizirano mjesto za prikupljanje, definiranje i dijeljenje takvih pojmova. U ovom poglavlju istražiti ćemo proces razvoja Žargonetke sa *backend* strane koristeći Django, te sa *frontend* strane koristeći React.

7.1. Backend

U ovom potpoglavlju detaljno će se analizirati ključne komponente backend arhitekture projekta "Žargonetka" i njegovih aplikacija base_app, phrases_app i words_app. Base_app pruža osnovnu funkcionalnost za registraciju korisnika i upravljanje njihovim profilima, što je temelj za daljnje operacije unutar sustava. Aplikacije phrases_app i words_app proširuju ovu funkcionalnost specifičnim modelima i operacijama za upravljanje frazama i riječima.

Modeli u phrases_app i words_app definiraju strukturu podataka i njhove odnose, PhrasesViewSet i WordsViewSet omogućuju složene operacije kao što su pretraživanje, označavanje i nasumični odabir stavki. Serijalizatori osiguravaju pravilnu transformaciju podataka u JSON format za API komunikaciju, omogućujući interakciju s korisnicima. Konfiguracija URL putanja, putem DefaultRouter, automatizira generiranje URL putanja i omogućuje pristup dokumentaciji putem Swagger i ReDoc alata.

7.1.1. Lokalno razvojno okruženje

Razvoj *web* aplikacije započeo je postavljanjem lokalnog okruženja korištenjem virtualnog okruženja (*virtual environment*). Prvi korak bio je kreiranje virtualnog okruženja pomoću naredbe python -m venv myproject, koja stvara izolirani prostor za instalaciju svih potrebnih Python paketa, uključujući Django. Sljedeći korak bio je aktiviranje virtualnog okruženja. To se postiže izvršavanjem naredbe venv\Scripts\activate. Nakon uspješnog aktiviranja virtualnog

okruženja, instaliran je Django koristeći naredbu pip install Django, koja preuzima i instalira najnoviju verziju ovog *web* okvira. Posljednji korak u postavljanju okruženja bio je kreiranje Django projekta. Django projekt je Python paket koji sadrži sve potrebne postavke za pokretanje web aplikacije, uključujući konfiguraciju baze podataka, opcije specifične za Django i postavke aplikacije. Projekt je kreiran pomoću alata django-admin uz naredbu django-admin startproject `ZargonetkaProjekt`, koja generira sve potrebne datoteke i direktorije za daljnji razvoj aplikacije.

7.1.2. Struktura Django projekta

Projekt Žargonetka organiziran je u nekoliko ključnih direktorija i datoteka, koji zajedno čine osnovu Django aplikacije. Struktura projekta osigurava jasan i logičan raspored potrebnih konfiguracijskih datoteka, modula i skripti koje omogućuju efikasan razvoj, održavanje i pokretanje *web* aplikacije.

Glavni direktorij projekta

ZargonetkaProjekt/: Ovo je korijenski direktorij Django projekta, koji sadrži sve osnovne konfiguracijske datoteke i direktorije potrebne za postavljanje i upravljanje aplikacijom. U ovom direktoriju nalaze se važne komponente koje omogućuju definiranje postavki aplikacije, URL putanja, te pokretanje aplikacije na različitim poslužiteljima.

__init__.py: Datoteka koja označava da je direktorij Python paket, omogućujući Django-u i Pythonu da pravilno prepoznaju projekt i njegove module.

asgi.py i wsgi.py: Konfiguracijske datoteke za ASGI (*Asynchronous Server Gateway Interface*) i WSGI (*Web Server Gateway Interface*). Ove datoteke omogućuju pokretanje Django aplikacije na različitim vrstama poslužitelja, osiguravajući kompatibilnost s asinkronim i sinkronim okruženjima.

settings.py: Središnja konfiguracijska datoteka projekta koja sadrži sve ključne postavke, uključujući konfiguraciju baze podataka, aplikacija, sigurnosnih postavki, lokalizacije, statičkih datoteka, i ostalih opcija relevantnih za rad aplikacije.

urls.py: Datoteka koja definira URL putanje projekta, upravljajući usmjeravanjem HTTP zahtjeva na odgovarajuće aplikacije i poglede unutar Django aplikacije. Omogućuje strukturiranje URL-ova i mapiranje na funkcije ili klase pogleda.

manage.py: Skripta za upravljanje projektom, koja omogućuje izvršavanje različitih administrativnih zadataka, kao što su pokretanje razvojnog servera, izvođenje migracija baze podataka, kreiranje superkorisnika, te druge radnje vezane uz razvoj i održavanje aplikacije.

Ostali direktoriji i datoteke:

venv/: Virtualno okruženje specifično za ovaj projekt koje sadrži sve instalirane Python pakete i njihove verzije. Korištenje virtualnog okruženja osigurava izoliranost i konzistentnost razvojnih uvjeta, što je ključno za upravljanje ovisnostima projekta i izbjegavanje konflikata između različitih verzija paketa.

db.sqlite3: Datoteka koja sadrži SQLite bazu podataka, korištenu za pohranjivanje svih podataka aplikacije, uključujući korisničke podatke, fraze, riječi i druge relevantne informacije. Ova baza podataka je jednostavna za korištenje i konfiguraciju, što je čini idealnim izborom za manje aplikacije ili razvojne svrhe.

requirements.txt: Tekstualna datoteka koja sadrži popis svih potrebnih Python paketa i njihovih specifičnih verzija koje su potrebne za ispravno funkcioniranje projekta. Ova datoteka olakšava postavljanje projektnog okruženja na različitim sustavima i osigurava da sve potrebne ovisnosti budu instalirane i ažurirane.

7.1.3. Datoteka settings.py u Django projektu

Datoteka settings.py u projektu ključna je za konfiguraciju svih aspekata aplikacije, uključujući postavke baze podataka, sigurnost, instalirane aplikacije i predloške. U projektu Žargonetka, glavne postavke uključuju:

- BASE_DIR: Definira osnovnu putanju do glavnog direktorija projekta, omogućujući relativne putanje unutar projekta. U projektu Žargonetka, BASE_DIR je definiran kao:

```
BASE_DIR = Path(file).resolve().parent.parent
```

  - Ova postavka osigurava da sve putanje budu relativne u odnosu na glavni direktorij projekta, što pojednostavljuje rad u različitim okruženjima.

- SECRET_KEY: Ključ koji Django koristi za kriptografske operacije, kao što su potpisivanje kolačića i sesija. Mora biti jedinstven za svaki projekt i ne smije se dijeliti kako bi se osigurala sigurnost podataka.

- ALLOWED_HOSTS: Lista domena ili IP adresa s kojih projekt može primati zahtjeve. Ova postavka štiti aplikaciju od napada kao što je DNS *rebinding*.

- INSTALLED_APPS: Definira sve aplikacije koje su instalirane i aktivne u projektu. Osim osnovnih Django aplikacija poput administracijskog sučelja i autentifikacije, projekt Žargonetka uključuje specifične aplikacije kao što su words_app, phrases_app i base_app.

  - Da bi se automatski dodale sve aplikacije sa sufiksom _app, koristi se funkcija discover_apps:

```
def discover_apps(base_dir='apps', pattern='*_app'):

    apps = []

    app_dirs = glob.glob(os.path.join(base_dir, pattern))

   for app_dir in app_dirs:

        if os.path.isdir(app_dir):
```

```
            app_name = os.path.basename(app_dir)

            apps.append(f'{base_dir}.{app_name}')

    return apps
```

- Ova funkcija pretražuje direktorij apps za sve poddirektorije koji odgovaraju obrascu imena sa sufiksom _app i dodaje ih u listu aplikacija. Postavka `INSTALLED_APPS += discover_apps()` omogućuje dinamično dodavanje aplikacija u projekt.

- MIDDLEWARE: Sloj koji obrađuje zahtjeve i odgovore aplikacije. Middleware upravlja sesijama, kolačićima, sigurnosnim postavkama (poput CORS-a), i drugim ključnim funkcijama.

- DATABASES: Definira konfiguraciju baze podataka. U projektu Žargonetka koristi se SQLite baza podataka:

```
DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': BASE_DIR / 'db.sqlite3',}}
```

- REST_FRAMEWORK i SIMPLE_JWT: Konfiguriraju rad Django REST okvira i JWT (JSON Web Token) autentifikacije. REST_FRAMEWORK definira osnovne postavke za rad s REST API-jem, dok SIMPLE_JWT omogućuje rad s JWT autentifikacijom, osiguravajući sigurnost API-ja.

- TEMPLATES: Sekcija koja definira upravljanje predlošcima za generiranje HTML stranica. Specifikacija putanja do direktorija s predlošcima i kontekst procesora omogućuje prilagodbu izgleda i sadržaja aplikacije prema potrebama korisnika.

- STATIC_URL: Definira URL putanju za pristup statičkim datotekama poput CSS, JavaScript i slikovnih datoteka unutar projekta.

7.1.4. Datoteka urls.py u Django projektu

Datoteka urls.py unutar glavnog direktorija Django projekta Žargonetka definira ključne URL putanje koje povezuju različite aplikacije i funkcionalnosti unutar projekta. Ove putanje omogućavaju pristup administrativnom sučelju, API endpointima, autentifikaciji i dokumentaciji.

- admin.site.urls: Ova putanja omogućava pristup administracijskom sučelju Django-a:

```
path('admin/', admin.site.urls)
```

Administratori mogu koristiti ovaj *panel* za upravljanje korisnicima, frazama, riječima i drugim podacima aplikacije putem ugrađenog sučelja koje nudi Django.

- rest_framework.urls: Putanja za autentifikaciju korisnika putem Django REST Frameworka (DRF):

```
path('api-auth/', include('rest_framework.urls'))
```

Ovaj URL pruža osnovne putanje za prijavu i odjavu korisnika, omogućujući sigurnu autentifikaciju unutar aplikacije.

- Sljedeće putanje povezuju aplikacije "words_app", "phrases_app" i "base_app" s glavnim URL-ovima projekta:

```
path('api/words/', include('words_app.urls'))

path('api/phrases/', include('phrases_app.urls'))

path('api/accounts/', include('base_app.urls'))
```

Ove putanje omogućuju pristup REST API-jima za manipulaciju riječima, frazama i korisničkim podacima u aplikaciji.

- JWT Autentifikacija: Putanje za rad s JWT (JSON Web Token) autentifikacijom:

```
path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair')

path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh')
```

    - api/token/: Endpoint za dobivanje JWT tokena, omogućava korisnicima prijavu i autentifikaciju.
    - api/token/refresh/: Endpoint za osvježavanje postojećeg JWT tokena, omogućuje produljenje sesije bez potrebe za ponovnom prijavom.

- Swagger i ReDoc Dokumentacija: Putanje za automatsko generiranje dokumentacije za API koristeći Swagger i ReDoc alate:

```
path('swagger/',  schema_view.with_ui('swagger',  cache_timeout=0),  name='schema-swagger-ui')

path('redoc/', schema_view.with_ui('redoc', cache_timeout=0), name='schema-redoc')
```

    - swagger/ i redoc/: Omogućuju pregled svih dostupnih API endpoint-a, njihovih parametara i odgovora, pružajući korisnicima i developerima uvid u način korištenja API-ja.

- Konfiguracijski objekt za Swagger i ReDoc: Definira osnovne informacije o API-ju:

```
openapi.Info(
    title="Žargonetka API",
    default_version='v1',
    description="API za aplikaciju Žargonetka",
    contact=openapi.Contact(email="kontakt@zargonetka.hr"),
    license=openapi.License(name="BSD License"),
),
public=True,
permission_classes=(permissions.AllowAny,),
```

7.1.5. Analiza aplikacije base_app

Aplikacija base_app unutar projekta "Žargonetka" sadrži ključne komponente koje omogućuju registraciju korisnika, upravljanje korisničkim sadržajem te interakciju s riječima i frazama.

RegisterSerializer

Datoteka register.py unutar aplikacije base_app sadrži serijalizator nazvan RegisterSerializer, koji je zadužen za registraciju novih korisnika u aplikaciji.

RegisterSerializer nasljeđuje ModelSerializer iz Django REST Frameworka i koristi se za validaciju i obradu podataka prilikom registracije novih korisnika. Uključuje dva polja za unos lozinke (password1 i password2) kako bi se osigurala provjera podudaranja lozinki.

```
Class RegisterSerializer(serializers.ModelSerializer):
        class Meta:
        model = User
        fields = ['username', 'password1', 'password2']
```

- Polja: password1 i password2 su definirana kao CharField s opcijom write_only=True kako bi se osigurala njihova zaštita.

```
        password1 = serializers.CharField(write_only=True)
        password2 = serializers.CharField(write_only=True)
```

- validate metoda: Provjerava jesu li unesene lozinke iste. Ako nisu, podiže grešku (ValidationError) s odgovarajućom porukom: "Passwords do not match".

```
        def validate(self, data):
```

```
        if data['password1'] != data['password2']:
            raise serializers.ValidationError({"password2": "Passwords do not
match"})
        return data
```

- create metoda: Stvara novog korisnika pomoću Django-ove funkcije create_user, koja automatski upravlja enkripcijom lozinke i sprema korisničke podatke u bazu.

```
def create(self, validated_data):
        user = User.objects.create_user(
            username=validated_data['username'],
            password=validated_data['password1']
        )
        return user
```

7.1.5.1. Analiza pogleda

RegisterView

RegisterView nasljeđuje APIView iz Django REST Frameworka i koristi se za obradu zahtjeva za registraciju korisnika.

- post metoda:
  - Prima korisničke podatke putem POST zahtjeva i instancira RegisterSerializer s tim podacima.
    ```
    def post(self, request):
        serializer = RegisterSerializer(data=request.data)
    ```
  - Ako su podaci ispravni, serijalizator kreira novog korisnika u bazi podataka i vraća odgovor s porukom "User registered successfully" i statusnim kodom 201 (Created).
    ```
    if serializer.is_valid():
        serializer.save()
        return Response({"message": "User registered successfully"},
        status=status.HTTP_201_CREATED)
    ```
  - Ako podaci nisu ispravni, vraća se odgovor s greškama i statusnim kodom 400 (Bad Request).
    ```
    return Response(serializer.errors,status=status.HTTP_400_BAD_REQUEST)
    ```

UserCreatedWordsView i UserCreatedPhrasesView

UserCreatedWordsView i UserCreatedPhrasesView nasljeđuju ListAPIView i koriste se za prikaz popisa korisničkih riječi i fraza. Pogledi su u principu isti, razlikuju se po imenima varijabli.

- serializer_class: Definira serijalizator koji će se koristiti. `serializer_class = WordsSerializer`

- permission_classes: Ograničava pristup samo na autentificirane korisnike `permissions.IsAuthenticated`
- get_queryset metoda: Filtrira riječi koje je kreirao trenutno prijavljeni korisnik i koje su odobrene (approved=True).

```
Def get_queryset(self):
       return Words.objects.filter(created_by=self.request.user, approved=True)
```

WordLikesViewSet i PhraseLikesViewSet

WordLikesViewSet i PhraseLikesViewSet nasljeđuju ViewSet i omogućuju korisnicima dohvaćanje popisa omiljenih riječi i fraza:

```
class WordLikesViewSet(viewsets.ViewSet):

    def list(self, request):

        user = request.user

        if not user.is_authenticated:

            return Response({'detail': 'Authentication credentials were not provided.'},

                            status=status.HTTP_401_UNAUTHORIZED)
```

- Provjerava je li korisnik autentificiran; ako nije, vraća status 401 (Unauthorized).

```
        liked_words = user.liked_words.values_list('id', flat=True)

        return Response({'likes': list(liked_words)})
```

- Ako je korisnik autentificiran, dohvaća sve riječi ili fraze koje je korisnik označio kao omiljene (lajkao) i vraća njihov popis u JSON formatu.

7.1.5.2. Analiza urls.py

Datoteka urls.py unutar aplikacije base_app definira URL obrasce za različite API krajnje točke povezane s korisnicima i njihovim interakcijama s riječima i frazama.

Registracija API-a

```
    path('register/', RegisterView.as_view(), name='register'),
```

- Definira krajnju točku za registraciju korisnika. Kada se pošalje POST zahtjev na /register/, koristi se RegisterView za obradu zahtjeva.

```
    path('', include(router.urls)),
```

- Uključuje sve URL obrasce generirane od strane DefaultRouter-a, što omogućuje automatsko upravljanje URL-ovima za registrirane viewsetove (WordLikesViewSet i PhraseLikesViewSet).

Router za lajkove

```
router = DefaultRouter()
```

- Instancira DefaultRouter koji automatski generira URL obrasce za *viewset*-ove.

```
router.register(r'user/word-likes', WordLikesViewSet, basename='word-likes')

router.register(r'user/phrase-likes', PhraseLikesViewSet, basename='phrase-likes')
```

Registriraju se WordLikesViewSet i PhraseLikeViewSet. Ovi viewset-ovi omogućuju dohvaćanje popisa lajkanih riječi.

- Korisnički sadržaj API-a

```
path('user/created-words/',   UserCreatedWordsView.as_view(),   name='user-created-
words'),

path('user/created-phrases/', UserCreatedPhrasesView.as_view(), name='user-created-
phrases'),]
```

Definira se krajnja točka za dohvaćanje riječi i fraze koje je kreirao prijavljeni korisnik. UserCreatedWordsView i UserCreatedPhrasesView se koriste za obradu zahtjeva.

7.1.6. Analiza phrases_app i words_app aplikacija

Aplikacije words_app i phrases_app unutar projekta *Žargonetka* ključne su za upravljanje riječima i frazama. Ove aplikacije koriste Django REST Framework kako bi omogućile učinkovito rukovanje i interakciju s podacima.

words_app: Ova aplikacija fokusira se na upravljanje riječima. Sadrži modele za pohranu informacija o riječima, uključujući njihovo značenje, primjere upotrebe te oznaku odobrenja. Korisnici mogu označiti riječi kao omiljene, a aplikacija omogućuje razne funkcionalnosti poput pretraživanja, sortiranja prema broju lajkova ili datumu kreiranja, te generiranje trivijalnih pitanja za testiranje znanja.

phrases_app: Ova aplikacija upravlja frazama i slična je words_app, ali je prilagođena za rad s frazama umjesto pojedinačnih riječi. Sadrži modele za pohranu fraza, njihovih značenja i primjera upotrebe. Također podržava označavanje fraza kao omiljenih, pretraživanje i sortiranje prema različitim kriterijima, te generiranje trivijalnih pitanja.

7.1.6.1. Modeli: Words i Phrases

word / phrase (CharField):

- Words: Polje word pohranjuje tekstualnu vrijednost riječi. Maksimalna duljina ovog polja je 50 znakova, a vrijednosti moraju biti jedinstvene unutar baze podataka.

```
class Words(models.Model):
    word = models.CharField(max_length=50, unique=True)
```

- Phrases: Polje phrase pohranjuje frazu, s maksimalnom duljinom od 255 znakova, koja također mora biti jedinstvena.

```
class Phrases(models.Model):
    phrase = models.CharField(max_length=255, unique=True)
```

word_meaning / phrase_meaning (TextField):

Polje sadrži opis značenja riječi/fraze. Ovo je obavezno polje koje ne smije biti prazno.

word_example / phrase_example (TextField):

Polje pruža primjer upotrebe riječi/fraze. Nije obavezno, što znači da može biti prazno.

approved (BooleanField):

Oba modela koriste polje approved za označavanje je li unos odobren. Po zadanim postavkama, ovo polje je postavljeno na False.

created_by (ForeignKey):

Modeli Words i Phrases povezuju unos s korisnikom koji je kreirao podatke. Ova relacija koristi on_delete=models.CASCADE, što znači da će svi povezani unosi biti obrisani ako se korisnik obriše.

```
created_by = models.ForeignKey(User, on_delete=models.CASCADE)
```

likes (ManyToManyField):

Polje likes pohranjuje korisnike koji su označili riječ/frazu kao omiljenu, omogućujući praćenje broja lajkova.

```
likes= models.ManyToManyField(User, related_name='liked_words', blank=True)
```

created_at (DateTimeField):

Oba modela imaju polje created_at koje automatski pohranjuje datum i vrijeme kada je unos stvoren, što pomaže u praćenju kada su podaci dodani.

```
created_at = models.DateTimeField(auto_now_add=True, null=True)
```

Metode:

- __str__: Vraća tekstualnu reprezentaciju unosa koja je korisna za prikaz u administrativnim sučeljima. Za model Words, metoda vraća riječ, dok za Phrases, vraća frazu.
- like_count: Broji ukupan broj lajkova za riječ ili frazu, pružajući uvid u njihovu popularnost među korisnicima.

```
def __str__(self):
        return self.word
    def like_count(self):
        return self.likes.count()
```

7.1.6.2. Views

U aplikaciji *Žargonetka*, WordsViewSet i PhrasesViewSet nude slične funkcionalnosti za upravljanje riječima i frazama. Oba viewset-a koriste Django REST Framework za upravljanje podacima, ali se odnose na različite vrste entiteta.

Get_queryset:

Ova metoda get_queryset unutar WordsViewSet klase služi za dohvaćanje skupa podataka (*queryset*) riječi iz baze podataka, uz primjenu različitih kriterija filtriranja i sortiranja. Metoda omogućuje fleksibilnost u prikazu podataka.

```
def get_queryset(self):
    user = self.request.user
    sort_option = self.request.query_params.get('sort', 'date')
```

- Dohvaćanje korisnika i sortiranja:
  - user = self.request.user: Dohvaća trenutno prijavljenog korisnika.
  - sort_option = self.request.query_params.get('sort', 'date'): Dohvaća kriterij sortiranja iz URL parametara (sort). Ako kriterij nije definiran, zadana vrijednost je 'date'.
- Inicijalizacija skupa podataka (queryset):
  - queryset = Words.objects.all(): Inicijalizira skup podataka sa svim riječima.
- Filtriranje po odobrenju:

```
if user.is_staff:
        queryset = Words.objects.all()
    else:
        queryset = Words.objects.filter(approved=True)
```

- Ako korisnik nije administrator (not user.is_staff), metoda filtrira riječi koje su odobrene (approved=True). Administratori vide sve riječi.

- Sortiranje prema različitim opcijama:

```
if sort_option == 'likes':
           queryset = queryset.annotate(like_count=Count('likes')).order_by('-like_count')
        elif sort_option == 'date':
            queryset = queryset.order_by('-created_at')  # Sortiraj po datumu kreiranja
        elif sort_option == 'alphabetical':
            queryset = queryset.order_by('word')  # Sortiranje abecedno
        else:
            queryset = queryset.order_by('-created_at')  # Defaultno sortiranje
        return queryset
```

  - Po broju lajkova (likes): Ako je odabran kriterij 'likes', koristi se annotate za brojanje lajkova (like_count=Count('likes')) i order_by za sortiranje u padajućem redoslijedu prema broju lajkova.
  - Po datumu (date): Ako je odabran kriterij 'date', riječi se sortiraju prema datumu kreiranja u padajućem redoslijedu (order_by('-created_at')).
  - Abecedno (alphabetical): Ako je odabran kriterij 'alphabetical', riječi se sortiraju abecedno (order_by('word')).
  - Zadano sortiranje: Ako nijedan od navedenih kriterija nije zadovoljen, koristi se zadano sortiranje prema datumu kreiranja u padajućem redoslijedu.

Perform_create:
Funkcija koja postavlja created_by polje na trenutno prijavljenog korisnika prilikom stvaranja nove riječi:

```
def perform_create(self, serializer):
        serializer.save(created_by=self.request.user)
```

random:

Vraća nasumičnu odobrenu riječ iz baze podataka. Ako nema odobrenih riječi, vraća poruku o grešci sa statusom 404.

- Dekorator @action:
  - @action(detail=False, methods=['get']): Ovaj dekorator definira prilagođenu akciju koja se ne odnosi na pojedinačan objekt (detail=False) i odgovara na HTTP GET zahtjeve.

- Dohvaćanje odobrenih riječi:
    - approved_words = self.get_queryset(): Metoda koristi get_queryset() kako bi dohvatila skup podataka koji sadrži sve odobrene riječi, ovisno o korisničkim privilegijama.
- Provjera postojanja odobrenih riječi:
    - if not approved_words.exists(): Provjerava postoji li barem jedna odobrena riječ. Ako nema odobrenih riječi, vraća HTTP odgovor sa statusom 404 (nije pronađeno) i odgovarajuću poruku.
- Odabir nasumične riječi:
    - random_word = random.choice(approved_words): Odabire nasumičnu riječ iz skupa odobrenih riječi koristeći Pythonovu ugrađenu funkciju random.choice().
- Serijalizacija i vraćanje odgovora:
    - serializer = self.get_serializer(random_word): Serijalizira nasumično odabranu riječ koristeći odgovarajući serializer definiran za WordsViewSet.
    - return Response(serializer.data): Vraća serijalizirane podatke kao HTTP odgovor.

Trivia:

Vraća nasumično odabranu riječ i tri druge riječi za koje se traži od korisnika da odabere ispravan odgovor. To se koristi za stvaranje trivijalnih pitanja s jednim točnim odgovorom i tri netočna odgovora.

Kao i u prethodnim primjerima, @action(detail=False, methods=['get']) dekorator označava da je trivia prilagođena akcija koja odgovara na GET zahtjeve, a nije vezana za pojedinačan objekt.

```
@action(detail=False, methods=['get'])
```

Definicija funkcije:

Dohvaćanje odobrenih riječi: Metoda prvo dobiva sve odobrene riječi iz baze podataka pomoću get_queryset() i pretvara ih u popis.

```
approved_words = list(self.get_queryset())
```

- Provjera postojanja riječi: Ako nema odobrenih riječi, vraća odgovor s porukom "*No approved words found.*" i statusnim kodom 404.

```
if not approved_words:

 return          Response({'detail':          'No          approved          words
found.'}status=status.HTTP_404_NOT_FOUND)
```

- Odabir nasumične riječi: Ako postoje odobrene riječi, metoda koristi random.choice za odabir jedne nasumične riječi kao točnog odgovora.

```
 random_word = random.choice(approved_words)
```

- Odabir netočnih odgovora: Generiraju se tri netočna odgovora tako da se nasumično biraju tri riječi iz baze podataka koje nisu identične s točnim odgovorom, koristeći random.sample.

```
other_words = random.sample([word for word in approved_words if word != random_word],
3
```

- Kreiranje liste odgovora: Kreira se popis odgovora, pri čemu se točan odgovor označava s 'is_correct': *True*, a netočni odgovori s 'is_correct': *False*.

```
answers = [

            {'text': random_word.word_meaning, 'is_correct': True},

            {'text': other_words[0].word_meaning, 'is_correct': False},

            {'text': other_words[1].word_meaning, 'is_correct': False},

            {'text': other_words[2].word_meaning, 'is_correct': False},

        ]
```

- Vraćanje odgovora: Metoda vraća odgovor u JSON formatu koji sadrži riječ ('word') i moguće odgovore ('answers').

```
return Response({

            'word': random_word.word,

            'answers': answers
```

Like:

Omogućuje korisnicima da označe riječ kao omiljenu ili da je uklone iz svojih omiljenih riječi. Ako je riječ već označena kao omiljena, uklanja oznaku, inače je dodaje. Ako korisnik nije prijavljen, vraća poruku o grešci sa statusom 401.

- Dohvaćanje objekta riječi: Metoda koristi self.get_object() kako bi dohvatila određenu riječ iz baze podataka koristeći primarni ključ (pk).

```
 word = self.get_object()
```

- Provjera autentifikacije: Metoda prvo provjerava je li korisnik autentificiran koristeći user.is_authenticated.
- Provjera je li riječ već "lajkana": Ako je korisnik autentificiran, provjerava se je li korisnik već "lajkao" tu riječ:

  ```
  if word.likes.filter(id=user.id).exists()
  ```

- Ako je korisnik već 'lajkao' riječ (word.likes.filter(id=user.id).exists()), metoda uklanja korisnika iz popisa korisnika koji su "lajkali" tu riječ i vraća odgovor s porukom *'Unliked'*.
- Ako korisnik nije 'lajkao' riječ, dodaje se korisnik u popis korisnika koji su 'lajkali' riječ i vraća se odgovor s porukom *'Liked'*.
- Neautentificirani korisnik: Ako korisnik nije autentificiran, vraća se odgovor s porukom „*Authentication credentials were not provided.“* i statusnim kodom 401.

Top10:

Vraća 10 najpopularnijih odobrenih riječi, sortirane prema broju lajkova, od najviše prema najmanje. Koristi annotate za brojanje lajkova i order_by za sortiranje.

- Dohvaćanje najpopularnijih riječi:
  - Words.objects.filter(approved=True): Filtrira riječi koje su odobrene (approved=True).
  - .annotate(like_count=Count('likes')): Dodaje anotaciju like_count koja sadrži broj lajkova za svaku riječ.
  - .order_by('-like_count')[:10]: Sortira riječi prema broju lajkova u silaznom redoslijedu i ograničava rezultat na prvih 10 riječi.
- Serijalizacija i vraćanje odgovora:
  - serializer = self.get_serializer(top_words, many=True): Serijalizira skup podataka top_words koristeći odgovarajući serializer. Parametar many=True koristi se jer serijaliziramo više objekata.
  - return Response(serializer.data): Vraća serijalizirane podatke kao HTTP odgovor.

Search:

Omogućuje pretraživanje riječi prema dijelu teksta. Koristi icontains za pretraživanje riječi koje sadrže uneseni pojam i vraća sve odgovarajuće rezultate.

- Dohvaćanje pojma pretraživanja:

- query = request.query_params.get('search', ''): Dohvaća pretraživački pojam iz URL parametara. Ako pojam nije naveden, koristi prazni string kao zadanu vrijednost.

- Filtriranje riječi:
  - Words.objects.filter(approved=True, word__icontains=query): Filtrira sve odobrene riječi (approved=True) koje sadrže pretraživački pojam unutar polja word. Korištenje icontains omogućuje pretraživanje koje nije osjetljivo na velika i mala slova.

- Serijalizacija i vraćanje odgovora:
  - serializer = self.get_serializer(words, many=True): Serijalizira skup podataka words koristeći odgovarajući serializer. Parametar many=True označava da se serijaliziraju višestruki objekti.
  - return Response(serializer.data): Vraća serijalizirane podatke kao HTTP odgovor.

PhrasesViewSet

Kod PhrasesViewSet-a nije potrebno dodatno analizirati jer su WordsViewSet i PhrasesViewSet vrlo slični u svojoj strukturi i funkcionalnosti. Oba viewseta naslijeđena su od viewsets.ModelViewSet i koriste Django REST Framework za upravljanje podacima preko API-ja. Imaju slične metode za filtriranje, sortiranje i prilagođene akcije poput random, trivia, like, top10, i search. Također, oba viewset-a koriste specifične serializere (WordsSerializer i PhrasesSerializer) za serijalizaciju podataka i postavljanje korisnika koji stvara objekte. Jedina razlika između dva pogleda je naziv varijabla.

7.1.6.3. Urls.py

U aplikacijama phrases_app i words_app, datoteke urls.py koriste Django REST Framework-ov DefaultRouter za automatizirano upravljanje URL putanjama i API funkcionalnostima.

```
router = DefaultRouter()

router.register(r'', PhrasesViewSet, basename='phrases')

urlpatterns = [

    path('', include(router.urls)),

    path('swagger/', schema_view.with_ui('swagger', cache_timeout=0), name='schema-swagger-
ui'),

    path('redoc/', schema_view.with_ui('redoc', cache_timeout=0), name='schema-redoc'),
```

```
]
```

DefaultRouter registrira PhrasesViewSet, što omogućuje automatsko generiranje URL putanja za CRUD operacije nad frazama. Također, URL putanje za Swagger i ReDoc dokumentaciju omogućuju interaktivni pregled i testiranje API-ja, čime se olakšava razvoj i integracija.

Slično kao u phrases_app, DefaultRouter u words_app registrira WordsViewSet za automatsko generiranje URL putanja za CRUD operacije nad riječima. URL putanje za Swagger i ReDoc dokumentaciju također su prisutne kako bi se omogućio pregled i testiranje API-ja.

7.2. Frontend

7.2.1. Postavljanje Radnog Okruženja za React Aplikaciju

Za razvoj react aplikacije potrebno je instalirati node.js, koji uključuje npm (Node Package Manager), alat za upravljanje paketima.

```
npx create-react-app naziv-aplikacije
```

Ova naredba kreira novi direktorij s potrebnim datotekama i konfiguracijama. Unutar direktorija src nalaze se ključne datoteke poput App.js i index.js, koje su temelj aplikacije. Da bi se pokrenula aplikaciju u razvojnom načinu rada, koristi se npm start.

7.2.2. Ključne Datoteke i Direktoriji

- /src: Glavni direktorij za izvornu kodnu bazu React aplikacije.
- /components: Sadrži sve React komponente aplikacije. Svaka komponenta obično ima svoj vlastiti direktorij s JavaScript/JSX datotekama.
- /auth: Komponente vezane za autentifikaciju, poput prijave i registracije.
- /search: Komponente za pretraživanje, uključujući SearchBar i SearchResults.
- AddPhrase.js, AddWord.js, Home.js, ...: Specifične komponente za dodavanje fraza i riječi, početnu stranicu i druge funkcionalnosti.
- /context: Sadrži kontekstne datoteke koje omogućuju globalno upravljanje stanjem aplikacije koristeći React Context API.
- /styles: Stilovi aplikacije, obično u CSS datotekama.
- /src/App.js: Glavna komponenta aplikacije koja često služi kao korijenska komponenta za sve ostale komponente.
- /src/index.js: Ulazna točka aplikacije gdje se React aplikacija "priključuje" na DOM, obično u index.html datoteci.

- .env: Datoteka za konfiguraciju varijabli okoline. Koristi se za pohranu osjetljivih informacija i konfiguracija koje aplikacija koristi.
- package.json: Datoteka koja sadrži informacije o projektu i njegove zavisnosti, uključujući skripte za izgradnju i pokretanje aplikacije.
- package-lock.json: Automatski generirana datoteka koja osigurava da svi članovi tima koriste iste verzije paketa definiranih u package.json.

7.2.3. App.js

U datoteci App.js, postavljamo osnovnu strukturu React aplikacije koristeći react-router-dom za upravljanje navigacijom. Ključne komponente i njihove funkcije su:

- Router: Omogućuje navigaciju između različitih stranica aplikacije.
- Routes: Definira sve putanje (*Route*) koje aplikacija podržava i povezuje ih s odgovarajućim komponentama.
- Rute: Svaka ruta predstavlja putanju URL-a i komponentu koja se prikazuje kada korisnik posjeti tu putanju. Na primjer:
    - Ruta /words prikazuje komponentu Words.
    - Ruta /phrases/:id omogućuje prikaz detalja fraze prema ID-u.

Pored toga, koristimo AuthProvider i SearchProvider:

- AuthProvider: Upravljanje autentifikacijom korisnika, omogućujući pristup korisničkim informacijama i sesijama unutar aplikacije.
- SearchProvider: Omogućuje upravljanje i dijeljenje konteksta pretraživanja između različitih komponenti.

```
function App() {
  return (
    <AuthProvider>
      <SearchProvider>
        <Router>
          <div>
            <Navbar />
            <Routes>
              <Route path="/" element={<Home />} />
              <Route path="/words" element={<Words />} />
              <Route path="/phrases" element={<Phrases />} />
               ...
            </Routes>
          </div>
        </Router>
      </SearchProvider>
    </AuthProvider>
  );
}

export default App;
```

7.2.4. Index.js

Datoteka index.js u React aplikaciji služi kao ulazna točka za aplikaciju i postavlja osnovnu strukturu za renderiranje React komponenti u pregledniku. Koristi ReactDOM.createRoot za inicijalizaciju korijenskog elementa u DOM-u s ID-jem root-a, te renderira aplikaciju unutar tog korijenskog elementa koristeći root.render. Aplikacija se prikazuje unutar <React.StrictMode>, što omogućava dodatne provjere i upozorenja tijekom razvoja.

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
        <React.StrictMode>
        <App />
        </React.StrictMode>
              );
```

7.2.5. Home.js

Datoteka Home.js predstavlja React komponentu koja se koristi za prikazivanje glavne stranice aplikacije. Evo kako funkcionalnost komponente Home izgleda:

- Inicijalizacija **s**tanja**:**
    - Koristi useState za definiranje stanja koje uključuje topWords, topPhrases i showMessage, dok useLocation iz react-router-dom se koristi za pristup podacima iz URL-a,

```
const [topWords, setTopWords] = useState([]);
  const [topPhrases, setTopPhrases] = useState([]);
  const [showMessage, setShowMessage] = useState(false);
  const location = useLocation();
  const message = location.state?.message;
```

    - Funkcija za **f**ormatiranje **d**atuma:
        - formatDate funkcija pretvara datum u lokalni format. Ako datum nije prisutan, vraća "nepoznato".

```
const formatDate = (date) => {
    return date ? new Date(date).toLocaleDateString() : "nepoznato";
  };
```

- Učita**vanje p**od**ataka** s API-ja:
    - useEffect se koristi za dohvaćanje podataka kada se komponenta učita.
    - Poziva API za dohvaćanje top 10 riječi i postavlja ih u stanje topWords.
    - Poziva API za dohvaćanje top 10 fraza i postavlja ih u stanje topPhrases.
    - U slučaju pogreške tijekom dohvaćanja podataka, ispisuje se poruka u konzolu.

```
useEffect(() => {
    axios.get(`${process.env.REACT_APP_API_URL}/words/top10/`)
      .then(response => {
```

```
        setTopWords(response.data);
      })
      .catch(error => {
        console.error('There was an error fetching the top words!', error);
      });
    axios.get(`${process.env.REACT_APP_API_URL}/phrases/top10/`)
      .then(response => {
        setTopPhrases(response.data);
      })
      .catch(error => {
        console.error('There was an error fetching the top phrases!', error);
      });
  }, []);
```

- Prikaz **p**oruka:

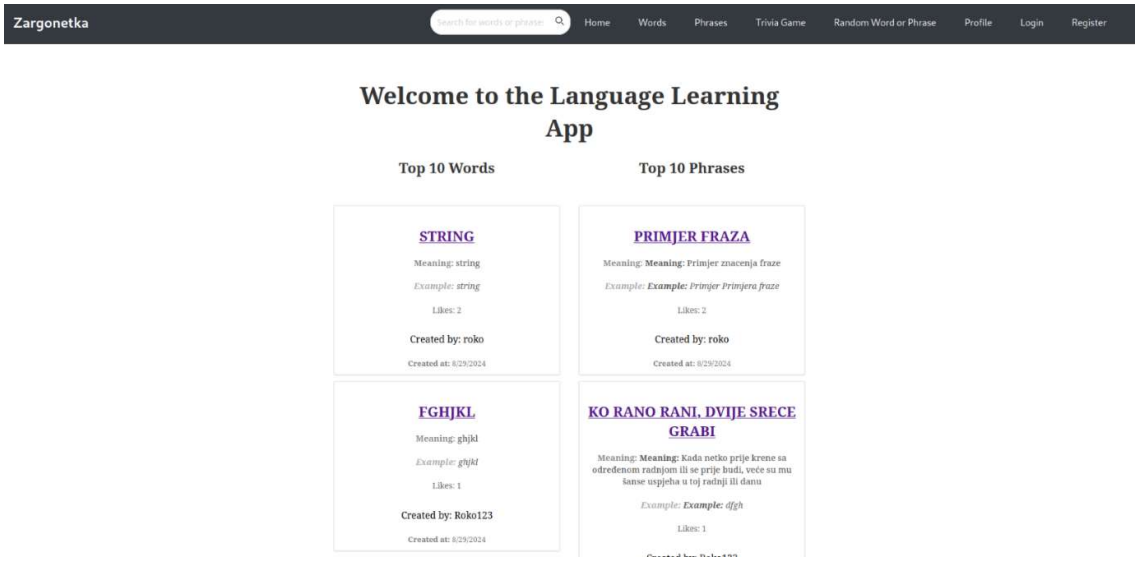
    - Drugi useEffect se koristi za upravljanje prikazom poruka.
    - Ako je message prisutan, postavlja showMessage na true i koristi setTimeout za automatsko sakrivanje poruke nakon 5 sekundi.
    - Čisti timer kada se komponenta demontira ili message promijeni.

```
useEffect(() => {
    if (message) {
      setShowMessage(true);
      const timer = setTimeout(() => {
        setShowMessage(false);
      }, 5000);       return () => clearTimeout(timer);  timer
    }
  }, [message]);
```

- Renderiranje **k**omponente:

- Prikazuje *pop-up* poruku ako je showMessage *true*.
- Renderira naslov i podnaslove za top 10 riječi i fraza.
- Prikazuje popis riječi i fraza:
    - Ako topWords ili topPhrases sadrže podatke, prikazuje ih u obliku listi. Svaka stavka uključuje poveznicu na detalje, značenje, primjer, broj lajkova, autora i datum kreiranja.
    - Ako nema dostupnih riječi ili fraza, prikazuje odgovarajuću poruku.

```
<div className="item-list">
  {topWords.length > 0 ? (
    topWords.map(word => (
      <div className="item-box" key={word.id}>
        <h3 className="item-title">
          <Link to={`/words/${word.id}`}>{word.word}</Link>
        </h3>
        <p className="item-meaning">{word.word_meaning}</p>
        ...
```



Slika 2. Prikaz Home stranice

Na Slici 2. prikazana je renderirana Home kompontenta.

7.2.6. Navbar

Datoteka Navbar.js definira navigacijsku traku za aplikaciju. Ova komponenta omogućava navigaciju kroz aplikaciju i prilagođava prikaz izbornika ovisno o prijavi korisnika i veličini ekrana.

- Stanje i **k**ontekst:
    - Koristi useContext za provjeru je li korisnik prijavljen (isAuthenticated).
    - Koristi useState za upravljanje otvorenjem/zatvaranjem hamburger izbornika.

```
const { isAuthenticated } = useContext(AuthContext);
const [isOpen, setIsOpen] = useState(false);
```

- Hamburger **i**zbornik:
    - handleToggle funkcija prebacuje stanje izbornika između otvorenog i zatvorenog.
    - Prikazuje hamburger ikonu za mobilne uređaje.

```
const handleToggle = () => {
    setIsOpen(!isOpen);
```

- Prikaz **s**adržaja:
    - Sadrži poveznice za navigaciju (Home, Words, Phrases, itd.).

```
  </li>
          <li className="nav-item">
            <Link     className="nav-link"     to="/"     onClick={()     =>
setIsOpen(false)}>Home</Link>
              ....
```

- Uključuje SearchBar unutar izbornika za mobilne uređaje.
- Ako je korisnik prijavljen, prikazuje LogoutButton. Ako nije, prikazuje poveznice za prijavu i registraciju.

```
                {!isAuthenticated ? (
            <>
              <li className="nav-item">
                <Link     className="nav-link"     to="/login"     onClick={()     =>
setIsOpen(false)}>Login</Link>
              </li>
              <li className="nav-item">
                <Link    className="nav-link"    to="/register"    onClick={()    =>
setIsOpen(false)}>Register</Link>
              </li>
            </>
          ) : (
            <li className="nav-item">
              <LogoutButton  onClick={()  =>  setIsOpen(false)}  />  {/*  Show  logout
button if authenticated */}
            </li>
```

7.2.7. Phrases i Words

Komponente Words i Phrases imaju sličnu funkcionalnost s malim razlikama u detaljima. Ove komponente služe za prikaz popisa riječi i fraza u aplikaciji te omogućuju korisnicima da pregledavaju, sortiraju i lajkaju stavke.

- Stanje i **u**voz:
    - Koriste useState za upravljanje stanjem podataka kao što su popis riječi/fraza, trenutna stranica, ukupni broj stranica, opcija sortiranja i lajkovi korisnika.
    - useContext koristi AuthContext za provjeru je li korisnik prijavljen.

```
const fetchPhrases = async () => {
      try {
        const                response                =                await
axios.get(`${process.env.REACT_APP_API_URL}/phrases/`, {
          params: {
            page: currentPage,
            sort: sortOption
```

- Dohvaćanje **p**odataka:
    - Obje komponente koriste useEffect za dohvaćanje podataka s API-ja pri učitavanju ili promjeni stranice/sortiranja:
        - Words dohvaća popis riječi i informacije o lajkovima korisnika.
        - Phrases dohvaća popis fraza i lajkovima korisnika.
    - API pozivi su prilagođeni s obzirom na opciju sortiranja i stranicu.

```
useEffect(() => {
```

```
    const fetchPhrases = async () => {
      try {
        const                    response                    =                    await
axios.get(`${process.env.REACT_APP_API_URL}/phrases/`, {
          params: {
            page: currentPage,
            sort: sortOption // Proslijedite opciju sortiranja
          }
        });
        setPhrases(response.data.results);
        setTotalPages(Math.ceil(response.data.count / 10));
      } catch (error) {
        console.error('There was an error fetching the phrases!', error);
      }
    };

    fetchPhrases();
     }, [currentPage, sortOption]);
```

- Upravljanje **l**ajkovima:
  - Obje komponente omogućuju korisnicima da lajkaju stavke (riječi ili fraze) ako su prijavljeni.
    ```
    const handleLike = async (phraseId) => {
        if (!isAuthenticated) {
          console.error('User must be logged in to like a phrase.');
          return;
        }
    ```
  - handleLike metoda šalje POST zahtjev za lajk, ažurira stanje riječi ili fraza i stanje lajkova korisnika:
    - Ako je stavka već lajkana, ikona se mijenja u odgovarajuću (thumbs down za odabir).
    - Ako korisnik nije prijavljen, ispisuje se pogreška u konzolu.
    ```
    setUserLikes(prevLikes => {
            const updatedLikes = new Set(prevLikes);
            if (response.data.status === 'Liked') {
              updatedLikes.add(phraseId);
            } else {
              updatedLikes.delete(phraseId);
            }
            return updatedLikes;
          });
    ```

- Sortiranje i **p**aginacija:
  - Koriste SortSelector komponentu za promjenu opcije sortiranja. Kada se opcija promijeni, stranica se resetira na 1.
  - Implementiraju paginaciju kroz prikaz gumba za promjenu stranica.
    ```
    const handlePageChange = (pageNumber) => {
        setCurrentPage(pageNumber);
      };

      const handleSortChange = (sortOption) => {
    ```

```
    setSortOption(sortOption);
    setCurrentPage(1); // Resetirajte stranicu na 1 kad se sortira
  };
```

- Renderiranje:
    - Prikazuju popis riječi ili fraza s pripadajućim informacijama (naslov, značenje, primjer, broj lajkova, autor, datum kreiranja).
    - Ako je korisnik prijavljen, prikazuju se opcije za dodavanje novih stavki (Add Word ili Add Phrase) te mogućnost lajkanja stavki.
    - Za svaku stavku, uključuje se poveznica na detalje i ikona za lajkanje koja reagira na klikove korisnika.

```
{isAuthenticated && (<div className="like-icon-container" onClick={() => handleLike(phrase.id)}>

    <FontAwesomeIcon

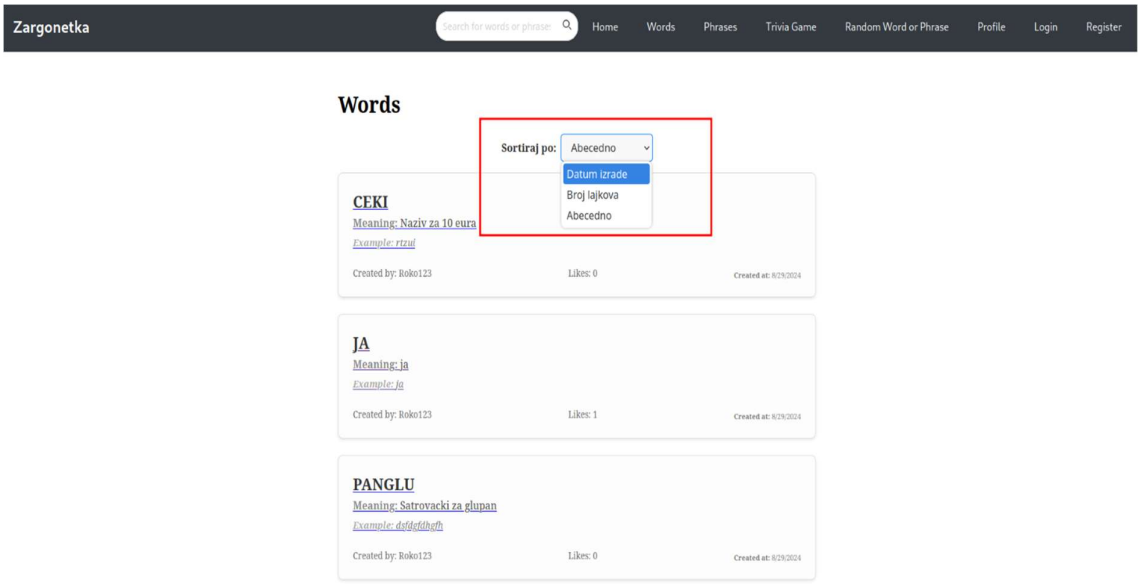
      icon={userLikes.has(phrase.id) ? faThumbsDown : faThumbsUp}

      className={`like-icon ${userLikes.has(phrase.id) ? 'liked' : ''}`}/>
```

Razlike:

- Komponenta Words i Phrases razlikuju se u URL-ovima za API pozive (/words/ vs. /phrases/) i u pripadajućim CSS datotekama.
- Phrases komponenta ne koristi funkciju formatDate unutar useEffect, dok Words koristi ovu funkciju za formatiranje datuma.

Na slici 3. prikazana je Word komponenta te je označen SortSelector crvenim kvadratom.



Slika 3. Prikaz Words komponente i sortiranja

7.2.8. Komponente addPhrase i addWord

Komponente AddPhrase i AddWord omogućuju korisnicima da dodaju nove fraze ili riječi u aplikaciju. Iako su vrlo slične u strukturi i funkcionalnosti, postoji nekoliko ključnih razlika. Evo sažetka njihovih glavnih funkcionalnosti:

Stanje i **u**voz:

- Obje komponente koriste useState za upravljanje lokalnim stanjem obrazaca (phrase/word, phraseMeaning/wordMeaning, phraseExample/wordExample).
- Koriste useContext za provjeru je li korisnik prijavljen (AuthContext).
- useNavigate se koristi za navigaciju nakon uspješnog dodavanja stavke.

Obrada **o**brazaca:

AddPhrase:

- Prikupi podatke o frazi, značenju i primjeru te ih pošalje putem POST zahtjeva na /phrases/ endpoint. Koristi user.id za označavanje autora fraze.

AddWord:

- Prikupi podatke o riječi, značenju i primjeru te ih pošalje putem POST zahtjeva na /words/ endpoint. Koristi podatke o korisniku kao autora riječi.

```
const newPhrase = {
phrase: phrase,
phrase_meaning: phraseMeaning,
phrase_example: phraseExample,
```

```
created_by: user.id
};

const config = {
headers: {
'Authorization': `Bearer ${localStorage.getItem('access')}`
}
};
```

Provjera **a**utentifikacije:

- AddPhrase/AddWord: Provjerava user iz AuthContext da osigura da je korisnik prijavljen. Ako korisnik nije prijavljen, ispisuje pogrešku u konzolu.

Slanje **p**odataka:

- Obje komponente šalju POST zahtjeve s pripadajućim podacima:
    - AddPhrase: axios.post šalje podatke na /phrases/ uz autentičnost tokena u zaglavlju.
    - AddWord: axios.post šalje podatke na /words/ uz autentičnost tokena u zaglavlju.

Navigacija i **p**ovratna **i**nformacija:

- Nakon uspješnog dodavanja fraze ili riječi, obje komponente navigiraju korisnika na početnu stranicu ('/') s porukom o uspješnom dodavanju.
- AddPhrase: Koristi poruku 'Fraza uspješno dodana, admin će je uskoro provjeriti.'
- AddWord: Koristi poruku 'Riječ uspješno dodana, admin će je uskoro provjeriti.'

Renderiranje:

- Obje komponente renderiraju obrazac za unos sa sljedećim poljima:
    - AddPhrase: Polja za frazu, značenje i primjer.
    - AddWord: Polja za riječ, značenje i primjer.
- Stilizacija obrazaca se razlikuje prema pripadajućim CSS klasama (add-phrase za fraze i add-word za riječi).

7.2.9. Auth/

Ove četiri komponente upravljaju autentifikacijom i registracijom korisnika u aplikaciji. Evo sažetka njihove funkcionalnosti i strukture:

AuthContext

- AuthProvider služi za upravljanje autentifikacijom korisnika u React aplikaciji. Koristi createContext za pružanje informacija o autentifikaciji korisnicima aplikacije i useState

te useEffect za upravljanje stanjima prijave korisnika i spremanje podataka u localStorage.

```
const AuthContext = createContext();
export function AuthProvider({ children }) {
  const [isAuthenticated, setIsAuthenticated] = useState(false);
  const [user, setUser] = useState(null);
  useEffect(() => {
    const token = localStorage.getItem('access');
    if (token) {
      setIsAuthenticated(true);
      setUser(jwtDecode(token));
    } else {
      setIsAuthenticated(false);
      setUser(null);
    }
  }, []);
```

- AuthProvider Komponenta: Ova komponenta koristi useState za upravljanje stanjem autentifikacije (isAuthenticated) i korisničkim podacima (user).
    - useEffect: Provjerava postoji li token u localStorage kada se komponenta prvi put učita. Ako postoji, postavlja stanje kao autentificirano i dekodira korisničke podatke iz tokena.
    - login: Funkcija koja postavlja token u localStorage, ažurira stanje autentifikacije i postavlja korisničke podatke iz tokena.
    - logout: Funkcija koja uklanja token iz localStorage i resetira stanje autentifikacije i korisničke podatke.

```
  const login = (token) => {
    localStorage.setItem('access', token);
    setIsAuthenticated(true);
    setUser(jwtDecode(token));
  };

  const logout = () => {
    localStorage.removeItem('access');
    setIsAuthenticated(false);
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ isAuthenticated, user, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
}

export default AuthContext;
```

LoginForm

- Stanje: Upravljanje lokalnim stanjem za korisničko ime (username) i lozinku (password).
- handleLogin: Funkcija koja obrađuje prijavu. Kada se obrazac pošalje, šalje POST zahtjev na /token/ s korisničkim imenom i lozinkom.
    - Ako je prijava uspješna, poziva login funkciju iz AuthContext da postavi token i informacije o korisniku, a zatim navigira na početnu stranicu.
    - Ako dođe do greške, ispisuje grešku u konzolu (moguće dodati prikaz greške na korisničkom sučelju).

Ova komponenta omogućuje korisnicima da se prijave u aplikaciju.

RegisterForm

- Stanje: Upravljanje lokalnim stanjem za korisničko ime (username), lozinke (password1 i password2), i grešku (error).

```
function RegisterForm() {

  const [username, setUsername] = useState('');

  const [password1, setPassword1] = useState('');

  const [password2, setPassword2] = useState('');

  const [error, setError] = useState(null);

  const navigate = useNavigate();
```

- handleRegister: Funkcija koja obrađuje registraciju. Kada se obrazac pošalje, šalje POST zahtjev na /accounts/register/ s korisničkim imenom i lozinkama.
    - Ako je registracija uspješna, ispisuje uspješnu poruku i navigira na stranicu za prijavu.
    - Ako dođe do greške, postavlja grešku u lokalnom stanju koja se zatim prikazuje na obrascu.

Ova komponenta omogućuje korisnicima da se registriraju račun u aplikaciji:

```
const handleRegister = async (e) => {

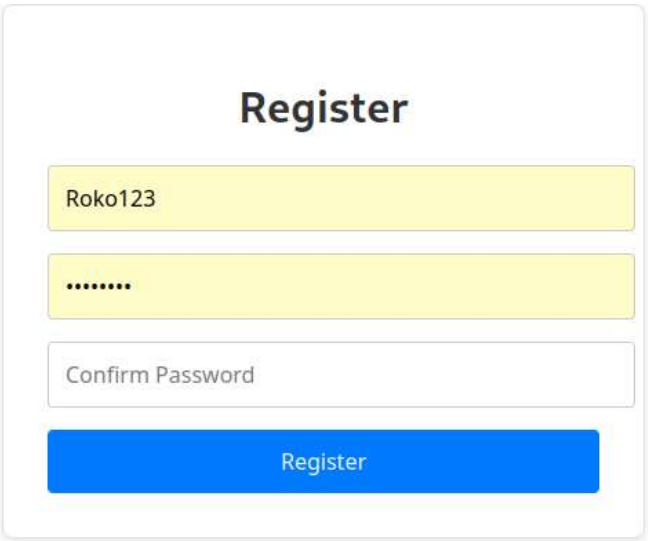
    e.preventDefault();

    setError(null)

    try {

      const response = await axios.post(
```

```
      `${process.env.REACT_APP_API_URL}/accounts/register/`,
      { username, password1, password2 }
    );
    console.log('Registration successful:', response.data);
    navigate('/login');
  } catch (error) {
    if (error.response)
      console.error('Server error:', error.response.data);
      setError(error.response.data);
    } ...
          return ( <form onSubmit={handleRegister} className="register-form">
```



Slika 4. Prikaz forme za registriranje korisnika

Slika 4. prikazuje formu za registriranje, dok je forma za prijavu jednaka no bez potvrde lozinke.

LogoutButton

- handleLogout: Funkcija koja poziva logout funkciju iz AuthContext za uklanjanje tokena i resetiranje stanja autentifikacije, a zatim navigira na početnu stranicu.
- Renderiranje: Prikazuje dugme koje, kada se klikne, izvršava funkciju za odjavu.

```
function LogoutButton() {
  const { logout } = useContext(AuthContext);
  const navigate = useNavigate();
  const handleLogout = () => {
    logout();    navigate('/');
  };
  return (
    <button onClick={handleLogout} className="logout-button">
      Logout
    </button>
  );
}
export default LogoutButton;
```

7.2.10. WordDetail i PhraseDetail

- Namijenjene su za prikaz detaljnih informacija o riječima i frazama u React aplikaciji.
- Iako se bave različitim vrstama podataka (riječi i fraze), komponente imaju sličnu strukturu i funkcionalnost.

Upotreba useParams:

- Obje komponente koriste useParams iz react-router-dom za dohvat id-a iz URL-a.
- Ovaj id se koristi za specifikaciju resursa koji se traži na serveru (riječ ili fraza).

```
function WordDetail() {
  const { id } = useParams();
```

Korištenje axios i useEffect:

axios: Korišten za slanje GET zahtjeva prema API-ju kako bi se dohvatili detalji o specifičnom resursu.

- WordDetail šalje zahtjev na /words/${id}/.
- PhraseDetail šalje zahtjev na /phrases/${id}/.
- useEffect: Korišten za učitavanje podataka prilikom prvotnog prikazivanja komponente ili kada se id promijeni.

- Ako zahtjev uspije, podaci se spremaju u stanje komponente.
- Ako dođe do greške, ispisuje se poruka o grešci u konzolu.

- Prikaz dok se podaci učitavaju:
  - Ako podaci nisu dostupni (tj. word ili phrase su null), komponente prikazuju poruku „Loading...“.

```
useEffect(() => {
    axios.get(`${process.env.REACT_APP_API_URL}/words/${id}/`)
      .then(response => {
        setWord(response.data);
      })
      .catch(error => {
        console.error('There was an error fetching the word details!', error);
      });
  }, [id]);
  if (!word) return <p>Loading...</p>;
```

Obrada podataka

- Stanje i funkcije:
  - Stanje (useState):
    - WordDetail koristi stanje word za pohranu informacija o riječi.
    - PhraseDetail koristi stanje phrase za pohranu informacija o frazi.

```
const [word, setWord] = useState(null);
```

  - Funkcija formatDate:
    - Pretvara datum u čitljiv format (lokalni datum).
    - Ako datum nije dostupan, vraća "nepoznato".

```
const formatDate = (date) => {
    return date ? new Date(date).toLocaleDateString() : "nepoznato";
  };
```

Prikaz podataka

- Detalji koje prikazuju komponente:
  - Obje komponente prikazuju naziv, znacenje, primjer uporabe, broj lajkova, autora i datum izrade.

7.2.11. Pregled Komponenti za Pretraživanje

Komponente SearchBar i SearchResults omogućuju korisnicima pretraživanje fraza i riječi u aplikaciji te prikazivanje rezultata pretraživanja.

Komponenta SearchBar

SearchBar je komponenta koja omogućuje korisnicima unos pretraživačkog upita. Kada korisnik pošalje upit, komponenta šalje upit za pretraživanje i preusmjerava korisnika na stranicu s rezultatima pretraživanja.

Stanje (useState):

- query: Pohranjuje trenutni unos u pretraživaču.

```
setSearchQuery(query);
```

Context (SearchContext):

- setSearchQuery: Koristi se za ažuriranje pretraživačkog upita u kontekstu aplikacije.

```
const { setSearchQuery } = useContext(SearchContext);
```

Navigacija (useNavigate):

- navigate: Koristi se za preusmjeravanje korisnika na stranicu s rezultatima pretraživanja (/search).

Funkcija handleSubmit:

- Sprječava zadani postupak slanja obrasca.
- Postavlja pretraživački upit u kontekst.
- Preusmjerava korisnika na stranicu s rezultatima i čisti unos u pretraživaču.

```
return (
    <form onSubmit={handleSubmit} className="search-bar">
      <input
        type="text"
        value={query}
        onChange={(e) => setQuery(e.target.value)}
        placeholder="Search for words or phrases..."
      />
      <button type="submit" className="search-button">
```

Komponenta SearchResults

SearchResults prikazuje rezultate pretraživanja za fraze i riječi. Komponenta prikazuje rezultate prema upitu pretraživanja, omogućuje sortiranje i kombinira rezultate iz dva različita API poziva (za fraze i riječi).

Stanje (useState):

- results: Pohranjuje rezultate pretraživanja.
- sort: Pohranjuje trenutno odabrani kriterij sortiranja (početno je postavljen na 'date').

```
const [results, setResults] = useState([]);
```

```
const [sort, setSort] = useState('date');
```

Context (SearchContext):

- searchQuery: Dohvaća trenutni pretraživački upit iz konteksta.

```
const { searchQuery } = useContext(SearchContext);
```

Funkcija useEffect:

- Izvršava se svaki put kad se promijeni searchQuery.
- Pokreće paralelne API zahtjeve za pretraživanje riječi i fraza.
- Kombinira rezultate i postavlja ih u stanje results.

```
useEffect(() => {
    const fetchResults = async () => {
      if (searchQuery.trim()) {
        try {
            const [wordsResponse, phrasesResponse] = await Promise.all([
            axios.get(`${process.env.REACT_APP_API_URL}/words/search/`, {
              params: { search: searchQuery, sort }             }),
            axios.get(`${process.env.REACT_APP_API_URL}/phrases/search/`, {
              params: { search: searchQuery, sort }             })
          ]);
          const combinedResults = [
            ...wordsResponse.data.map(word => ({ ...word, type: 'word' })),
            ...phrasesResponse.data.map(phrase => ({ ...phrase, type: 'phrase' }))
          ];
          setResults(combinedResults);
        } catch (error) {
          console.error('There was an error fetching search results!', error);
        }
      } else {
        setResults([]);
      }
    };
```

Prikaz rezultata:

- Prikazuje rezultate pretraživanja u obliku popisa.
- Pruža mogućnost navigacije na detalje o riječi ili frazi putem Link komponente.

7.2.12. Komponenta Profile

Komponenta Profile omogućuje korisnicima da vide informacije o svojem profilu, uključujući riječi i fraze koje su kreirali ili koje su im se svidjele. Stanje (useState):

- createdWords: Pohranjuje riječi koje je korisnik kreirao.
- createdPhrases: Pohranjuje fraze koje je korisnik kreirao.
- likedWords: Pohranjuje riječi koje je korisnik označio kao omiljene.
- likedPhrases: Pohranjuje fraze koje je korisnik označio kao omiljene.

```
const [createdWords, setCreatedWords] = useState([]);
const [createdPhrases, setCreatedPhrases] = useState([]);
const [likedWords, setLikedWords] = useState([]);
const [likedPhrases, setLikedPhrases] = useState([]);
```

Context (AuthContext):

- isAuthenticated: Provjerava je li korisnik prijavljen.
- logout: Funkcija za odjavu korisnika.

```
const { isAuthenticated, logout } = useContext(AuthContext);
```

Funkcija formatDate:

- Pomaže u formatiranju datuma u ljudski čitljiv oblik.

```
const formatDate = (date) => {
    return date ? new Date(date).toLocaleDateString() : "nepoznato";
};
```

Funkcija fetchAllPaginatedData:

- Dohvaća sve podatke s paginacijom. Koristi se za preuzimanje svih riječi i fraza kreiranih od strane korisnika.

```
const fetchAllPaginatedData = async (url, token) => {
  let results = [];
  let nextPage = url;
  while (nextPage) {
    try {
      const response = await axios.get(nextPage, {
        headers: { 'Authorization': `Bearer ${token}` }
      });
```

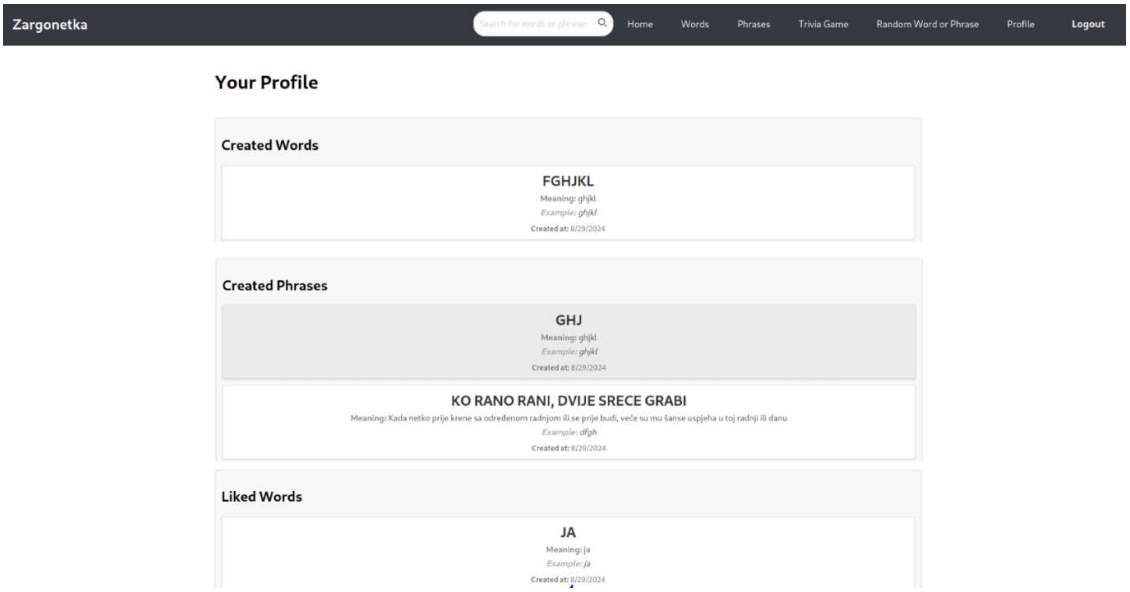
```
      results = [...results, ...response.data.results];
      nextPage = response.data.next;
    } catch (error) {
      console.error('There was an error fetching paginated data!', error);
      break; }}
  return results; };
```

useEffect Hook:

- Pokreće se kada se komponenta učita ili kada se stanje isAuthenticated ili logout promijeni.
- Izvršava više API poziva za dohvaćanje svih kreiranih i označenih riječi i fraza.
- Ako korisnik nije prijavljen, preusmjerava ga na stranicu za prijavu (/login).

Prikaz **p**odataka:

- Profile komponenta prikazuje sve informacije o riječima i frazama koje je korisnik kreirao ili lajkao što je vidiljivo na Slici 5.



Slika 5. Prikaz Profile komponente

7.2.13. Komponenta TriviaGame

Komponenta TriviaGame omogućuje korisnicima da igraju igru pogađanja pitanja u vezi s frazama ili riječima. Korisnici mogu odabrati odgovore i vidjeti rezultate, a također mogu prebacivati između različitih vrsta trivijalnih pitanja.

Ključne Funkcionalnosti

Stanje komponente:

- question: Drži trenutno pitanje i njegove odgovore.
- selectedAnswer: Drži trenutno odabrani odgovor od strane korisnika.
- result: Drži rezultat (ispravnost odgovora) nakon što korisnik odabere odgovor.
- isPhrase: Boolean koji označava je li trenutno pitanje vezano uz fraze (*true*) ili riječi (*false*).

Refs:

- buttonsRef: Koristi se za pohranu referenci na dugmad za odgovore, omogućujući prilagodbu visine svih dugmadi da budu jednake.

```
const buttonsRef = useRef([]);
```

Funkcija shuffleArray:

- Miješa (nasumično razmještava) elemente u polju.
- ```
  const shuffleArray = (array) => {
      return array.sort(() => Math.random() - 0.5);
  };
  ```

Funkcija fetchTriviaQuestion:

- Dohvaća pitanja s API-a na temelju trenutne vrste trivijalnog pitanja (isPhrase).
- Ako odgovori nisu prisutni, ispisuje grešku i postavlja pitanje na null.
- Ako su prisutni, miješa odgovore i postavlja novo pitanje.

```
const endpoint = isPhrase ? '/phrases/trivia/' : '/words/trivia/';
      const response = await axios.get(`${process.env.REACT_APP_API_URL}${endpoint}`);
      const fetchedQuestion = response.data;
      if (!fetchedQuestion.answers) {
        console.error('No answers found in the response:', fetchedQuestion);
        setQuestion(null);
        return;
      }
      const shuffledAnswers = shuffleArray(fetchedQuestion.answers);
      setQuestion({
        ...fetchedQuestion,
        answers: shuffledAnswers,
```
```

```
      });
      setSelectedAnswer(null);
      setResult(null);
    } catch (error) {
      console.error('Error fetching trivia question:', error);
    }
  }, [isPhrase]);
```

useEffect Hook-ovi:

- Prvi useEffect: Poziva fetchTriviaQuestion za učitavanje pitanja kada se komponenta učita ili kada se isPhrase promijeni.
```
useEffect(() => {
    fetchTriviaQuestion();
  }, [fetchTriviaQuestion]);
```
- Drugi useEffect: Prilagođava visinu dugmadi nakon što je pitanje učitano, kako bi dugmadi imale istu visinu.

```
useEffect(() => {
    if (buttonsRef.current.length) {
      const heights = buttonsRef.current.map(button => button ? button.clientHeight : 0);
      const maxHeight = Math.max(...heights);
      buttonsRef.current.forEach(button => {
        if (button) button.style.height = `${maxHeight}px`;
      });
    }
  }, [question]);
```

Funkcije:

- handleAnswerClick: Postavlja odabrani odgovor i rezultat (ispravnost odgovora).
```
const handleAnswerClick = (answer) => {
    setSelectedAnswer(answer);
    const isCorrect = answer.is_correct;
    setResult(isCorrect);
  };
```
- handleRepeatQuestion: Resetira stanje kako bi se isti pitanje moglo ponovo pokušati.
```
const handleRepeatQuestion = () => {
    setSelectedAnswer(null);
```

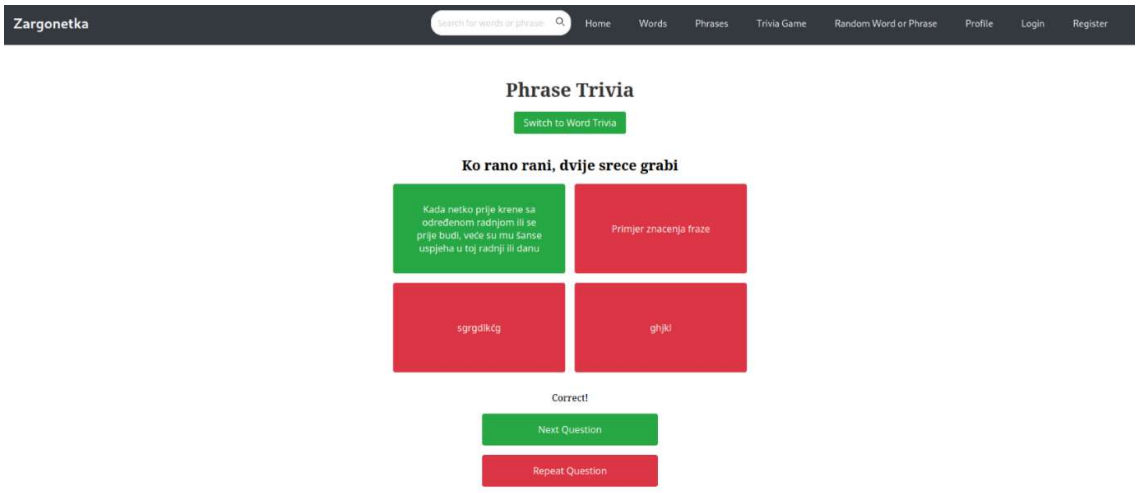
```
    setResult(null);
  };
```

- handleSwitchTrivia: Prebacuje između fraza i riječi trivijalnih pitanja, resetirajući pitanje, odabrani odgovor i rezultat.

```
const handleSwitchTrivia = () => {
    setIsPhrase(prevState => !prevState);
    setQuestion(null);
    setSelectedAnswer(null);
    setResult(null);
  };
```

Renderiranje:

- Prikazuje naslov s trenutnim vrstom trivijalnog pitanja.
- Omogućuje prebacivanje između fraza i riječi trivijalnih pitanja putem dugmeta.
- Prikazuje pitanje i odgovore (ako su dostupni), s dinamički podešenim visinama dugmadi.
- Prikazuje rezultate nakon što je odgovor odabran, s mogućnošću prelaska na sljedeće pitanje ili ponovnog pokušaja, što se može vidjeti na Slici 6.



Slika 6. prikaz komponente Trivia

7.2.14. Komponenta RandomWordOrPhrase

Komponenta RandomWordOrPhrase je dizajnirana za prikaz nasumično odabrane fraze ili riječi korisnicima, s mogućnošću prebacivanja između ove dvije vrste podataka. Evo kako komponenta funkcionira:

Ključne Funkcionalnosti:

Stanje komponente:

- randomItem: Drži trenutno nasumično odabrani element (frazu ili riječ). Ovaj element se prikazuje korisnicima.
- error: Drži poruku o grešci ako dođe do problema prilikom dohvaćanja podataka s API-a.
- isPhrase: Boolean koji označava je li trenutni odabrani tip „fraza“ (*true*) ili „riječ“ (*false*). Početno stanje je postavljeno na "fraza".

```
const [randomItem, setRandomItem] = useState(null);

const [error, setError] = useState(null);

const [isPhrase, setIsPhrase] = useState(true);
```

Dohvaćanje nasumičnog elementa:

- Komponenta koristi fetchRandomItem funkciju koja šalje zahtjev API-u za nasumičnu frazu ili riječ, ovisno o trenutnom stanju isPhrase.
- Ako je zahtjev uspješan, rezultati se pohranjuju u randomItem.
- Ako dođe do greške, prikazuje se poruka o grešci.

Prebacivanje između riječi i fraza:

- Funkcija handleSwitch omogućuje korisnicima da se prebacuju između fraza i riječi.
- Kada se prebacuje, komponenta resetira prikaz nasumičnog elementa tako da se ekran očisti.

```
const handleSwitch = () => {

    setIsPhrase((prevIsPhrase) => !prevIsPhrase);

    setRandomItem(null);

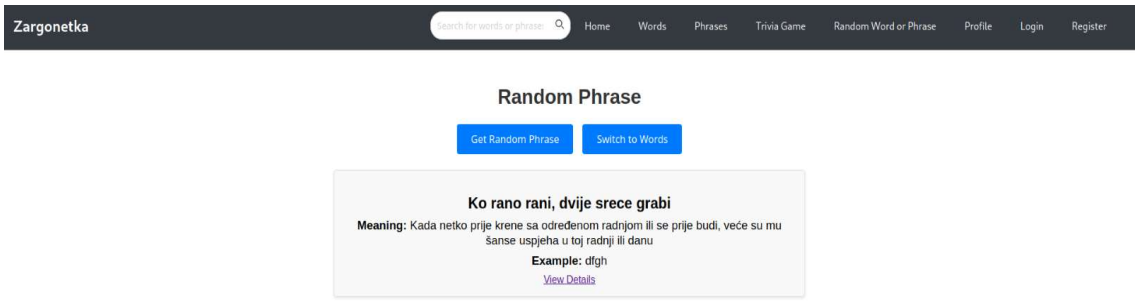
  };
```

Prikaz podataka:

- Kada je randomItem dostupan, komponenta prikazuje detalje o nasumičnom elementu.
- Ako je trenutni odabrani tip "fraza", prikazuje naziv fraze, značenje i primjer. Ako je "riječ", prikazuje naziv riječi, značenje i primjer.
- Također nudi poveznice za pregled detalja o frazi ili riječi.

Korisničko sučelje

- Naslov: Prikazuje trenutni odabrani tip ("Random Phrase" ili "Random Word") kao što je prikazano na Slici 7.
- Gumbi:
    - Get Random: Dohvaća nasumičnu frazu ili riječ.
    - Switch: Prebacuje između prikazivanja fraza i riječi.
- Detalji o Nasumičnom Elementu: Ako je dostupan, prikazuje detalje o frazi ili riječi s poveznicom za pregled detalja.
- Poruke o Greškama: Prikazuje poruke ako dođe do problema prilikom dohvaćanja podataka.



Slika 7. Prikaz komponente Random

8. Implementacija Žargonetke na web

WS EC2 (Elastic Compute Cloud) instanca koristi se za udomljavanje aplikacije. Postupak uključuje:

- **Kreiranje EC2 Instance**:
    - Pristup AWS EC2 konzoli i kreiranje instance s odabirom Linux AMI (npr. Ubuntu 20.04).
    - Odabir t2.micro tipa instance koji je besplatan unutar AWS Free Tier opcije.
    - Konfiguracija sigurnosnih grupa za dopuštanje HTTP (port 80), HTTPS (port 443) i SSH (port 22) prometa.

- **Pristup Instance putem SSH-a**: Pomoću preuzetog .pem ključa, moguće je pristupiti instanci putem SSH-a:
  `ssh -i "your-key.pem" ubuntu@your-ec2-public-ip`

- **Instalacija Potrebnih Paketa i Ovisnosti**
    - Na instanci je potrebno instalirati sve potrebne pakete:
    - Ažuriranje i Instalacija Paketa:

      ```
      sudo apt update && sudo apt upgrade

      sudo apt install python3-pip python3-dev nginx curl
      ```

- **Instalacija virtualenv i Kreiranje Virtualnog Okruženja**

  ```
  pip3 install virtualenv

  virtualenv venv

  source venv/bin/activate
  ```

  - **Kloniranje Django Projekta**:
      - Kloniranje repozitorija s GitHub-a ili prijenos projektnih datoteka:

        ```
        git clone https://github.com/your-username/your-django-project.git

        cd your-django-project
        ```

  - **Instalacija Django Ovisnosti**:

    ```
    pip install -r requirements.txt
    ```

  - **Migracija SQLite Baze Podataka**: Migracija baze podataka lokalno ili na serveru za generiranje db.sqlite3 datoteke:

```
python manage.py migrate
```

Konfiguracija Gunicorn-a i Nginxa

Gunicorn se koristi za posluživanje Django aplikacije, dok Nginx služi kao reverzni proxy server.

- **Instalacija i Konfiguracija Gunicorn-a**:
  - Instalacija Gunicorn-a:

    ```
    pip install gunicorn
    ```

  - Kreiranje Gunicorn konfiguracijske datoteke:

    ```
    sudo nano /etc/systemd/system/gunicorn.service
    ```

  - Postavljanje servisa za Žargonetku:

    ```
    [Unit]

    Description=gunicorn daemon

    After=network.target

    [Service]

    User=ubuntu

    Group=www-data

    WorkingDirectory=/home/ubuntu/zargonetka_backend

    ExecStart=/home/ubuntu/zargonetka_backend/venv/bin/gunicorn --workers 3 --
    bind                        unix:/home/ubuntu/zargonetka_backend/gunicorn.sock
    ZargonetkaProjekt.wsgi:application

    [Install]

    WantedBy=multi-user.target
    ```

  - Omogućavanje i pokretanje servisa:

    ```
    sudo systemctl start gunicorn

    sudo systemctl enable gunicorn
    ```

- **Instalacija i konfiguracija Nginx-a:**
  - Instalacija Nginxa:

    ```
    sudo apt install nginx
    ```

- Kreiranje Nginx konfiguracijske datoteke:

```
sudo nano /etc/nginx/sites-available/zargonetka_backend
```

- Dodavanje potrebnih postavki za Žargonetku:

```
server {

    listen 80;

    server_name 51.20.5.235;

    location / {

        root /home/ubuntu/zargonetka_frontend/build;

        try_files $uri /index.html;

    }

    location /api/ {

        proxy_pass
http://unix:/home/ubuntu/zargonetka_backend/gunicorn.sock;

        proxy_set_header Host $host;

        proxy_set_header X-Real-IP $remote_addr;

        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        proxy_set_header X-Forwarded-Proto $scheme;

    }

    location /static/ {

        alias /home/ubuntu/zargonetka_backend/static/;

    }

    # Serve Django media files

    location /media/ {

        alias /home/ubuntu/zargonetka_backend/media/;

    }}
```

- Aktivacija konfiguracije i ponovno pokretanje Nginxa:

```
sudo ln -s /etc/nginx/sites-available/zargonetka_backend /etc/nginx/sites-
enabled

sudo systemctl restart nginx
```

9. ZAKLJUČAK

Kroz implementaciju aplikacije Žargonetka, kao dokaza o konceptu, demonstrirano je kako kombinacija tehnologija poput Reacta, Django-a, AWS-a i Cloudflare-a može osigurati cjelovito rješenje za izradu moderne web aplikacije. Korištenjem ovih tehnologija, postignuta je brzina i responzivnost na strani klijenta, sigurnost i skalabilnost na strani poslužitelja, te sveobuhvatna optimizacija performansi i sigurnosne mjere na mrežnom sloju. Takav pristup omogućava jednostavnu integraciju, visoku razinu prilagodljivosti te optimizaciju troškova i resursa.

Razvoj aplikacije Žargonetka također je pokazao izazove s kojima se susreću razvojni timovi prilikom implementacije ovakvih rješenja, uključujući postavljanje infrastrukturnih komponenti, osiguravanje visokih performansi i pouzdanosti, te održavanje sigurnosti podataka i korisnika.

Kroz ovu aplikaciju ilustrirano je kako suvremene tehnologije mogu biti iskorištene za stvaranje inovativnih rješenja, pružajući korisnicima vrijednost i unaprjeđujući cjelokupno iskustvo korištenja web aplikacija. Žargonetka tako postaje ne samo aplikacija, već i studija slučaja koja pokazuje snagu i fleksibilnost modernog web razvoja.

Popis literature

[1] J. Webber, "Understanding Web APIs," Journal of Internet Technology, vol. 45, no. 3, pp. 23-30, 2020.

[2] Django Documentation - https://docs.djangoproject.com/

[3] React Documentation - https://react.dev/

[4] L. Griffiths, *Pro Django*, Apress, 2019.

[5] M. H. Austin, *Web Development with Node and Express: Leveraging the JavaScript Stack*, O'Reilly Media, 2016.

[6] A. G. Croft, *Understanding HTTP: The Definitive Guide*, Springer, 2020.

[8] Amazon Web Services Documentation - https://aws.amazon.com/documentation/

[9] Cloudflare Documentation - https://developers.cloudflare.com/

Popis priloga