Web programsko rješenje za upravljanje flash karticama kao potpore učenju

Puhanić, Antonio

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

Sveučilišni studij

Web programsko rješenje za upravljanje flash karticama kao potpore učenju

Završni rad

Antonio Puhanić

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 19.09.2022.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Antonio Puhanić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4454, 24.10.2019.
OIB Pristupnika:	86108392180
Mentor:	Izv.prof.dr.sc. Zdravko Krpić
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Web programsko rješenje za upravljanje flash karticama kao potpore učenju
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	Zadatak završnog rada je istražiti postojeća web programska rješenja za izradu i upravljanje flash karticama kao potpore učenju. Potrebno ih je također kategorizirati i usporediti te iznaći prednosti i nedostatke istih. Na osnovu navedenog definirati zahtjeve na web programski sustav za istu svrhu. Zahtjevi, između ostalih, trebaju obuhvaćati mogućnosti stvaranja i uređivanja flash kartica, grupiranje istih u
Prijedlog ocjene završnog rada:	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene od strane mentora:	19.09.2022.
Datum potvrde ocjene od strane Odbora:	

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

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 22.09.2022.

Ime i prezime studenta:	Antonio Puhanić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4454, 24.10.2019.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: Web programsko rješenje za upravljanje flash karticama kao potpore učenju**

izrađen pod vodstvom mentora Izv.prof.dr.sc. Zdravko Krpić

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Kada je potrebno naučiti veliku količinu gradiva koje sadrži puno teksta, definicija i izraza, istraživanja su pokazala da je jedna od najučinkovitijih metoda učenja korištenje kartica za učenje. Svaka kartica obuhvaća minimalni dio gradiva koje se može razlomiti na pitanje i odgovor, međutim ona ne mora biti ograničena samo na pitanja. Na karticu se može staviti nekakav naslov i određeni tekst, slika ili formula. Zbog toga što je na kartici uvijek samo jedan dio gradiva za razliku od gledanja u cijelu stranicu teksta iz slike, puno je lakše se fokusirati na tu informaciju i naučiti iz toga.

U ovom projektu će se napraviti web aplikacija za učenje iz kartica koja će olakšati njihovu organizaciju i izradu. Potrebno je napraviti da svaki korisnik se može registrirati i imati svoje jedinstvene kartice koje izradio. Za to je potrebno imati bazu podataka i sustav za registraciju korisnika. Za lakšu organizaciju kartica potrebno je napraviti nekakve špilove tj. liste tih kartica koje su dio istog gradiva ili kako god ih korisnik želi svrstati. Uz izradu treba omogućiti i učenje iz tih istih kartica. Kartica osim što treba imati tekst mora imati mogućnost dodavanja slika jer preko slika se mogu staviti mnogi grafovi, formule, skice i ostalo što se može pojaviti u gradivu iz kojeg se uči.

1.1. Zadatak završnog rada

Zadatak završnog rada je istražiti postojeća web programska rješenja za izradu i upravljanje flash karticama kao potpore učenju. Potrebno ih je također kategorizirati i usporediti te iznaći prednosti i nedostatke istih. Na osnovu navedenog definirati zahtjeve na web programski sustav za istu svrhu. Zahtjevi, između ostalih, trebaju obuhvaćati mogućnosti stvaranja i uređivanja flash kartica, grupiranje istih u špilove, mogućnost umetanja slika i formula u njih, mogućnost upisivanja pitanja i odgovarajućih odgovora, mogućnost samoocjenjivanja i dijeljenja kartica s drugim korisnicima. U praktičnom dijelu rada potrebno je, na osnovu definiranih zahtjeva, implementirati programsko rješenje za web platforme korištenjem JavaScripta i React biblioteke. Aplikaciju testirati na grupi korisnika i provesti anketu na osnovu koje treba ocijeniti korisničko iskustvo.

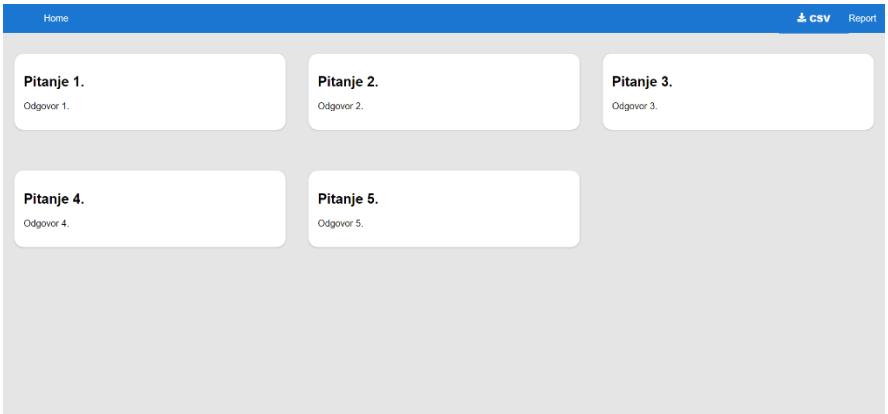
2. PREGLED PROGRAMSKIH RJEŠENJA

Učenje putem kartica pruža veću interaktivnost osobe i gradiva kojeg uči jer nakon svakog pitanja osoba provjerava svoje znanje nakon što se pogleda koji je točan odgovor. Ponavljanje istih kartica također pomaže shvatiti osobi koje gradivo može lakše ili teže zapamtiti. Kartice se mogu koristiti kao jedan od načina za brže učenje gradiva koje je tekstualno opsežno ili kada potrebno učiti mnogo stvari napamet. Jedna strana kartice sadrži pitanje, a na drugoj strani kartice se nalazi odgovor. Klasično su se kartice pravile i koristile na papiru. Jedna strana papira je imala pitanje ,a druga je imala odgovor. Iz špila su se kartice izvlačile nasumično. ali to je danas puno lakše i brže na računalu jer se zaobilazi ručno izrađivanje svake kartice zasebno. Pomoću kartica je lakše se fokusirati na određene dijelove gradiva koje treba naučiti jer u bilo kojem trenutku pri učenju na kartici se nalazi samo jedno pitanje i odgovor na to pitanje. Nema velikih odlomaka tekstova na koje se teško fokusirati i zapamtiti sve čitajući cijelo vrijeme.

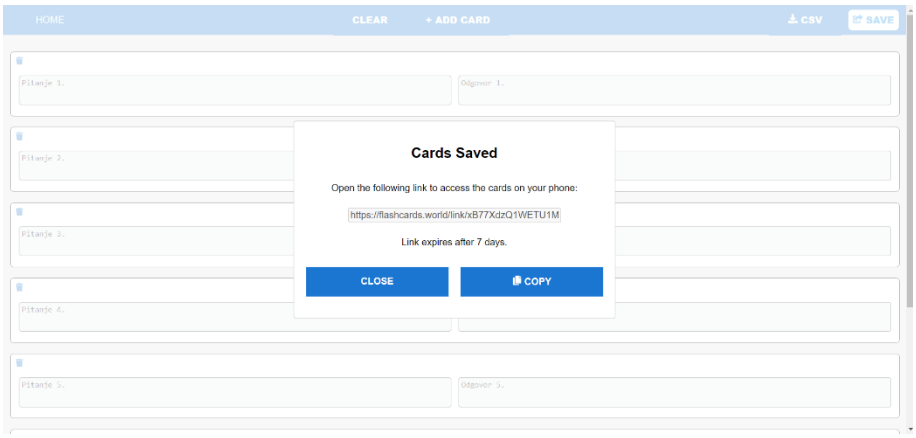
2.1. Pregled postojećih web aplikacija za učenje s karticama

Na internetu već postoji mnogo različitih web aplikacija koje se zasnivaju na konceptu tih kartica. Razlikuju se po izgledu kartica, načinu prikazivanja, načinu izrade i sl.

Web stranica FlashCards World (Slika 2.1) je jednostavno rješenje koje pokazuje sve kartice odjednom, te se na svakoj kartici nalazi pitanje i odgovor istovremeno [1]. Nema mogućnost registracije korisnika, te svaki set kartica se mora otvoriti kopiranjem dobivene poveznice nakon izrade (Slika 2.2). Svaka poveznica važi samo sedam dana. Ovakav pristup karticama je veoma limitiran i forsira korisnika da pravi iste kartice ponovno nakon sedam dana što može biti vrlo frustrirajuće. U ovom projektu će se to izbjegavati, te će biti moguće jednostavno s gumbom početi učiti bez da se mora ručno otvarati kartica s poveznicom.



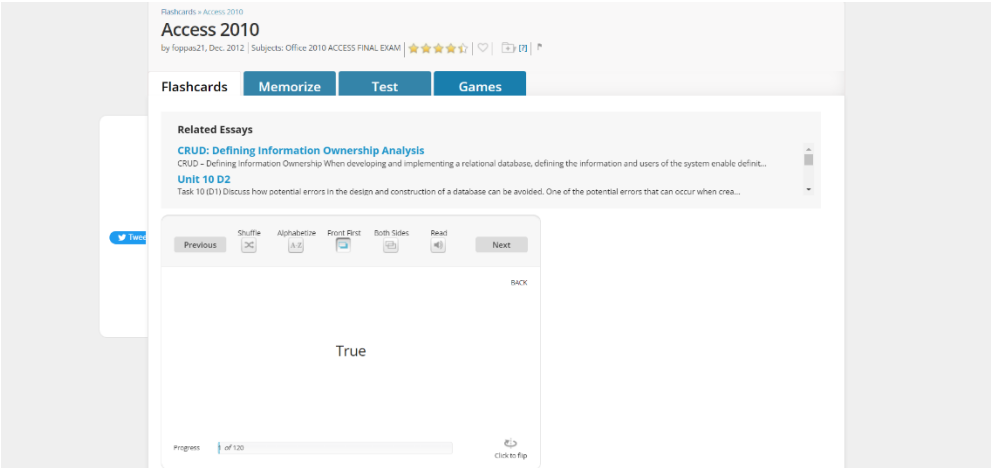
Slika 2.1. FlashCards World web stranica za učenje



Slika 2.2. Izrada kartica na FlashCards Worlds web stranici

Web stranica Cram.com je jedna od rješenja za kartice [2]. Ima javno dostupne špilove iz kojih se može učiti, pruža mogućnost registracije i trajno spremanje špilova. To je pristup koji će se koristiti u ovom projektu, bitno je imati kartice spremljeno da se korisnik o tome ne treba brinuti.

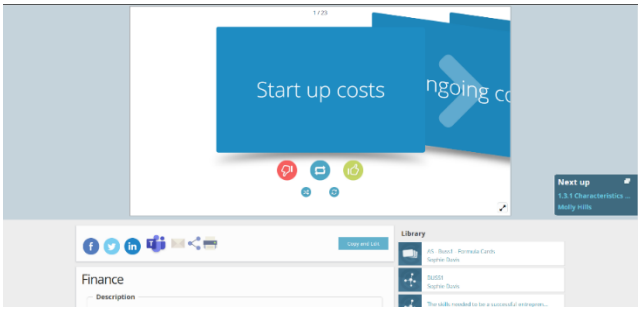
Karte prikazuju pitanje i odgovor zasebno. Veličina prikaza kartica je mala, te u tom sučelju prikazivanja se nalazi mnoštvo stvari koje su distrakcija pri učenju. To je sve što nije trenutna kartica iz koje se uči. U ovom slučaju je to tipka za *Tweet* s lijeve strane kartice, dio stranice koji prikazuje eseje koji mogu biti slični temi učenja, ocjena i detalji ispod naziva špila. Jedna od najvećih distrakcija pri učenju koje se često pojave na ovakvim stranicama su reklame. To je zbog načina na koji su reklame dizajnirane da uhvate pažnju korisnika.



Slika 2.3. Prikaz sučelja za kartice od web stranice cram.com

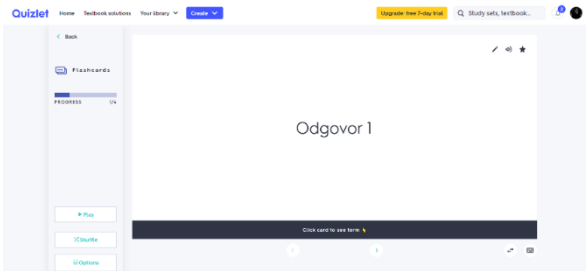
Ostali primjeri su Goconqr, Quizlet i Brainscape [3]. Funkcioniraju tako da pruže korisniku izradu i spremanje kartica. Prikazuju kartice jednu po jednu koje imaju mogućnost provjeravanja znanja

sa ocjenama od jedan do pet, sa točnim ili netočnim odgovorom i sl. Na taj način će i ova aplikacija biti napravljena.

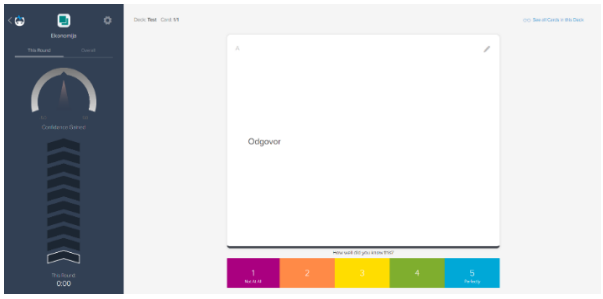


Slika 2.4. Prikaz sučelja Goconqr.com

Na ovom primjeru su prikazane kartice kao trodimenzionalne kartice, ali je mijenjanje kartica veoma sporo i nezgrapno za koristiti [3]. U ovoj aplikaciji će se izbjegavati takve animacije jer ne pridonose značajke koje bi bile korisne pri učenju. Cilj će biti napraviti da je prikazivanje svake sljedeće kartice brzo i jednostavno.



Slika 2.5. Prikaz sučelja Quizlet.com



Slika 2.6. Prikaz sučelja Brainscape.com

Stranice Quizlet i Brainscape će se koristiti kao inspiracija za dizajn kartica ovog projekta [5]. Kartice su velike i lagano se fokusirati na svaku karticu pri učenju, no imaju dugačak postupak stvaranja špilova i kartica [6].

2.2. Zahtjevi na programsko rješenje

Glavni zahtjev je napraviti sučelje koje će se fokusirati na jednostavnost korištenja i preglednost pri učenju. Cilj je imati brzo i jednostavno dodavanje špilova i kartica. Koristiti će se biblioteka *React* jer funkcionira pomoću koncepta komponenti što je prikladno za ovaj projekt.

Iz već postojećih programskih rješenja se može vidjeti da je potrebno imati sučelje koje će sadržavati samo ono što je u trenutku potrebno da se izbjegnu distrakcije pri učenju. Također nije dobro imati privremeno spremanje kartica jer te kartice mogu korisniku zatrebati u budućnosti. Poželjno je korisnika potaknuti na učenje kroz samocjenjivanje kao što to omogućava aplikacija Brainscape. Na osnovu tih karakteristika postojećih rješenja će se formirati sljedeći funkcionalni i nefunkcionalni zahtjevi.

Funkcionalni zahtjevi koje aplikacija mora omogućiti :

1. Registraciju i prijavu korisnika
2. Oporavak lozinke
3. Stvaranje špilova
4. Stvaranje kartica
5. Prikaz svih postojećih špilova
6. Prikaz svih kartica tijekom izrade
7. Pokretanje načina za učenje
8. Brisanje kartica
9. Dodavanje slika u kartice uz tekst

Nefunkcionalni zahtjevi koji se očekuje od aplikacije:

1. Kartice se na popisu trebaju pojaviti unutar 1s od trenutka dodavanja
2. Učitavanje slike iz kartice prilikom učenja treba biti unutar 1s od trenutka prikazivanja
3. Sustav treba onemogućiti dodavanje praznih kartica
4. Sučelje treba biti jednostavno, s maksimalno dvije tipke i s karticama koje se ističu u odnosu na pozadinu
5. Prikaz kartica za učenje na sučelju sustava treba se nalaziti na sredini sučelja neovisno o veličini zaslona
6. Na svakoj kartici se treba jasno iskazati prikladnom bojom koja je ocjena
7. Sustav treba omogućiti trajnu pohranu podataka na Firebase Cloudstore bazi podataka

3. TEHNOLOGIJE KORIŠTENE ZA IZRADU WEB APLIKACIJE

Cijela aplikacija će se većinom bazirati na JavaScript programskom jeziku. Iako je to veoma dobar programski jezik za web aplikacije, postoji mnoštvo dodatnih načina na koje se taj programski jezik se može proširiti i poboljšati. Za izvođenje JavaScript-a koristiti će se Node.js. S obzirom da aplikacija treba omogućiti prikazivanje špilova i kartica koristiti će se React. Potrebno je obrađivati podatke koje korisnik unosi što će se postići korištenjem Firebase-a.

3.1. Node.js

Node.js je posebno asinkrono JavaScript razvojno okruženje napravljeno 2009. godine koje se još uvijek unaprjeđuje. To je prvo razvojno okruženje za JavaScript na kojem se skripte izvršavaju na strani servera [6]. Jedna od važnijih značajki je ugrađena podrška za upravljanje paketima. Node.js sadržava kolekciju modula koji rukuju s osnovnim funkcionalnostima web aplikacija te podržava instalaciju dodatnih modula tj. paketa. Namjera svih paketa je olakšano stvaranje aplikacija i pojednostavljivanje njenog razvoja u budućnosti. Iako je JavaScript jednonitni programski jezik ono svejedno omogućuje asinkrono programiranje, što znači da može prividno raditi više zadataka odjednom. Omogućuje izvršavanje skripti na strani servera te nakon što se izvrši šalje se klijentu kao gotov paket koji je spreman za prikazivanje. Node.js će biti korišten kao izvršno okruženje (engl. *Runtime Environment,* RTE) na kojemu će se izvoditi ova aplikacija, također će biti korištena zbog upravljanja dodatnim paketima koji će pomoći prilikom izrade aplikacije.

3.1.1. NPM

To je upravitelj paketima Node.js razvojnog okruženjima s kojim je lako instalirati dodatne pakete za izradu aplikacija u JavaScriptu [7]. Trenutno postoji preko milijun različitih paketa koji se mogu instalirati. Pomoću NPM-a se može lako ažurirati, konfigurirati i brisati paketi uz vrlo jednostavne naredbe. Pomoću package.json datoteke NPM može instalirati sve ovisnosti (eng. d*ependecies*) odjednom bez da se korisnik brine o svakom paketu zasebno. Ta datoteka također omogućuje automatsko ažuriranje svih instaliranih paketa. Nekada nije poželjno ažurirati sve pakete te se stoga koristi package-lock.json datoteka za instaliranje točno određenih verzija. Omogućuje da bilo tko objavi svoje pakete što može dovesti do toga da korisnik instalira paket loše kvalitete, stoga pakete ocjenjuju korisnici te se na osnovu toga može zaključiti kakav je paket.

NPM dolazi već instaliran uz Node.js, ali da bi se mogao početi koristiti potrebno ga je inicijalizirati uz naredbu „npm init“. Paketi se instaliraju uz naredbu „npm i <ime paketa>“. Za instaliranje React biblioteke korištena je drugačija NPM naredba: „npx create-react-app“ jer ta

naredba odmah instalira i pokrene aplikaciju. NPM će se koristiti zbog toga što pokreće lokalni poslužitelj na kojoj će se web aplikacija nalaziti i zbog instaliranja dodatnih paketa.

3.2. React

Za izradu svih vizualnih komponenata i sučelja(engl. *Front-end*) ove aplikacije korištena je React biblioteka za JavaScript. React je napravio Facebook 2013. godine [8]. Budući da služi samo za vizualne dijelove, da bi se napravila cjelovita React aplikacija potrebno je instalirati dodatne biblioteke poput React-bootstrap za uređivanje i React-router-dom za usmjerivanje. Zasniva se na izradi programskih komponenti (engl. *Software Component*) koje se kasnije mogu koristiti bilo gdje u aplikaciji. Jedna od značajki je to da koristi „virtualni DOM“ koji mijenja samo one elemente koje je potrebno. React zbog svoje virtualne memorije vidi razlike naspram onog što se nalazi u pretraživaču i prema tome mijenja komponente. Po tome se kôd piše tako da se zamisli da se svaki put cijela stranica ponovo renderira, iako se u stvarnosti samo jedan dio stranice mijenja.

Komponente se najčešće izrađuju u JSX sintaksi koja je specifična po tome što uz JavaScript sadrži HTML komponente, zbog toga je vrlo lagano interpretirati takav kod jer je jasno koji dio je JavaScript, a koji dio je HTML. Sintaksa stvaranja komponente u JSX se može vidjeti na slici 3.1.

```
import React from 'react'
export default function Deck({decks}) {
    return (
        <div>
            <label>
                Deck Name: {deck.deckName}
            </label>
        </div>
    )
}
```

Slika 3.1. Sintaksa za izradu komponente

Zbog koncepta komponenti je odlučeno da se koristi *React* za ovaj projekt jer se špilovi, kartice, prikaz popisa istih najlakše može realizirati pomoću *React-a*. Svaki špil će biti reprezentiran komponentom, prikazivanje tih špilova će se postići sa komponentom koja sadrži još komponenti tj. špilova. Isto se to odnosi i za same kartice. Prilikom izrade lako je vidjeti definirati i vidjeti što koja komponenta sadrži što čini proces izrade same aplikacije lakše.

3.2.1. Nove značajke React-a u verziji 16.8

U prijašnjim verzijama za programiranje komponenti i njihovih funkcionalnosti korištene su klase. U verziji 16.8 su uvedene udice (engl. *Hooks*) koje su zamijenile klase. Budući da se ne koriste klase, a potrebno je pratiti nekakvo stanje komponente koristi se „*useState hook*“. Svrha tih *„hook“* značajki je da se „zahvate“ za određenu React komponentu i omogući nekakvu dodatnu funkcionalnost. Mogu se pozvati samo unutar React komponente te se ne smiju pozivati unutar petlji ili drugih funkcija koji koje nisu na najvišoj razini komponente. Postojeći *hook* elementi su *useEffect, useContex, useRef, useReducer, useCallback, useMemo* i uz to je moguće napraviti prilagođene *hook* elemente [10]. Za ovu aplikaciju ova verzija je bitna jer ne zahtjeva stvaranje novih klasa. Proširuje funkcionalnost komponenti na jednostavniji način naspram klasa.

3.2.2. Komponente

React komponenta je ponovno iskoristivi dio koda koji predstavlja JavaScript funkciju koja nakon što se pozove vraća HTML elemente. Svaka komponenta može prikazati bilo kakav HTML sadržaj i nakon što je definirana može se pozvati bilo gdje u programskom kodu. Prilikom izrade React aplikacije, generiraju se datoteke *App.js* i *indeks.js*. Datoindeksindex.js je glavni dio React aplikacije koji sadrži jedinstvenu komponentu *App* koja se renderira u DOM (Slika 3.11). *App.js* sadrži glavnu komponentu tj. funkciju koja se izvozi pod također pod istim imenom da bi se mogla indeksti u *index.js*.

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

Slika 3.1. Izradu App komponente u indeks.js datoteci

Da bi se komponenta iz druge datoteke mogla pozvati i koristiti prvo se mora izvesti uz ključnu riječ „export“ (Slika 3.2). U komponentu *App* se stavljaju se stavljaju sve ostale komponente cijele React aplikacije.

```
function App() {
  return (
    <>
    </>
  );
}

export default App;
```

Slika 3.2. Primjer početnog koda App.js datoteke

Kroz komponente se mogu prosljeđivati varijable s kojima je onda moguće upravljati sadržajem unutar te komponente i svih ostalih koji se unutar nje pozivaju. U React-u argumenti se mogu samo prosljeđivati u jednom smjeru, od roditelja do djeteta. Ako je potrebno komponenti na najnižoj razini proslijediti argument, a postoji mnogo komponenti između koristi se *useContext*

3.3. React-boostrap biblioteka

Bootstrap je programski okvir za CSS i uređivanje stranica. Sadrži gotove predloške za HTML elemente koji se mogu dodatno urediti. Za React napravljena je posebna biblioteka po uzoru na Bootstrap koja već sadržava gotove i stilizirane komponente [9]. Implementiranje takve komponente zahtijeva instaliranje React-boostrap biblioteke te izvršavanje naredbe „*import*“ da bi se određen element mogao iskoristiti. Instalacija se vrši pomoću NPM. Poznato je da se za stiliziranje HTML elemenata koristi svojstvo *class* koje se kasnije pozove u CSS datoteci gdje se dodatno uređuje. U react-bootstrap biblioteci već postoji *stylesheet* preko kojeg se mogu implementiraju vizualne promjene preko imena klasa u svojstvu *className* umjesto c*lass*. Primjer toga može se vidjeti u slici 3.3. Umjesto da ručno se izrađuje CSS datoteka u kojoj će se primijeniti oblikovanje, s ovom bibliotekom se direktno pozove oblikovanje koje je potrebno unutar *className*. Sada na sve elemente unutar *div* taga je postavljena širina na 100% od komponente u kojoj se nalazi, sav tekst je centriran i postavljena je gornja margina.

```
“div className="w-100 t“xt-center mt-2">

        Need an account?
        <Link to='/SignUp'> Sign up</Link>
</div>
```

Slika 3.2. Primjer uređivanja uz react-boostrap

Za ovaj projekt je *React-bootstrap* najbolje rješenje za uređivanje zbog toga što će olakšati stvaranje sučelja za špilove i kartice. Zbog mnogih gotovih predložaka za HTML elemente omogućuje da se prvobitno fokusira na funkcionalnost stranice jer prilikom implementacije ovog programskog okvira odmah se primjenjuju mnogi CSS stilovi na stranicu čini stranicu preglednijom tijekom njene izrade i samog korištenja stranice.

3.4. React-router-dom

To je biblioteka koja omogućuje usmjeravanje na određene poveznice, prikazivanje određenog sadržaja i manipuliranje URL-om. Kao i ostale biblioteke mora se instalirati pomoću NPM. U App.js se postavljaju sve poveznice i rute koje će se koristiti za cijelu aplikaciju uz pomoć *useContext* da bi se moglo sa bilo koje stranice pristupiti bilo kojoj drugoj stranici unutar

aplikacije. Postavljanje ruta se postiže također pomoću React komponenti. Glavna komponenta za usmjeravanje je *BrowserRouter* , unutar te komponente se stavlja komponenta *Routes* koja će sadržavati sav popis stranica i ruta [11]. Svaka stranica tj. komponenta u ovom slučaju se stavlja pod *Route* komponentu kao što je prikazano na slici 3.3.

```
<Route path='/Profile' element={<Profile />}></Route>
```

Slika 3.3. Primjer uređivanja uz react-boostrap

Pod *path* se stavlja dio poveznice na koji se spaja, a na *element* se stavlja komponenta koja prikazuje sadržaj za tu stranicu.

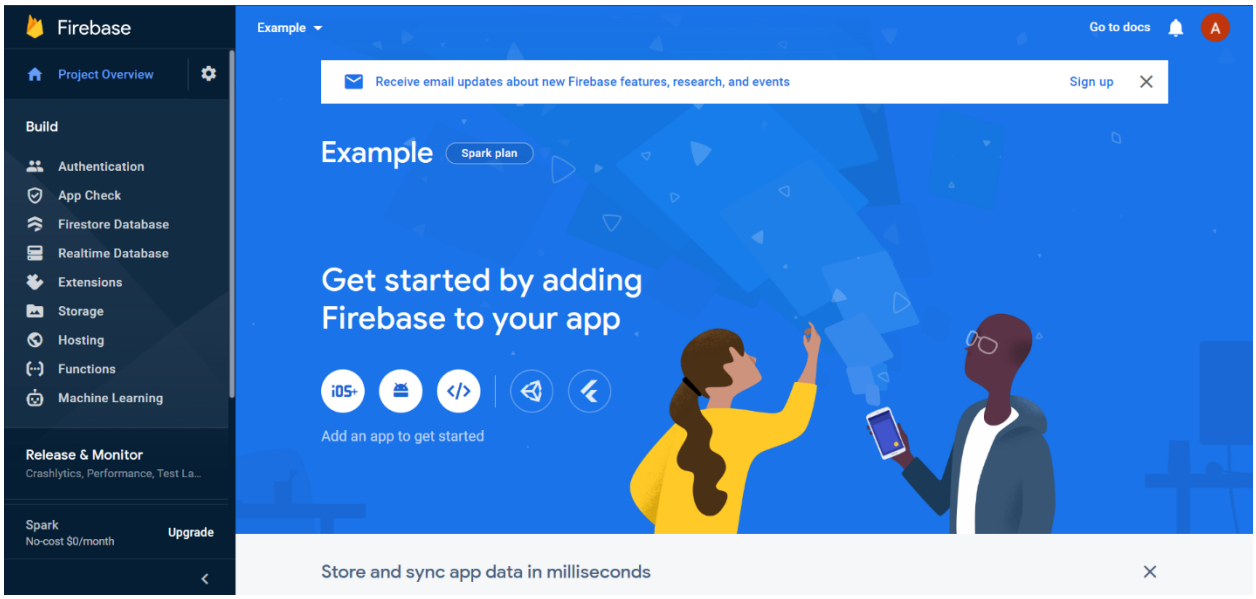
Da bi se uopće omogućilo usmjeravanje na poveznice korištena je ova biblioteka, iako je moguće ostvarivanje usmjeravanje na drugačije načine, ovaj način je najbolji zbog toga što je specifično dizajniran za *React* projekte.

3.5. Firebase

Firebase je platforma za razvoj koja se može koristiti za web i mobilne aplikacije za mnoštvo različitih programskih jezika [12]. Za ovu aplikaciju će se najviše koristiti kao mjesto za spremanje podataka i autentifikaciju korisnika. Za autentifikaciju korisnika kroz *Log in* i *Sign up* stranice u ovom projektu je također korišten Firebase koristeći e-mail. Automatski se brine za sigurnost svih registriranih korisnika i pomaže pri prikazivanju sadržaja za određenog korisnika. Za registriranje korisnika osim e-maila može se koristiti vanjske stranice poput Google, Facebook, Twitter i sl.

Firebase je odabran zbog toga što pruža automatsko rukovanje registracijama i prijavama, također daje biblioteku funkcija koje olakšavaju stavljanje i izradu dokumenata u bazu podataka.

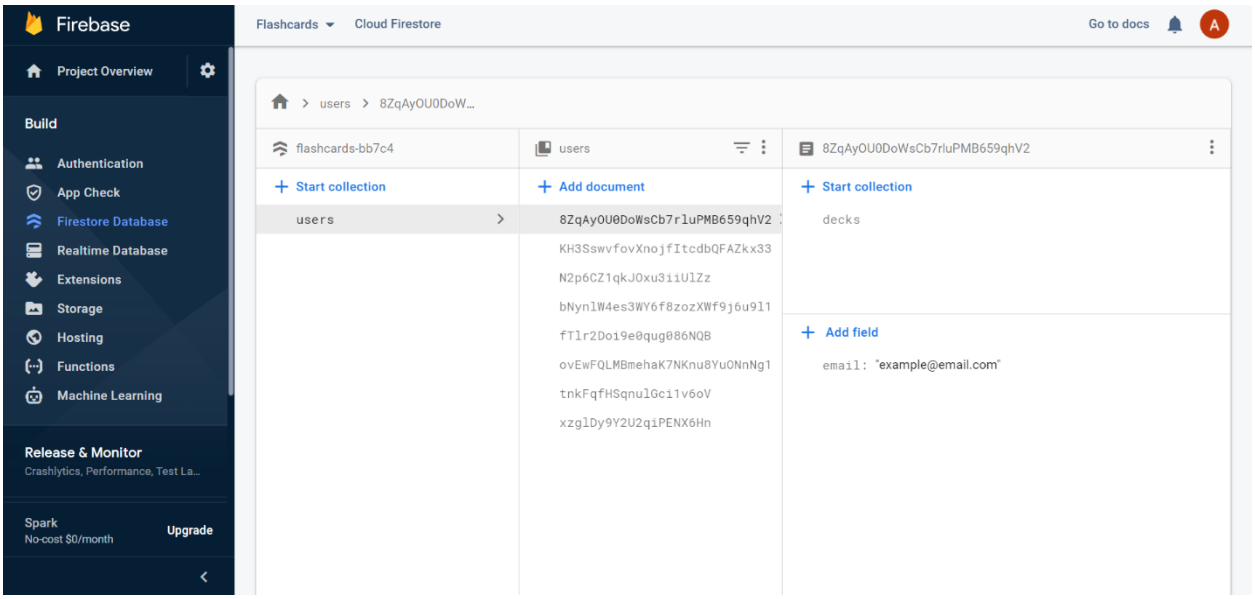
Za instalaciju se koristi NPM s naredbom „*npm install firebase“*. NPM će automatski postaviti sve potrebno da bi se veza između aplikacije i baze podataka mogla uspostaviti. Također je potrebno napraviti projekt na Firebase stranici tj. samu bazu podataka gdje će se upravljati korisnicima i njihovima stranicama. Firebase se brine za fizičko spremanje podataka, autentifikaciju i ostale pozadinske servise potrebne za rad web stranica.



Slika 3.4. Sučelje Firebase aplikacije

3.5.1. Pohranjivanje podataka o korisnicima

Dohvaćanje i spremanje podataka se odvija kroz Cloud Firestore s dokumentima i kolekcijama. Cloud Firestore je dio Firebase projekta koji služi kao mjesto za pohranjivanje podataka i njenom obradom. Kolekcija je samo spremnik dokumenata i ne može sadržavati vrijednosti, dok dokumenti sadrže podatke i u sebi mogu imati svoju kolekciju koja će imati dokumente itd.



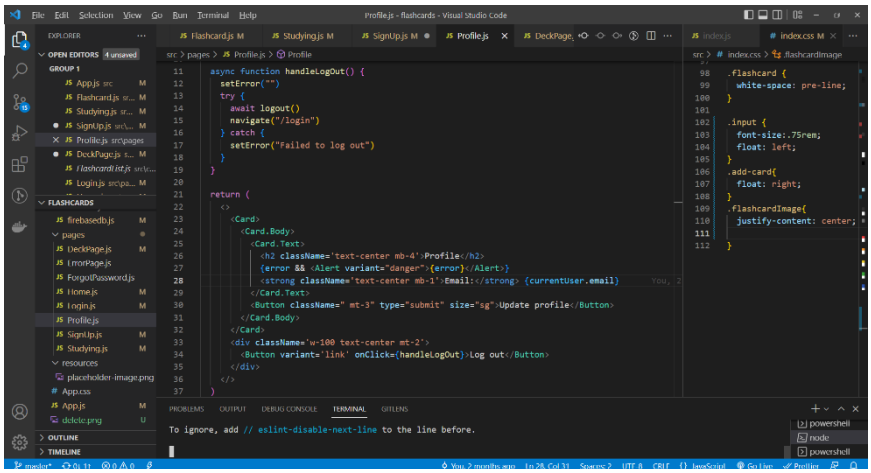
Slika 3.5. Prikaz sučelja Firebase-a i pohranjenih podataka.

Na slici 3.4. se vidi jedna kolekcija *users* koja sadrži dokumente. Svaki dokument u ovom slučaju predstavlja pojedine registrirane korisnike koji su imenovani po njihovom jedinstvenom UID-om koji generira Firebase. Dokumenti mogu sadržavati podatke bilo kojeg tipa. U ovom primjeru

dokumenti tj. korisnici imaju podatak za njihovu e-mail adresu. Svi korisnici imaju svoju daljnju kolekciju *decks* koja sadržava popis špilova.

3.6. Visual Studio Code

Visual Studio Code je uređivač koda otvorenog koda za razvoj programa. Dostupan je na Windows, MacOS i Linux operacijskim sustavima [13]. Podržava pisanje koda u mnogim programskim jezicima. Olakšava programiranje kroz mnoga proširenja koja se mogu instalirati. Ovaj uređivač koda će se koristi jer ima ugrađeno IntelliSense značajku za dovršavanje koda, GitHub sučelje za upravljanje repozitorijima, podršku za otklanjanje grešaka i testiranje koda i ugrađen terminal.



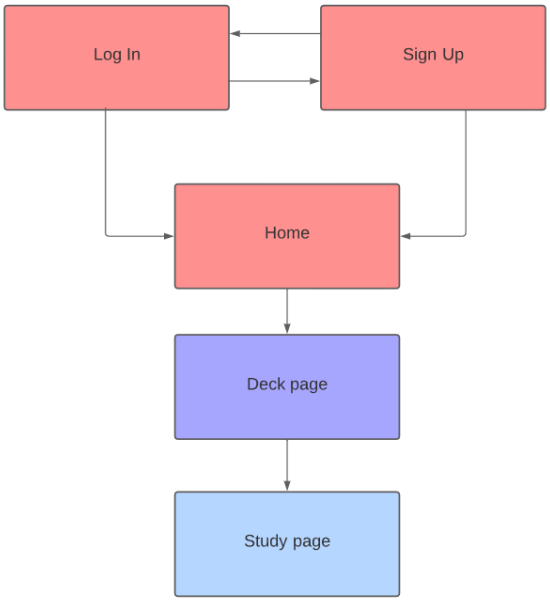
Slika 3.5. Prikaz sučelja Visual Studio Code programa.

Osim navedenih značajki, ovaj uređivač će se koristiti zbog dodatnih proširenja poput automatskog zatvaranja HTML tagova, automatskog uređivanja koda i alata za otklanjanje grešaka u React kodu.

4. IZRADA WEB APLIKACIJE ZA POMOĆ PRI UČENJU

Kao što je već spomenuto ovaj projekt je napravljen koristeći React biblioteku i Firebase bazu podataka. Aplikacija izrađena u okviru ovog rada je napravljena pomoću React biblioteke i Firebase platforme za obradu podataka. U ovom poglavlju će se objasniti koraci pri izradi aplikacije. Svrha aplikacije je napraviti sučelje koje će olakšati proces učenja. Glavni cilj izrade je napraviti sustav za autorizaciju korisnika koji će imati pristup samo svojim karticama, te napraviti da se kartice prikazuju korisniku u stilu za učenje.

Prije same izrade potrebno je imati isplanirano strukturu same aplikacije u kojoj će se vidjeti koje stranice će biti međusobno povezane i kakva hijerarhija komponenti treba biti.



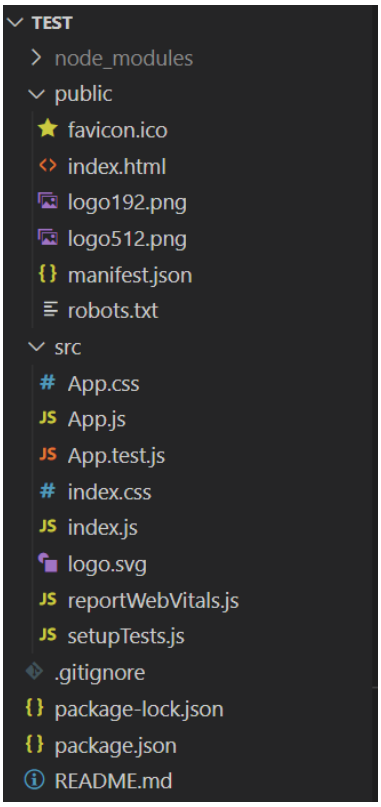
Slika 4.1. Struktura web aplikacije

Glavna struktura aplikacije je prikazana na slici 4.1. Ova struktura ima oblik Model-Pogled-Upravljač (engl. *Model-View-Controller, MVC*) arhitekturnog obrasca. Prvo što korisnik vidi je *Log In* stranica, ako korisnik nije registriran onda se može preusmjeriti na stranicu *Sign Up*. Nakon što se korisnik registrira ili prijavi preusmjeren je na *Home* stranicu gdje se nalazi sučelje za stvaranje špilova i popis svih već postojećih špilova. Stvaranjem špila ili klikom na pojedini postojeći špil korisnik je preusmjeren na *Deck page*. Na toj stranici se izrađuju kartice gdje se i nalazi popis svih kartica tog špila. Korisnik može raditi neograničeni broj kartica, te pritiskom na gumb *Start studying* bit će preusmjeren na *Study page* gdje se kartice prikazuju u načinu za učenje.

Ispod kartice se nalaze ocjene od jedan do pet te klikom na bilo koju od ocjena prikazuje se sljedeća kartica u nizu.

4.1. Konfiguracija React-a

Prvenstveno se na računalo mora instalirati Node.js okruženje. Tek nakon instalacije se može krenuti postavljati aplikacija za rad u React-u. U Visual Studio Code programu će se otvoriti mapu u kojoj će se nalaziti projekt i otvoriti terminal *powershell* s kojim će se izraditi početna React aplikacija. U terminal će se upisati naredba „*npx create-react-app .*“. Točka na kraju naredbe definira da se React instalira u mapi koja je trenutno otvorena u Visual Studio Code programu.



Slika 4.2. Početne datoteke prilikom instalacije React-a

Mapa *node_module* sadrži sve pakete za Node.js i programski kod potreban za programiranje i korištenje vanjskih alata. U mapi *public* se nalaze resursi za početak rada, ali u ovom slučaju su nepotrebni te će se obrisati sve datoteke osim *index.html*, *manifest.json* i *robots.txt*. Datoteka *index.html* služi kao predložak na kojem se sadržaj komponente *App* zapravo konačno prikazuje.

```
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
    To create a production bundle, use `npm run build` or `yarn build`.
  -->
</body>
```

Slika 4.3. Prikaz elementa u kojem se prikazuje sadržaj stranice

U slici 3.1. se može vidjeti programski kod pomoću kojeg se generira sadržaj cijele stranice u *root* elementu sa slike 4.2. Komentar koji se nalazi na istoj slici se automatski postavlja prilikom instalacije. Upozorava kako funkcionira aplikacija, te govori što je potrebno napraviti da se pokrene. U terminalu će se napisati naredba „*npm start*“ , nakon izvršavanja naredbe u zadanom pretraživaču otvoriti će se stranica adresi lokalnog poslužitelja i portu 3000: *localhost:3000* na kojoj se može vidjeti pravi sadržaj stranice.

Jedna od prednosti u ovom načinu rada je što prilikom spremanja promjena u datotekama za stranicu, automatski nakon promjene, sadržaj na stranici će se ažurirati u stvarnom vremenu bez potrebe za osvježavanjem stranice

Pri izradi ovog projekta koristiti će se već spomenute biblioteke *react-router-dom* i *react-bootstrap*. Oba dvije biblioteke se instaliraju uz NPM pomoću naredbe „*npm install*“.

4.2. Integriranje Firebase značajki u web aplikaciju

Kako bi se mogla stvoriti komunikacija između aplikacije i Firebase platforme napraviti će se projekt na *console.firebase.google.com* stranici kroz čije sučelje će se generirati programski kod s kojim će se aplikacija spajati na Firebase. Na slici 4.2. je primjer programskog koda s kojim se aplikacija spaja na Firebase bazu podataka.

```
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";

const firebaseConfig = {
  apiKey: "AIzaSyAtRzBa-HmrCJuUdaqF-3eVePbi0vS-Wnk",
  authDomain: "flashcards-bb7c4.firebaseapp.com",
  projectId: "flashcards-bb7c4",
  storageBucket: "flashcards-bb7c4.appspot.com",
  messagingSenderId: "110458463030",
  appId: "1:110458463030:web:6d194310aa03d9a3e00997",
  measurementId: "G-HD77TP8R73"
};

const app = initializeApp(firebaseConfig);
export const auth = getAuth(app);
export default app
```

Slika 4.4. Kod potreban za povezivanje na Firebase bazu podataka

Svaka komponenta zasebno mora imati pristup bazi podataka. Najlakše rješenje za to je korištenje značajke *useContext* s kojom će se metode za registriranje, prijavu i identifikaciju predati svim komponentama. Napraviti će se datoteka *AuthProvider* koja će sadržavati sve te funkcionalnosti.

Da bi se osiguralo da korisnici pristupaju samo svojim podatcima u Cloud Firestore se moraju postaviti pravila s kojima se definira što korisnici smiju raditi nad bazom podataka. To se postiže sa postavljanjem pravila (engl. *Rules*) u Firestore bazi podataka.

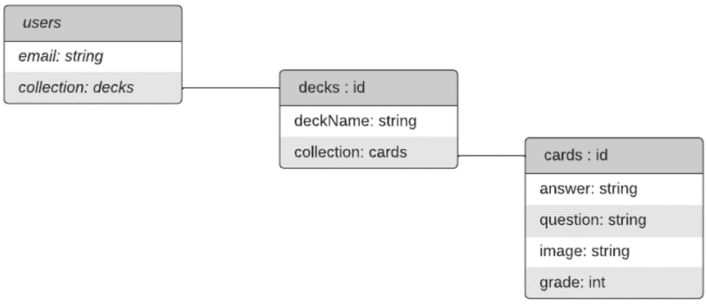
Na slici 4.4. je programski kod kojim se omogućuje da prijavljeni korisnici mogu pristupati samo svojim dokumentima u Firebase kolekcijama. Pravila također definiraju i dodatne sigurnosne mjere za zaštitu podataka.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if true;
    }
  }
  match /users/{userID}{
    allow read,update,delete: if request.auth != null  && request.auth.uid == userID;
    allow create: if request.auth!=null;
  }
}
```

Slika 4.5. Definiranje pravila za korisnike na bazi podataka

4.2.1. Firebase model baze podataka

Model baze podataka koji se koristi u ovoj aplikaciji je prikazan slikom 4.6.



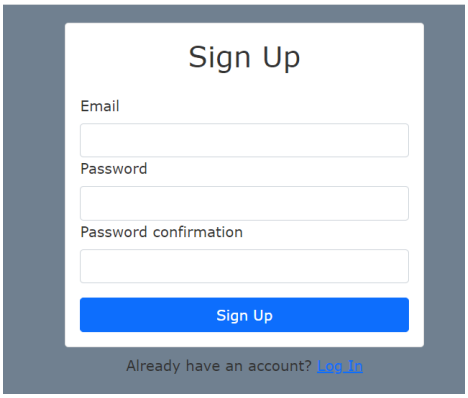
Slika 4.6 Struktura u bazi podataka

U ovom projektu postoji glavna kolekcija *users* koja služi za spremanje svih registriranih korisnika. Registrirani korisnici su pohranjeni u dokument koji sadrži *email* vrijednost i kolekciju za špilova. Svaki dokument koji je generiran ima za naziv ID prijavljenog korisnika. Kada korisnik

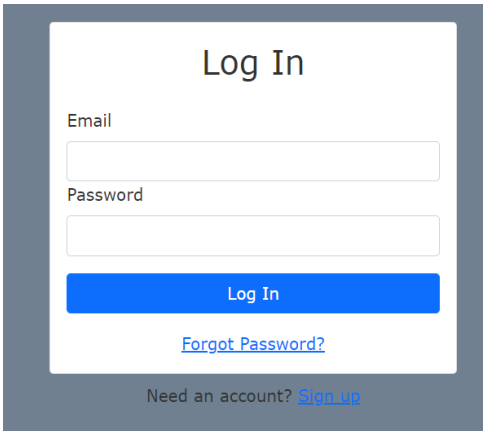
stvara špilove stvaraju se dokumenti koji sadrže ime špila i kolekciju karata. U zadnjoj kolekciji se nalaze samo karte koje imaju podatke potrebne za prikazivanje. Karta nema svoju kolekciju nego samo potrebne podatke za prikaz pri učenju, a to su pitanje, odgovor, slika i ocjena.

4.3. Registracija i prijava

Nakon što je aplikacija spojena s Firebase Cloudstore-om, implementirati će se stranice za registraciju i prijavu korisnika. Za izradu forme u kojoj će se upisivati podatci koristiti će se *react-boostrap* komponente.



Slika 4.7. Prikaz komponente za registraciju



Slika 4.8. Prikaz komponente za prijavu

U komponenti *Button*, kada se kline na tipku *Sign Up* pokreće se funkcija *handleSubmit* koja obrađuje podatke unesene u formi.

```
<Button disabled={loading} className="w-100 mt-3" type="submit">Sign Up</Button>
```

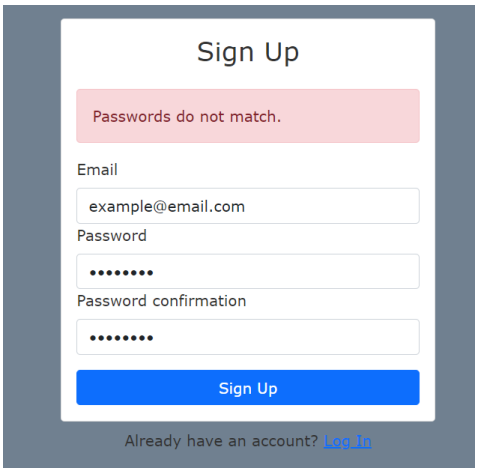
Slika 4.9. Prikaz komponente za prijavu

Funkcija je asinkrona zbog toga što se u trenutku pozivanja funkcije *signup* podatci šalju na Firebase bazu podataka te se mora sačekati da se podatci sa poslužitelja vrate i da izvođenje programa može nastaviti. Pri registraciji bitno je usporediti lozinke koje su unesene da se potvrdi

da su jednake. Ako lozinke nisu jednake u komponenti se pojavi crvena obavijest koja upozorava korisnika da se lozinke ne podudaraju (Slika. 4.9).

```
async function handleSubmit(e) {
    e.preventDefault()
    if (passwordConfirmationRef.current.value !== passwordRef.current.value)
    {
        return setError("Passwords do not match.")
    }
    try {
        setError("")
        setLoading(true)
        await signup(emailRef.current.value, passwordRef.current.value)
        navigate('/')
    } catch (error) {
        setError('Failed to create an account.')
    }
    setLoading(false)
}
```

Slika 4.10. Registriranje korisnika



Slika 4.11. Upozorenje da se lozinke ne podudaraju

Nakon što se korisnik registrira ili prijavi, potrebno ga je postaviti kao aktivnog korisnika koji je trenutno prijavljen i ima samo pristup svojim podatcima. Budući da je to potrebno napraviti samo jednom prilikom registracije ili prijave koristiti će se *useEffect hook*.

4.4. Dohvaćanje i dodavanje podataka

Da bi se podatci mogli dohvaćati i obrađivati potrebno je napraviti funkcije koje će to raditi. Za dohvaćanje će se koristiti funkcija *useEffect* koji definira što komponenta mora napraviti nakon renderiranja ili promjene. Prvi parametar je funkcija, a drugi je vrijednost onoga što se prati. Stanje koje se prati je ono što definira kada će se funkcija izvršiti. Svaki puta kada se dogodi promjena na vrijednosti koja se prati funkcija će se izvršiti. Za drugi parametar postaviti će se prazni niz koje React interpretira tako na način će funkcija izvršiti će se samo jednom jer nije moguće promijeniti prazan argument što osigurava da se uvijek izvrši samo jednom.

```
useEffect(() => {
    const unSubscribe = auth.onAuthStateChanged(user => {
        setCurrentUser(user)
        if (user) {
            setUserDBref(user.uid);
        } else {
            clearuserDBref();
        }
        setLoading(false)
    })
    return unSubscribe
}, [])
```

Slika 4.12. Postavljanje korisnika u prijavljeni status

Ovdje se također postavlja referenca na bazu podataka preko koje se spremaju podatci s kojima prijavljeni korisnik radi. Na toj referentnoj točki se nalazi dokument imenovan po posebnom ID-u koji generira Firebase za svakog korisnika prilikom registracije. Taj dokument sadrži kolekciju svih špilova korisnika.

```
async function createUser() {
  try {
    setDoc(doc(db, "users", auth.currentUser.uid), {
      email: currentUser.email
    })
  }
```

Slika 4.13. Postavljanje korisnika u prijavljeni status

Da bi se moglo sukladno s definiranim pravilima baze podataka (slika 4.3) mogli spremati podatci potrebno je postaviti strukturu baze koja tome odgovara. Pomoću funkcije *createUser* se pod kolekciju *users* stvara novi dokument s UID-em trenutno prijavljenog korisnika.

Za dohvaćanje i dodavanje podataka potrebno je imati referencu na mjesto gdje se nalaze podatci (slika 4.12). Pri dohvaćanju podataka iz baze oni se moraju prilagoditi. TO se čini tako što se stavljaju u niz podataka koje React može interpretirati.

```
export async function getDeck(deckID){
    if(deckRef){
        const userDecks=[]
        const snapshot=await getDocs(deckRef)
        snapshot.forEach((doc)=>{
            userDecks.push(doc.data())
        })
        return userDecks
    }
}
export async function addDeck(NewDeckName){
    if(deckRef){
        const docRef= await addDoc(deckRef, {
            deckName: NewDeckName
        })
    }
}
```

Slika 4.14. Dohvaćanje i dodavanje špilova u bazu podataka

4.5. Struktura komponenti

Bilo koji dio web stranice je u cijelosti predstavljen nekom od komponenata iz React-a. Stranica je strukturirana tako što se navigacijska traka nalazi uvijek na vrhu stranice dok se ispod nalazi sadržaj koji se mijenja.

```
<AuthProvider>
  <Router>
    <Navbar />
    <div className='content'>
      <Routes>
        <Route exact path='/' element={<PrivateRoute />}>
          <Route exact path='/' element={<Profile />} />
        </Route>
        <Route path='/Home' element={<Home />}></Route>
        <Route path='/Profile' element={<Profile />}></Route>
        <Route path='/Home/:id/' element={<DeckPage />}></Route>
        <Route path='/Home/:id/studying' element={<Studying />}></Route>
        <Route path='/Classes' element={<Classes />} />
        <Route path='/DeckList' element={<DeckList />} />
        <Route path='/Deck' element={<Deck />} />
        <Route path='/ForgotPassword' element={<ForgotPassword />}></Route>
        <Route path='/SignUp' element={
          <Container className='d-flex justify-content-center'>
            <div className='w-100' style={{ maxWidth: "400px", minWidth: "150px" }}>
              <SignUp />
            </div>
          </Container>
        }>
        </Route>
        <Route path='/Login' element={
          <Container className='d-flex justify-content-center'>
            <div className='w-100' style={{ maxWidth: "400px", minWidth: "150px" }}>
              <Login />
            </div>
          </Container>
        }>
        </Route>
        <Route path="*" element={<ErrorPage />}></Route>
      </Routes>
    </div>
  </Router>
</AuthProvider>
```

Slika 4.15. Struktura komponenti

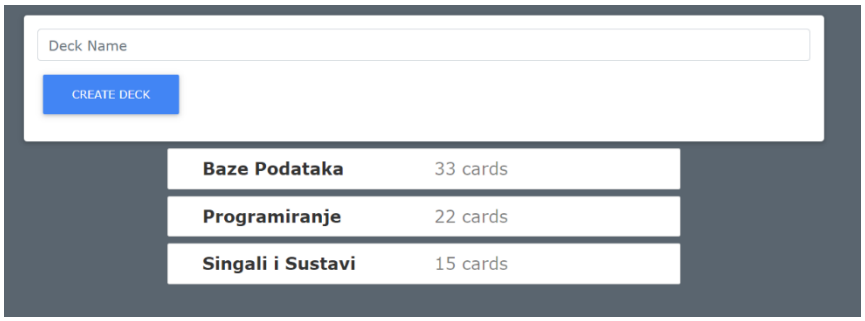
To se postiže tako što se u *App.js* datoteci postavi komponenta *navbar*, a ispod u zaseban *div* tag se stavljaju komponente koje prikazuju sadržaj stranice kao što je prikazano na slici 4.13.

4.6. Špilovi i kartice

Ono što će korisnik najviše koristiti prilikom učenja su špilovi i kartice. Stoga je važno napraviti da je to dio aplikacije koji neće imati greške pri izvođenju. Proces stvaranja kartica i učenja iz njih može biti dugotrajan i zamoran. Bitno je napraviti da je sučelje ima malo tipki i da omogući lako stvaranje kartica i špilova. Pritom mora biti pregledno i postavljeno na način da u sučelju uvijek prikazano samo jedan ili dva interaktivna elementa. U sljedećim poglavljima će biti objašnjeno kako će se to realizirati.

4.6.1. Špilovi

Na stranici *Home* se nalazi sučelje za stvaranje špilova i popis svih špilova što korisnik ima ispod toga. Svaki špil u ovom popisu je poveznica na novu stranicu za taj špil. Nakon što se dohvate svi špilovi iz baze podataka (Slika 4.12) stavljaju se u *useEffect* objekt koji se prosljeđuje komponenti *DeckList*.



Slika 4.16. Prikaz popisa špilova

Zatim komponenta *DeckList* raspodjeljuje sve vrijednosti u niz te se za svaku vrijednost poziva se komponenta *Deck* u kojoj se špil prikazuje i vraćanjem tih vrijednosti nazad prikazuje se traženi popis špilova. Da bi svaki špil bio prikazan u listi i da se može koristiti kao poveznica koristiti će se komponenta *ListGroup* i svojstvo *onClick*.

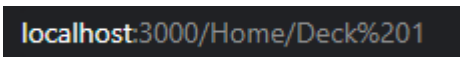
```
import React from 'react'
export default function DeckList({ decks, onDeckClick }) {
    return (
        decks.map(deck=>{
            return <Deck key={deck.id} deck={deck} onClick={onDeckClick}/>
        })
    )
}
export default function Deck({ deck, onClick }) {
    return (
        <ListGroup className='mt-2' onClick={_=>onClick(deck.deckName)}>
            <ListGroup.Item action>
                <Container fluid>
                        <Col>{deck.deckName}</Col>
                </Container>
            </ListGroup.Item>
        </ListGroup>
    )}
```

Slika 4.17. Komponente za stvaranje popisa špilova

4.6.2. Usmjeravanje na stranicu špila

Klikom na bilo koji špil otvara poveznicu s imenom tog špila kao što je prikazano na primjeru na slici 4.16. Pretraživači interpretiraju razmak kao „%20“ znak i zbog toga se tako prikazuje u poveznici. Popis će se sastaviti od dvije komponente, *DeckList i Deck*.

Generiranje poveznica s imenima koje je korisnik generirao postiže se stvaranjem rute koja sadrži svojstvo za primanje parametara. Parametar se dobiva sa klikom na špil iz popisa s kojim se izvršava funkcija *onDeckClick* te se pomoću funkcije *navigate* korisnik usmjerava na prikladnu stranicu. Na svakoj od tih stranica kartice se dohvaćaju uz parametar iz poveznice na špilu u kojoj se korisnik nalazi.



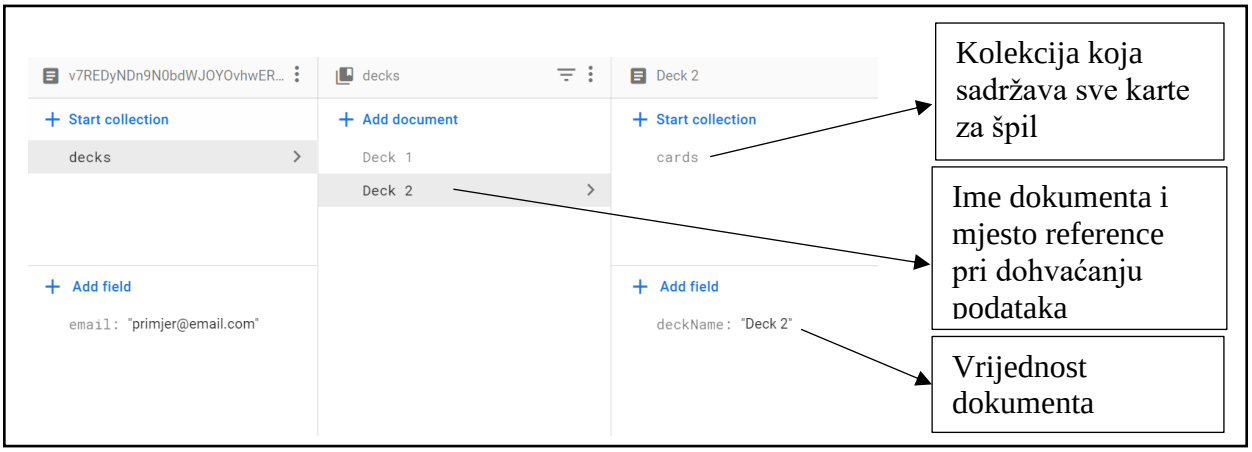
Slika 4.18. Prikaz poveznice za „Deck 1“

```
<Route path='/Home/:id/' element={<DeckPage />}></Route>
  function onDeckClick(deck) {
    navigate(`/Home/${deck}`);
  }
```

Slika 4.19. Usmjeravanje na stranicu za špil

4.6.3. Špilovi u bazi podataka

Klikom na *Create Deck* izvršava se funkcija *HandleNewDeck*. Prilikom stvaranja dokumenata u Firebase bazu podataka važno je postaviti točnu referencu na mjesto gdje će se stvoriti. Uz referencu na popis svih špilova korisnika još će se postaviti ime špila pomoću funkcije *doc* s kojom će se imenovati dokument koji se generira u Firebase kolekciji.



Slika 4.20 Prikaz sučelja gdje su spremljeni podatci za špilova i kartice

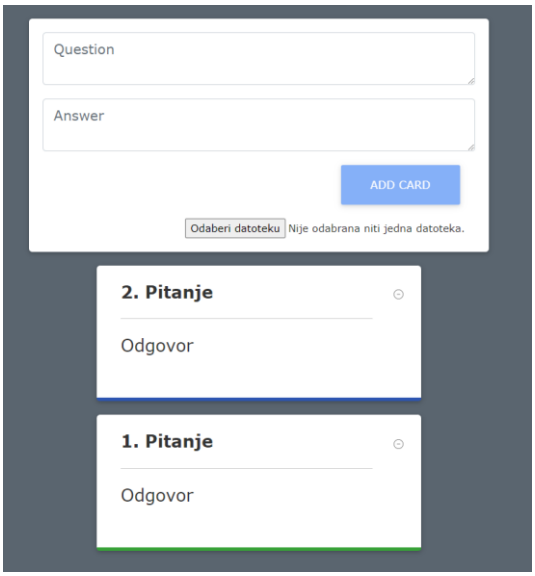
Treba razlikovati ime dokumenta i ime dodijeljene vrijednosti za taj špil. Ime dokumenta u Firebase-u se koristi za postavljanje reference s koje će se podatci dohvaćati, a za prikazivanje ime špila na stranici koristi se vrijednost *deckName*.

```
decksRef = collection(db, "users", userID, "decks");
function HandleNewDeck(e) {
    e.preventDefault()
    createUser()
    try {
      addDeck(deckNameRef.current.value);
    }
    catch (error) { console.log(error) }
    navigate(`/Home/${deckNameRef.current.value}`);}
{decks && <DeckList decks={decks} onDeckClick={onDeckClick} />}
```

Slika 4.21. Prikaz koda za izradu i prikazivanje špilova.

4.6.4. Kartice

Klikom na špil otvara se nova stranica kao što je prikazano na slici 4.20. gdje se nalazi sučelje za dodavanje kartica. Ispod se nalazi lista svih kartica koja se automatski osvježi kada se napravi nova kartica. Za izradu kartica koristiti će se ista logika kao i za izradu špilova.



Slika 4.22. Prikaz sučelja za izradu i prikazivanje kartica

Da bi se spriječilo dodavanje praznih kartica u komponentu *Button* je postavljeno svojstvo *disabled* (Slika 4.21) koje onemogućuje tipku dok je polje za pisanje pitanja prazno. S time se čuva baza podataka od koji su nepravilni i sprječavaju se moguće pogreške pri izvođenju.

```
<Button disabled={question.length === 0} type="submit">Add Card</Button>
```

Slika 4.23. Upravljanje tipkom za izradu

4.6.5. Prikaz kartica za učenje

Tipka *Start Studying* će usmjeriti korisnika na stranicu gdje će se svaka kartica prikazivati jedna po jedna. Čim se stranica učita odmah se prikazuje prva kartica u nizu i sa tipkom *Next* renderirati će se sljedeća kartica iz špila. Nakon što korisnik dođe do kraja kartice će se početi prikazivati ispočetka.

Neće se koristiti komponenta za popis kartica nego će se podatci spremiti u polje kao komponente koje su spremne za renderiranje.

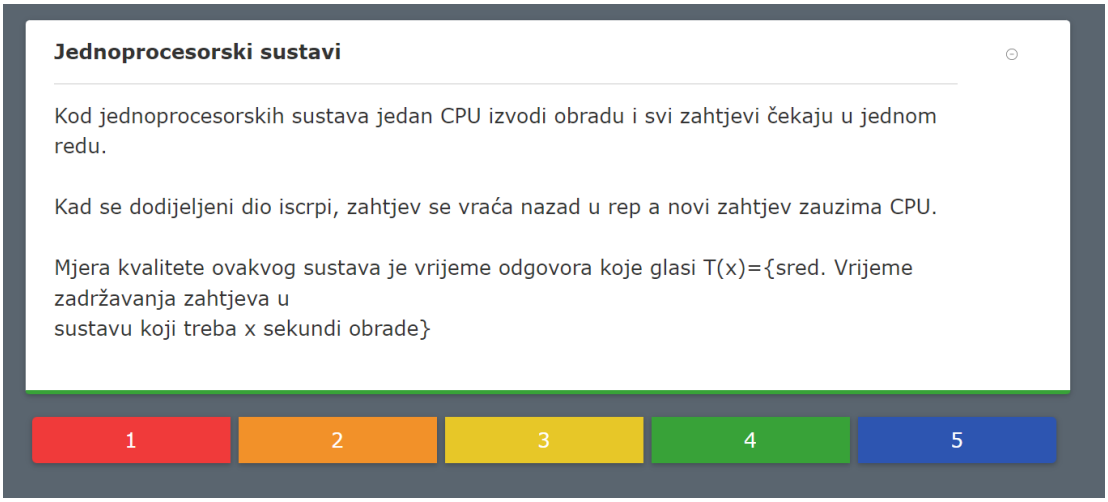
```
const StudyCards = flashcards.map(flashcard => {
  return <Flashcard flashcard={flashcard}></Flashcard>
})

function nextQuestion(event,param) {
  changeGrade(deck.id,flashcards[currentCard], param)
  updateCards();
  const nextCard = currentCard + 1;
  if (nextCard < StudyCards.length) {
    setCurrentCard(nextCard)
  } else {
    setCurrentCard(0)
  }
}
    <Container >
      <div className='h-100'>
        {StudyCards[currentCard]}
          <Button onClick={nextQuestion} size="lg">Next</Button>
      </div>
    </Container>
```

Slika 4.24. Kod za prikazivanje kartica

Svaka kartica se prikazuje jedna po jednu s ocjenama ispod nje. Ocjene imaju pridijeljene boje. Učenje bi trebalo idealno funkcionirati tako što korisnik pročita pitanje pokuša bez gledanja odgovoriti i onda pogledati odgovor te onda se samostalno ocijeniti. Nakon što klikne na jednu od ocjena, kartica će zapamtiti ocjenu i sljedeći puta kada se korisniku prikaže kartica bit će označena tom bojom da naznači korisniku kako je odgovorio zadnju puta. Naznačeno je prikladnom bojom ispod kartice, a pri tome nije distrakcija pri učenju i lagano je vidjeti kako je zadnji put ocjenjena ta kartica. Glavna smisao toga je da se potakne daljnje učenje jer se nakon svake kartice vidi

napredak. Tekst pitanja je uvijek prikazan podebljanim slovima zbog čitljivosti. Ispod odvojeno crtom se nalazi odgovor za koji je bitno da podržava stvaranje novih linija zbog preglednosti.



Slika 4.25. Izgled kartice pri učenju

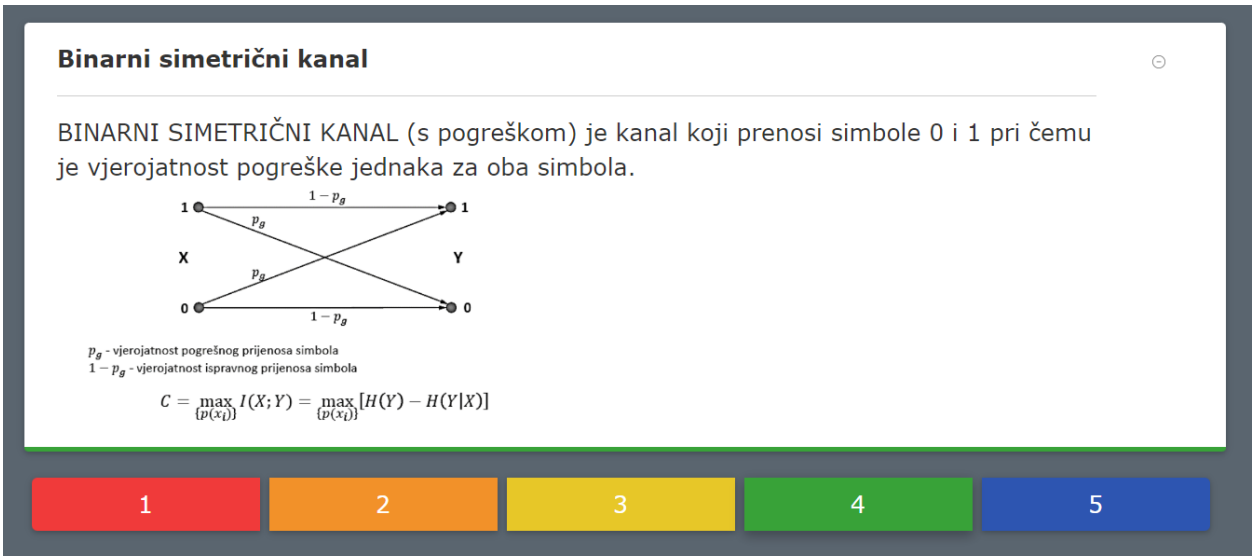
4.7. Dodavanje slika

Često prilikom učenja potrebno je učiti iz nekakve slike ili grafa, stoga nije dovoljno imati samo opciju za dodavanje kartica tekstom, također je implementirano stavljanje slika u kartice. U Firebase-u nije moguće dodavanje slika zajedno pitanjem i odgovorom. Firebase sprema datoteke u poseban *Storage* odjel gdje generira za svaku datoteku poveznicu na koju se kasnije može pozvati. Također je potrebno imati strukturu za datoteke, u ovom slučaju je potrebno samo imati za svakog registriranog korisnika jednu mapu u kojoj će se nalaziti sve slike.

```
if (file != null) {
        const path = `/${currentUser.uid}/${file.name}`;
        const storageRef = ref(storage, path);
        const uploadTask = uploadBytesResumable(storageRef, file);
        const downloadURL = await getDownloadURL(uploadTask.snapshot.ref);
        image = downloadURL;
        setFile(null);
      }
<input type="file" ref={fileRef} onChange={handleImageAsFile} className="input"/>
```

Slika 4.26. Kod za dodavanje slika

Preko *input* elementa se stavlja slika koja se sprema u varijablu *file* pomoću funkcije *setFile*. Varijabla *path* se koristi za definiranje mjesta u Firebase-u gdje će se datoteka spremiti. Funkcija *uploadBytesResumble* učitava datoteku u Firebase i nakon što je datoteka pohranjena moguće ju je dohvatiti pomoću funkcije *getDownloadURL* koja vraća poveznicu za tu datoteku.



Slika 4.27. Izgled kartice sa slikom

5. ZAKLJUČAK

Izradom projekta za pomoć pri učenju je postignuta je funkcionalna web aplikacija za izradu kartica. Cilj je bio olakšati stvaranje kartica koje su se inače pravile s papirima te pružiti pogodan način za učiti iz njih. Već postoje mnoga rješenja, ali često imaju mane koje predstavljaju problem pri korištenju i učenju. Iz tih problema su se definirali glavni ciljevi za izradu ove aplikacije. Problem koji je riješen primarno uz pomoć React-a je jednostavnost izrade kartica uz mogućnost dodavanja slika. Sljedeći problem što je bilo važno riješiti je izradu čistog i jednostavnog sučelja iz kojeg je lagano učiti. To je postignuto tako što prilikom korištenja aplikacije u svakom trenutku postoji minimalni broj interaktivnih elemenata koji su uvijek u sredini zaslona.

Firebase baza podataka se pokazala kao izvrsni pozadinski sustav koji je upravljao spremanjem podataka i autorizacijom korisnika. Pružila je mnoge funkcije koje su olakšale pisanje koda za upravljanje podatcima i korisnicima. Dodavanje je bilo postignuto implementiranjem funkcija koje su se povezivale na bazu podataka i koje su se mogle pozivati bilo gdje u kodu.

React se isto pokazao kao izvrsni jezik za stvaranje bilo kojeg vizualnog elementa. Stvaranjem komponenata s kojima su se prikazivale špilovi i kartice cijela aplikacija se brzo realizirala. Osim što je pružilo laganu izradu ponovno iskoristivih elemenata, također je bilo jednostavno kontrolirati stanja i omogućiti interakciju. Također je izrada aplikacije znatno olakšana s mnogim mogućnostima i proširenjima Visual Studio Code programa. Aplikacija je brza i jednostavna za koristiti.

Preglednost i dodatna uređivanja su postignuta programskim okvirom *React-boostrap* koji je znatno pomogao i ubrzao taj proces jer omogućuje mnoga već definirana oblikovanja koja su se koristila da bi se uredio izgled kartica. Zbog samog načina funkcioniranja *React-a,* ako bi se za ovu odlučilo postaviti nove značajke, bilo bi ih jednostavno implementirati u obliku novih *React* komponenti.

LITERATURA

[1] FlashCards World, Andev [online], dostupno na: https://flashcards.world/, pristup: 26.06.2022

[2] Cram, Cram LLC., [online] , dostupno na: https://www.cram.com/, pristup: 26.06.2022

[3] GoConqr, ExamTime Ltd [online], dostupno na: https://www.goconqr.com/, pristup: 26.06.2022

[4] A. Sutherland, Quizlet, Quizlet Inc. [online], dostupno na: https://quizlet.com/, pristup: 26.06.2022

[5] A. Cohen, Brainscape, Bold Learning Solutions [online], web stranica za učenje, dostupno na: https://www.brainscape.com/subjects, pristup: 26.06.2022

[6] Ryan Dahl, node.js, 27.05.2022, resursi [online], dostupno na: https://en.wikipedia.org/wiki/Node.js, pristup; 15.05.2022

[7] NPM.js, GitHub, npm, Inc. [online], 12.01.2010, dostupno na: https://www.npmjs.com/, pristup: 15.5.2022

[8] React, GitHub [online], 29.05.2013,dostupno na: https://reactjs.org/, pristup: 17.08.2022

[9] GitHub, resursi i dokumentacija [online], dostupno na: https://github.com/facebook/react, pristup: 17.08.2022

[10] React Bootstrap, resursi i dokumentacija [online], dostupno na: https://react-bootstrap.github.io/, pristup: 15.08.2022

[11] React Router, resursi i dokumentacija [online], dostupno na: https://v5.reactrouter.com/, pristup: 17.06.2022

[12] A.Lee, Firebase, resursi i dokumentacija [online], Google LLC, 2011, dostupno na: https://firebase.google.com/docs/firestore, pristup: 17.08.2022

[13] Visual Studio Code, resursi i dokumentacija [online], Microsoft, 25.04.2015 https://code.visualstudio.com/, pristup: 26.06.2022

SAŽETAK

U već postojećim rješenjima za izradu kartica uočeno je da je često nemoguće postaviti slike u kartice i može se primijetiti da često imaju elemente koje su distrakcija pri učenju. Na osnovu toga u okviru ovog rada izrađena je web aplikacija za pomoć pri učenju koja pruža pregledno sučelje i mogućnost dodavanja slika. Realizirana je uz dvije glavne tehnologije, *React* i *Firebase.* U radu u objašnjene navedene tehnologije, objašnjeno je kako se postavljaju i kako su se koristile da se dobiju željeni rezultati. Aplikacija omogućuje registraciju korisnika, stvaranje špilova i kartica, brisanje kartica, pokretanje načina rada za učenje i ocjenu. Za kartice je bitna značajka da se uz tekst može postaviti proizvoljna slika. Kartice za svakog korisnika pohranjuju se u bazu podataka.

Ključne riječi: Firebase, kartice za učenje, React, špil

ABSTRACT

Web-based flashcard management software for learning support

In existing web-based flashcard management applicaions it is often impossible to place images in the flashcards. Also they often have elements that distract user while learning. Based on these flaws, the application in this work provides a clear interface and the ability to add images to the flashcards. The application has been implemented using React and Firebase. These technologies were elaborate din this thesis, and it was also explained how they were set up and used to get the desired results. The application allows users to register, create decks and cards, delete cards and start learning mode and to grade. An important feature is the ability to add an arbitrary image next to the text. Cards for each user are stored in the database.

Keywords: Firebase, flashcards for studying, React, deck

PRILOZI

Prilog P.4

Kod web aplikacije nalazi se na poveznici : https://github.com/APuhanic/flashcards