Primjena LLM-a u razumijevanju i generiranju savjeta o prehrani

Tokić, Mateo

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

Sveučilišni diplomski studij Računarstvo

Primjena LLM-a u razumijevanju i generiranju savjeta o prehrani

Diplomski rad

Mateo Tokić

Osijek, 2024

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

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Mateo Tokić
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1333R, 07.10.2022.
JMBAG:	0165083586
Mentor:	izv. prof. dr. sc. Časlav Livada
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Alfonzo Baumgartner
Član Povjerenstva 1:	izv. prof. dr. sc. Časlav Livada
Član Povjerenstva 2:	doc. dr. sc. Tomislav Galba
Naslov diplomskog rada:	Primjena LLM-a u razumijevanju i generiranju savjeta o prehrani
Znanstvena grana diplomskog rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak diplomskog rada:	U radu je potrebno napraviti aplikaciju koja će nakon fotografiranja nutritivnih vrijednosti i sadržaja proizvoda primjenom LLM-a (Large language model) generirati savjete o prehrani. Tema rezervirana za: Mateo Tokić

Datum ocjene pismenog dijela diplomskog rada od strane mentora:	01.12.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	11.12.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	11.12.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 11.12.2024.

Ime i prezime Pristupnika:	Mateo Tokić
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1333R, 07.10.2022.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Primjena LLM-a u razumijevanju i generiranju savjeta o prehrani**

izrađen pod vodstvom mentora izv. prof. dr. sc. Časlav Livada

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 pristupnika:

SADRŽAJ

1. UVOD

U današnjem svijetu sve veća se pažnja posvećuje zdravom načinu života koji podrazumijeva redovite tjelesne aktivnosti i zdravu prehranu. Ljudima svih dobnih skupina, prisiljenih modernim načinom života, smanjene su fizičke aktivnosti, a mnogi na poslu provode u sjedećem položaju većinu radnog vremena. Osim toga, prehrana im se temelji na velikom unosu ugljikohidrata i pržene hrane koja sadrži puno masti. Razvojem tehnologije, sve je više digitalnih alata i aplikacija koje nastoje pomoći korisnicima praćenje prehrambenih navika i optimizaciju unosa hranjivih tvari. Tradicionalne aplikacije za praćenje prehrane često se oslanjaju na jednostavne baze podataka ili ručni unos podataka, što može biti zamorno i sklono pogreškama. S pojavom naprednih tehnologija iz područja strojnog učenja i umjetne inteligencije, poput velikih jezičnih modela (engl. LLM - *Large Language Models*), otvara se mogućnost za unaprjeđenje tih aplikacija kroz moderno razumijevanje i generiranje personaliziranih savjeta o prehrani.

LLM-ovi, kao što su GPT (engl. *Generative Pre-trained Transformer*) modeli, predstavljaju suvremeni pristup obradi prirodnog jezika, omogućujući računalnim sustavima da razumiju i generiraju tekst na način koji oponaša ljudsku komunikaciju. Primjenom ovih modela u području prehrane, moguće je razviti aplikacije koje ne samo da interpretiraju nutritivne informacije, već i pružaju korisnicima personalizirane savjete temeljene na njihovim individualnim podacima, potrebama, preferencijama i zdravstvenim ciljevima. Ova tehnologija analizira slike nutritivnih vrijednosti prehrambenih proizvoda te automatski generira preporuke za obroke, čime se značajno pojednostavljuje proces donošenja odluka vezanih uz prehranu.

Cilj ovoga rada je istraživanje primjene velikih jezičnih modela u kontekstu pružanja savjeta o prehrani te razvijanje aplikacije koja na osnovu korisničkih podataka i slika nutritivnih vrijednosti generira korisne i personalizirane preporuke za obroke. Aplikacija koristi model optičkog prepoznavanja znakova (engl. OCR - *Optical Character Recognition*) za prepoznavanje teksta na slikama prehrambenih proizvoda te LLM za analiziranje unesenih informacija i prilagodbu savjeta specifičnim potrebama korisnika, čime se postiže viši stupanj personalizacije u usporedbi s postojećim sličnim rješenjima na tržištu.

Rad je podijeljen u nekoliko poglavlja. Drugo poglavlje pruža pregled sličnih aplikacija, dok treće poglavlje opisuje tehnologije upotrijebljene za izradu aplikacije i rješavanje ovoga zadatka. Četvrti dio prikazuje razvoj i implementaciju aplikacije. Na kraju rada opisano je testiranje aplikacije i prikaz rezultata, a zaključni dio raspravlja o izazovima i potencijalnim smjerovima za budući razvoj.

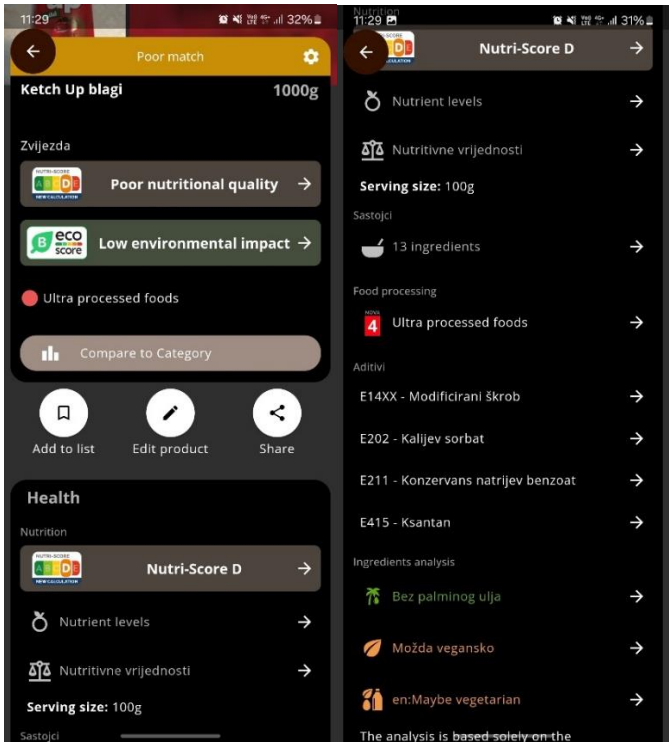
Kroz ovu aplikaciju, nastoji se demonstrirati kako napredni jezični modeli mogu unaprijediti korisničko iskustvo u području prehrane, pružajući brže, točnije i personalizirane savjete koji su u skladu s modernim trendovima u prehrani i zdravom životu.

2. POSTOJEĆA RJEŠENJA

U ovom poglavlju objašnjeno je i uspoređeno nekoliko aplikacija koje nude prehrambene savjete i omogućuju korisnicima praćenje nutritivnih vrijednosti i unosa kalorija tijekom dana, s obzirom na korisnikove potrebe i ciljeve. Aplikacija koja je napravljena i predstavljena u ovome radu nudi savjete na osnovu LLM-a te se razlikuje u odnosu na sljedeće primjere.

2.1. OpenFoodFacts

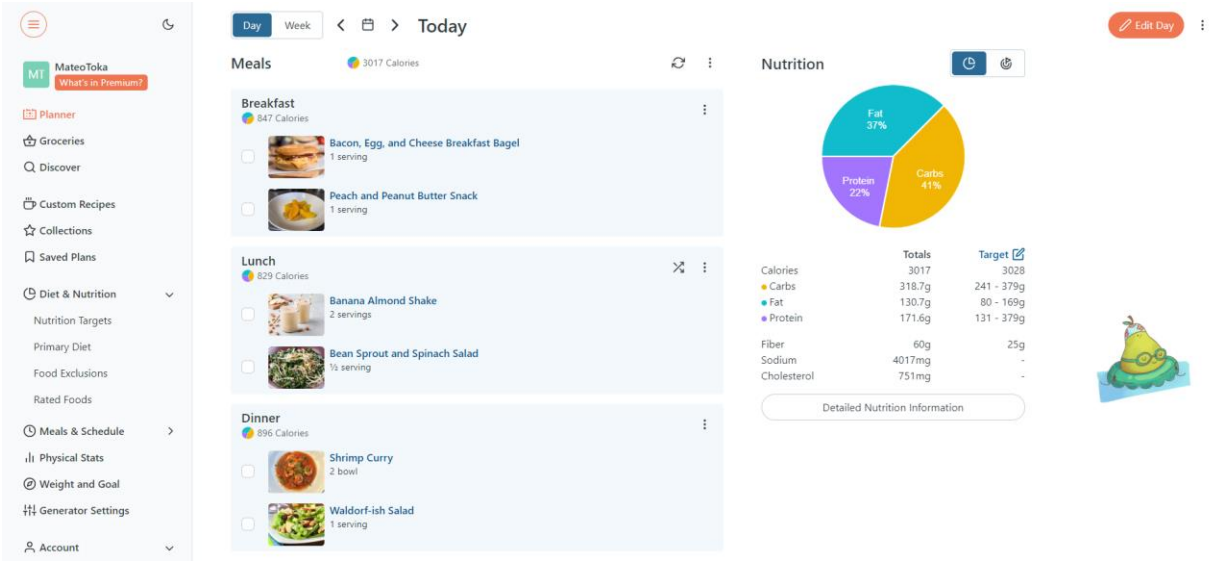
OpenFoodFacts je baza podataka s prehrambenim proizvodima, a svatko ima mogućnost dodavanja i korištenja njezinih podataka [1]. Dostupna je također i mobilna aplikacija čija je glavna značajka skeniranjem barkoda proizvoda prikazati korisniku sastojke, nutritivne vrijednosti i velik broj ostalih podataka koji opisuju nutritivne kvalitete toga proizvoda uz moguću usporedbu s drugim proizvodima. Korisniku je omogućena prijava, a u postavkama aplikacije nalazi se forma gdje se određenim prehrambenim vrijednostima dodjeljuje važnost prikaza kako bi se prilikom odabira prehrambenog proizvoda informacije s većom važnosti prve prikazale. Uz to moguće je i odabrati ima li korisnik alergije na neke proizvode, koliko mu je bitan utjecaj proizvoda na okoliš te je li vegan ili vegetarijanac. Slika 2.1. prikazuje sučelje aplikacije kada se skenira barkod prehrambenog proizvoda.



Slika 2.1. *Informacije o skeniranom proizvodu*

2.2. Eat This Much

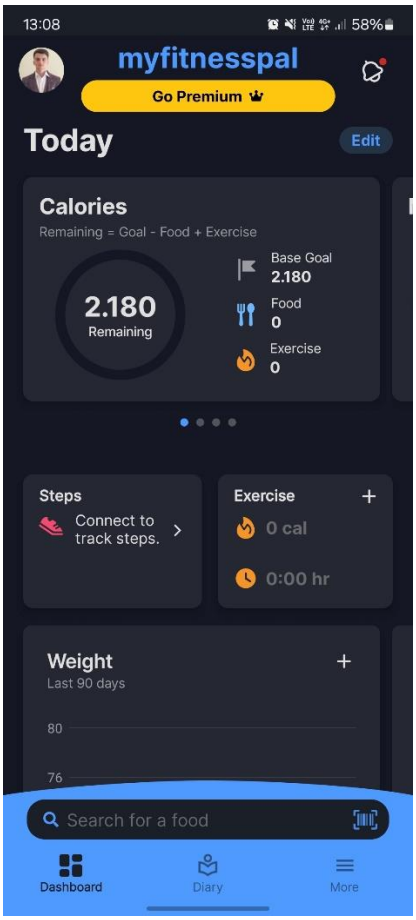
Eat This Much web je i mobilna aplikacija koja korisniku nudi personalizirane planove obroka na temelju korisnikovih preferencija hrane [2]. Služi korisniku kao planer za izradu dnevnih menija i plana prehrane s obzirom na njegov profil s osobnim podacima koji podrazumijevaju njegovu masu, visinu, godine i tjelesnu aktivnost. Aplikacija također nudi prikaz nutritivnih vrijednosti različitih prehrambenih proizvoda, a plan prehrane korisniku je prilagođen s obzirom na njegovu ciljanu kilažu i moguće alergije i specifičnost njegove ishrane. Na slici 2.2. prikazan je dnevni meni aplikacije preporučen autoru ovoga diplomskog rada.



Slika 2.2. *Dnevni meni Eat This Much aplikacije*

2.3. MyFitnessPal

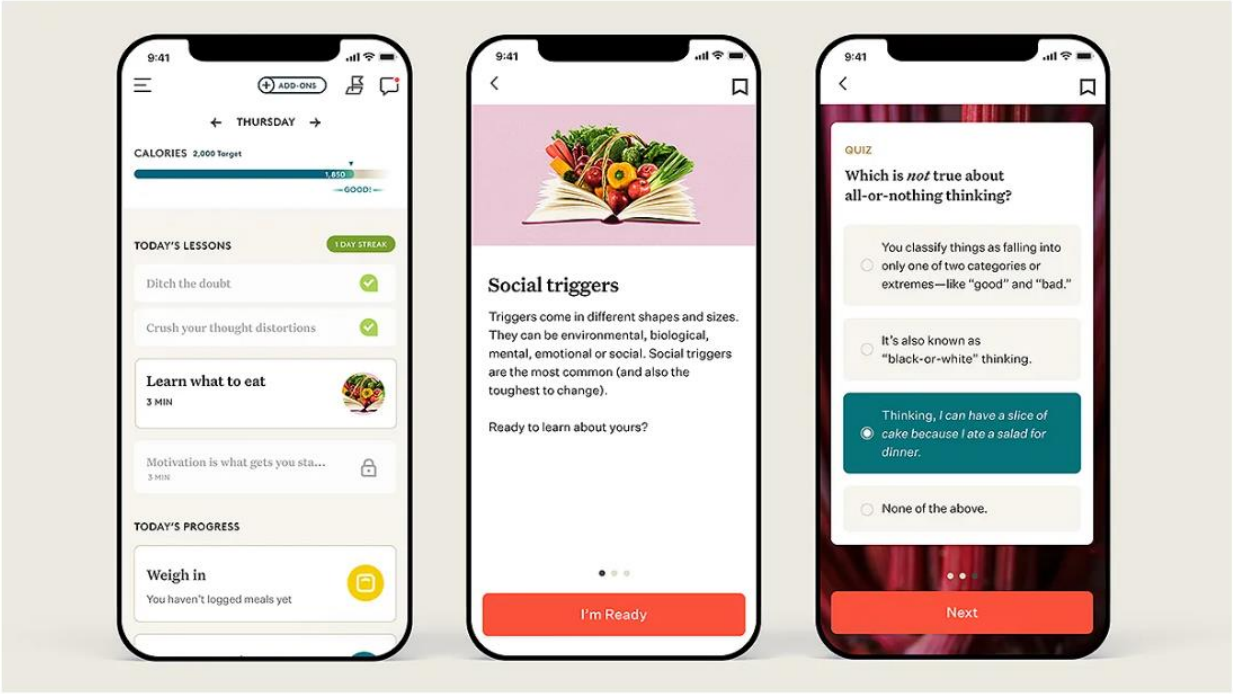
MyFitnessPal aplikacija korisniku nudi preporuku dnevnog unosa kalorija i nutritivnih vrijednosti na osnovu njegovog profila i dnevne aktivnosti. Velika prednost aplikacije je ta što korisnik ima mogućnost izbora između brojnih planova koji su prilagođeni različitim pojedincima i njihovim potrebama [3]. Za potpunu analizu namirnica i preporuku nutrijenata po obroku potrebno je pretplatiti se na Premium korisnički račun. Prikaz početnog zaslona aplikacije nakon unosa osobnih podataka prikazan je na slici 2.3.



Slika 2.3. *Početni zaslon MyFitnessPlan aplikacije*

2.4. Noom

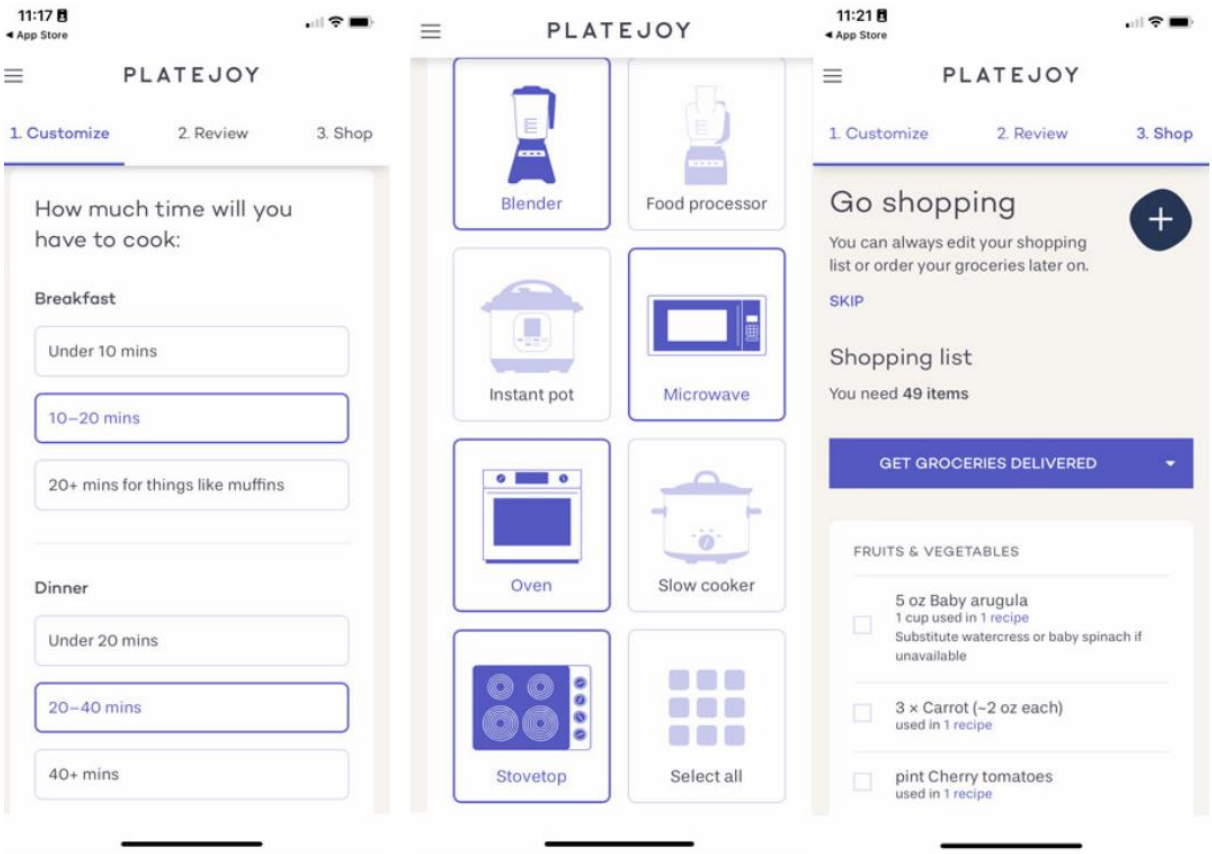
Noom je aplikacija dostupna za Android i iOS uređaje koja nudi besplatno korištenje u razdoblju od 14 dana, a nakon toga zahtjeva mjesečnu pretplatu. Zanimljivost je ta što korisnik rješavajući svakodnevne lekcije stvara zdrave prehrambene navike [3]. Osim toga nudi i alate za praćenje napretka, a tim *Noomove* zajednice omogućava individualiziranu podršku preko telefonski poziva i poruka unutar aplikacije. Na slici 2.4. prikazani su zasloni *Noom* aplikacije.



Slika 2.4. *Noom aplikacija* [4]

2.5. PlateJoy

PlateJoy još je jedna mobilna aplikacija koja koristi upitnike za prikupljanje detalja o korisnikovom životnom stilu i navikama kuhanja, a zatim stvara prilagođeni plan obroka i popis namirnica za domaćinstvo [5]. Aplikacija je korisna ukoliko korisnik često kuha kod kuće te ima problema s novim receptima ili se bori s bacanjem hrane radi neorganiziranog plana kupnje namirnica [6]. Navedene funkcionalnosti prikazane su na snimkama zaslona na slici 2.5.

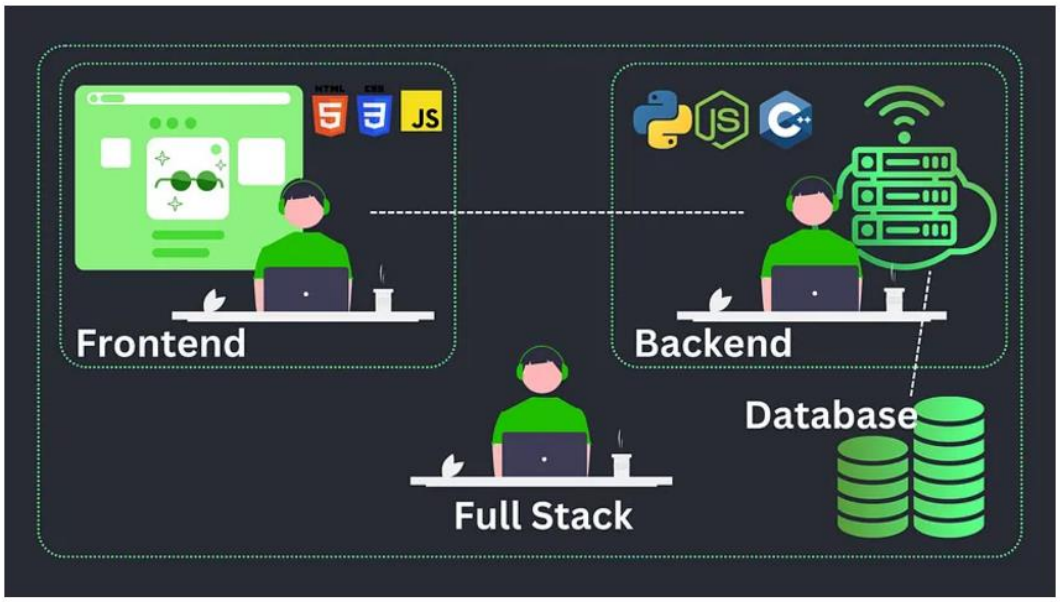


Slika 2.5. *Zasloni PlateJoy aplikacije* [6]

3. TEHNOLOGIJE I ALATI

U razvoju ove nutritivne mobilne aplikacije korišten je skup alata i tehnologija koji omogućuju jednostavno i praktično korisničko iskustvo, ali i sigurnu i kvalitetnu obradu podataka. Aplikacija zahtjeva osiguranje prijave korisnika u sustav, ispunjavanja njegovih osobnih podataka relevantnih za nutritivne savjete, mijenjanje istih te mogućnost dobivanja savjeta o obroku i prehrambenim proizvodima na osnovu slike sastojaka i tablice nutritivnih vrijednosti.

U ovom poglavlju objašnjene su tehnologije i alati za izradu *full stack* mobilne aplikacije. Razvoj *full stack* aplikacije ujedinjuje komponente korisničkog sučelja i komponente logike i obrade podataka [7] kao što je i prikazano slikom 3.1.



Slika 3.1. *Razvoj full stack aplikacije* [8]

3.1. Frontend

Klijentska strana aplikacije (engl. *Frontend*) definira korisničko sučelje (engl. UI - *User Interface*), tj. ono što korisnik vidi na svome mobilnom uređaju pri korištenju aplikacije i interakcije sa serverskom stanom. Kao dio stvaranja privlačnog korisničkog sučelja razvoj *frontend* dijela aplikacije temelji se na odabiru specifičnih interaktivnih elemenata dizajna kao što su boje i stilovi teksta, tablice, slike i gumbi [9]. Odabirom navigacijskih izbornika, skočnih obrazaca i ostalih interaktivnih elemenata utječe se na vizualnu privlačnost aplikacije i elegantno korištenje aplikacije. Samim time poboljšava se korisnikovo iskustvo (engl. UX - *User Experience*) i postiže

veća konkurentnost na tržištu. Uz HTML i CSS, JavaScript je jedan od najzastupljenijih jezika i tehnologija kojima se ostvaruju elementi aplikacije s kojima je korisnik u interakciji.

3.1.1. React Native

React Native ubraja se među najpopularnije radne okvire za razvoj višeplatformnih aplikacija temeljen na JavaScript programskom jeziku. Velika prednost ovog radnog okvira je izvoz aplikacija iz istoga koda na različite platforme, uključujući iOS i Android. Razvijen od strane Facebook-a 2015. godine kao projekt otvorenog-koda [10], temelji se na *React* biblioteci čime je programerima omogućena interakcija s postojećim izvornim kodom. Dodatne prednosti su velika dokumentacija i velik broj gotovih rješenja.

Za kreiranje izgleda sučelja koristi se JSX (JavaScript XML) sintaksno proširenje koje omogućuje pisanje oznaka nalik HTML unutar JavaScript datoteci [11]. Pisanjem oznaka i logike korisničkog sučelja unutar iste datoteke, pruža se jednostavnije i lakše održavanje programskog koda.

3.2. Backend

Razvoj serverskog dijela aplikacije (engl. *Backend*) podrazumijeva razvoj na poslužiteljskoj strani programa fokusirajući se na ono što korisnik ne vidi na svojoj aplikaciji. *Backend* programeri u svojemu poslu rukuju pohranom podataka, obavljaju obradu korisničkih unosa, upravljaju sigurnosnim mjerama i osiguravaju funkcioniranje cjelokupne aplikacije [12]. Kako bi se omogućila komunikacija i interakcija između sučelja i pozadine aplikacije, koristi se aplikacijsko programsko sučelje (engl. API - *Application Programming Interface*). API definira pravila i protokole za interakciju različitih sustava, pružajući standardizirani način za programere da integriraju različite usluge i funkcionalnosti u svoje aplikacije [12].

3.2.1. Python Flask

Flask je mikro web okvir za Python koji je dizajniran da bude jednostavan i fleksibilan, što ga čini izuzetno popularnim izborom za razvoj manjih aplikacija i API servisa. Temelji se na filozofiji minimalizma, pružajući osnovne funkcionalnosti potrebne za razvoj aplikacija, dok dodatne značajke mogu biti uključene putem ekstenzija koje proširuju njegove mogućnosti [13]. Prednost *Flaska* je u tome što omogućava programerima da slobodno oblikuju strukturu projekta prema vlastitim potrebama, čineći ga prilagodljivim za različite aplikacije. Podsustav sučelja pristupnika web poslužitelja (engl. WSGI - *Web Server Gateway Interface*) omogućen je od strane *Werkzeug-a*.

U *Flasku* ne postoji izvorna podrška za pristup bazama podataka, provjeru web obrazaca, autentikaciju korisnika ili druge zadatke visoke razine, već su takve ključne usluge dostupne putem ekstenzija koje se integriraju s jezgrenim paketima. [13]

3.3. Firebase

Firebase je cjelovita platforma za razvoj aplikacija koja nudi *backend* usluge i alate za jednostavnu integraciju funkcionalnosti u aplikaciji. Omogućuje usluge autentikacije korisnika, baze podataka u stvarnom vremenu unutar oblaka (engl. *cloud*), pohranu multimedijskih sadržaja i alate za praćenje performansi, hosting i treniranje modela strojnog učenja. Usluge je moguće integrirati za razne aplikacije i tehnologije, uključujući Android, iOS, JavaScript, Node.js, Unity i druge [14].

3.3.1. Firebase Authentication

Pohrana informacija o identitetu korisnika prisutna je u gotovo svim aplikacijama. Osim što pružatelju usluga omogućava poznavanje korisničkog identiteta, također pruža i personalizirano korisničko iskustvo na svim korisnikovim uređajima. *Firebase Authentication* pruža pozadinske usluge i gotove UI biblioteke za autentikaciju korisnika u aplikaciji. Autentikacija je podržana korištenjem lozinki, telefonskih brojeva te ostalih popularnih pružatelja identiteta kao što su Google, Facebook, Twitter i drugi [15]. Omogućena je integracija *Firebase Authentication* usluge s drugim *Firebase* uslugama. Ova usluga koristi industrijski standarde poput *OAuth* 2.0 i *OpenID Connect*, tako da je moguća jednostavna prilagodba s različitim *backendom* [15].

3.3.2. Cloud Firestore

Cloud Firestore je fleksibilna, skalabilna noSQL baza podataka za razvoj poslužitelja te razvoj mobilnih i web aplikacija uz besprijekornu integraciju s drugim Google Cloud i *Firebase* proizvodima [16]. Održava sinkronizaciju korisnikovih podataka u klijentskim aplikacijama u stvarnom vremenu i nudi izvanmrežnu podršku za web i mobilne uređaje [16].

Prema *NoSQL* modelu podataka u *Cloud Firestoreu,* podaci se pohranjuju u dokumentima koji sadrže polja s pridruženim vrijednostima. Dokumenti su organizirani unutar zbirki koje omogućuju strukturiranje podataka i izradu upita. Podržavaju različite vrste podataka, od jednostavnih nizova i brojeva do složenih, ugniježđenih objekata. Unutar dokumenata moguće je kreirati i podzbirke te graditi hijerarhijske strukture podataka koje se lako skaliraju s rastom baze podataka. Bitna prednost koju *Firestore* omogućava kada je riječ o ažurnosti podataka aplikacije, svakako su slušatelji u stvarnom vremenu. Njihovom upotrebom nije potrebno učitavati cijelu bazu podataka pri svakoj promjeni, jer se prilikom promjene dohvaćaju samo promijenjeni podaci [16].

Ova robusna struktura podataka i mogućnosti sinkronizacije u stvarnom vremenu čine *Cloud Firestore* jednim od najpouzdanijih rješenja za upravljanje podacima za mobilne i web aplikacije, koje zahtijevaju visoku prilagodljivost, učinkovite upite i stalnu dostupnost podataka.

3.4. Veliki jezični modeli

Veliki jezični modeli (engl. LLM - *Large Language Models*) kategorija su modela treniranih na velikim količinama podataka što ih čini sposobnima za razumijevanje i generiranje prirodnog jezika i drugih vrsta sadržaja širokog raspona zadataka [17]. Razvojem modela strojnog učenja, algoritama, neuronskih mreža i transformator modela, koji pružaju arhitekturu za sve sustave umjetne inteligencije, mnoge su tvrtke provele godine implementirajući LLM na različitim razinama kako bi poboljšale mogućnosti razumijevanja prirodnog jezika (engl. NLU - *natural language understanding*) i obrade prirodnog jezika (engl. NLP - *natural language processing*). Veliki jezični modeli zahvaljujući milijardama parametara imaju sposobnosti generiranja koherentnih i kontekstualno relevantnih odgovora, zaključivanja iz konteksta, prevođenja na druge jezike, odgovaranja na pitanja, sažimanja teksta, pa čak i pomoći pri generiranju programskog koda.

3.4.1. Duboko učenje

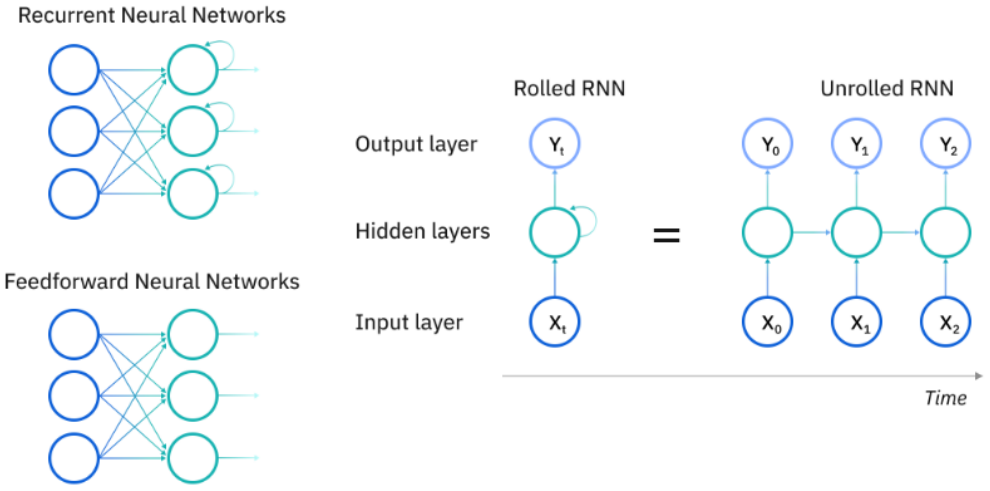
Duboko učenje (engl. *deep learning*) se ubraja među najbrže rastuća i najinovativnija područja računalne znanosti. Pripada grani strojnog učenja, a za razliku od višestrukih perceptrona arhitekture modela korištenih za duboko učenje sadrže mnogo skrivenih slojeva [18].

3.4.2. Obrada prirodnog jezika

Obrada prirodnog jezika jedan je od ključnih koncepata LLM-a čiji se razvoj temelji na algoritmima i modelima kojima bi se omogućilo strojevima da obrađuju i analiziraju tekstualne i govorne podatke na način sličan ljudima. Ovo interdisciplinarno područje znanja iz područja računalnih znanosti ujedinjuje i koristi znanja sintakse i semantike preuzeto iz lingvistike. Tehnike iz područja sintakse za cilj imaju prepoznati početak i kraj rečenice, analizirati redoslijed riječi u rečenici te odrediti korijene riječi i raščlaniti riječi na morfeme [19]. S druge strane, riječima je potrebno dodijeliti njihovo značenje na osnovu rezultata tehnika sintakse. Stoga, prvo je potrebno prepoznati imenovane entitete te ih svrstati po sličnosti u unaprijed postavljene grupe. Nakon toga procesom generacije prirodnog jezika iz baze podataka transformiraju se strukturirani podaci u prirodni jezik.

Budući da je važan redoslijed informacija, tj. riječi u rečenici, koriste se ponavljajuće neuronske mreže (engl. RRN - *retrieval neural network*) koje imaju mogućnost čuvanja informacija iz prethodnih koraka u obuci. Ta funkcionalnost je omogućena skrivenim slojem koji pamti i koristi za predviđanje u komponenti kratkoročne memorije (engl. *short-term memory component*) [20]. Uzmimo za primjer rečenicu „Nebo je plavo“. Ako je ulaz u našu RNN „Nebo je“ očekujemo da izlaz bude „plavo“. Kada skriveni sloj mreže procesuira riječ „Nebo“ sprema u memoriju njezinu kopiju. Kada do sloja dođe sljedeća riječ iz niza, „je“, koristi riječ „Nebo“ prethodno spremljenu i razumije cijeli ulazni kontekst kao „Nebo je“. Nakon toga pretpostavlja riječ „plavo“ kao izlaz.

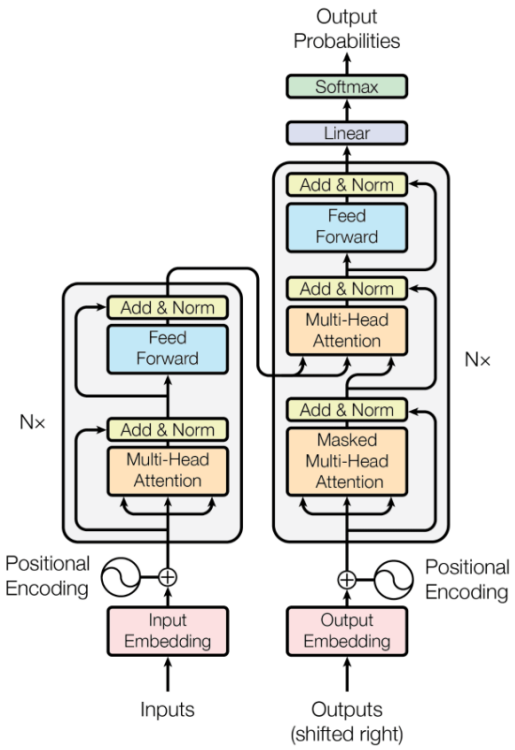
Na slici 3.4. lijevo je prikazana struktura ponavljajuće neuronske mreže (RNN) i unaprijedne neuronske mreže (engl. *feedforward neural network*). S desne strane je prikazano kako RNN dijele isti parametar težine unutar svakog sloja mreže.



Slika 3.4. *Razlika ponavljajuće neuronske mreže i unaprijedne neuronske mreže* [21]

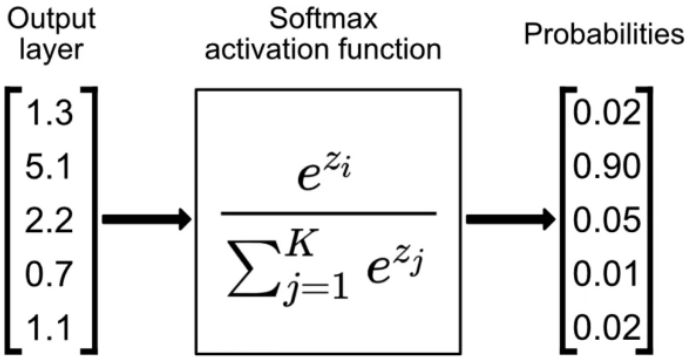
3.4.1. Transformator modeli

Transformator (engl. *transformer*) je model koji se prvi puta spominje 2017. godine u radu naziva „*Attention is all you need*“ Googlovih znastvenika [22].Arhitektura transformatora prikazana je slikom 3.5.



Slika 3.5. *Arhitektura transformatora* [23]

Transformator modeli osmišljeni su kako bi obrađivali ulazne podatke strukturirane kao niz sekvenci, a princip rada modela moguće je opisati u nekoliko koraka. Prvi korak je pretvorba ulazne rečenice u numeričke prikaze koji se zovu ugradnje (engl. *embeddings*), tj. svakoj riječi se dodjeljuje numerička vrijednost koja joj daje semantičko značenje. Sljedeći korak se naziva pozicijsko kodiranje (engl. *positional encoding*). Pozicijskim kodiranjem svakom se tokenu dodjeljuje vrijednost ili vektor koji označava informaciju o položaju tokena u rečenici. Inovacija koju su transformatori uveli u području dubokog učenja je mehanizam samopažnje (engl. *self-attention*). Ovim mehanizmom se izračunavaju odnosi između tokena [22]. Softmax funkcija najčešće je korištena aktivacijska funkcija kojom se izračunavaju težine, a formula je prikazana na slici 3.6. Kako bi se ubrzao i stabilizirao trening, model sadrži slojeve normalizacije i rezidualne veze. Izlaz iz sloja samopažnje proslijeđen je slojevima unaprijednih neuronskih mreža koje primjenjujući nelinearne transformacije na tokenima, omogućuju modelu da nauči složene obrasce i odnose između podataka. Transformatori se sastoje od više slojeva naslaganih jedan na drugi kako bi se poboljšali prikazi svakog tokena. Na kraju se dodaje modul dekodera kako bi se generirala izlazna sekvenca [24].



Slika 3.6. *Formula softmax aktivacijske funkcije* [25]

3.4.2. GPT-3.5-turbo

GPT-3.5-turbo najbolji je model iz serije GPT-3.5 modela koje je razvila tvrtka OpenAI 2022. godine. [26]. Smatra se dovoljno dobrim za većinu potreba. Zbog niske cijene i brzog generiranja savjeta još uvijek je u upotrebi iako je razvijena novija verzija GPT-4 modela. Na slici 3.7. prikazana je tablica s cijenama korištenja modela u odnosu na broj tokena. Osim toga, navedeno je i da je model obučen na skupu podataka ažuriranim do rujna 2021. godine.

Model	Input	Output	Max tokens	Training data
gpt-3.5-turbo	$0.0015 / 1K tokens	$0.002 / 1K tokens	4,096 tokens	Up to Sep 2021
gpt-3.5-turbo-16k	$0.003 / 1K tokens	$0.004 / 1K tokens	16,384 tokens	Up to Sep 2021

Slika 3.7. *Cijene korištenja GPT-3.5-turbo modela* [27]

GPT-3.5 serija modela temelji se na GPT-3 modelu nad kojim se tehnikom finog podešavanja uspjelo smanjiti broj parametara za 1,3 milijarde. Tehnika finog podešavanje koja se koristila pri izgradnji ovog modela je podržano učenje s ljudskom povratnom informacijom (engl. RLHF – *reinforcement learning with human feedback*). U ovakvom podržanom učenju čovjek daje povratnu informaciju algoritmu strojnog učenja koji se koristi za prilagodbu ponašanja modela.

Takvim se pristupom rješava problem ograničenosti tehnika nadziranog i nenadziranog učenja gdje se modeli obučavaju samo na označenim ili neoznačenim podacima [28].

3.4.3. Halucinacije

Kada je riječ o generiranju odgovora velikih jezičnih modela, ponekad se javlja i fenomen halucinacije. Ovaj pojama se odnosi na pružanje činjenično netočnih ili izmišljenih odgovora [29]. Budući da su modeli statički trenirani na podacima koji mogu biti zastarjeli, nemaju mogućnost pristupa svim podacima u svakom trenutku. Također, dajući modelu dvosmislene upite, otežavamo njegovo razumijevanje i generiranje relevantnog odgovora.

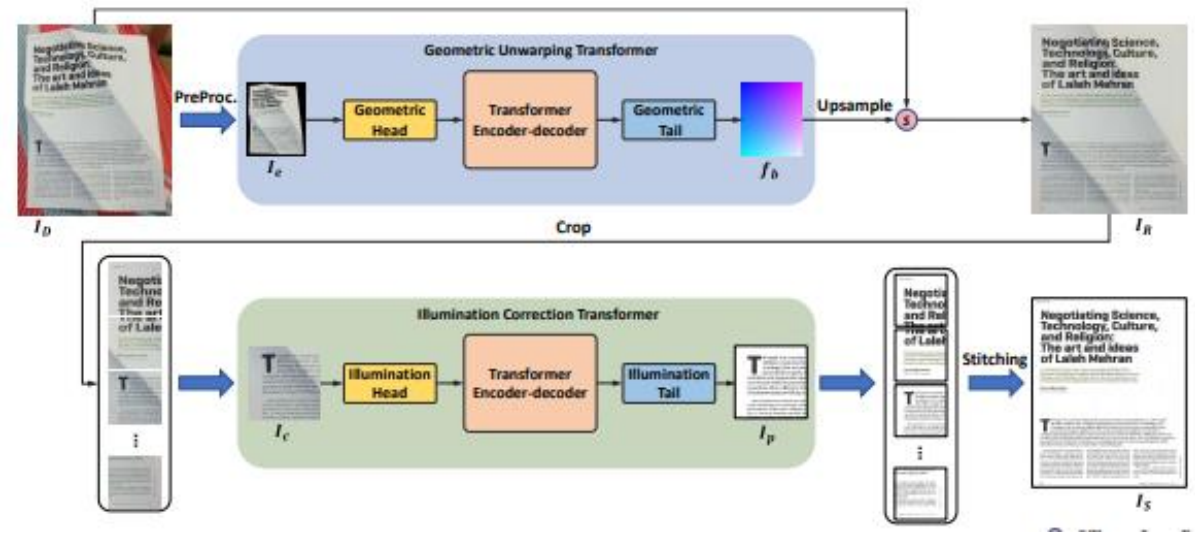
3.5. Optičko prepoznavanje znakova

Optičko prepoznavanje znakova (engl. OCR - *optical character recognition*) proces je elektroničke ili mehaničke pretvorbe slike teksta u strojno kodirani tekst [30]. OCR je široko rasprostranjena metoda računalnog vida jer omogućuje brzu i efikasnu digitalizaciju raznih dokumenata. Budući da predajom fotografiranog dokumenta ili, u slučaju ovog diplomskog rada, fotografiranih informacija o prehrambenom proizvodu, računalo će taj dokument spremiti kao slikovnu datoteku. Kako bi se omogućilo manipuliranje tekstom koji se nalazi na slikovnoj datoteci, preporučeno je korištenje nekih od metoda i modela OCR-a.

OCR tehnologija izdvaja slova na slici, slaže ih u riječi, a potom riječi slaže u rečenice omogućujući pristup i uređivanje izvornog sadržaja [31].

3.5.1. DocTr model

DocTr (skraćeno od *Document Image Transformer*) model je temeljen na transformatorima koji rješava zahtjeve iskrivljene geometrije dokumenta i distorzije osvjetljenja, koji čine glavni problem pri prepoznavanju teksta sa slike [24]. Model se sastoji od dva glavna modula: *Geometric Unwarping* transformatora za korekciju geometrije i *Illumination Correction* transformatora za ispravljanje osvjetljenja. Transformatori primjenjuju evoluirajući skup matematičkih tehnika, koji se nazivaju pažnja (engl. *attention*) ili samopažnja (engl. *self-attention*), kako bi otkrili suptilne načine na koje čak i udaljeni elementi podataka u nizu utječu i ovise jedni o drugima [32].



Slika 3.8. *Prikaz glavnih komponenti Doctr modela* [24]

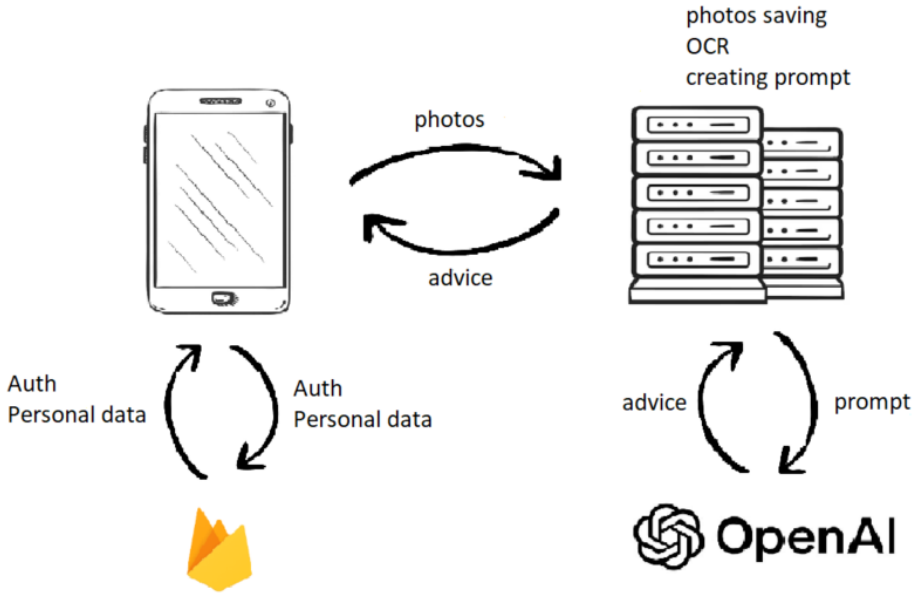
Na slici 3.8. **Error! Reference source not found.**prikazane su glavne komponente od kojih se sastoji *Document Image Transformer*. U transformator zadužen za geometrijsko ispravljanje dokumenta kao ulaz se predaje slika proizvoljne rezolucije. Cilj je predvidjeti pomak po pikselima za odmotavanje. Slika se prvo umanji, zatim se unosi u modul za predprocesiranje kako bi se dobila slika bez pozadine koja se nakon toga ubacuje u transformator geometrijskog rasklapanja za predviđanje pozicijskih polja unatrag. Slika se potom povećava uzorkovanjem na izvornu veličinu. Neiskrivljena slika dokumenta i dalje sadrži nepravilnosti nastale uzorkovanjem i neravnomjernim osvjetljenjem. Iz toga razloga, slika se izrezuje u dijelove s preklapanjem od 12,5% (svaki dio slike ima rezoluciju 128×128). Dijelovi slike se predaju u transformator za korekciju osvjetljenja, a rezultati se spajaju kako bi se dobila kompletna ispravljena slika [24]. Oba transformatora imaju encoder-decoder arhitekturu s točno definiranom početnom i izlaznom mrežom [24].

4. OPIS MOBILNE APLIKACIJE

Ovo poglavlje donosi cjeloviti prikaz arhitekture, funkcionalnosti i tehnologija korištenih za razvoj ove mobilne aplikacije. Aplikacija je osmišljena kako bi korisnicima omogućila izradu vlastitoga profila s informacijama relevantnim za prikupljanje uvida u korisnikove prehrambene navike te im pružila preporuke temeljene na analizi unosa hrane. Ključne funkcionalnosti uključuju prepoznavanje teksta s fotografija prehrambenih proizvoda pomoću OCR tehnologije, analizu nutritivnih vrijednosti te personalizirane preporuke prilagođene individualnim potrebama korisnika, što je omogućeno implementacijom velikog jezičnog modela.

4.1. Arhitektura aplikacije

Aplikacija je izrađena koristeći klijent-poslužitelj arhitekturu, pri čemu je klijentska strana predstavljena mobilnom aplikacijom razvijenom u *React Native* radnom okviru za *JavaScript* programski jezik, dok je poslužiteljski dio implementiran koristeći *Python Flask* radni okvir. Ovakvom arhitekturom omogućena je fleksibilnost u razvoju i održavanju aplikacije, posebno jer klijent i poslužitelj mogu biti održavani i ažurirani neovisno jedan o drugome. Prijava/registracija korisnika i spremanje osobnih podataka omogućeni su korištenjem usluga *Google Firebase-a*. Na poslužiteljskoj strani odvija se komunikacija s GPT-3.5-turbo modelom putem *OpenAI API-ja* slanjem zahtjeva (engl. *request*) i dobivanjem odgovora (engl. *respond*) koji se prosljeđuje korisnikovoj mobilnoj aplikaciji. Prethodno opisanu arhitekturu prikazuje slika 4.1.



Slika 4.1. *Skica arhitekture aplikacije*

Korisničko sučelje (engl. UI - *User Interface*) izrađeno *React Native* elementima nudi korisniku jednostavnu komunikaciju s aplikacijom, što uključuje intuitivnu prijavu i registraciju u sustav, unos i ažuriranje osobnih podataka, fotografiranje proizvoda i dobivanje prehrambenih preporuka. S druge strane, *Flask* poslužitelj obrađuje zahtjeve primljene od klijenta, izvršava logiku aplikacije, i upravlja složenijim funkcijama kao što su obrada podataka i *OpenAI API* pozivi.

Podjela arhitekture između klijenta i poslužitelja također omogućava jednostavniju implementaciju sigurnosnih mjera jer se svi osjetljivi podaci i logika aplikacije obrađuju na poslužiteljskoj strani, dok klijent ostaje samo kao korisničko sučelje. Integracijom *Firebase-a* za autentikaciju korisnika i sigurno pohranjivanje korisničkih podataka dodatno se štiti privatnost korisnika.

4.2. Autentikacija i sigurnost s Firebaseom

Prijava i registracija korisnika predstavljaju ključnu funkcionalnost svake mobilne aplikacije. Time se korisniku omogućuje siguran pristup personaliziranim sadržajima i pohranu njihovih podataka. U ovoj aplikaciji prijava je izvedena pomoću *Firebase Authentication* sustava, koji omogućava jednostavnu i sigurnu prijavu korisnika korištenjem e-mail adrese i lozinke.

Slika 4.2. prikazuje programski kod *handleAuthentication* funkcije koja se predaje gumbu koji služi za prijavu/registraciju korisnika. Funkcija *setPersistence* postavlja način na koji će sesija korisnika biti pohranjena na uređaju. Koristi se lokalna pohrana *ReactNativeAsyncStorage* kako bi se omogućilo da se sesija korisnika zadrži i nakon zatvaranja ili ponovno pokretanja aplikacije. Provjerom varijable *isLogin* označava se pokušaj prijave korisnika. Prijava korisnika odvija se pozivom *signInWithEmailAndPassword* funkcije čiji su parametri varijabla koja sadrži instancu *Firebase Authentication* servisa, e-mail adresa i lozinka korisnika. Ako su predani podaci točni, funkcija vraća *userCredential* objekt s podacima korisnika koji se predaju objektu *user*. Aplikacija potom prikazuje obavijest o uspješnoj prijavi i preusmjerava korisnika na glavni zaslon aplikacije s proslijeđenim korisničkim ID-jem kako bi se omogućilo pretraživanje njegovih osobnih podataka. Navigacija se omogućuje pomoću *navigation.replace* metode s proslijeđenim *userId* parametrom kako bi ostale funkcionalnosti prepoznale korisnika. Ako je vrijednost varijable *isLogin False*, označava se pokušaj registracije korisnika. Za registraciju je zaslužna funkcija *createUserWithEmailAndPassword*, koja stvara novi korisnički račun s e-mailom i lozinkom. Nakon što je račun uspješno kreiran, dodatni podaci o korisniku, kao što su korisničko ime (*username*) i datum kreiranja (*createdAt*), spremaju se u *Firestore*. Registracija uključuje spremanje dodatnih podataka u *Firestore* bazu podataka korištenjem *setDoc* funkcije. Aplikacija

sprema korisničko ime, e-mail i vrijeme registracije s korisničkim ID-om (*user.uid*) u kolekciju *'users'*. Na taj način osigurava se trajno čuvanje podataka korisnika. Nakon uspješne registracije korisnik se preusmjerava na glavni zaslon aplikacije. Ukoliko dođe do greške korisniku je prikazana poruka obavijesti.

```
const handleAuthentication = async () => {
  try {

    await setPersistence(auth, getReactNativePersistence(ReactNativeAsyncStorage));
    if (isLogin) {
      const userCredential = await signInWithEmailAndPassword(auth, email, password);
      const user = userCredential.user;
      Alert.alert('Logged in successfully!');
      navigation.replace('HomeTabs', { screen: 'MealAdviceScreen', params: { userId: user.uid } });
    } else {
      const userCredential = await createUserWithEmailAndPassword(auth, email, password);
      const user = userCredential.user;

      await setDoc(doc(db, 'users', user.uid), {
        username,
        email: user.email,
        createdAt: new Date(),
      });

      Alert.alert('Account created successfully!');
      navigation.replace('HomeTabs', { screen: 'MealAdviceScreen', params: { userId: user.uid } });
    }
  } catch (error) {
    Alert.alert('Error', error.message);
  }
};
```

Slika 4.2 *Funkcija za prijavu korisnika*

Na slici 4.3. prikazan je kod funkcije *return* koji koristi JSX sintaksu za definiranje strukture korisničkog sučelja koje će se prikazati na zaslonu mobilne aplikacije. Kao glavni kontejner odabran je <*ScrollView*> generički spremnik koji omogućuje okomito i vodoravno pomicanje [33]. Osim <*Text*> elementa koji prikazuje naziv aplikacije, na zaslonu se prikazuje i središnji kontejner s poljima za unos korisničkog imena, e-mail adrese i lozinke. Naziv forme je dinamički prikazan, tj. ovisi o stanju varijable *isLogin*. Kada je varijabla *isLogin True*, polje za unos imena se ne prikazuje. Interaktivni element <*Pressable*> reagira na pritiske korisnika te poziva prethodno opisanu funkciju *handleAuthentication*. Na dnu forme se još nalazi i tekstualni link čijim se pritiskom mijenja vrijednost *isLogin* varijable koja označava prijavu ili registraciju.

Slika 4.3. *Definiranje strukture korisničkog sučelja*

4.3. Upravljanje osobnim podacima korisnika

Aplikacija omogućava korisnicima pregled i ažuriranje njihovih osobnih podataka, koji su pohranjeni u *Firestore* bazi podataka. Ova funkcionalnost implementirana je putem dva ključna dijela: prikaz osobnih podataka na *HomeScreen* zaslonu i mogućnost njihovog ažuriranja na *UpdatePersonalData* zaslonu.

Slika 4.3. **Error! Reference source not found.**prikazuje kod funkcije *fetchPersonalData* koja služi za dohvaćanje osobnih podataka korisnika iz *Firestore* baze u *HomeScreen*. Prvo se provjerava postoji li prijavljeni korisnik uporabom *auth* objekta koji vraća podatke o korisniku. Ako je korisnik prijavljen, *doc* funkcijom preuzetom iz *firebase/firestore* biblioteke kreira se referenca na određeni dokument u bazi. Parametri koje ona prima su *Firestore* instanca, naziv kolekcije u bazi i jedinstveni identifikator trenutnog korisnika. Referenca na dokument trenutnog korisnika je spremljena u konstantu *docRef*. Na temelju reference predane kao parametar funkciji *getDoc* dohvaćaju se podaci iz dokumenta i spremaju u *docSnap* konstantu. Ako dokument postoji, pozivom funkcije *data* na objektu *docSnap* podaci se spremaju u obliku objekata. *setPersonalData* funkcijom ažurira se vrijednost komponente *personalData* s dohvaćenim podacima. Funkcijom *setUsername* postavlja se stanje konstante *userName*. Ukoliko dokument ne postoji ispisuje se prigodna poruka u konzoli.

```
const fetchPersonalData = async () => {
  const user = auth.currentUser;
  if (user) {
    const docRef = doc(db, 'users', user.uid);
    const docSnap = await getDoc(docRef);
    if (docSnap.exists()) {
      const data = docSnap.data();
      setPersonalData(data);
      setUsername(data.username);
    } else {
      console.log("No such document!");
    }
  }
};
```

Slika 4.3. *Funkcija za dohvaćanje osobnih podataka*

Ako su osobni podaci uspješno dohvaćeni, prikazuju se u obliku tablice koja sadrži atribute kao što su dob, spol, razina aktivnosti, visina, težina i ciljana težina. Na slici 4.4. **Error! Reference source not found.**prikazan je kod za stvaranje prikaza korisničkog sučelja za pregled njegovih podataka i odjavu iz aplikacije. Unutar glavnog kontejnera ugniježđeni su <*Text*> elementi za naslov i podnaslov. Nakon provjere jesu li osobni podaci uspješno spremljeni u objekt *personalData*, prikazuje se tablica s korisnikovim podacima. Svaki redak tablice prikazan je unutar <*View*> elementa u kojem se nalaze dva tekstualna prikaza. Prvi označava vrstu podatka, a drugi vrijednost podatka. U slučaju da podaci nisu pronađeni, korisniku se prikazuje poruka "*No personal data found*." Komponenti <*Pressable*> korištenoj za implementaciju gumba, dodana je *handleSignOut* funkcija *onPress* atributu kako bi se korisnik klikom na gumb mogao odjaviti.

Slika 4.4. *Prikaz stvaranja tablice*

UpdatePersonalData ekran omogućava korisnicima unos novih ili promjenu postojećih osobnih podataka. Aplikacija omogućava korisnicima unos osnovnih atributa, a nakon unosa ti se podaci ažuriraju u *Firestore*-u. Programski kod na slici 4.5. prikazuje funkciju *handleUpdate* koja je zadužena za ažuriranje osobnih podataka. Nakon dohvaćanja trenutno prijavljenog korisnika pomoću *auth.currentUser*, provjerava se sadrži li *user* podatke o korisniku. Prije ažuriranja provjerava se jesu li popunjeni obavezni podaci (spol i razina aktivnosti). Ukoliko nisu, prikazuje

se poruka upozorenja korisniku kako bi ispunio obavezna polja. Unutar *try* bloka funkcija *setDoc* postavlja ili ažurira dokument u određenoj kolekciji. Nakon uspješnog ažuriranja prikazuje se poruka o uspjehu. Funkcijom *navigation.goBack* korisnik se vraća na prethodni zaslon. Ukoliko se dogodi greška ispisuje se poruka s detaljima o pogrešci.

```
const handleUpdate = async () => {
  const user = auth.currentUser;
  if (user) {
    if (!gender || !activityLevel) {
      Alert.alert('Error', 'Please fill all the required fields (Gender, Activity Level)');
      return;
    }

    try {
      await setDoc(doc(db, 'users', user.uid), {
        age,
        gender,
        activity_level: activityLevel,
        height,
        weight,
        target_weight: targetWeight,
      });
      alert('Personal data updated successfully');
      navigation.goBack();
    } catch (error) {
      alert('Error updating personal data: ' + error.message);
    }
  }
};
```

Slika 4.5. *Funkcija za ažuriranje osobnih podataka korisnika*

Slika 4.6. prikazuje ulazne komponente <*TextInput*> i <*RNPickerSelect*> koje korisničkom sučelju nude unos podataka. <*TextInput*> osnovna je komponenta koja omogućuje unos teksta u aplikaciju putem tipkovnice. Za čitanje korisničkog unosa koristi se *onChangeText* događaj [34] koji u ovom primjeru ažurira vrijednost upisanih godina. <*RNPickerSelect*> je *React Native* komponenta odabira koja oponaša izvorno *select* sučelje za Android i iOS [35]. *onValueChange* osnovno je svojstvo kojim se ažurira stanje komponente na odabranu vrijednost. Vrijednosti koje su korisniku ponuđene su nabrojane u listi objekata *items* s oznakama (engl. *label*) i vrijednosti (engl. *value*).

```
<TextInput
  style={[pickerSelectStyles.input, styles.textInput]}
  placeholder="Age"
  value={age}
  onChangeText={setAge}
  keyboardType="numeric"
/>



<RNPickerSelect
  onValueChange={(value) => setGender(value)}
  items={[
    { label: 'Male', value: 'male' },
    { label: 'Female', value: 'female' },
  ]}
  placeholder={{ label: 'Select Gender', value: '' }}
  style={pickerSelectStyles}
/>
```

Slika 4.6. *Komponente za ispunjavanje i odabir osobnih podataka*

Pritiskom na gumb "*Update Data*", pokreće se funkcija *handleUpdate* za ažuriranje podataka u bazi. <*Pressable*> je temeljna komponenta *React Nativea* koja može otkriti različite faze interakcije pritiska na bilo koju od ugniježđenih elemenata [36]. Navedeno je prikazano kodom na slici 4.7.

```
<Pressable
  onPressIn={handlePressIn}
  onPressOut={handlePressOut}
  onPress={handleUpdate}
  style={({ pressed }) => [
    styles.updateButton,
    pressed && styles.buttonPressed,
  ]}>
  <Animated.View style={{ transform: [{ scale: scaleValue }] }}>
    <Text style={styles.updateButtonText}>Update Data</Text>
  </Animated.View>
</Pressable>
```

Slika 4.7. *Gumb za ažuriranje podataka*

4.4. Prehrambeni savjeti o proizvodu

Aplikacija koristi *ProductAdviceScreen* zaslon za prikupljanje slika jednog proizvoda (sastojaka i tablice nutritivnih vrijednosti) putem kamere na uređaju korisnika, zatim prenosi te slike na *backend* stranu, gdje se obavlja OCR analiza za prepoznavanje teksta. Na temelju analiziranih informacija, *backend* generira savjet putem *OpenAI API-ja* i vraća ga klijentu.

Slika 4.8. prikazuje programski kod funkcije *takePicture* koja koristi *Camera* komponentu za upravljanje kamerom i sprema fotografije na lokalnu pohranu uređaja. Provjerava se postoji li

referenca na kameru i je li broj fotografiranih slika manji od 2. *takePictureAsync* funkcija omogućava snimanje fotografije te vraćeni objekt s informacijama o fotografiji sprema se u varijablu *photo*. Zatim se generira putanja za spremanje slike koristeći *FileSystem* direktorij za predmemoriju. Fotografija se premješta funkcijom *moveAsync* s njene privremene lokacije (*photo.uri*) na novu lokaciju u predmemoriji. Time se osigurava da fotografija ostane dostupna i nakon zatvaranja kamere. Putanja nove fotografije dodaje se u niz svih fotografija, te se povećava broj snimljenih fotografija kako bi se omogućila kontrola nad maksimalnim brojem fotografija.

```
const takePicture = async () => {
  if (camera && photoCount < 2) {
    let photo = await camera.takePictureAsync();
    const cacheDir = FileSystem.cacheDirectory + `photo_${photoCount + 1}.jpg`;

    await FileSystem.moveAsync({
      from: photo.uri,
      to: cacheDir,
    });

    setPhotos([...photos, cacheDir]);
    setPhotoCount(photoCount + 1);
  }
};
```

Slika 4.8. *Funkcija za snimanje fotografija*

Nakon što su obje fotografije (sastojci i nutritivne vrijednosti) snimljene, korisnik može pritisnuti gumb *Get Advice* kako bi pokrenuo prijenos slika na *backend*. Programski kod funkcije za slanje podataka prikazan je na slici 4.9. Funkcija ima ključnu ulogu u aplikaciji jer prenosi slike na *backend* poslužitelj, šalje osobne podatke korisnika za generiranje preporuke, prima i prikazuje savjet te briše lokalno spremljene fotografije nakon uspješnog postupka. Vrše se provjere broja fotografija i dostupnost korisnikovih podataka. Postavljanjem *setIsProcessing* korisnik se obavještava da je zahtjev u procesu obrade. Kreira se novi objekt tipa *FormData* koji će sadržavati slike i podatke za prijenos. *forEach* petljom prolazi se kroz sve slike u nizu te se svaka dodaje s nazivom photo_1.jpg i photo_2.jpg. Dodaju se i osobni podaci u JSON formatu. Aplikacija koristi *axios* za slanje POST zahtjeva na *backend* krajnju točku */product-advice/* s fotografijama i osobnim podacima. U zaglavlju (engl. *headers*) naznačuje se format podataka koji se šalju. Odgovor poslužitelja sprema se u *advice* konstantu, a funkcijom *setExtractedText* ažurira se stanje aplikacije za prikaz primljenog savjeta. Nakon primljenog savjeta brišu se sve lokalne fotografije, resetira se lista i broj fotografija, isključuje indikator obrade i aktivira prikaz savjeta.

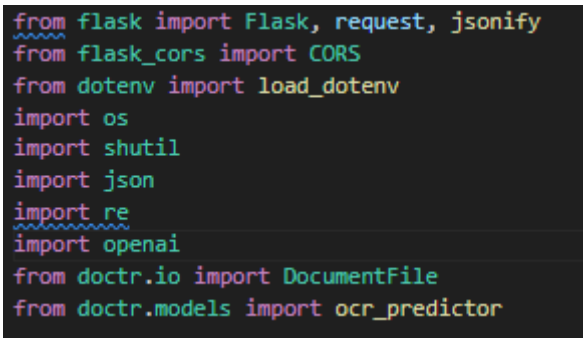
```
const uploadPhotos = async () => {
  if (photos.length === 2 && personalData) {
    setIsProcessing(true);
    const formData = new FormData();

    photos.forEach((photo, index) => {
      formData.append(`file${index + 1}`, {
        uri: photo,
        type: 'image/jpeg',
        name: `photo_${index + 1}.jpg`,
      });
    });

    formData.append('personal_data', JSON.stringify(personalData));

    try {
      const response = await axios.post('http://192.168.1.7:5000/product-advice/', formData, {
        headers: { 'Content-Type': 'multipart/form-data' },
      });

      const { advice } = response.data;
      setExtractedText([{ sender: "App", text: `${advice}`, type: "Advice" }]);

      photos.forEach(async (photo) => {
        await FileSystem.deleteAsync(photo, { idempotent: true });
      });

      setPhotos([]);
      setPhotoCount(0);
      setIsProcessing(false);
      setShowAdvice(true);
    } catch (error) {
      console.error('Error uploading photos:', error);
    }
  } else {
    console.log('Please take 2 photos and ensure personal data is loaded');
  }
};
```

Slika 4.9. *Funkcija za slanje podataka*

Na slici 4.10. prikazan je kod kojim se definiraju dva glavna prikaza korisničkog sučelja zaslona za generiranje savjeta o proizvodu. Prvi služi korištenju kamere i snimanju fotografija, a drugi za prikaz preporuka korisniku. Prikaz kamere putem komponente <*Camera*> preuzete iz biblioteke *expo-camera* odvija se ukoliko je varijabla *showAdvice* postavljena na neistinu. Unutar prikaza kamere nalazi se tekstualni vodič prikazan pomoću <*Text*> komponente čija se vrijednost prilagođava ovisno o stanju varijable *photoCount*. Gumb za snimanje fotografija implementiran je pomoću komponente <*Pressable*> kojoj je predana prethodno opisana funkcija. Ikona kamere unutar gumba dolazi iz *react-native-vector-icons* biblioteke. Nakon snimanja dvije fotografije aktivira se gumb *Get Advice* za prijenos slika na poslužitelj. Klikom na gumb poziva se funkcija *uploadPhotos*. Ukoliko je vrijednost *showAdvice* istinita, korisniku se prikazuju rezultati generiranog savjeta. Svaka preporuka prikazuje se pomoću komponente <*View*> iteracijom kroz polje *extractedText*. Na dnu se nalazi gumb za povratak kojim se resetira stanje aplikacije i vraća na početni prikaz s kamerom.

Slika 4.10. *Korisničko sučelje ProductAdviceScreen-a*

Postavljanje i inicijalizacija *backend* okruženja započinje uključivanjem potrebnih biblioteka i modula. Biblioteka *Flask* se koristi za postavljanje web poslužitelja, a *Flask-CORS* za omogućavanje zahtjeva prema drugim izvorima (engl. *cross-origin*). Uz to koriste se biblioteke *dotenv* za učitavanje varijabli kruženja, *os* za rad s datotečnim sustavom, *shutil* za upravljanje direktorijima, *json* za rukovanje JSON podacima i *re* za rad s regularnim izrazima. Biblioteka *openai* omogućava interakciju s *OpenAI API-jem*, a iz *doctr* biblioteke se učitava *ocr_predictor* model za izvedbu OCR metode. Učitavanje biblioteka prikazano je slikom 4.11.



Slika 4.11. *Učitavanje biblioteka*

Sljedeći korak prikazan programskim kodom na slici 4.12. opisuje inicijalizaciju modela za OCR i učitavanje OpenAI API ključa iz *.env* datoteci. Potrebno je postaviti *Flask* aplikaciju, omogućiti CORS i definirati tajni ključ sesije. Također definiraju se direktoriji za učitavanje slika i poddirektoriji za različite vrste sadržaja (*meal* i *product*).

```
load_dotenv("openai_api_key.env")
openai_api_key = os.getenv('OPENAI_API_KEY')
openai.api_key = openai_api_key
ocr_model = ocr_predictor(pretrained=True)

app = Flask(__name__)
CORS(app)
load_dotenv("flask_key.env")
app.secret_key = os.getenv('FLASK_SECRET_KEY')

UPLOAD_FOLDER = './uploads'
MEAL_FOLDER = os.path.join(UPLOAD_FOLDER, 'meal')
PRODUCT_FOLDER = os.path.join(UPLOAD_FOLDER, 'product')

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

for folder in [UPLOAD_FOLDER, MEAL_FOLDER, PRODUCT_FOLDER]:
    if not os.path.exists(folder):
        os.makedirs(folder)
```

Slika 4.12. *Postavljanje modela i postavki aplikacije*

Glavna ruta */product-advice/* prima HTTP POST zahtjev s fotografijama proizvoda i osobnim podacima korisnika. Programski kod ove rute prikazan je na slici 4.13. Funkcija najprije briše sve prethodne datoteke iz direktorija za proizvode. Zatim se provjerava broj primljenih slika i ukoliko nemaju točno dvije slike vraća se HTTP odgovor s kodom 400 i porukom o grešci. Ako su poslane točno 2 slike, učitavaju se osobni podaci, a slike spremaju u odgovarajuće datoteke unutar direktorija *PRODUCT_FOLDER*. U *ingredients_doc* i *nutrition_doc* varijable učitavaju se dokumenti iz slika, a primjenom funkcije *ocr_model* primjenjuje se OCR nad predanim joj dokumentima. Prepoznati tekst prosljeđuje se funkcijama *extract_ingredients* i

extract_nutrition_values koje služe za prepoznavanje potrebnih dijelova teksta, a čiji je programski kod objašnjen u nastavku. Izdvojeni sadržaj sprema se u JSON datoteke. Pomoću funkcije *create_product_advice_prompt* kreira se upit, a njegovim prosljeđivanjem *generate_advice* funkciji generira se savjet u obliku teksta. Rezultat se vraća korisniku.

```python
@app.route('/product-advice/', methods=['POST'])
def process_images():
    clear_directory(PRODUCT_FOLDER)

    files = request.files
    if len(files) != 2:
        return jsonify({"error": "Please upload exactly two images"}), 400

    personal_data_json = request.form.get('personal_data')
    if personal_data_json:
        personal_data = json.loads(personal_data_json)
    else:
        return jsonify({"error": "No personal data received"}), 400

    ingredients_img_path = os.path.join(PRODUCT_FOLDER, files['file1'].filename)
    nutrition_img_path = os.path.join(PRODUCT_FOLDER, files['file2'].filename)

    files['file1'].save(ingredients_img_path)
    files['file2'].save(nutrition_img_path)

    ingredients_doc = DocumentFile.from_images(ingredients_img_path)
    nutrition_doc = DocumentFile.from_images(nutrition_img_path)

    ingredients_page = ocr_model(ingredients_doc)
    nutrition_page = ocr_model(nutrition_doc)

    extracted_ingredients = extract_ingredients(ingredients_page.pages[0].export())
    extracted_nutrition = extract_nutrition_values(nutrition_page.pages[0].export())

    save_json({"ingredients": extracted_ingredients}, os.path.join(PRODUCT_FOLDER, "ingredients.json"))
    save_json({"nutrition_values": extracted_nutrition}, os.path.join(PRODUCT_FOLDER, "nutrition_values.json"))

    prompt = create_product_advice_prompt(personal_data, extracted_nutrition, extracted_ingredients)
    advice = generate_advice(prompt)

    return jsonify({"advice": advice}), 200
```

Slika 4.13. *Ruta /product-advice/*

Funkcija *extract_ingredients* služi za izdvajanje popisa sastojaka iz podataka koje generira OCR model. Varijabla *found_sastoj* koristi se za praćenje je li riječ „sastoj“ pronađena u tekstu čime se označuje početak popisa sastojaka. OCR podatkovna struktura je raspoređena u blokove koji sadrže linije u kojima su prepoznati znakovi složeni u riječi. S toga je potrebno petljama proći kroz blokove, linije i riječi kako bi se pronašla vrijednost „sastoj“ i označio početak popisa sastojaka. Ukoliko se pronađe '.' označava se završetak popisa sastojaka. Ako se pronađe ',' riječ se pridružuje trenutnom sastojku, a ako riječ nije povezana zarezom, jednostavno se dodaje u string. Time se osigurava ukoliko neki sastojak sadrži u nazivu više riječi. Funkcija vraća listu svih sastojaka, a programski kod je prikazan slikom 4.14.

```python
def extract_ingredients(page_dict):
    found_sastoj = False
    extracted_ingredients = []
    current_ingredient = ""

    for block in page_dict['blocks']:
        for line in block['lines']:
            for word in line['words']:
                word_value = word['value']
                if re.search(r'[Ss]astoj', word_value):
                    found_sastoj = True
                    continue

                if found_sastoj:
                    if '.' in word_value:
                        if current_ingredient:
                            extracted_ingredients.append(current_ingredient.strip())
                        found_sastoj = False
                        break

                    if ',' in word_value:
                        current_ingredient += f" {word_value}"
                    else:
                        current_ingredient += f" {word_value}"

                    if word_value.endswith(','):
                        extracted_ingredients.append(current_ingredient.strip())
                        current_ingredient = ""

    return extracted_ingredients
```

Slika 4.14. *Funkcija za izdvajanje sastojaka*

Na slici 4.15. prikazan je programski kod funkcije za izdvajanje nutritivnih vrijednosti. Funkcija vraća rječnik *nutrition_data* koji sadrži sve izdvojene nutritivne vrijednosti. *current_label* varijabla služi za privremeno pohranjivanje trenutnih oznaka dok se podudaranje s mapiranim ključnim riječima ne ostvari. Budući da je zbog osvjetljenja i ostalih čimbenika nemoguće točno prepoznati sve riječi sa slike, rječnik *nutrition_labels_mapping* mapira ključne riječi na kategorije nutritivnih vrijednosti koristeći regularne izraze. Prolaskom petlje kroz blokove, linije i riječi pronalaze se spremljene vrijednosti. Kako bi se održala dosljednost brojeva, ',' u vrijednostima se mijenja u '.'. Provjerom odgovara li neka riječ bilo kojem uzorku iz rječnika *nutrition_labels_mapping*, varijabla *current_label* se postavlja na odgovarajuću kategoriju. Uvjetom *re.search(r'\d+[.,]?\d*\s*[g|mg|kcal|kJ]', word_value)* provjerava se sadrži li riječ brojčanu vrijednost i mjernu jedinicu. Ukoliko je uvjet zadovoljen, trenutna vrijednost se dodaje pod odgovarajući ključ u rječnik *nutrition_data*.

```python
def extract_nutrition_values(page_dict):
    nutrition_data = {}
    current_label = None

    nutrition_labels_mapping = {
        'Energy': [r'Energ'],
        'Fats': [r'Mas', r'Fa'],
        'Carbs': [r'Ugl', r'Carboh'],
        'Protein': [r'Bjel', r'Prot'],
        'Fiber': [r'Vla', r'Fib'],
        'Sodium': [r'So'],
        'Sugar': [r'Se', r'Su'],
        'Saturated': [r'zasic', r'satur'],
        'Unsaturated': [r'nezas', r'unsat'],
    }

    for block in page_dict['blocks']:
        for line in block['lines']:
            for word in line['words']:
                word_value = word['value'].strip().replace(',', '.')

                for label, patterns in nutrition_labels_mapping.items():
                    if any(re.search(pattern, word_value, re.IGNORECASE) for pattern in patterns):
                        current_label = label
                        break

                if current_label and re.search(r'\d+[.,]?\d*\s*[g|mg|kcal|kJ]', word_value):
                    nutrition_data[current_label] = word_value
                    current_label = None

    return nutrition_data
```

Slika 4.15. *Funkcija za izdvajanje nutritivnih vrijednosti*

Funkcija za generiranje personaliziranih tekstualnih upita prikazana je programskim kodom na slici 4.16. Funkcija kao ulazne parametre prima rječnik s osobnim podacima korisnika, rječnik hranjivih vrijednosti i listu sastojaka. Tekstualni upit vraća se kao string, a sadrži personalizirane informacije o korisniku i proizvodu, formatirane tako da su lako razumljive modelu. Uključuje smjernice da preporuka bude kratka i da sadrži preporuku količine proizvoda u obroku i koje dodatne sastojke je moguće uključiti.

```python
def create_product_advice_prompt(personal_data, nutritional_values, ingredients):

    prompt = f"""
    The user has uploaded the nutritional information and ingredients list of
    a single product they wish to include in a meal.

    **User Profile:**
    - Age: {personal_data.get('age', 'N/A')} years old
    - Gender: {personal_data.get('gender', 'N/A')}
    - Activity level: {personal_data.get('activity_level', 'N/A')}/5
    - Height: {personal_data.get('height', 'N/A')} cm tall
    - Wight: {personal_data.get('weight', 'N/A')} kg
    - Target Weight: {personal_data.get('target_weight', 'N/A')} kg.

    Here are the nutritional values for product:
    {', '.join(f"{key}: {value}" for key, value in nutritional_values.items())}

    Ingredients of this product written in Croatian language:
    {', '.join(ingredients)}

    Provide a short, personalized dietary recommendation (no more than 5 sentences)
    based on the product's nutritional content and the user's dietary goals.
    Include a suggestion for the amount of this product to consume in a single meal and
    recommend any complementary ingredients that would create a balanced meal with this product.
    """
    return prompt
```

Slika 4.16. *Funkcija za kreiranje naputka savjeta o proizvodu*

Kreirani upit predaje se funkciji *generate_advice* koja služi za dohvaćanje generiranog savjeta. Pozivom metode *openai.ChatCompletion.create* šalje se upit prema OpenAI API-ju gdje se koristi GPT-3.5-turbo model za generiranje odgovora. Modelu je potrebno postaviti kontekst pa se u ovom slučaju model predstavlja kao nutricionistički stručnjak, a upit se postavlja kao ulazna poruka modelu. Duljina generiranog odgovora ograničena je na 250 tokena kako bi odgovor bio relevantan i sažet. Postavljanjem vrijednosti parametra temperature na 0.4 osigurana je konzistentnost odgovora s manjom vjerojatnosti halucinacije. Funkcija vraća prvi generirani odgovor iz odgovora API-ja. Opisana funkcija prikazana je slikom 4.17.

```python
def generate_advice(prompt):

    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a nutrition expert."},
            {"role": "user", "content": prompt}
        ],
        max_tokens=250,
        temperature=0.4
    )
    return response.choices[0].message['content']
```

Slika 4.17. *Funkcija za generiranje savjeta*

4.5. Prehrambeni savjeti o obroku

Glavna funkcionalnost aplikacije je dobivanje preporuka o obroku na osnovu analize svih prehrambenih proizvoda koje korisnik planira konzumirati u jednome obroku.

Prvo je potrebno odrediti koliko proizvoda korisnik planira konzumirati u obroku. Odabir je bitan jer analiza zahtijeva fotografije dva tipa za svaki proizvod: jednu za sastojke i drugu za nutritivne vrijednosti. Koristi se komponenta *<RNPickerSelect>*, čija je implementacija prikazana na slici 4.18., a pruža padajući izbornik s brojem proizvoda. Kada korisnik odabere broj proizvoda, taj se broj sprema u varijablu *numProducts* pomoću funkcije *setNumProducts*.

```
<View style={styles.inputContainer}>
  <Text style={styles.label}>Select number of products:</Text>
  <RNPickerSelect
    onValueChange={(value) => setNumProducts(value)}
    items={[...Array(10).keys()].map((i) => ({ label: `${i + 1}`, value: i + 1 }))}
    style={{
      inputIOS: styles.pickerInput,
      inputAndroid: styles.pickerInput,
      iconContainer: styles.pickerIconContainer,
    }}
    placeholder={{ label: "Select...", value: 0 }}
  />
  <Pressable
    onPressIn={() => handlePressIn(startButtonScale)}
    onPressOut={() => handlePressOut(startButtonScale)}
    onPress={startMealCapture}
  >
    <Animated.View style={[styles.startButton, { transform: [{ scale: startButtonScale }] }]}>
      <Text style={styles.startButtonText}>Start Camera</Text>
    </Animated.View>
  </Pressable>
</View>
```

Slika 4.18. *Prikaz komponenti za odabir broja proizvoda*

Nakon što korisnik odabere broj proizvoda i klikne na "*Start Camera*", poziva se funkcija *startMealCapture* prikazana slikom 4.19. Ova funkcija prvo provjerava je li korisnik odabrao broj proizvoda, a zatim postavlja početne varijable potrebne za proces fotografiranja: *currentProduct* se postavlja na 1, a *photoType* se postavlja na *ingredients*, označavajući da će se prva fotografija za svaki proizvod odnositi na sastojke.

```
const startMealCapture = () => {
  if (!numProducts) {
    Alert.alert('Please select the number of products.');
    return;
  }
  setCurrentProduct(1);
  setPhotoType('ingredients');
};
```

Slika 4.19. *Programski kod funkcije startMealCapture*

Kod svakog proizvoda potrebno je snimiti dvije fotografije – jednu za sastojke i drugu za nutritivne vrijednosti. Funkcija *takePicture* koristi kameru uređaja za snimanje fotografije, nakon čega premješta snimljenu sliku u privremeni direktorij na uređaju. Ovisno o trenutnom tipu fotografije generira se naziv datoteke. Datoteka se sprema u privremeni direktorij aplikacije pomoću *FileSystem.cacheDirectory*. Nakon toga fotografija se premješta u lokalnu pohranu. Ažurira se stanje aplikacije dodavanjem nove fotografije u niz *photos*. Ako je trenutna fotografija tipa „*ingredients*“, sljedeća će biti „*nutrition*“. Kada su obje fotografije za jedan proizvod snimljene, prelazi se na sljedeći proizvod i ponovno se počinje sa tipom „*ingredients*“. Programski kod funkcije *takePicture* prikazan je na slici 4.20.

```
const takePicture = async () => {
  if (camera && photos.length < numProducts * 2) {
    let photo = await camera.takePictureAsync();
    const photoTypeName = photoType === 'ingredients' ? `ingredients_image_${currentProduct - 1}` : `nutrition_image_${currentProduct - 1}`;
    const cacheDir = `${FileSystem.cacheDirectory}${photoTypeName}.jpg`;

    await FileSystem.moveAsync({ from: photo.uri, to: cacheDir });

    setPhotos([...photos, { type: photoType, uri: cacheDir }]);

    if (photoType === 'ingredients') {
      setPhotoType('nutrition');
    } else {
      if (currentProduct < numProducts) {
        setCurrentProduct(currentProduct + 1);
        setPhotoType('ingredients');
      }
    }
  }
};
```

Slika 4.20. *Programski kod takePicture funkcije*

Nakon što korisnik snimi sve potrebne fotografije, funkcija *uploadPhotos* priprema i šalje podatke na *backend*. Njezin kod je prikazan na slici 4.21., a prikazuje njezinu provjeru je li korisnik snimio sve potrebne slike i jesu li učitani osobni podaci korisnika. Svaka slika se zatim dodaje u *FormData* objekt pomoću *photoKey* ključa koji označava o kojoj se vrsti fotografije radi. Osobni podaci korisnika se pretvaraju u JSON string i također dodaju u *FormData*. Koristeći HTTP POST zahtjev podaci se šalju na *backend*. Odgovor s *backend-a* sadrži *advice* koji predstavlja generirani savjet.

Taj se savjet postavlja kao tekstualna poruka u aplikaciji putem funkcije *setExtractedText*. Nakon uspješnog slanja brišu se sve lokalno pohranjene fotografije, resetiraju se brojač i niz fotografija i omogućuje se prikaz savjeta.

```
const uploadPhotos = async () => {
  if (photos.length === numProducts * 2 && personalData) {
    setIsProcessing(true);
    const formData = new FormData();

    photos.forEach((photo, index) => {
      const photoKey = photo.type === 'ingredients'
        ? `ingredients_image_${Math.floor(index / 2)}`
        : `nutrition_image_${Math.floor(index / 2)}`;

      formData.append(photoKey, {
        uri: photo.uri,
        type: 'image/jpeg',
        name: `${photoKey}.jpg`,
      });
    });

    formData.append('personal_data', JSON.stringify(personalData));

    try {
      const response = await axios.post('http://192.168.1.7:5000/meal-advice', formData,
        headers: { 'Content-Type': 'multipart/form-data' },
      });

      const { advice } = response.data;
      setExtractedText([{ sender: "App", text: `${advice}`, type: "Advice" }]);

      photos.forEach(async (photo) => {
        await FileSystem.deleteAsync(photo.uri, { idempotent: true });
      });

      setPhotos([]);
      setIsProcessing(false);
      setShowAdvice(true);
    } catch (error) {
      console.error('Error uploading photos:', error);
    }
  } else {
    Alert.alert('Please capture all required photos and ensure user data is loaded');
  }
};
```

Slika 4.21. *Programski kod funkcije uploadPhotos*

Na *backend* strani funkcija *meal_advice* implementira API rutu koja procesira podatke obroka i generira nutricionističke preporuke. Riječ je o POST zahtjevu koji prihvaća podatke o korisniku i slike sastojaka i nutritivnih tablica za proizvode. Njezin programski kod prikazan je na slici 4.22. Prvo se brišu sve prethodno spremljene slike u direktoriju *MEAL_FOLDER* kako bi se osiguralo da je direktorij prazan prije spremanja novih slika. Nakon toga, podaci o korisniku dohvaćaju se iz *form* dijela POST zahtjeva. Inicijaliziraju se prazne liste za sastojke i nutrijente svih proizvoda. Zatim, u petlji se prolazi kroz sve proizvode i dohvaćaju se slike za proizvod. Slike se spremaju u privremeni direktorij. Nakon toga primjenjuje se OCR model za prepoznavanje teksta iz slika i

pozivaju funkcije za izdvajanje sastojaka i nutritivnih vrijednosti kao i u */product-advice/* ruti. Potrebne vrijednosti se također spremaju u JSON datoteke. Upit se generira funkcijom *create_meal_advice_prompt* na osnovu svih sastojaka, nutrijenata i osobnih podataka korisnika. Zatim se poziva funkcija za generiranje savjeta i vraća generirani savjet u JSON formatu.

```python
@app.route('/meal-advice', methods=['POST'])
def meal_advice():
    try:
        clear_directory(MEAL_FOLDER)

        personal_data_json = request.form.get('personal_data')
        if personal_data_json:
            personal_data = json.loads(personal_data_json)
        else:
            return jsonify({"error": "No personal data received"}), 400

        all_ingredients = []
        total_nutritional_values = []

        product_count = len([key for key in request.files.keys() if 'ingredients_image_' in key])
        for index in range(product_count):
            ingredients_image = request.files.get(f'ingredients_image_{index}')
            nutrition_image = request.files.get(f'nutrition_image_{index}')

            if not ingredients_image or not nutrition_image:
                return jsonify({"error": f"Images for product {index + 1} are missing."}), 400

            ingredients_img_path = os.path.join(MEAL_FOLDER, f'ingredients_{index}.jpg')
            nutrition_img_path = os.path.join(MEAL_FOLDER, f'nutrition_{index}.jpg')
            ingredients_image.save(ingredients_img_path)
            nutrition_image.save(nutrition_img_path)

            ingredients_doc = DocumentFile.from_images(ingredients_img_path)
            nutrition_doc = DocumentFile.from_images(nutrition_img_path)

            ingredients_page = ocr_model(ingredients_doc)
            nutrition_page = ocr_model(nutrition_doc)

            extracted_ingredients = extract_ingredients(ingredients_page.pages[0].export())
            nutritional_values = extract_nutrition_values(nutrition_page.pages[0].export())

            all_ingredients.extend(extracted_ingredients)
            total_nutritional_values.append(nutritional_values)

        save_json({"ingredients": all_ingredients}, os.path.join(MEAL_FOLDER, "ingredients.json"))
        save_json({"nutrition_values": total_nutritional_values}, os.path.join(MEAL_FOLDER, "nutrition_values.json"))

        prompt = create_meal_advice_prompt(personal_data, total_nutritional_values, all_ingredients)
        advice = generate_advice(prompt)

        return jsonify({"advice": advice})

    except Exception as e:
        print(f"Error: {e}")
        return jsonify({"error": "An error occurred while processing the meal advice.", "details": str(e)}), 500
```

Slika 4.22. *Prikaz /meal-advice/ rute*

Funkcija *create_meal_advice_prompt* za generiranje upita koji sadrži sve korisničke podatke i informacije o obroku prikazana je na slici 4.23. Upit je vrlo sličan onomu za savjet o proizvodu, ali ovaj upit sadrži objašnjenje da je riječ o listi proizvoda planiranih za obrok i da je potreban nutricionistički savjet o obroku temeljen na korisnikovim osobnim podacima.

```python
def create_meal_advice_prompt(personal_data, total_nutritional_values, all_ingredients):
    prompt = f"""
    The user has provided their profile and a list of products they plan to consume in a meal.
    Assess if these products, as a combined meal, satisfy the user's dietary goals and needs.

    **User Profile:**
    - Age: {personal_data.get('age', 'N/A')} years old
    - Gender: {personal_data.get('gender', 'N/A')}
    - Activity level: {personal_data.get('activity_level', 'N/A')}/5
    - Height: {personal_data.get('height', 'N/A')} cm tall
    - Wight: {personal_data.get('weight', 'N/A')} kg
    - Target Weight: {personal_data.get('target_weight', 'N/A')} kg.

    **Meal Nutritional Summary:**
    {"; ".join([f"Product {i + 1}: " + ', '.join(f"{key}: {value}" for key, value in values.items())
                for i, values in enumerate(total_nutritional_values)])}

    **Ingredients in the Meal**:
    {', '.join(all_ingredients)}

    Provide a short, personalized dietary recommendation (no more than 5 sentences) that highlights
    any key benefits or concerns, and suggests any minor adjustments if needed to better align with the user's goals.
    """
    return prompt

def generate_advice(prompt):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a nutrition expert."},
            {"role": "user", "content": prompt}
        ],
        max_tokens=250,
        temperature=0.4
    )
    return response.choices[0].message['content']
```

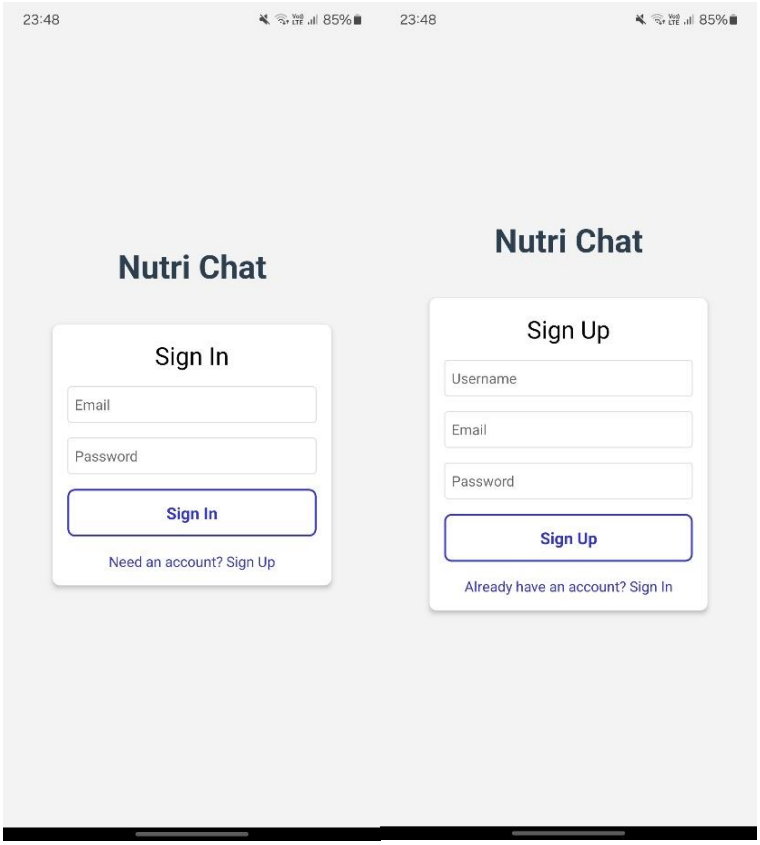
Slika 4.23. *Funkcija za kreiranje upita za savjet o obroku*

5. TESTIRANJE

U ovome poglavlju opisani su rezultati testiranja prethodno opisanih funkcionalnosti mobilne aplikacije. Testiranje je ključni korak kojim se u procesu razvoja aplikacije omogućuje detaljno identificiranje potencijalnih problema i nedostataka. Provjerom korisničkog iskustva doznaje se zadovoljava li aplikacija zadane korisničke potrebe i radi li prema očekivanjima. Provedena je analiza rada svakog dijela aplikacije, počevši od registriranja korisnika, unosa korisničkih podataka te obrade fotografija i generiranja savjeta temeljenih na korisničkim potrebama. U izradi aplikacije naglasak je bio na jednostavnom i intuitivnom korisničkom iskustvu (engl. UX - *user experience*) koje će svakom korisniku omogućiti jednako i brzo korištenje svake funkcionalnosti.

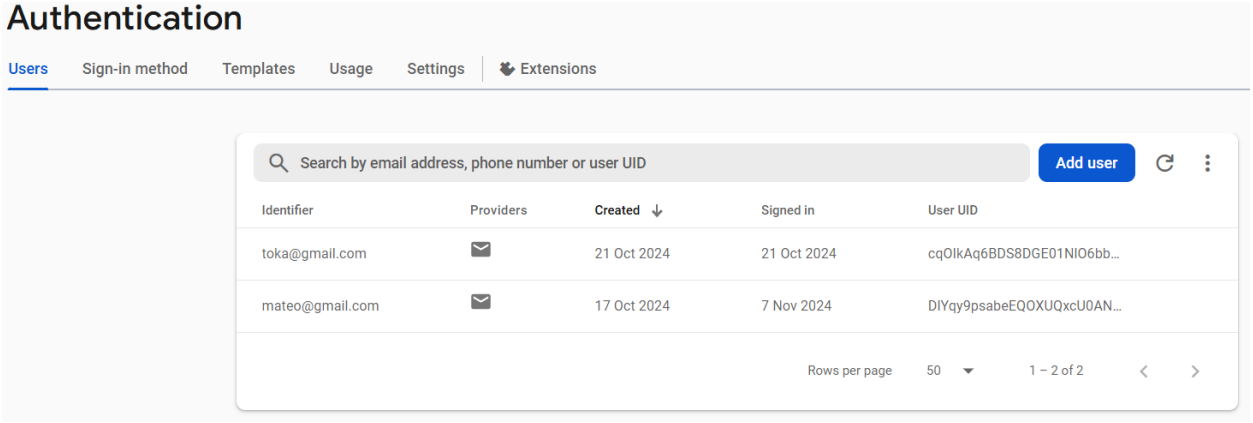
5.1. Registracija i prijava korisnika

Kako je prikazano na slici 5.1. prilikom pokretanja aplikacije, na početnom zaslonu prikazuje se forma za prijavu korisnika u sustav. Ukoliko korisnik nije registriran, pritiskom na dio forme označen s tekstom „*Need an account? Sign Up*“ preusmjeren je na formu za registraciju.



Slika 5.1. *Zasloni za prijavu/registraciju*

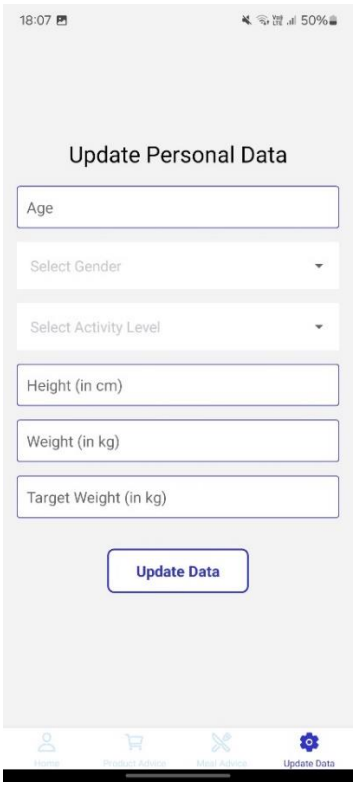
U tablici na slici 5.2 prikazani su podaci unutar *Firebase Authentication* baze podataka s registriranim korisnicima i njihovim email adresama, datumima kreiranja računa, datumima prijave i *User UID* vrijednostima kao glavnim ključem.



Slika 5.2. *Firebase baza podataka korisnika*

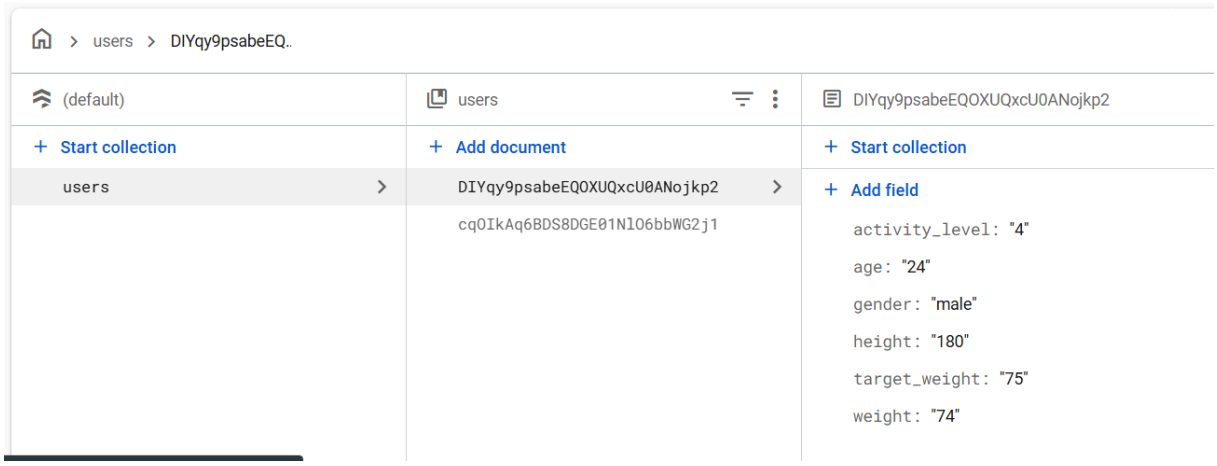
5.2. Unos korisničkih podataka

Nakon prijave u sustav, korisnik je preusmjeren na zaslon za unos i promjenu korisničkih podataka prikazan na slici 5.3. Na zaslonu je prikazana forma za unos godina, spola, razine aktivnosti, visine, težine i ciljane težine korisnika.



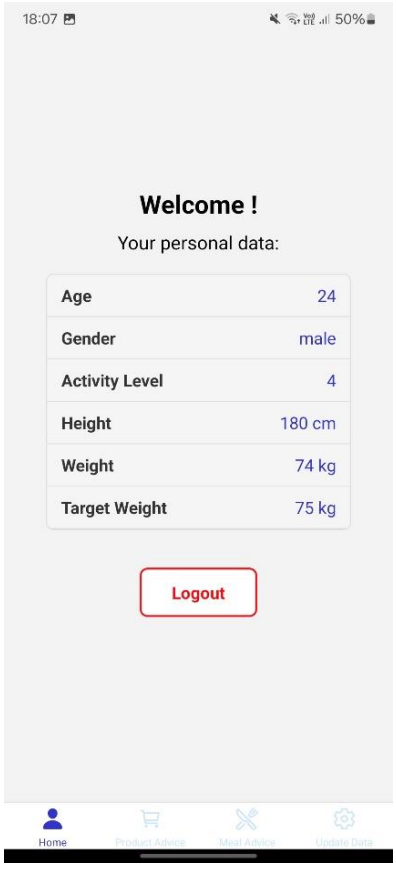
Slika 5.3. *Zaslon za unos korisničkih podataka*

Ispunjavanjem podataka u formi, mijenjaju se podaci pohranjeni u *Firestore Database* tablici. Na slici 5.4. prikazana je baza podataka s korisničkim podacima.



Slika 5.4. *Firestore Database baza podataka s korisničkim podacima*

Korisnički podaci su također prikazani i na glavnom zaslonu aplikacije u obliku tablice. Glavni zaslon je prikazan na slici 5.5. za korisnika koji je registriran, a na prethodnoj slici su prikazani njegovi osobni podaci iz baze podataka.



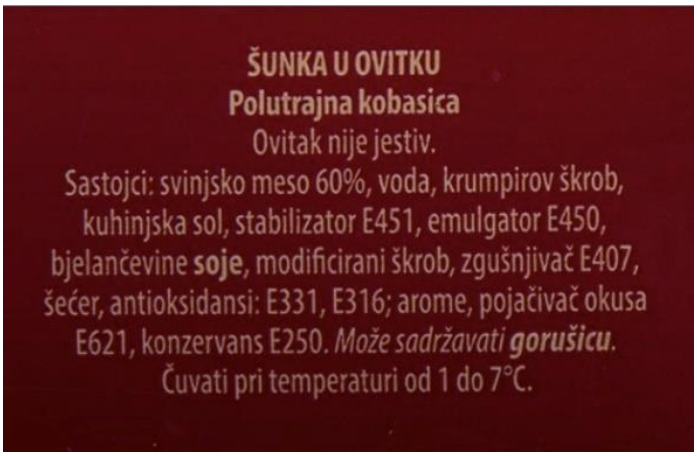
Slika 5.5. *Prikaz osobnih podataka korisnika*

5.3. Generiranje savjeta o proizvodu

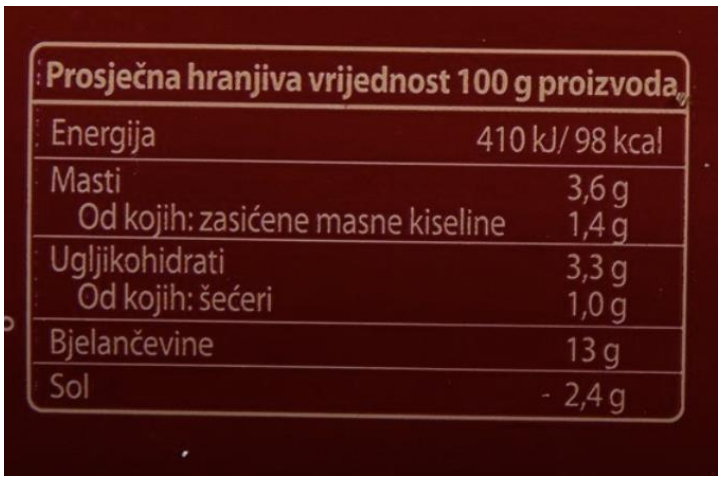
Pritiskom na „*Product Advice*“ na navigacijskoj traci koja se nalazi na dnu zaslona, korisniku se otvara zaslon s kamerom prikazan na slici 5.6. te s uputstvom da prvo uslika sliku sastojaka. Nakon fotografiranja sastojaka pritiskom na ikonu kamere, potrebno je uslikati tablicu nutritivnih vrijednosti.

Slika 5.6. *Zaslon Product advice zaslona*

Za generiranje savjeta odabran je proizvod šunke u ovitku čiji su sastojci i hranjive vrijednosti prikazani na slici 5.7. i 5.8.

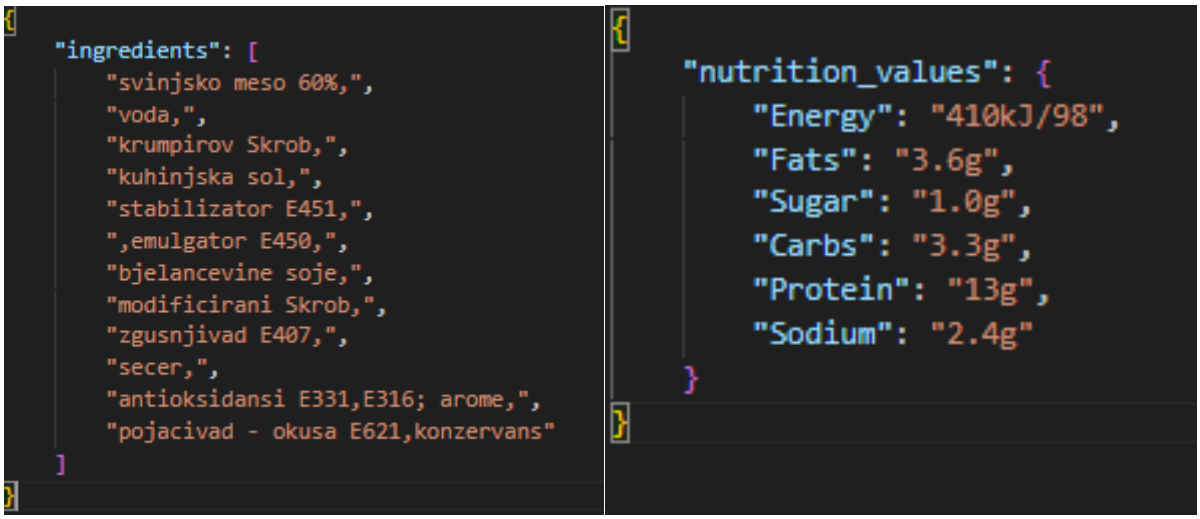


Slika 5.7. *Sastojci šunke u ovitku*



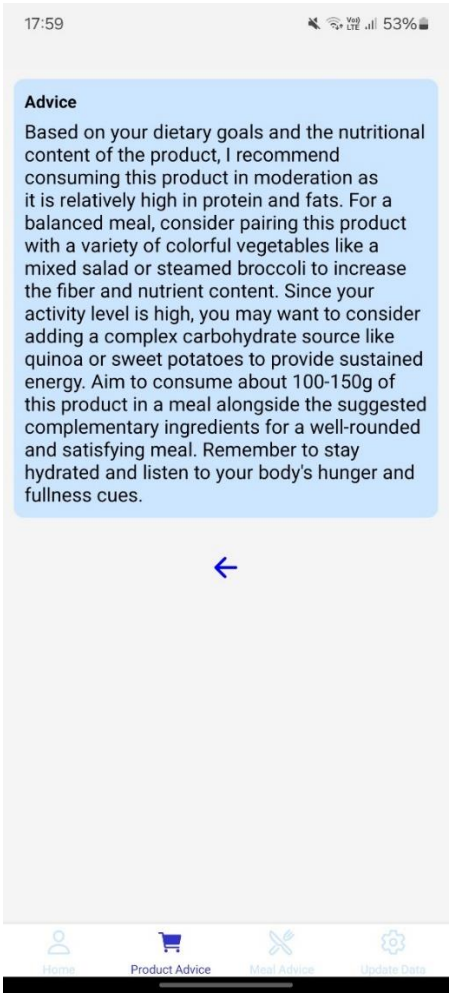
Slika 5.8. *Hranjive vrijednosti šunke u ovitku*

Fotografiranjem ovoga proizvoda kreiraju se *json* dokumenti na serverskoj strani s vrijednostima koje je *docTR* model uspio prepoznati na slikama. Na slici 5.9 prikazani su prepoznati sastojci i nutritivne vrijednosti spremljeni u *ingredients.json* i *nutritional_values.josn* datoteku unutar *product* mape.



Slika 5.9. *Prepoznati sastojci i nutritivne vrijednosti*

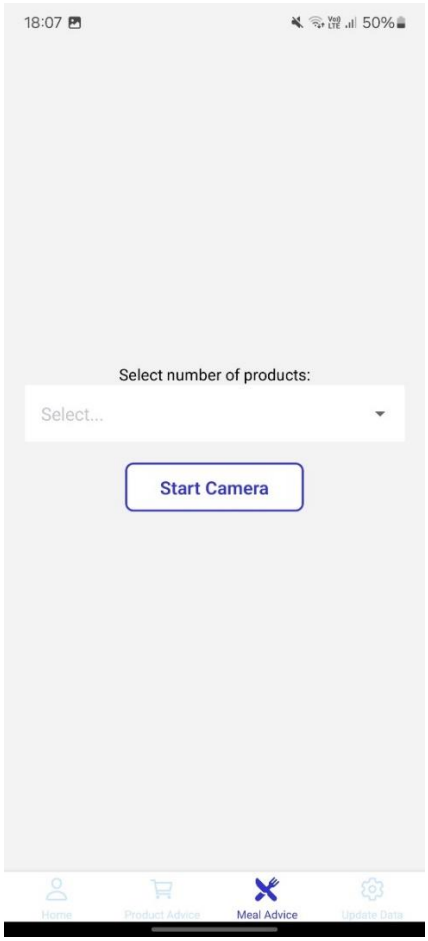
Dobiveni podaci o prehrambenom proizvodu i podaci o korisniku korišteni su za generiranje savjeta prikazanog na slici 5.10.



Slika 5.10. *Savjet o prehrambenom proizvodu*

5.4. Generiranje savjeta o obroku

Ukoliko korisnik na navigacijskoj traci klikom na „*Meal Advice*“ opciju izabere mogućnost dobivanja savjeta o obroku, otvorit će mu se zaslon prikazan slikom 5.11. koji od njega traži da odabere od 1 do 10 proizvoda ovisno koliko želi uvrstiti u svoj obrok.



Slika 5.11. *MealAdvice zaslon*

Nakon odabira broja namirnica, korisnik mora uslikati sastojke i tablice nutrijenata svakog proizvoda. Za potrebe testiranja aplikacije korištene su 4 namirnice: ementaler (prvi uslikani proizvod), šunka u ovitku (drugi uslikani proizvod), kiseli krastavci (treći uslikani proizvod) i jogurt (četvrti uslikani proizvod). Na slikama 5.12., 5.13. i 5.14. prikazani su sastojci i nutritivne vrijednosti ovih proizvoda, s iznimkom slika sastojaka i nutritivnih vrijednosti šunke u ovitku koje su korištene i za testiranje funkcionalnosti generiranja savjeta o proizvodu (slike 5.7. i 5.8.).

(CZ) **Ementál, přírodní sýr zrající, plátky. Obsah tuku v sušině 45 %. Obsah sušiny 60 %. Složení: mléko**, jedlá sůl, mikrobiální syřidlo, mlékárenské kultury. Baleno v ochranné atmosféře. Po otevření uchovejte v chladničce při teplotě od +2 °C do +7 °C a spotřebujte do 3 dnů. (HR) **Emmentaler, tvrdi punomasni sir u listićima, min. 45 % mliječne masti u suhoj tvari. Sastojci:** pasterizirano kravlje **mlijeko**, sol, mikrobiološko sirilo, mljekarske kulture. Pakirano u zaštitnoj atmosferi. Nakon otvaranja čuvati u hladnjaku pri temperaturi od +2 °C do +7 °C konzumirati unutar 3. (RO) (MD) **Brânză Emmental, cu pastă tare, maturată, feliată. Conținut de grăsime raportat la substanța uscată min. 45 %; minimum 60 % substanța uscată. Conținut de grăsime 30%. Ingrediente: lapte** de vacă pasteurizat, sare, cheag microbian, culturi lactice (contine **lapte**). Produs în Germania cu lapte din Germania și Cehia. Ambalat în atmosferă protectoare. A se păstra la temperaturi cuprinse între +2 °C și +7 °C. După deschidere, depozitați în frigider și consumați produsul în decurs de 3 zile. **RO: Distribuitor: Kaufland România SCS, str. Barbu Văcărescu, nr. 120-144, Sector 2, 020284 Bucureşti, România. MD: Importator/ Distribuitor: Kaufland SRL, str. Sfatul Țării, nr. 29, Chişinău, MD-2012, Republica Moldova.**

Výživové údaje / Prosječne hranjive vrijednosti / Declarație nutrițională	Na / U / per 100 g
Energetická hodnota / Energija / Valoare energetică	1536 kJ / 370 kcal
Tuky / Masti / Grăsimi	28,5 g
z toho nasycené mastné kyseliny / od kojih zasićene masne kiseline/ din care acizi graşi saturaţi	20 g
Sacharidy / Ugljikohidrati / Glucide	<0,1 g
z toho cukry / od kojih šećeri / din care zaharuri	<0,1 g
Bílkoviny / Bjelančevine / Proteine	28,2 g
Sůl / Sol / Sare	0,70 g

#37830 - 4007650

Výrobce / Proizvođač / Procesator şi ambalator: Goldsteig Käsereien Bayerwald GmbH, Siechen 11, 93413 Cham, Německo / Njemačka / Germania

Hmotnost / Neto količina / Cantitate netă:

150 g e

Slika 5.12. *Sastojci i nutritivne vrijednosti ementalera*

Krastavci delikates, pasterizirani
Sastojci: krastavci, voda, ocat, kuhinjska sol, šećer, ***gorušica*** i začini.
Nakon otvaranja čuvati u hladnjaku i potrošiti u roku od 5 dana. Proizvođač: Naturala d.o.
Nikole Šubića Zrinskog 28, 33520 Slatina, Hrvatska, tel.:+385 33 673 988, fax: +385 33 673 940
e-mail: info@naturala.eu
Zemlja podrijetla: Hrvatska
Neotvoreno čuvati na suhom i hladnom mjestu, zaštićeno od direktne sunčeve svjetlosti.
Najbolje upotrijebiti do: datuma označenog na poklopc
Neto količina: 540 g e
Ocijeđena neto masa: 290 g

Prosječna hranjiva vrijednost (100g proizvoda):	
Energija	56 kJ / 13kcal
Masti	0,0 g
-od kojih zasićene masne kiseline	0,0 g
Ugljikohidrati	2,5 g
-od kojih šećeri	1,7 g
Bjelančevine	0,8 g
Sol	1,3 g

3 858894 541669

Slika 5.13. *Sastojci i nutritivne vrijednosti kiselih krastavaca*

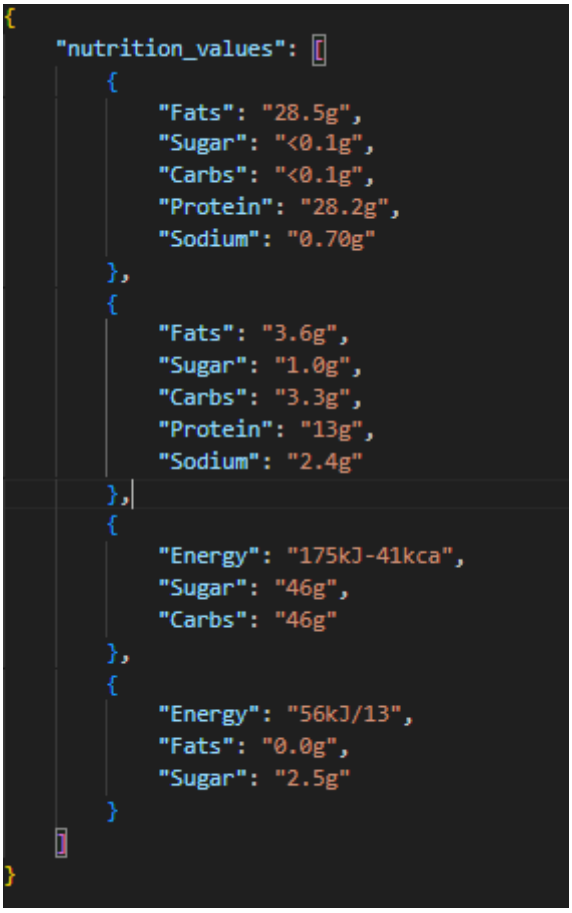
Slika 5.14. *Sastojci i nutritivne vrijednosti jogurta*

Nakon što korisnik uslika sve potrebne slike, na serverskoj strani odvija se spremanje slika u *meals* mapu. Nakon uspješnog prepoznavanja sastojaka i nutritivnih vrijednosti, kreirane su *ingredients.json* i *nutritional_values.json* datoteke. Kako je prikazano na slici 5.15., *ingredients.json* datoteka sadrži listu svih sastojaka.


```
{
    "ingredients": [
        "C2) Ement\u00f41,",
        "prirodni syr zrajici,",
        "pasterizirano kravlje mlijeko,",
        "sol,",
        "mikrobiolosko sirilo,",
        "suhoj Pakirano u zastitnoj",
        "svinjsko meso 60%,",
        "voda,",
        "krumpirov Skrob,",
        "kuhinjska sol,",
        "stabilizator E451,",
        "emulgator E450,",
        "bjelancevine soje,",
        "modificirani Skrob,",
        "zgusnjivad E407,",
        "secer,",
        "antioksidansi E331,",
        "- E316; arome,",
        "pojacivad okusa E621,",
        "konzervans",
        "pasterizirano mlijeko st%mijedner masti,",
        "jogurtna kultura,",
        "mikrobioloske kultureLA-50 (Lactobacillus hophilus)iBB-120 Bifidobacterium) Cuvaf empeaturiod4\"C-do",
        "krastavci,voda,ocat kuhinjska - sol secer,"
    ]
}
```

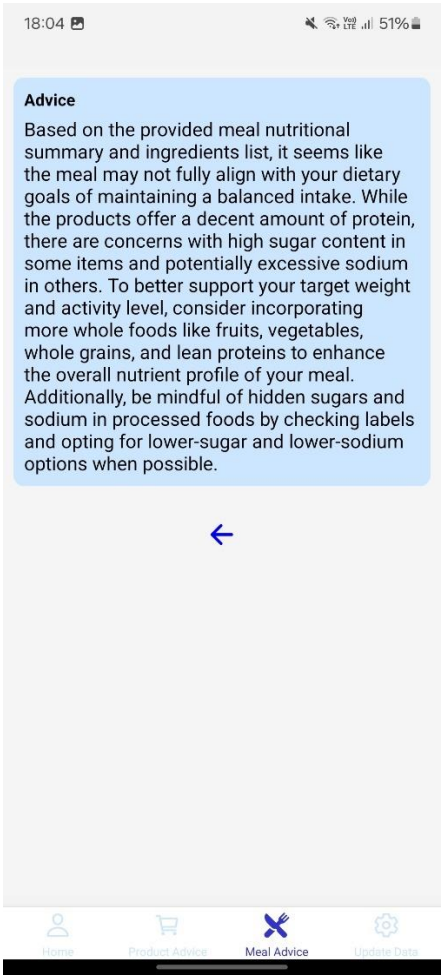

Slika 5.15. *Sastojci svih proizvoda*

Slika 5.16. prikazuje *nutritional_values.json* datoteku unutar koje se nalazi lista s objektima koji sadrže nutritivne parametre svakog proizvoda.



Slika 5.16. *Slika liste nutritivnih vrijednosti proizvoda*

Na osnovu prepoznatog sadržaja, korisnikovih podataka i naredbe predane pozivom OpenAI API ključa, GPT model je korisniku generirao savjet prikazan slikom 5.17.



Slika 5.17. *Generirani savjet o obroku*

6. ZAKLJUČAK

Izradom mobilne aplikacije ostvareni su ciljevi potrebni za ispunjenje zadatka ovog diplomskog rada. Uporabom React Native radnog okvira JavaScript programskog jezika uspješno je izrađena klijentska mobilna aplikacija koja osigurava sigurnost i zaštitu korisnikovih osobnih podataka, potrebne funkcionalnosti poput korisničkog sučelja i fotografiranja slika uz uspješnu komunikaciju sa serverskom stranom. Serverski dio aplikacije ostvaren Flask radnim okvirom programskog jezika Python pokazao se uspješnim u prijemu i slanju podataka s klijentskom stranom. Korištenjem alata i tehnologija poput Firebase Authentication-a i Firestore Database-a dokazane su uspješne i jednostavne implementacije registracije/prijave korisnika te pohrane i izmjene osobnih podataka. Ove tehnologije uvelike ubrzavaju razvoj aplikacije te omogućuju skalabilnost i sigurnost prilikom uporabe njezinih alata.

Osigurano je lako korištenje i dostupnost širokoj publici jednostavnim i intuitivnim korisničkim sučeljem. Osim toga, najznačajniji ostvareni rezultati postignuti su integracijom moderne tehnologije računalnog vida i velikih jezičnih modela. Testiranjem aplikacije pokazalo se da sustav može prepoznati sastojke i nutritivne informacije iz slika pomoću docTR modela te na temelju tih podataka generirati prilagođene savjete za korisnike korištenjem GPT modela predajom upute kroz OpenAI API ključ.

Iako je aplikacija značajkama zadovoljila potrebe diplomskog rada, uočene su poteškoće i nedostaci koje je moguće daljnjim istraživanjem i razvojem riješiti. Kod same obrade slike i postupka OCR-a prepoznavanje teksta i znakova uvelike ovisi o kvaliteti slike, boji pozadine i neujednačenom osvjetljenju. Budući da su proizvodi kupljeni na hrvatskom tržištu, samim time tekst je pisan znakovima hrvatske abecede. To je razlog nemogućnosti ispravnog prepoznavanja slova koja sadrže dijakritičke znakove, kao što su č, ć, đ, š, ž.

Dodatno ograničenje predstavlja i savjet koji je generiran GPT-3.5-turbo modelom. Za generiranje savjeta u ovoj aplikaciji korišten je mali broj korisnikovih osobnih podataka. Da bi se detaljnije odradila analiza i priprema za generiranje savjeta, potrebno je uzeti u obzir velik broj podataka o korisniku. Godine, spol, razina aktivnosti, visina, trenutna i ciljana težina samo su osnovni podaci, a uz njih je potrebno imati uvid u korisnikove prehrambene navike i zdravstveno stanje.

Korišteni veliki jezični model GPT-3.5-turbo model je opće namijene obučen na velikom broju podataka. Za generiranje boljih savjeta preporučljivo je određeni model fino ugađati na podacima

iz baza koje sadrže licencirane i stručne prehrambene rezultate analiza izrađene od strane nutricionista i relevantnog osoblja.

Uz sve navedeno, ova aplikacija zadovoljava potrebne zahtjeve zadatka diplomskog rada. Moguća je njezina šira upotreba u prehrambenim savjetodavnim centrima i zdravstvenim ustanovama. Također, moguća je primjena u obrazovne svrhe za podizanje svijesti o važnosti pravilne prehrane. Dodatnim razvojem i istraživanjima u suradnji sa nutricionistima, aplikacija bi se dodatno prilagodila stvarnim uvjetima uporabe i proširila na veći broj korisnika. Ovim radom otvara se mogućnost za daljnji razvoj u smjeru potpune automatizacije prehrambenog savjetovanja putem mobilne aplikacije.

LITERATURA

[1] "OpenFoodFacts", OpenFoodFacts, Dostupno na: https://world.openfoodfacts.org/discover [19.11.2024.].

[2] "Put your diet on autopilot.", EatThisMuch, Dostupno na: https://www.eatthismuch.com/ [19.11.2024.].

[3] M. Morell, L. DeSoto, C. Kacyon i L. Robinson, "7 Best Nutrition Apps: Tried and Tested in 2024", HelpGuide.org, 17.11.2024., Dostupno na: https://www.helpguide.org/handbook/healthy-living/best-nutrition-apps. [19.11.2024.]

[4] S. Forman, "The Noom Trial 2024: Here's Everything You Need to Know to Get Started", US Magazine, 23.6.2022., Dostupno na: https://www.usmagazine.com/shop-with-us/news/noom-free-trial-exercise-diet-weight-loss-health-home/. [19.11.2024.]

[5] R. Ajmera, "A Dietitian's Picks of the 10 Best Nutrition Apps to Download", healthline, 14.10.2022., Dostupno na: https://www.healthline.com/nutrition/top-iphone-android-apps#our-picks [19.11.2024.]

[6] L. Strong, "PlateJoy Review (2024): Meal Planning Just Got Easier With This App", Garage Gym Reviews, 23.2.2024., Dostupno na: https://www.garagegymreviews.com/platejoy-review [19.11.2024.]

[7] "What is Full Stack Development?", AWS, Dostupno na: https://aws.amazon.com/what-is/full-stack-development/ [22.11.2024.]

[8] H. Simonyan, "Web Development Basics: Frontend, Backend, Full Stack", Medium, 23.1.2023., Dostupno na: https://levelup.gitconnected.com/web-development-basics-frontend-backend-full-stack-5411406e1e36 [12.11.2024.]

[9] "Front-End Development: The Complete Guide", Claudinary, 23.10.2024., Dostupno na: https://cloudinary.com/guides/front-end-development/front-end-development-the-complete-guide [7.11.2024.]

[10] M. Budziński, "What Is React Native? Complex Guide for 2024" netguru, 2024., Dostupno na: https://www.netguru.com/glossary/react-native [22.11.2024.]

[11] "Writing Markup with JSX", React, Dostupno na: https://react.dev/learn/writing-markup-with-jsx [22.11.2024.]

[12] N. L. Thanh, "What Is Backend Development?", Medium, 18.7.2023., Dostupno na: https://namtheartist95.medium.com/what-is-backend-development-bcc6a15f8472 [22.11.2024.]

[13] M. Grinberg, Flask Web Development, Sebastopol, CA: O'Reilly Media, 2014.

[14] "Firebase", Wikipedia, 16.11.2024., Dostupno na: https://en.wikipedia.org/wiki/Firebase [18.11.2024.]

[15] "Firebase Authentication", Google Firebase, 6.11.2024., Dostupno na: https://firebase.google.com/docs/auth [8.11.2024.]

[16] "Cloud Firestore", Google Firebase, 6.11.2024., Dostupno na: https://firebase.google.com/docs/firestore [8.11.2024.]

[17] "What are large language models (LLMs)?", IBM, Dostupno na: https://www.ibm.com/topics/large-language-models [9.11.2024.]

[18] D. Nelson, "Što je duboko učenje?", Unite.ai, 3.9.2024., Dostupno na: https://www.unite.ai/hr/%C5%A1to-je-duboko-u%C4%8Denje/ [17.11.2024.]

[19] D. Nelson, "Umjetna inteligencijaŠto je NLP (obrada prirodnog jezika)?" Unite.ai, 2024., Dostupno na: https://www.unite.ai/hr/%C5%A1to-je-obrada-prirodnog-jezika/ [17.11.2024.]

[20] "What is RNN (Recurrent Neural Network)?", AWS, 2024., Dostupno na: https://aws.amazon.com/what-is/recurrent-neural-network/ [17.11.2024.]

[21] "What is a recurrent neural network (RNN)?", IBM, Dostupno na: https://www.ibm.com/topics/recurrent-neural-networks [14.11.2024.]

[22] "What is a transformer model?", IBM, Dostupno na: https://www.ibm.com/topics/transformer-model [17.11.2024.]

[23] S. Cristina, "The Transformer Model", Machine Learning Mastery, 6.1.2023., Dostupno na: https://machinelearningmastery.com/the-transformer-model/ [18.11.2024.]

[24] H. Feng, Y. Wang, W. Zhou, J. Deng i H. Li, "DocTr: Document Image Transformer for Geometric Unwarping", pp. 3-4, University of Science and Technology of China

[25] P. Belagatti, "Understanding the Softmax Activation Function: A Comprehensive Guide", SingleStore, 11.3.2024., Dostupno na: https://www.singlestore.com/blog/a-guide-to-softmax-activation-function/ [17.11.2024.]

[26] "GPT-3", Wikipedia, 15.11.2024., Dostupno na: https://en.wikipedia.org/wiki/GPT-3#GPT-3.5. [16.11.2024.]

[27] A. Ipsen, "GPT Base, GPT-3.5 Turbo & GPT-4: What's the difference?", Pluralsight, 31.8.2023., Dostupno na: https://www.pluralsight.com/resources/blog/ai-and-data/ai-gpt-models-differences [7.11.2024.]

[28] A. Mandour, "GPT-3.5 model architecture", OpenGenus IQ, Dostupno na: https://iq.opengenus.org/gpt-3-5-model/ [19.11.2024.]

[29] A. Mittal, "Rješavanje halucinacija u modelima velikih jezika: pregled najsuvremenijih tehnika", Unite.ai, 19.1.2024., Dostupno na: https://www.unite.ai/hr/tackling-hallucination-in-large-language-models-a-survey-of-cutting-edge-techniques/ [17.11.2024.]

[30] "Optical character recognition", Wikipedia, 15.11.2024., Dostupno na: https://en.wikipedia.org/wiki/Optical_character_recognition [16.11.2024.]

[31] IBM, "What is optical character recognition (OCR)?", IBM, Dostupno na: https://www.ibm.com/think/topics/optical-character-recognition [8.11.2024.]

[32] R. Merritt, "What Is a Transformer Model?", Nvidia, 25.3.2022., Dostupno na: https://blogs.nvidia.com/blog/what-is-a-transformer-model/ [8.11.2024.]

[33] "Using a ScrollView", React Native, 23.10.2024., Dostupno na: https://reactnative.dev/docs/using-a-scrollview [20.11.2024.]

[34] "TextInput", React Native, 29.10.2024., Dostupno na: https://reactnative.dev/docs/textinput [21.11.2024.]

[35] L. Eagles, "How to use react-native-picker-select", LogRocket, 27.1.2022., Dostupno na: https://blog.logrocket.com/how-to-use-react-native-picker-select/ [21.11.2024.]

[36] "Pressable", React Native, 23.10.2024., Dostupno na: https://reactnative.dev/docs/pressable [21.11.2024.]

[37] "What Does a Back-End Developer Do?", Cursera, 2024., Dostupno na: https://www.coursera.org/articles/back-end-developer [7.11.2024.]

[38] M. Herman, "What is Werkzeug?", testdriven.io, 18.6.2022., Dostupno na: https://testdriven.io/tips/b8604459-e836-47f9-8c6c-f22f9420b39e/ [8.11.2024.]

[39] Microsoft, "What is middleware?", Microsoft Azure, 2024., Dostupno na: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-middleware [8.11.2024.]

SAŽETAK

Ovaj diplomski rad bavi se primjenom velikih jezičnih modela (LLM) za razumijevanje i generiranje savjeta o prehrani. Glavni problem obuhvaća prepoznavanje sastojaka i nutritivnih vrijednosti s prehrambenih proizvoda te integraciju tih podataka u sustav za pružanje korisnih preporuka. Problem je riješen razvojem mobilne aplikacije pomoću React Native i Python Flask tehnologija. Implementacijom docTR modela za optičko prepoznavanje znakova omogućeno je čitanje nutritivnih vrijednosti i sastojaka sa slike proizvoda. Sustav na temelju prepoznatih sastojaka i nutritivnih vrijednosti te korisničkih podataka iz Firestore baze podataka generira savjet koristeći OpenAI-jev GPT-3.5-turbo model. Ovaj rada svojim rezultatima potvrdio je mogućnost primjene LLM-a za pružanje personaliziranih prehrambenih preporuka otvarajući nove mogućnosti u digitalizaciji prehrane.

Ključne riječi: LLM, mobilna aplikacija, OCR, prehrambeni savjeti, React Native

ABSTRACT

Application of LLMs in understanding and generating dietary advice

This thesis deals with the application of large language models (LLM) in understanding and generating dietary advice. The main problem involves identifying ingredients and nutritional values from food products and integrating this data into a system for providing useful recommendations. The problem was solved by developing a mobile application using React Native and Python Flask technologies. By implementing the docTR model for optical character recognition, it is possible to read the nutritional values and ingredients from the product image. Based on recognized ingredients and nutritional values and user data from the Firestore database, the system generates advice using OpenAI's GPT-3.5-turbo model. With its result, this thesis confirmed the possibility of applying LLM to provide personalized nutritional recommendations, opening up new possibilities in the digitization of nutrition.

Keywords: dietary advice, LLM, mobile application, OCR, React Native

ŽIVOTOPIS

Autor ovog diplomskog rada je Mateo Tokić, student druge godine diplomskog studija Računarstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Nakon završetka Osnovne škole „Ljudevit Gaj“ Lužani, pohađao je Klasičnu gimnaziju fra Marijana Lanosovića s pravom javnosti u Slavonskom Brodu. Preddiplomski sveučilišni studij Komunikacija i informacijskih tehnologija završava na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku 2022. godine te iste godine upisuje i diplomski studij Računarstva, izborni blok Informacijske i podatkovne znanosti. Stručnu praksu odradio je u Protostar Labs-u u sklopu Data Science Internshipa.

PRILOZI

Prilog 1. Programsko rješenje aplikacije : https://github.com/MateoTokic/nutri-chat