

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

Sveučilišni studij

Recipe Realm: mobilna aplikacija za personaliziranu
pohranu recepata

Završni rad

Dorja Pozder

Osijek, 2024

**FERIT**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK**Obrazac Z1P: Obrazac za ocjenu završnog rada na sveučilišnom prijediplomskom studiju****Ocjena završnog rada na sveučilišnom prijediplomskom studiju**

Ime i prezime pristupnika:	Dorja Pozder
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	R4707, 28.07.2021.
JMBAG:	0165091810
Mentor:	izv. prof. dr. sc. Zdravko Krpić
Sumentor:	doc. dr. sc. Ivana Hartmann Tolić
Sumentor iz tvrtke:	
Naslov završnog rada:	Recipe Realm: mobilna aplikacija za personaliziranu pohranu recepata
Znanstvena grana završnog rada:	Programsko inženjerstvo (zn. polje računarstvo)
Zadatak završnog rada:	Zadatak je izraditi mobilnu aplikaciju koja će omogućiti korisnicima personalizirano spremanje recepata. Aplikacija će od korisnika zahtijevati registraciju, nakon koje oni moći stvarati osobnu biblioteku recepata uz razne funkcionalnosti aplikacije, kao što je kategorizacija, izmjena mjernih jedinica, pretraživanje, generiranje plana ishrane i slično. Aplikacija je namijenjena za osobe sklone prilagođavanju originalnih recepata, kojima će aplikacija omogućavati spremanje i organizaciju istih." (Rezervirano: Dorja Pozder) Sumentorica: dr. sc. Ivana Hartmann-Tolić
Datum prijedloga ocjene završnog rada od strane mentora:	04.09.2024.
Prijedlog ocjene završnog rada od strane mentora:	Izvrstan (5)
Datum potvrde ocjene završnog rada od strane Odbora:	11.09.2024.
Ocjena završnog rada nakon obrane:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio sveučilišni prijediplomski studij:	11.09.2024.



FERIT

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

IZJAVA O IZVORNOSTI RADA

Osijek, 11.09.2024.

Ime i prezime Pristupnika:

Dorja Pozder

Studij:

Sveučilišni prijediplomski studij Računarstvo

Mat. br. Pristupnika, godina upisa:

R4707, 28.07.2021.

Turnitin podudaranje [%]:

5

Ovom izjavom izjavljujem da je rad pod nazivom: **Recipe Realm: mobilna aplikacija za personaliziranu pohranu recepata**

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

i sumentora doc. dr. sc. Ivana Hartmann Tolić

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	1
1.1 Zadatak završnog rada.....	1
2. PREGLED POSTOJEĆIH RJEŠENJA	2
2.1 Recipe Keeper.....	2
2.2 Side Chef.....	3
2.3 Tasty.....	4
2.4 Idejno rješenje vlastite aplikacije za pohranu recepata.....	4
3. OPIS KORIŠTENIH TEHNOLOGIJA	5
3.1 Android Studio.....	5
3.2 Kotlin.....	5
3.2.1 Jetpack Compose.....	6
3.3 Firebase – Firestore Database.....	6
3.4 Figma.....	6
4. IMPLEMENTACIJA APLIKACIJE	8
4.1 Baza podataka.....	8
4.2 Registracija korisnika.....	9
4.3 Arhitektura mobilne aplikacije.....	10
4.4 Zaslone.....	11
4.4.1 Registracija i prijavljivanje.....	11
4.4.2 Zaslone sa svim receptima.....	12
4.4.3 Dodavanje novog recepta i kategorije.....	18
4.4.4 Generiranje tjednog plana ishrane.....	22
4.4.5 Detaljan prikaz recepta.....	23
5. KORIŠTENJE APLIKACIJE	25
5.1 Zaslone za registraciju i prijavu.....	25
5.2 Zaslone za pregled svih recepata.....	26
5.3 Zaslone za dodavanje novog recepta i kategorije.....	28
5.4 Zaslone za detaljan prikaz recepta.....	31
5.5 Zaslone za generiranje plana prehrane.....	32
5.6 Vrednovanje funkcionalnosti aplikacije.....	34
6. ZAKLJUČAK	36

LITERATURA.....	37
POPIS SLIKA.....	38
SAŽETAK.....	40
ABSTRACT.....	41
PRILOZI.....	42

1. UVOD

Priprema hrane oduvijek je bila dio svakodnevice većine ljudi. Za mnoge, kuhanje, osim nužne aktivnosti za održavanje života, predstavlja izvor kreativnosti i zadovoljstva. Ono je također znanost koja uključuje povezivanje sastojaka, u određenim količinama i omjerima kako bi se postigli točno određeni okusi. Pohrana recepata je ključan element u očuvanju ove umjetnosti koja se prenosi generacijama.

Davno su se recepti pohranjivali na komadiće papira, u bilježnice, neki su se i urezivali u kuhinjske elemente. No, rukopis izbljedi, papir se pokida ili zagubi te je vidljivo da su svi ti prijašnji oblici pohrane na neki način kratkotrajni. Također, koliko je kuhanje znanost, toliko je i umjetnost te je svačiji ukus različit. Ljudi su skloni eksperimentiranju i prilagođavanju recepata, a nije praktično čuvati 10 papira za 10 varijacija svakog recepta ili križati prijašnje napisan tekst kako bi se negdje na rub papira upisao novi.

Napretkom tehnologije otvorila su se vrata novim mogućnostima pohrane recepata na siguran i dugotrajan način. Izrade aplikacija za pohranu recepata omogućile su unos recepata u baze podataka kako bi oni bili dostupni za pregled na više različitih uređaja te bez obzira na njihovu starost. Također je moguće spremati recepte uz odgovarajuće slike što olakšava pretraživanje recepata unutar aplikacija. Pretraga je dodatno olakšana uvođenjem kategorizacije recepata. Ovaj završni rad će se usredotočiti na razvoj i implementaciju Android aplikacije za personaliziranu i jednostavnu pohranu recepata. Cilj je razviti aplikaciju koja omogućuje korisnicima jednostavan unos sastojaka i postupka pripreme za recept te ostalih bitnih informacija kao što su vrijeme pripreme i energetska vrijednost obroka. Razvijena aplikacija omogućuje i jednostavnu i personaliziranu kategorizaciju recepata te filtriranje prema istima te generiranje tjednog plana ishrane prema željenim kategorijama.

1.1. Zadatak završnog rada

Izraditi mobilnu aplikaciju koja bi korisnicima omogućila personaliziranu organizaciju i pohranu recepata. Potrebno je omogućiti unos novog recepta, filtriranje po korisnički kreiranim kategorijama te generiranje plana ishrane.

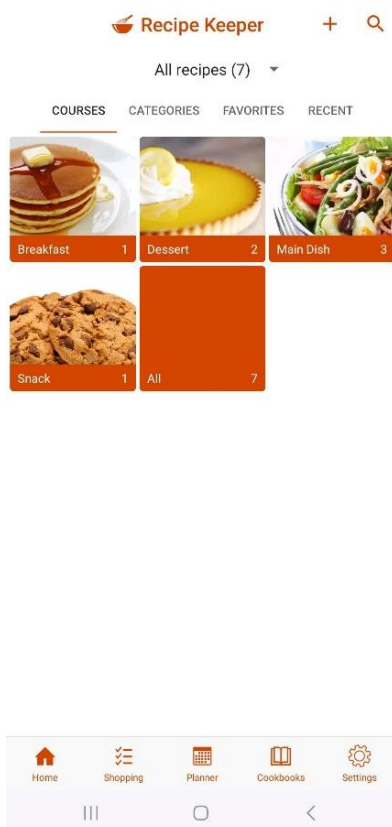
2. PREGLED POSTOJEĆIH RJEŠENJA

Budući da je kuhanje svakodnevnica većine ljudi, postoje već mnoge aplikacije za pohranu, otkrivanje i stvaranje recepata. Tržište ovakvih aplikacija dominiraju poznata imena prehrambenih mreža koja su prethodno bili poznati u časopisima ili prema drugim mrežnim sadržajima te su svoje usluge odlučili sažeti u mobilnu aplikaciju. Također postoje manje poznate aplikacije neovisnih programera koje nude slična rješenja.

Česta pojava u pregledanim aplikacijama je otključavanje potrebnih usluga plaćanjem unutar aplikacije, jednokratno ili preko mjesečne pretplate. Neke od aplikacija također nude mogućnosti pregleda recepata koje su objavili drugi korisnici te time proširuju bazu dostupnih recepata svakog korisnika. Aplikacije pregledane u nastavku su prisutne na tržištu te pružaju uvid u različita rješenja funkcionalnosti i pristupa korisnicima.

2.1. Recipe Keeper

Prvo istraženo rješenje je aplikacija Recipe Keeper čije je sučelje prikazano slikom 2.1.



Slika 2.1. Prikaz izgleda aplikacije *Recipe Keeper*, preuzeto s

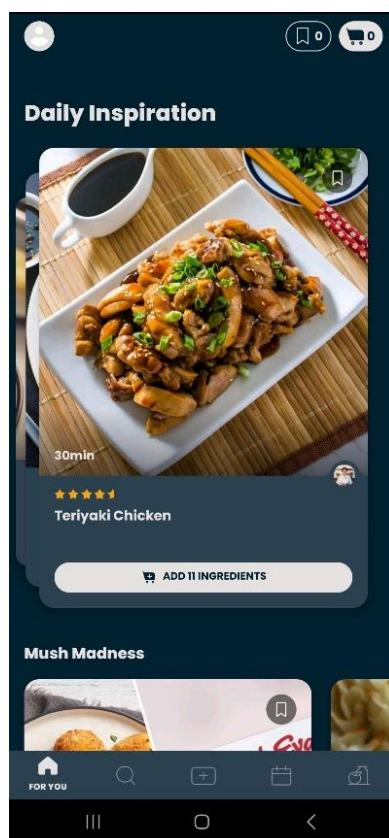
<https://play.google.com/store/apps/details?id=com.tudorspan.recipekeeper>

Prema [1] aplikacija nudi unos novih recepata ručno ili preko mrežnog linka na stranicu na kojoj se nalazi recept. Također omogućava naknadno uređivanje recepta, kreiranje plana

prehrane ručnim odabirom recepata i stvaranje popisa za kupovinu. Uz navedeno, aplikacija nudi mogućnost stvaranja vlastite kuharice pomoću već unesenih recepata u svrhu kreiranja dokumenta koji se može dijeliti bilo kojim putem. Dok Recipe Keeper ima opciju stvaranja plana prehrane, nedostaje mogućnost nasumične generacije koja će biti implementirana u ovom radu.

2.2. Side Chef

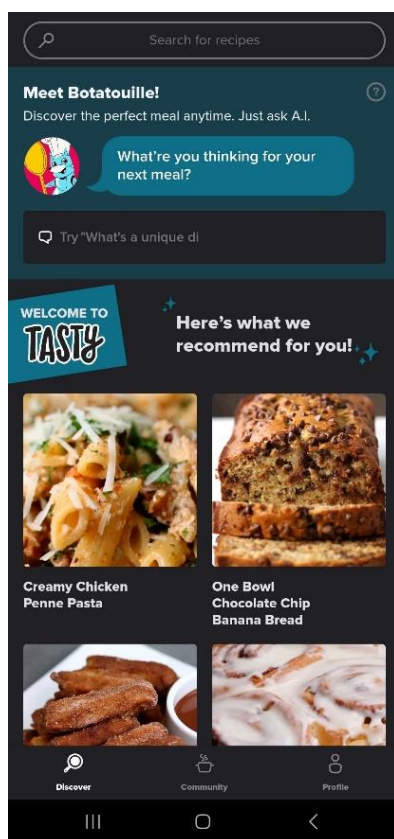
Iduća istražena aplikacija je Side Chef. Prema [2], ova se aplikacija više orijentira na dijeljenje recepata s drugim korisnicima. Također ima funkcije dodavanja vlastitih recepata ručno ili putem mrežne veze te prilikom stvaranja recepata korisnici imaju mogućnost učiniti recept privatnim ili javnim pri čemu će dobiti vezu koju mogu podijeliti s drugima. Iako Side Chef ima razne opcije za unos recepata, koraci pripreme unose se kao jedan tekst, dok će aplikacija ovog rada omogućiti unošenje koraka u obliku zasebnih stavki. U istom unosu također ne postoje opcije za unos vremena pripreme i nutritivnih vrijednosti te će iste biti implementirane u ovom radu.



Slika 2.2. Prikaz izgleda aplikacije *Side Chef*, preuzeto s <https://play.google.com/store/apps/details?id=com.sidechef.sidechef>

2.3. Tasty

Sljedeća istražena aplikacija je Tasty. Tasty je prehrambeni blog koji je kroz godine prikupio puno pratitelja te su na njihov zahtjev odlučili napraviti aplikaciju. Aplikacija je postavljena kao blog za hranu. Prema [3], korisnici mogu pretraživati objavljene recepte, komentirati i spremati recepte, međutim ne postoji opcija za unos vlastitog recepta. Tasty je također uveo opciju korištenja umjetne inteligencije pri potrazi za idućim receptom prema zahtjevima korisnika. Ovaj će se rad osvrnuti na nedostatak unosa vlastitog recepta te će isti biti omogućen u razvijenoj aplikaciji.



Slika 2.3. Prikaz izgleda aplikacije *Tasty*, preuzeto s

<https://play.google.com/store/apps/details?id=com.buzzfeed.tasty>

2.4. Idejno rješenje vlastite aplikacije za pohranu recepata

Prethodno navedene aplikacije i još mnoge druge na tržištu su motivirale razvoj ove aplikacije. Korisnik će moći unositi vlastite recepte i za njih bitne informacije, moći će recepte dijeliti u kategorije koje će biti moguće pretraživati. Razlika u odnosu na ostala slična rješenja je što ova aplikacija nudi mogućnost unosa i drugih informacija o receptu poput vremena pripreme, nutritivne vrijednosti te vlastite ocjene recepta. U poglavlju 4 je objašnjena implementacija navedenih funkcionalnost, a u poglavlju 5 je rad istih prikazan pomoću same aplikacije.

3. OPIS KORIŠTENIH TEHNOLOGIJA

Razvoj Android aplikacije zahtijeva korištenje različitih tehnologija i alata koji omogućuju implementaciju raznovrsnih funkcionalnosti, dizajna i performansi. Ovo će poglavlje detaljno predstaviti sve tehnologije koje su korištene tijekom razvoja aplikacije, kako bi bila osigurana njezina učinkovitost, pouzdanost i pristupačnost korisnicima. Naglasak poglavlja biti će na opisu razvojnih okvira, programskih jezika te baze podataka. Bit će pružen jasan uvid u tehničku osnovu aplikacije, objašnjavajući kako svaka od ovih komponenti doprinosi ukupnoj kvaliteti i funkcionalnosti krajnjeg proizvoda.

3.1. Android Studio

Prema [4] Android Studio je integrirano razvojno okruženje (IDE) koje Google nudi za razvoj aplikacija za Android operativni sustav. Temeljen na IntelliJ IDEA platformi, Android Studio pruža alate za razvoj, testiranje i otklanjanje pogrešaka Android aplikacija. Sadrži bogat skup funkcionalnosti, uključujući uređivač koda s pametnim nadopunama, vizualni uređivač za izradu korisničkog sučelja, emulator za testiranje aplikacija na različitim uređajima te alat za upravljanje projektima i verzijama. Android Studio također podržava integraciju s alatima kao što su Gradle za automatizaciju izgradnje aplikacija.

3.2. Kotlin

Prema [5] Kotlin je moderan, statički tipiziran programski jezik koji je razvio JetBrains, a koristi se prvenstveno za razvoj Android aplikacija. Dizajniran je tako da programeri mogu koristiti Kotlin i Java kôd unutar istog projekta. Kotlin je postao službeni jezik za Android razvoj 2017. godine, kada ga je Google službeno podržao kao alternativu Javi.

Kotlin se ističe po svojoj jednostavnosti i čitljivosti, smanjujući količinu "obimnog" koda koji je često potreban u Javi. Nudi mnoge napredne značajke, kao što su lambda funkcije, ekstenzijske funkcije, null-sigurnost, i korutine za asinkrono programiranje, što olakšava pisanje sigurnijeg i učinkovitijeg koda. Lambda funkcije omogućuju pisanje kratkih izraza koji direktno definiraju ponašanje bez potrebe eksplicitne deklaracije metoda. Ekstenzijske funkcije omogućuju programerima proširenje funkcionalnosti postojećih klasa bez korištenja nasljeđivanja ili dizajnerskih obrazaca sa sličnom ulogom. Null-sigurnost značajno smanjuje mogućnost pojave "null pointer" iznimaka, a pomoću korutina moguće je pisati kôd koji se ponaša sekvencijalno, ali se izvršava asinkrono.

3.2.1. Jetpack Compose

Prema [6], Jetpack Compose je moderni alat za izradu korisničkih sučelja (eng. User Interface, UI) za Android aplikacije, razvijen od strane Google-a. Za razliku od tradicionalnog načina razvoja Android sučelja koji koristi XML za definiranje izgleda, Jetpack Compose omogućuje izradu UI komponenata pomoću Kotlin koda. Ovaj pristup koristi deklarativni stil programiranja, gdje se opisuje kako bi korisničko sučelje trebalo izgledati na temelju trenutnog stanja aplikacije, a *framework* automatski upravlja promjenama stanja i osvježavanjem sučelja.

Osnovni gradivni element Jetpack Composea su `@Composable` funkcije koje omogućuju modularno i ponovljivo kreiranje UI komponenti. Jetpack Compose također podržava moderne prakse kao što su reaktivno programiranje i jednosmjerni tokovi podataka, što olakšava upravljanje stanjem i poboljšava održavanje koda.

Dodatno, alat dolazi s podrškom za teme i prilagođene stilove, omogućujući razvoj konzistentnog izgleda aplikacija. Jetpack Compose je optimiziran za performanse i uključuje značajke poput lijenog učitavanja, što poboljšava učinkovitost pri radu s velikim skupovima podataka.

3.3. Firebase – Firestore Database

Firebase je platforma koja programerima pruža razne vrste podrške za backend funkcionalnosti kao što su pohrana podataka, autentifikacija korisnika, slanje obavijesti i analiza podataka pri razvijanju aplikacija. Prema [7], najpoznatiji, a samim time i najčešće korišteni Firebase alati su njihove baze podataka. Firestore Database i Realtime Database obje korisnicima pružaju dobru organiziranost i pohranu podataka, skalabilnost te brzu upotrebu upita. Za potrebe ovog projekta koristi se Firestore Database.

3.4. Figma

Prema [8], Figma je cloud-based alat za dizajn i suradnju koji se koristi za izradu korisničkih sučelja (UI) i korisničkog iskustva (UX). Omogućuje dizajnerima da kreiraju, prototipiraju i dijele dizajne u stvarnom vremenu, što olakšava suradnju među timovima, bez obzira na njihovu fizičku lokaciju.

Figma pruža moćan skup alata za dizajn, uključujući alate za vektorsku grafiku, uređivanje boja, tipografiju i kreiranje složenih komponenata koje se mogu ponovno koristiti. Također

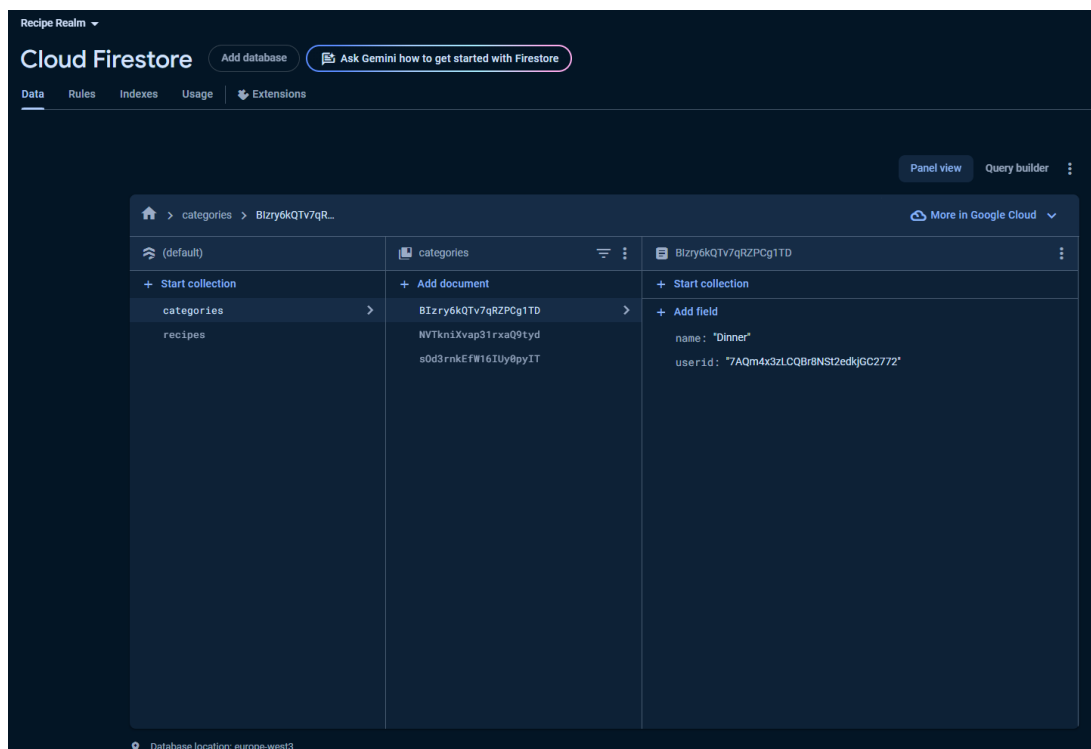
podržava prototipiranje, što omogućuje dizajnerima da izrade interaktivne prototipove kako bi testirali korisničko iskustvo prije nego što se dizajn implementira.

4. IMPLEMENTACIJA APLIKACIJE

Implementacija Android aplikacije predstavlja ključni dio razvoja, gdje se ideje i dizajnerski koncepti pretvaraju u funkcionalan softver. U ovom poglavlju detaljno će biti opisan postupak implementacije aplikacije, korak po korak. Obraditi će se glavne komponente sustava, uključujući arhitekturu aplikacije, podatkovne slojeve te korisničko sučelje. Cilj ovog poglavlja je pružiti sveobuhvatan pregled tehničkih aspekata razvoja aplikacije, te omogućiti dublje razumijevanje procesa koji su doveli do stvaranja konačnog proizvoda.

4.1. Baza podataka

Kao što je prethodno spomenuto, za izradu ove aplikacije korištena je Firestore Database baza podataka. Ovo je dokumentno-orijentirana baza podataka, što znači da podaci nisu organizirani u tablice od redova i stupaca. Umjesto toga, podaci su organizirani u dokumente koji se potom grupiraju u kolekcije. Svaki dokument ima svoju jedinstvenu oznaku te se mora nalaziti unutar neke kolekcije. Unutar dokumenata moguće je unijeti jednostavna polja poput tekstualnih polja, a i složenije strukture poput lista. Također, dokument može sadržavati ugniježdene kolekcije i dokumente [7].



Slika 4.1. Prikaz sučelja Firebase baze podataka

Na slici 4.1 nalazi se prikaz baze podataka ove aplikacije. Kolekcija „categories“ sadrži dokumente kategorija za recepte. Svaki taj dokument sastoji se od zapisa jedinstvene oznake korisnika te imena kategorije. Kolekcija „recipes“ sadrži dokumente s detaljima o svakom pojedinom receptu.

```
class Recipe {
    var id: String = ""
    var image: String = ""
    var title: String = ""
    var ingredients: List<String> = emptyList()
    var instructions: List<String> = emptyList()
    var kcal: String = ""
    var cookingTime: String = ""
    var rating: String = ""
    var category: String = ""
    val userId: String = ""
}
```

Slika 4.2. Klasa *Recipe*

```
class Category {
    var categoryName: String = ""
    val userId: String = ""
}
```

Slika 4.3. Klasa *Category*

Slike 4.2 i 4.3 prikazuju modele klasa *Recipe* i *Category* koji odgovaraju strukturi dokumenata koji se nalaze u bazi podataka. Klasa *Recipe* sastoji se od jedinstvene identifikacijske oznake, naslova recepta, slike, popisa sastojaka i koraka pripreme, kalorija, vremena potrebnog za pripremu, ocjene, kategorije kojoj recept pripada i jedinstvene oznake korisnika. Sva polja su znakovni nizovi osim sastojaka i koraka pripreme koji su liste znakovnih nizova. Klasa *Category* se sastoji od imena kategorije i jedinstvene oznake korisnika od kojih su oba polja znakovni nizovi.

4.2. Registracija korisnika

Za registraciju i autentifikaciju korisnika korišten je alat kojeg pruža Firebase. Kao metoda prijavljivanja u aplikaciju odabrana je „Email/Password“ metoda koja zahtijeva unos adrese elektroničke pošte te proizvoljne lozinke.



Slika 4.4. Prikaz sučelja Firebase autentifikacijskog alata

Na slici 4.4 moguće je vidjeti sučelje alata iz kojeg je moguće iščitati adresu elektroničke pošte svakog korisnika, datum kreiranja računa, datum zadnje prijave u aplikaciju te jedinstvenu oznaku korisnika. Alat također omogućuje ručno dodavanje korisnika te njihovo pretraživanje prema adresi elektroničke pošte, mobilnom broju ili jedinstvenoj oznaci korisnika.

4.3. Arhitektura mobilne aplikacije

Za izradu dobro strukturirane aplikacije postoje obrasci i tehnike koje je potrebno slijediti. Prema [9], jedna od najpopularnijih mobilnih arhitektura u posljednje vrijeme je MVVM arhitektura, odnosno Model-View-ViewModel arhitektura. Ona dijeli aplikaciju u 3 dijela: *Model*, *View* i *ViewModel*. Pojednostavljeno, dio *Model* predstavlja podatke, dio *View* predstavlja vizualnu reprezentaciju podataka, a dio *ViewModel* je posrednik između Modela i View-a.

Model je čista klasa koja je u potpunosti neovisna o vizualnim prikazima ili bilo kojim drugim komponentama aplikacije. Njena je uloga upravljati podacima i poslovnom logikom.

View upravlja korisničkim sučeljem aplikacije. Njegova je uloga prikazati podatke koje dobiva od *ViewModel*-a te *ViewModel*-u proslijediti reakcije i zahtjeve korisnika na daljnje obrađivanje.

ViewModel je ključan dio ove arhitekture. Često se opisuje kao most između *Model*-a i *View*-a zbog svoje uloge u njihovom povezivanju. On je odgovoran za pripremanje podataka koje će *View* pokazati te za odgovore na korisničke radnje kao što su pritisak gumba ili unos podataka.

4.4.Zasloni

Aplikacija je podijeljena na zaslone; svaki zaslon ima svoje korisničko sučelje te je povezan s drugim zaslonima pomoću navigacijskih elemenata. U nastavku će biti objašnjen svaki zaslon aplikacije te njegova implementacija i funkcionalnosti.

4.4.1.Registracija i prijavljivanje

Kao što je prethodno spomenuto, za proces registracije i prijave korisnika koristi se alat za autentifikaciju kojeg pruža Firebase. On je predstavljen pomoću ViewModel-a pod nazivom *AuthViewModel* koji je prikazan na slici 4.5. Ova klasa je odgovorna za upravljanje registracijom, prijavom i odjavom korisnika te spremanje korisničkih podataka.

```
class AuthViewModel : ViewModel() {
    private val auth: FirebaseAuth = FirebaseAuth.getInstance()

    fun loginUser(email: String, password: String, onComplete: (Boolean, String?) -> Unit) {
        auth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    onComplete(true, null)
                } else {
                    onComplete(false, task.exception?.message)
                }
            }
    }

    fun registerUser(email: String, password: String, onComplete: (Boolean, String?) -> Unit) {
        auth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    onComplete(true, null)
                } else {
                    onComplete(false, task.exception?.message)
                }
            }
    }

    fun logout() {
        auth.signOut()
    }
}
```

Slika 4.5. Klasa *AuthViewModel*

Prvi zaslon koji se prikazuje korisniku je zaslon za registraciju. Na njemu se nalaze dva polja za unos teksta, predviđena za unos e-mail adrese i lozinke. Ispod polja za unos nalaze se dva gumba, jedan označen s „Register“ te drugi označen s „I already have an account“. Pritiskom na gumb „Register“ poziva se funkcija *registerUser()* iz *AuthViewModel* klase. Zatim se poziva funkcija *createUserWithEmailAndPassword()* koju pruža Firebase te je odgovorna za stvaranje novog korisnika u Firebase bazi korisnika te ukoliko je registracija uspješna, korisnika se prosljeđuje na zaslon za prijavu.

Svako usmjerivanje na drugi zaslon odvija se pomoću funkcije *Navigation()* čiji je dio prikazan na slici 4.6. Cijela funkcija se sastoji od mnogo funkcija od kojih je svaka odgovorna za usmjeravanje na određeni zaslon. Pomoću *NavHost* komponente određuje se početni zaslon te *composable()* funkcija s rutom „register“ usmjerava korisnika na zaslon za registraciju pri otvaranju aplikacije. Na slici 4.6 je vidljivo kako funkcija *onRegisterSuccess()* pomoću *navController*-a i njegove funkcije *navigate()* usmjerava korisnika na sljedeći zaslon s rutom „login“.

Ukoliko korisnik već ima račun, može pritisnuti gumb „I already have an account“ te odmah biti preusmjeren na zaslon za prijavu.

```
@Composable
fun Navigation(
    viewModel: RecipeViewModel,
    authViewModel: AuthViewModel,
) {
    val navController = rememberNavController()
    NavHost(
        navController = navController,
        startDestination = "register"
    ){ this: NavGraphBuilder
        composable( route: "register" ) { this: AnimatedContentScope it: NavBackStackEntry
            RegisterScreen(
                viewModel = authViewModel,
                onRegisterSuccess = {
                    navController.navigate( route: "login" )
                },
                navController = navController
            )
        }
    }
}
```

Slika 4.6. Početak funkcije *Navigation()*

Zaslon za prijavljivanje je vrlo sličan zaslonu za registraciju. Korisniku su predstavljena dva polja za unos te gumb označen s „Login“. Nakon unosa ispravnih podataka, pritiskom na gumb poziva se funkcija *loginUser()* iz *authViewModel*-a te koristi Firebase funkciju *signInWithEmailAndPassword()* za prijavljivanje korisnika. Na uspješnu provjeru podataka, korisnika se usmjerava na glavni zaslon za pregledavanje recepata.

4.4.2. Zaslon sa svim receptima

Recipes Screen je glavni zaslon na kojem korisnici mogu pregledavati recepte. Sastoji se od tri gumba koji vode na druge zaslone, polja za pretraživanje, padajućeg izbornika za filtriranje recepata te prikaza sažetih informacija o receptima. Pri samom vrhu zaslona se nalazi gumb za odjavu s računa.

Prvi prvom otvaranju ovog zaslona klasa *RecipeViewModel*, vidljiva na slici 4.7, koja predstavlja ViewModel komponentu mobilne arhitekture ove aplikacije, vrši dohvaćanje instance baze podataka te postavljanje svih drugih varijabli potrebnih za ispravan rad aplikacije.

```
class RecipeViewModel : ViewModel()
{
    private val db = Firebase.firestore
    private val storage = FirebaseStorage.getInstance()
    var recipeData = mutableStateListOf<Recipe>()

    private val currentUserUid: String?
        get() = FirebaseAuth.getInstance().currentUser?.uid
    init {
        fetchDatabaseData()
    }

    private fun fetchDatabaseData(){
        val uid = currentUserUid

        if (uid == null) {
            Log.e( tag: "RecipeViewModel", msg: "User not logged in")
            return
        }
        db.collection( collectionPath: "recipes" ) CollectionReference
            .whereEqualTo( field: "userId", uid ) Query
            .get() Task<QuerySnapshot>
            .addOnSuccessListener { result ->
                for ( data in result.documents){
                    val recipe = data.toObject(Recipe::class.java)
                    if(recipe != null){
                        recipe.id = data.id
                        recipeData.add(recipe)
                    }
                }
            }

        db.collection( collectionPath: "categories" ) CollectionReference
            .whereEqualTo( field: "userId", uid ) Query
            .get() Task<QuerySnapshot>
            .addOnSuccessListener { result ->
                categoryList.clear()
                for (document in result) {
                    val categoryName = document.getString( field: "name" )
                    if (categoryName != null) {
                        categoryList.add(categoryName)
                    }
                }
            }
            .addOnFailureListener { exception ->
                Log.e( tag: "RecipeViewModel", msg: "Error getting categories: ", exception)
            }
    }
}
```

Slika 4.7. Klasa *RecipeViewModel*

Na samom početku izvodi se funkcija *init()* koja dohvaća podatke iz baze podataka te sprema podatke o receptima u varijablu *recipeData* i podatke o kategorijama u *categoryList*. Pri dohvaćanju podataka, također se vrši filtriranje prema jedinstvenoj oznaci korisnika tako da svaki korisnik dobije samo one recepte koje je sam unio.

Na zaslonu se nalaze tri gumba pomoću kojih korisnik pristupa ostalim zaslonima. Označeni su s: „Add new recipe“ koji korisnika vodi na zaslon na kojem može dodati novi recept, „Add new category“ koji korisnika vodi na zaslon koji mu omogućava unos nove kategorije za recepte te „Generate meal plan“ koji korisnika usmjerava na zaslon na kojem može generirati plan recepata.

Na zaslonu je zatim moguće vidjeti polje za pretraživanje pomoću kojeg korisnik može pretraživati recepte prema nazivu.

Na slikama 4.8. i 4.9. nalazi se *SearchBar()* funkcija kojom je implementirano polje za pretraživanje. Koristi se *TextField* komponenta za unos teksta te se promatra varijabla *searchQuery()*. Na promjenu varijable *searchQuery* varijabla *showResults* mijenja se iz stanja „false“ u „true“ te time započinje provjera „if“ uvjeta koji je naveden na slici 4.14.

Ukoliko je uvjet ispunjen, stvara se *LazyColumn* komponenta koja omogućuje vertikalno pomicanje sadržaja, a njezin sadržaj čine podaci iz *matchingRecipes* varijable. Na slici 4.13. vidljivo je kako se sadržaj ove varijable postavlja filtriranjem kroz *recipeData*. Za svaki recept u *recipeData* provjerava se sadrži li njegov naslov niz znakova od kojih se sastoji *searchQuery* te se zatim naslovi rezultata koji odgovaraju pretrazi, odnosno sadrže *searchQuery* u svom naslovu, ispisuju ispod polja za pretragu. Nakon klika na rezultat korisnika se preusmjerava na zaslon koji sadrži detalje o receptu.

```

@Composable
fun SearchBar(
    viewModel: RecipeViewModel,
    navController: NavController
) {
    var searchQuery by remember { mutableStateOf( value: "" ) }
    var showResults by remember { mutableStateOf( value: false ) }

    val matchingRecipes = remember( searchQuery ) {
        viewModel.recipeData.filter { recipe ->
            recipe.title.contains( searchQuery, ignoreCase = true )
        }
    }

    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier.fillMaxWidth()
    )
    { this: ColumnScope
        TextField(
            value = searchQuery,
            onValueChange = { query ->
                searchQuery = query
                showResults = query.isNotEmpty()
            },
            label = { Text( text: "Search Recipes" ) },
            modifier = Modifier
                .fillMaxWidth()
                .padding( 16.dp )
                .height( 65.dp ),
            colors = TextFieldDefaults
                .textFieldColors(
                    focusedIndicatorColor = buttonBlue,
                    unfocusedIndicatorColor = Color.Gray,
                    containerColor = Color.Transparent,
                    focusedLabelColor = buttonBlue,
                    unfocusedLabelColor = Color.Gray
                ),
            textStyle = TextStyle( fontSize = 16.sp )
        )
    }
}

```

Slika 4.8. Funkcija *SearchBar()*

```

if ( showResults ) {
    LazyColumn(
        modifier = Modifier
            .fillMaxWidth()
            .height( 200.dp )
    ) { this: LazyListScope
        items( matchingRecipes ) { this: LazyItemScope recipe ->
            recipeSearchResultItem( recipe = recipe ) {
                navController.navigate( route: "${Routes.recipe_detail}/${recipe.id}" )
                showResults = false
            }
        }
    }
}
}

```

Slika 4.9. Nastavak funkcije *SearchBar()* – *if* uvjet

Neposredno ispod polja za pretraživanje nalazi se padajući izbornik koji omogućuje korisniku filtriranje recepata prema kategoriji. Padajući izbornik realiziran je funkcijom *DropDownMenu()* koja se nalazi na slikama 4.10. i 4.11. Korištene su komponente *ExposedDropDownMenuBox* i *TextField* za prikaz same kutije izbornika te *ExposedDropDownMenu* i *DropDownMenuItem* za prikaz sadržaja padajućeg izbornika. Kao što je vidljivo na slici 4.10. promatraju se dvije varijable; *selectedCategory* čija je uloga pamtitu odabranu kategoriju recepta koju je potrebno filtrirati iz svih recepata i *expanded* koja upravlja stanjem padajućeg izbornika, odnosno time je li otvoren ili zatvoren. Klikom na padajući izbornik, on se otvara mijenjanjem stanja varijable *expanded* iz „false“ u „true“ te se prikazuju sve moguće opcije. Na slici 4.11. vidi se kako se sadržaj izbornika filtrira iz popisa kategorija koji je inicijaliziran u prethodno pokazanom *RecipeViewModel*-u. Prikazuju se sve kategorije trenutnog korisnika te se klikom na jednu od kategorija zatvara padajući izbornik i poziva se funkcija *filterRecipesByCategory()*.

```

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun DropDownMenu(
    viewModel: RecipeViewModel,
) {
    var selectedCategory by remember { mutableStateOf<String>(value: "All") }
    var expanded by remember { mutableStateOf<Boolean>(value: false) }
    ExposedDropDownMenuBox(
        expanded = expanded,
        onExpandedChange = { expanded = !expanded }
    ) { this: ExposedDropDownMenuBoxScope
        TextField(
            value = selectedCategory,
            onValueChange = {},
            readOnly = true,
            label = { Text(text = "Category") },
            trailingIcon = { TrailingIcon(expanded = expanded) },
            modifier = Modifier
                .fillMaxWidth()
                .padding(8.dp)
                .menuAnchor()
                .clip(RoundedCornerShape(8.dp)),
            colors = TextFieldDefaults
                .textFieldColors(
                    focusedIndicatorColor = buttonBlue,
                    unfocusedIndicatorColor = Color.Gray,
                    containerColor = textFieldBlue,
                    focusedLabelColor = buttonBlue,
                    unfocusedLabelColor = Color.Gray
                )
        ),
    )
}

```

Slika 4.10. Funkcija *DropDownMenu()*

```

ExposedDropDownMenu(
  expanded = expanded,
  onDismissRequest = { expanded = false }
) { this: ColumnScope
  DropdownMenuItem(
    text = { Text(text: "All") },
    onClick = {
      selectedCategory = "All"
      expanded = false
      viewModel.filterRecipesByCategory(category: "All")
    }
  )

  viewModel.categoryList.forEach { category ->
    DropdownMenuItem(
      text = {Text(category)},
      onClick = {
        selectedCategory = category
        expanded = false
        viewModel.filterRecipesByCategory(selectedCategory)
      }
    )
  }
}
}
}

```

Slika 4.11. Nastavak funkcije *DropDownMenu()*

Funkcija *filterRecipesByCategory()* prikazana slikom 4.12. uzima predanu kategoriju te provjerava o kojoj je kategoriji riječ. Filtrirani recepti će se spremi kao popis recepata u varijablu *filteredRecipes*. Kategorija „All“ je kategorija koja se nalazi na računu svakog korisnika kako bi se mogli prikazati svi dostupni recepti te ukoliko je odabrana ta kategorija, u varijablu *filteredRecipes* se spremaju svi postojeći recepti. Ukoliko je odabrana neka druga kategorija iz svih recepata se filtriraju oni koji pripadaju tom korisniku i čija je kategorija jednaka odabranoj.

```

fun filterRecipesByCategory(category: String) {
  val currentUserId = FirebaseAuth.getInstance().currentUser?.uid

  filteredRecipes.clear()
  if (category == "All") {
    filteredRecipes.addAll(recipeData)
  } else {
    filteredRecipes.addAll(recipeData.filter { it.category == category && it.userId == currentUserId })
  }
}

```

Slika 4.12. Funkcija *filterRecipesByCategory()*

Preostali dio zaslona koristi se za prikaz sažetih informacija o receptima pomoću kartica na kojima se nalaze slika recepta, naslov te vrijeme pripreme, ocjena i količina kalorija. Kartice su raspoređene u dva stupca te je na zaslonu omogućeno vertikalno pomicanje. Prikazani

sadržaj se uzima iz prethodno navedene varijable *filteredRecipes*. Klikom na karticu recepta korisnika se usmjerava na zaslon s detaljima o receptu.

4.4.3. Dodavanje novog recepta i kategorije

Kao što je prethodno navedeno, klikom na gumb „Add new recipe“ korisnika se usmjerava na zaslon za dodavanje novog recepta. Na samom vrhu ovog zaslona nalazi se gumb za povratak na prethodni zaslon. Na zaslonu su potom vidljiva polja za unos informacija o novom receptu, a definirana su unutar funkcije *RecipeInputScreen()*. Za sve informacije su definirane varijable u koje će se one pohraniti te su još definirane i varijable potrebne za spremanje slike i za upravljanje padajućim izbornikom. (Slika 4.13.)

```
var ingredient by remember { mutableStateOf( value: "" ) }
var recipeName by remember { mutableStateOf( value: "" ) }
var cookingTime by remember { mutableStateOf( value: "" ) }
var calories by remember { mutableStateOf( value: "" ) }
var rating by remember { mutableStateOf( value: "" ) }
var instruction by remember { mutableStateOf( value: "" ) }

var selectedImageUri by remember { mutableStateOf<Uri?>( value: null ) }
var imageBitmap by remember { mutableStateOf<ImageBitmap?>( value: null ) }
var expanded by remember { mutableStateOf( value: false ) }
var selectedCategory by remember { mutableStateOf<String?>( value: null ) }
```

Slika 4.13. Prikaz definiranih varijabli za rad funkcije *RecipeInputScreen()*

Polja za unos naslova recepta, kalorija, ocjene i vremena za pripremu su implementirana pomoću komponente *TextField*. Na slici 4.14. prikazan je primjer implementacije za naslov recepta. Kao vrijednost se postavlja varijabla *recipeName* koja predstavlja naslov. Njena početna vrijednost, kao i vrijednost ostalih polja implementiranih na ovaj način, je postavljena na prazan znakovni niz te se njena vrijednost mijenja na unos teksta od strane korisnika.

```
TextField(
    value = recipeName,
    onChange = { recipeName = it },
    label = { Text( text: "Recipe Title" ) },
    modifier = Modifier.fillMaxWidth(),
    colors = TextFieldDefaults
        .textFieldColors(
            focusedIndicatorColor = buttonBlue,
            unfocusedIndicatorColor = Color.Gray,
            containerColor = textFieldBlue,
            focusedLabelColor = buttonBlue,
            unfocusedLabelColor = Color.Gray
        )
)
```

Slika 4.14. Prikaz implementacije polja za unos naslova recepta

Nakon ovih polja slijedi padajući izbornik za odabir kategorije recepta koji radi na isti način kao i prethodno objašnjeni padajući izbornik za filtriranje kategorija. Jedina razlika je što klikom na kategoriju u ovom izborniku se samo vrijednost varijable *selectedCategory* postavlja na odabranu vrijednost te se ne događa ništa nakon toga. Zatim slijede dva tekstualna polja za unos sastojaka i koraka. U *Recipe* modelu su ove varijable kao popisi tekstualnih nizova. Unutar funkcije *RecipeInputScreen()* definirane su dvije varijable koje služe kao privremeni spremnici za svaki pojedini sastojak odnosno korak pripreme. Ova dva polja su implementirana pomoću *BasicTextField* komponente prikazanoj na slici 4.15. Pomoću postavki *keyboardOptions* i *keyboardActions* je moguće gumb za novi red na tipkovnici promijeniti u gumb za potvrdu. Time se omogućuje dodavanje napisanog teksta u popis sastojaka odnosno koraka pripreme.

```

BasicTextField(
  value = ingredient,
  onChange = { ingredient = it },
  modifier = Modifier
    .fillMaxWidth()
    .padding(8.dp),
  keyboardOptions = KeyboardOptions.Default.copy(imeAction = ImeAction.Done),
  keyboardActions = KeyboardActions(
    onDone = { this: KeyboardActionScope
      if (ingredient.isNotEmpty()) {
        viewModel.addIngredient(ingredient)
        ingredient = ""
      }
    }
  ),
  decorationBox = { innerTextField ->
    Box(
      Modifier
        .background(MaterialTheme.colorScheme.surface)
        .padding(8.dp)
    ) { this: BoxScope
      if (ingredient.isEmpty()) {
        Text(text = "Enter Ingredient")
      }
      innerTextField()
    }
  }
)

```

Slika 4.15. Komponenta *BasicTextField* funkcije *RecipeInputScreen()*

Pritiskom gumba za potvrdu na tipkovnici poziva se funkcija *addIngredient()* prikazana slikom koja dodaje vrijednost privremene varijable *ingredient* popisu sastojaka definiranom u *ViewModel*-u *ingredientList*. Nakon dodavanja sastojka, on se prikazuje na dnu zaslona s ikonom za uklanjanje pored zapisa. Klikom na ikonu za uklanjanje poziva se funkcija *removeIngredients()* te se taj sastojak uklanja iz popisa sastojaka. Funkcije s istim radnjama definirane su i za popis koraka pripreme.

Na zaslonu se zatim nalazi gumb za odabir slike recepta označen sa „Select Image“. Odabir slike moguć je zbog postavljene varijable *imagePickerLauncher* vidljive na slici 4.16. *Image Picker* je alat koji omogućava korisniku da odabere sliku iz uređaja te se u ovom slučaju na njegov poziv otvara sučelje koje prikazuje sve dostupne slike na uređaju. Funkcija *rememberLauncherForActivityResult()* zajedno s *ActivityResultContracts.GetContent()* definira radnju dohvaćanja sadržaja odnosno u ovom slučaju jedne datoteke koja će predstavljati recept.

Nakon odabira slike izvršava se *onResult()* funkcija koja postavlja vrijednost varijable *selectedImageUri* na jedinstveni identifikator resursa (eng. Unique Resource Identifier, URI). Varijabla *inputStream* omogućuje aplikaciji da čita slikovnu datoteku te se pomoću *BitmapFactory.decodeStream()* funkcije tok podataka slike pretvara u Bitmap format koji je lakše čitljiv format slike. Na posljetku se taj zapis pretvara u ImageBitmap format koji Jetpack Compose koristi za prikazivanje slika.

```
val context = LocalContext.current
val imagePickerLauncher = rememberLauncherForActivityResult(
    contract = ActivityResultContracts.GetContent(),
    onResult = { uri ->
        selectedImageUri = uri
        uri?.let { it: Uri
            val inputStream: InputStream? = context.contentResolver.openInputStream(it)
            imageBitmap = inputStream?.use { stream ->
                android.graphics.BitmapFactory.decodeStream(stream).asImageBitmap()
            }
        }
    }
)
```

Slika 4.16. Prikaz implementacije sučelja za izbor slika

Pritisak na gumb pokreće opisani proces odabira slike, a nakon odabira, slika se prikazuje ispod gumba. Prikaz slike implementiran je pomoću komponente *Image* i prethodno postavljene varijable *bitmap* koja sadrži podatke o slici. (Slika 4.17.)

```
Button(  
    onClick = { imagePickerLauncher.launch( input: "image/*") },  
    colors = ButtonDefaults.filledTonalButtonColors(  
        containerColor = buttonBlue,  
        contentColor = Color.White  
    )  
) { this: RowScope  
    Text( text: "Select Image")  
}  
  
Spacer(modifier = Modifier.height(8.dp))  
  
selectedImageUri?.let { it: Uri  
    imageBitmap?.let { bitmap ->  
        Image(  
            bitmap = bitmap,  
            contentDescription = "Selected Recipe Image",  
            contentScale = ContentScale.Crop,  
            modifier = Modifier  
                .fillMaxWidth()  
                .height(200.dp)  
                .background(Color.Gray)  
        )  
    }  
}
```

Slika 4.17. Implementacija gumba za odabir slike i prikaza odabrane slike

Spremanje napisanog recepta pokreće se pritiskom gumba „Save Recipe“. Klikom na gumb provjerava se jesu li sva polja popunjena te se poziva funkcija *saveRecipe()* prikazana slikom 4.18. Funkcija *saveRecipe()* kao argumente prima podatke o naslovu recepta, kalorijama, vremenu pripreme, sastojcima, ocjeni, kategoriji i slici recepta te funkcije u slučaju uspješnog odnosno neuspješnog spremanja recepta. Kreira se mapa sa svim tim podacima te se oni spremaju u bazu podataka u kolekciju *recipes*. Ukoliko dođe do pogreške funkcija vraća poruku s informacijama o pogrešci.

```

fun saveRecipe(
    recipeName: String,
    cookingTime: String,
    calories: String,
    rating: String,
    imageUrl: String,
    category: String,
    onSuccess: () -> Unit,
    onFailure: (Exception) -> Unit
) {
    val userId = FirebaseAuth.getInstance().currentUser?.uid ?: return
    viewModelScope.launch { this: CoroutineScope
        try {
            if (imageUrl.isNullOrEmpty()) {
                throw IllegalArgumentException("Image URL must not be empty")
            }

            val recipeData = hashMapOf(
                "title" to recipeName,
                "cookingTime" to cookingTime,
                "kcal" to calories,
                "rating" to rating,
                "image" to imageUrl,
                "category" to category,
                "ingredients" to ingredients,
                "instructions" to instructions,
                "userId" to userId
            )
            db.collection(collectionPath: "recipes").add(recipeData).await()
            onSuccess()
        } catch (e: Exception) {
            onFailure(e)
        }
    }
}

```

Slika 4.18. Funkcija *saveRecipe()*

Zaslону za dodavanje kategorije pristupa se putem gumba „Add new category“ sa zaslona sa svim receptima. Na vrhu zaslona nalazi se prethodno objašnjena funkcija *TopBar()*, a ispod nje nalazi se polje za unos teksta gdje korisnik može unijeti ime nove kategorije. Polje za unos teksta implementirano je na isti način kao i polja za unos teksta na zaslonu za dodavanje novog recepta. Spremanje nove kategorije potvrđuje se pritiskom gumba „Save Category“ koji na pritisak poziva funkciju *addCategory()*. Ova je funkcija slična funkciji *saveRecipe()* samo ima manje argumenata. Stvara se mapa podataka od imena i jedinstvene oznake korisnika te se sprema u bazu podataka unutar kolekcije *categories*.

4.4.4. Generiranje tjednog plana ishrane

Zaslону za dodavanje kategorije pristupa se putem gumba „Generate meal plan“ sa zaslona sa svim receptima. Na vrhu zaslona nalazi se prethodno objašnjena funkcija *TopBar()*, a ispod nje se nalazi 5 padajućih izbornika pomoću kojih korisnik bira iz kojih kategorija želi

generirati recepte. Padajući izbornici implementirani su na isti način kao padajući izbornik iz funkcije *DropDownMenu()* na slici 4.10. Za svaki izbornik potrebno je definirati posebnu varijablu koja kontrolira njegovo stanje kako se izbornici ne bi istovremeno otvarali i zatvarali. Korisnik može odabrati jednu ili svih 5 kategorija, za svaku odabranu kategoriju se odabire jedan recept. Proces generiranja nasumičnih recepata definiran je unutar funkcije *generateMealPlan()* na slici 4.19. Svaki padajući izbornik postavlja stanje odgovarajuće varijable *selectedCategory* te se zatim stvara lista filtriranih recepata za svaku kategoriju koristeći metodu *filter*, koja vraća samo recepte koji pripadaju odgovarajućoj kategoriji. Zatim, za svaku od filtriranih listi, bira se nasumični recept, ako lista nije prazna, i dodaje se u plan obroka. Na kraju, *mealPlan* sadrži nasumične recepte iz odabranih kategorija, jedan recept iz svake kategorije koja ima barem jedan recept.

```
var selectedCategory1 by mutableStateOf<String?>( value: null)
var selectedCategory2 by mutableStateOf<String?>( value: null)
var selectedCategory3 by mutableStateOf<String?>( value: null)
var selectedCategory4 by mutableStateOf<String?>( value: null)
var selectedCategory5 by mutableStateOf<String?>( value: null)

var mealPlan by mutableStateOf<List<Recipe>>(emptyList())

fun generateMealPlan() {

    val filteredRecipes = listOf(
        recipeData.filter { it.category == selectedCategory1 },
        recipeData.filter { it.category == selectedCategory2 },
        recipeData.filter { it.category == selectedCategory3 },
        recipeData.filter { it.category == selectedCategory4 },
        recipeData.filter { it.category == selectedCategory5 }
    )

    mealPlan = filteredRecipes.mapNotNull { recipes ->
        if (recipes.isNotEmpty()) recipes.random() else null
    }
}
```

Slika 4.19. Funkcija *generateMealPlan()*

4.4.5. Detaljan prikaz recepta

Zaslону na kojem se nalaze detaljne informacije o odabranom receptu pristupa se klikom na karticu recepta ili na rezultat pretrage nakon korištenja polja pretraživanja. Podatak o kojem se točno receptu radi se predaje *composable()* funkciji unutar funkcije *Navigation()* u obliku jedinstvene oznake recepta. (Slika 4.20.)

```

composable(
    route = "${Routes.recipe_detail}/{recipeId}",
    arguments = listOf(
        navArgument( name: "recipeId" ) { this: NavArgumentBuilder
            type = NavType.StringType
        }
    )
) { this: AnimatedContentScope, backStackEntry ->
    val recipeId = backStackEntry.arguments?.getString( key: "recipeId" )
    recipeId?.let { it: String
        RecipeDetailScreen(
            navController = navController,
            viewModel = viewModel,
            recipeId = it
        )
    }
}
}

```

Slika 4.20. *Composable()* funkcija za navigaciju do zaslona s detaljima o receptu

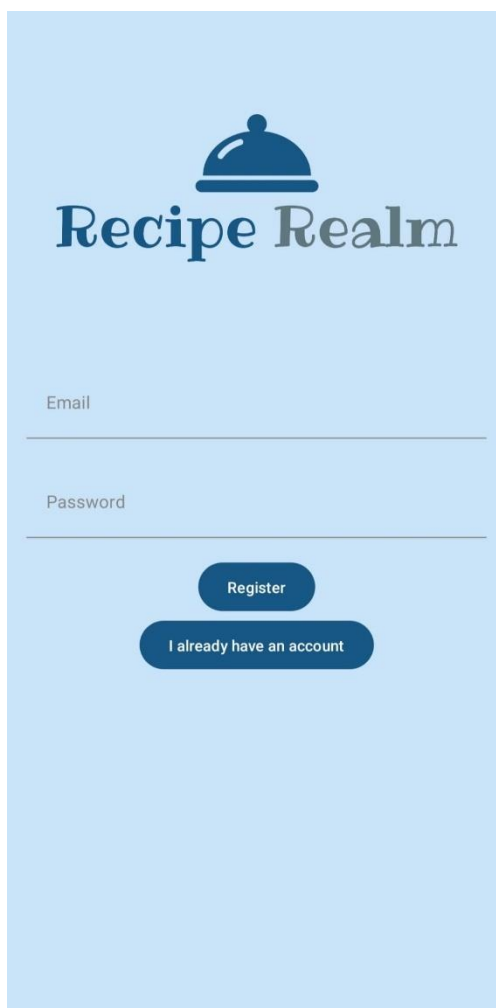
Na zaslonu je prikazana slika recepta koju je korisnik odabrao sa svog uređaja te se na slici nalazi naslov recepta. Ostatak zaslona je iskorišten za prikaz popisa sastojaka i koraka pripreme.

5. KORIŠTENJE APLIKACIJE

Aplikacija za personaliziranu pohranu recepata ima nekoliko različitih zaslona i funkcionalnosti. Korisnici najprije moraju registrirati račun kako bi mogli koristiti aplikaciju. Nakon registracije omogućen im je unos vlastitih recepata i njihova kategorizacija. Također su im dostupne i druge funkcionalnosti poput filtriranja recepata prema kategoriji i generiranja plana prehrane. U ovom će poglavlju biti predstavljene sve funkcionalnosti aplikacije i njihovo korištenje.

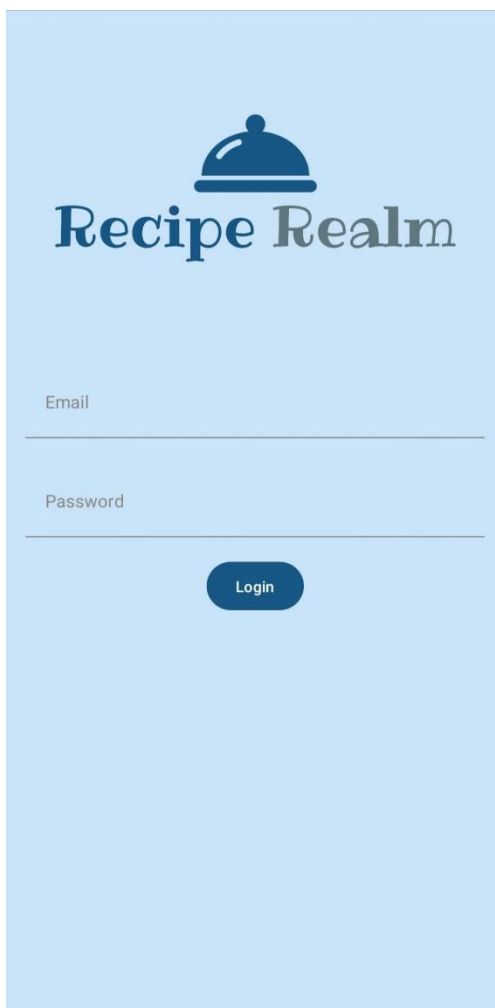
5.1. Zaslone za registraciju i prijavu

Prvi zaslon koji se otvara korisniku kada pokrene aplikaciju je zaslon za registraciju. Na slici 5.1 prikazan je izgled zaslona za registraciju. Korisnik u predviđena polja unosi adresu svoje elektroničke pošte i željenu lozinku te klikom na gumb „Register“ registrira račun. Ukoliko već postoji račun s danom adresom, aplikacija informira korisnika.



Slika 5.1. Zaslon za registraciju

Nakon uspješne registracije korisnika se usmjerava na zaslon za prijavu. Ukoliko korisnik već ima račun, može pritisnuti gumb „I already have an account“ i odmah biti usmjeren na odgovarajući zaslon. Prikaz zaslona za prijavu nalazi se na slici 5.2. Kao i na zaslonu za registraciju, korisnik unosi adresu elektroničke pošte svog računa te lozinku. Aplikacija će porukom obavijestiti korisnika ukoliko unese krivu lozinku.



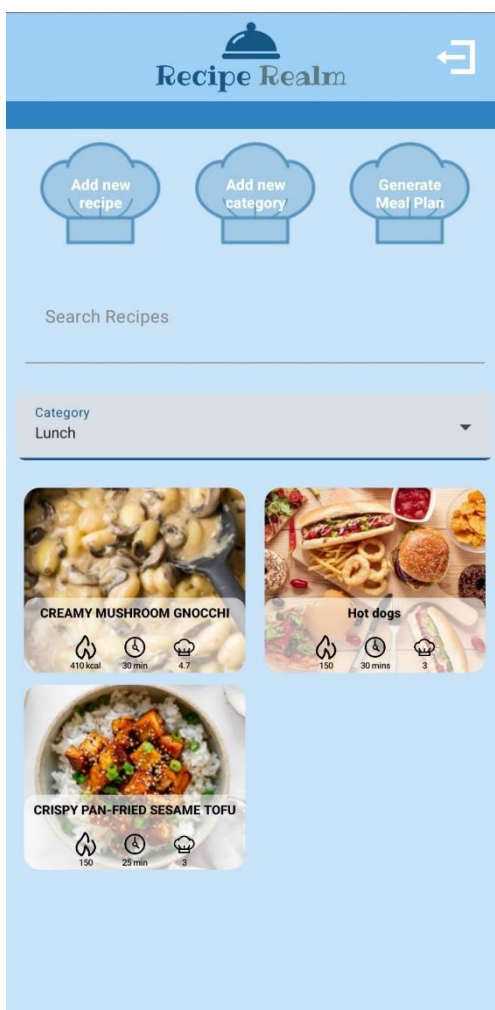
Slika 5.2. Zaslon za prijavu

Pritiskom na gumb „Login“ izvodi se proces prijave i korisnika se prosljeđuje na glavni zaslon aplikacije sa svim receptima.

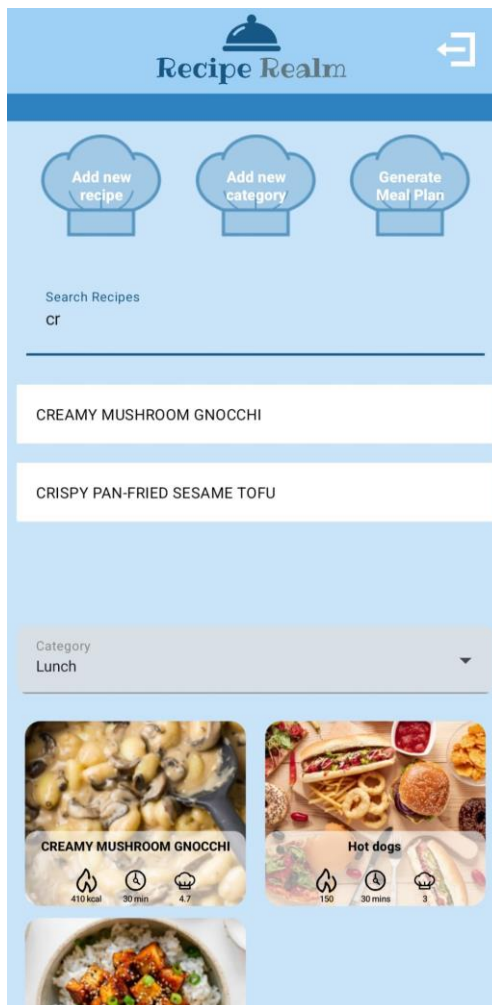
5.2. Zaslon za pregled svih recepata

Na slici 5.3. prikazan je glavni zaslon aplikacije. Na njemu se nalaze 3 gumba koja vode do drugih zaslona. Pritiskom na prvi gumb korisnik odlazi na zaslon za dodavanje novog recepta, pritiskom na drugi gumb se prikazuje zaslon za dodavanje nove kategorije, a posljednji gumb

vodi korisnika na zaslon za generiranje plana prehrane. Na ovom je zaslonu također moguće vidjeti sažete informacije o svim receptima u obliku informacijskih kartica na kojima se nalaze slike recepata i osnovne informacije poput naziva, vremena pripreme, kalorija i ocjene. Recepte je moguće filtrirati pomoću padajućeg izbornika odabirom kategorije čije se recepte želi prikazati. Izgled rezultata pretrage prikazan je slikom 5.4. Ukoliko je teško pronaći neki određeni recept unutar kategorije, moguće ga je potražiti putem polja za pretragu po imenu recepta. U gornjem desnom kutu zaslona nalazi se i gumb za odjavu s računa.



Slika 5.3. Zaslon sa svim receptima

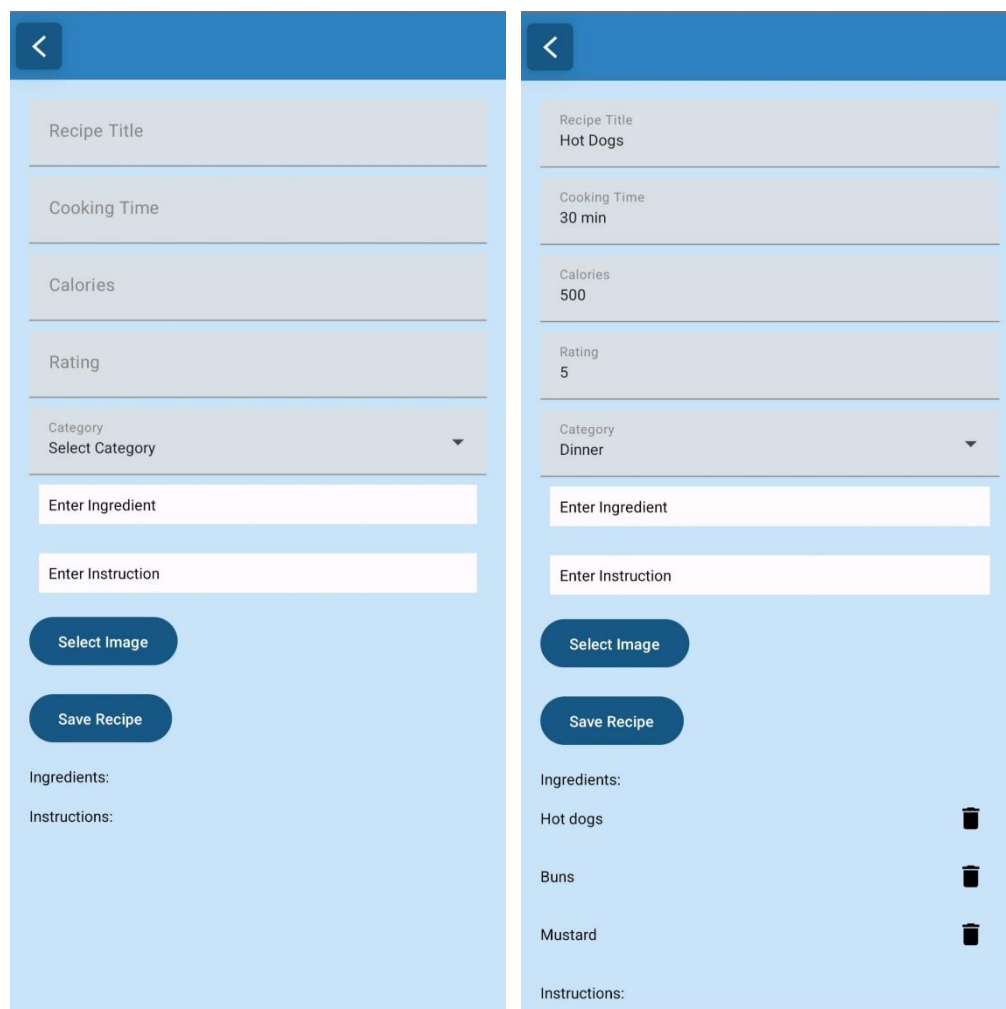


Slika 5.4. Prikaz rezultata pretraživanja

5.3. Zaslone za dodavanje novog recepta i kategorije

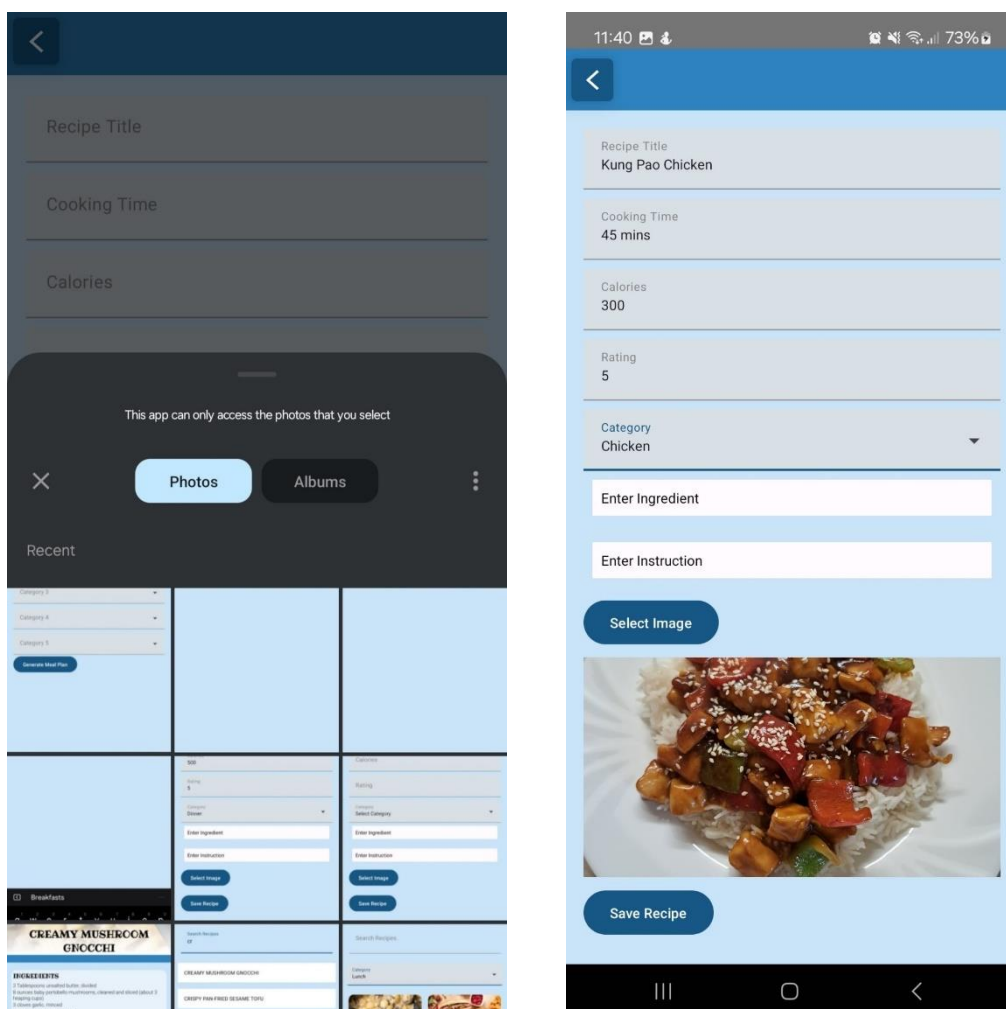
Sučelje zaslona za dodavanje novog recepta prije i nakon unosa informacija prikazan je slikom 5.5. Na zaslonu se nalaze 4 polja za unos teksta u koja korisnik unosi informacije o naslovu recepta, vremenu pripreme, kalorijama i ocjeni. Kako bi recept bio pridružen određenoj kategoriji, potrebno je istu odabrati pomoću padajućeg izbornika. Potom je potrebno unijeti sastojke i korake pripreme čiji unos je omogućen poljem za unos teksta te se svaki korak ili

sastojak prema pritiskom gumba za potvrdu na tipkovnici. Prilikom unosa sastojaka ili koraka, moguće je iste ukloniti prije spremanja cijelog recepta pritiskom ikone za uklanjanje.



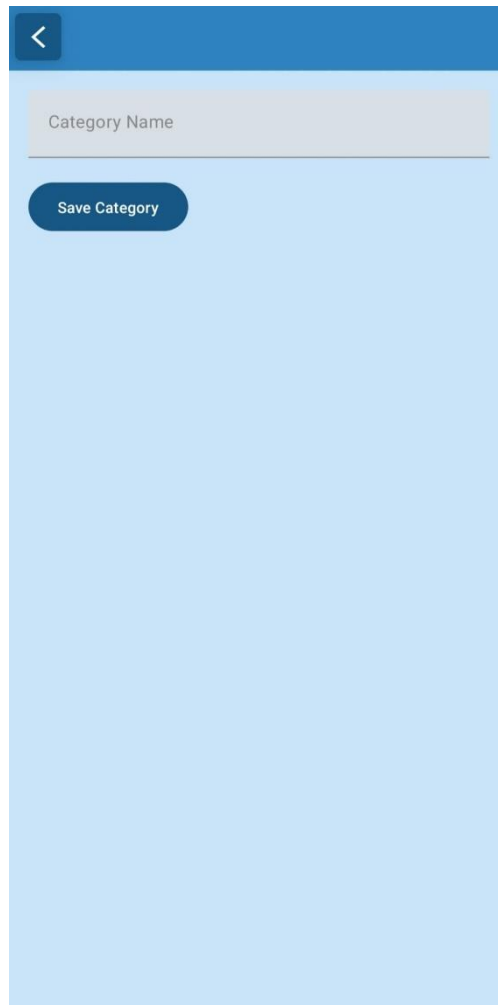
Slika 5.5. Sučelje zaslona za dodavanje novog recepta

Sučelje za odabir fotografije koja će se koristiti za recept te izgled zaslona nakon odabira slike moguće je vidjeti na slici 5.6. Sučelje za odabir fotografije se otvara pritiskom na gumb „Select Image“ te fotografiju korisnik bira iz skupa svih slikovnih datoteka na svom uređaju. U slučaju odabira krive slike, odabir je moguće izmijeniti ponovnim pritiskom na gumb „Select Image“.



Slika 5.6. Prikaz sučelja za odabir slike i rezultat odabira

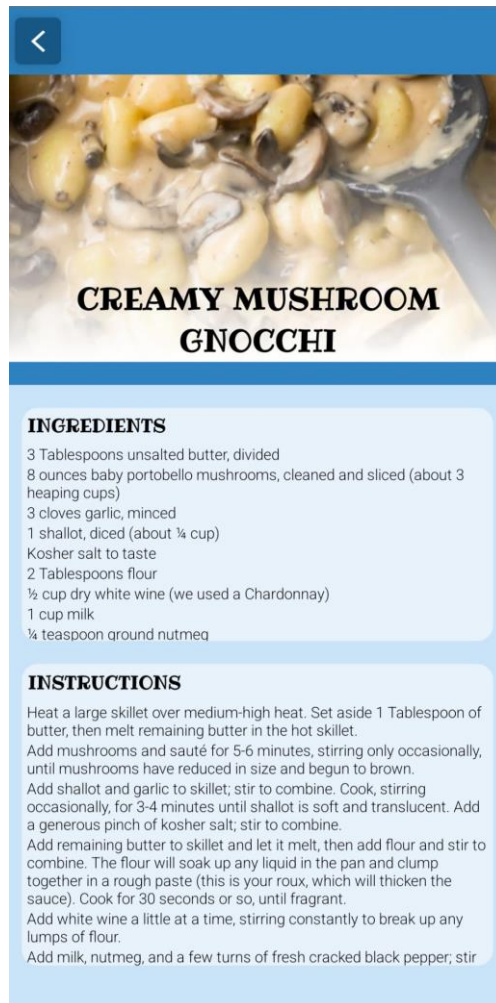
Zaslon za odabir kategorije prikazan je na slici 5.7. Na zaslonu se nalazi jedno polje za upis teksta u koji korisnik unosi ime nove kategorije. Pritiskom na gumb „Save Category“ kategorija se sprema u bazu podataka unutar kolekcije *categories*. Nakon spremanja kategorije, moguće je pohranjivati nove recepte u nju i filtrirati recepte iz te kategorije na prethodnim zaslonima.



Slika 5.7. Zaslون za dodavanje nove kategorije

5.4. Zaslون za detaljan prikaz recepta

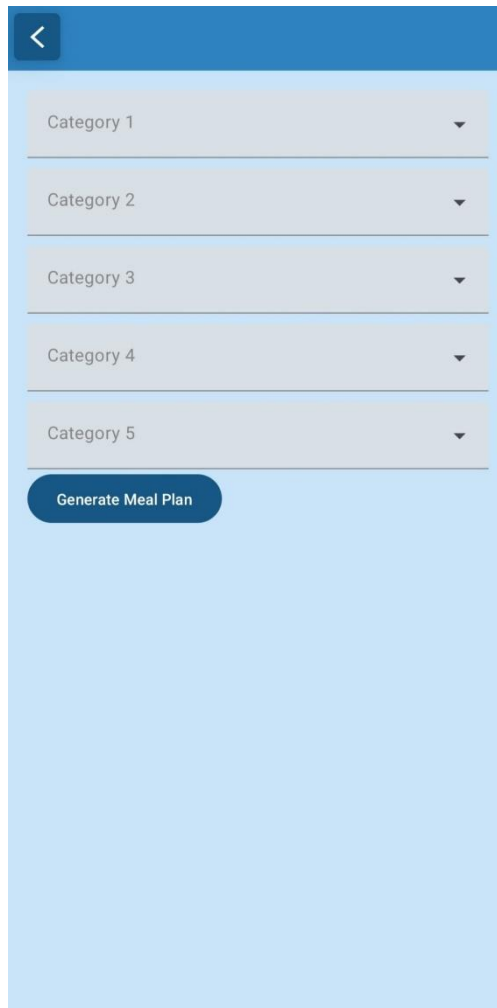
Klik na karticu recepta ili rezultat prehrane vodi korisnika na zaslون čiji je primjer prikazan na slici 5.8. Na zaslONU se nalazi naslov recepta te u vizualno odvojenim područjima se nalaze popis sastojaka i popis koraka pripreme. Oba je područja moguće vertikalno pomicati.



Slika 5.8. Zaslón s detaljnim informacijama o receptu

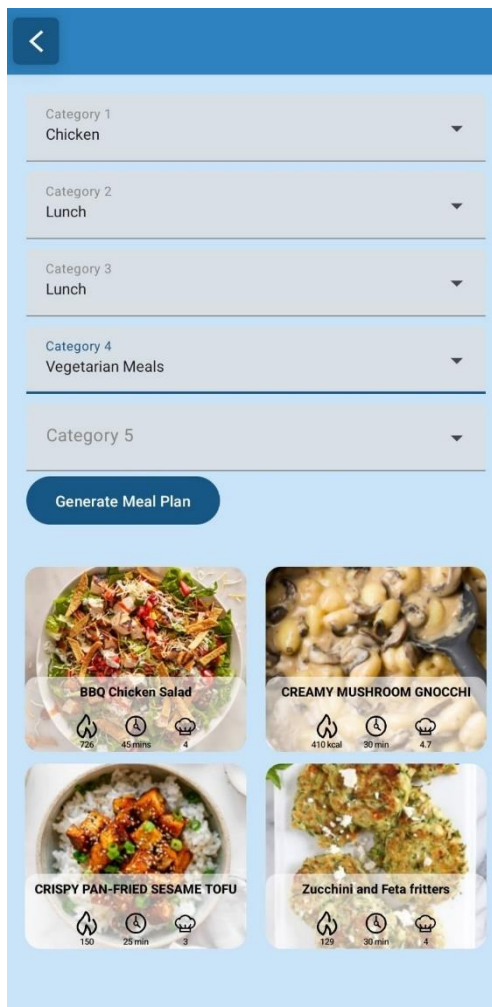
5.5. Zaslón za generiranje plana prehrane

Sučelje za zaslón za generiranje plana prehrane nalazi se na slici 5.9. Pomoću pet prikazanih padajućih izbornika korisnik bira koje kategorije recepata želi uključiti u generirani plan. Za svaku odabranu kategoriju se bira jedan recept. Korisnik može odabrati istu kategoriju više puta te će tako dobiti više recepata iz iste kategorije. Nije potrebno odabrati ni svih pet kategorija, broj odabranih kategorija određuje broj recepata koji će se generirati.



Slika 5.9. Zaslona za generiranje plana ishrane

Pritiskom gumba „Generate Meal Plan“ započinje proces generiranja plana ishrane te se isti prikazuje ispod gumba u obliku istih kartica koje se nalaze na glavnom zaslonu poredanih u dva stupca. Sučelje zaslona nakon generiranja plana nalazi se na slici 5.10.



Slika 5.10. Sučelje zaslona nakon generiranja plana ishrane

5.6. Vrednovanje funkcionalnosti aplikacije

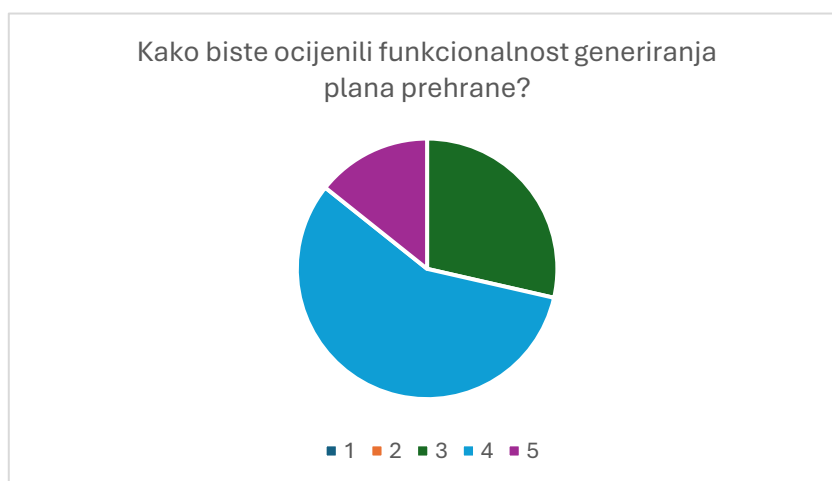
U svrhu vrednovanja funkcionalnosti aplikacije provedena je anketa s ukupno 10 ispitanika. Ispitanici su isprobali aplikaciju te su im nakon korištenja postavljena sljedeća pitanja:

- Kako biste ocijenili jednostavnost korištenja aplikacije na skali od 1 do 5
- Kako biste ocijenili funkcionalnost generiranja plana prehrane?
- Kako biste ocijenili preglednost prikaza recepata?
- Imate li prijedloge za poboljšanje aplikacije?
- Je li vam aplikacija omogućila jednostavno dodavanje i upravljanje receptima?

Slike 5.11. i 5.12. prikazuju rezultate odgovora na prva dva pitanja. Korisnici su uvelike zadovoljni jednostavnošću korištenja aplikacije te im se svidjela mogućnost generiranja plana prehrane.



Slika 5.11. Dijagram odgovora na prvo pitanje ankete



Slika 5.12. Dijagram odgovora na drugo pitanje ankete

Svi odgovori na treće pitanje bili su ocijenjeni s 5 te su svi ispitanici odgovorili s „Da“ na posljednje pitanje. Ispitanici su prilikom odgovaranja na četvrto pitanje uz nedostatke istaknuli iznimno zadovoljstvo s korisničkim sučeljem. Nekoliko ispitanika je potvrdilo kako su sve funkcionalnosti jasno prikazane i jednostavne za korištenje. Na temelju provedenog korisničkog ispitivanja zaključeno je da aplikacija ispunjava sve ključne funkcionalnosti bez značajnih grešaka. Ispitanici su istaknuli želju za mogućnošću spremanja generiranih planova prehrane za buduće korištenje te za dodavanjem više kriterija prema kojima se generira plan. Unatoč ovim prijedlozima, opći dojam aplikacije je pozitivan te su svi korisnici zadovoljni s dodavanjem i pretraživanjem recepata te jednostavnošću korištenja aplikacije.

6. ZAKLJUČAK

Kuhanje je, neovisno o tome predstavlja li užitek ili samo neophodnu radnju, i dalje svakodnevna aktivnost za većinu ljudi. Neki se služe intuicijom, no mnogi se okreću gotovim receptima koji ih lagano mogu dovesti do željenog rezultata. Problem se pojavljuje u spremanju i održavanju recepata. Listanje gomile kuharica u potrazi za jednim receptom postaje zamorno i nepraktično. Pojavljuje se potreba za načinom očuvanja recepata koji ne zauzima puno prostora, omogućuje brzu i jednostavnu pretragu te daje jednake rezultate bez obzira na prolaznost vremena. U pomoć ovom problemu dolaze mrežne stranice i mobilne aplikacije čija je svrha očuvanje i lagana pretraga recepata. Na tržištu već postoje rješenja poput aplikacija Tasty, Side Chef i Recipe Keeper koje omogućuju unos, spremanje i pretragu recepata. Cilj razvoja ove aplikacije je nadopuniti nedostatke navedenih aplikacija omogućujući korisnicima opcije za unos više informacija o svojim receptima te eliminiranje potrebe za odlučivanjem o budućim jelima uvođenjem nasumične generacije plana prehrane.

Aplikacija razvijena u ovom radu ispunjava sve prethodno navedene zahtjeve. Omogućeni su jednostavan i brz unos, pretraga i pregled recepata te je time svakom korisniku omogućeno stvaranje vlastite personalizirane baze recepata. Jednostavna je za korištenje i prikladna za bilo koga tko stvara recepte ili želi stvoriti svoju jedinstvenu biblioteku recepata.

Za izradu aplikacije korišteno je razvojno okruženje Android Studio te je korišten programski jezik Kotlin. Za izradu korisničkog sučelja korišten je deklarativni okvir Jetpack Compose. Korisničko sučelje je jednostavno i intuitivno za korištenje. Funkcionalnost aplikacije je testirana i prikazana.

Provedeno je korisničko ispitivanje kojim je zaključeno da aplikacija ispunjava potrebe korisnika te je utvrđeno opće zadovoljstvo korisničkim sučeljem i jednostavnošću korištenja. Ispitivanjem je također zaključeno da postoji prostor za poboljšanje aplikacije proširenjem funkcionalnosti generiranja plana dodavanjem različitih kriterija za generaciju te dodavanjem mogućnosti spremanja plana.

Ovim radom razvijena je Android aplikacija za rješavanje svakodnevnog problema kuhanja olakšavanjem pohrane i pregleda recepata. Omogućen je jednostavan unos recepata te generiranje plana prehrane. Na temelju povratne informacije ispitanih korisnika zaključeno je da je cilj aplikacije uspješno ostvaren uz prostor za poboljšanje.

LITERATURA

- [1] Recipe Keeper, Turdospan Software, Oxfordshire, 2024, dostupno na: <https://recipekeeperonline.com/> , [Pristupljeno 24.8.2024.]
- [2] Side Chef, SideChef Inc., 2024, dostupno na: <https://www.sidechef.com/> [Pristupljeno 24.8.2024.]
- [3] Tasty, BuzzFeed Inc., 2024, dostupno na: <https://tasty.co/> [Pristupljeno 24.8.2024.]
- [4] Meet Android Studio, Google for Developers, 2024, dostupno na: <https://developer.android.com/studio/intro> [Pristupljeno 24.8.2024.]
- [5] Kotlin, JetBrains, 2024, dostupno na: <https://kotlinlang.org/> [Pristupljeno 24.8.2024.]
- [6] Jetpack Compose, Google for Developers, 2024, dostupno na: <https://developer.android.com/compose> [Pristupljeno 24.8.2024.]
- [7] Firebase, Google for Developers, 2024, dostupno na: <https://firebase.google.com/> [Pristupljeno 24.8.2024.]
- [8] Figma, Figma, 2024, dostupno na: <https://www.figma.com/about/> [Pristupljeno 24.8.2024.]
- [9] O. Cem Işık, Introduction to MVVM Architecture, Medium, 2023, dostupno na: <https://medium.com/@onurcem.isik/introduction-to-mvvm-architecture-5c5558c3679> [Pristupljeno 24.8.2024.]

POPIS SLIKA

Slika 2.1. Prikaz izgleda aplikacije <i>Recipe Keeper</i>	2
Slika 2.2. Prikaz izgleda aplikacije <i>Side Chef</i>	3
Slika 2.3. Prikaz izgleda aplikacije <i>Tasty</i>	4
Slika 4.1. Prikaz sučelja Firebase baze podataka.....	8
Slika 4.2. Klasa <i>Recipe</i>	9
Slika 4.3. Klasa <i>Category</i>	9
Slika 4.4. Prikaz sučelja Firebase autentifikacijskog alata.....	10
Slika 4.5. Klasa <i>AuthViewModel</i>	11
Slika 4.6. Početak funkcije <i>Navigation</i>	12
Slika 4.7. Klasa <i>RecipeViewModel</i>	13
Slika 4.8. Funkcija <i>SearchBar()</i>	15
Slika 4.9. Nastavak funkcije <i>SearchBar()</i> – if uvjet.....	15
Slika 4.10. Funkcija <i>DropDownMenu()</i>	16
Slika 4.11. Nastavak funkcije <i>DropDownMenu()</i>	17
Slika 4.12. Funkcija <i>filterRecipesByCategory()</i>	17
Slika 4.13. Prikaz definiranih varijabli za rad funkcije <i>RecipeInputScreen()</i>	18
Slika 4.14. Prikaz implementacije polja za unos naslova recepta.....	18
Slika 4.15. Komponenta <i>BasicTextField</i> funkcije <i>RecipeInputScreen()</i>	19
Slika 4.16. Prikaz implementacije sučelja za izbor slika.....	20
Slika 4.17. Implementacija gumba za odabir slike i prikaza odabrane slike.....	21
Slika 4.18. Funkcija <i>saveRecipe()</i>	22
Slika 4.19. Funkcija <i>generateMealPlan()</i>	23
Slika 4.20. <i>Composable()</i> funkcija za navigaciju do zaslona s detaljima o receptu.....	24

Slika 5.1. Zaslona za registraciju.....	25
Slika 5.2. Zaslona za prijavu.....	26
Slika 5.3. Zaslona sa svim receptima.....	27
Slika 5.4. Prikaz rezultata pretraživanja.....	28
Slika 5.5. Sučelje zaslona za dodavanje novog recepta.....	29
Slika 5.6. Prikaz sučelja za odabir slike i rezultat odabira.....	30
Slika 5.7. Zaslona za dodavanje nove kategorije.....	31
Slika 5.8. Zaslona s detaljnim informacijama o receptu.....	32
Slika 5.9. Zaslona za generiranje plana ishrane.....	33
Slika 5.10. Sučelje zaslona nakon generiranja plana ishrane.....	34
Slika 5.11. Dijagram odgovora na prvo pitanje ankete.....	35
Slika 5.12. Dijagram odgovora na drugo pitanje ankete.....	35

SAŽETAK

S obzirom na sve brži tempo života, mobilne aplikacije pomažu ljudima u snalaženju u užurbanosti života, a u njih se ubrajaju i aplikacije za pohranu recepata koje čine proces kuhanja bezbolnijim za one koji ga ne vole i ugodnijim za one koji već u njemu uživaju. Na tržištu postoje razne aplikacije koje omogućuju pohranu i dijeljenje recepata, a neke koriste umjetnu inteligenciju za otkrivanje novih recepata ili nude kreiranje popisa za kupovinu. Ipak, mnoge od njih ne omogućuju unos važnih detalja poput nutritivnih vrijednosti i vremena pripreme. Aplikacija ovog rada razvijena je u razvojnom okruženju Android Studio uz programski jezik Kotlin, deklarativni okvir Jetpack Compose za dizajn korisničkog sučelja, koristi MVVM strukturu te Firebase za pohranu podataka. Uključuje funkcionalnosti kao što su detaljan unos recepata, pregled i pretraga, kategorizacija i nasumično generiranje plana prehrane. Testiranjem na grupi od deset ispitanika pokazano je da su glavni ciljevi aplikacije zadovoljeni uz mogućnost naknadnih poboljšanja.

Ključne riječi: Android aplikacija, kategorizacija recepata, personalizirana pohrana, pregled recepata

ABSTRACT

Given the increasingly fast pace of life, mobile applications help people navigate the hustle and bustle of everyday life. Among these are recipe storage apps, which make the cooking process less burdensome for those who don't enjoy it and more enjoyable for those who do. There are various apps on the market that allow users to store and share recipes, with some even using artificial intelligence to discover new recipes or offer the creation of shopping lists. However, many of these apps lack the ability to input important details like nutritional values and preparation times. The application presented in this work was developed in Android Studio using the Kotlin programming language and the Jetpack Compose framework for UI design. It employs the MVVM architecture and uses Firebase for data storage. The app includes features such as detailed recipe input, viewing and searching, categorization, and random meal plan generation. Testing with a group of ten participants showed that the main goals of the app were met, with potential for future improvements.

Keywords: Android application, recipe categorization, personalized storage, recipe viewing

PRILOZI

1. Programski kôd mobilne aplikacije dostupan na:

<https://gitlab.com/dorja.pozder/recipe-realm> (posljednje ažurirano: 24.8.2024)