Android aplikacija za planiranje zadataka

Šnajder, Brigita

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:370930

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-01-30**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Preddiplomski sveučilišni studij računarstva

ANDROID APLIKACIJA ZA PLANIRANJE ZADATAKA

Završni rad

Brigita Šnajder

Osijek, 2024.

SADRŽAJ

1. UVOD

Tehnologija je posvuda oko nas, a pametni uređaji uvelike olakšavaju odvijanje ljudskih obaveza i služe kao alat za razonodu. Često ljudi zaborave koliko im zapravo njihovi pametni uređaji mogu pomoći u svakodnevnim aktivnostima, od korištenja kalkulatora za izračun cijene namirnica, do povezivanja s cijelim svijetom uz samo jedan dodir prsta. S obzirom na to koliko tehnologija može olakšati život, osobno smatram da je ključno maksimalno iskoristiti njen potencijal za organizaciju i učinkovitost.

Motivacija za odabir ove teme leži upravo u toj ideji: željela sam istražiti i razviti rješenje koje pomaže pojedincima i grupama u svakodnevnoj organizaciji, omogućavajući im lakše planiranje i izvršavanje obaveza. U današnjem ubrzanom načinu života, kvalitetno upravljanje vremenom postaje sve važnije, a aplikacije koje pomažu u praćenju zadataka i rasporeda mogu značajno doprinijeti smanjenju stresa i povećanju produktivnosti. Zbog toga sam odlučila razviti aplikaciju koja će služiti kao planer ili popis zadataka, čime se olakšava organizacija aktivnosti kroz određeno vrijeme.

Odabir tehnologija također je bio važan aspekt u razvoju ovog rada. Korištenjem Angular *frameworka* za klijentski dio aplikacije omogućuje se brza izrada responzivnih i intuitivnih web aplikacija, dok je Android platforma odabrana kako bi se aplikacija mogla koristiti na velikom broju mobilnih uređaja. Angular pruža fleksibilnost u razvoju, što omogućuje jednostavne nadogradnje i modifikacije aplikacije ovisno o korisničkim potrebama, dok Android kao široko rasprostranjena platforma omogućuje velik broj korisnika i visok stupanj prilagodljivosti.

Ovaj rad namijenjen je olakšavanju organiziranja i izvršavanja obaveza pojedinca ili skupine. Predviđeno je korištenje aplikacije kao planera za buduće aktivnosti ili kao običan popis poslova koje je potrebno obaviti kroz neko određeno vrijeme. Aplikacija je napravljena tako da je u svakom trenutku moguća modifikacija i nadogradnja, ovisno o korisničkim potrebama.

U nastavku je dan pregled sličnih rješenja aplikacija i kako svaka od njih na svoj način pridonosi korisnicima. U trećem poglavlju je opisano koje točno tehnologije su se koristile pri izradi ovog rada. Nastavno na to opisan je i postupak izrade aplikacije, koju prati detaljan prikaz korištenja aplikacije.

1.1. Zadatak završnog rada

Cilj završnog rada je izraditi Android aplikaciju koja omogućava korisnicima učinkovito upravljanje svojim zadacima kroz funkcionalnosti dodavanja novih lista i zadataka unutar tih lista. Aplikacija treba pružiti intuitivno sučelje za kreiranje i organizaciju zadataka, omogućujući korisnicima da strukturiraju svoje obaveze na način koji im najbolje odgovara.

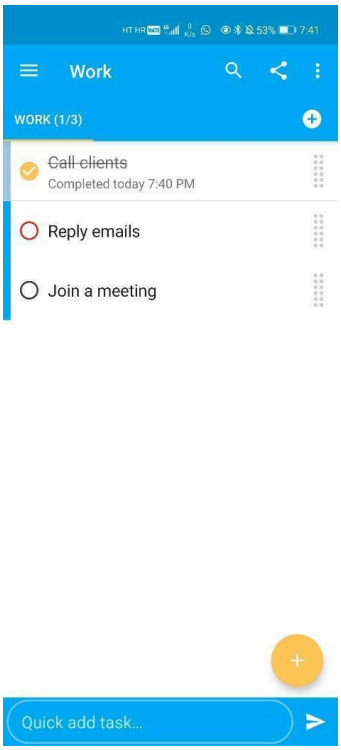
2. PREGLED SLIČNIH RJEŠENJA

U ovom poglavlju navedene su Android aplikacije koje su slične tematike i sve se bave organiziranjem obaveza ili funkcioniraju kao običan popis obaveza i zadataka. Sve ih je moguće preuzeti s Googleove trgovine *Google Play Store*.

2.1. Tasks: to do list & tasks

Aplikacija *Tasks: to do list & tasks* omogućuje korisnicima kreiranje zadataka unutar određenih lista, kao i dodavanje brzih zadataka. Korisnici mogu sortirati zadatke prema važnosti i grupirati ih prema statusu izvršenosti (izvršeni/neizvršeni), što olakšava pregled nad obavezama. Iako aplikacija nudi osnovne funkcionalnosti, nedostaje joj fleksibilnost i naprednije opcije kao što je upravljanje projektima. Aplikacija koja je predmet ovog rada nudi slične funkcionalnosti, ali s dodatnim mogućnostima prilagodbe sučelja i fleksibilnim pristupom organizaciji zadataka.

Na slici 2.1. prikazan je zaslon aplikacije s listom od nekoliko zadataka.

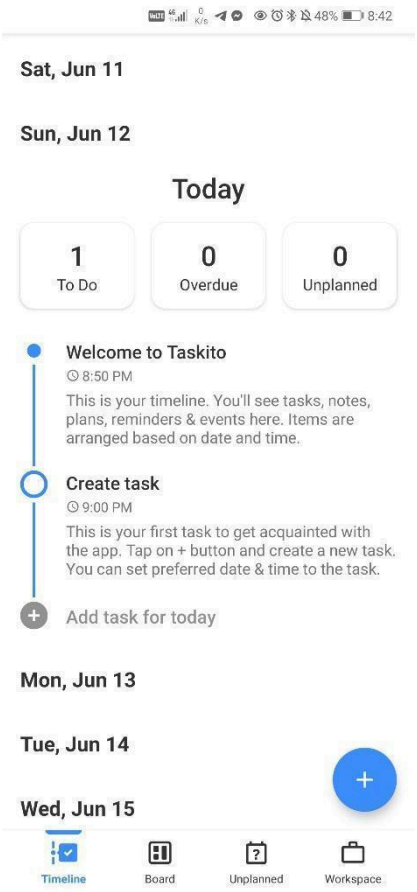


Slika 2.1. Tasks: to do list & tasks [1]

2.2. Taskito: To-Do List, Planner

Aplikacija *Taskito: To-Do List, Planner* pruža korisnicima pregled zadataka po danima te mogućnost organiziranja zadataka po projektima, uz bilješke i upute za izvršenje. Ova funkcionalnost korisnicima omogućuje bolji uvid u dugoročne obaveze i olakšava planiranje. Aplikacija koja se razvija u ovom radu nudi sličnu fleksibilnost, ali omogućuje i dodatnu prilagodbu sučelja prema potrebama korisnika te jednostavnije nadogradnje funkcionalnosti.

Na slici 2.2. prikazan je zaslon aplikacije koji sadrži pregled svih zadataka po danima.

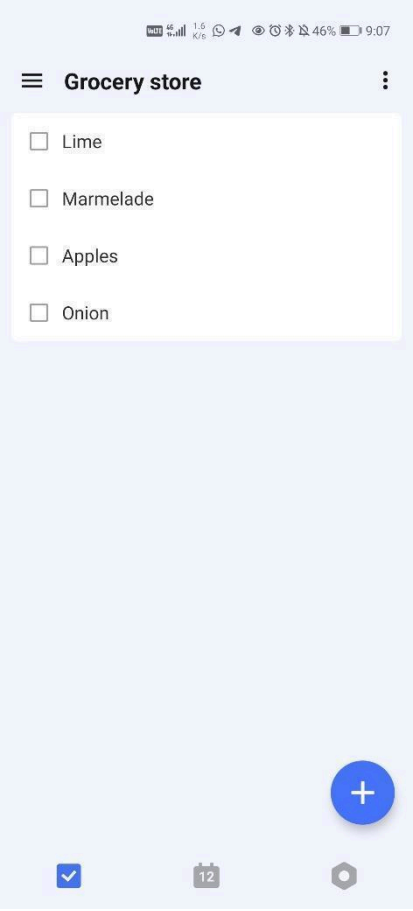


Slika 2.2. Taskito: To-Do List, Planner [2]

2.3. TickTick: To-do list & Tasks

Aplikacija *TickTick: To-do list & Tasks* omogućuje korisnicima kreiranje računa kako bi mogli spremiti svoje podatke u udaljenu bazu podataka, što omogućuje pristup s više uređaja. Aplikacija je usmjerena na praćenje obavljenih zadataka, no nema napredne opcije za organizaciju zadataka ili projekata. Aplikacija iz ovog rada nudi sličnu mogućnost sinkronizacije podataka, uz naprednije funkcionalnosti za organizaciju i prilagodbu zadataka.

Na slici 2.3. prikazan je jedan od načina primjene ove aplikacije, za popisivanje namjernica koje je potrebno kupiti u idućoj nabavci.

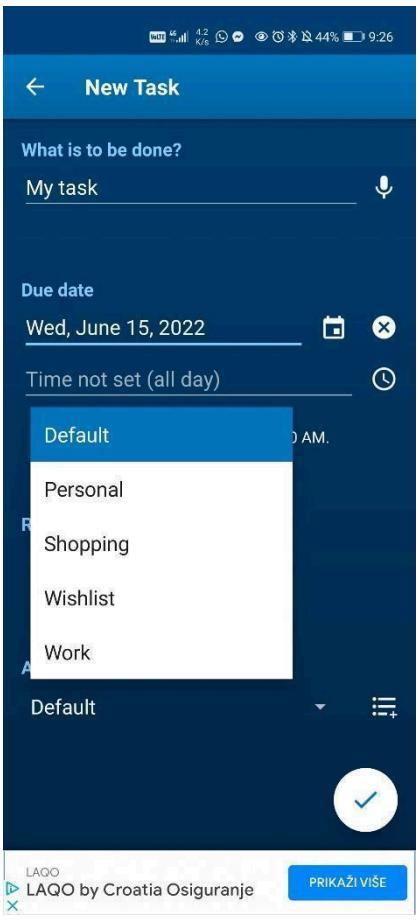


Slika 2.3. TickTick: To-do list & Tasks [3]

2.4. To Do List

Aplikacija *To Do List,* je jednostavna aplikacija koja se koristi za organizaciju zadataka kroz osnovni postupak kreiranja zadatka, nakon čega korisnik bira u koju će listu zadatak biti upisan. Minimalistički dizajn može biti privlačan korisnicima koji traže jednostavno rješenje, ali aplikacija razvijena u ovom radu nudi širi spektar mogućnosti, omogućujući detaljniju organizaciju i prilagodbu zadataka prema korisničkim potrebama.

Na slici 2.4. prikazano je kreiranje jednog zadatka.

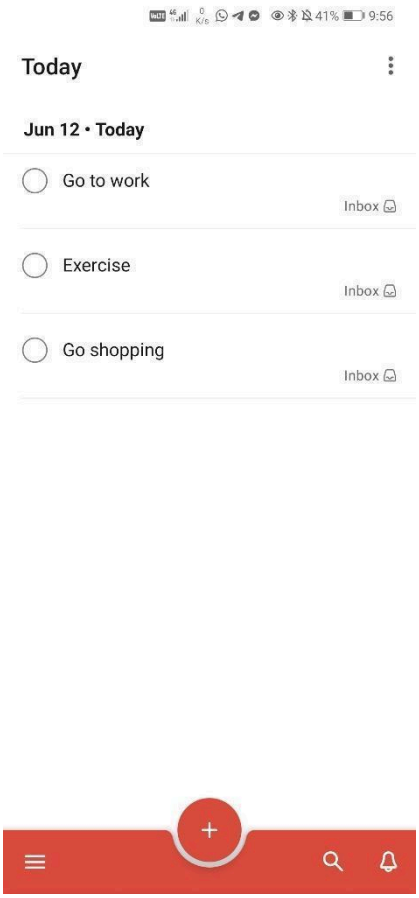


Slika 2.4. To Do List [4]

2.5. Todoist: To-Do List & Tasks

Aplikacija *Todoist: To-Do List & Tasks* nudi jednostavno sučelje koje omogućava korisnicima da se lako snađu u aplikaciji. No, aplikacija zahtijeva registraciju za korištenje te ne nudi dodatne opcije za sortiranje unutar lista zadataka. Aplikacija iz ovog rada pruža veću fleksibilnost, bez obavezne registracije, te nudi personalizirane funkcije i prilagodbu prema različitim korisničkim zahtjevima.

Slika 2.5. prikazuje par kreiranih zadataka.



Slika 2.5. Todoist: To-Do List & Tasks [5]

3. ZAHTJEVI ZA APLIKACIJU

Razvoj aplikacije koja se koristi kao planer zadataka zahtijeva detaljno definiranje kako funkcionalnih, tako i nefunkcionalnih zahtjeva. Ovi zahtjevi pomažu u usmjeravanju dizajna i implementacije aplikacije, osiguravajući da ona zadovolji potrebe korisnika i funkcionalne ciljeve. Kroz definiranje zahtjeva, razvojni inženjeri mogu stvoriti jasnu sliku o tome što aplikacija treba raditi, kako bi trebala raditi i na koji način će zadovoljiti korisničke potrebe.

Funkcionalni zahtjevi definiraju specifične funkcionalnosti koje aplikacija mora pružiti. Oni se odnose na konkretne zadatke i operacije koje aplikacija mora obavljati kako bi korisnicima omogućila učinkovito upravljanje svojim zadacima. Ovi zahtjevi su usmjereni na osnovne funkcionalnosti aplikacije, kao što su autentikacija korisnika, upravljanje todo listama, i pohrana podataka.

S druge strane, nefunkcionalni zahtjevi odnose se na karakteristike aplikacije koje nisu izravno povezane s njenim funkcionalnostima, ali su ključne za pružanje kvalitetnog korisničkog iskustva. Ovi zahtjevi uključuju aspekte poput sigurnosti, performansi, stabilnosti i korisničkog sučelja. Oni osiguravaju da aplikacija ne samo da ispunjava svoje osnovne funkcionalnosti, već i da radi efikasno, pouzdano i korisniku prijateljski.

U nastavku su detaljno opisani funkcionalni i nefunkcionalni zahtjevi koji su ključni za razvoj aplikacije. Ovi zahtjevi predstavljaju osnovu za dizajn, implementaciju i testiranje aplikacije, osiguravajući da se postignu ciljevi projekta i ispune očekivanja korisnika.

3.1. Funkcionalni

Aplikacija koja je osmišljena kao planer zadataka treba ispunjavati nekoliko ključnih funkcionalnih zahtjeva kako bi omogućila korisnicima učinkovito upravljanje svojim zadacima i obavezama.

Prvo, aplikacija omogućava korisnicima registraciju i prijavu korištenjem Firebase Authentication. Proces registracije zahtijeva unos adrese elektroničke pošte i lozinke, uz verifikaciju elektroničke pošte radi potvrde valjanosti korisničkog računa. Postojeći korisnici mogu se prijaviti sa svojim

e-mail adresama i lozinkama, a postoji i opcija za oporavak lozinke u slučaju zaborava. Nakon uspješne prijave, korisnici mogu uređivati svoje podatke, uključujući promjenu lozinke ili e-mail adrese, uz pridržavanje sigurnosnih pravila[6].

Na glavnom zaslonu aplikacije korisnici mogu vidjeti popis svojih todo lista. Svaka lista prikazuje osnovne informacije, poput naziva i broja zadataka. Klikom na naziv bilo koje liste, korisnici mogu pristupiti detaljnom prikazu zadataka unutar te liste. U ovom prikazu, korisnici mogu filtrirati i sortirati zadatke prema različitim kriterijima, uključujući datum i važnost, radi lakšeg upravljanja.

Korisnici imaju mogućnost kreiranja novih todo lista unosom naziva i, po potrebi, dodatnih informacija kao što su opis ili boja liste. Nakon kreiranja liste, korisnici mogu uređivati naziv ili dodavati opis kako bi poboljšali organizaciju svojih zadataka. Unutar svake liste, korisnici mogu vidjeti sve zadatke s osnovnim informacijama kao što su naziv, status i rok izvršenja. Klikom na zadatak, dostupni su dodatni detalji poput opisa, prioritet i napomene.

Dodavanje novih zadataka unutar odabrane liste uključuje unos naziva zadatka, definiranje statusa (npr. dovršen ili nedovršen) i opcionalno dodavanje roka izvršenja. Korisnici također mogu dodati kategorije ili oznake zadacima radi lakšeg filtriranja i pretraživanja. Uređivanje zadataka omogućuje korisnicima promjenu naziva, statusa, roka izvršenja ili dodavanje/uklanjanje napomena. Brisanje zadataka je također omogućeno, uz opciju potvrde kako bi se spriječilo nenamjerno uklanjanje.

Svi podaci o listama i zadacima pohranjuju se u Firebase Firestore, što omogućuje automatsku sinkronizaciju između različitih uređaja korisnika.

3.2. Nefunkcionalni

U pogledu nefunkcionalnih zahtjeva, aplikacija mora osigurati visoku razinu sigurnosti podataka koristeći Firebase Authentication za zaštitu korisničkih informacija. Svi podaci moraju biti pohranjeni i obrađeni u skladu s važećim zakonima o zaštiti privatnosti, uključujući GDPR ili lokalne propise.

Performanse aplikacije trebaju biti optimizirane za brzo učitavanje i responzivnost. Aplikacija treba

osigurati da se korisničko sučelje brzo učitava i odgovara na korisničke akcije, bez obzira na broj zadataka ili veličinu lista. Stabilnost aplikacije je ključna, a svi kritični dijelovi trebaju biti temeljito testirani kako bi se osigurala pouzdanost.

Sučelje aplikacije mora biti intuitivno i lako za navigaciju. Dizajn sučelja treba omogućiti korisnicima brz pristup svim funkcionalnostima bez potrebe za složenim obukama. Sučelje mora biti prilagodljivo različitim veličinama zaslona i orijentacijama, osiguravajući dosljedno iskustvo na svim uređajima.

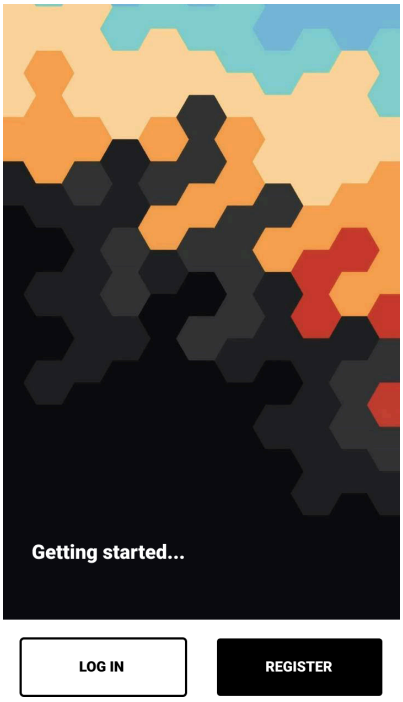
Konačno, aplikacija treba omogućiti sinkronizaciju podataka između različitih uređaja korisnika i biti kompatibilna s različitim verzijama Android operativnog sustava. To uključuje osiguranje da aplikacija funkcionira ispravno na različitim modelima uređaja i verzijama sustava, pružajući dosljedno korisničko iskustvo.

4. OPIS POSTUPKA IZRADE

U ovom poglavlju opisana je sama struktura aplikacije, koju dodatno upotpunjava objašnjen cijeli proces izrade aplikacije. U sljedećim poglavljima opisano je kako je zamišljen dizajn aplikacije, ideja programskog rješenja (koja je dalje podijeljena u dva dijela, klijentski dio aplikacije i pozadinski dio aplikacije).

4.1. Dizajn aplikacije

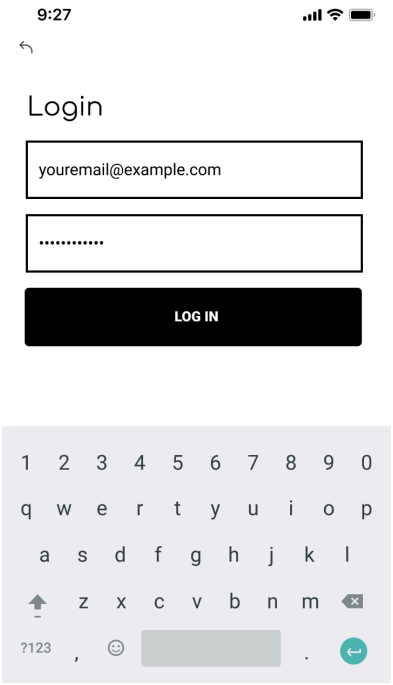
Za kreiranje dizajna aplikacije korišten je alat Figma. Figma je web-bazirani alat za dizajn i prototipiranje koji omogućuje suradnju u stvarnom vremenu, idealan za timove koji rade na korisničkom sučelju (UI) i korisničkom iskustvu (UX). Omogućuje dizajnerima stvaranje, dijeljenje i testiranje dizajna direktno u pregledniku, uz mogućnost simultanog rada više korisnika [8]. Ulaskom u aplikaciju, prikazan je zaslon vidljiv na slici 4.1. na kojem će biti moguće odabrati želi li se korisnik prijaviti s postojećim podacima ili želi kreirati novi korisnički račun.



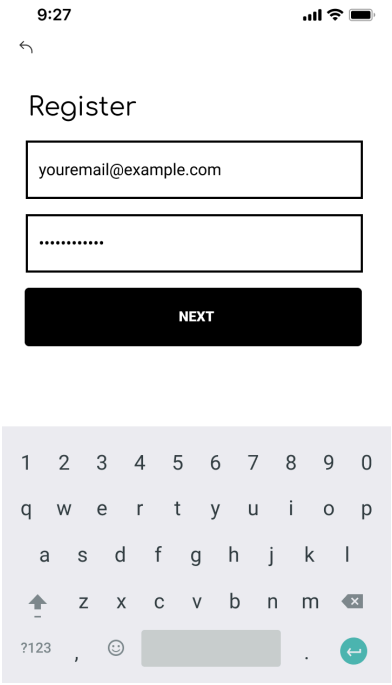
Slika 4.1. Početni zaslon

Odabirom jedno od ponuđenog, otvoren je odgovarajući zaslon, ovisno o odabiru. Odabere li

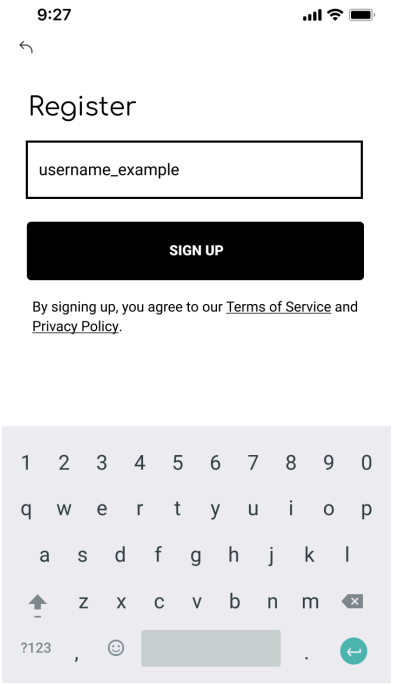
korisnik samo prijavu, otvori se zaslon gdje će moći upisati svoje podatke (slika 4.2.). Na toj slici prikazan je zaslon gdje korisnik za prijavu u aplikaciju treba upisati svoju adresu e-pošte i lozinku. Slika 4.3. slika prikazuju proces kreiranja novog korisničkog računa, korisnik prvo treba upisati adresu e-pošte i lozinku koju želi koristiti te potom prelazi na drugi zaslon (slika 4.4.) gdje upisuje i željeno korisničko ime.



Slika 4.2. Zaslon za prijavu

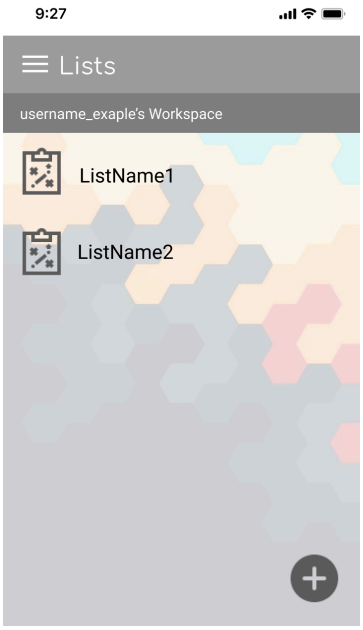


Slika 4.3. Zaslon za kreiranje računa



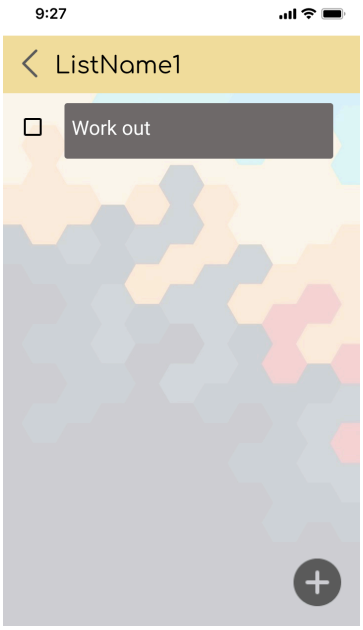
Slika 4.4. Zaslon za dodavanje korisničkog imena

Nakon što se korisnik prijavi u aplikaciju prikazan mu je zaslon kao na slici 4.5.. Na tom zaslonu prikazane su sve kreirane liste koje sadržavaju zadatke. Na početku neće postojati nijedna lista već će taj dio biti prazan. Kada korisnik odabere jednu listu prikazan je ekran kao na slici 4.6. gdje je prazan zaslon kada je lista tek kreirana ili, ako lista postoji od prije, prikazani su zadaci dodani u listu. Ukoliko korisnik odabere dodavanje nove liste prikazan je zaslon kao na slici 4.8. gdje korisnik upisuje željeno ime liste. Na slici 4.9. prikazano je kreiranje zadatka, u tom dijelu korisnik može upisati ime zadatka, početni i krajnji datum za izvršenje zadatka, može dodati opis zadatka i odabrati važnost zadatka. Za kraj, može birati u koju listu želi dodati zadatak.

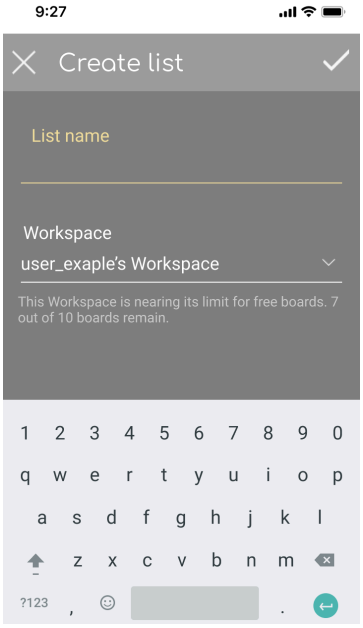


Slika 4.5. Prikaz lista

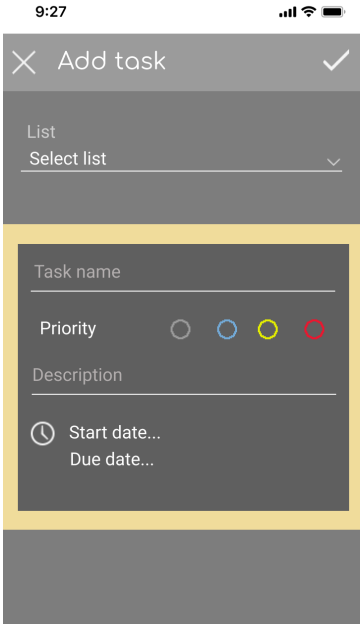
Slika 4.6. Prikaz liste bez zadataka



Slika 4.7. Prikaz liste s dodanim poslom



Slika 4.8. Zaslon kreiranja liste



Slika 4.9. Zaslon kreiranja zadatka

4.2. Ideja programskog rješenja

Drugi korak pri realizaciji aplikacije je kreiranje ideje programskog rješenja. U ovom poglavlju opisano je ukratko kojim redom se aplikacija radila te će u idućim naslovima je detaljnije objašnjen svaki korak zasebno.

Prvo će prema izrađenom Figma dizajnu biti kreirani *XML layouti* koji odgovaraju njima. Najčešće će se koristiti *Relative layout* za obuhvaćanje sadržaja u cjelinu. Za prikaz sadržaja korišten je *TextView*, a za unos sadržaja koristit će se *EditText*. Od ostalih dodataka korišteni su npr. *RecyclerView* i *Floating action button*.

Nakon kreiranja layouta kreirani su *fragmenti* i povezani s odgovarajućim *layoutima*.

Za svaku akciju koja se nalazi u pojedinom fragmentu, kreirana je aktivnost i prijelaz koji povezuje layout i fragment u jednu cjelinu i čini ih funkcionalnim.

Nakon povezivanja svega u funkcionalnu cjelinu, implementiran je Firebase autentikacija kako bi mogli osigurati ispravnu autentikaciju korisnika. Uz to potrebno je implementirati i Firebase Firestore i time omogućiti komunikaciju s Googleovim *Firebase* C*loud* sustavom. Služi za slanje i primanje podataka unesenih ili zatraženih od strane korisnika.

Zadnji korak pri izradi aplikacije je prikazivanje podataka korisniku na zaslonu iz unaprijed kreirane baze podataka.

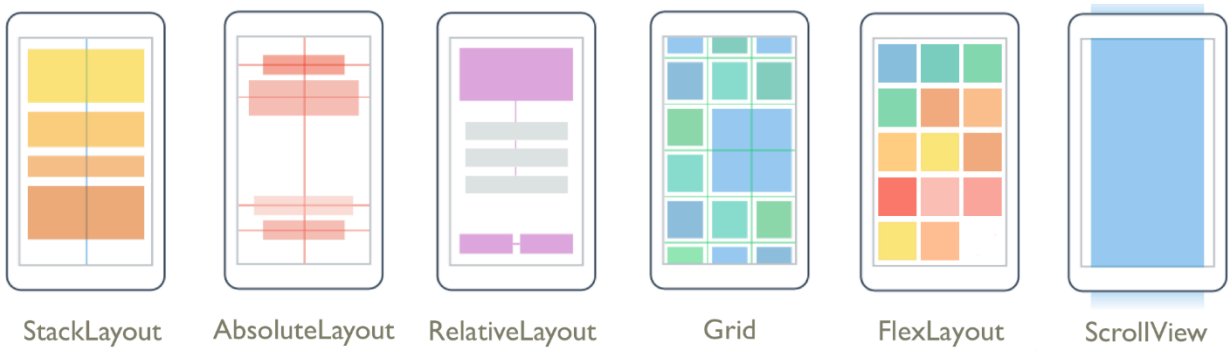
4.3. Klijentski dio aplikacije

Frontend obuhvaća sve značajke koje su vidljive korisniku. S programerske strane podrazumijeva sve funkcionalnosti potrebne za prikaz podataka korisniku te omogućavanje unosa od strane korisnika. Zadužen je za komunikaciju i razmjenu podataka s backendom aplikacije.

U idućim podnaslovima objašnjen je svaki dio frontenda posebno.

4.3.1. XML Layout

Glavni layout korišten pri izradi aplikacije je *Relative layout*. *Relative layout* je *layout* koji slaže elemente prema poziciji nekog drugog unaprijed određenog elementa u layoutu. Pozicija elementa određuje se prema relativnoj poziciji nekog drugog elementa koji pripada istom „roditelju“. Neke od značajki koje *Relative layout* pruža jesu centriranje vertikalno, horizontalno ili oboje, poravnavanje prema „roditelju“ elementu gore, dolje, lijevo i desno, poravnavanje prema nekom od elemenata s gornje, donje, lijeve i desne strane i ostale[7]. Prikaz *RelativeLayout* u usporedbi sa ostalim *layoutima* prikazuje Slika 4.10..



Slika 4.10. Primjeri različitih layouta

Svakom od layouta i elementa dodan je ID prema kojemu ih se u kodu poziva po potrebi. To je veoma važno kod Relative layouta jer se prema ID-u elementa određuje pozicija drugog elementa.

Prvi element koji je korišten je *TextView*, u kojemu se prikazuju sve tekstualne poruke i upute u aplikaciji. Za pravilno funkcioniranje, potrebno mu je bilo kodirati visinu i širinu koje ne moraju biti neka određena vrijednost već element može zauzeti punu visinu/širinu „roditelja“. Ukoliko se odredi točna visina/širina elementa, njegova mjerna jedinica je dp. Postoje i ostali atributi koje je moguće postaviti na element, a neki od njih su font teksta, veličina teksta (njegova mjerna jedinica izražava se u sp), boja teksta itd.

Idući korišten element je gumb (engl. *Button*). Gumb je element koji je responzivan na korisnikov dodir ili slično. Gumb na to odgovara unaprijed programiranom radnjom. Pri izradi gumba korišteni su već spomenuti atributi.

ImageView je element koji je korišten za učitavanje slika ili ikona u aplikaciju. Na slici 4.11. prikazan je dio koda koji se odnosi na element *ImageView* i njegovo korištenje. Tim kodom prikazano je učitavanje ikone koja će služiti kao tipka za vraćanje na prethodno stanje.

```
<ImageView
    android:id="@+id/undoButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/undo"
    android:contentDescription="@string/undo_button"
    android:layout_marginStart="30dp"
    android:layout_marginTop="50dp"
    />
```

Slika 4.11. Kod ImageView elementa

Za unos teksta korišten je element *EditText*. On za ispravno funkcioniranje treba imati samo atribute koji određuju visinu i širinu elementa, ostali nisu obavezni. U izradi aplikacije u nekim elementima se kasnije može ograničiti što točno korisnik unosi, u ovom slučaju to je bilo ograničavanje korisnika na unos točne adrese e-pošte tako da se provjerava znak '@' i ispravna domena iza njega.

RecyclerView korišten je za fleksibilan prikaz kompleksnijih elemenata u listi. U aplikaciji se koristi lista koja ima promjenjiv broj elemenata, stoga je preporučeno koristiti ovaj element za prikaz elemenata liste. U ovom radu *RecyclerView* je napravljen od liste elemenata koji su tipa *RelativeView*. Taj *RelativeView* je unaprijed isprogramiran sadržavajući elemente koji su nam potrebni za prikaz jednog elementa liste, a sadrži *ImageView* i *TextView*.

U donjem desnom kutu zaslona nalazi se *FloatingActionMenu* koji se koristi kao gumb koji reagira na klik (dodir) te on otvara izbornik s gumbima koji su elementa imena *FloatingActionButton*. Primjer jednog akcijskog gumba nalazi se na slici 4.12..

```
<com.github.clans.fab.FloatingActionButton
    android:id="@+id/fabAddTask"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:fab_size="mini"
    app:fab_label="Add task"
    android:src="@drawable/add"
    app:fab_colorNormal="@color/orange"
    app:fab_colorPressed="@color/light_orange"/>
```

Slika 4.12. Prikaz jednog FloatingActionButton-a

Za njihovu implementaciju potrebno je bilo dodati *dependency*, jer je za korištenje tih elemenata korišten gotov vanjski element, što je vidljivo je na slici 4.13..

```
dependencies {
    implementation 'com.github.clans:fab:1.6.4'
}
```

Slika 4.13. Dodavanje dependency

4.3.2. Fragmet

Nakon što su XML layouti gotovi iza njih su kreirani fragmenti. Fragmenti predstavljaju više zaslona unutar jedne aktivnosti. Nakon kreiranja klase koja opisuje ponašanje fragmenta korišten je *layout inflater* kako bi fragment znao koji layout treba koristiti. Korištenja *inflatera* prikazano je na slici 4.14..

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    return inflater.inflate(R.layout.fragment_logged_out, container,  attachToRoot: false);
}
```

Slika 4.14. Dodavanje layouta fragmentu s layout inflaterom

Nakon što je svaki *layout* pridružen pripadajućem *fragmentu* dohvaća se referenca svakog elementa koji se nalazi u *layoutu* tako da je moguće koristiti ih u daljnjem kodu. Za dohvaćanje reference elementa korištena je metoda *findViewById* kojoj je predan ID elementa (prethodno postavljen u layoutu). Procedura ili funkcija *Event listener* koja čeka neki određeni događaj i može se postaviti na bilo koji element. U primjeru na slici 4.15. prikazano je korištenje event *listenera* na elementu

gumba.

```
private void initButtons(View view) {
    Button btnLogIn = view.findViewById(R.id.buttonLogIn);
    btnLogIn.setOnClickListener(view1 ->  openRegisterLogin(Type.LOGIN));
    Button btnRegister = view.findViewById(R.id.buttonRegister);
    btnRegister.setOnClickListener(view12 -> openRegisterLogin(Type.REGISTER));
}
```

Slika 4.15. Dohvaćanje reference i korištenje listenera

Za prijelaz na drugi fragment korištena je funkcija koja kreira novi fragment, pokreće se transakcija i na kraju mijenja stari fragment s novim, a način korištenja vidljiv je na slici 4.16..

```
private void openRegisterLogin(Type type){
    Fragment registerLoginFragment = new RegisterLogin(type);
    FragmentTransaction transaction = getActivity().getSupportFragmentManager().beginTransaction();
    transaction.replace(R.id.authActivity, registerLoginFragment);
    transaction.commit();
}
```

Slika 4.16. Funkcija za prijelaz na drugi fragment

4.3.3. Aktivnost

Za prikaz fragmenata u aplikaciji korišteni su aktivnosti. Aktivnost predstavlja zaseban zaslon u korisničkom sučelju kao što su prozori i okviri. Aktivnost nema doticaj s korisnikom jer on samo radi u pozadini i zbog toga je njegov *XML layout* prazan. Pomoću njega njega su prikazani svi fragmenti i on je zadužen za komunikaciju između *fragmenata* i pozadinskog dijela aplikacije, također u njemu se nalazi sva funkcionalnost za prijelaze između aktivnosti. Oni sadrže varijable, a to su reference na bazu podataka i na firebase autentikaciju, tako da fragmenti nemaju nikakvu veću odgovornost nego samo prosljeđuju podatke aktivnosti koji dalje šalje upite na pozadinski dio aplikacije odnosno na firebase. Prikaz među aktivnostima prikazuje Slika 4.17. dok Slika 4.18. prikazuje varijable za pristup sustavu autentikacije i referencu na bazu podataka.

```
Intent myIntent = new Intent( packageContext: AuthActivity.this, MainActivity.class);
AuthActivity.this.startActivity(myIntent);
```

Slika 4.17. Prijelaz između aktivnosti

```
public FirebaseAuth firebaseAuth;
public FirebaseFirestore db = FirebaseFirestore.getInstance();
```

Slika 4.18. Varijable za pristup sustavu autentikacije i referenca na bazu podataka

Kako prilikom korištenja aplikacije ne bi došlo do neželjenih radnji/akcija prilikom pritiska na tipku nazad (engl. *back*), unutar aktivnosti je implementirana i prepisana funkcija *onBackPressed* na način da se prilikom pritiska na tipku nazad ne dogodi ništa, prikaz Slika 4.19..

```
@Override
public void onBackPressed() {

}
```

Slika 4.19. Prepisana funkcija onBackPressed

U aktivnosti *AuthActivity* nalaze se funkcije koje se pozivaju prilikom prijave i registracije korisnika.

4.3.4. Autentikacija

Za korištenje firebase autentikacije, potrebno je omogućiti autentikaciju u razvojnom okruženju putem alata Android Studio Belito S prijavom u Android studio, aplikacija je povezana s Google računom, nakon čega je u postavkama Firebase konzole na internetskom pregledniku uključena autentikacija i odabran način na koji način će se korisnik prijaviti u aplikaciju (u slučaju ove aplikacije prijaviti se je moguće pomoću e-maila i lozinke). Firebase-ovom autentikacijom je omogućeno upravljanje i nadzor nad korisnicima i njihovim prijavama. Prije korištenja autentikacije potrebno je dodati *dependency*, što je prethodno već objašnjeno. Za potrebu autentikacije koriste se dvije funkcije, jedna za prijavu i druga za registraciju korisnika, Slika 4.20.. Prije samog pozivanja funkcija od firebase autentikacije, odjavi se trenutnog korisnika da ne bi došlo do konflikata i grešaka kod korisničkih računa. Nakon toga šalje se zahtjev za prijavu/registraciju s podacima koje je korisnik unio (pri tome korisnik vidi samo fragment u kojem trenutno radi, ostalo je njemu nevidljivo) te se dodaju listeneri ako je poziv uspješan da pređe na idući korak. Ukoliko poziv nije bio uspješan tj. krivi podaci su uneseni program ispiše poruku greške.

```
private void login(String email, String pass) {
    this.authActivity.firebaseAuth.signOut();
    this.authActivity.firebaseAuth.signInWithEmailAndPassword(email, pass)
            .addOnSuccessListener(task -> {
                Toast.makeText(getContext(), text: "Logged in successfully.", Toast.LENGTH_SHORT).show();
                this.authActivity.openMainActivity(Objects.requireNonNull(task.getUser()));
            })
            .addOnFailureListener(e -> Toast.makeText(getContext(), e.getMessage(), Toast.LENGTH_SHORT).show());
}

private void register(String email, String pass) {
    this.authActivity.firebaseAuth.signOut();
    this.authActivity.firebaseAuth.createUserWithEmailAndPassword(email, pass)
            .addOnSuccessListener(task -> {
                Toast.makeText(getContext(), text: "Registered successfully.", Toast.LENGTH_SHORT).show();
                Fragment registerStep2 = new RegisterStep2();
                FragmentTransaction transaction = this.authActivity.getSupportFragmentManager().beginTransaction(

                Map<String, Object> data = new HashMap<>();
                data.put( k: "username", Objects.requireNonNull(task.getUser().getEmail()).split( regex: "@")[0]);

                this.authActivity.db.collection( collectionPath: "users").document(task.getUser().getUid())
                        .set(data);

                transaction.replace(R.id.authActivity, registerStep2);
                transaction.addToBackStack(null);
                transaction.commit();
            })
            .addOnFailureListener(e -> Toast.makeText(getContext(), e.getMessage(), Toast.LENGTH_SHORT).show());
}
```

Slika 4.20. Funkcije za prijavu i registraciju

4.3.5. Rukovanje podacima iz baze podataka

Rukovanje podacima iz baze ključno je za učinkovito pohranjivanje, pretraživanje i upravljanje informacijama unutar aplikacija. Ovaj rad fokusiran je na korištenje *Firebase Realtime Database*, popularne usluge koju pruža Google, koja omogućuje razvoj aplikacije s podacima u stvarnom vremenu. Ovaj sustav omogućava jednostavno pohranjivanje podataka i njihovu sinkronizaciju između klijenta i servera, bez potrebe za upravljanjem vlastitim infrastrukturnim rješenjima.

Podaci se mogu učitati putem različitih metoda, ali najčešće se koristi *ValueEventListener*, koji osigurava da aplikacija uvijek ima najnovije podatke. Slika 4.21prikazuje primjer kako se podaci učitavaju iz Firebase baze podataka unutar aplikacije.

```
private void fetchTaskLists() {  1 usage
    taskLists = new ArrayList<>();

    databaseReference = FirebaseDatabase.getInstance().getReference( path: "taskLists");

    databaseReference.addValueEventListener(new ValueEventListener() {
        @Override  2 usages
        public void onDataChange(DataSnapshot dataSnapshot) {
            taskLists.clear();
            for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
                TaskList taskList = snapshot.getValue(TaskList.class);
                taskLists.add(taskList);
            }
            if (adapter != null) {
                adapter.notifyDataSetChanged();
            } else {
                adapter = new TaskListAdapter(taskLists, taskList -> {
                });
                recyclerView.setAdapter(adapter);
            }
        }

        @Override
        public void onCancelled(DatabaseError databaseError) {
            Log.w( tag: "TaskListFragment",  msg: "loadPost:onCancelled", databaseError.toException());
        }
    });
}
```

Slika 4.21. Funkcija za dohvaćanje liste zadataka

4.3.6. Prikaz podataka

U ovom poglavlju se fokusira na korištenje adaptera za prikaz podataka u Android aplikaciji. Adapter je ključna komponenta koja prikazuje UI elemente, kao što su *RecyclerView,* s izvorom podataka koji je u ovom slučaju lista zadataka *TaskList.* Ovdje je objašnjeno kako se koristi adapter da bi se podaci prikazali pravilno i kako se korisničke interakcije mogu obraditi putem korisničkog sučelja.

Adapter koji smo razvili naziva se *TaskListAdapter* i pruža način za upravljanje listom objekata tipa *TaskList*. U ovoj klasi korišten je *RecyclerView.Adapter* koji omogućava efikasno upravljanje velikim brojem stavki, bez zastoja i s optimizacijom resursa [9].

Da bi se koristio TaskListAdapter u aplikaciji, kreirana je instanca adaptera i postavljena na

RecyclerView. Osnovna konfiguracija za to može izgledati ovako: prvo, potrebno je inicijalizirati *RecyclerView*, a zatim kreirati instancu *TaskListAdapter*, proslijediti mu listu zadataka i implementaciju *OnTaskListClickListener* sučelja za obradu klika na stavku. Nakon toga, adapter se postavlja na *RecyclerView*.

Korištenjem *TaskListAdapter*, omogućuje se efikasan i modularan pristup prikazu podataka u aplikaciji. Ovaj pristup ne samo da olakšava manipulaciju s listama, već omogućuje i lako dodavanje korisničkih interakcija. Kroz ovaj adapter, može se na jednostavan način raditi s podacima, što je ključno za izgradnju interaktivnih i responzivnih aplikacija.

4.4. Pozadinska aplikacija

U ovom poglavlju daje se osvrt na pohranu podataka putem Firebase. Firebase je platforma koja pruža raznolike usluge za razvoj mobilnih i web aplikacija, uključujući i jednostavnu pohranu podataka. U aplikaciji, korišten je Firebase Firestore kao baza podataka, a podaci su strukturirani u dokumente i kolekcije.

U Firestore-u centralna jedinica za pohranu podataka je kolekcija koja sadrži dokumente. Svaka kolekcija može sadržavati neograničen broj dokumenata, a svaki dokument predstavlja pojedinačni zapis s jedinstvenim identifikatorom. *Backend* je dizajniran tako da pohranjuje listu zadataka u okviru kolekcije "*taskLists*" i pojedinačne zadatke u okviru kolekcije "*tasks*".

Prva ključna komponenta koje je implementirana je klasa *TaskList*. Ova klasa predstavlja listu zadataka, gdje svaki objekt ima svoj jedinstveni identifikacijski broj i naziv. Prilikom pohrane ovih objekata u Firestore, svaki *TaskList* se dodaje kao dokument unutar kolekcije "*taskLists*". Na ovaj način, se lako upravlja listama zadataka, pretražuje i ažurira informacije kada je to potrebno.

Druga komponenta, klasa *Task*, predstavlja pojedinačni zadatak. Svaki zadatak uključuje informacije kao što su naziv liste kojoj pripada, identifikator zadatka, opis, prioritet, kao i datume početka i ispunjenja zadatka. Ovi podaci omogućuju korisnicima da jasno prate i upravljaju svojim zadacima. U Firestore-u, zadaci se pohranjuju kao dokumenti unutar kolekcije "*tasks*". Svaki dokument zadatka može referencirati odgovarajući *TaskList*, čime se stvara veza između zadatka i liste to jest kontekst u kojem se zadatak nalazi.

Za povezivanje zadatka s njegovom listom, koristiti se *listName* kao referenca, ali također se može pohraniti listId kako bi se imao puni identitet veze. Na taj način, backend omogućuje lako upravljanje i pretraživanje, kao i održavanje integriteta podataka.

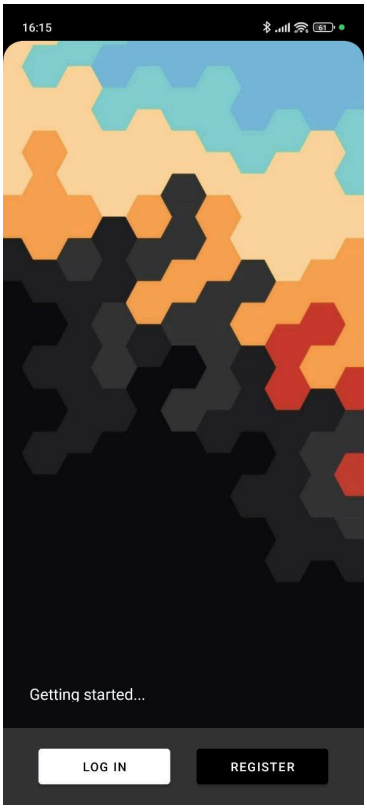
Uz korištenje Firestore-a, podaci su lako dostupni s različitih uređaja i platformi, omogućujući korisnicima da pristupe svojim zadacima bilo kada i bilo gdje. Firebase se također brine o sigurnosti, autentikaciji i kontroli pristupa, čime omogućuje zaštitu senzitivnih informacija i pruža fleksibilnost u razvoju aplikacija.

5. PRIKAZ RADA APLIKACIJE

Ovo poglavlje pruža pregled rada aplikacije kroz vizualne prikaze ključnih ekrana. Sljedeće slike ilustriraju glavne funkcionalnosti aplikacije, uključujući procese prijave, registracije, pregled lista korisnika i prikaz zadataka unutar lista.

5.1. Početni zaslon

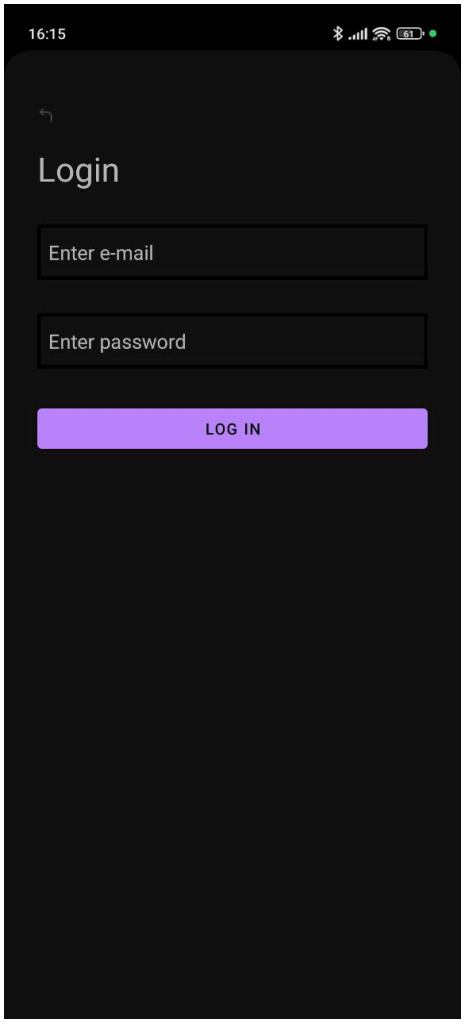
Početni zaslon aplikacije je prvi ekran koji korisnici vide pri pokretanju aplikacije. Slika 5.1. prikazuje početni zaslon s dvije glavne tipke: "Login" i "Register". Ove tipke omogućuju korisnicima da se prijave u postojeći račun ili da se registriraju ako nemaju račun. Ako je korisnik već registriran i prijavljen, aplikacija automatski preusmjerava korisnika na zaslon s listama, bez potrebe za dodatnim radnjama.



Slika 5.1. Početni zaslon

5.2. Zaslon za prijavu korisnika

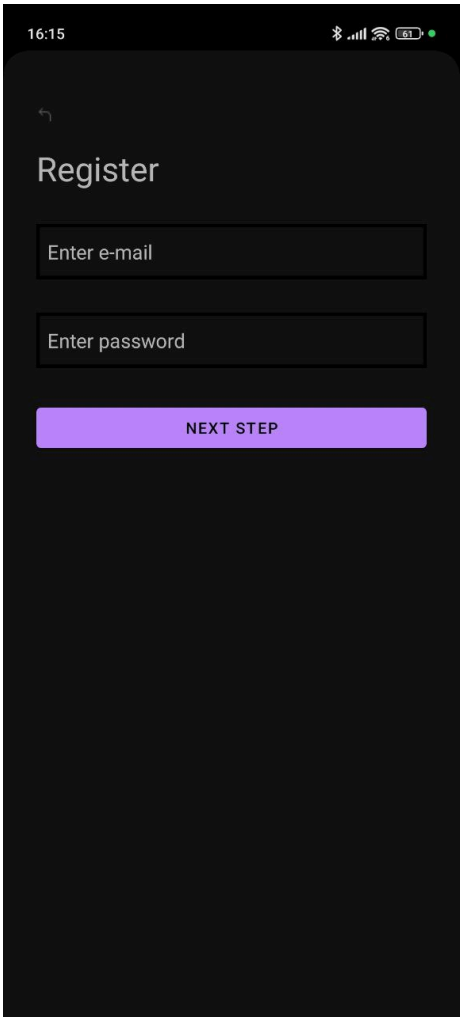
Prvi korak u korištenju aplikacije je prijava. Slika 5.2. prikazuje ekran za prijavu gdje korisnici unose svoje e-mail adrese i lozinke. Ovaj ekran omogućuje autentifikaciju korisnika prije pristupa aplikaciji. Korisnici mogu koristiti opciju za oporavak lozinke ako zaborave svoje podatke za prijavu.



Slika 5.2. Login ekran

5.3. Registracija korisnika

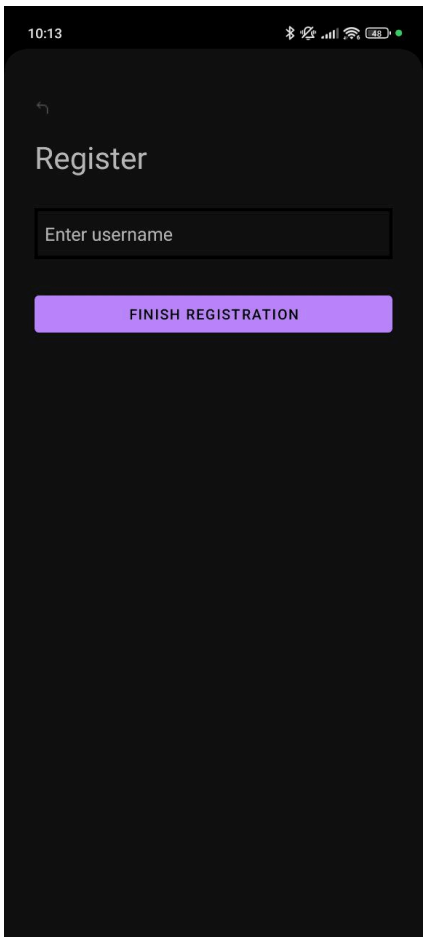
Nakon što korisnici odluče koristiti aplikaciju, trebaju se registrirati. Slika 5.3. prikazuje ekran za registraciju. Ovdje korisnici unose svoje e-mail adrese i lozinke. Registracija uključuje verifikaciju e-mail adrese kako bi se osigurala točnost podataka.



Slika 5.3. Registracija korisnika

5.3.1. Zaslon za dodavanje korisničkog imena

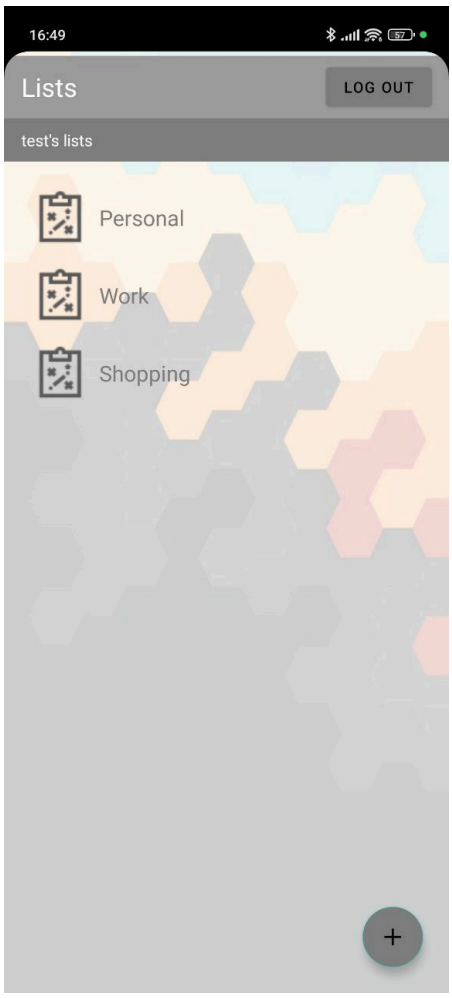
Korisničko ime dodaje se na drugom koraku registracije, ukoliko korisnik ne unese zatraženo korisničko ime pri registraciji traži se unos kod prve prijave. Slika 5.4. predstavlja zaslon unosa korisničkog imena te završavanja registracije.



Slika 5.4. Unos korisničkog imena

5.4. Pregled korisnikovih listi zadataka

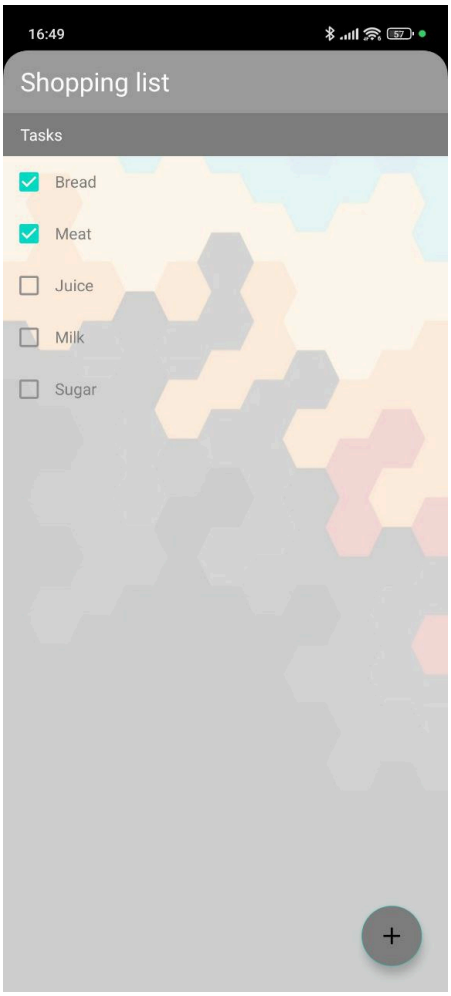
Nakon prijave, korisnici mogu vidjeti sve svoje todo liste na glavnom ekranu aplikacije. Slika 5.5. prikazuje pregled lista zadataka. Ovdje korisnici mogu vidjeti naziv svake liste. Klikom na naziv liste, korisnici mogu pristupiti detaljnom prikazu zadataka unutar te liste.



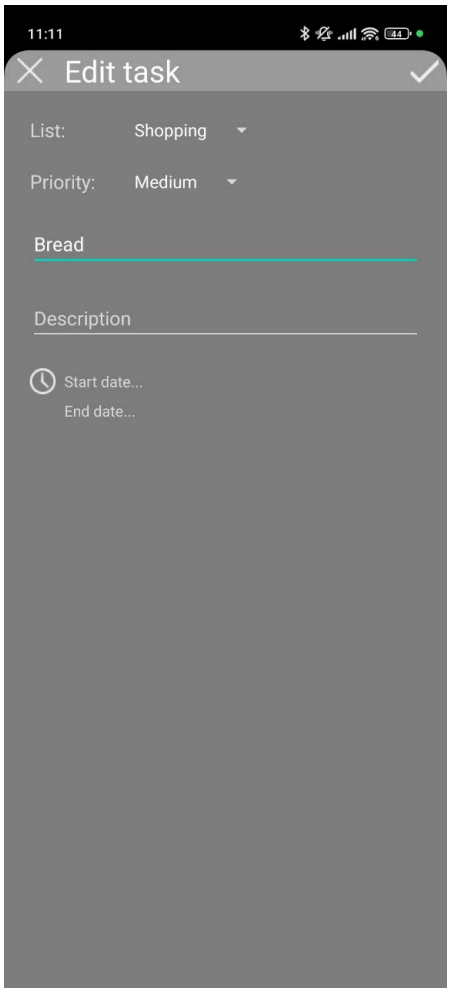
Slika 5.5. Pregled lista zadataka

5.5. Pregled zadataka unutar liste

Kada korisnik odabere određenu listu, otvara se ekran s prikazom svih zadataka unutar te liste. Slika 5.6. ilustrira kako se zadaci unutar liste prikazuju. Na slici 5.7. vidljivi su detalji svakog zadatka, uključujući naziv, status i rok izvršenja.



Slika 5.6. Prikaz zadataka unutar liste



Slika 5.7. Prikaz detalja liste

6. ZAKLJUČAK

Task Planner aplikacija predstavlja alat za upravljanje zadacima koji omogućuje korisnicima organizaciju svojih svakodnevnih obaveza i ciljeva. Kroz jednostavno korisničko sučelje, korisnici mogu lako kreirati liste zadataka, dodavati detalje, postavljati prioritete i pratiti rokove. Integracija s Firebase-om osigurava sigurnu i pouzdanu pohranu podataka, omogućujući pristup informacijama u stvarnom vremenu s bilo kojeg uređaja.

Prilikom izrade aplikacije *Task Planner*, glavni izazov bio je implementacija pouzdane pohrane podataka koja omogućuje korisnicima spremanje i pristup zadacima u stvarnom vremenu, uz sigurnost i dostupnost na različitim uređajima.

Problem je riješen korištenjem Firebase, točnije Firestore baze podataka, koja omogućuje pohranu podataka u kolekcijama i dokumentima. Zadaci su organizirani unutar kolekcija, a svaki zadatak predstavlja dokument. Korištenjem Firebase autentikacije osigurana je privatnost korisničkih podataka, dok je rad u stvarnom vremenu omogućio automatsko ažuriranje i sinkronizaciju podataka.

U današnjem ubrzanom tempu života, organizacija zadataka je ključna za postizanje produktivnosti i efikasnosti. Task Planner ne samo da olakšava upravljanje obvezama, već i motivira korisnike da ostvare svoje ciljeve kroz jasne i dostižne korake.

7. LITERATURA

[1] Tasks: to do list & tasks, https://play.google.com/store/apps/details?id=com.tasks.android, (datum zadnje posjete: 12. lipnja 2022.)

[2] Taskito: To-Do List, Planner, https://play.google.com/store/apps/details?id=com.fenchtose.reflog, (datum zadnje posjete: 12. lipnja 2022.)

[3] TickTick:To-do list & Tasks, https://play.google.com/store/apps/details?id=com.ticktick.task, (datum zadnje posjete: 12. lipnja 2022.)

[4] To Do List, https://play.google.com/store/apps/details?id=com.splendapps.splendo, (datum zadnje posjete: 12. lipnja 2022.)

[5] Todoist: To-Do List & Tasks, https://play.google.com/store/apps/details?id=com.todoist, (datum zadnje posjete: 12. lipnja 2022.)

[6] N. Smyth, Firebase Essentials, Payload Media, USA, 2017.

[7] XML, https://developer.android.com/studio/intro, (datum zadnje posjete: 23. rujna 2024.)

[8] Figma, https://webdesign.tutsplus.com/articles/what-is-figma--cms-32272, (datum zadnje posjete: 23. rujna 2024.)

[9] A. Stroud, Android Database Best Practices, O'Reilly Media, Sebastopol, 2020.

SAŽETAK

Tijekom razvoja aplikacije Task Planner suočili smo se s izazovom implementacije pouzdane i učinkovite pohrane podataka, koja bi korisnicima omogućila spremanje, pristup i ažuriranje zadataka u stvarnom vremenu, uz garantiranu sigurnost i dostupnost na različitim uređajima. Ključni problem bio je osigurati neprekidnu sinkronizaciju podataka među uređajima, tako da promjene koje korisnici naprave budu odmah vidljive.

Rješenje ovog problema pronašli smo u korištenju Firebase platforme, točnije Firestore baze podataka, koja podržava strukturiranu pohranu podataka kroz kolekcije i dokumente. Svaki zadatak unutar aplikacije predstavlja dokument unutar određene kolekcije, što omogućuje jednostavnu organizaciju i dohvaćanje podataka. Firestore podržava rad u stvarnom vremenu, što znači da se sve promjene u podacima automatski sinkroniziraju između korisničkih uređaja bez odgode.

Korištenjem Firebase autentikacije, uspjeli smo osigurati privatnost korisničkih podataka. Svaki korisnik ima svoj jedinstveni račun, a podaci su dostupni samo korisniku kojem pripadaju, što garantira sigurnost i povjerljivost informacija. Na ovaj način, riješili smo ključni izazov, omogućivši sigurnu, pouzdanu i uvijek dostupnu pohranu zadataka, uz automatsku sinkronizaciju i zaštitu podataka.

Ključne riječi: Android, aplikacija, firebase, organizacija

ABSTRACT

Android task scheduling application

During the development of the Task Planner application, we faced the challenge of implementing a reliable and efficient data storage solution that would allow users to save, access, and update tasks in real time, while ensuring security and availability across multiple devices. The key issue was ensuring continuous data synchronization between devices so that any changes made by users would be immediately reflected.

We solved this problem by utilizing the Firebase platform, specifically Firestore, which supports structured data storage through collections and documents. Each task within the app is represented as a document within a specific collection, allowing for easy data organization and retrieval. Firestore supports real-time functionality, meaning all changes to data are automatically synchronized between user devices without delay.

By integrating Firebase authentication, we were able to secure user data privacy. Each user has a unique account, and their data is accessible only to them, ensuring the security and confidentiality of information. In this way, we addressed the main challenge, providing secure, reliable, and always available task storage, along with automatic synchronization and data protection.

Keywords: Android, application, firebase, organization