LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

SLUDINĀJUMU PORTĀLA MOBILĀS APLIKĀCIJAS IZSTRĀDE

KVALIFIKĀCIJAS DARBS

Autors:
Daniils Vitjuks, apl.Nr. dv19041
Darba vadītājs: B. dat. Tairs Rzajevs

RĪGA 2022

ANOTĀCIJA

Darbā ir aprakstīts Flutter lietotnes “FastMarket” izstrādes process, kas sastāvēja no programmatūras dokumentācijas izstrādēs, projektējuma, kā arī pirmkoda izstrādes un testēšanas dokumentācijas izveidošanas un izpildes. “FastMarket” lietotne ļauj lietotājiem izveidot un publicēt savus sludinājumus kā arī atrast citu lietotāju publicētus sludinājumus un filtrēt tos pēc kategorijas. Viena no atšķirošam īpašībām ir čats, kas ļauj sazināties ar konkrēta sludinājuma īpašnieku tieši lietotnē, bez kādas ārējas lietotnes palīdzības. Lietotne tika izstrādāta ar Flutter ietvari, kas ļauj kompilēt to gan uz iOS, gan uz Android platformām, kā datubāze tika izmantots Firebase serviss.

Atslēgas vārdi: Flutter, iOS, Android, Firebase, mobilā lietotne, sludinājumi.

ABSTRACT

Development of advertisement portal mobile application

The paper describes development stages of mobile Flutter application “FastMarket”, which consisted of documentation, design specifications, source code development, as well as testing documentation and testing itself. “FastMarket” application allows users to create and publish their own advertisements in specific categories, as well as browse and filter other user’s published advertisements based on category. One of differentiating factors is the inclusion of in-app chat, which allows users to contact owners of specific advertisements without the need of secondary chat apps. Application was written using Flutter framework, which allows to compile it to both iOS and Android systems, Firebase service was used as an application’s database.

Keywords: Flutter, iOS, Android, Firebase, mobile application, advertisements.

SATURA RADĪTĀJS

IEVADS

Nolūks

Kvalifikācijas darbā izveidotas lietotnes “FastMarket” nolūks ir izveidot platformu, kurā pārdevēji var viegli publicēt sludinājumus par savam precēm un pircēji var ērti apskatīt sev interesējošas preces konkrētajā kategorijā. Lietotne arī atļaus ātru saziņu starp pircēju un pārdevēju izmantojot iebūvētu čata funkciju.

Programmatūras prasības palīdzēs izstrādāt pareizo programmas pirmkodu, kā arī ļaus pārliecināties par visas funkcionalitātes implementāciju lietotnē.

Darbības sfēra

“FastMarket” lietotne ir paredzēta tiem lietotājiem, kuri vēlas pārdot/nopirkt konkrētas kategorijas jaunas vai lietotās preces. Lietotnei nav noteiktas pamatauditorijas, jo tā ir viegli lietojama visām demogrāfiskām grupām.

Lietotne darbosies uz iOS un Android viedierīcēm, uz kurām ir interneta pieslēgums un ir neobligāta pieeja kamerai.

Saistība ar citiem dokumentiem

Dokumenta noformēšanā izmantotas standarta *LVS 68:1996 PPS “Programmatūras Prasību Specifikācijas”*, kā arī standarta *LVS 72:1996 PPS “Ieteicamā prakse programmatūras projektējuma aprakstīšanai”* prasības.

Pārskats

Dokuments sastāvēs no sešām daļām:

Pirmā daļa ietver vispārējo produkta aprakstu, kurā aprakstītas produkta perspektīvas un funkcijas, vispārējie ierobežojumi, pieņēmumi un atkarības.

Otrā daļa ir aprakstītas lietotnes funkcionālas un nefunkcionālas prasības, kuras ietver sevī lietotnes funkciju sadalījumu pa moduļiem un to detalizēts apraksts, kā arī 2. un 1. līmeņa DPD.

Trešā daļa ir datu bāzes entitāšu projektējums, daļējais funkciju projektējums, kā arī nepilns lietotāja saskarņu projektējums.

Ceturtā daļa ir veltīta testēšanas procesa aprakstīšanai, tā saturēs testa metodiku un testēšanas žurnālus.

Piektajā daļā ir aprakstīti pasākumi lietotnes izstrādes kvalitātes nodrošināšanai, kā arī konfigurāciju pārvaldes apraksts.

Sestajā daļā ir izrēķināts projekta darbietilpīgums.

APZĪMĒJUMU SARAKSTS

Flutter – satvars vietējo mobilo lietotņu izstrādei, kas ir bāzēts uz Dart programmēšanas valodas.

DPD – datu plūsmas diagramma.

Android – Google izstrādāta operētājsistēma, kas tiek lietotā mobilajās ierīcēs.

iOS – Apple izstrādāta operētājsistēma "iPhone" viedtālruņiem.

Git – atvērta pirmkoda versiju kontroles sistēma projektu pārvaldībai.

Firebase – Google izstrādāts servisu paka, kas palīdz izveidot un uzturēt mobilās un web lietotnes.

Firebase Cloud Firestore – mākoņu noSQL datubāze, kurai var piekļūt ar programmatūras izstrādes komplektu no mobilajām vai web lietotnēm.

Firebase Storage – failu uzglabāšanas serviss no Firebase servisu pakas.

Firebase Auth – autentifikācijas serviss, kas ir daļa no Firebase servisu pakas un atbild par lietotāju autentifikācijas datu glabāšanu šifrētā veidā.

BLoC – saīsinājums no Business Logic Component, viens no modeļiem Flutter biznesa loģikas un lietotāja saskarnes sadalīšanai.

1. VISPĀRĒJAIS APRAKSTS

1.1. Esošā stāvokļa apraksts

Jau eksistē sludinājuma portāli, viens no kuriem ir populārs "SS.lv", bet tajā nav pieejama čata funkcija, kura atļautu ātru un tiešu saskaru ar pārdevēju neizmantojot e-mail vai tālruni.

1.2. Pasūtītājs

Sistēmai nav pasūtītāja; tā ir aprakstīta pēc studenta iniciatīvas kvalifikācijas darba nolūkos.

1.3. Produkta perspektīva

Nākotnē var tikt izstrādāts web administratora panelis ar ērtu dizaina interfeisu, kurā varētu pārvaldīt eksistējošus sludinājumus, pievienot jaunas sludinājumu kategorijas un pārvaldīt lietotājus. Uz doto brīdi šādas darbības var tikt veiktas tikai ar pirmkoda izmaiņām un Firebase konsoli, kas nav visērtākie varianti.

Nākošajās versijās varētu arī pievienot Google Maps integrāciju sludinājuma lokācijas pievienošanai un attēlošanai, sludinājuma modelis pirmkodā paredzēs tādu jaunievedumu.

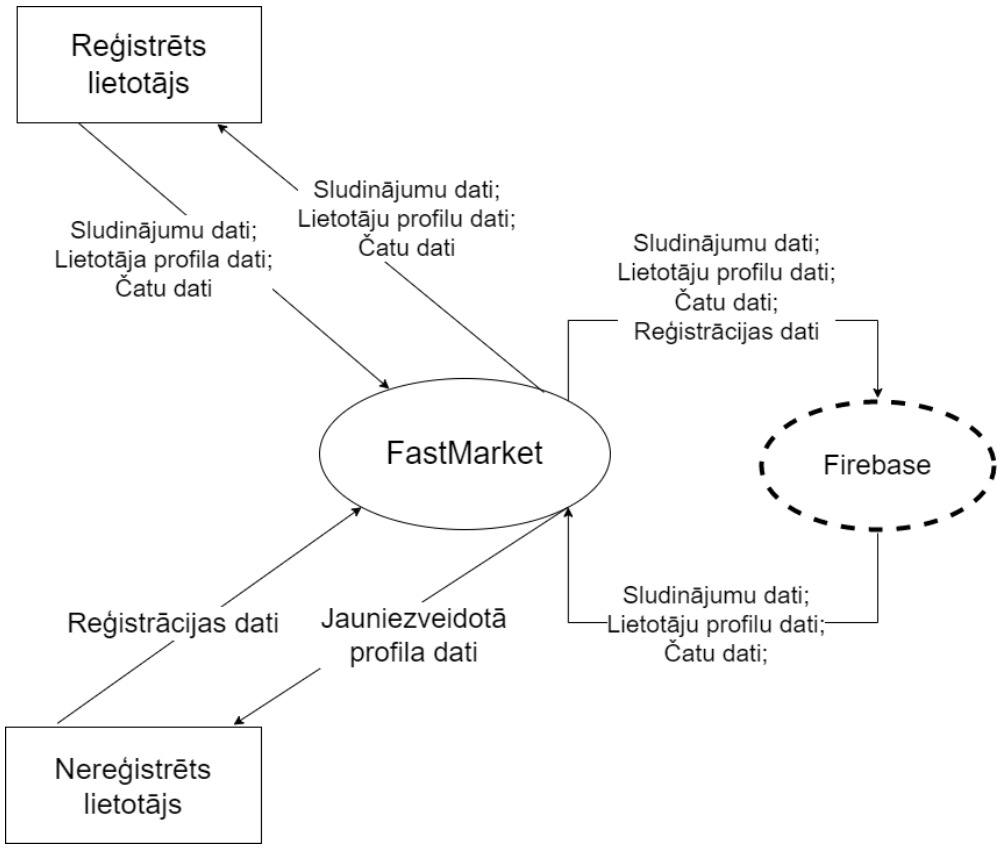
1.4. Darījumprasības

Lietotnes galvenās funkcijas:

- nereģistrētiem lietotājiem:
    - reģistrēties.
- reģistrētiem lietotājiem:
    - pievienot/dzēst sludinājumu;
    - sazināties ar cita sludinājuma īpašnieku
    - apskatīt citu lietotāju sludinājumus
    - filtrēt citu lietotāju sludinājumus pēc kategorijas
    - rediģēt savu sludinājumu;
    - rediģēt savu profila attēlu.

1.5. Sistēmas lietotāji

Sistēmas lietotāji ir nereģistrēti lietotāj un reģistrēti lietotāji



1.1.att. **0. līmeņa DPD diagramma**

1.6. Vispārējie ierobežojumi

Lietotnes darbībai ir vajadzīgs stabils interneta savienojums. Firebase Firestore pieprasījumu tehnisku limitu dēļ nav iespējams filtrēt sludinājumus pēc vairākām kategorijām.

Lietotne neatbild par sludinājuma apraksta un attēla pareizību un atbilstību realitātei.

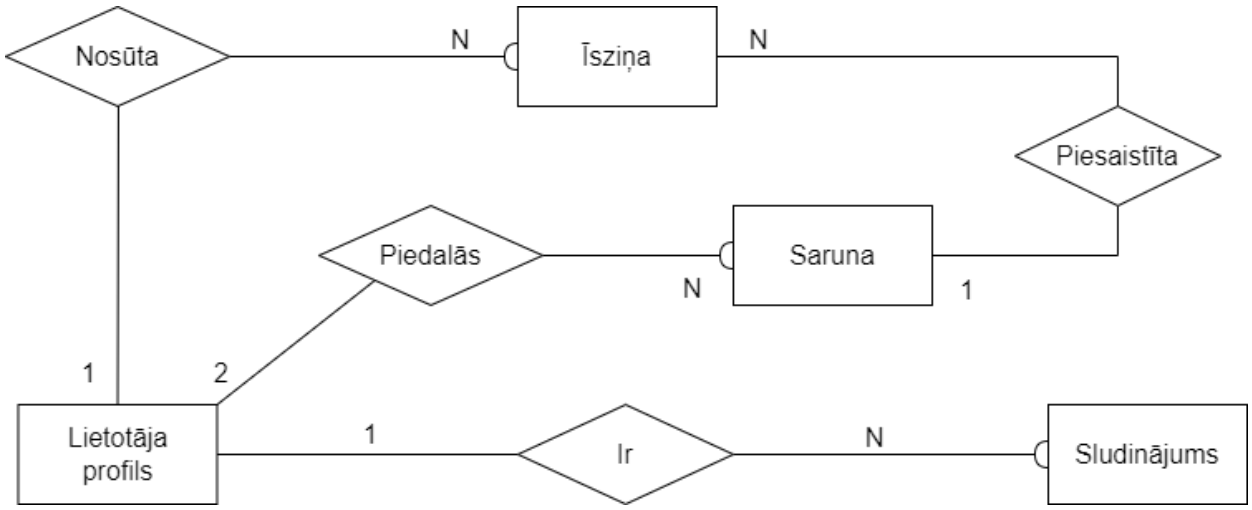
1.7. Pieņēmumi un atkarības

Pievienojot sludinājuma vai profila attēlu lietotājs ir devis atļauju uz failu sistēmas/galerijas/kameras izmantošanu savā operētājsistēmā.

Sistēmas darbībai jābūt atbalstītai uz divām operētājsistēmām: Android un iOS.

2. FUNKCIONĀLAS PRASĪBAS

2.1. Konceptuālais datu bāzes modelis

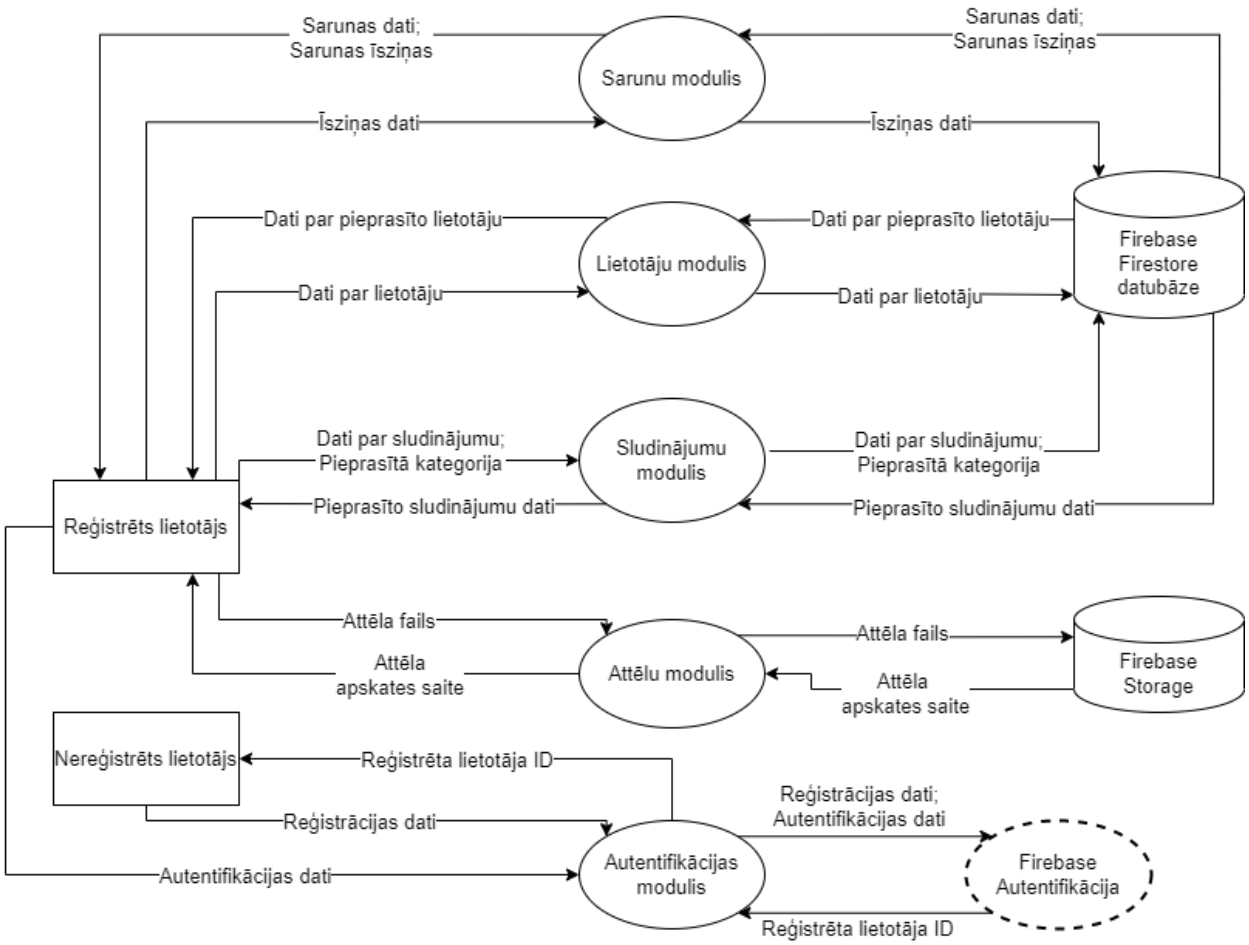


2.1. att. ***Datu bāzes konceptuālā ER modeļa diagramma***

Konceptuālais ER modelis (2.1. att.) attēlo, kā datubāzē glabāsies informācijā par visiem lietotājiem, sludinājumiem un sarunām ar tiem piesaistītiem īsziņām. Lietotājam var būt no 0 līdz N sludinājumi, savukārt sludinājumam obligāti jābūt piesaistītam kādam lietotājām. Sarunā piedalās vienmēr tieši 2 lietotāji un tām ir piesaistīta no 1 līdz N īsziņām. Ka var redzēt, ne visas saites ir obligātas (atzīmētas ar puslodi).

2.2. Funkcionālas prasības

2.2.1. Funkciju sadalījums pa moduļiem



2.2 att. **1. līmeņa DPD**

2.1. tabula ***Funkciju sadalījums pa moduļiem***

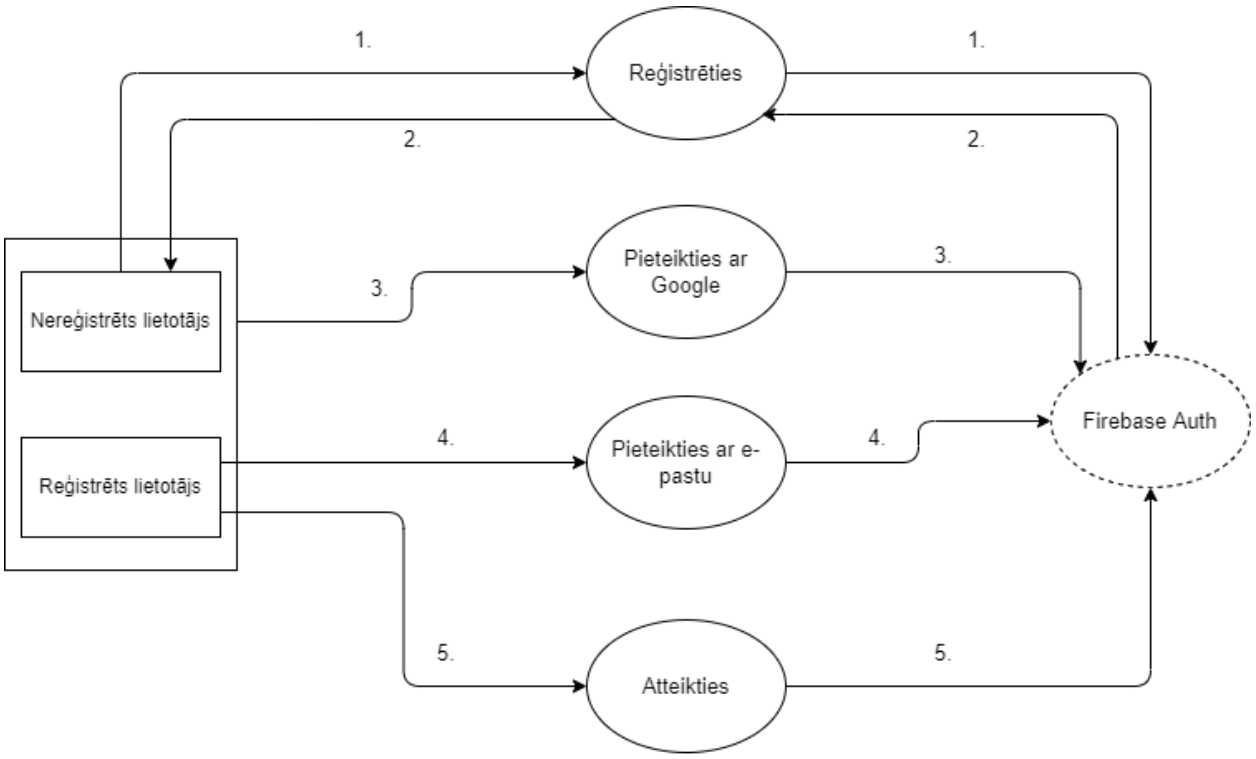
Modulis	**Funkcija**	**Lietotāju grupas**
Autentifikācijas modulis	Reģistrēties	Nereģistrētie lietotāji
	Pieteikties sistēmā (ar e-pastu)	Reģistrētie lietotāji

	Pieteikties sistēmā (ar Google)	Reģistrētie lietotāji, nereģistrētie lietotāji
	Atteikties no sistēmas	Reģistrētie lietotāji
Sludinājumu modulis	Pievienot sludinājumu	Reģistrētie lietotāji
	Dzēst sludinājumu	Reģistrētie lietotāji
	Rediģēt sludinājumu	Reģistrētie lietotāji
	Ielādēt sludinājumus kategorijā	Reģistrētie lietotāji
	Ielādēt manus sludinājumus	Reģistrētie lietotāji
Attēlu modulis	Augšupielādēt profila attēlu	Reģistrētie lietotāji
	Izdzēst profila attēlu	Reģistrētie lietotāji
	Ielādēt sludinājuma attēlu	Reģistrētie lietotāji
	Izdzēst sludinājuma attēlu	Reģistrētie lietotāji
Lietotāju modulis	Izveidot lietotāja profilu	Reģistrētie lietotāji
	Rediģēt lietotāja profilu	Reģistrētie lietotāji

	Saņemt lietotāja profilu no lietotāja ID	Reģistrētie lietotāji
Sarunu modulis	Iesūtīt īsziņu sarunā	Reģistrētie lietotāji
	Izveidot sarunu	Reģistrētie lietotāji
	Ielādēt manas sarunas	Reģistrētie lietotāji
	Ielādēt sarunas īsziņas	Reģistrētie lietotāji

2.2.2. Autentifikācijas modulis

Autentifikācijas modulis atbildēs par lietotnes saiti ar Firebase Auth servisu un lietotāju reģistrāciju, pieteikšanos un atteikšanos no lietotnēs. Tiek paredzētas 2 ierakstīšanas metodes: ar e-pastu un ar Google lietotāja kontu. 2. līmeņa DPD var redzēt 2.3.att.



2.3 att. **2. līmeņa DPD Autentifikācijas modulim**

Apzīmējumi:

1. Reģistrācijas dati (e-pasts, parole);
2. Izveidota lietotāja ID;
3. Google konta pieteikšanas dati;
4. E-pasts, parole;
5. Pieteikta lietotāja ID.

2.2.2.1. Reģistrēties

2.2. Tabula **“Reģistrēties”**

Identifikators
FM_REGISTER
Ievads
Reģistrē lietotāju ar e-pastu un paroli
Ievade
• E-pasts – e-pasta adrese ar pareizo sintaksi; • Parole – teksta virkne vismaz 6 simbolu garumā;
Apstrāde
1. Pārbauda vai e-pasta adrese ir ar pareizo sintaksi. a) Ja e-pasts ir ar nepareizo sintaksi izvada kļūdu paziņojumu [1] 2. Pārbauda vai parole ir vismaz 6 simbolus gara virkne a) Ja parolē ir mazāk nekā 6 simboli izvada kļūdu paziņojumu [2] 3. Nodod reģistrācijas datus FirebaseAuth
Izvade
Ja lietotājs ir veiksmīgi reģistrēts, novirza viņu uz profila izveides lapu, ja FirebaseAuth atgriež kļūdu, attēlo to uz ekrāna
Kļūdu paziņojumi
1. “Enter correct email” 2. “Password should be at least 6 character long”

2.2.2.2. Pieteikties ar e-pastu

2.3. Tabula **“Pieteikties ar e-pastu”**

Identifikators
FM_LOGIN_EMAIL
Ievads

Pieteic reģistrēto lietotāju sistēmā izmantojot e-pastu un paroli
Ievade
• E-pasts – e-pasta adrese ar pareizo sintaksi; • Parole – teksta virkne vismaz 6 simbolu garumā;
Apstrāde
1. Pārbauda vai e-pasta adrese ir ar pareizo sintaksi. a) Ja e-pasts ir ar nepareizo sintaksi izvada kļūdu paziņojumu [1] 2. Pārbauda vai parole ir vismaz 6 simbolus gara virkne a) Ja parolē ir mazāk nekā 6 simboli izvada kļūdu paziņojumu [2] 3. Nodod ierakstīšanas datus FirebaseAuth
Izvade
Ja lietotājs ir veiksmīgi pieteicies, novirza viņu uz lietotnes galveno lapu, ja no FirebaseAuth atgriežas kļūda, attēlo to uz ekrāna
Kļūdu paziņojumi
1. “Enter correct email” 2. “Password should be at least 6 character long”

2.2.2.3. Pieteikties ar Google kontu

2.4. Tabula **“Pieteikties ar Google kontu”**

Identifikators
FM_LOGIN_GOOGLE
Ievads
Pieteic reģistrēto lietotāju sistēmā izmantojot Google kontu, ar paplašinājuma Google Sign- In palīdzību
Ievade
• Google credentials – datu tips, ko atgriež paplašinājums Google Sign-In

Apstrāde
1. Nodod Google credentials datus uz FirebaseAuth
Izvade
Ja lietotājs ir veiksmīgi pieteicies, novirza viņu uz lietotnes galveno lapu, ja no FirebaseAuth atgriežas kļūda, attēlo to uz ekrāna

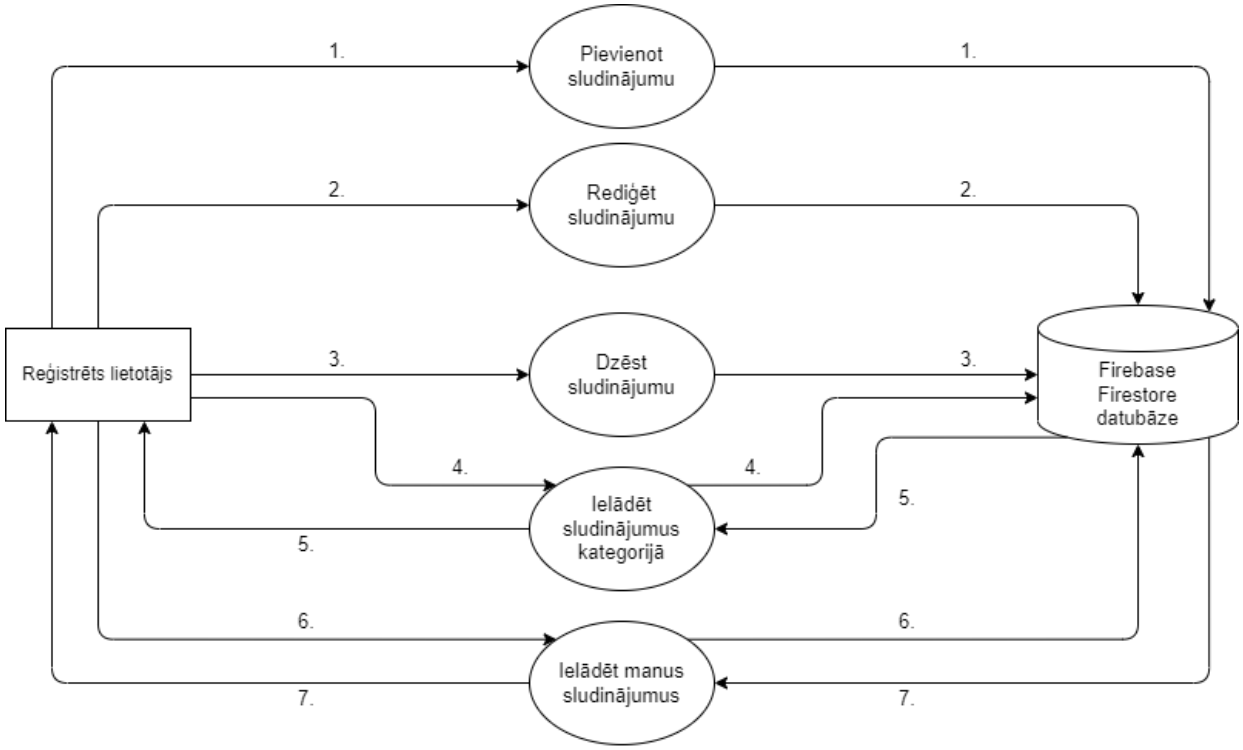
2.2.2.4. Atteikties

2.5. Tabula **“Atteikties”**

Identifikators
FM_LOGOUT
Ievads
Izrakstā ierakstīto lietotāju no sistēmas
Ievade
• User ID – pašlaik pieteikta lietotāja ID
Apstrāde
1. Nodod pieprasījumu atteikt lietotāju ar padotu ID uz FirebaseAuth
Izvade
Ja lietotājs ir veiksmīgi atteicies, novirza viņu uz pieteikšanas lapu, ja no FirebaseAuth atgriežas kļūda, attēlo to uz ekrāna

2.2.3. Sludinājumu modulis

Sludinājumu modulis atbild par visam darbībām ar sludinājumiem (CRUD funkcionalitāte), radīšanas funkcija ir sadalīta uz divām daļām: visi sludinājumi un tikai mani sludinājumi, tas ir izdarīts tāpēc, kā lietotnē būs atsevišķas nodaļas visu sludinājumu apskatīšanai un savējo sludinājumu apskatīšanai. Uz doto brīdi nav paredzēts filtrēt savējos sludinājumus pēc kategorijas, šāda funkcionalitāte ir tikai visu sludinājumu funkcijai. 2.4.att. var redzēt 2. līmeņa DPD sludinājumu modulim.



2.4. att. **2. līmeņa DPD Sludinājumu modulim**

Apzīmējumi:

1. Jauna sludinājuma nosaukums, kategorija, apraksts, lietotāja ID, attēla saite;
2. Rediģēta sludinājuma nosaukums, kategorija, apraksts, lietotāja ID, attēla saite;
3. Dzēšama sludinājuma ID;
4. Izvēlēta kategorija;
5. Pēc kategorijas nofiltrēts sludinājumu saraksts;
6. Pieslēgta lietotāja ID;
7. Pēc lietotāja ID nofiltrēts sludinājumu saraksts.

2.2.3.1. Pievienot sludinājumu

2.6. Tabula **"Pievienot sludinājumu"**

Identifikators
FM_PUBLISH_ADVERT
Ievads
Pievieno jaunu sludinājumu
Ievade
• Nosaukums – simbolu virkne garuma 100, obligāts lauks; • Apraksts – teksts garuma 1000, obligāts lauks; • Kategorija – viena no pieejamam kategorijām, obligāts lauks; • Attēls – saite uz attēlu Firebase Storage (neobligāts)
Apstrāde
1. Pārbauda, vai visi ievadītie dati atbilst nepieciešamo datu formātam. a) Ja neatbilst, atgriež kļūdas paziņojumu [1]. 2. Pārbauda vai visi obligāti lauki ir aizpildīti a) Ja kāds no obligātiem laukiem nav aizpildīts atgriež kļūdu paziņojumu [2]. 3. Nosūt ievadīto informāciju uz Firebase Firestore datu bāzi
Izvade
Ja sludinājums veiksmīgi pievienots datu bāzē, tad atgriež lietotāju uz pagājušo ekrānu
Kļūdu paziņojumi
1. "Incorrect data format!". 2. "Please fill all required fields".

2.2.3.2. Rediģēt sludinājumu

2.7. Tabula **"Rediģēt sludinājumu"**

Identifikators
FM_EDIT_ADVERT
Ievads

Rediģē sludinājumu
Ievade
• Nosaukums – simbolu virkne garuma 100, obligāts lauks; • Apraksts – teksts garuma 1000, obligāts lauks; • Kategorija – viena no pieejamam kategorijām, obligāts lauks; • Attēls – saite uz attēlu Firebase Storage (neobligāts)
Apstrāde
1. Pārbauda, vai visi ievadītie dati atbilst nepieciešamo datu formātam. a) Ja neatbilst, atgriež kļūdas paziņojumu [1]. 2. Pārbauda vai visi obligāti lauki ir aizpildīti a) Ja kāds no obligātiem laukiem nav aizpildīts atgriež kļūdu paziņojumu [2]. 3. Nosūt ievadīto informāciju uz Firebase Firestore datu bāzi
Izvade
Ja sludinājums veiksmīgi rediģēts datu bāzē, tad atgriež lietotāju uz pagājušo ekrānu
Kļūdu paziņojumi
1. “Incorrect data format!”. 2. “Please fill all required fields”.

2.2.3.3. Dzēst sludinājumu

2.8. Tabula **“Dzēst sludinājumu”**

Identifikators
FM_DELETE_ADVERT
Ievads
Izdzēš sludinājumu
Ievade
• Sludinājuma ID – unikāls vesels skaitlis, netiek pa tiešo ievadīts, bet ņemts no atvērta sludinājuma skata;

Apstrāde
1. Pārbauda, vai izvēlēta Sludinājuma īpašnieka ID ir pašlaik autentificētā lietotāja ID a) Lietotāja ID nesakrīt ar Sludinājuma īpašnieka atgriež kļūdu paziņojumu [1] b) Lietotāja ID sakrīt ar Sludinājuma īpašnieka ID, turpina uz punktu 2. 2. Izsauc funkciju FM_DELETE_ADVERT_IMAGE lai izdzēstu Firebase Storage glabājamo sludinājuma attēlu (ja tāds eksistē) 3. Izdzēš sludinājumu no Firebase Firestore datubāzes
Izvade
Ja sludinājums veiksmīgi izdzēsts no datubāzes, tad atgriež lietotāju uz pagājušo ekrānu un atjauno sludinājumu sarakstu
Kļūdu paziņojumi
1. “You are not the owner of this advertisement”

2.2.3.4. Ielādēt sludinājumus kategorijā

2.9. Tabula **“Ielādēt sludinājumus kategorijā”**

Identifikators
FM_GET_ADVERTS
Ievads
Ielādē sludinājumu sarakstu pēc izvēlētas kategorijas
Ievade
• Kategorija – viena no pieejamam kategorijām
Apstrāde
Izveido Firestore pieprasījumu ar izvelēto kategoriju, ja kategorija nav izvēlēta, atgriež sludinājumus no visām kategorijām.
Izvade
Skats ar lietotāja izvelētas kategorijas sludinājumiem
Kļūdu paziņojumi

NAV

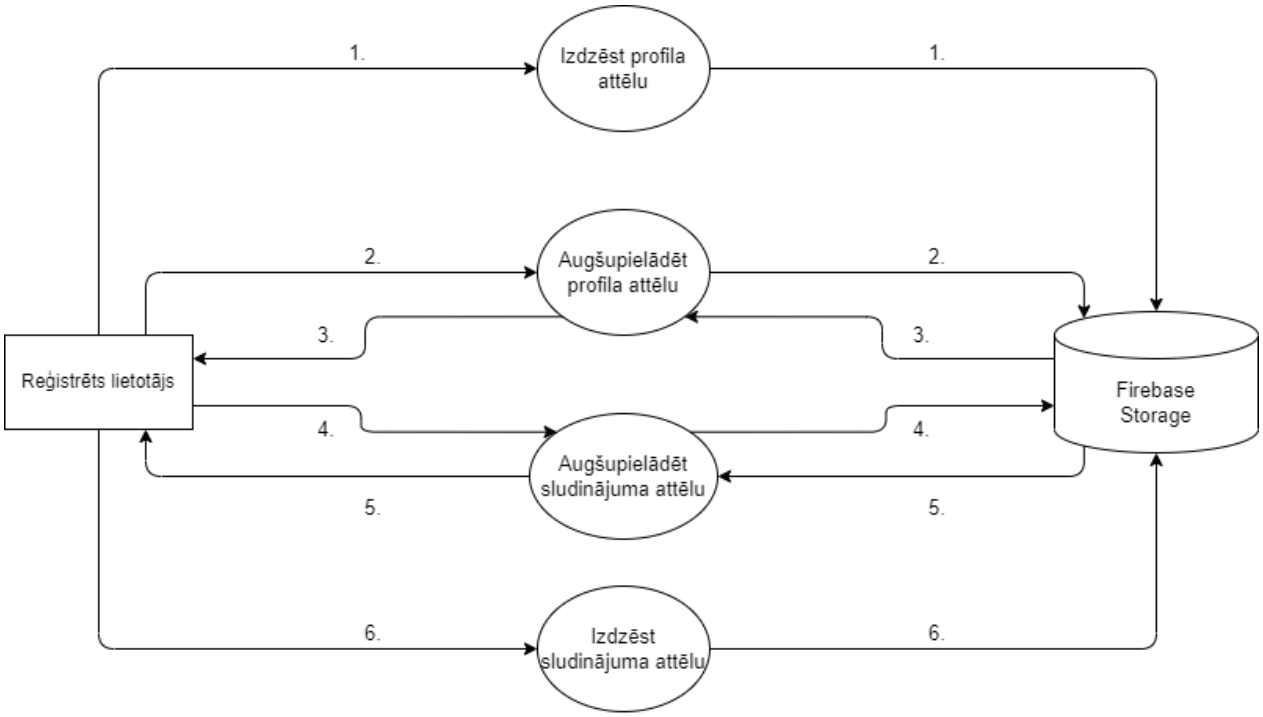
2.2.3.5. Ielādēt manus sludinājumus

2.10. Tabula **“Ielādēt manus sludinājumus”**

Identifikators
FM_GET_MY_ADVERTS
Ievads
Ielādē pieteiktā lietotāja sludinājumus
Ievade
NAV
Apstrāde
1. Saņem no lietotnes pieteiktā lietotāja ID 2. Izveido pieprasījumu uz Firestore kur sludinājuma īpašnieka ID ir pašlaik pieteiktā lietotāja ID
Izvade
Skats ar lietotāja sludinājumiem
Kļūdu paziņojumi
NAV

2.2.4. Attēlu modulis

Attēlu modulis atbild par savienojumu ar Firebase Storage attēlu glābšanai. Funkcijas ir projektētas lietotāju profila attēlu un sludinājumu attēlu apstrādei, t.i. augšupielādēšana un izdzēšana no Firebase Storage. 2. līmeņa DPD var redzēt 2.5.att.



2.5. att. **2. līmeņa DPD Attēlu modulim**

Apzīmējumi:

1. Pašlaik pieteiktā lietotāja ID;
2. Profila attēla fails;
3. Piekļuves saite augšupielādētām profila attēlam, faila nosaukums;
4. Sludinājuma attēla fails;
5. Piekļuves saite augšupielādētām sludinājuma attēlam, faila nosaukums.

2.2.4.1. Augšupielādēt profila attēlu

2.11. Tabula **“Augšupielādēt profila attēlu”**

Identifikators
FM_UPLOAD_PROFILE_IMAGE
Ievads
Augšupielādē lietotāja izvēlēto profila attēlu uz Firebase Storage datu glabātuvi
Ievade
Attēla fails – attēls png/jpg formātā
Apstrāde
1. Ģenerē nejauši izveidotu faila nosaukumu 2. Augšupielāde izvēlēto attēlu uz Firebase Storage attiecīgā lietotāja mapē ar tik ko uzģenerēto faila nosaukumu 3. Pieprasa Firebase Storage tiešo saiti uz attēla glabāšanas vietu
Izvade
Funkcijas darbība neietekmē nevienu no skatiem (to, ko redz lietotājs), bet atgriež tik ko augšupielādētā faila nosaukumu un saiti uz attēlu
Kļūdu paziņojumi
NAV

2.2.4.2. Izdzēst profila attēlu

2.12. Tabula **“Izdzēst profila attēlu”**

Identifikators
FM_DELETE_PROFILE_IMAGE
Ievads

Izdzēš lietotāja profila attēlu no Firebase Storage
Ievade
Lietotāja ID - netiek ievadīts, bet paņemts lietotnē no pašlaik pieteiktā lietotāja
Apstrāde
1. No Firebase Storage atrod failu lietotājam attiecīgajā mapē 2. Salīdzina pieteiktā lietotāja profila attēla faila nosaukumu ar atrasto failu Firebase Storage a) Ja profila attēla faila nosaukums nesakrīt ar Firebase Storage atrasto attēlu, izvada kļūdas paziņojumu [1] 3. Izdzēš failu no Firebase Storage
Izvade
NAV
Kļūdu paziņojumi
1. “The image you are trying to delete does not exist!”

2.2.4.3. Augšupielādēt sludinājuma attēlu

2.13. Tabula **“Augšupielādēt sludinājuma attēlu”**

Identifikators
FM_UPLOAD_ADVERT_IMAGE
Ievads
Augšupielādē lietotāja izvēlēto sludinājuma attēlu uz Firebase Storage datu glabātuvi
Ievade

Attēla fails – attēls png/jpg formātā
Apstrāde
1. Ģenerē nejauši izveidotu faila nosaukumu 2. Augšupielāde izvēlēto attēlu uz Firebase Storage attiecīgā sludinājuma mapē ar tik ko uzģenerēto faila nosaukumu 3. Pieprasa Firebase Storage tiešo saiti uz attēla glabāšanas vietu
Izvade
Funkcijas darbība neietekmē nevienu no skatiem (to, ko redz lietotājs), bet atgriež tik ko augšupielādētā faila nosaukumu un saiti uz attēlu
Kļūdu paziņojumi
NAV

2.2.4.4. Izdzēst sludinājuma attēlu

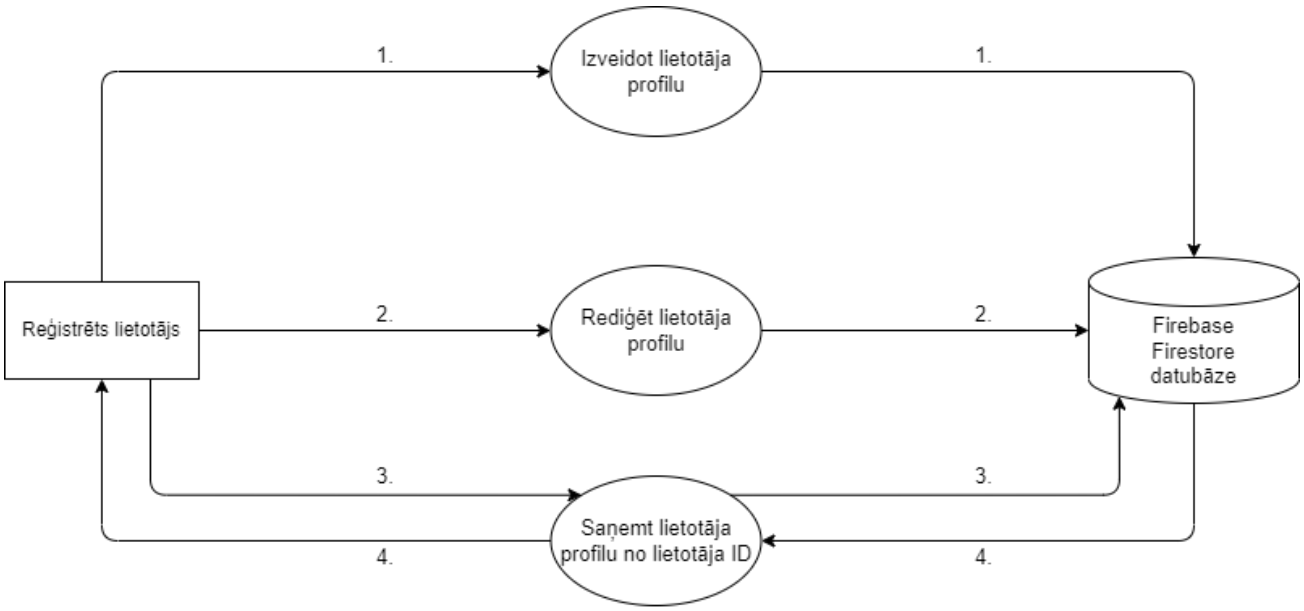
2.14. Tabula **"Izdzēst sludinājuma attēlu"**

Identifikators
FM_DELETE_ADVERT_IMAGE
Ievads
Izdzēš sludinājuma attēlu no Firebase Storage
Ievade
Sludinājuma ID - netiek ievadīts, bet paņemts lietotnē no atvērta sludinājuma
Apstrāde

1. No Firebase Storage atrod failu sludinājumam attiecīgajā mapē 2. Salīdzina atvērta sludinājuma attēla faila nosaukumu ar atrasto failu Firebase Storage a) Ja sludinājuma attēla faila nosaukums nesakrīt ar Firebase Storage atrasto attēlu, izvada kļūdas paziņojumu [1] 3. Izdzēš failu no Firebase Storage
Izvade
NAV
Kļūdu paziņojumi
1. “The image you are trying to delete does not exist!”

2.2.5. Lietotāju modulis

Lietotāju modulis atbild par lietotāju profilu izveidošanu un rediģēšanu, ka arī par citu lietotāju profila informācijas atgūšanas pēc ID (ir vajadzīgs piemērām sludinājuma vai sarunas skatā). Ir svarīgi saprast, ka lietotāju modulis ir atdalīts no autentifikācijas moduļa, tāpēc šeit nenotiks darbības ar parolēm un to apstrādi. Lietotāju moduļa 2. līmeņa DPD var aplūkot 2.6.att.



2.6. att. **2. līmeņa DPD Lietotāju modulim**

Apzīmējumi:

1. Jauna profila dati: Vārds, Uzvārds, e-pasts, attēla saite, lietotāja ID;
2. Rediģējama profila dati: Vārds, Uzvārds, e-pasts, attēla saite, telefona Nr.;
3. Pieprasīta lietotāja ID;
4. Pieprasīta lietotāja profila dati: Vārds, Uzvārds, attēla saite, e-pasts.

2.2.5.1. Izveidot lietotāja profilu

*2.15. Tabula "***Izveidot lietotāja profilu***"*

Identifikators
FM_CREATE_USER_PROFILE
Ievads
Izveido lietotāja profilu pēc veiksmīgas reģistrācijas ar e-pastu vai pirmās pieteikšanās ar Google kontu
Ievade
• Lietotāja ID – tiek paņemts no pagājuša reģistrācijas skata • Vārds – tiek ievadīts, obligāts lauks • Uzvārds – tiek ievadīts, obligāts lauks • E-pasts – tiek paņemts no pagājuša reģistrācijas skata • Profila attēls – saite uz profila attēlu, tiek padota no Google credentials vai tiek atstāta tukša
Apstrāde
1. Pārbauda, vai visi obligātie lauki ir ievadīti a) Ja nav ievadīts kaut viens obligāts lauks, izvada kļūdas paziņojumu [1]. 2. Izveido jaunu Firebase Firestore profilu ar ievadītiem datiem
Izvade
Ja profils ir izveidots veiksmīgi, nosūta lietotāju uz lietotnes galveno skatu.
Kļūdu paziņojumi
1. "Please fill all required fields"

2.2.5.2. Rediģēt lietotāja profilu

2.16. Tabula **"Rediģēt lietotāja profilu"**

Identifikators
FM_EDIT_USER_PROFILE

Ievads
Rediģē lietotāja profilu
Ievade
• Lietotāja ID – tiek paņemts no pašreizēja pieteiktā lietotāja • Vārds – tiek ievadīts, nav obligāts • Uzvārds – tiek ievadīts, nav obligāts • E-pasts – tiek ievadīts, nav obligāts • Profila attēls – saite uz profila attēlu (atgriežas no FM_ULPOAD_PROFILE_PICTURE), nav obligāts
Apstrāde
1. Atjauno lietotāja ierakstu ar izvelēto lietotāja ID nosūtot Firebase Firestore jaunus datus (ja kādi lauki netika ievadīti, tie netiek izmainīti)
Izvade
Ja profils ir veiksmīgi rediģēts, atgriež lietotāju uz sava profila skatu
Kļūdu paziņojumi
NAV

2.2.5.3. Saņemt lietotāja profilu no lietotāja ID

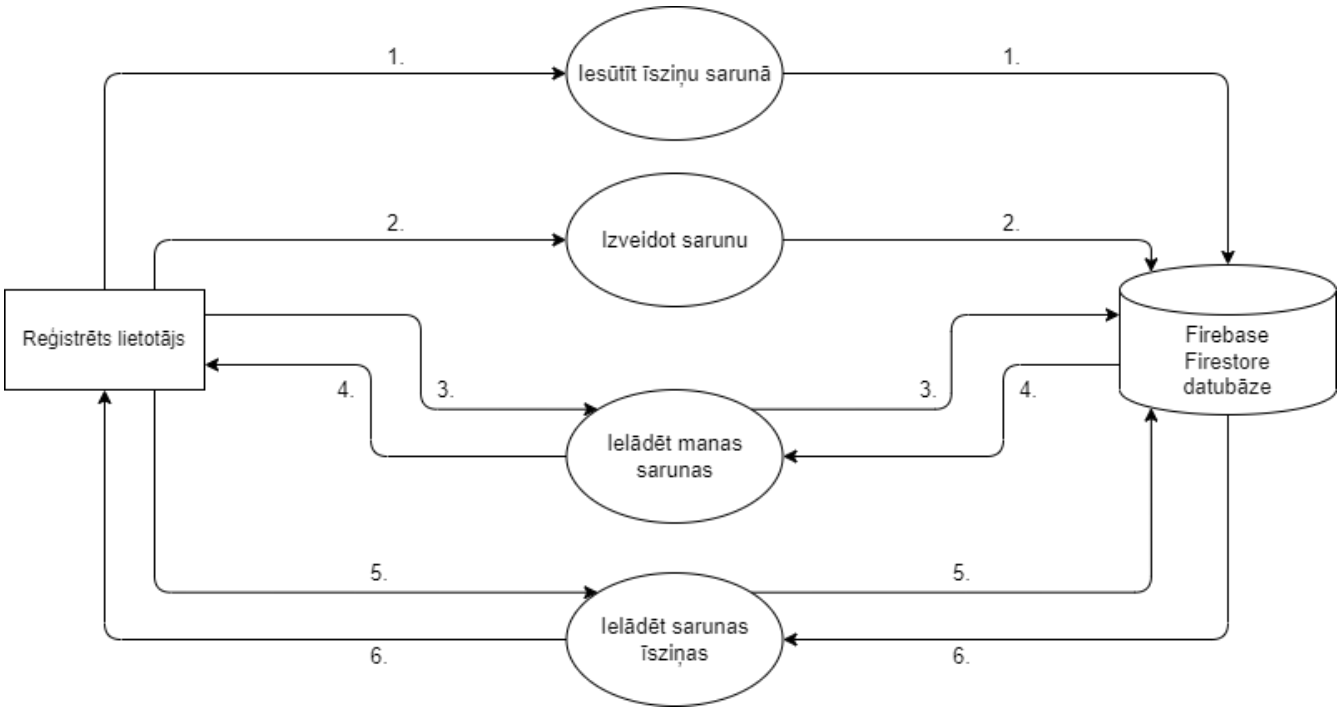
2.17. Tabula **“Saņemt lietotāja profilu no lietotāja ID”**

Identifikators
FM_GET_USER_PROFILE_BY_ID
Ievads
Saņem profila datus lietotājam ar izvēlēto ID.

Ievade
• Lietotāja ID – pieprasīta lietotāja ID
Apstrāde
1. Izveido Firebase Firestore pieprasījumu atgriezt profilu, kura ID ir ievadīts lietotāja ID 2. Pārbauda vai profils eksistē a) Ja profils neeksistē, izvada kļūdu paziņojumu [1]. b) Ja profils eksistē, atgriež tā informāciju.
Izvade
Atgriež atrasta profila pieejamo informāciju (Vārds, Uzvārds, attēla saite, e-pasts)
Kļūdu paziņojumi
1. “User with that ID doesn’t exist”

2.2.6. Sarunu modulis

Sarunu modulis atbild par saziņu starp lietotājiem, t.i. izveido sarunas, iesūta īsziņas konkrētajā sarunā, kā arī atgriež lietotāja sarunu dinamisko straumi un atvērto sarunu īsziņu dinamisko straumi. Straumes tiks izmantotas lai pēc katras informācijas atjaunošanas neveikt manuālu atjaunošanu lietotnē. Sarunu moduļa 2. līmeņa DPD var redzēt 2.7.att.



2.7. att. **2. līmeņa DPD Sarunu modulim**

Apzīmējumi:

1. Īsziņas dati: Teksts, Sūtītāja ID, Sūtīšanas Laiks;
2. Sarunas dalībnieku dati: Dalībnieku ID, dalībnieku pilni vārdi, dalībnieku profila attēlu saites;
3. Pašlaik pieteiktā lietotāja ID;
4. Straume ar sarunām, kuros pašlaik pieteiktais lietotājs ir iesaistīts;
5. Atvērtas sarunas ID;
6. Straume ar atvērtas sarunas īsziņām.

2.2.6.1. Iesūtīt īsziņu sarunā

2.18. Tabula **“Iesūtīt īsziņu sarunā”**

Identifikators
FM_SEND_CHAT_MESSAGE
Ievads
Nosūta īsziņu izvelētajā sarunā
Ievade
• Sarunas ID – tiek paņemts no pašreizēji atvērtas sarunas ID • Īsziņas teksts – tiek ievadīts, obligāts • Īsziņas laiks – DateTime, tiek ģenerēts nosūtīšanas brīdī • Sūtītāja ID – pieteiktā lietotāja ID
Apstrāde
1. Izveido datu objektu ChatMessage ar ievadīto informāciju. 2. Ievieto objektu atvērtā sludinājuma Firestore kolekcijā “messages”. 3. Atjauno atvērta sludinājuma datus Firestore datubāzē, nomainot lauku “lastMessage” uz izveidotas īsziņas tekstu un lauku “lastMessageTimestamp” uz izveidotas īsziņas laiku, kā arī uzstāda lauku “hasMessages” uz true.
Izvade
Īsziņa pēc nosūtīšanas parādās atvērtas sarunas īsziņu saraksta apakšpusē, teksta ievades lauks tiek notīrīts.
Kļūdu paziņojumi
NAV

2.2.6.2. Izveidot sarunu

2.19 . Tabula **“Izveidot sarunu”**

Identifikators

FM_CREATE_CHAT_ROOM
Ievads
Izveido sarunu starp diviem lietotājiem
Ievade
• Sarunas dalībnieku pilni vārdi – masīvs no 2 teksta virknēm, kurā tiek ierakstīti dalībnieku vārda un uzvārda konkatenācija • Sarunas dalībnieku ID – masīvs no 2 lietotāju ID • Sarunas dalībnieku profila attēla saites – masīvs no 2 teksta virknēm ar sarunas dalībnieku profila attēliem • Sarunas ID – tiek izveidots konkatenācijas veida ar 2 lietotāju ID un "_" pa vidu
Apstrāde
1. Nosaka, kura lietotāja ID ir "mazāks" alfabētiski. 2. Izveido sarunas ID no vispirms mazāka ID, tad "_" un tad lielāka ID (ar konkatenāciju) 3. Aizpilda masīvus, uz 0 pozīcijas informācija lietotājam ar mazāku ID, uz 1 pozīcijas informācija lietotājam ar lielāku ID. 4. Aizpildītu un izveidoto informāciju nosūta uz Firebase Firestore kolekciju "chatRooms"
Izvade
Pēc sarunas izveidošanas atver tukšo sarunas skatu, kurā vēl nav iesūtīto īsziņu un ir tikai īsziņas ievades lauks.
Kļūdu paziņojumi
NAV

2.2.6.3. Ielādēt manas sarunas

2.20 . Tabula **"Ielādēt manas sarunas"**

Identifikators

FM_LOAD_MY_CHAT_ROOMS
Ievads
Parāda sarunas, kurās ir iesaistīts pieteikts lietotājs
Ievade
• Lietotāja ID – pašlaik pieteiktā lietotāja ID
Apstrāde
1. Izveido pieprasījumu uz Firebase Firestore, lai atgriež sarunas, kuru lauka "participatingUserIds" masīvs satur pašlaik pieteiktā lietotāja ID un sakārtotu pēc lastMessageTimestamp. 2. Atgriež Firebase Firestore straumi ar izveidoto pieprasījumu
Izvade
Lietotne saņem atpakaļ nevis vienreizējo datu kopumu, bet Firestore straumi, kas atļauj dinamiski atjaunot sarunu sarakstu, kā arī parādīt katras sarunas pēdējo īsziņu un tās nosūtīšanas laiku.
Kļūdu paziņojumi
NAV

2.2.6.4. Ielādēt sarunas īsziņas

2.21 . Tabula **"Ielādēt sarunas īsziņas**

Identifikators
FM_LOAD_CHAT_ROOM_MESSAGES
Ievads
Parāda atvērtas sarunas visas īsziņas
Ievade

• Sarunas ID – atvērtas sarunas ID.
Apstrāde
1. Izveido Firebase Firestore pieprasījumu uz atvērtas sarunas kolekciju “messages”, sakārtotu pēc lauka “sentAt” 2. Atgriež Firebase Firestore straumi ar izveidoto pieprasījumu
Izvade
Lietotne saņem atpakaļ Firestore straumi, kas atļauj dinamiski atjaunot īsziņu sarakstu atvērtajā sarunā, neizdarot papildus Firebase pieprasījumus.
Kļūdu paziņojumi
NAV

2.3. Nefunkcionālas prasības

2.3.1. Lietojamas platformas

Lietotne tiek paredzētā gan iOS, gan Android platformu viedierīcēm. To atļauj tas, ka lietotne tiks izstrādāta ar Flutter tehnoloģijas palīdzību, kas ļaus lietotnei kompilēties abām mobilajām platformām.

2.3.2. Pieejamība

Lietotnei jābūt pieejamai 24 stundas 7 dienas nedēļā, savukārt tas ir iespējams tikai ar interneta pieslēgumu, jo pamata lietotnes informācija glabājas Firebase servera mākonī. Lai pievienot attēlu, lietotājam jānodrošina lietotnes pieeja failu sistēmai/galerijai/kamerai atšķirība no tā, ko izvēlas lietotājs.

2.3.3. Veiktspēja

Datubāzes izmaiņas parādīsies aplikācija pēc pieprasījuma (gadījuma kad nav izmantotas straumes) vai arī automātiski <1s laikā pēc izmaiņām datubāze (kad tiek izmantotas straumes, piemēram īsziņu attēlošanai sarunā).

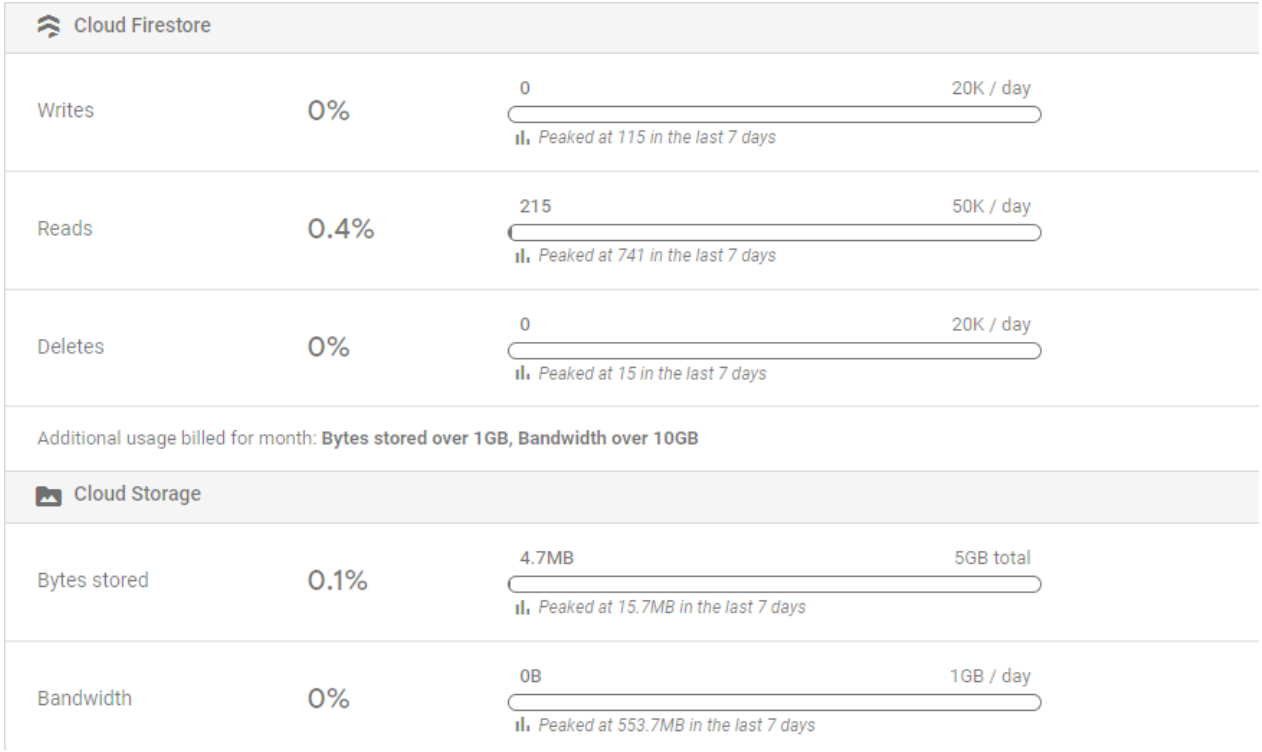
2.3.4. Drošība

Lietotnē tiks izmantoti Firebase servisi, t.s. Firebase Autentifikācija, Firebase Storage un Firebase Firestore. Visos no minētājiem servisiem ir iebūtās drošības noteikumi, kas atļauj limitēt neautorizētu lietotāju piekļuvi datiem. Firebase Auth glābās reģistrētu lietotāju paroles šifrētājā veidā, kas pat cilvēki ar piekļuvi Firebase konsolei nevarēs redzēt to nešifrētus variantus.

Firebase Storage un Firebase Firestore ir paredzēti speciāli “noteikumi” kas nosaka kas var un kas nevar piekļūt konkrētiem datu bāzes ierakstiem. Piemēram piekļuve sludinājumu datubāzei ir tikai autentificētiem lietotājiem, bet izdzēst savu profila attēlu no Firebase Storage var tikai profila autentificēts īpašnieks.

2.3.5. Uzturamība

Lietotnē tiks izmantoti Firebase servisi, kuri piedāvā bezmaksas plānu, kura limitus var redzēt 2.8.att. Gadījumā ja lietotāju skaits palielināsies un limiti tiks pārsniegti, to varēs viegli pārslēgt uz "Spark" plānu, kurš piedāvā mērāmo projekta paplašinājumu ar fiksēto cenu par N vienībām (piemērām 0.14$/1GB Firebase Cloud Storage).



2.8. att. **Firebase servisu limiti ar bezmaksas plānu**

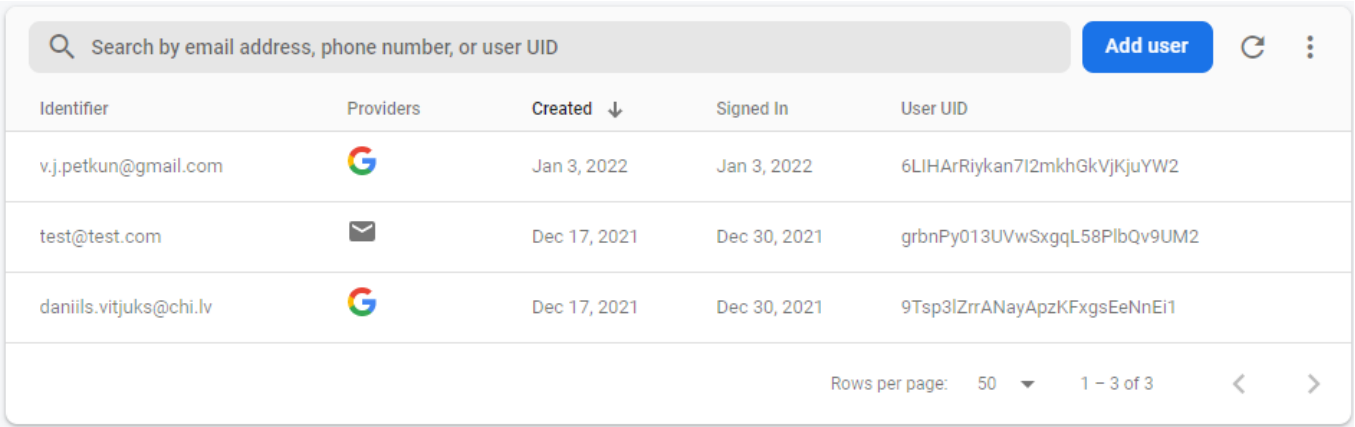
3. PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS

3.1. Datu bāzes projektējums

Kā lietotnes datu bāze tika izvēlēta Firebase Firestore – mākoņu bāzētā NoSQL tipa datubāze. Ieraksti tajā glabājas ļoti līdzīgi JSON formātam, kas atļauj ātru piekļuvi datiem, to nolasīšanai un ierakstīšanai.

Datu bāze tika sadalīta 3 pirmā līmeņa kolekcijas: “users”, “chatRooms”, “adverts”. Kā arī ir viena otrā līmeņa kolekcija (tas nozīmē, ka katrs no kādas pirmā līmeņa kolekcijas ierakstiem sevī iekšā saturēs veselu kolekciju) – “messages” kolekcija, kas atradīsies iekšā kolekcijas “chatRooms” ierakstiem.

Par lietotāju autentifikācijas datu uzglabāšanu atbildēs Firebase Auth serviss, kuru ir nolemts neaprakstīt šajā nodaļā, jo tas netiek pa tiešo izmantots kā datu bāze, bet vairāk kā serviss. Var tikai piebilst, ka reģistrācijas laikā Firebase Auth izveido lietotājam unikālo UID, kurš tiek izmantots kā lietotāja identifikators Firestore datubāzes kolekcijās. Kā izskatās Firebase Auth konsole var redzēt 3.1. att.



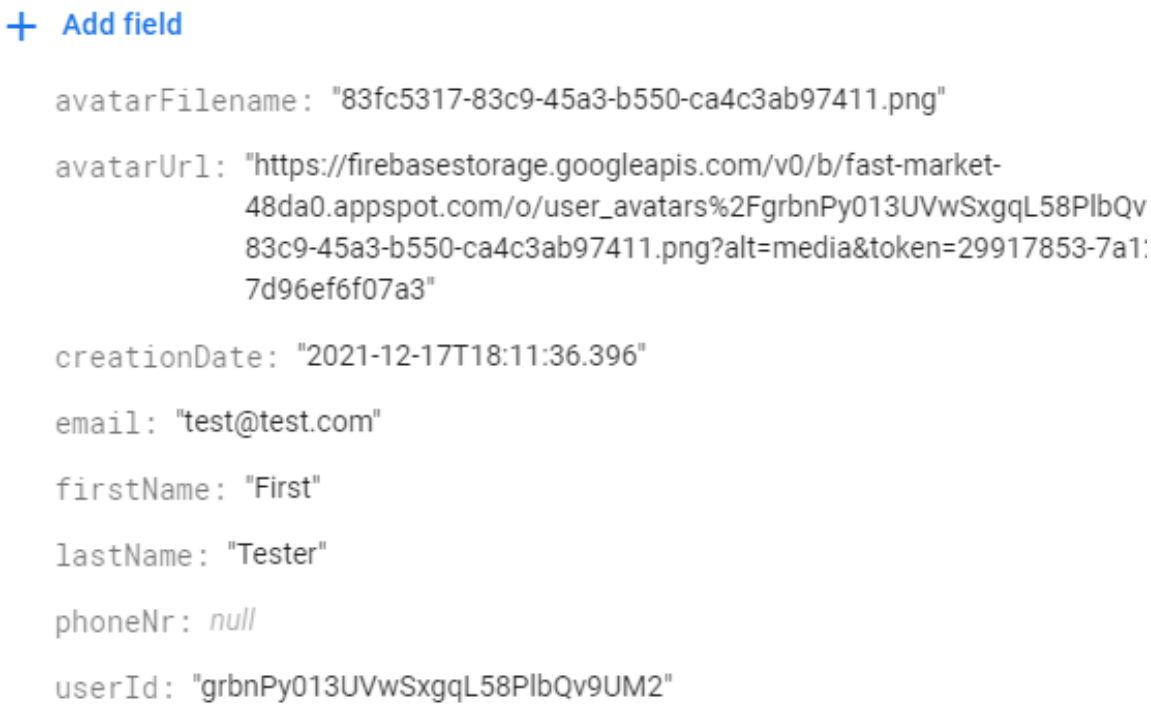
3.1. att. **Firebase autentifikācijas konsole ar lietotāju sarakstu**

“users” kolekcijas ieraksti saturēs sevī laukus:

- userId – lietotāja ID, kas tiks ņemts no Firebase Auth
- email – lietotāja izmantots e-pasts
- creationDate – lietotāja profila izveidošanas datums, tiek ņemts no Firebase Auth
- firstName – lietotāja vārds
- lastName – lietotāja uzvārds
- avatarUrl – saite uz augšupielādētu Firebase Storage lietotāja profila attēlu, var būt NULL

- avatarFilename – augšupielādēta profila attēla faila nosaukums, var būt NULL
- phoneNr – lietotāja pievienots telefona Nr., var būt NULL

Piemēru no Firestore datubāzes "users" kolekcijas ieraksta var redzēt 3.2.att.



3.2. att. **Firestore datubāzes kolekcijas "users" ieraksts**

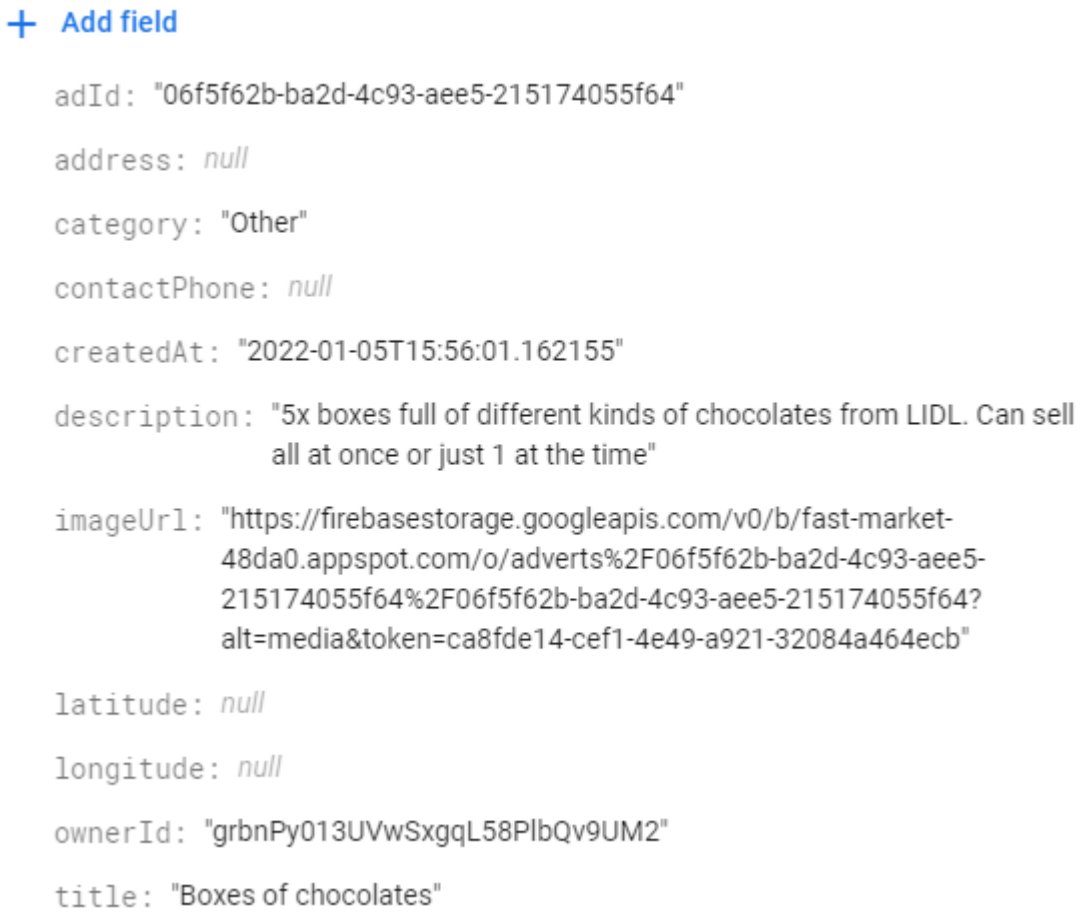
"adverts" kolekcijas ieraksti saturēs sevī laukus:

- adId – sludinājuma unikāls identifikators
- ownerId – sludinājuma autora lietotāja ID
- title – sludinājuma nosaukums
- category – sludinājuma kategorija
- description – sludinājuma apraksts
- imageUrl – sludinājuma attēla saite, var būt NULL
- createdAt – datums, kurā tika publicēts sludinājums, tiks izmantots kārtošanai
- contactPhone – sludinājuma autora telefona Nr., ja tāds eksistē, var būt NULL

Nākamie 3 lauki ir implementēti sludinājuma modelī, bet vēl nav nekur izmantoti (tiek plānots nākotnē pievienot Google Maps integrāciju):

- address – sludinājuma adrese tekstuālā veidā, var būt NULL
- latitude – sludinājuma platuma ģeogrāfiskā koordināte, var būt NULL
- longitude – sludinājuma garuma ģeogrāfiskā koordināte, var būt NULL

Piemēru no Firestore datubāzes kolekcijas "adverts" ieraksta var redzēt 3.3.att.



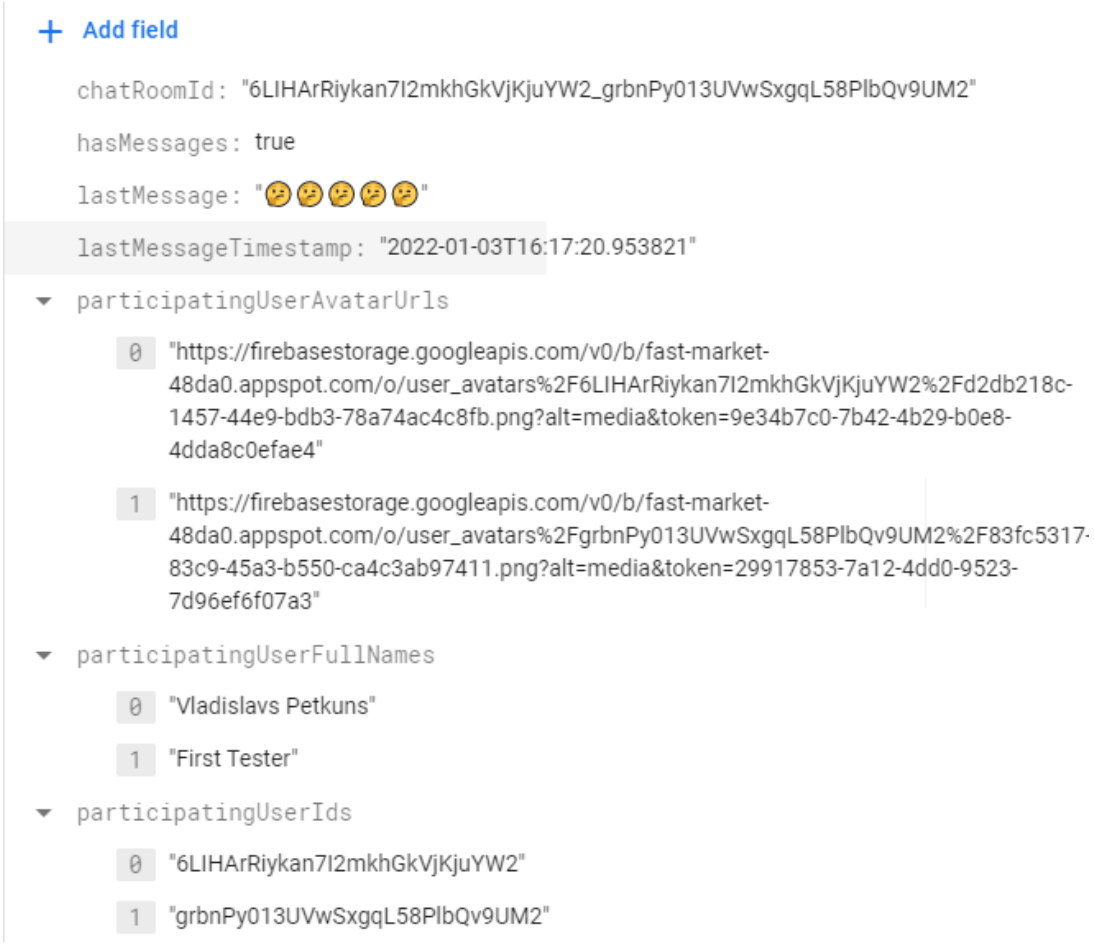
3.3. att. **Firestore datubāzes kolekcijas "adverts" ieraksts**

"chatRooms" kolekcijas ieraksti saturēs sevī laukus:

- chatRoomId – sarunas identifikators, kas sastāv no 2 lietotāju ID
- hasMessages – bool, parāda vai saruna ir vai nav tukša
- lastMessage – sarunas pēdējās īsziņas teksts, var būt NULL (ja sarunā nav īsziņu)
- lastMessageTimestamp – sarunas pēdējās īsziņas sūtīšanas laiks, var būt NULL (ja sarunā nav īsziņu)

- participatingUserAvatarUrls – 2 sarunas dalībnieku profila attēla saites
- participatingUserFullNames – 2 sarunas dalībnieku pilni vārdi (vārds, uzvārds)
- participatingUserIds – 2 sarunas dalībnieku ID

Kā arī bija teikts, papildus pie parastiem laukiem (fields), katrs "chatRooms" ieraksts saturēs arī kolekciju "messages". Ieraksta no Firestore kolekcijas "chatRooms" piemēru var aplūkot 3.4.att.

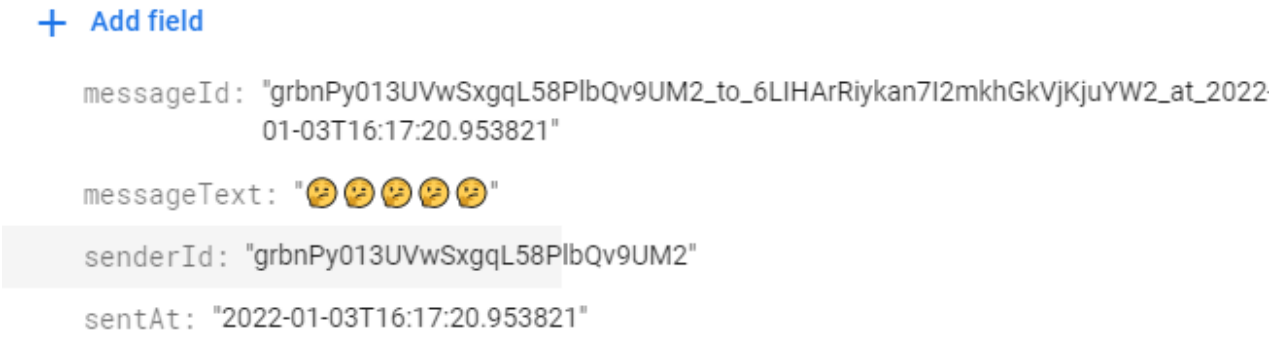


3.4. att. **Firestore datubāzes kolekcijas "chatRooms" ieraksts**

Pēdējā kolekcija ir "messages", kas atradīsies iekšā katram "chatRooms" ierakstam. "messages" kolekcijas ieraksti saturēs sevī laukus:

- messageId – īsziņas unikālais identifikators
- messageText – īsziņas teksts
- senderId – īsziņas autora lietotāja ID
- sentAt – datums, kurā tika nosūtīta īsziņa

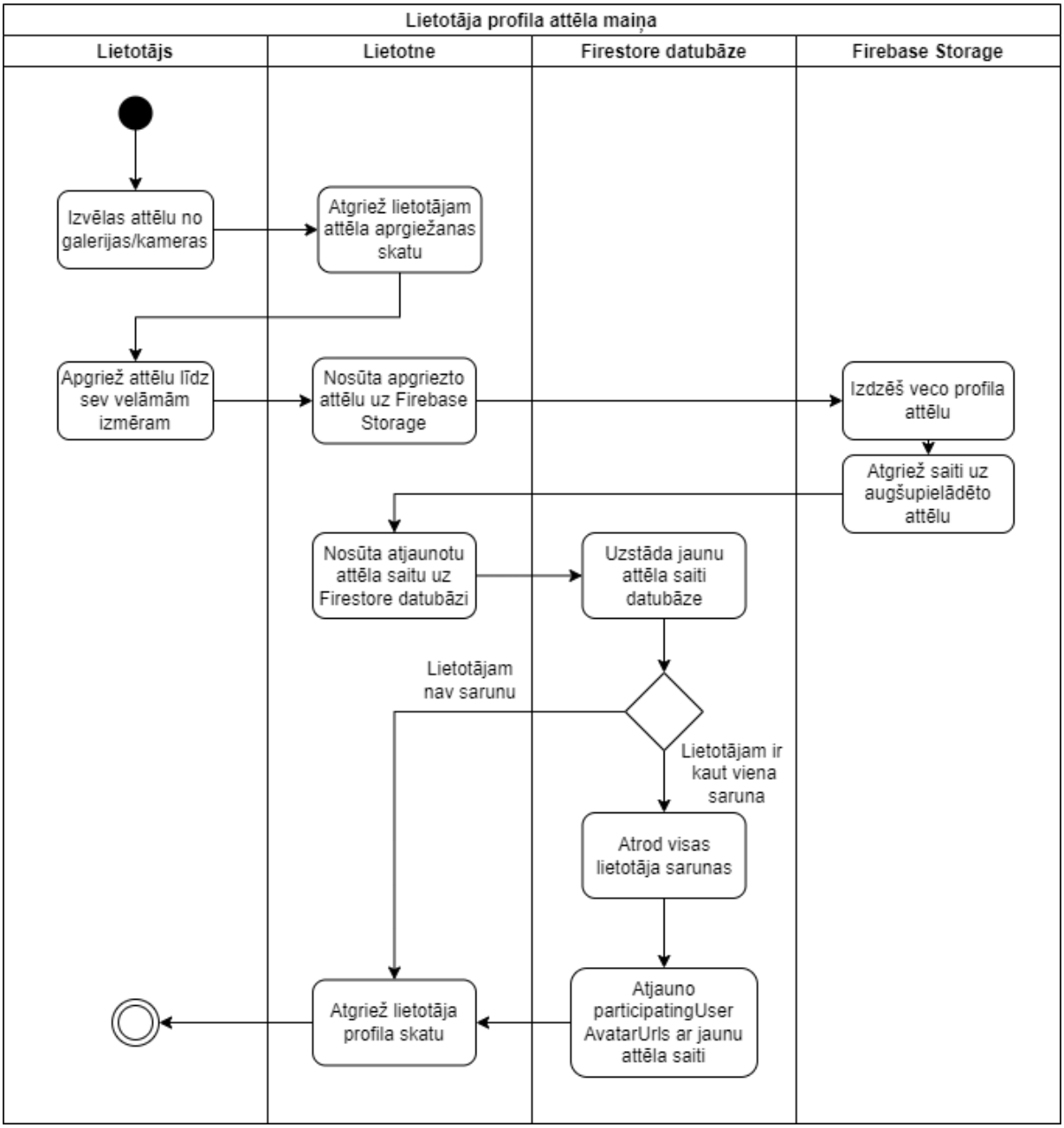
Piemēru no Firestore datubāzes kolekcijas “messages” ieraksta var redzēt 3.5.att.



3.5. att. **Firestore datubāzes kolekcijas “messages” ieraksts**

3.2. Funkciju daļējais projektējums

3.2.1. Lietotāja profila attēla maiņa

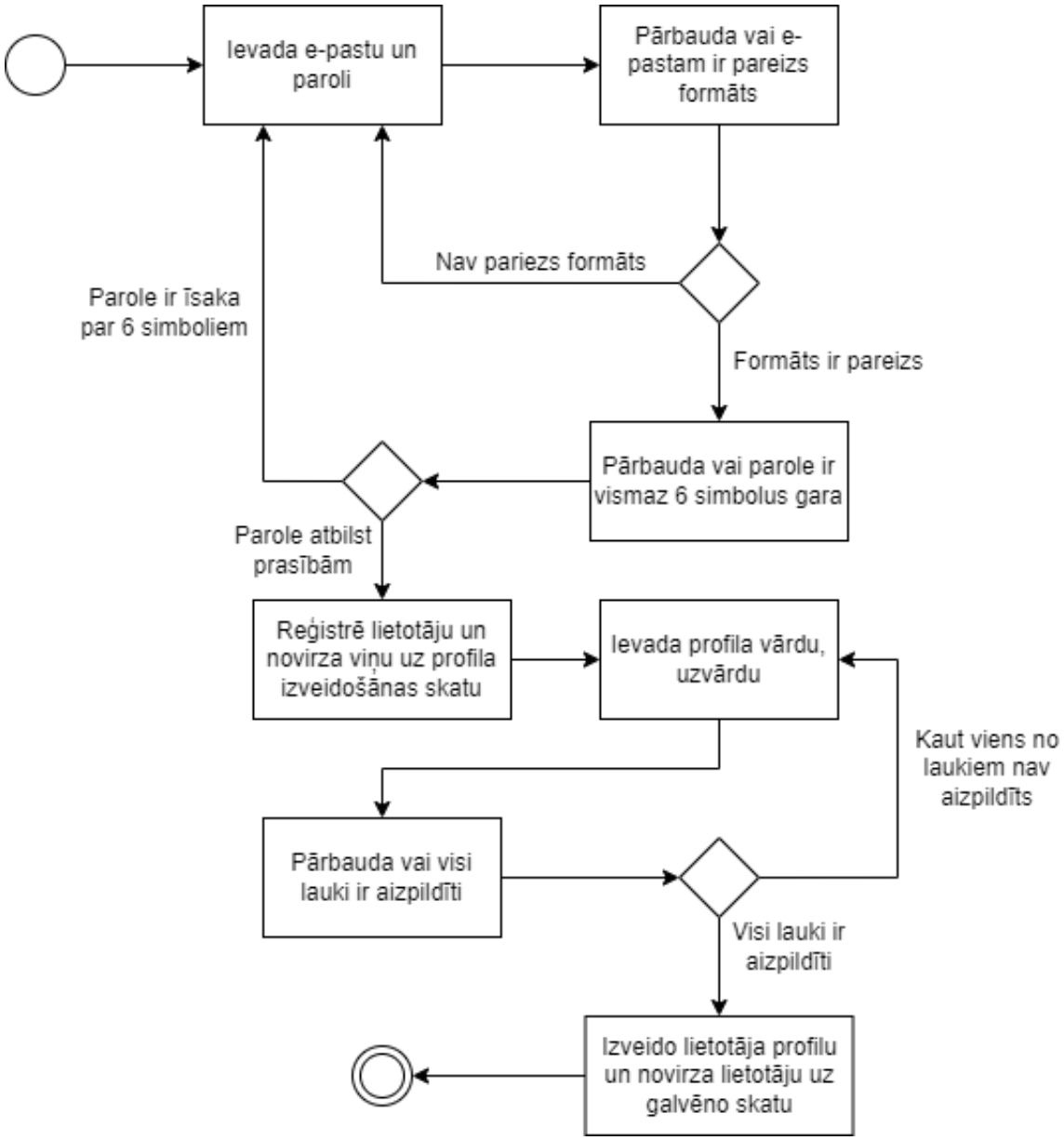


3.6. att. **"Peldceliņu diagramma lietotāja profila attēla maiņai"**

3.6.att. redzamajā diagramma tiek attēlotas darbības, kuras ir nepieciešamas lai veiksmīgi izmainīt lietotāja profila attēlu. Kā pirmie soļi ir pirmkārt attēla izvēle un izvēlēta attēla apgriešana līdz lietotājam vajadzīgām izmēram. Kad attēls ir apgriezts, tas tiek augšupielādēts uz Firebase Storage datu glabātuvi, kur vispirms tiek izdzēsts vecs profila attēls (glabāšanas vietas atbrīvošanai) un tad tiek augšupielādēts jauns attēls. Pēc attēla ielādēs, no Firebase Storage tiek

pieprasīta saite uz jaunu attēlu, kas tiek saņemta lietotnē un ar tās vērtību tiek atjaunota Firestore datubāze. Tālāk seko solis, kas ļaus lietotāja sarunas biedriem redzēt profila attēla izmaiņu viņu lietotņu sarunu sarakstā (tas ir saistīts ar to, ka visu sarunu attēlošanai tiek izmantots "chatRooms" kolekcijas ieraksti, ko atribūts "participatingUserAvatarUrls" tiek uzstādīts sarunas izveidošanas laikā un var saturēt neaktuālu informāciju). Firestore datubāze atlasa visas sarunas pēc lietotāja ID un atjauno katru lietotāja sarunu ar jaunu saiti uz attēlu. Ja lietotājam nav sarunu, pagājušais solis tiek izlaists un lietotājs tiek atgriezts uz profila skatu.

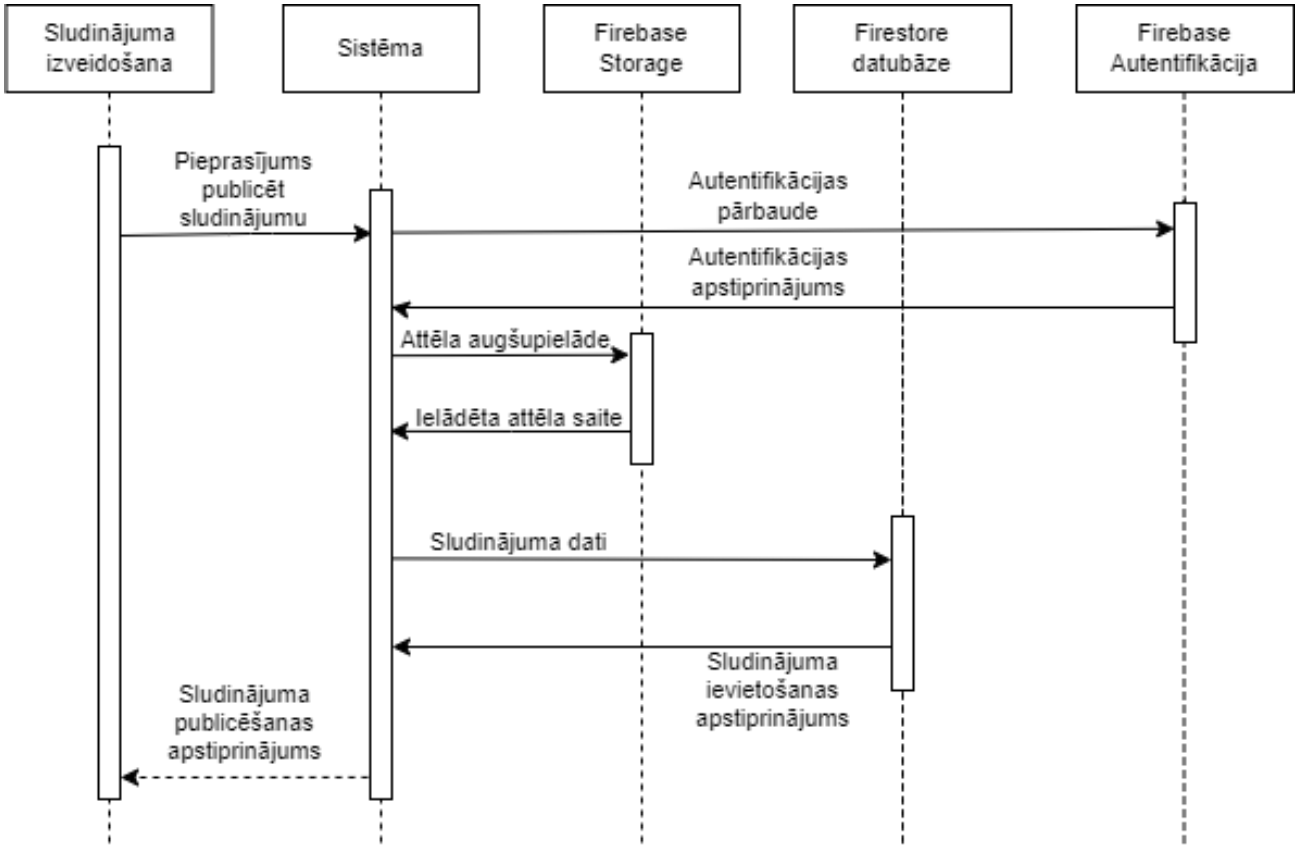
3.2.2. Reģistrācijas soļi



3.7. att. **"Aktivitāšu diagramma reģistrācijas soļiem"**

3.7. att. redzamajā aktivitāšu diagrammā var redzēt kā norisināsies reģistrācijas soļi. Uzreiz vajag piebilst, ka par reģistrāciju atbildēs gan autentifikācijas, gan lietotāju modulis. Reģistrācija uzsākas ar lietotāja e-pasta un paroles ievadi. Lietotne vispirms pārbaudīs e-pasta formātu un ja tas neatbilst standarta e-pasta formātam, parādīs attiecīgo kļūdas paziņojumu un uzprasīs ievadīt citu e-pastu, tas pats notiek ar paroles pārbaudi (vai tā ir vismaz 6 simbolus gara). Ja dati ir atbilstošā formātā, lietotājs tiek piereģistrēts ar Firebase Auth un tiek novirzīts uz profila izveidošanas skatu, kur tam pieprasa ievadīt vārdu un uzvārdu (abiem nav ierobežojumu, bet tie nevar būt tukši). Pēc vārda un uzvārda ievadīšanas, tiek izveidots lietotāja profils ar iepriekš ievadīto informāciju un lietotājs tiek novirzīts uz galveno lietotnes skatu.

3.2.3 Sludinājuma publicēšana

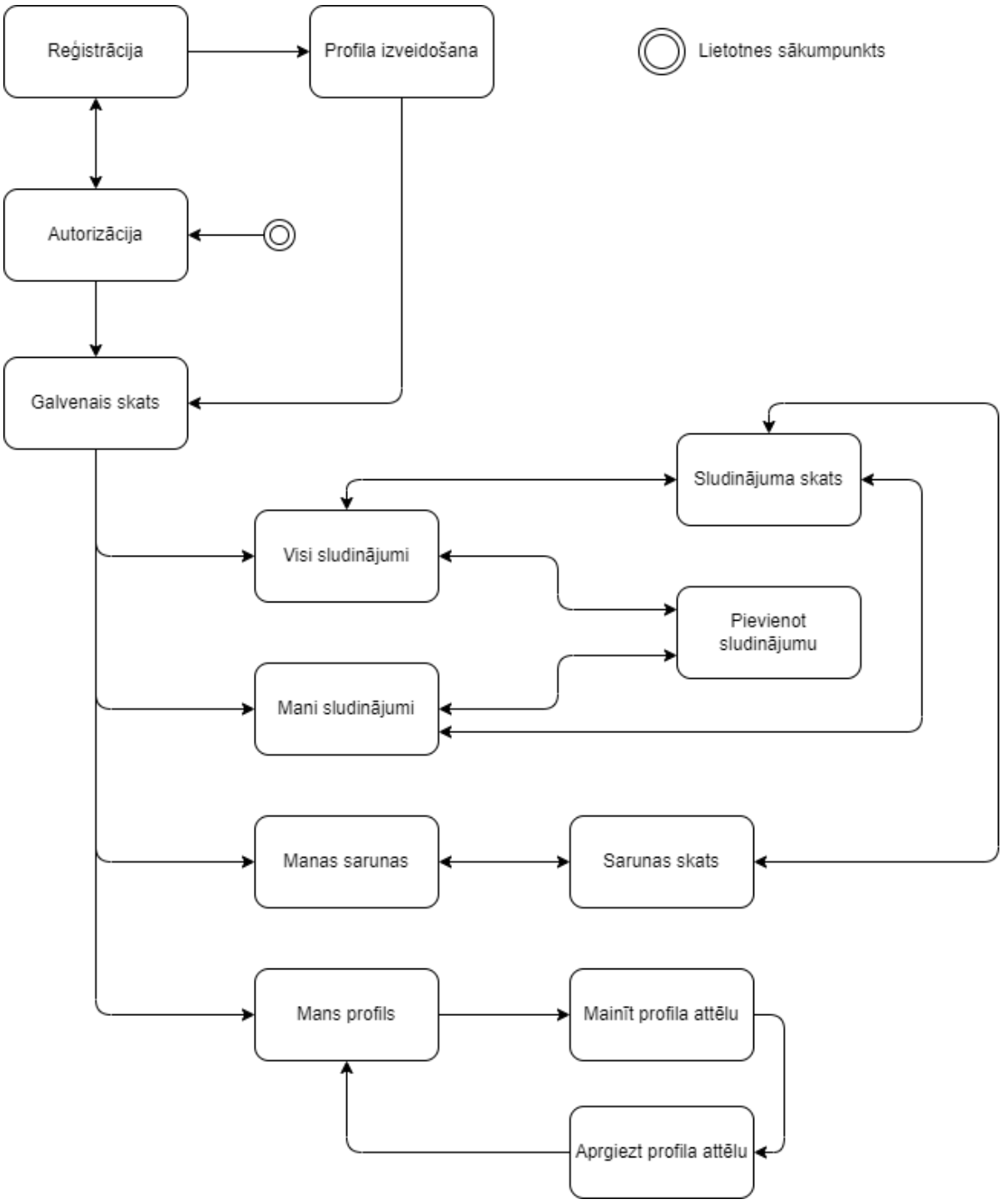


3.8. att. **"Secību diagramma sludinājuma izveidošanai un publicēšanai"**

3.8. att. redzama secību diagramma, kas apraksta sadarbību starp sistēmu, Firebase Storage, Firestore datubāzi un Firebase Autentifikāciju. Ir svarīgi, ka sludinājuma izveidošana un publicēšanā piedalās visi 3 izmantotie Firebase servisi, kas nodrošina visu datu pareizo attēlošanos lietotnē pēc sludinājuma publicēšanas. Pirmais solis pēc pieprasījuma publicēt sludinājumu ir

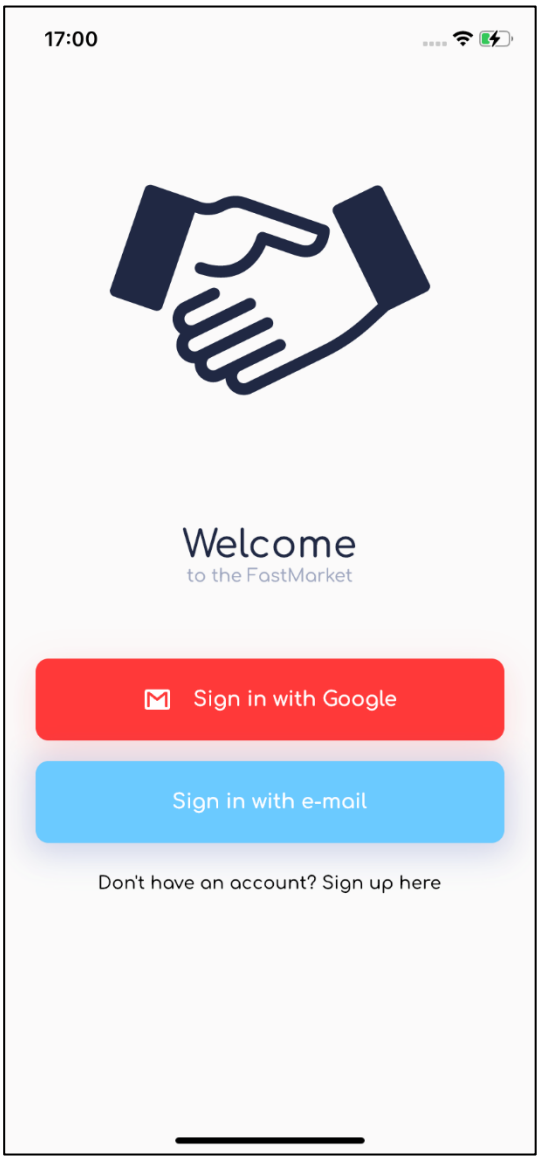
pārbaudīt, vai lietotājs ir ierakstīts sistēmā, to izdara izmantojot Firebase Autentifikācijas palīdzību. Pēc apstiprinājuma par to, vai lietotājs ir autorizēts, sistēma augšupielādē izvēlēto attēlu uz Firebase Storage, no kuras atpakaļ saņem saiti uz šo attēlu. Tikai tad, kad sistēmai ir pieejami visi dati par publicējamo sludinājumu, tā ieraksta jaunu sludinājumu Firestore datubāzes kolekcijā "adverts". Ja viss notika bez kļūdām, sistēma atgriež apstiprinājumu par sludinājuma pievienošanu.

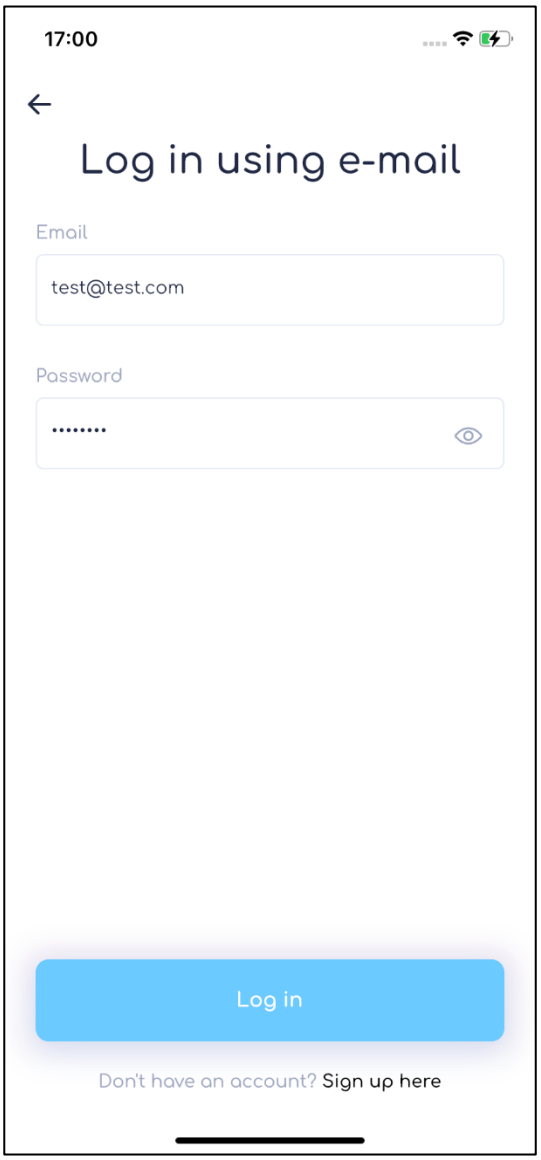
3.3. Lietotāja saskarņu projektējums.



3.9. att. "**Lietotnes skatu diagramma"**

3.3.1. Pieslēgšanas un reģistrācijas skati

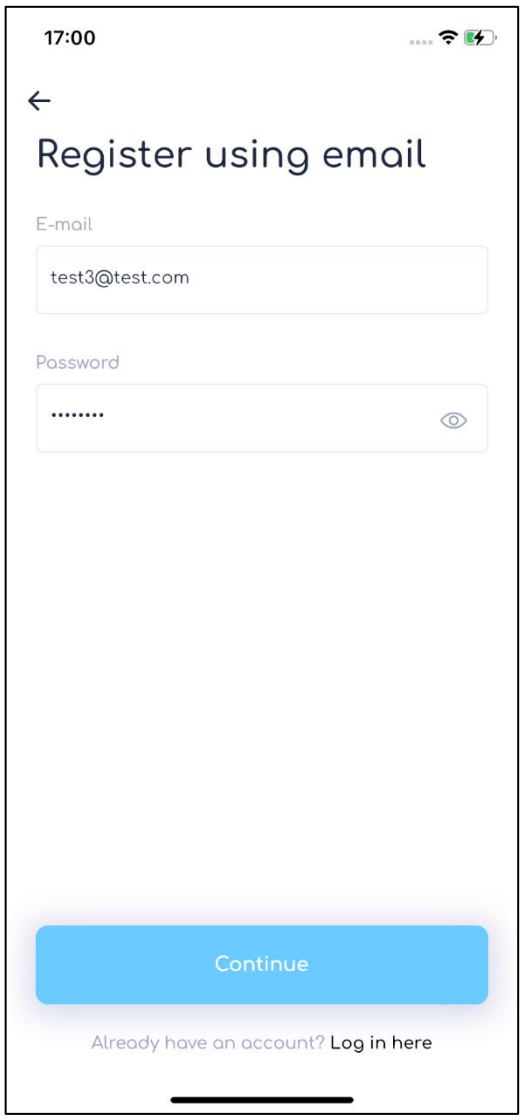




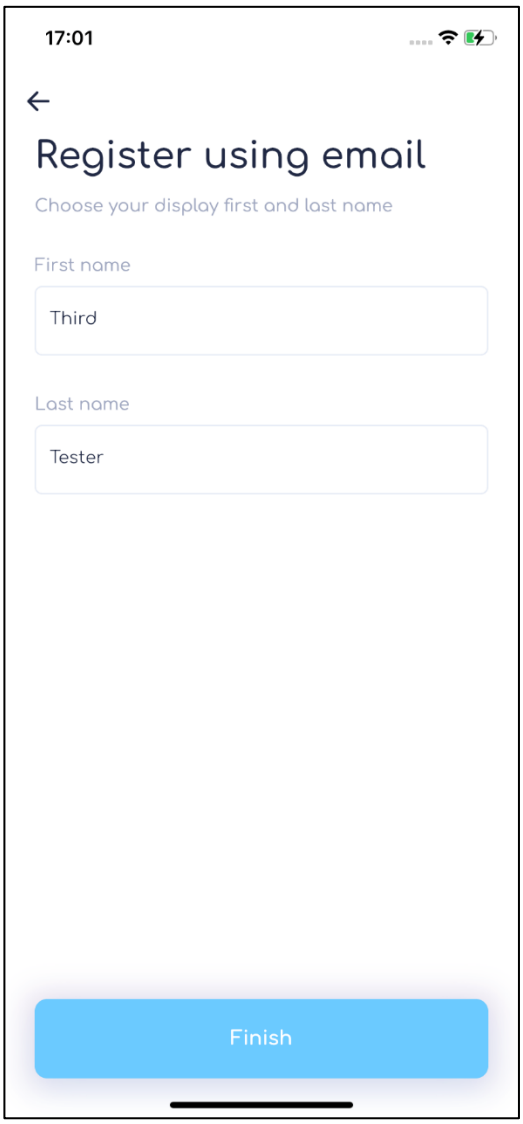
3.10. att. **"Pieslēgšanas galvenais skats"**

3.11. att. **"Pieslēgšanas ar e-pastu skats"**

Atvērot lietotni lietotājs nonāk uz pieslēgšanas galveno skatu (redzams 3.10.att), no kurienes ir 3 ceļi – pieteikties sistēmai ar Google kontu, pieteikties ar e-pastu un paroli (skats uz 3.11. att.), reģistrēties sistēmā ar e-pastu un paroli (skats uz 3.12. att.).



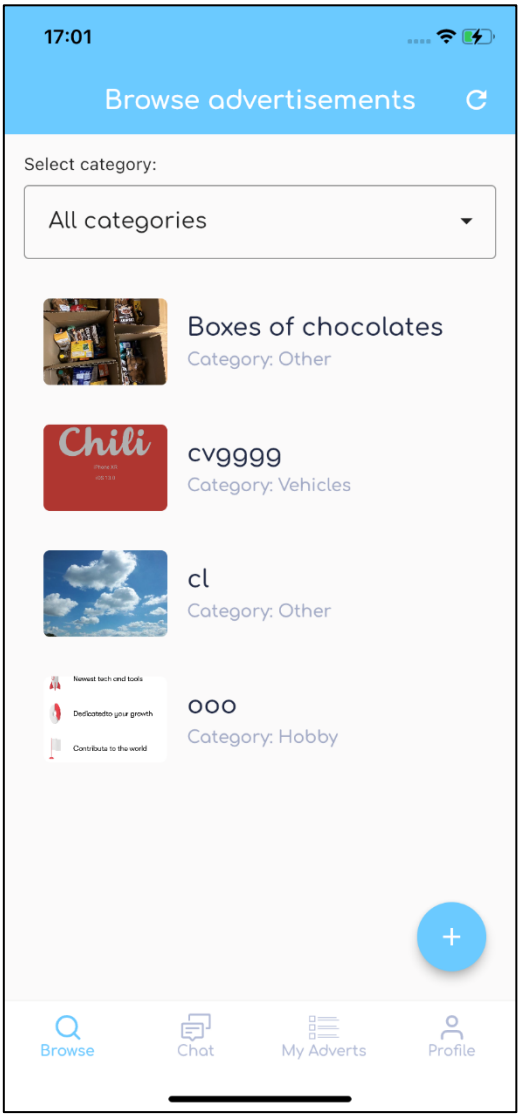
3.12. att. **“Reģistrēšanas skats”**



3.13. att. **“Profila izveidošanas skats”**

Reģistrācijas skatā lietotājs var reģistrēties, ievadot abus laukus un nospiežot uz pogu “Continue”, kas aizvadīs viņu uz profila izveidošanas skatu (redzams 3.13. att.). Profila izveidošanas skatā lietotājam vajag izvēlēties savu vārdu un uzvārdu, kas būs redzami lietotnē citiem lietotājiem.

3.3.2. Sludinājumu nodaļa

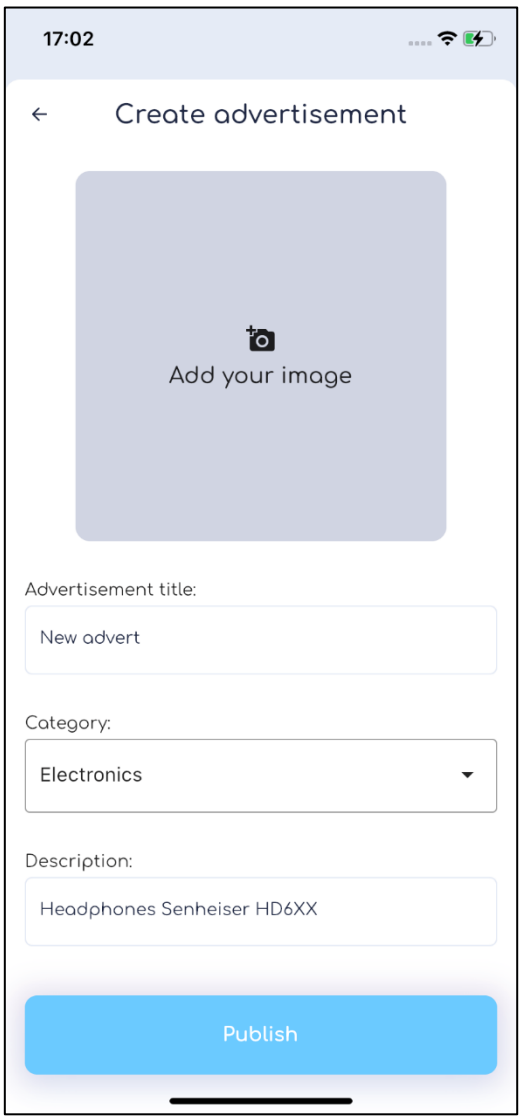


3.14. att. **"Visu sludinājumu skats"**

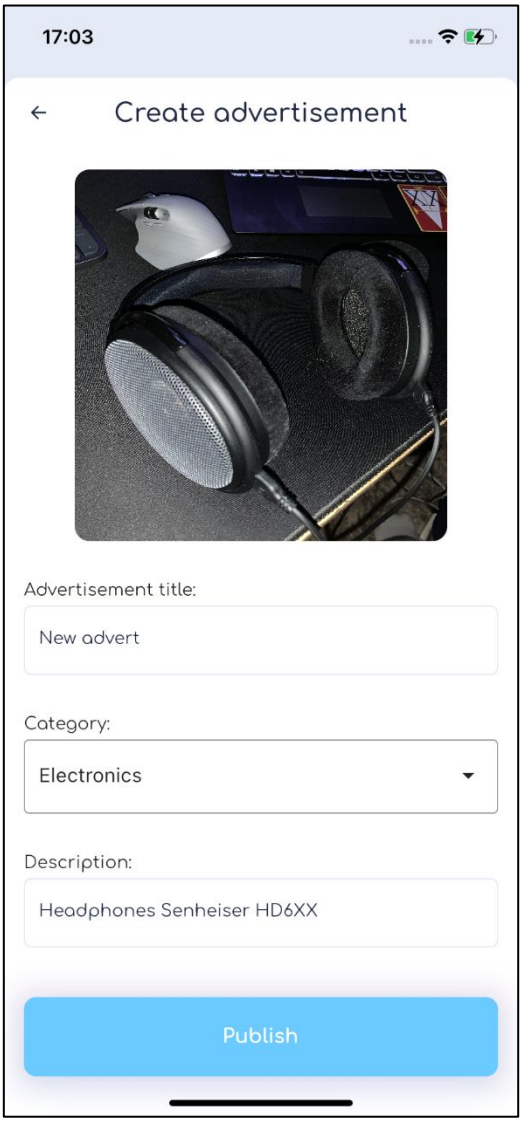
3.15. att. **"Tukšs sludinājumu skats"**

Nonākot uz galveno skatu, pirmā no nodaļām ir visu sludinājumu nodaļa, pēc noklusējuma kā filtrs ir izvēlēts attēlot visas kategorijas. Kā izskatās visu sludinājumu skats ar sludinājumu sarakstu var redzēt 3.14. att. Nospiežot uz kādu no sludinājumiem atvērsies sludinājuma detalizēts skats (3.18. att.).

Ja lietotājs izvēlas kategoriju, kurā vēl nav neviens publicēts sludinājums, viņam tiek paradīts tukšs sludinājumu skats (redzams 3.15. att.), kurā ir paziņojums par to, ka lietotājs var pievienot savu sludinājumu nospiežot uz pogu ar "+" zīmi. Ja lietotājs uzspiež uz "+" pogu, atvērās modāls ar sludinājuma pievienošanas skatu (3.16. att.).



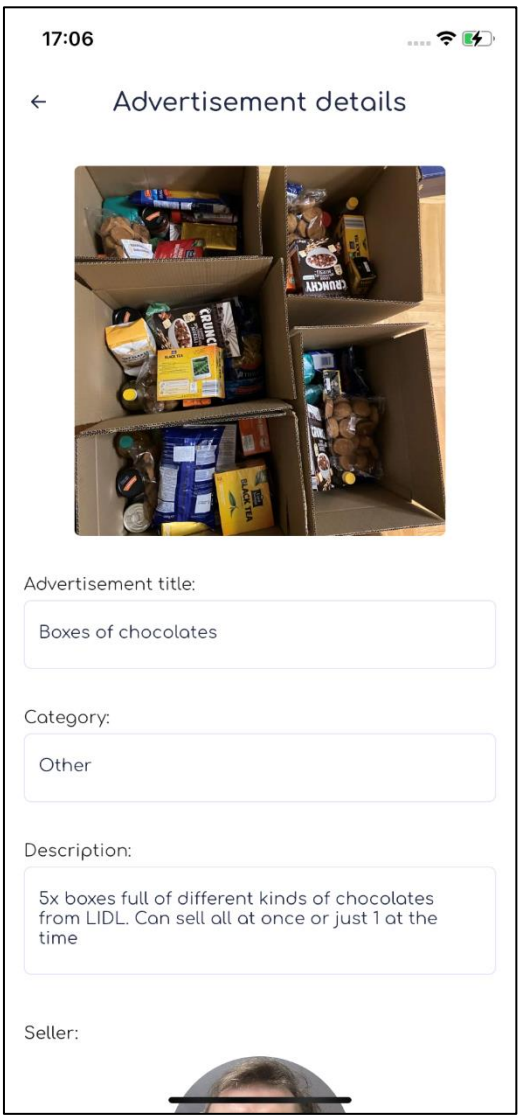
3.16. att. **“Sludinājuma pievienošanas skats”**



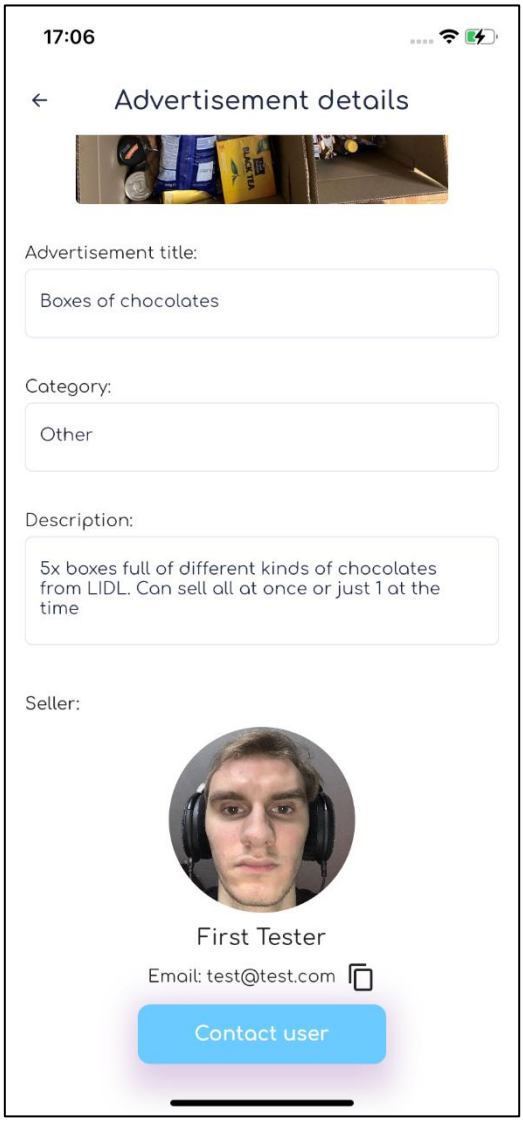
3.17. att. **“Sludinājuma pievienošanas skats ar izvēlēto attēlu”**

Nonākot sludinājuma pievienošanas skatā lietotājam tiks pieprasīts izvēlēties sludinājuma nosaukumu, kategoriju un aprakstu. Attēlu pievienot nav obligāti, 3.16. att. var redzēt aizpildītus laukus bez izvēlēta attēla un 3.17. att. ir redzams kā izskatīsies skats ja lietotājs attēlu ir izvelējies.

3.3.3. Sludinājuma detalizēts skats



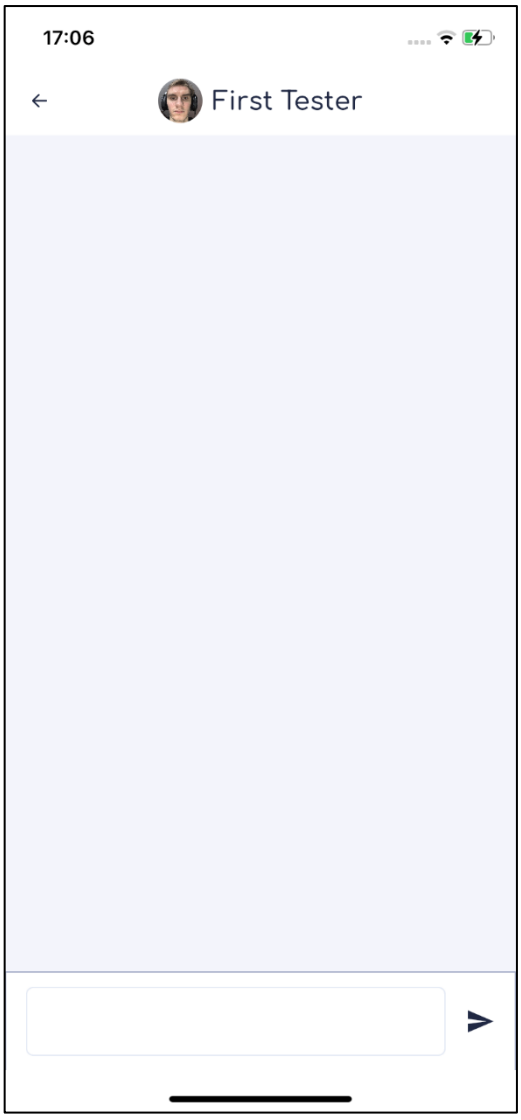
3.18. att. **“Sludinājuma detalizēta skata 1.daļa”**



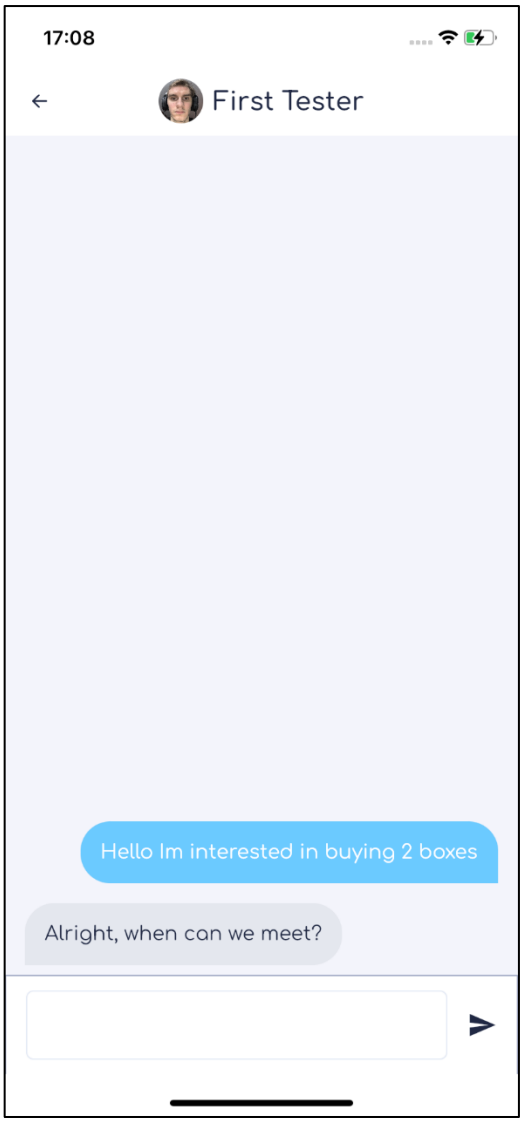
3.19. att. **“Sludinājuma detalizēta skata 2.daļa”**

Nonākot uz sludinājuma detalizētu skatu lietotājam ir redzams sludinājuma attēls (ja tāds ir, ja nav tad rādās noklusējuma attēls), nosaukums, kategorija un apraksts. Kā arī var redzēt 3.19. att., ja lietotājs atver ne savu sludinājumu, tad skata apakšpusē radās sludinājuma autora detaļas un poga “Contact user”, kura novirzīs lietotāju uz sarunas skatu ar sludinājuma autoru.

3.3.4. Sarunas skats



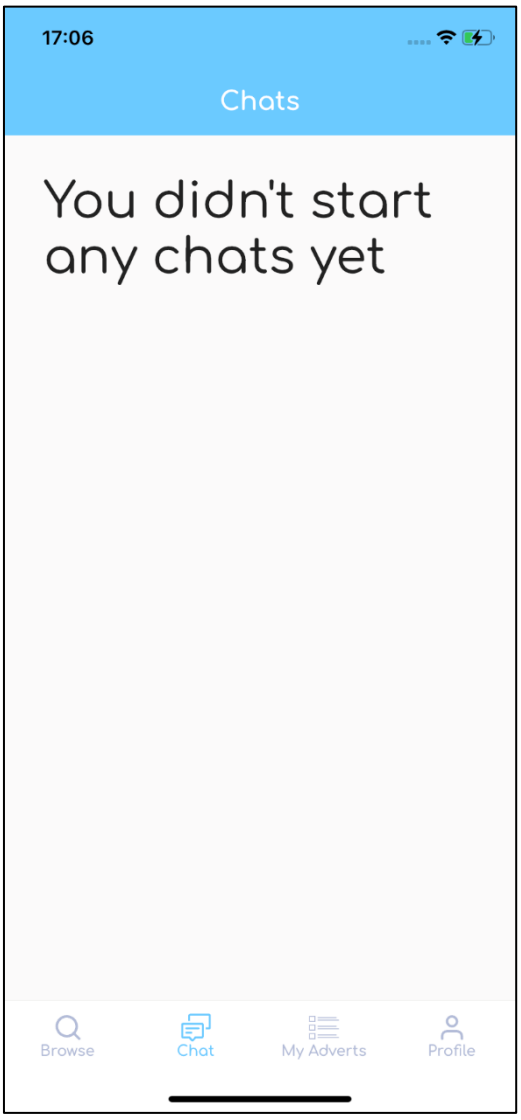
3.20. att. **“Sarunas tukšais skats”**



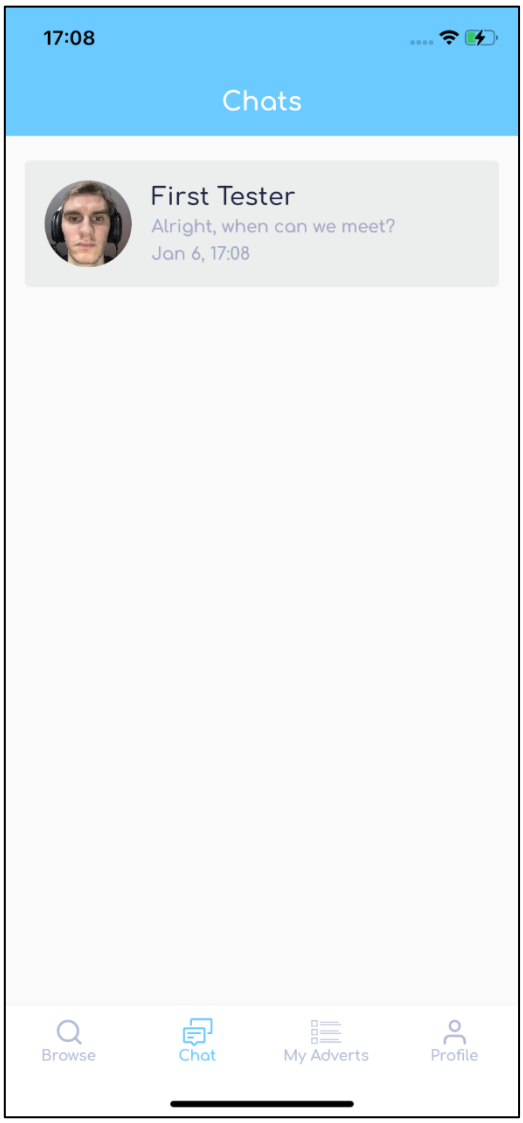
3.21. att. **“Sarunas skats ar īsziņām”**

Ja lietotājs atver sarunas skatu ar pogu “Contact user” un viņš pirms tam nav iesācis sarunu ar dotu lietotāju, tad radīsies sarunas tukšais skats (3.20. att.), kad lietotāji ierakstīs savas īsziņas sarunā skats izskatīsies kā 3.21. att.

3.3.5. Manu sarunu nodaļa



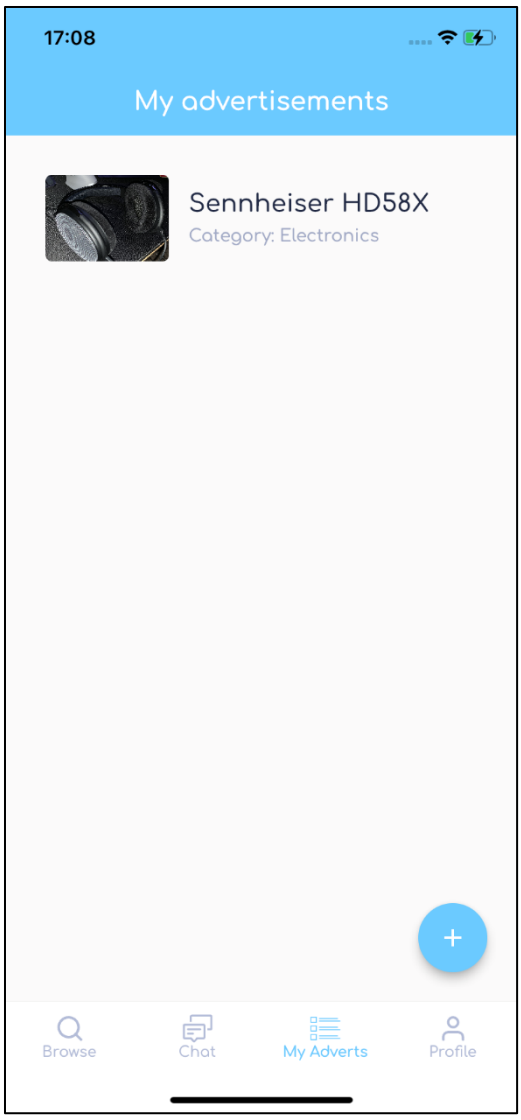
3.22. att. **"Manu sarunu tukšais skats"**



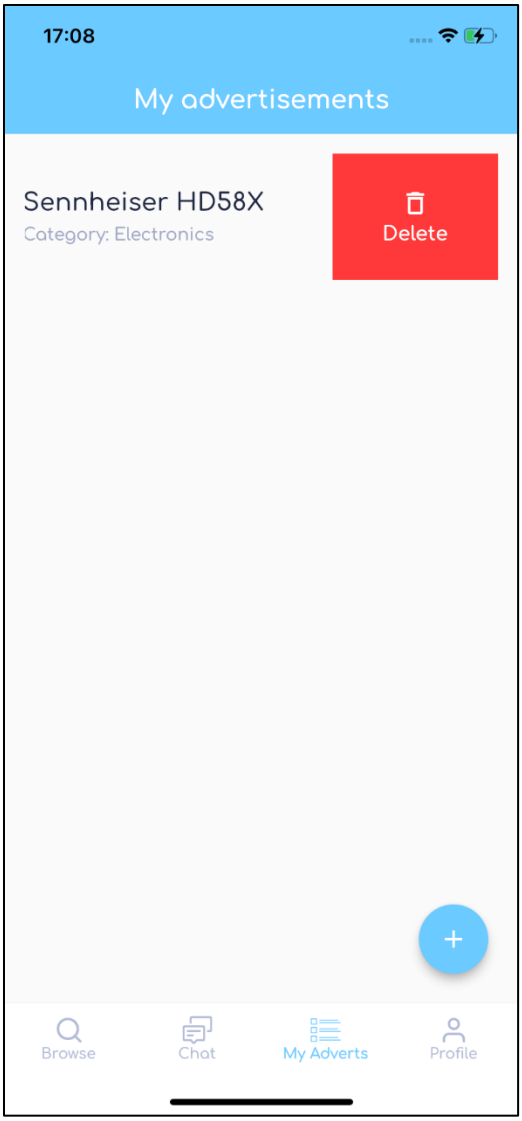
3.23. att. **"Manu sarunu saraksta skats"**

Manu sarunu nodaļā lietotāja varēs apskatīt visas savas uzsāktas sarunas. Ja lietotājs vēl nav uzsācis nevienu sarunu, šajā nodaļā radīsies 3.22. att. redzams skats, ja lietotājam ir kaut viena uzsākta saruna, tam būs pieejams 3.23. att. redzamais skats, kur būs saraksts ar visām sarunām. Katrās sarunas saraksta elements saturēs adresāta profila attēlu, pilno vārdu, pēdējās īsziņas tekstu un sūtīšanas laiku. Nospiežot uz kādu no sarunām atvērsies sarunas skats (kas jau bija redzams 3.21. att.).

3.3.6. Manu sludinājumu nodaļa



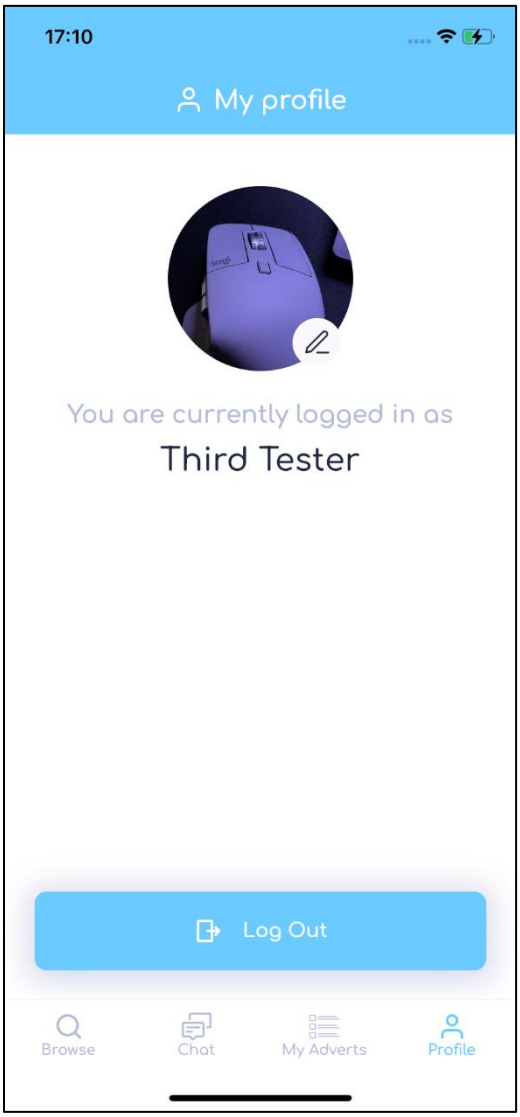
3.24. att. **"Manu sludinājumu skats"**



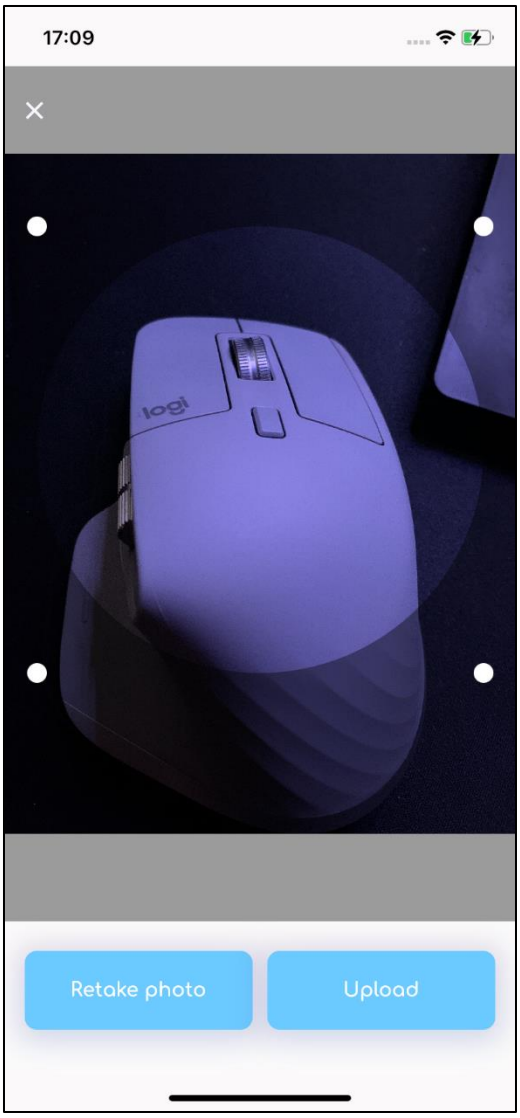
3.25. att. **"Dzēst manu sludinājumu"**

Manu sludinājumu nodaļā radīsies tikai lietotāja pievienotie sludinājumi, tā izskatīsies kā ir redzams 3.24. att.. Nospiežot uz kādu no saviem sludinājumiem atvērsies sludinājuma detalizēts skats (jau bija redzams 3.18. att., protams tajā nebūs informācijas par autoru, jo autors ir pieteikts lietotājs). Lai izdzēst savu sludināju vajadzēs pavilkt saraksta elementu no labās uz kreiso pusi, lai parādās 3.25. att. redzama sarkana poga "Delete". Nospiežot uz to lietotājs izdzēš sludinājumu un atjauno sarakstu.

3.3.7. Mana profila nodaļa



3.26. att. **"Mana profila nodaļās skats"**



3.27. att. **"Jauna profila attēla apgriešanas skats"**

Nonākot uz sava profila nodaļu (redzama 3.26. att.) lietotājam radīsies viņa profila attēls ar pogu tā izmaiņai, lietotāja vārds, uzvārds, kā arī poga atteikšanai no sistēmas. Kad lietotājs uzspiež uz profila attēla izmaiņas pogu viņam atvērsies attēla izvēlne no kameras vai galerijas. Kad attēls ir izvēlēts, lietotājam tiks piedāvāts apgriezt attēlu līdz sev vēlamām izmēram (apgriešanas skats ir redzams 3.27. att.). Kad lietotājs ir apmierināts ar apgrieztu attēlu, nospiežot uz pogu "Upload" tas tiek augšupielādēts un lietotājs tiek atgriezts uz sava profila skatu. Ja lietotāju neapmierina savs izvēlēts attēls viņš var izvēlēties citu attēlu nospiežot uz pogu "Retake photo" (ja attēls bija uzņemts ar kameru) / "Choose another" (ja attēls bija izvēlēts no galerijas) vai atgriezties uz sava profila skatu nospiežot uz "X" simbolu augšējā kreisajā stūrī.

4. TESTĒŠANAS DOKUMENTĀCIJA

4.1. Testēšanas metodika

Lietotnes testēšana tiek veiktā divās mobilajās platformās: iOS un Android, lai pārbaudīti tās pareizdarbību un saderību. iOS un Android testēšana tiks veikta gan uz emulatoriem (kas ļaus notestēt dažādas operētājsistēmas versijas), gan uz fiziskām ierīcēm: šajā gadījumā tika izmantoti iPhone XR ar iOS 14.9 un Poco X3 Pro ar Android 11.

Testēšanas princips būs "melna kaste", kas nozīmē, ka lietotne tiks testētā tāpat kā to būtu lietojis īsts lietotājs, darot to tiks pārbaudīts, vai lietotne strādās atbilstoši aprakstītām prasībām.

Testēšanās procesā ir:

- jāpārbauda katra lietotnes moduļa funkciju darbību
- jāpārbauda saskarņu un skatu pareizo darbību
- jāpārbauda pareizo datu ierakstīšanu un nolasīšanu no Firebase Firestore datubāzes

4.2. Saistība ar citiem dokumentiem

Testēšana tika veikta balstoties uz iepriekš aprakstītiem funkciju darbībām no programmatūras prasību specifikācijas un programmatūras projektējuma apraksta.

4.3. Testēšanas plāns

Vispirms tiks izplānots testēšanas žurnāls atsevišķajiem moduļiem. Žurnālos tiks definēts testa apraksts, vēlamais rezultāts un testa gaitā tiks ierakstīts reāls rezultāts, kā arī vai šīs tests ir iziets vai nav iziets. Kamēr tiks testēta funkciju darbība, paralēli tiks pārbaudīts, vai Firebase Firestore datubāzē nonāk pareizi dati (tas ir attiecīgs tikai uz funkcijām, kas sadarbojās ar datu bāzi) un vai pareizi strādā pārejas starp lietotnes skatiem. Testi tiks veikti pēc katra moduļa izstrādes.

4.4. Testēšanas žurnāli

Nākamajās nodaļās ir aprakstīti testa žurnāli katram no lietotnes moduļiem, kā arī žurnāls saskarņu testiem.

4.4.1. Autentifikācijas moduļa testēšanas žurnāls

4.1. Tabula **“Autentifikācijas moduļa testēšanas žurnāls”**

Testa ID	Apraksts	Vēlamais rezultāts	Reāls rezultāts	Tests ir iziets
Auth_01	Lietotājs autorizējas ar savu Google kontu	Lietotājs tiek novirzīts uz lietotnes galveno skatu	Lietotājs tiek novirzīts uz lietotnes galveno skatu	+
Auth_02	Lietotājs autorizējas ar nepareizo e-pasta un paroles kombināciju	Lietotne izvada kļūdas paziņojumu	Lietotne izvada kļūdas paziņojumu	+
Auth_03	Lietotājs autorizējoties neievada kādu no obligātiem laukiem	Poga “Sign In” nav aktīva un uz to nevar uzspiest	Poga “Sign In” nav aktīva un uz to nevar uzspiest	+
Auth_04	Lietotājs autorizējas ar pareizo e-pastu un paroli	Lietotājs tiek novirzīts uz lietotnes galveno skatu	Lietotājs tiek novirzīts uz lietotnes galveno skatu	+
Auth_05	Lietotājs reģistrējoties ievada pareiza veida e-pastu, bet parole ir <6 simbolus gara	Virs paroles lauka pāradās paziņojums par paroles garumu, poga “Continue” nav aktīva un uz to nevar uzspiest	Virs paroles lauka pāradās paziņojums par paroles garumu, poga “Continue” nav aktīva un uz to nevar uzspiest	+

Auth_06	Lietotājs reģistrējoties ievada nepareiza formāta e-pastu, bet parole ir >6 simbolus gara	Virs e-pasta lauka pāradās paziņojums par nepareizo formātu, poga "Continue" nav aktīva un uz to nevar uzspiest	Virs e-pasta lauka pāradās paziņojums par nepareizo formātu, poga "Continue" nav aktīva un uz to nevar uzspiest	+
Auth_07	Lietotājs reģistrējas un ievada pareiza formāta e-pastu un paroli kas ir >6 simbolus gara	Lietotājs tiek reģistrēts un novirzīts uz profila izveidošanas skatu	Lietotājs tiek reģistrēts un novirzīts uz profila izveidošanas skatu	+
Auth_08	Lietotājs nospiež uz "Logout" pogu profila skatā	Lietotājs tiek atteikts no sistēmas un novirzīts uz galveno autorizācijas skatu	Lietotājs tiek atteikts no sistēmas un novirzīts uz galveno autorizācijas skatu	+

4.4.2. Lietotāju moduļa testēšanas žurnāls

4.2. Tabula **"Lietotāju moduļa testēšanas žurnāls"**

Testa ID	Apraksts	Vēlamais rezultāts	Reāls rezultāts	Tests ir iziets
User_01	Lietotājs izveidojot profilu neievada kādu no laukiem	Poga "Finish" nav aktīva un uz to nevar uzspiest	Poga "Finish" nav aktīva un uz to nevar uzspiest	+

User_02	Lietotājs izveido profilu un aizpilda visus laukus	Tiek izveidots lietotāja profils, lietotājs tiek novirzīts uz galveno skatu	Tiek izveidots lietotāja profils, lietotājs tiek novirzīts uz galveno skatu	+
User_03	Lietotājs izmaina savu profila attēlu	Profila skatā pāradās lietotāja izvelēts jauns attēls	Profila skatā joprojām radās lietotāja vecais attēls	-
User_03_try_2	Lietotājs izmaina savu profila attēlu	Profila skatā pāradās lietotāja izvelēts jauns attēls	Profila skatā pāradās lietotāja izvelēts jauns attēls	+
User_04	Lietotājs atver ne sava sludinājuma detalizēto skatu	Skata apakšā ielādējas sludinājuma īpašnieka profils	Skata apakšā ielādējas sludinājuma īpašnieka profils	+

4.4.3. Attēlu moduļa testēšanas žurnāls

4.3. Tabula **"Attēlu moduļa testēšanas žurnāls"**

Testa ID	Apraksts	Vēlamais rezultāts	Reāls rezultāts	Tests ir iziets
Images_01	Lietotājs augšupielādē savu pirmo profila attēlu	Firebase Storage mapē ar lietotāja ID pāradās lietotāja augšupielādēts attēls	Firebase Storage mapē ar lietotāja ID pāradās lietotāja augšupielādēts attēls	+

Images_02	Lietotājs izmaina savu eksistējošo profila attēlu	Firebase Storage mapē ar lietotāja ID izdzēšas fails ar veco attēlu un pāradās jauna attēla fails	Firebase Storage mapē ar lietotāja ID pāradās otrs attēls, vecais netiek izdzēsts	-
Images_02_try2	Lietotājs izmaina savu eksistējošo profila attēlu	Firebase Storage mapē ar lietotāja ID izdzēšas fails ar veco attēlu un pāradās jauna attēla fails	Firebase Storage mapē ar lietotāja ID izdzēšas fails ar veco attēlu un pāradās jauna attēla fails	+
Images_03	Lietotājs pievieno jaunu sludinājumu ar izvēlēto attēlu	Firebase Storage mapē ar sludinājuma ID pāradās izvēlēts sludinājuma attēls	Firebase Storage mapē ar sludinājuma ID pāradās izvēlēts sludinājuma attēls	+
Images_04	Lietotājs izdzēš savu sludinājumu	Firebase Storage izdzēšas mape ar sludinājuma ID	Firebase Storage nenotiek nekādas izmaiņas	-
Images_04_try_2	Lietotājs izdzēš savu sludinājumu	Firebase Storage izdzēšas mape ar sludinājuma ID	Firebase Storage izdzēšas mape ar sludinājuma ID	+

4.4.4. Sludinājumu moduļa testēšanas žurnāls

4.4. Tabula **"Sludinājumu moduļa testēšanas žurnāls"**

Testa ID	Apraksts	Vēlamais rezultāts	Reāls rezultāts	Tests ir iziets
Adverts_01	Lietotājs pirmo reizi atver visu sludinājumu nodaļu	Filtrētas kategorijas ir uzstādītas uz "All categories" un skats attēlo visus sludinājumus	Filtrētas kategorijas ir uzstādītas uz "All categories" un skats attēlo visus sludinājumus	+
Adverts_02	Lietotājs izmaina filtrēšanas kategoriju (kategorijā ir sludinājumi)	Skats atjauno sludinājumu sarakstu tikai ar izvēlētas kategorijas sludinājumiem	Skats atjauno sludinājumu sarakstu tikai ar izvēlētas kategorijas sludinājumiem	+
Adverts_03	Lietotājs izmaina filtrēšanas kategoriju (kategorija ir tukša)	Lietotājam tiek attēlots tukša saraksta skats	Lietotājam tiek attēlots tukša saraksta skats	+
Adverts_04	Lietotājs atver savu sludinājumu nodaļu (lietotājam ir sludinājumi)	Lietotājam tiek attēlots skats ar viņa sludinājumu sarakstu (kur "ownerId" ir viņa ID)	Lietotājam tiek attēlots skats ar viņa sludinājumu sarakstu (kur "ownerId" ir viņa ID)	+

Adverts_05	Lietotājs atver savu sludinājumu nodaļu (lietotājam nav sludinājumu)	Lietotājam tiek attēlots tukša saraksta skats	Lietotājam tiek attēlots tukša saraksta skats	+
Adverts_06	Lietotājs izdzēš savu sludinājumu	Sludinājums tiek izdzēsts no manu sludinājumu saraksta, tas tiek atjaunots, kā ari tiek atjaunots saraksts nodaļā "visi sludinājumi"	Sludinājums tiek izdzēsts no manu sludinājumu saraksta, tas tiek atjaunots, bet saraksts nodaļā "visi sludinājumi" joprojām satur izdzēsto sludinājumu	-
Adverts_06_try_2	Lietotājs izdzēš savu sludinājumu	Sludinājums tiek izdzēsts no manu sludinājumu saraksta, tas tiek atjaunots, kā ari tiek atjaunots saraksts nodaļā "visi sludinājumi"	Sludinājums tiek izdzēsts no manu sludinājumu saraksta, tas tiek atjaunots, kā ari tiek atjaunots saraksts nodaļā "visi sludinājumi"	+
Adverts_07	Lietotājs izveidojot sludinājumu neaizpilda kādu no obligātiem laukiem	Poga "Publish" nav aktīva un uz to nevar uzspiest	Poga "Publish" nav aktīva un uz to nevar uzspiest	+

Adverts_08	Lietotājs publicē sludinājumu	Tiek aizvērts pievienošanas skats un tiek atjaunots visu sludinājumu (izvelēta kategorija netiek mainīta) un manu sludinājumu saraksts	Tiek aizvērts pievienošanas skats bet tiek atjaunots tikai visu sludinājumu skats	-
Adverts_08_try_2	Lietotājs publicē sludinājumu	Tiek aizvērts pievienošanas skats un tiek atjaunots visu sludinājumu (izvelēta kategorija netiek mainīta) un manu sludinājumu saraksts	Tiek aizvērts pievienošanas skats un tiek atjaunots visu sludinājumu (izvelēta kategorija netiek mainīta) un manu sludinājumu saraksts	+

4.4.5. Sarunu moduļa testēšanas žurnāls

4.5. Tabula **"Sarunu moduļa testēšanas žurnāls"**

Testa ID	**Apraksts**	**Vēlamais rezultāts**	**Reāls rezultāts**	**Tests ir iziets**
Chats_01	Lietotājs atver savu sarunu nodaļu (lietotājam ir sarunas)	Tiek paradīts lietotāja sarunu saraksts	Tiek paradīts lietotāja sarunu saraksts	+

Chats_02	Lietotājs atver savu sarunu nodaļu (lietotājam nav sarunu)	Tiek paradīts sarunu tukšais skats	Tiek paradīts sarunu tukšais skats	+
Chats_03	Lietotājs uzsāk sarunu, bet neuzrakstot nevienu īsziņu atgriežas atpakaļ	Saruna tiek izveidota Firestore datubāze, bet netiek attēlota "manas sarunas" nodaļā, jo tā ir tukša	Saruna tiek izveidota Firestore datubāze, bet netiek attēlota "manas sarunas" nodaļā, jo tā ir tukša	+
Chats_04	Lietotājs uzsāk sarunu un uzraksta kaut vienu īsziņu	Saruna tiek izveidota Firestore datubāze un parādās "manas sarunas" nodaļā	Saruna tiek izveidota Firestore datubāze un parādās "manas sarunas" nodaļā	+
Chats_05	Lietotājs atver eksistējošo netukšu sarunu un ieraksta kādu īsziņu, atgriežas uz "manas sarunas" nodaļu	Atvērtas sarunas pēdējās īsziņas teksts un laiks mainās nodaļas "manas sarunas" sarakstā	Atvērtas sarunas pēdējās īsziņas teksts un laiks mainās nodaļas "manas sarunas" sarakstā	+
Chats_06	Ir atvērts sarunas skats, sarunas biedrs iesūta jaunu īsziņu sarunā	Jauna īsziņa reaktīvi parādās sarunas īsziņu sarakstā	Jauna īsziņa reaktīvi parādās sarunas īsziņu sarakstā	+

Chats_07	Atvērta “manas sarunas” nodaļa un viens no sarunas biedriem izmaina savu profila attēlu	Jauns profila attēls reaktīvi parādās sarunas kartītē	Profila attēls nemainās	-
Chats_07_try_2	Atvērta “manas sarunas” nodaļa un viens no sarunas biedriem izmaina savu profila attēlu	Jauns profila attēls reaktīvi parādās sarunas kartītē	Jauns profila attēls reaktīvi parādās sarunas kartītē	+

5. PROJEKTA PĀRVALDĪBA

5.1 Programmatūras projekta organizācija

Darbs tika izstrādāts patstāvīgi, komandā bija tikai darba autors. Lietotnes izstrādei tika izmantotā ūdenskritumā metode – pirms koda rakstīšanas tika izstrādātas visas nepieciešamas prasības.

5.2. Kvalitātes nodrošināšana

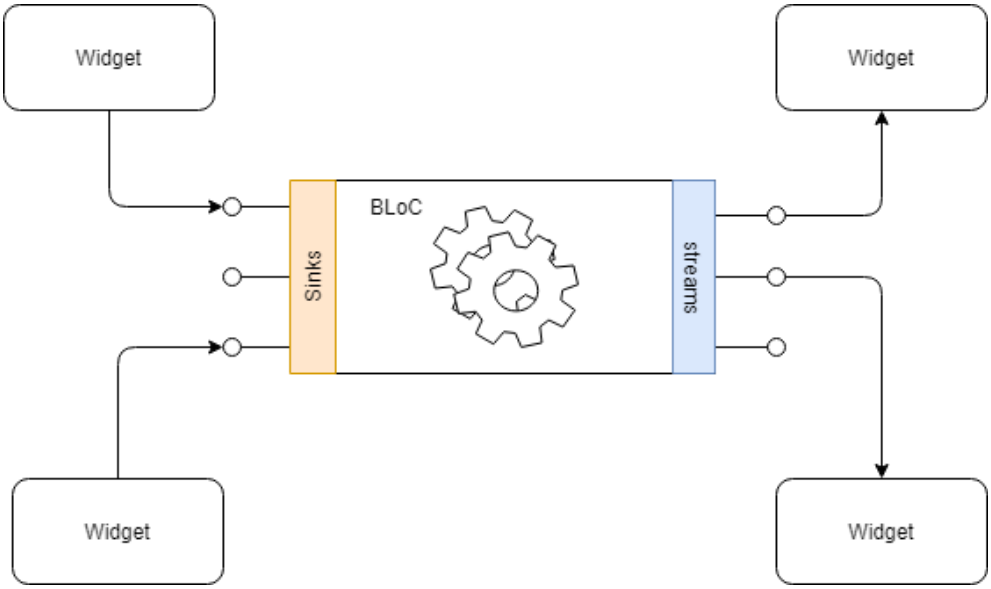
Projektā laikā tika parūpēts par dažām svarīgām kvalitātes nodrošināšanas pasākumiem, tajā skaitā:

- programmatūras prasību definēšana pirms izstrādes;
- programmatūras dažu funkciju projektēšana;
- programmas koda rakstīšana balstoties uz "tīra koda" paradigmām;
- lietotnes testēšana pēc katra jauna moduļa izveidošanas;
- versiju kontrole izmantojot Github.

Programmas kods tika izstrādāts balstoties uz "tīra koda" paradigmām Flutter projektu izstrādē. Šādas paradigmas iekļauj sevī: mainīgo saprotamu nosaukšanu, failu pareizo nosaukšanu, loģikas un lietotāja interfeisu sadalīšanu, kā arī vispārēja projekta failu organizācija. Par loģikas un interfeisu sadali, kā arī par projekta failu organizāciju tiks detalizētāk aprakstīts tālāk.

5.2.1. BLoC modelis

Biznesa loģikas un lietotāja saskarņu sadalīšanai tika izmantots BLoC[1] modelis. Tas atļauj izveidot uz straumēm un notikumiem bāzētu lietotni, kurā var neatkarīgi mainīt biznesa loģiku un lietotāja interfeisu. No lietotāja interfeisiem (Flutter gadījumā tie ir vidžeti) notikumi nonāks BLoC'os, kuros notiks biznesa loģikas darbības, kas atgriezīs straumes atpakaļ uz vidžetiem, tadējadi veidojot dinamisku un neatkarīgo interfeisu. BLoC modeļa grafisko attēlojumu var redzēt 5.1. att.



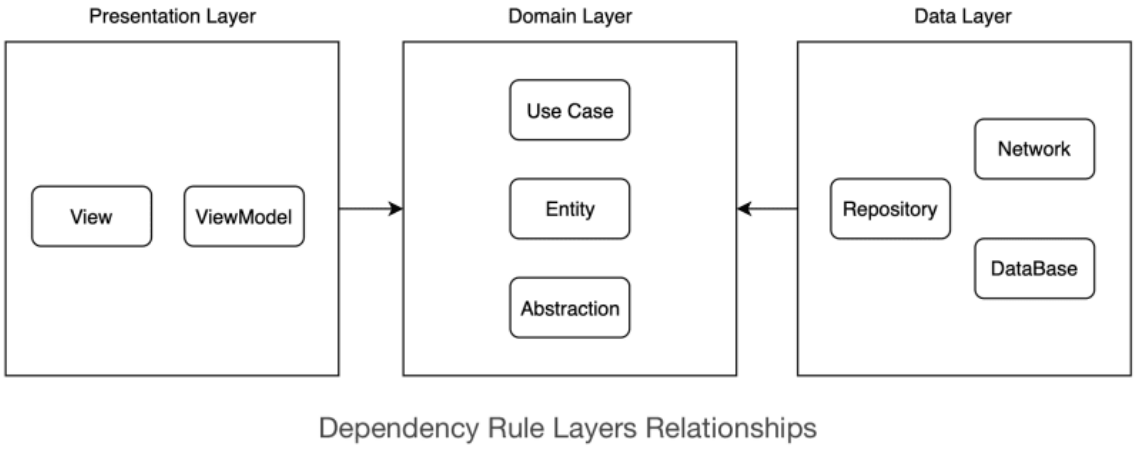
5.1. att. **"BLoC modeļa diagramma"**

5.2.2. "Tīras Arhitektūras" failu organizācijā

Failu organizācija un vispārēja projekta izveidošana tika veikta pēc "Tīras Arhitektūras" principa, kas sadala lietotnes kodu uz 3 slāņiem: prezentācijas slānis, domēna slānis, datu slānis. Šī projekta gadījumā prezentācijas slānis saturēja sevī organizētas mapes ar vidžetiem un to BLoC'iem, šīs slānis atbildēja par to, ko redz lietotājs un loģiku interfeisu darbībai. Domēnā slāņi glabājās entitāšu modeļi un abstraktas repozitorijas (kurās ir definētas funkcijas, bet ne to implementācijas). Savukārt datu slānis saturēja sevī visas klases ar svarīgākām funkcijām (kas bija definētas arī programmatūras prasību specifikācija) un to implementācijas. Tāds princips palīdz ar abstrakcijas palīdzību atdalīt operācijas ar datiem no lietotāju interfeisa, kas savukārt palīdz veikt

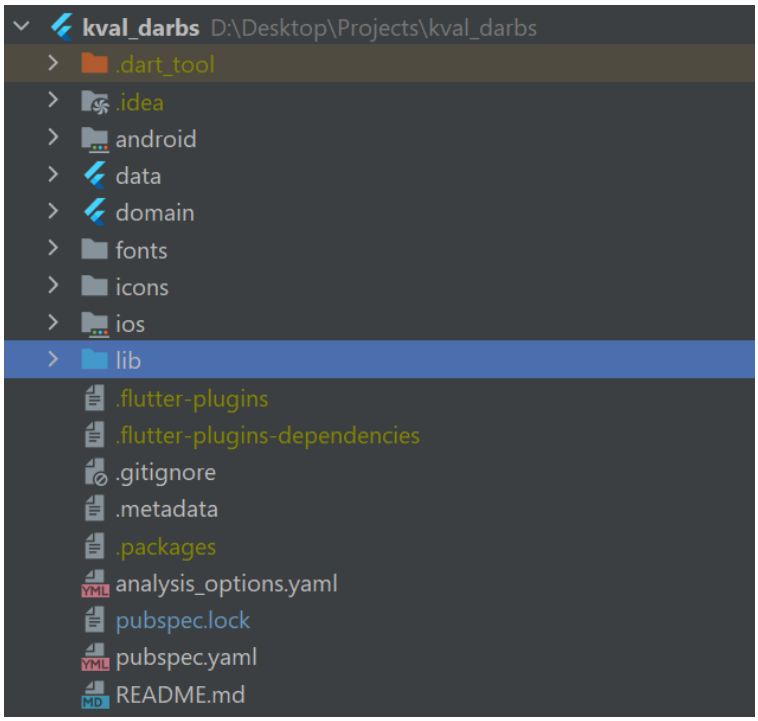
[1] https://www.didierboelens.com/2018/08/reactive-programming-streams-bloc/

izmaiņas tikai tajā vietā, kur tas ir vajadzīgs, piemērām izmainot kāda skata izskatu, nemainot zema līmeņa funkcijas ar datiem. Izmantojot "Tīras Arhitektūras" pieeju tiks izmantoti atkarību noteikumi: prezentācijas slānis ir atkarīgs no domēna slāņa (jo tas izmantos entitāšu modeļus un abstraktās klases un metodes) un datu slānis ir atkarīgs no domēna slāņa (jo arī izmanto entitāšu modeļus un implementē abstrakto klašu metodes), to var redzēt 5.2. att.



5.2. att. **"Atkarības starp tīras arhitektūras slāņiem"**

Attēlā 5.3. var redzēt projekta sadalījumu pa moduļiem/slāņiem, šeit par prezentācijas slāni kalpo Flutter projekta galvenā mape "lib".



5.3. att. **"Projekta sadalījums pa moduļiem"**

5.3. Konfigurāciju pārvaldība

Projekta konfigurāciju pārvaldībai tika izmantots Git rīks, kopā ar repozitoriju Github servisā. Uzglabājot kodu attālinātajā repozitorijā ļāva neuztraukties par koda pazušanu vai kādas kļūdas ietekmi uz tālāko izstrādi, jo vienmēr varēja atgriezties uz pagājušas versijas kodu.

6. DARBIETILPĪBAS NOVĒRTĒJUMS

Darbietilpības novērtējumam tika izvēlēts optimistiska, reālistiska, un pesimistiska vērtējuma metode. Ņemot vērā to, ka pirms darba uzsākšanas autoram nebija pagājušas pieredzes ar Flutter izstrādi, novērtējumā tika iekļauts arī jaunas tehnoloģijas apguve un labu Flutter koda prakšu iemācīšana (labas koda prakses atļautu izveidot kvalitatīvu gala produktu). Papildus pie jaunas tehnoloģijas apguves novērtējuma tika iekļautas:

- Pirmkoda rakstīšana
- Testēšana
- Programmatūras prasību un dokumentācijas izstrāde

Darba izstrādes laikā stundas tika reģistrētas un beigās ar tiem tika izkalkulēta nomināla darbietilpība, skatīt tabulu 6.1.

6.1. Tabula **"Darbietilpības novērtējums"**

	Optimistiskais novērtējums (dienās)	Reālistiskais novērtējums (dienās)	Pesimistiskais novērtējums (dienās)	Izrēķināts novērtējums (dienās)
Flutter tehnoloģijas un Firebase servisa apgūšana	12	14	17	14.1
Prasību un dokumentācijas izstrāde	9	12	16	12.2
Pirmkoda rakstīšana	25	38	48	37.5
Testēšana	2	4	6	4
Kopā	48	68	87	67.8

Darba apjoms tika aprēķināts pēc formulas:

$$S = \frac{S(optimistiskais) + 4 * S(reālistiskais) + S(pesimistiskais)}{6}$$

Ievietojot dotās vērtības formulā sanāk:

$$S = \frac{48 + 4 * 68 + 87}{6} = \frac{407}{6} = 67.8\ darba\ dienas$$

Ja uzskata, kā vienā darba mēnesī ir 20 darba dienas, tad gala darba apjoms cilvēk mēnešos ir:

$$S = \frac{67.8\ darba\ dienas}{20} = 3.4\ cilvēk\ mēneši$$

SECINĀJUMI

Kvalifikācijas darba izstrādes laikā tika izstrādāta lietotne sludinājumu publicēšanai un saziņai starp sludinājumu īpašniekiem. Darba izstrādes laikā tika apgūts Flutter ietvars mobilo lietotņu izstrādei uzreiz uz abām platformām, kas atļautu taupīt laiku izstrādei atsevišķi uz Android un iOS, papildus pie tā izstrādi uz Flutter autors uzskata par patīkamu pieredzi.

Darba ietvaros netika izstrādātas dažas funkcijas, kuru implementācija prasītu lielāku laika apjomu, taču pirmkodā tam ir jau veikta pirmās kārtas sagatave (adreses pievienošana sludinājumam un pievienotās adreses attēlošana Google kartē).

Kopumā kvalifikācijas darba pieredze ir neapšaubāmi vērtīga autora izaugsmei kā izstrādātājam un palīdzes arī turpmākajos mobilās izstrādes projektos.

IZMANTOTĀ LITERATŪRA UN AVOTI

1. Rudīte Čevere, LVS 68:1996, "Programmatūras prasību specifikācijas ceļvedis", 1996.gads, 22 lpp.
2. "Didier Boelens", "Introduction to the notions of Streams, Bloc and Reactive Programming" – [atsauce 04.01.2022] Pieejams: https://www.didierboelens.com/2018/08/reactive-programming-streams-bloc/
3. "Robert C. Martin", "The Clean Architecture" [atsauce 04.01.2022] Pieejams: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
4. "Flutter", "Flutter documentation" [atsauce 04.01.2022] Pieejams: https://docs.flutter.dev
5. "Dart", "Dart documentation" [atsauce 04.01.2022] Pieejams: https://dart.dev/guides
6. "Firebase Flutter", "FlutterFire Overview" [atsauce 04.01.2022] Pieejams: https://firebase.flutter.dev/docs/overview/
7. "Firebase Flutter", "Cloud Firestore" [atsauce 04.01.2022] Pieejams: https://firebase.flutter.dev/docs/firestore/overview/
8. "Firebase Flutter", "Cloud Storage" [atsauce 04.01.2022] Pieejams: https://firebase.flutter.dev/docs/storage/overview/
9. "Firebase Flutter", "Authentication" [atsauce 04.01.2022] Pieejams: https://firebase.flutter.dev/docs/auth/overview/

PIELIKUMI

1. Pielikums. Visu sludinājumu lapa

```
import 'dart:async';
import 'package:dropdown_search/dropdown_search.dart';
import 'package:fastmarket/adverts/advert_details/advert_details_page.dart';
import 'package:fastmarket/adverts/advert_list_tile.dart';
import 'package:fastmarket/adverts/all_adverts_page_bloc.dart';
import 'package:fastmarket/adverts/create_advert_page.dart';
import 'package:fastmarket/theme/colors.dart';
import 'package:fastmarket/theme/typography.dart';
import 'package:fastmarket/widget/show_bottom_sheet_modal.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

class AllAdvertsPage extends StatefulWidget {
  @override
  _AllAdvertsPageState createState() => _AllAdvertsPageState();

  //Create bloc on widget initialization
  static Widget withBloc() => BlocProvider(
    //Add load event right after bloc creation
      create: (context) =>
AllAdvertsPageBloc(context.read())..add(LoadEvent()),
      child: AllAdvertsPage());
}

class _AllAdvertsPageState extends State<AllAdvertsPage> {
  late AllAdvertsPageBloc _bloc;
  final ScrollController _controller = ScrollController();
  StreamSubscription<void>? _refreshStream;

  @override
  void initState() {
    _bloc = context.read();
    _controller.addListener(() {
      //If list reaches max extent - 100px -> load more adverts
      if (_controller.position.pixels >=
              _controller.position.maxScrollExtent - 100 &&
          !_bloc.state.loadingNewData) {
        _bloc.add(LoadMoreEvent());
      }
    });
    _refreshStream = _bloc.refreshStream().listen((event) async {
      //If refresh stream yields new value -> load adverts again
      _bloc.add(LoadEvent());
    });
    super.initState();
  }

  @override
  void dispose() {
    //Don't forget to dispose controllers and streams at end of widget
lifecycle
    _refreshStream?.cancel();
    _controller.dispose();
    super.dispose();
```

```
  }

  @override
  Widget build(BuildContext context) {
    //Return bloc builder to feed body widget the latest state of bloc
    return BlocBuilder<AllAdvertsPageBloc, AllAdvertsPageState>(
        builder: (context, state) {
      return _buildBody(state);
    });
  }

  //Build adverts depending on state of the bloc
  Widget _buildAdverts(AllAdvertsPageState state) {
    if (state.progress) { //Loading body
      return Container(
        height: MediaQuery.of(context).size.width,
        padding:
            const EdgeInsets.only(top: 16, bottom: 16, left: 16, right: 16),
        child: const Center(child: CircularProgressIndicator.adaptive()),
      );
    } else if (state.allAdverts.isEmpty) { //Empty body
      return Container(
          padding:
              const EdgeInsets.only(top: 160, bottom: 16, left: 16, right:
16),
          child: Center(
            child: Column(
              children: const <Widget>[
                Text("There are no advertisements in this selection",
                    style: AppTypography.title),
                SizedBox(height: 24),
                Text("You can add your advertisement by pressing '+' button",
                    style: AppTypography.body1)
              ],
            ),
          ));
    } else { //Body with adverts
      return Expanded(
        child: Padding(
          padding: const EdgeInsets.only(top: 16.0),
          child: ListView.builder(
            //Use builder to save resources/unload unrendered elements
              controller: _controller, //Controller for pagination
              itemBuilder: (item, index) {
                final advert = state.allAdverts[index];
                return AdvertListTile(
                    onTap: () {
                      Navigator.of(context).push(MaterialPageRoute(
                          builder: (context) =>
                              AdvertDetailsPage.withBloc(advert)));
                    },
                    advertisement: advert);
              },
              itemCount: state.allAdverts.length),
        ),
      );
    }
  }
```

```
  //Body of the page, receives bloc's state
  Widget _buildBody(AllAdvertsPageState state) => Scaffold(
        appBar: AppBar(
          actions: <Widget>[
            //Button to refresh adverts manually
            Padding(
                padding: const EdgeInsets.only(right: 20.0),
                child: GestureDetector(
                  onTap: () {
                    _bloc.add(LoadEvent());
                  },
                  child: const Icon(
                    Icons.refresh,
                    size: 24.0,
                  ),
                )),
          ],
          title: Text("Browse advertisements",
              style: AppTypography.app_bar.copyWith(color: AppColors.white)),
          backgroundColor: AppColors.patternBlue,
          shadowColor: AppColors.transparent,
        ),
        body: Stack(children: [
          Container(
              padding: const EdgeInsets.only(top: 16, left: 16, right: 16),
              child: Column(
                children: [
                  //Category selector
                  const Align(
                      child: Text("Select category:"),
                      alignment: Alignment.centerLeft),
                  const SizedBox(height: 8),
                  DropdownSearch<String>(
                      mode: Mode.MENU,
                      showSelectedItems: true,
                      items: const [
                        "All categories",
                        "Electronics",
                        "Clothes",
                        "Vehicles",
                        "Hobby",
                        "Other"
                      ],
                      dropdownBuilder: (context, category) {
                        return Padding(
                          padding: const EdgeInsets.only(left: 8.0),
                          child:
                              Text(category ?? "", style:
AppTypography.body1),
                        );
                      },
                      popupItemBuilder: (context, category, selected) {
                        return Padding(
                          padding: const EdgeInsets.symmetric(
                              vertical: 16.0, horizontal: 16.0),
                          child: Text(category,
                              style: AppTypography.body1.copyWith(
                                  color: selected
                                      ? AppColors.textBlack
```

```
                                        : AppColors.disabledShadow)),
                          );
                        },
                        onChanged: (category) {
                          //When user changes category -> trigger bloc event
                          if (category != null &&
                              category != state.selectedCategory) {
                            _bloc.add(SetCategoryEvent(category));
                          }
                        },
                        selectedItem: state.selectedCategory),
                    _buildAdverts(state) //Build adverts body
                  ],
                )),
            //Button to add new advert
            Positioned(
                right: 24,
                bottom: 24,
                child: FloatingActionButton(
                    heroTag: null,
                    child: const Icon(Icons.add),
                    onPressed: () async {
                      await showFMModalBottomSheet(
                          context: context,
                          builder: (context) => CreateAdvertPage.withBloc());
                    }))
          ]),
        );
}
```

2. Pielikums. Visu sludinājumu lapas BLoC

```
import 'package:domain/model/advertisement.dart';
import 'package:domain/model/error_type.dart';
import 'package:domain/repository/advertisement_repository.dart';
import 'package:equatable/equatable.dart';
import 'package:fimber/fimber.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

class AllAdvertsPageState extends Equatable {
  //State's main parameters
  final bool progress;
  final bool loadingNewData;
  final List<Advertisement> allAdverts;
  final String selectedCategory;
  final ErrorType? error;

  const AllAdvertsPageState(this.progress, this.loadingNewData,
this.allAdverts,
      this.selectedCategory, this.error);

  //Copy constructor to update the state with new values
  AllAdvertsPageState copyWith(
          {bool? progress,
          bool? loadingNewData,
          List<Advertisement>? allAdverts,
          String? selectedCategory,
          ErrorType? error}) =>
```

```
      AllAdvertsPageState(
          progress ?? this.progress,
          loadingNewData ?? this.loadingNewData,
          allAdverts ?? this.allAdverts,
          selectedCategory ?? this.selectedCategory,
          error);

  @override
  List<Object?> get props =>
      [progress, loadingNewData, allAdverts, selectedCategory, error];
}

abstract class AllAdvertsPageEvent extends Equatable {
  @override
  List<Object> get props => [];
}
//Event to load initial data
class LoadEvent extends AllAdvertsPageEvent {}

//Event to clear error state
class ClearErrorEvent extends AllAdvertsPageEvent {}

//Event to change category
class SetCategoryEvent extends AllAdvertsPageEvent {
  final String category;

  SetCategoryEvent(this.category);

  @override
  List<Object> get props => [category];
}

//Event on pagination request
class LoadMoreEvent extends AllAdvertsPageEvent {}

class AllAdvertsPageBloc
    extends Bloc<AllAdvertsPageEvent, AllAdvertsPageState> {
  //Advert repository to use firestore functions
  final AdvertisementRepository _advertisementRepository;

  //Default constructor of bloc has "All Categories" selected
  AllAdvertsPageBloc(this._advertisementRepository)
      : super(AllAdvertsPageState(false, false, [], "All categories", null));

  //Stream to listen for refresh request from advert repository
  Stream<void> refreshStream() => _advertisementRepository.refreshStream();

  @override
  //Map incoming event to correct stream
  Stream<AllAdvertsPageState> mapEventToState(
      AllAdvertsPageEvent event) async* {
    if (event is ClearErrorEvent) {
      yield state.copyWith(error: null);
    } else if (event is LoadEvent) {
      yield* _mapLoadEventToState();
    } else if (event is SetCategoryEvent) {
      yield* _mapSetCategoryEventToState(event);
    } else if (event is LoadMoreEvent) {
      yield* _mapLoadMoreEventToState();
```

```
  }
}

Stream<AllAdvertsPageState> _mapLoadEventToState() async* {
  //Show loading indicator while loading
  yield state.copyWith(progress: true, loadingNewData: true);
  try {
    //Get the first batch of adverts from firestore
    var adverts = await _advertisementRepository
        .getAdvertisements(state.selectedCategory);
    //Yield state with first batch of adverts and cancel loading indicator
    yield state.copyWith(
        progress: false, loadingNewData: false, allAdverts: adverts);
  } catch (ex, st) {
    //On error throw error and log the stacktrace, cancel loading
    Fimber.w("Failed to load initial adverts", ex: ex, stacktrace: st);
    yield state.copyWith(
        error: (ex is AppError ? ex.type : ErrorType.generalError),
        progress: false,
        loadingNewData: false);
  }
}

Stream<AllAdvertsPageState> _mapSetCategoryEventToState(
    SetCategoryEvent event) async* {
  //Show loading indicator while loading
  yield state.copyWith(progress: true, loadingNewData: true);
  try {
    //Load first adverts from selected category
    var adverts =
        await _advertisementRepository.getAdvertisements(event.category);
    //Yield updated state with new loaded adverts and selected category
    //Don't forget to cancel loading indicator
    yield state.copyWith(
        progress: false,
        loadingNewData: false,
        allAdverts: adverts,
        selectedCategory: event.category);
  } catch (ex, st) {
    //On error throw error and log the stacktrace, cancel loading
    Fimber.w("Failed to load initial adverts", ex: ex, stacktrace: st);
    yield state.copyWith(
        error: (ex is AppError ? ex.type : ErrorType.generalError),
        progress: false,
        loadingNewData: false);
  }
}

Stream<AllAdvertsPageState> _mapLoadMoreEventToState() async* {
  yield state.copyWith(loadingNewData: true);
  try {
    //Check if selected category can load more
    final canLoadMore = _advertisementRepository.canLoadMore;
    List<Advertisement> nextAdverts = [];
    if (canLoadMore) {
      //Load next paginated adverts
      nextAdverts = await _advertisementRepository
          .getMoreAdvertisements(state.selectedCategory);
    }
```

```
      //Yield updated state with concatenated next adverts, cancel the loading
indicator
      yield state.copyWith(
          allAdverts: state.allAdverts + nextAdverts, loadingNewData: false);
    } catch (ex, st) {
      //On error throw error and log the stacktrace, cancel loading
      Fimber.w("Failed to load more advertisements", ex: ex, stacktrace: st);
      yield state.copyWith(
          error: ex is AppError ? ex.type : ErrorType.generalError,
          loadingNewData: false);
    }
  }
}
```

3. Pielikums. Sludinājumu repozitorijs no datu slāņa (funkciju implementācija)

```
import 'dart:async';
import 'package:cloud_firestore/cloud_firestore.dart';
import
'package:data/images/profile_avatar/repository/firebase_images_storage_reposit
ory.dart';
import 'package:domain/model/advertisement.dart';
import 'package:domain/model/error_type.dart';
import 'package:domain/repository/advertisement_repository.dart';
import 'package:fimber/fimber.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:rxdart/rxdart.dart';
import 'package:uuid/uuid.dart';

class FirebaseAdvertRepository extends AdvertisementRepository {
  //Use images repository when handling advert deletion
  final FireBaseImagesStorageRepository _fireBaseImagesStorageRepository;
  //Stream to notify pages to refresh adverts
  final StreamController<void> _refreshController = BehaviorSubject();

  @override
  //Refresh stream accessor
  Stream<void> refreshStream() => _refreshController.stream;

  FirebaseAdvertRepository(this._fireBaseImagesStorageRepository);

  //Reference to firestore instance with converter type of Advertisement
  final _advertsRef = FirebaseFirestore.instance
      .collection('adverts')
      .withConverter<Advertisement>(
          fromFirestore: (snapshot, _) =>
              Advertisement.fromJson(snapshot.data()!),
          toFirestore: (advert, _) => advert.toJson());

  //Remember last loaded advert to use pagination (only way firebase allows)
  DocumentSnapshot<Advertisement>? lastAdvert;
  DocumentSnapshot<Advertisement>? myLastAdvert;
  //Remember if there are more items to load to save on excess calls to
firestore
  bool _canLoadMore = true;
  bool _canLoadMoreMyAdverts = true;

  @override
```

```
  Future<void> publishAdvertisement(Advertisement advert) async {
    final userId = FirebaseAuth.instance.currentUser?.uid;
    //Check if user is logged in
    if (userId != null) {
      //Generate unique ad ID
      var uuid = const Uuid();
      final adId = advert.adId.isEmpty ? uuid.v4() : advert.adId;
      try {
        //Add created advert to firestore
        await _advertsRef
            .doc(adId)
            .set(advert.copyWith(adId: adId, ownerId: userId));
        //After adding new advert, trigger refresh stream, as advert list
changed
        _refreshController.add(null);
        return;
      } catch (ex, st) {
        Fimber.w("Failed to publish advert", ex: ex, stacktrace: st);
        throw AppError(ErrorType.addAdvert);
      }
    } else {
      Fimber.w("Failed to get current user");
      throw AppError(ErrorType.generalError);
    }
  }

  @override
  Future<List<Advertisement>> getAdvertisements(String? category) async {
    List<DocumentSnapshot<Advertisement>> firstAdvertBatch;
    List<Advertisement> mappedAdverts = [];
    //If any exact category is chosen
    if (category != "All categories") {
      //Request first 20 documents of selected category (for pagination)
      firstAdvertBatch = await _advertsRef
          .where('category', isEqualTo: category)
          .orderBy("createdAt", descending: true)
          .limit(20)
          .get()
          .then((value) => value.docs);
      //Remember last document of the batch for future pagination
      lastAdvert = firstAdvertBatch.isNotEmpty ? firstAdvertBatch.last : null;
      //Map through DocumentSnapshots and fill mappedAdverts list
      for (var snapshot in firstAdvertBatch) {
        final advert = snapshot.data();
        if (advert != null) {
          mappedAdverts.add(advert);
        }
      }
    } else { //If "All Categories" is selected
      //Load just first 20 of all adverts
      firstAdvertBatch =
          await _advertsRef.limit(20).get().then((value) => value.docs);
      //Remember last document of the batch for future pagination
      lastAdvert = firstAdvertBatch.isNotEmpty ? firstAdvertBatch.last : null;
      //Map through DocumentSnapshots and fill mappedAdverts list
      for (var snapshot in firstAdvertBatch) {
        final advert = snapshot.data();
        if (advert != null) {
          mappedAdverts.add(advert);
```

```
      }
    }
  }
  //If mapped adverts length is < 20, then there are no more adverts to load
  _canLoadMore = mappedAdverts.length == 20;
  return mappedAdverts;
}

@override
Future<List<Advertisement>> getMoreAdvertisements(String? category) async {
  List<DocumentSnapshot<Advertisement>> nextAdvertBatch;
  List<Advertisement> mappedAdverts = [];
  //If any exact category is chosen
  if (category != "All categories") {
    //Request next 20 documents of selected category
    nextAdvertBatch = await _advertsRef
        .where('category', isEqualTo: category)
        .orderBy("createdAt", descending: true)
    //Use saved lastAdvert of previous batch to set starting point
        .startAfterDocument(lastAdvert!)
        .limit(20)
        .get()
        .then((value) => value.docs);
    //Remember last document of the batch for future pagination
    lastAdvert = nextAdvertBatch.isNotEmpty ? nextAdvertBatch.last : null;
    //Map through DocumentSnapshots and fill mappedAdverts list
    for (var snapshot in nextAdvertBatch) {
      final advert = snapshot.data();
      if (advert != null) {
        mappedAdverts.add(advert);
      }
    }
  } else { //If "All Categories" is selected
    //Request next 20 documents of selected category
    nextAdvertBatch = await _advertsRef
    //Use saved last advert from last batch
        .startAfterDocument(lastAdvert!)
        .limit(20)
        .get()
        .then((value) => value.docs);
    //Remember last document of the batch for future pagination
    lastAdvert = nextAdvertBatch.isNotEmpty ? nextAdvertBatch.last : null;
    //Map through DocumentSnapshots and fill mappedAdverts list
    for (var snapshot in nextAdvertBatch) {
      final advert = snapshot.data();
      if (advert != null) {
        mappedAdverts.add(advert);
      }
    }
  }
  //If mapped adverts length is < 20, then there are no more adverts to load
  _canLoadMore = mappedAdverts.length == 20;
  return mappedAdverts;
}

@override
bool get canLoadMore => _canLoadMore;

@override
```

```
  bool get canLoadMoreMyAdverts => _canLoadMoreMyAdverts;

  @override
  Future<void> deleteAdvertisement(Advertisement advert) async {
    final String? userId = FirebaseAuth.instance.currentUser?.uid;
    final ownerId = advert.ownerId;
    //Check if user is logged in and is the owner of deletable advert
    if (userId != null && userId == ownerId) {
      try {
        //First delete image from Firebase Storage
        _fireBaseImagesStorageRepository.deleteAdvertImage(advert);
        //Then delete document from Firestore
        _advertsRef.doc(advert.adId).delete();
        //After deleting advert, trigger refresh stream, as advert list
changed
        _refreshController.add(null);
      } catch (ex, st) {
        Fimber.w("Failed to delete an advert", ex: ex, stacktrace: st);
        throw AppError(ErrorType.generalError);
      }
    } else {
      Fimber.w(
          "Can't delete this advertisement, you are either not authorized or
aren't owner of the advert");
      throw AppError(ErrorType.generalError);
    }
  }

  //Basically the same as getMoreAdvertisements, but without category
  @override
  Future<List<Advertisement>> getMoreMyAdvertisements() async {
    final String? userId = FirebaseAuth.instance.currentUser?.uid;
    if (userId != null) {
      List<DocumentSnapshot<Advertisement>> nextMyAdvertBatch;
      List<Advertisement> mappedAdverts = [];
      nextMyAdvertBatch = await _advertsRef
          .where('ownerId', isEqualTo: userId)
          .orderBy("createdAt", descending: true)
          .startAfterDocument(lastAdvert!)
          .limit(20)
          .get()
          .then((value) => value.docs);
      myLastAdvert =
          nextMyAdvertBatch.isNotEmpty ? nextMyAdvertBatch.last : null;
      for (var snapshot in nextMyAdvertBatch) {
        final advert = snapshot.data();
        if (advert != null) {
          mappedAdverts.add(advert);
        }
      }
      _canLoadMoreMyAdverts = mappedAdverts.length == 20;
      return mappedAdverts;
    } else {
      Fimber.w("Can't load user's adverts because user is not authorized");
      throw AppError(ErrorType.generalError);
    }
  }

  //Basically the same as getAdvertisements, but without category
```

```
  @override
  Future<List<Advertisement>> getMyAdvertisements() async {
    final String? userId = FirebaseAuth.instance.currentUser?.uid;
    if (userId != null) {
      List<DocumentSnapshot<Advertisement>> firstMyAdvertBatch;
      List<Advertisement> mappedAdverts = [];
      firstMyAdvertBatch = await _advertsRef
          .where('ownerId', isEqualTo: userId)
          .orderBy("createdAt", descending: true)
          .limit(20)
          .get()
          .then((value) => value.docs);
      myLastAdvert =
          firstMyAdvertBatch.isNotEmpty ? firstMyAdvertBatch.last : null;
      for (var snapshot in firstMyAdvertBatch) {
        final advert = snapshot.data();
        if (advert != null) {
          mappedAdverts.add(advert);
        }
      }
      _canLoadMoreMyAdverts = mappedAdverts.length == 20;
      return mappedAdverts;
    } else {
      Fimber.w("Can't load user's adverts because user is not authorized");
      throw AppError(ErrorType.generalError);
    }
  }
}
```

Kvalifikācijas darbs „***Sludinājumu portāla mobilās aplikācijas izstrāde***” izstrādāts Latvijas Universitātes Datorikas fakultātē.

Ar savu parakstu apliecinu, ka darbs izstrādāts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: ***Daniils Vitjuks*** **_____ ___.01.2022**.

Rekomendēju darbu aizstāvēšanai
Darba vadītājs: ***B.dat. Tairs Rzajevs*** **_____ ___.01.2022.**

Recenzents: ***Valdis Vizulis***

Darbs iesniegts **10.01.2022**
Kvalifikācijas darbu pārbaudījumu komisijas sekretārs: _____

Darbs aizstāvēts kvalifikācijas darbu pārbaudījuma komisijas sēdē
___.01.2022. prot. Nr. _____

Komisijas sekretārs(-e): _____