TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Rando Luik 164417IAIB

MOBIILIRAKENDUS KROONILISELT HAIGETE LASTE RAVITEEKONNA VIRTUALISEERIMISEKS

Bakalaureusetöö

Juhendaja: Inna Švartsman
MSc

Tallinn 2020

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Rando Luik

23.05.2020

Annotatsioon

Lõputöö eesmärk on luua heategevusorganisatsioonile Vaprusehelmed, kelle eesmärk on krooniliste või eluohtlike haigustega laste psühhosotsiaalne toetamine, eestikeelne mobiilirakendus, mis võimaldab lastel saada oma raviteekonnast digitaalne ülevaade vaprusekee kujul.

Rakendus võimaldab lapsel näha oma vaprusehelmestest kokku lükitud keed, millelt on võimalik teha selgeks, millised helmed ning millises järjekorras on kogunenud. Läbinud raviperioodi, protseduuri, võtte või muu sündmuse, saab laps valida helmeste nimekirjast läbitud sündmusele vastava helme ning selle oma keesse lükkida. Kee pikkusest aimu saamiseks on võimalik seda haiglas leiduvate esemetega pikkuselt võrrelda.

Rakendus loodi kasutades *front-endi* jaoks Flutter SDKd ja *back-endi* jaoks Firebase platvormi.

Valmis rakenduse töötav prototüüp, mis vajab edasiarendamist. Lahendust valideeriti ühe organisatsiooni Vaprusehelmed töötaja poolt, lastes tal rakendust proovida. Tagasiside oli valdavalt positiivne.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 31 leheküljel, 5 peatükki, 19 joonist.

Abstract

Mobile application for virtualizing the journey to recovery of chronically ill children

The aim of this thesis is to develop a mobile application for the Vaprusehelmed charity organization, whose goal is the psychosocial support of chronically ill children or children with life-threatening symptoms. The application enables the children to get an overview of their journey to recovery in the form of a digital bravery cord. The application is in Estonian.

The application enables the child to get an overview of their collected bravery beads to determine which beads has (s)he collected and in which order. Gone through a treatment period, procedure, take or some other event the child can choose the corresponding bead from the bravery beads list and thread it to his/her bravery cord. To get an estimate of how long one's bravery cord has become, it is possible to compare it in length with objects typically found in hospitals.

The application was developed using the Flutter SDK for the front-end development and the Firebase platform for the back-end development.

A working prototype of the application was created, which needs further development.

The solution was validated by allowing one of the employees of the Vaprusehelmed organization to get a hands-on experience with the application. The feedback was mostly positive.

The thesis is in Estonian and contains 31 pages of text, 5 chapters, 19 figures.

Lühendite ja mõistete sõnastik

API	*application programming interface*, rakendusliides
APK	*Android Package*, Android paketi formaat rakenduste levitamiseks ja installeerimiseks
Baas	*Backend-as-a-Service*, pilveteenuse mudel, mille korral rakendus ühendatakse kolmanda osapoole pakutava teenuse *back-endi* poolt kättesaadavaks tehtud APIde ja muude funktsionaalsuste külge
consume	*providerit* tarbima, seal toimuvaid muutusi pealt kuulama
font	kindla suuruse ja kaaluga kirjastiil
JWT	*JSON Web Token*, kahe osapoole vaheliste nõuete turvalise esindamise meetod
NoSQL	mitterelatsiooniline
OS	*operating system*, operatsioonisüsteem
PNG	*Portable Network Graphics*, rastergraafika failivorming, mis toetab kadudeta tihendust
provider	segu sõltuvuste süstimisest (*dependency injection*) ja seisu haldamisest (*state management*)
real-time data	andmed, mis esitatakse koheselt pärast omandamist
SDK	*software development kit*, tarkvaraarenduskomplekt
SVG	*Scalable Vector Graphics*, XMLil põhinev keele spetsifikatsioonide hulk, mis kirjeldab kahemõõtmelist vektorgraafikat
UI	*user interface*, kasutajaliides
widget	hoiab informatsiooni komponendi konfiguratsiooni ning *state'i* kohta, UI komponendi korral ka selle väljanägemise kohta

Sisukord

Jooniste loetelu

1 Sissejuhatus

Vaprusehelmed MTÜ on heategevusorganisatsioon, kelle eesmärk on krooniliste või eluohtlike haigustega laste psühhosotsiaalne toetamine. Haigusdiagnoosiga lapsele kingitakse kee, millesse ta hakkab lükkima vaprusehelmeid. Iga helmes tähistab raviperioodi, protseduuri, võtte või muu sündmuse läbimist.

Lõputöö eesmärk on luua organisatsiooni Vaprusehelmed nägemuse põhjal haigetele lastele eestikeelne mobiilirakendus, mis võimaldab neil saada oma raviteekonnast digitaalne ülevaade vaprusekee kujul.

Eesmärgi saavutamiseks analüüsiti kliendi saadetud Adobe XDs[1] valminud rakenduse disaini alusel rakenduse nõudmisi: määratleti kasutajaliidese nõuded, funktsionaalsus, mida rakendus peab toetama ning funktsionaalsetest nõuetest tulenevalt fikseeriti andmemudelite kirjeldused.

Analüüsi alusel valiti rakenduse realiseerimiseks kasutatud tehnoloogiad, milleks osutus *front-endi* jaoks Flutter SDK ning *back-endi* jaoks Firebase platvorm. Järgnes iteratiivne rakenduse arendamine ja testimine.

Rakendus võimaldab kasutajal näha oma vaprusehelmestest kokku lükitud keed, millelt on võimalik teha selgeks, millised helmed on kogunenud ning millises järjekorras, võrdlemaks seda oma füüsilise keega.

Läbinud raviperioodi, protseduuri, võtte või muu sündmuse, saab laps valida helmeste nimekirjast läbitud sündmusele vastava helme ning selle oma keesse lükkida. Kee pikkusest aimu saamiseks on võimalik seda haiglas leiduvate esemetega pikkuselt võrrelda.

[1] Rakenduste kasutajaliidese ning -kogemuse disainirakendus

Võrdluseks on Google Play Store'is ja App Store'is organisatsiooni Vaprusehelmed [1] rahvusvahelise Hollandi päritolu partnerettevõtte VOKK (Vereniging Ouders, Kinderen en Kanker, inglise keeles Parents, Children and Cancer Association) ja mobiilirakenduste arendusettevõtte Momentous Apps koostöös valminud rakendus BraveryCord. Nimetatud rakendus on loodud samal eesmärgil, kuid sel puudub eesti keele tugi ning lahkneb kasutajaliidese vaatevinklist organisatsiooni Vaprusehelmed ettekujutusest.

Lõputöö teoreetilises osas tehti rakenduse nõutava kasutajaliidese ning toetatava funktsionaalsuse analüüs, kirjeldati realiseeritud andmemudeleid ning põhjendati tehnoloogiate valikut.

Lõputöö praktilises osas kirjeldati rakenduse struktuuri, vaateid, arendusprotsessi, rakenduse valideerimisel saadud tagasisidet ning tulevikuvaateid.

Valmis rakenduse töötav prototüüp, mis vajab edasiarendamist. Valideeriti seda, lastes ühel organisatsiooni Vaprusehelmed töötajal rakendust katsetada.

2 Analüüs

Antud peatükis analüüsitakse rakenduse kasutajaliidese ja funktsionaalseid nõudeid. Samuti kirjeldatakse selles peatükis loodud andmemudeleid ning põhjendatakse rakenduse loomiseks kasutatud tehnoloogiate valikut.

2.1 Funktsionaalsed ja mittefunktsionaalsed nõuded

Kasutajaliides peab olema võimalikult minimalistlik, et lapsel oleks võimalik ekraanil toimuvat kerge jälgida. Samuti peab olema rakendus võimalikult intuitiivne kuna on suunatud eelkõige lastele.

Kasutajale peab andma märku tema andmetega tehtud muudatustest kas siis teatise (*notification*) või tema seadme ekraanil asetleidvate muudatuste näol.

Kasutajal peab olema võimalik luua endale konto ning sellega sisse logida. Õnnestunud autentimise järel tuleb suunata ta tema vaprusekee vaatesse, kuhu navigeerimisel tuleb süsteemi poolt automaatselt lükkida tema keesse kee algust tähistav helmes.

Kasutaja peab saama muuta oma andmeid. Ta peab saama lisada endale profiilipildi, sisestada oma ees- ja perekonnanime, mille tähtedega kaunistatud helmed lükitakse pärast tema vaprusekeesse kee algust tähistava helme järele. Veel peab kasutaja saama määrata, millises haiglas ning osakonnas ta ravil viibib.

Kasutaja peab saama lükkida oma vaprusekeesse helmeid, valides helmeste nimekirjast läbitud raviperioodile, protseduurile, võttele või muule läbitud sündmusele vastava helme. Vajaduse ilmnemisel peab võimaldama tal lisatud helmes ka keest eemaldada.

Kee pikkusest ettekujutuse saamiseks peab saama seda haiglas leiduvate esemetega pikkuselt võrrelda. Sellisteks esemeteks on näiteks pliiats ja tilguti post.

Kasutajate andmed peavad olema turvatud, et mõni autentimata või teine isik ei pääseks teiste kasutajate andmetele ligi.

Rakendus peab töötama nii Android kui ka iOS operatsioonisüsteemi peal töötavatel seadmetel.

2.2 Andmemudelid

Andmete hoiustamiseks kasutatakse Google'i Firebase Realtime Database teenust. Rakenduses luuakse mudelid `Auth`, `User`, `Hospital`, `BraveryCord`, `Bead`, `CompareItem` ja `HttpException`. Joonis 1 esitab Firebase Realtime Database arhitektuuri.

Joonis 1. Firebase Realtime Database arhitektuur.

Autentimise (`Auth`) mudelis on kirjeldatud kasutaja autentimisega seotud andmed. Sinna hulka kuuluvad Firebase Auth API (*Application programming interface*) poolt kasutajale genereeritud ID, samuti Firebase Auth API poolt kasutajale genereeritud JWT (*JSON Web Token)*, JWT aegumiskuupäev ning JWT kehtivustähtaja möödudes kasutaja automaatselt välja logimise *timer*.

Kasutaja (`User`) mudel on vajalik kasutaja andmete käitlemiseks. Selles mudelis hoitakse sarnaselt autentimise mudelile kasutaja IDd, siis kasutaja ees- ja perekonnanime, andmeid haigla kohta, kus kasutaja ravil viibib, ning kasutaja profiilipilti.

Haigla (`Hospital`) mudel hoiab andmeid haigla kohta, kus kasutaja ravil viibib. Haigla mudeli omadusteks on haigla nimi ja raviosakond.

Vaprusekee (`BraveryCord`) mudel kujutab endast kasutaja vaprusehelmestest kokku lükitud keed, mis on seotud kasutaja ID kaudu kasutajaga. Väljadest on tal lisaks kasutaja IDle `List` kasutaja ees- ja perekonnanime tähtedega kaunistatud helmestest ning `List` raviperioodi, protseduuri, võtte või muu sündmuse läbimist sümboliseerivatest helmestest.

Helme (`Bead`) mudel esindab keesse lükitavat helmest. Helmel on kirjeldatud ID väli, pildi väli, tähenduse väli ning pikkuse väli.

Vaprusekeega pikkuselt võrreldava eseme (`CompareItem`) mudelis hoitakse andmeid eseme nimetuse, pildi ning eseme pikkuse kohta.

Kuna Flutteriga arendades soovitatakse kasutada baas `Exception` instantsi asemel `Exception` klassi laiendavaid arendaja poolt defineeritud alamveaklasse, on loodud serverisse päringute tegemise käigus tõstatavate võrguedastusprotokolli erindite esitamiseks `HttpException` mudel.

`HttpException` instantsi kasutatakse serveri pihta tehtud päringute käigus ilmnenud vigade käsitlemiseks, mida rakenduses kasutatud `http` pakett kinni ei püüa. Vigade tunnistamine erineb protokolliti. `HttpException` tõstatakse näiteks kasutaja konto loomisel, kui konto loomiseks sisestatud meiliaadress on juba mõne teise rakenduse kasutaja poolt kasutusel või kasutaja sisestatud parool osutub liiga nõrgaks (Piisava tugevusega parooli tingimus on, et ta on pikem kui kuus tähemärki. Parooli tugevuse kontrolli teostab Firebase Auth API). Samuti tõstatakse eelnimetatud erind kasutaja sisselogimiskatsel, kui proovitakse logida sisse meiliaadressiga, mis ei kuulu ühelegi kasutajale või parool ei ühti sisestatud meiliaadressiga. `HttpExceptioni` mudelil on defineeritud üksnes veasõnumi väli.

2.3 Kasutatud tehnoloogiad

Mobiilirakendust arendades tuleb arvestada, et lõppkasutajateks osutuvad suure tõenäosusega nii Android kui ka iOS operatsioonisüsteemi peal töötavate seadmete omanikud [2]. Seda arvesse võttes otsustati luua rakendus kasutades Flutter SDKd

(*Software development kit*), kuna see võimaldab jooksutada rakendust mõlemal eelnimetatud operatsioonisüsteemi peal töötavatel seadmetel ühe koodibaasiga [3].

2.3.1 Flutter SDK

Flutter SDK on 2015. aastal Google'i poolt välja töötatud kasutajaliidese tööriistaarenduskomplekt. Algusaastail kutsuti seda Sky'ks. Laialdasele avalikkusele tehti Flutter kättesaadavaks 2017. aastal, kuid esimese stabiilse versiooniga tuldi välja alles 2018. aasta detsembris.

Flutter suudab jooksutada rakendusi seadme riistvaralise võimekuse väljavedamisel kuni 120 kaadrit sekundis. Flutter SDK on ehitatud Dart virtuaalmasina peale ning Flutteriga kirjutatud rakenduste programmeerimiskeeleks on samuti Dart [4].

Flutteriga arendatud rakendused töötavad nii Android, iOS kui ka Fuschia operatsioonisüsteemi peal jooksvatel seadmetel ning seda kõike ühe koodibaasiga. Mis veel, Flutter SDK võimaldab luua ka veebirakendusi (*beeta* versioonis) ning töölauarakendusi (*desktop applications*) (macOS operatsioonisüsteemile töölauarakenduste kirjutamine on *alpha* versioonis ning Windowsile ja Linuxile on alles tehnilise ülevaatamise faasis) [5].

2.3.2 Firebase platvorm

Back-endi tegemiseks otsustati Firebase platvormi kasuks. Firebase on Baas (*Backend-as-a-Service*) platvorm. See võimaldab arendajal keskenduda rohkem *front-endi* arendamisele, elimineerides vajaduse kirjutada kogu *back-end* ise, üksnes selle konfiguratsioon.

Firebase platvorm sai alguse 2012. aastal startupist Envolve ning omandati Google'i poolt 2014. aastal [6].

Firebase platvormi kasutati rakenduses serveri, API ning andmehoidla rollis. Kasutajate haldamiseks kasutati Firebase Auth API teenust, mis võimaldab kasutajaid hallata. Autentimiseks kasutati Firebase Auth API meiliaadressi ja parooliga autentimise süsteemi.

Kasutajate andmete hoiustamiseks kasutati Firebase Realtime Database teenust. Firebase Realtime Database on NoSQL (mitterelatsiooniline) andmebaasisüsteem, mis nagu

nimigi ütleb, kasutab andmete käitlemiseks *real-time datat* (andmed, mis esitatakse koheselt pärast omandamist).

Firebase Auth API teenus integreerub Firebase Realtime Database teenusega, mille tõttu on võimalik selle abil kontrollida ligipääsu kasutajate andmetele [7], [8]. Joonis 2 esitab Firebase Realtime Database reegleid piiramaks kasutajate ligipääsu andmetele.

```
{
  "rules": {
    "users": {
      "$uid": {
        ".read": "auth.uid === $uid",
        ".write": "auth.uid === $uid"
      }
    },
    "braveryCord": {
      "$uid": {
        ".read": "auth.uid === $uid",
        ".write": "auth.uid === $uid"
      }
    }
  }
}
```

Joonis 2. Firebase Realtime Database reeglid piiramaks kasutajate ligipääsu andmetele.

3 Rakenduse realisatsioon

Antud peatükis kirjeldatakse rakenduse struktuuri, vaateid, arendusprotsessi ning rakenduse tööd optimeerivaid arendusvõtteid. Samuti seletatakse lahti kasutajakogemuse mugavdamiseks implementeeritud funktsionaalsused.

3.1 Struktuur

Rakendus ehitati Flutter SDK genereeritud rakenduse mallile, mis lõi vajaliku struktuuri rakenduse jooksutamiseks nii Android kui ka iOS operatsioonisüsteemi peal töötavatel seadmetel.

`Android` kaustas paiknevad Android operatsioonisüsteemi spetsiifilised failid. Kataloogis leiduvad failid on vajalikud rakenduse tööle panemiseks Android seadmetel. Rakenduse valmides kasutatakse sealseid faile *app bundle'i*[1] või APK (*Android Package*) ehitamiseks ning seejärel Google Play Store'i avalikustamiseks.

`Ios` kaustas paiknevad iOS operatsioonisüsteemi spetsiifilised failid, mis on vajalikud rakenduse tööle panemiseks iOS seadmetel. Rakenduse valmides kasutatakse sealseid faile App Store'i poolt nõutava failivormingu *build archive* ehitamiseks.

`Assets` kataloogis asuvad rakenduse *fondid* (kindla suuruse ja kaaluga kirjastiil) ning kasutaja seadme ekraani suuruse alusel kaustadesse jagatud pildifailid.

`Pubspec.yaml` failis on defineeritud rakenduse sõltuvused (*dependencies*). Samuti on seal kirjeldatud, kus paiknevad rakenduse süsteemikataloogis rakenduses kasutatud pildifailid, *fondid* ning millised peaksid *fondid* välja nägema.

Rakenduse koodibaasi hõlpsamaks haldamiseks ning ühtse koodistiili hoidmiseks võeti kasutusele Dart *analyser* pakett, mis teostab koodi staatilist kontrolli. See võimaldab ennetada vigade tekkimist juba enne koodi kompileerimist ning suunab arendajat hoidma

[1] Androidi uus ametlik efektiivsem rakenduste avalikustamise ning installeerimise formaat

ühtset koodistiili. Paketi konfigureerimiseks on loodud `analysis_options.yaml`[1] fail. Joonis 3 esitab Dart *analyzer* paketi konfiguratsioonifaili `analysis_options.yaml` osade konfiguratsiooni reeglitega.

```
include: package:pedantic/analysis_options.yaml

analyzer:
  exclude: [build/**]
  errors:
    # treat missing required parameters as a warning (not a hint)
    missing_required_param: warning
    # treat missing returns as a warning (not a hint)
    missing_return: warning
    ...

linter:
  rules:
    # these rules are documented on and in the same order as
    # the Dart Lint rules page to make maintenance easier
    # https://github.com/dart-lang/linter/blob/master/example/all.yaml
    avoid_bool_literals_in_conditional_expressions: true
    avoid_unused_constructor_parameters: true
    cancel_subscriptions: true
    empty_statements: true
    prefer_const_constructors: true
    prefer_const_declarations: true
    prefer_typing_uninitialized_variables: true
    unnecessary_brace_in_string_interps: true
    unnecessary_parenthesis: true
    ...
```

Joonis 3. Dart *analyzer* paketi konfiguratsioonifail `analysis_options.yaml` osade konfiguratsiooni reeglitega.

Linteri reeglite alustalaks võeti `pedantic` paketis defineeritud reeglid. `Pedantic` pakett on Google'i poolt kokku pandud Dart *analyzer* paketi reeglite konfiguratsioonifail, mida Google'i arendajad kasutavad ise Flutter SDKd arendades. Lisaks `pedantic` paketis defineeritud reeglitele võeti kasutusse töö autori arust reeglid, mis suunavad arendajat kirjutama võimalikult puhast ning optimeeritud koodi.

[1] `Analysis_options.yaml` failis on kirjeldatud, millistele kataloogidele või failidele Dart *analyser* pakett kontrolli teostab ning millistele reeglitele peab või millisele stiilile võiks kood vastata. Samuti konfigureeritakse selles failis, millistel tingimustel kuvada arendajale veateateid või hoopis hoiatusi.

3.1.1 Rakenduse spetsiifilised failid

Rakenduse spetsiifiline kood asub `Lib` kataloogis. `Lib` kataloogis paiknevad rakenduse kasutajaliidese ning funktsionaalsuse ehitamiseks vajaminevad failid. Need jagati eesmärgi ja kasutusjuhtude alusel eri kaustadesse.

`Main.dart` on rakenduse kõige olulisem fail. Selles failis leiab aset rakenduse alglaadimine (*bootstrapping*). Peale rakenduse alglaadimise teostamise registreeritakse `main.dart` failis rakenduses realiseeritud *providerid* (segu *dependency injectionist* ja *state managementist*).

Providerites implementeeriti serverisse päringute tegemise loogika. Samuti hoitakse neis *application wide state'i*. See päästis arendajat kirjutamast ülemäärast koodi rakenduse *state'i* üles tõstmiseks (*lifting state up*), proovides teha muutujaid mitmele rakenduse osale kättesaadavaks. Sisuliselt täidavad *providerid* teenuse rolli. *Providerid* kirjeldati `providers` kataloogis.

Veel defineeriti `main.dart` failis rakenduse üldine *theme* ning rakenduse navigatsioonipuu ehk millistele vaadetele on rakenduses võimalik navigeerida. Navigatsioonipuu realiseerimisel otsustati nimeliste *route'ide* kasuks. Nimeliste *route'ide* alternatiiv oleks kasutada *route'ide* dünaamilist genereerimist. Nimeliste *route'ide* kasutamine tähendab, et vaadete vahel liikumine toimub igale vaatele määratud *route'i* nime alusel. Joonis 4 esitab nimelist *route'i* (autentimise vaade).

```
class AuthPage extends StatefulWidget {
  static const routeName = '/auth';

  @override
  _AuthPageState createState() => _AuthPageState();
}
```

Joonis 4. Nimeline *route* (autentimise vaade).

Route'i dünaamiliseks genereerimiseks tuleb erinevalt nimelisest *route'ist* luua vaatest hoopis instants. Vaated paiknevad `pages` kataloogis.

`Models` kaustas on defineeritud rakenduse mudelid. `Constants.dart` failis on kirjas konstandid. Praeguse seisuga on seal kirjas üksnes Firebase Realtime Database teenuse veebiaadress.

Helme mudeli põhjal loodud instantsid paiknevad `beads.dart` failis ning eseme mudeli põhjal loodud instantsid, millega saab kasutaja vaprusekeed pikkuselt võrrelda, kirjeldati `compare_items.dart` failis.

`Widgets` kaustas defineeriti *widgetid* (hoiab informatsiooni komponendi konfiguratsiooni ja *state'i* kohta, UI (*user interface*) komponendi korral ka selle väljanägemise kohta), mida ei kasutata rakenduses vaadetena.

Flutteris on tegelikkuses kõik asjad *widgetid*, sealhulgas ka vaated. Vaateid eristab teistest *widgetitest* asjaolu, et vaated katavad kogu seadme ekraani pinna. Vaate *widgeti* juurelemendiks on tüüpiliselt `Scaffold` *widget*. Joonis 5 esitab vaate *widgetit* (autentimise vaade).

```
class _AuthPageState extends State<AuthPage> {
  ...

  @override
  Widget build(BuildContext context) {
    return Scaffold(
       appBar: AppBar(
         ...
      ),
      body: ...
```

Joonis 5. Vaate *widget* (autentimise vaade).

Loodud failide struktuur aitab jagada rakenduse loogilisteks tükkideks. Proovitakse hoida eri ülesandeid täitvad koodijupid üksteisest eraldatuna (*seperation of concerns*).

3.2 Vaated

Selles peatükis kirjeldatakse rakenduses loodud vaateid. Kokku loodi seitse vaadet.

Kuna valmis rakenduse prototüüp, vajab kasutajaliides edasist tööd. Seepärast toodi järgnevates alampeatükkides lõpp-produkti disaini kõrval välja rakenduse kasutajaliidese praegune disain (ekraanitõmmised (*screenshot*) tehti telefoniga OnePlus 7T).

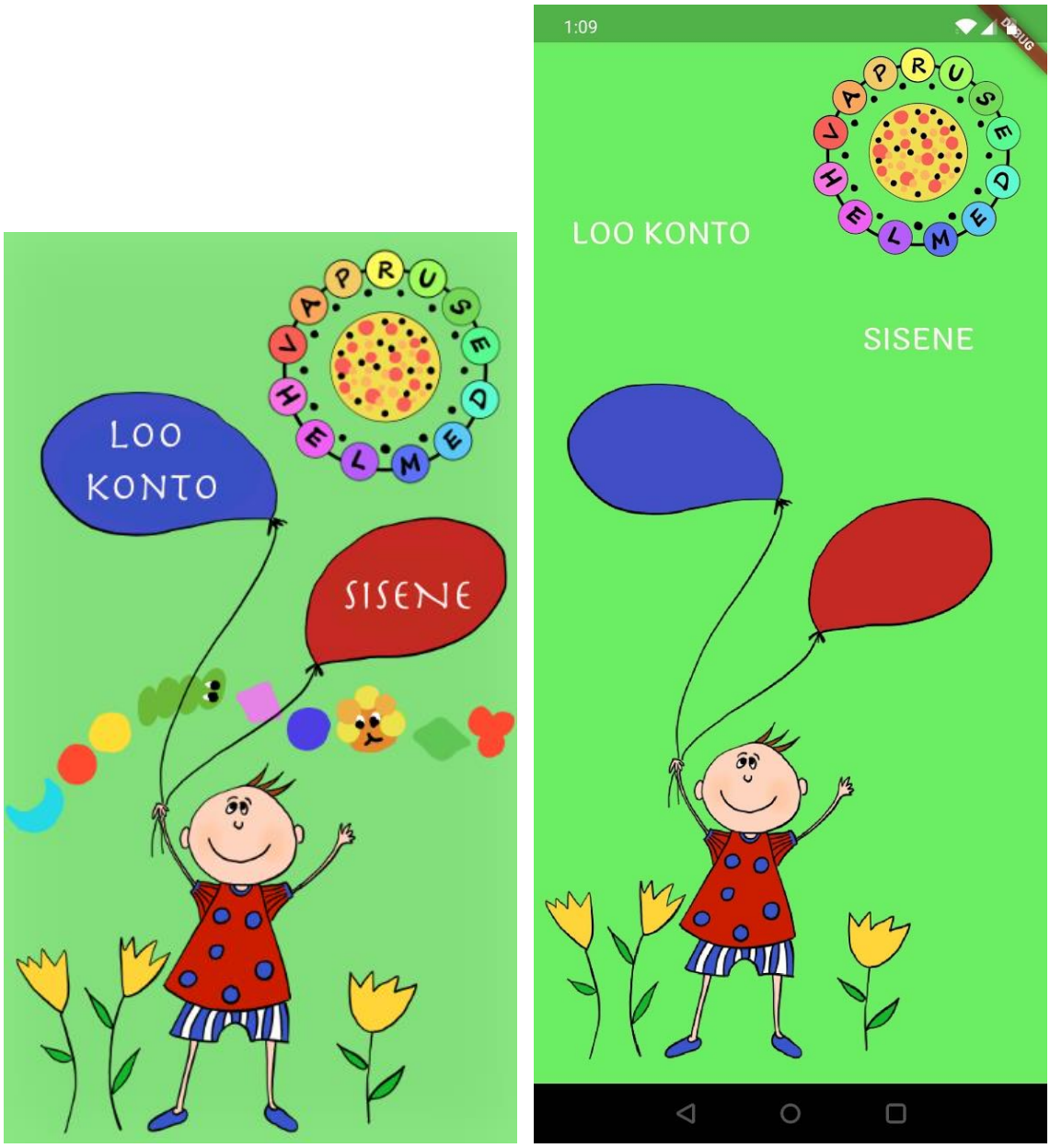
3.2.1 Avavaade

Avavaade (`splash_page`) on esimene vaade, kuhu kasutaja satub rakenduse käivitamisel. Avavaadet kuvatakse kasutajale seni, kuni `auth` *provider* kontrollib, kas kasutajat on võimalik automaatselt sisse logida ning võimaluse korral seda ka teeb.

Kasutaja automaatselt sisse logimise eeltingimus on, et ta on lähiajal eelnevalt juba korra rakenduses ennast autentinud ning Firebase Auth API poolt talle väljastatud JWT on veel kehtiv. Kui need tingimused on täidetud, suunatakse kasutaja tema vaprusekee vaatesse, vastasel juhul kuvatakse kasutajale rakenduse koduvaadet.

3.2.2 Koduvaade

Koduvaadet (`home_page`) kuvatakse kasutajale juhul, kui tuli välja, et kasutajat polnud võimalik süsteemi poolt automaatselt sisse logida. Joonis 6 esitab koduvaate lõpp-produkti disaini vasakul pool ning praegust disaini paremal pool.



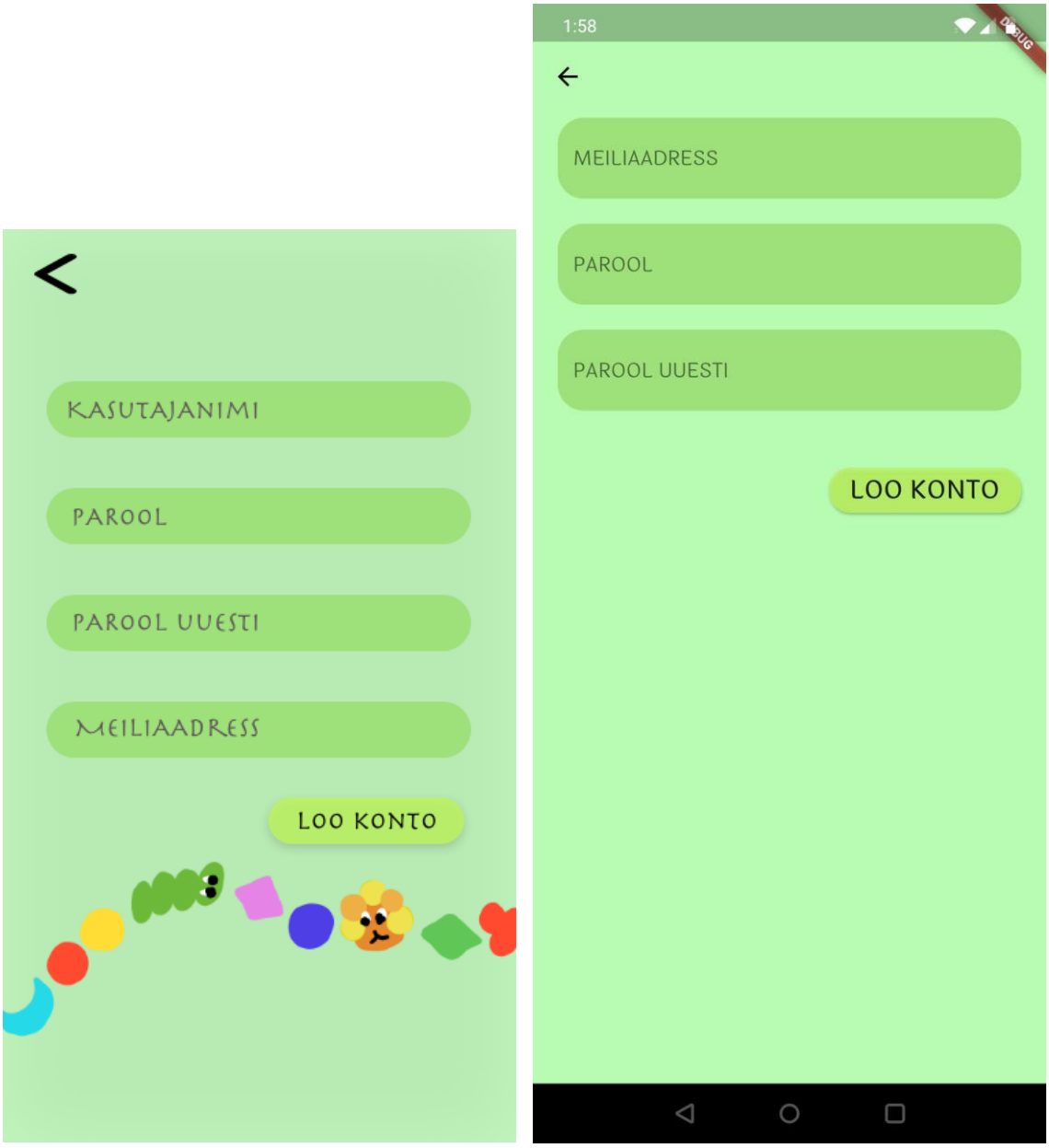
Joonis 6. Koduvaate lõpp-produkti disain (vasak pool) ning praegune disain (parem pool).

Rakenduse kasutamiseks peab kasutaja ennast autentima. Koduvaatest saab kasutaja navigeerida autentimise vaatesse, kus tal on võimalik konto puudumisel see endale luua või juba olemasoleva konto korral sellega sisse logida.

3.2.3 Autentimise vaade

Autentimise vaates (`auth_page`) saab kasutaja luua endale konto või juba olemasoleva korral sellega sisse logida. Sõltuvalt tegevusest kuvatakse kasutajale erinevat vormi.

Joonis 7 esitab autentimise vaate (konto loomise vorm) lõpp-produkti disaini vasakul pool ning praegust disaini paremal pool.



Joonis 7. Autentimise vaate (konto loomise vorm) lõpp-produkti disain (vasakul pool) ning praegune disain (paremal pool).

Lisa 1 esitab autentimise vaate (sisselogimise vorm) lõpp-produkti disaini vasakul pool ning praegust disaini paremal pool.

Rakenduses konto loomine ning sellega sisselogimine peaks disaini põhjal toimuma kasutades kasutajanime ja parooli. Lõputöö autor pidas mõistlikumaks realiseerida see kasutades kasutajanime asemel meiliaadressi. Kuna disainis on nii või teisiti palutud

kasutajal kasutaja vaates sisestada oma meiliaadress, võimaldab langetatud otsus kummaski vormis, nii konto loomise vormis kui ka kasutaja andmete vormis, üks väli ära jätta. Kasutaja jaoks tähendab see vähemate andmete meelespidamist.

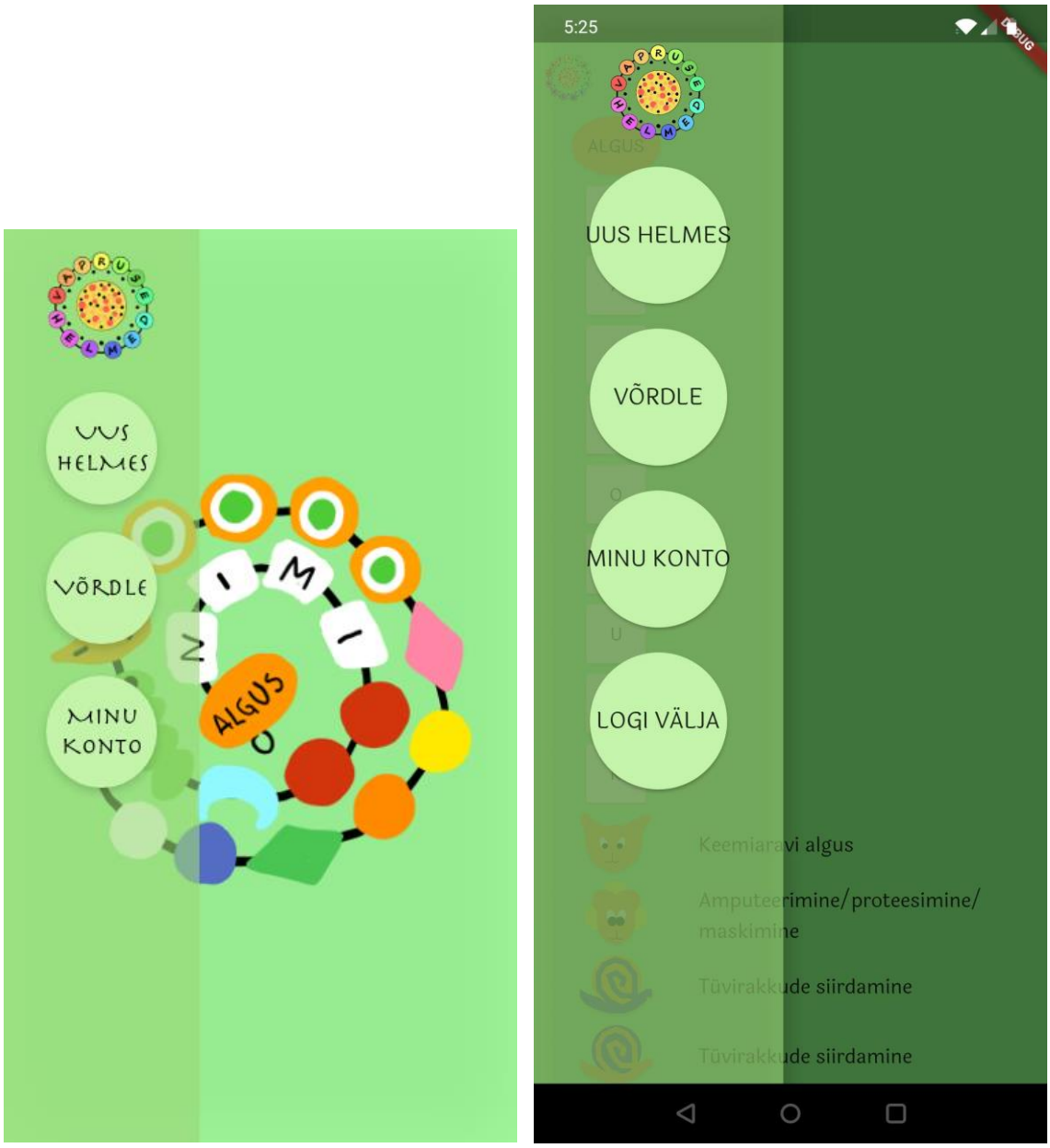
Tulevikus juurdearendatava funktsionaalsuse peale mõeldes on see samuti mõistlik otsus, kuna kasutaja parooli lähtestamine toimub samuti kasutades meiliaadressi.

Arendaja tööd lihtsustab see niivõrd, et Firebase Auth API pakub väga lihtsasti rakendusega integreeritavat meiliaadressiga kasutajate autentimise süsteemi.

Arendaja ei saa aga kliendiga läbiarutamata disaini enesealgatuslikult muuta, mille tõttu tuleb võetud lähenemine kindlasti organisatsiooniga Vaprusehelmed kooskõlastada.

3.2.4 Vaprusekee vaade

Õnnestunud autentimise järel suunatakse kasutaja tema vaprusekee vaatesse (`bravery_cord_page`). Joonis 8 esitab vaprusekee vaate (avatud peamenüü) lõpp-produkti disaini vasakul pool ning praegust disaini paremal pool.



Joonis 8. Vaprusekee vaate (avatud peamenüü) lõpp-produkti disain (vasakul pool) ning praegune disain (paremal pool).

Lisa 2 esitab vapruse kee vaate lõpp-produkti disaini vasakul pool ning praegust disaini paremal pool. Lisa 3 esitab vapruse kee vaate (avatud esemega võrdlemise alammenüü) lõpp-produkti disaini vasakul pool ning praegust disaini paremal pool.

Vaprusekeesse on esialgu lükitud üksnes kee algust tähistav helmes. Kasutaja nimetähtedega kaunistatud helmeste keesse lükkimiseks on vaja uuendada kasutaja vaates kasutaja andmeid, sisestades vastavatesse vormi lahtritesse kasutaja ees- ja

perekonnanimi. Nimetähtedega helmed lükitakse seejärel kee algust tähistava helme järele.

Kasutaja seadme ekraanil tema füüsilist vaprusekeed illustreeriva digitaalse kee instants saadakse vaprusekee *providerist*. Selle jaoks küsitakse esmalt serveri andmebaasist kasutaja andmed, saamaks kätte tema ees- ja perekonnanimi, ning seejärel raviperioode, protseduure jms sümboliseerivad kasutaja keesse lükitud helmed. Saanud serverist vastuse, annab vaprusekee *provider* vaprusekee vaatele teada, et kee instants on nüüd olemas ning palun uuenda kasutajale kuvatavat infot. *Provideris* toimunud muutustele reageerimist nimetatakse *provideri consume'imiseks* (*providerit* tarbima, seal toimuvaid muutusi pealt kuulama). Joonis 9 esitab *provideri consume'imist* (vaprusekee vaade).

```
class BraveryCordPage extends StatelessWidget {
  ...

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        ...
      ),
      drawer: Menu(),
      body: FutureBuilder(
        future: _reloadBeads(context),
        builder: (_, braveryCordSnapshot) => braveryCordSnapshot
                    .connectionState ==
                ConnectionState.waiting
            ? const Center(
                child: CircularProgressIndicator(),
              )
            : Consumer<BraveryCord>(
                builder: (context, braveryCord, _) {
                  final beads = braveryCord.beads;
                  return ListView.builder(
                    itemCount: beads.length,
                    itemBuilder: (_, i) {
                      final bead = beads[i];
                      return [STARTING_BEAD.id, CHARACTER_BEAD.id]
                                .contains(bead.id)
                            ? _buildBead(
                                stack: true,
                                image: bead.image,
                                meaning: bead.meaning,
                              )
                            : Dismissible(
                                key: UniqueKey(),
                                child: _buildBead(
                                  image: bead.image,
                                  meaning: bead.meaning,
                                ),
                                ...
```

Joonis 9. *Provideri* consume'imine (vaprusekee vaade).

Andmaks pealtkuulajatele märku, et *provideri state* on muutunud ning oleks tarvis UId värskendada, on `provider` paketis defineeritud `notifyListeners` nimeline funktsioon. Joonis 10 esitab `notifyListeners` funktsiooni (serverist helmeste pärimine).

```
Future<void> getAndSetBeads() async {
  var url = '${c.dbUrl}/users/$_userId.json';
  ...
  final List<Bead> beads = [];

  try {
    ...
    url = '${c.dbUrl}/braveryCord/$_userId.json';
    response = await http.get(url);
    data = json.decode(response.body);
    if (data != null) {
      (data['beads'] as List<dynamic>).forEach((id) {
        final bead = getBeadById(id);
        beads.add(
          Bead(
            id: id,
            image: bead.image,
            meaning: bead.meaning,
            length: bead.length,
          ),
        );
      });
    }
    _beads = beads;
    notifyListeners();
  } catch (error) {
    print(error);
    rethrow;
  }
}
```

Joonis 10. `NotifyListeners` funktsioon (serverist helmeste pärimine).

Kasutaja andmete haldamiseks on loodud kasutaja *provider*. Kui kasutaja *provideri state* muutub, oskab vaprusekee *provider* vastavalt sellele oma *state'i* uuendada. See on võimalik seetõttu, kuna *providerid* on samuti võimelised teistes *providerites* toimunud muutustele reageerima. Selle jaoks on olemas spetsiaalne `ChangeNotifierProxyProvider` *widget*. Joonis 11 esitab `ChangeNotifierProxyProvider` *widgetit* (vaprusekee *provider* kuulab pealt `Auth` ja User *providerit*).

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ...
        ChangeNotifierProxyProvider2<Auth, User, BraveryCord>(
          create: (_) => BraveryCord(),
          update: (_, auth, user, braveryCord) =>
              braveryCord..update(auth.userId, auth.token),
        ),
      ],
      child: ...
```

Joonis 11. `ChangeNotifierProxyProvider` *widget* (vaprusekee *provider* kuulab pealt `User` *providerit*).

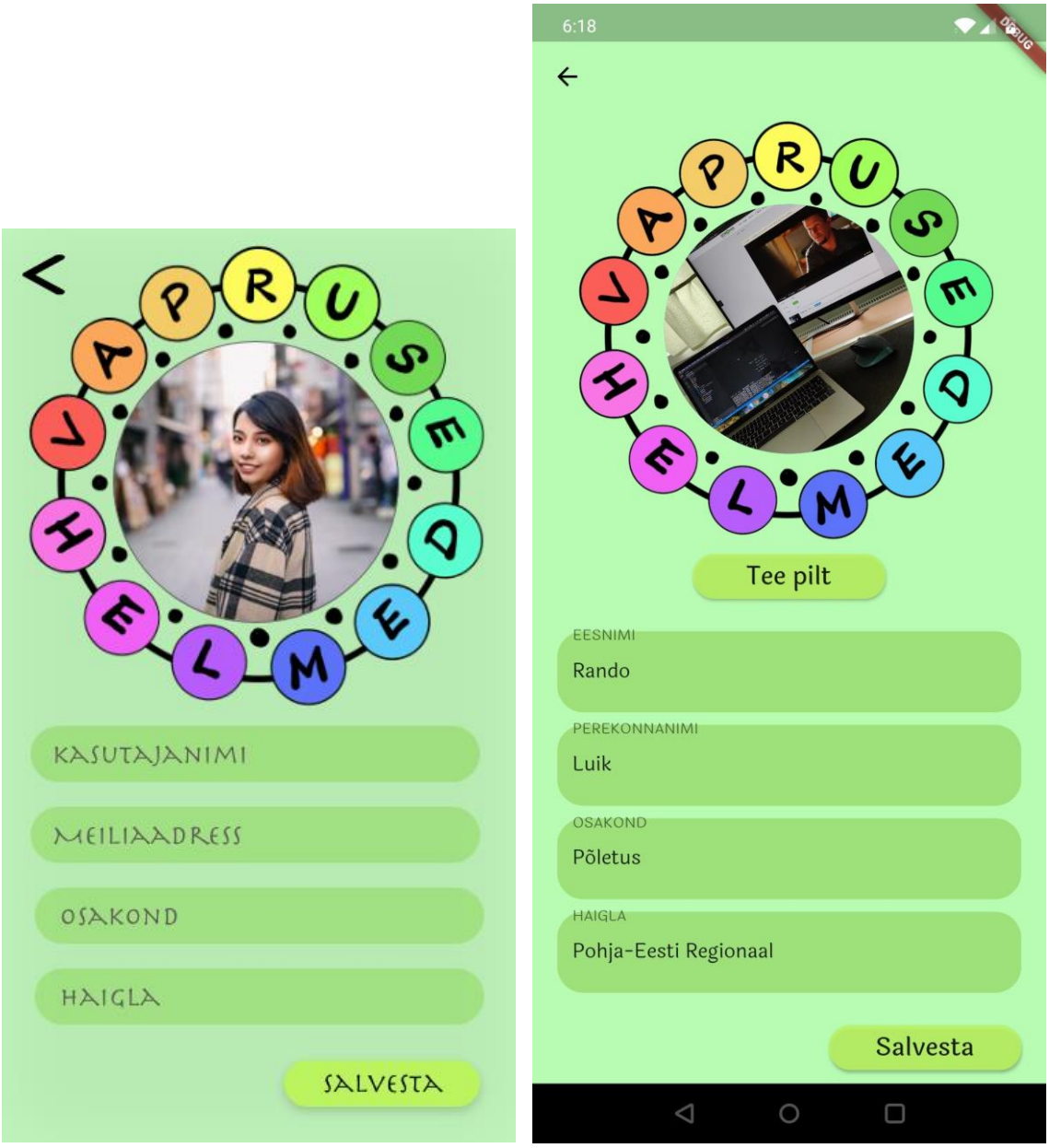
Kui kasutaja kee osutub pikemaks, kui seda on võimalik ekraanil kuvada, saab ülejäänud kee nägemiseks kerida vaates alla poole.

Helmeid saab ka keest eemaldada. Selle jaoks lohistada valitud helmest ekraani vasaku serva suunas. Selle peale küsitakse kasutajalt kinnitust, kas soovitakse tõepoolest helmes keest eemaldada, vältimaks apsaka puhul kogemata keega manipuleerimist. Kee algust tähistavat helmest ning kasutaja nimetähtedega kaunistatud helmeid pole võimalik keest nimetatud viisil eemaldada. Nimetähtedega helmeste välja vahetamiseks tuleb uuendada kasutaja vaates kasutaja ees- ja perekonnanime.

Vaate ülevalt vasakust nurgast on võimalik avada organisatsiooni Vaprusehelmed logole vajutades menüü, kust saab navigeerida uue helme lisamise vaatesse või kee pikkuselt esemega võrdlemise vaatesse. Viimase korral tuleb valida avanevast alammenüüst soovitud ese. Samuti saab menüüst navigeerida kasutaja vaatesse või rakendusest välja logida.

3.2.5 Kasutaja vaade

Kasutaja vaates (`user_page`) on võimalik uuendada kasutaja andmeid. Joonis 12 esitab kasutaja vaate lõpp-produkti disaini vasakul pool ning praegust disaini paremal pool.



Joonis 12. Kasutaja vaate lõpp-produkti disain (vasakul pool) ning praegune disain (paremal pool).

Kasutaja saab lisada endale oma telefoni kaameraga tehtud profiilipildi, sisestada oma ees- ja perekonnanime, mille tähtedest lükitakse pärast keesse vastavate nimetähtedega kaunistatud helmed. Andmete vormis on ka sisendid haigla andmete kohta, kus kasutaja ravil viibib. Haigla andmete alla kuuluvad haigla nimi ja raviosakond.

Nagu eelnevalt mainiti, on Adobe XDs[1] valminud disainiga võrreldes valitud kasutaja autentimiseks kasutajanime asemel meiliaadress, mille tõttu vastav sisend ka kasutaja andmete muutmise vormis puudub.

Hetkel on lisamata ka meiliaadressi lahter, kuna meiliaadressi muutmise funktsionaalsust veel ei toetata.

3.2.6 Helme lisamise vaade

Helme lisamise vaates (`add_bead_page`) kuvatakse kasutajale helmeste nimekirja, mille seast on võimalik lükkida oma keesse uusi helmeid. Joonis 13 esitab helme lisamise vaate lõpp-produkti disaini vasakul pool ning praegust disaini paremal pool.

Joonis 13. Helme lisamise vaate lõpp-produkti disain (vasakul pool) ning praegune disain (paremal pool).

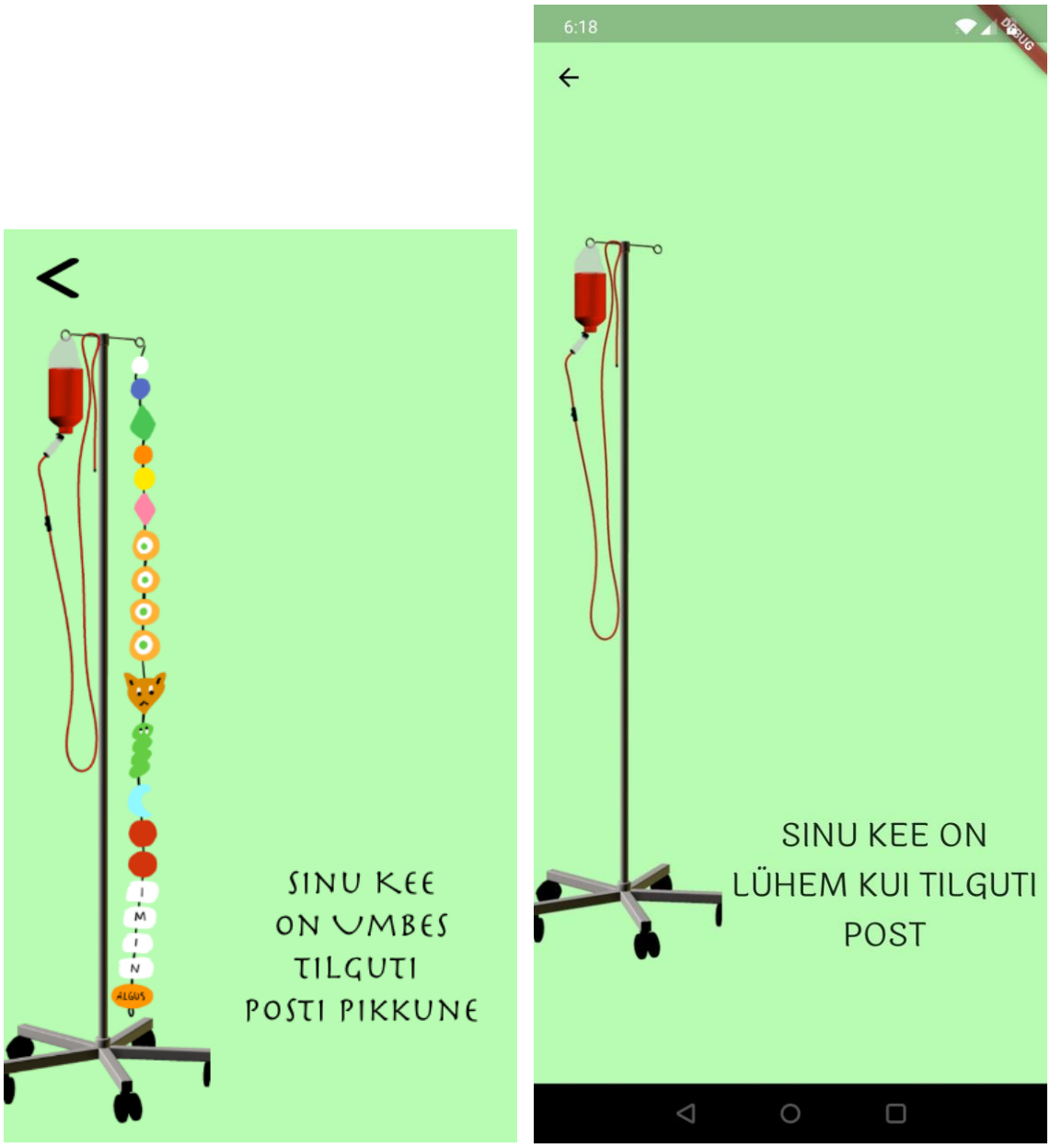
Lisa 4 esitab helme lisamise vaate (helme ülekate) lõpp-produkti disaini vasakul pool ning praegust disaini paremal pool.

Nimekirjast saab valida helme, mis pärast raviperioodi, protseduuri, võtte või muu sündmuse läbimist oma vaprusekeesse lükkida. Leidnud soovitud helme või tahtes teada saada, mis protseduuri, sündmust vms helmes sümboliseerib, tuleb vajutada ekraanil vastavale helmele, mille korral kuvatakse kasutajale helme detailidega ülekatet (*overlay*), kus on näha helme pilti ning tähendust. Avanenud vaatest on võimalik vastav helmes oma keesse lükkida või tegevus katkestada. Keesse lükitud helmest on võimalik näha kee lõpus.

3.2.7 Esemega võrdlemise vaade

Rakenduses on loodud esemete instantsid, millega kasutaja saab oma vaprusekeed pikkuselt võrrelda. Neil esemetel on defineeritud ühe omadusena pikkus. Sama omadus on defineeritud ka igal vaprusehelmel. Esemega võrdlemise vaatesse navigeerimisel arvutatakse vaprusekee *provideris* kasutaja kee pikkus, liites kokku kõikide sellesse lükitud helmeste pikkused ning seejärel võrreldakse seda valitud eseme pikkusega.

Sõltuvalt pikkuste erinevusest kuvatakse kasutajale vastava teade: kas tema vaprusekee on pikem, umbes sama pikkune või lühem kui valitud ese. Joonis 14 esitab esemega võrdlemise vaate (tilguti post) lõpp-produkti disaini vasakul pool ning praegust disaini paremal pool.



Joonis 14. Esemega võrdlemise vaate (tilguti post) lõpp-produkti disain (vasakul pool) ning praegune disain (paremal pool).

Lisa 5 esitab esemega võrdlemise vaate (pliiats) lõpp-produkti disaini vasakul pool ning praegust disaini paremal pool.

3.3 Arendusprotsess, rakenduse tööd optimeerivad arendusvõtted, kasutajakogemuse mugavdamine

Töö käigus loodud rakendus realiseeriti rakendades autorile parimaid teadaolevaid teadmisi ning oskusi.

Flutter SDK valiti rakenduse tegemiseks peamise põhjusega, et see võimaldab jooksutada rakendust sama koodibaasiga nii Android kui ka iOS operatsioonisüsteemi peal töötavatel seadmetel. See vähendas arendaja koormust kirjutada rakendus eraldi koodibaasiga kummalegi operatsioonisüsteemile, mis oleks nõudnud teadmisi Android operatsioonisüsteemi puhul Kotlin programmeerimiskeelest ja iOS operatsioonisüüstemi puhul Swift programmeerimiskeelest.

Leides vajaduse sellegipoolest kirjutada operatsioonisüsteemile *native* koodi põhjusel, et Flutter SDK ei võimalda vajatud funktsionaalsust implementeerida, võimaldab Flutter kirjutada ka Kotlinit, Swifti või muud OSi (operating system) poolt toetatud programmeerimiskeelt. Selle jaoks tuleb Flutter *engine'ile* öelda, kas soovitakse kasutada Androidi puhul Javat või Kotlinit, iOSi puhul aga ObjectiveCd või Swifti.

Kuna Flutter SDK oli lõputöö autori jaoks uus tehnoloogia, läbiti enne rakenduse planeerima hakkamist veebis Udemy õppeplatvormil Maximilian Schwarzmüller, professionaalne veebirakenduste arendaja ja instruktor, juhendamisel läbi kursus „*A Complete Guide to the Flutter SDK & Flutter Framework for building native iOS and Android apps*“ [9].

Veebikursust on ajakohastatud Flutteri edasiarendusega ning see katab väga hästi 323 videoloengu, mis kestavad kokku ligikaudu 36 tundi, kõik peamised Flutter SDK poolt arendajale pakutavad funktsionaalsused Flutteriga rakenduste loomiseks. Lisaks on kursuses pühendatud terve sektsioon Flutter SDK sisemisele ehitusele „*Widget & Flutter Internals - Deep Dive*“, mis koosneb 15 videoloengust, kestvusega umbes kaks tundi, tutvustamaks, kuidas Flutter töötab nö kardina taga. Sektsioonis pööratakse õppuri tähelepanu muuhulgas ka koodi optimeerimisele, võimalikult efektiivse koodi kirjutamisele, vähendamaks Flutter *engine'i* koormust ning tema poolt tehtava töö mahtu.

Veebikursuse käigus tutvuti ka Firebase platvormiga. Tehnoloogia tundus huvitav ning võimekas, mis tõttu otsustati seda lõputöös realiseeritavas rakenduses kasutada. Üheks põhjusteks võib tuua asjaolu, et Firebase Auth API pakub autentimise süsteemi, mis integreerub veel Firebase Realtime Database teenusega kontrollimaks kasutajate ligipääsu andmebaasis hoitavatele andmetele. Kogu selle funktsionaalsuse ise valmis kirjutamine oleks ressurssimahukas ning mitte just kõige lihtsam ettevõtmine.

3.3.1 Koodi optimeerimine

Koodi kirjutades prooviti vältida olukordi, kus *Flutter engine* peab *widgeteid*, mis kunagi ei muutu, uuesti ekraanile renderdama ning joonistama.

Iga kord, kui *widget* luuakse, kutsutakse välja tema konstruktor ning *build* meetod, mis annab süsteemile juhised, kuidas *widgeti* UI representatsioon üles ehitada ning seejärel ekraanile kuvada. Samuti kutsutakse *build* meetod välja iga kord kui muudetakse *widgeti state'i*, kutsudes välja spetsiaalset meetodit `setState`, mis ütleb Flutter *engine'ile*, et *widgeti state* on muutunud ning oleks aeg ta uuesti välja joonistada. Joonis 15 esitab *build* meetodit ja funktsiooni `setState` (autentimise vaade).

```
class _AuthPageState extends State<AuthPage> {
  ...
  final _formKey = GlobalKey<FormState>();
  var _isLoading = false;
  ...

  Future<void> _submit() async {
    if (_formKey.currentState.validate()) {
      _formKey.currentState.save();
      setState(() => _isLoading = true);
      ...
    }
  }

  ...

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      ...
      appBar: AppBar(
        ...
      ),
      body: SingleChildScrollView(
        child: ...
```

Joonis 15. *Build* meetod ja funktsioon `setState` (autentimise vaade).

Widgeti uuesti ekraanile renderdamisel kutsutakse välja ka kõikide tema järglaste *build* meetod. Kuigi Flutter *engine* on selliselt optimeeritud, et *widgetite* uuesti renderdamine ei ole nii kulukas protsess ning on võimalikult efektiivne, siis vastava protsessi parendamiseks on siiski ka arendaja poolseid võimalusi. Üheks selliseks võimaluseks, mida ka igal pool kasutati, on defineerida, kui võimalik, *widget* kui ka tema konstruktor konstantsena (*const*). See ütleb Flutter *engine'ile*, et vastav väärtus on teada juba koodi kompileerimise ajal ning ei muutu pärast seda mitte kunagi. Samuti annab see Flutter *engine'ile* teada, et konkreetset *widgetit* pole vaja *build* meetodi või `setState` meetodi väljakutsel uuesti ekraanile joonistada, kuna see pole vahepeal muutunud ning pole mõtet teha tühja tööd. Selle asemel, et ehitada *widget* uuesti, taaskasutatakse vana.

Võetud lähenemine vähendab rakenduse protsessideks kuluvat aega ning parendab kasutajakogemust. Väiksemate rakenduste korral ei pruugi see kasutajakogemusele suurt mõju avaldada, kuna erinevused rakenduse töö kiiruses on niivõrd väikesed, et jäävad silmale märkamata. Sellegipoolest on võtet rakendatud.

Selliste kohtade leidmiseks koodis, kus leidub võimalus vähendada UI komponentide uuesti välja joonistamist, on `analysis_options.yaml`[3] failis defineeritud reeglid, mis toovad vastavad kohad koodis alla joonituna esile. Reeglid vähendavad tõenäosust, et sellised kohad jäävad arendajale kahe silma vahele. Konkreetsed reeglid on `prefer_const_constructors: true` ja `prefer_const_declarations: true`.

3.3.2 Kasutajaliidese optimistlik uuendamine

Rakenduses on võetud kasutajaliidese uuendamiseks optimistlik lähenemine. See tähendab, et UId uuendatakse ootamata serveripoolset vastust eeldusega, et see läheb läbi ilma vigadeta. Kasutaja näeb muudatusi juba enne päringu teele saatmist.

Optimistliku kasutajaliidese uuendamise korral tuleb aga arvesse võtta, et muudatused serveris võivad ükskõik mis põhjusel ebaõnnestuda. Vastava olukorra tekkimisel tuleb rakenduse *state* päringueelsesse aega tagasi kerida, kuna muidu tekiksid lahkhelid rakenduse kohaliku *state'i* ning serveri *state*'*i* vahel.

Kasutajaliidese optimistliku uuendamise realiseerimiseks tuleb enne serverisse päringu alustamist salvestada *provideri* tolle hetke seis mällu. Seejärel viiakse muudatus

lokaalselt sisse ning jäädakse ootama serveri vastust. Õnnestunud päringu korral ei muudeta enam midagi, kuid serverist veateate tagasi saamisel kirjutatakse *provideri* väljad mälus hoitavate päringueelsete väärtustega üle. Samuti antakse *provideri consume'ivatele widgetitele* teada, et nad võtaksid samamoodi muudatuse tagasi. Joonis 16 esitab optimistlikku kasutajaliidese uuendamist (helme lükkimine keesse).

```
Future <void> addBead(Bead bead) async {
  final index = _beads.length;
  _beads.add(bead);
  notifyListeners();

  try {
    await patchBeads();
  } catch (error) {
    _beads.removeAt(index);
    notifyListeners();
  }
}
```

Joonis 16. Optimistlik kasutajaliidese uuendamine (helme lükkimine keesse).

3.3.3 Kasutaja automaatne sisse ja välja logimine

Kasutajakogemuse parendamiseks on implementeeritud võimaluse korral kasutaja süsteemi poolt automaatne sisselogimine ning talle väljastatud JWT aegumisel automaatne väljalogimine.

Kasutaja automaatseks sisselogimiseks peab tal olema rakenduses konto, ta peab olema lähiajal sellega eelnevalt sisse loginud ning kasutajale Firebase Auth API poolt väljastatud JWT peab olema kehtiv. Kui eelnevad tingimused on täidetud, logib süsteem kasutaja ise sisse ning suunab ta tema vaprusekee vaatesse.

Kasutaja autentimiseks vajaminevaid andmeid on seega vaja säilitada ka pärast rakenduse sulgemist. Selle jaoks on kasutusele võetud Flutter SDK plugin `shared_preferences`, mis võimaldab salvestada kasutaja seadmes olevasse süsteemifaili võti-väärtus paare. Õnnestunud kasutaja sisse logimise järel salvestatakse sellesse faili kasutaja ID, JWT ja JWT aegumiskuupäev.

Rakenduse käivitamisel küsitakse sellest rakendusele eraldatud võti-väärtus paaride süsteemifailist just neidsamu väärtusi. Kontrollitakse, kas JWT on olemas ning kas see on veel kehtiv. Sel puhul logitakse kasutaja sisse ning kuvatakse talle vaprusekee vaadet, vastasel juhul koduvaadet. Joonis 17 esitab kasutaja automaatset sisse logimist.

```
Future<bool> tryAutoLogin() async {
  final prefs = await SharedPreferences.getInstance();
  if (prefs.containsKey('userData')) {
    final userData =
        json.decode(prefs.getString('userData')) as Map<String, dynamic>;
    final expiryDate = DateTime.parse(userData['expiryDate']);
    if (expiryDate.isBefore(DateTime.now())) {
      return false;
    }
    _userId = userData['userId'];
    _token = userData['token'];
    _expiryDate = expiryDate;
    notifyListeners();
    _autoLogout();
    return true;
  }
  return false;
}
```

Joonis 17. Kasutaja automaatne sisse logimine.

Selline lähenemine on võetud, sest kui kasutajale eraldatud JWT on veel kehtiv, pole mõtet lasta kasutajal ennast uuesti autentida üksnes serverist uue JWT küsimise eesmärgil, kui vana on veel kasutuskõlblik. See oleks kasutaja jaoks väga tüütu ning halb kasutajakogemus.

JWT aegudes tuleb kasutaja rakendusest välja logida. Ilma kasutaja süsteemi poolt automaatselt välja logimiseta juhtuks see järgmine kord, kui ta proovib navigeerida mõnda teise vaatesse. Juhtuks see nii seepärast, et igal navigeerimisel kontrollitakse `main.dart` failis kirjeldatud *home route* parameetrit. Seal kontrollitakse, kas kasutaja on autenditud. Joonis 18 esitab kontrolli, kas kasutaja on autenditud.

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider.value(
          value: Auth(),
        ),
        ...
      ],
      child: Consumer<Auth>(
        builder: (context, auth, _) {
          ifAuth(targetPage) => auth.isAuth ? targetPage : HomePage();
          return MaterialApp(
            ...
          ),
          home: auth.isAuth
            ? BraveryCordPage()
            : FutureBuilder(
              future: auth.tryAutoLogin(),
              builder: (_, authSnapshot) =>
                authSnapshot.connectionState == ConnectionState.waiting
                  ? SplashPage()
                  : HomePage(),
              ),
              ...
```

Joonis 18. Kontroll, kas kasutaja on autenditud.

Selleks, et kasutaja logitaks kohe JWT aegudes rakendusest välja, mitte alles järgmisel navigeerimisel, on realiseeritud kasutaja automaatne välja logimine. Süsteemi poolt kasutaja automaatse välja logimise käigus tehakse puhtaks ka võti-väärtus paaride süsteemifaili salvestatud kasutaja autentimise andmed. Joonis 19 esitab kasutaja automaatset välja logimist.

```
void _autoLogout() {
  final timeToExpiry = _expiryDate.difference(DateTime.now()).inSeconds;
  _autoLogoutTimer = Timer(
    Duration(
      seconds: timeToExpiry,
    ),
    logout,
  );
}
```

Joonis 19. Kasutaja automaatne välja logimine.

4 Valideerimine, arengusuunad

Selles peatükis kirjeldatakse rakenduse valideerimisel saadud tagasisidet, lasknud ühel organisatsiooni Vaprusehelmed töötajal rakendust proovida ning tuuakse välja edasised sammud rakenduse arenduseks.

4.1 Valideerimine

Rakendust valideeriti seda lastes ühel organisatsiooni Vaprusehelmed töötajal rakendust proovida. Rakendust proovi Android operatsioonisüsteemiga seadmes.

Tagasiside oli valdavalt positiivne. Ei tulnud ootamatusena, et poolikut ning disainist erinevat kasutajaliidest tuleb muuta, kuid üldise rakenduse väljanägemisega oldi suures pildis rahul.

Ühe kindla muudatusena, mida tuleb teha, toodi välja helme eemaldamine keest. Palunud organisatsiooni Vaprusehelmed töötajal mõni helmes keest eemaldada, oli tal sellega raskusi. Välja mõeldi, kuidas seda teha, kuid kui täiskasvanul oli juba sellega raskusi, siis lapsele see kindlasti ei kõlba.

Sooviti, et kui keesse lükitakse uus helmes, siis keritakse vaprusekee vaates kee lõppu keesse lükitud helmeni, et kasutajal oleks võimalik näha, kas helmes tõepoolest tema keesse ka jõudis. Praegu näidatakse kasutajale helme lisamise ülekattest vaprusekee vaatesse tagasi navigeerimisel alati keed alates kee algust tähistavast helmest.

Praegune lähenemine, et kasutaja autentimiseks kasutati kasutajanime asemel meiliaadressi, kliendile vastumeelt ei tekitanud, kuid see läheb edasi arutamisele.

Veel toodi välja, et esemega võrdlemise vaates antakse hetkel kasutajale liiga vähe tagasisidet. Kasutajale öeldakse, kas tema kee on vaadeldavast esemest lühem, umbes sama pikk või pikem, kuid peab olema võimalik teada saada, kui suur nende pikkuste erinevus on.

Rakenduse nõutav funktsionaalsus oli olemas ning selle kohta märkusi ei tehtud. Öeldi üksnes, et funktsionaalsuse nõudeid võib juurde tulla.

4.2 Arengusuunad

Selles peatükis tuuakse välja edasised sammud rakenduse arenduses. Töö autoril tuli lisaks praegusele toetatavale funktsionaalsusele juurde ideid ning kliendi nõutud muudatused tuleb samuti realiseerida.

4.2.1 Lisafunktsionaalsus

Puudub kasutaja registreerimise järel kasutaja meilile kinnituse saatmine kontrollimaks, kas sisestatud meiliaadress ka reaalselt eksisteerib. Selle jaoks tuleb kirjutada valmis funktsionaalsus meilide saatmiseks. Samuti tahetakse lisada võimalus meiliaadressi vahetada.

Puudub võimalus vahetada parooli. Parooli vahetamiseks tuleb saata kasutaja meilile kinnitus.

Kasutajal peab olema võimalus oma kontot kustutada. Sellega kaasneb ka kõikide temaga seonduvate andmete kustutamine.

Tuleb lisada logimine ning koguda logid serverisse ühte kohta kokku, saamaks vajaduse ilmnemisel kuskilt vaadata, mis rakenduse töös tekkinud vea esile kutsus ning lihtsustamaks nende tekkimisel vea kolde kergemat avastamist ning parandamist.

Lisada võimalus kasutada kasutaja profiilipildina kasutaja seadme failisüsteemis olevat pilti. Hoiustada kasutaja profiilipilti serveris, et see samamoodi nagu kasutaja andmedki teise seadme kasutusele võttes alles oleks.

Vahetada PNG (*Portable Network Graphics*) pildifailid SVG (*Scalable Vector Graphics*) vastu välja, et piltide suurust muutes ei kaotataks kvaliteeti.

Tuleb implementeerida JWT kehtivuse pikendamine. Praegu logitakse kasutaja välja pärast esimese Firebase Auth API väljastatud JWT aegumist.

4.2.2 Disain iOS seadmetel

Flutter SDK võimaldab teha vahet, mis operatsioonisüsteemiga seadet kasutaja omab. Vastavalt sellele on võimalik kuvada kas siis Androidi korral Material Design *widgeteid* või iOSi korral Cupertino *widgeteid*. Rakendus töötab mõlemiga mõlemal operatsioonisüsteemil. Ainus põhjus, miks kasutada ühe operatsioonisüsteemi puhul

ühtesid ja teise puhul teisi *widgeteid*, on kasutaja harjumused. Kasutajad on harjunud nägema operatsioonisüsteemile omase väljanägemisega komponente. Material Design *widgetid* on väljanägemise ning animatsioonide poolest kodusemad Androidi kasutajatele, Cupertino *widgetid* aga iOSi kasutajatele.

Vaatamata sellele, et mõlemad tüüpi *widgetid* töötavad ka teise operatsioonisüsteemi peal, peab arvesse võtma asjaolu, et igale Material Design *widgetile* pole analoogset Cupertino *widgetit*. Võiks vältida olukordi, kus kasutajale kuvatakse *widgeteid*, mida nad pole oma operatsioonisüsteemiga seadme peal kunagi näinud. Seega on plaanis asendada iOS operatsioonisüsteemil töötavatel seadmetel Material Design *widgetid*, kus võimalik, neile vastavate Cupertino *widgetitega*.

4.2.3 Rakenduse kasutamine võrguühenduseta

Praeguse rakenduse ülesehituse juures ei ole rakendus ilma võrguühenduseta kasutuskõlblik. Kõik päringud, nii autentimine kui ka andmete pärimine sõltuvad võrguühenduse olemasolust. On kavas teha võimalus kasutada rakendust ilma interneti olemasoluta või võrguühenduspiirkonnast väljaspool. Selle jaoks on tarvis võtta kasutusele lokaalne andmebaasisüsteem. Seal saab hoida kogu rakenduses vajaminevaid andmeid, mis garanteerib rakenduse töövõime ka siis, kui puudub ligipääsu võrguühendusele.

Lokaalse andmebaasi kasutamine säästab ka kasutaja andmemahtu. Samuti kiirendab lokaalse andmebaasi tutvustamine rakenduse tööd, kuna päringud selle pihta võtavad vähem aega kui serveri vastu.

4.2.4 Animatsioonid

Praeguse seisuga on enamus UI komponentidest olemas, kuid animatsioonidele pole suuremat tähelepanu pööratud. Plaanis on lisada animatsioonid taustapiltidele, mis liiguvad ekraanil kui kasutaja navigeerib vaadete vahel. Menüüs navigeerides võiks üleminekute sujuvamaks muutmiseks implementeerida näiteks menüünuppude sisse ja välja *fade'imise*.

Nagu vaadete peatükis välja toodud piltidelt näha, on kasutajaliidese kallal vaja teha samuti lisatööd. Sealhulgas tuleb parandada vaadete *responsivenessi*.

5 Kokkuvõte

Lõputöö eesmärk on luua heategevusorganisatsioonile Vaprusehelmed, kelle eesmärk on krooniliste või eluohtlike haigustega laste psühhosotsiaalne toetamine, eestikeelne mobiilirakendus, mis võimaldab lastel saada oma raviteekonnast digitaalne ülevaade vaprusekee kujul.

Kliendi saadetud Adobe XDs[1] valminud rakenduse disaini põhjal tehti kindlaks rakenduse nõuded ning kirjutati valmis valdav osa rakenduse toetatavast funktsionaalsust ning kasutajaliidesest.

Kasutajal on võimalik oma vaprusehelmestest kokku lükitud keed vaadates teha selgeks, millised helmed tal on kogunenud ning millises järjekorras, sinna helmeid juurde lükkida ning soovi korral ka eemaldada. Kee pikkusest aimu saamiseks on võimalik seda haiglas leiduvate esemetega pikkuselt võrrelda.

Valmis rakenduse töötav prototüüp, mis vajab edasist tööd. Valideeriti olemasolevat lahendust, lastes ühel ettevõtte Vaprusehelmed töötajal rakendust kasutada ning tagasisidet anda. Tagasiside osutus enamjaolt positiivseks. Realiseerituga oldi rahul ning mõisteti, et rakendus pole valmis ning on alles arendusjärgus.

Kasutatud kirjandus

[1] „Vaprusehelmed,“ MTÜ Vaprusehelmed, 28 08 2018. [Võrgumaterjal]. Available: https://www.facebook.com/vaprusehelmed/. [Kasutatud 17 04 2020].

[2] A. Lastovetska, „Blog: Native App Development vs. Hybrid and Web App Building,“ MLSDev, 01 11 2018. [Võrgumaterjal]. Available: https://mlsdev.com/blog/native-app-development-vs-web-and-hybrid-app-development. [Kasutatud 17 04 2020].

[3] Sergey Korolev, „Blog: Development: Mobile App Development Approaches Explained,“ Railsware, 19 06 2019. [Võrgumaterjal]. Available: https://railsware.com/blog/native-vs-hybrid-vs-cross-platform/. [Kasutatud 17 04 2020].

[4] Dream Squad, „Blog: A brief Introduction to Flutter,“ 11 Aprill 2019. [Võrgumaterjal]. Available: https://www.dreamsquadgroup.com/whats-flutter-brief-introduction. [Kasutatud 27 Aprill 2020].

[5] Flutter, „Docs: Resources: Technical overview,“ [Võrgumaterjal]. Available: https://flutter.dev/docs/resources/technical-overview. [Kasutatud 27 Aprill 2020].

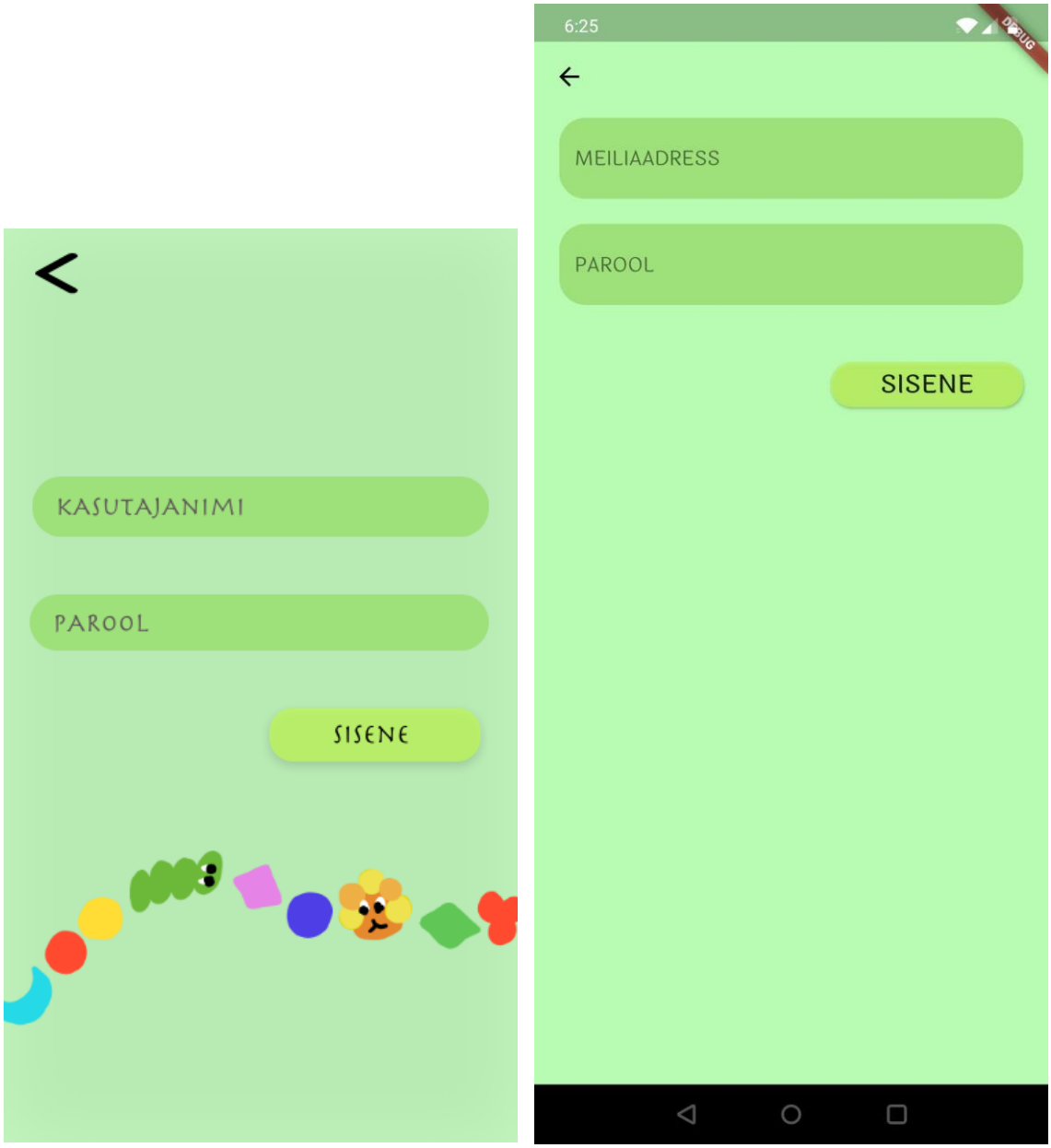
[6] R. S, „Hacker Noon: Tagged: Android: Introduction to Firebase,“ GeekyAnts, 27 Detsember 2017. [Võrgumaterjal]. Available: https://hackernoon.com/introduction-to-firebase-218a23186cd7. [Kasutatud 27 Aprill 2020].

[7] C. E, „Medium: Firebase-developers: What is Firebase?,“ 24 Oktoober 2016. [Võrgumaterjal]. Available: https://howtofirebase.com/what-is-firebase-fcb8614ba442. [Kasutatud 27 Aprill 2020].

[8] „Firebase: Docs: Guides,“ Google, 08 04 2020. [Võrgumaterjal]. Available: https://firebase.google.com/docs/guides. [Kasutatud 18 05 2020].

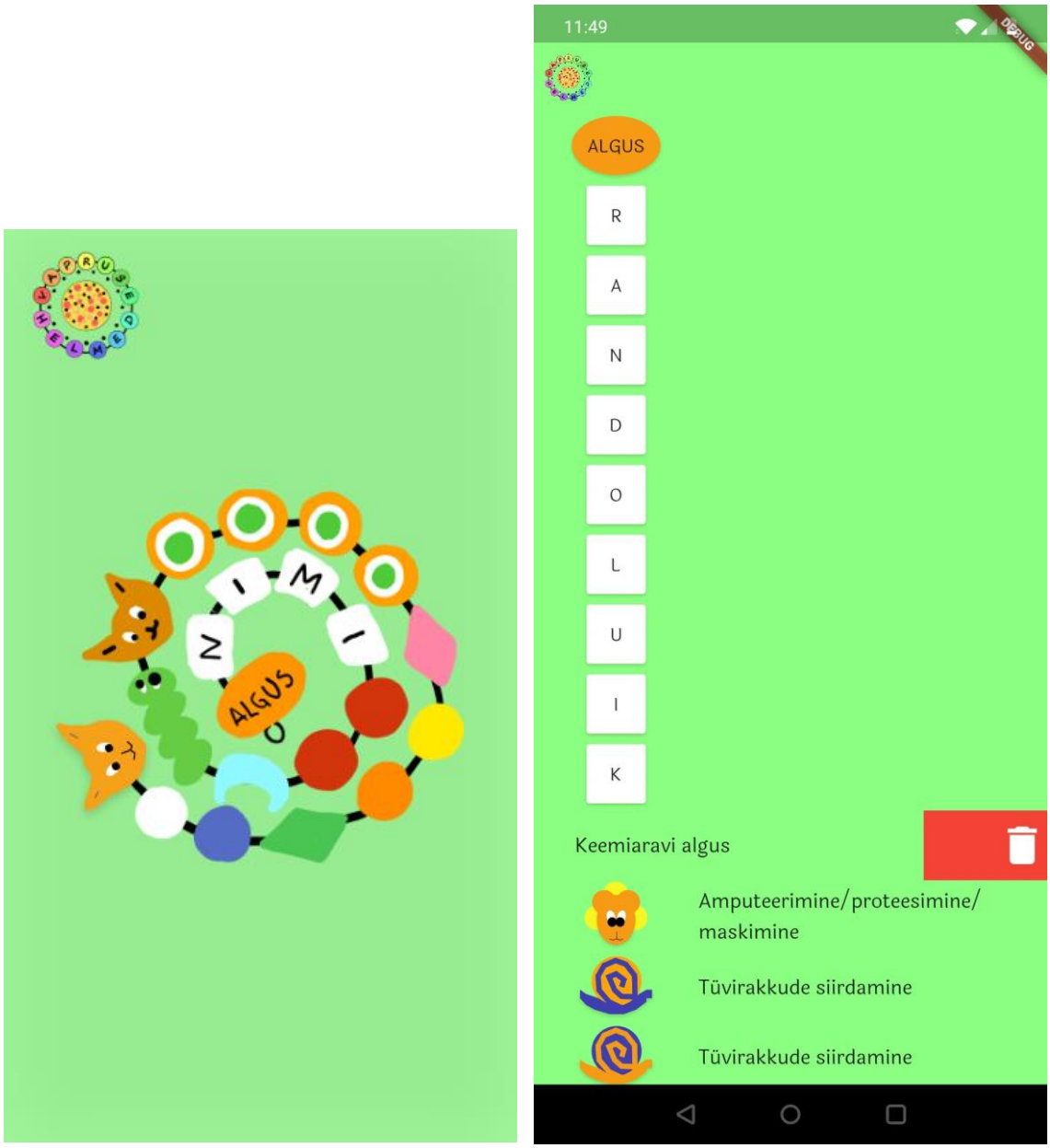
[9] Maximilian Schwarzmüller, „Udemy: Categories: Development: Mobile Apps: Dart Programming Language: Flutter & Dart - The Complete Guide [2020 Edition],“ Academind, Aprill 2020. [Võrgumaterjal]. Available: https://www.udemy.com/course/learn-flutter-dart-to-build-ios-android-apps/. [Kasutatud 30 Aprill 2020].

Lisa 1 – Autentimise vaate (sisse logimise vorm) lõpp-produkti disain vasakul pool ning praegune disain paremal pool



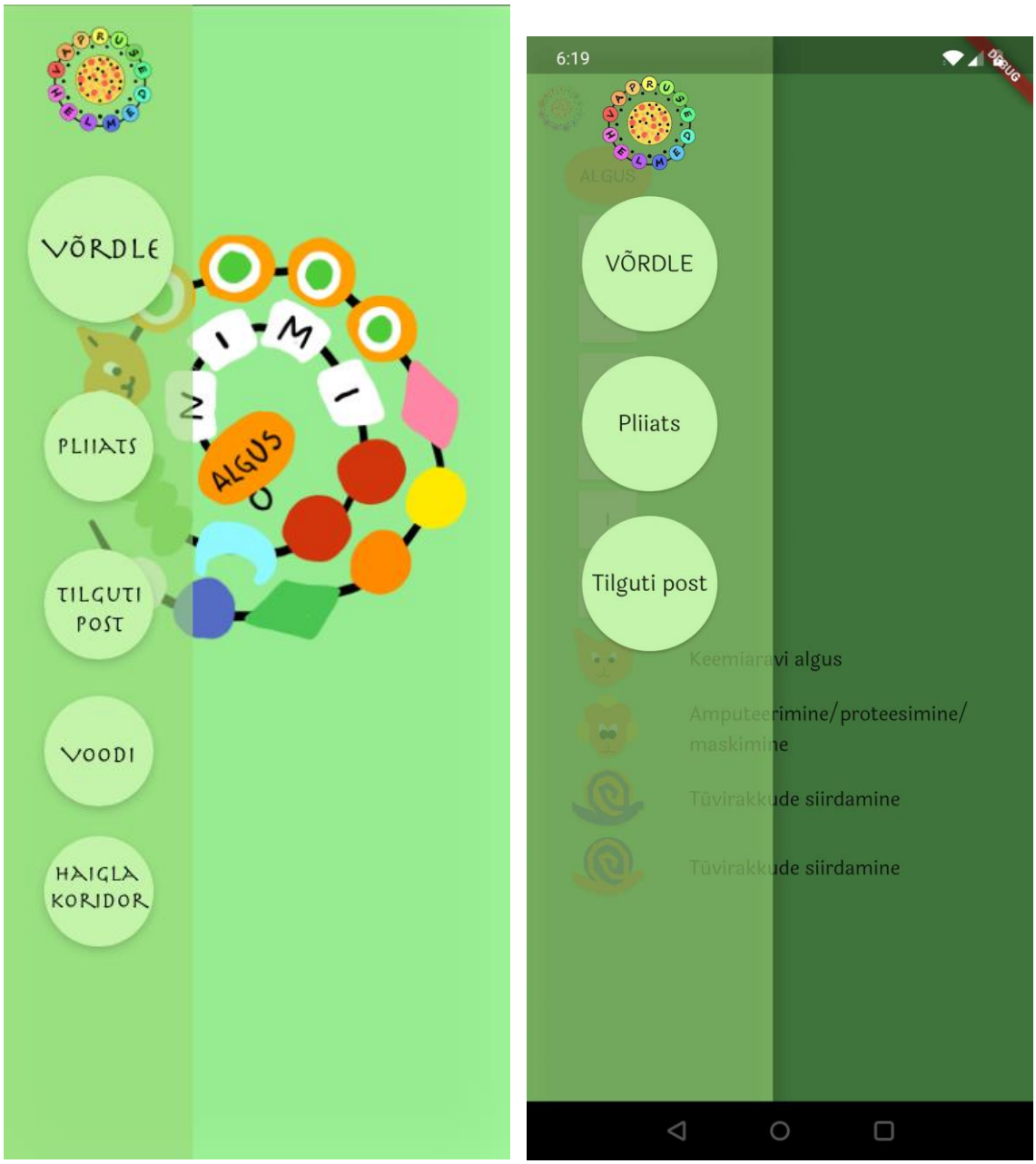
Joonis 20. Autentimise vaate (sisse logimise vorm) lõpp-produkti disain (vasakul pool) ning praegune disain (paremal pool).

Lisa 2 – Vaprusekee vaate lõpp-produkti disain vasakul pool ning praegune disain (helme eemaldamine) paremal pool



Joonis 21. Vaprusekee vaate lõpp-produkti disain (vasakul pool) ning praegune disain (helme eemaldamine) (paremal pool).

Lisa 3 – Vaprusekee vaate (esemega võrdlemise alammenüü) lõpp-produkti disain vasakul pool ning praegune disain paremal pool

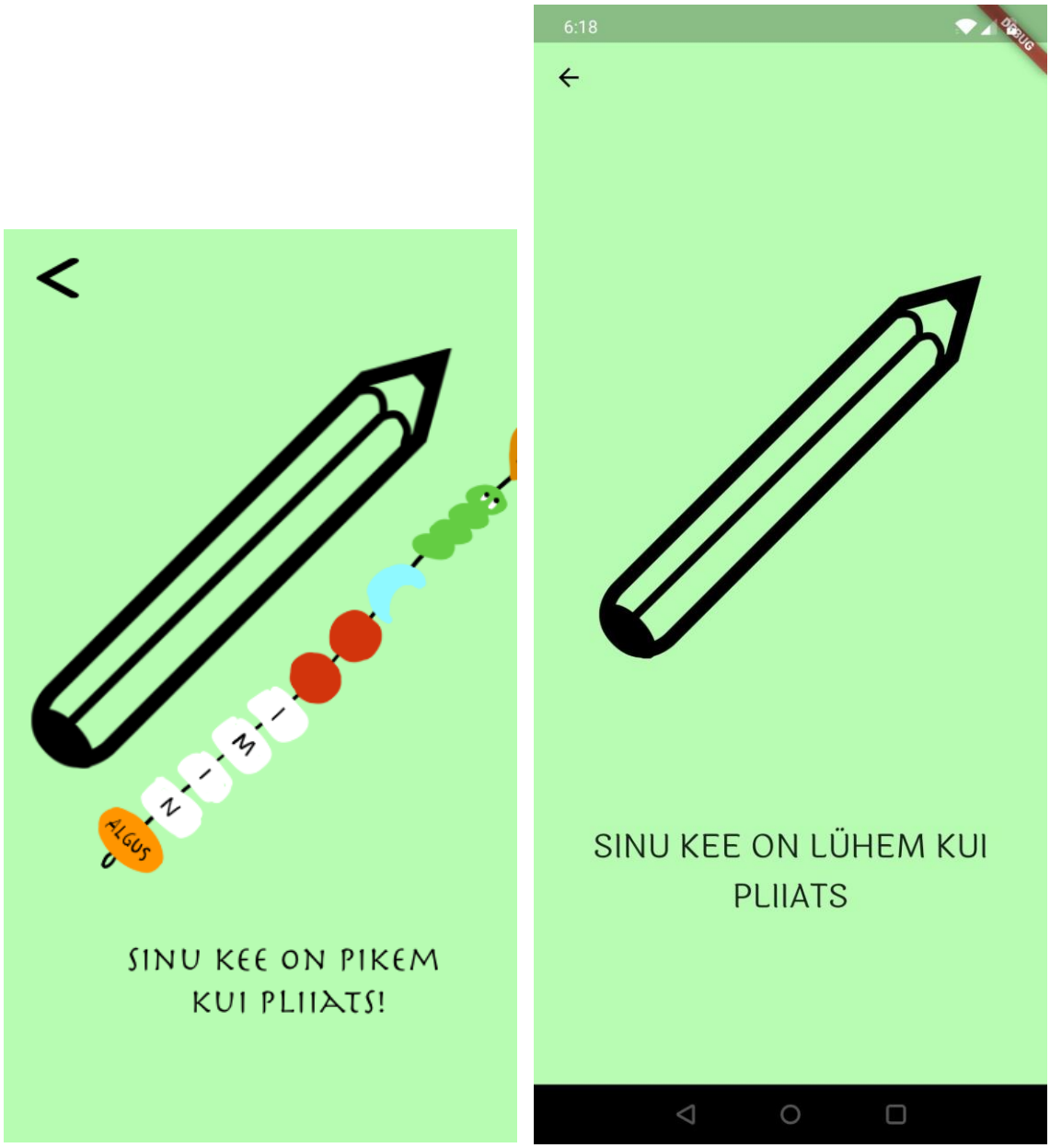


Joonis 22. Vaprusekee vaate (esemega võrdlemise alammenüü) lõpp-produkti disain (vasakul pool) ning praegune disain (paremal pool).

Lisa 4 – Helme lisamise vaate (helme ülekate) lõpp-produkti disain vasakul pool ning praegune disain paremal pool

Joonis 23. Helme lisamise vaate (helme ülekate) lõpp-produkti disain (vasakul pool) ning praegune disain (paremal pool).

Lisa 5 – Esemega võrdlemise vaate (pliiats) lõpp-produkti disain vasakul pool ning praegune disain paremal pool



Joonis 24. Esemega võrdlemise vaate (pliiats) lõpp-produkti disain (vasakul pool) ning praegune disain (paremal pool).