Tallinna Reaalkool

Mobiiliarenduse õppimine ja rakenduse loomine

Loovtöö

Hermann Käbi
140. lennu c-klassi õpilane
Juhendaja: Mari Käbi

Tallinn, 2021

Sisukord

Töö idee ja eesmärk

Sain idee selliseks loovtööks enda varasemast kogemusest programmeerimisega. Olen hobi korras ka varem programmeerinud, kuid täiesti teistsuguse keele ja tulemiga, nimelt Python 3-ga. Selle keelega saab väga kergesti kirjutada erinevaid algoritme, kuid mulle pakub rohkem huvi kirjutada midagi, mida saaksid kasutada paljud inimesed, näiteks äppe. Minu loovtöös kasutatav keel ja kasutajaliidese (ingl k UI või *user interface*) tarkvara töötavad Pythonist erineval põhimõttel. Erinevalt Pythonist, mis on mõeldud väljastama (*ingl k output*) teksti ja mida kasutatakse taustprogrammide kirjutamiseks (ingl k *backend*), on raamistik Flutter mõeldud just äppide ehk kõige selle, mida näeb kasutaja, loomiseks ja kujundamiseks (ingl k *front-end*). Eelkõige kasutatakse tausta kirjutamiseks keelt Dart, kuid natuke raamistiku koodi konfigureerides on võimalik kasutada ka teisi keeli. Raamistikust Flutter ja programmeerimiskeelest Dart räägin hiljem põhjalikumalt.

Teen loovtööd üksi, osaliselt sellepärast, et ei tea kedagi, kellel oleks sarnane huvi ja taustaoskused just äppide tegemiseks. Kuigi ma pole seda raamistikku ja keelt kasutanud, tunnen siiski OOP (*Object-oriented programming*, eesti keeles Objektorienteeritud programmeerimine) põhimõtteid, andmetüüpe, keeleloogikat ja paljusid teisi olulisi programmeerimise põhitalasid, mille õppimine nullist võtaks palju aega.

Minu eesmärk loovtöö lõpuks oleks õppida piisavalt palju, et kirjutada täisfunktsionaalne äpp. See eesmärk, kuigi võib tunduda aru saadav, vajab siiski selgitust. Programmeerimine pole kunagi olnud ega saa kunagi olema peast, ilma abivahenditeta kirjutamine. Keeled arenevad pidevalt, nad pole ideaalsed. Iga kasutatav programm, olgu ta kui kerge tahes, vajab silumist (ingl k *debugging*). Arenduses esineb pidevalt probleeme ja vigu (*error*), mida lahendada, ja pole mõeldav, et seda peaks peast tegema. Niisiis on loovtöö käigus loomilikult lubatud vigade parandamiseks kasutada internetti.

Täpsemad kriteeriumid, mis ma äpile seadsin, on olemas osas 4 (töö tulemus)

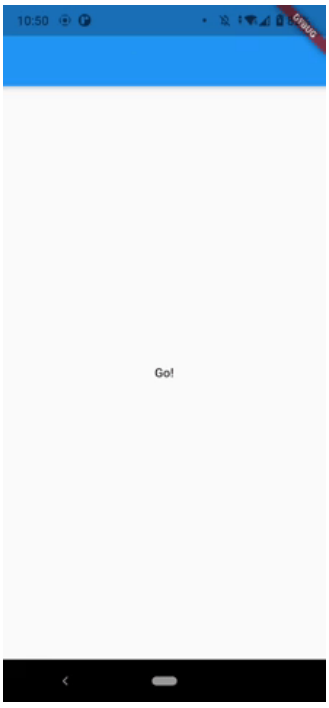
Progarammid ja ressursid, mida kasutan

Selles osas kirjeldan olulisemaid vahendeid, mida loovtööks kasutasin, olgu nendeks programmid, veebilehed, tarkvarad, raamistikud jne.

Flutter

Flutter on Google'i loodud raamistik, mille peal otsustasin mobiiliarendust õppida. Tegin selle otsuse järgmiste faktorite põhjal:

a. Flutter on mõeldud looma platvormiüleseid natiiväppe (*Cross-platform native apps*). See tähendab, et Flutter äppe saab jooksutada mitmel erineval platvormil, põhiliselt Android ja iOS. See on platvormiülene osa mõistest. Natiiväpp (ingl k *Native app*) tähendab, et Flutter kompileeritakse Dart Engine abil platvormi keelde. Androidil on selleks keeleks Java ja Kotlin, iOSil Swift ja veebil JavaScript. See tähendab, et mis iganes toetatud platvormil äppi ka ei jooksutataks, jookseb see väga hästi (120FPS, kaadrit sekundis). Lisaks sellele käitub äpp platvormile omaselt, nagu näha sellest GIF-ist.



Joonis 1. Flutter käitumine Androidil

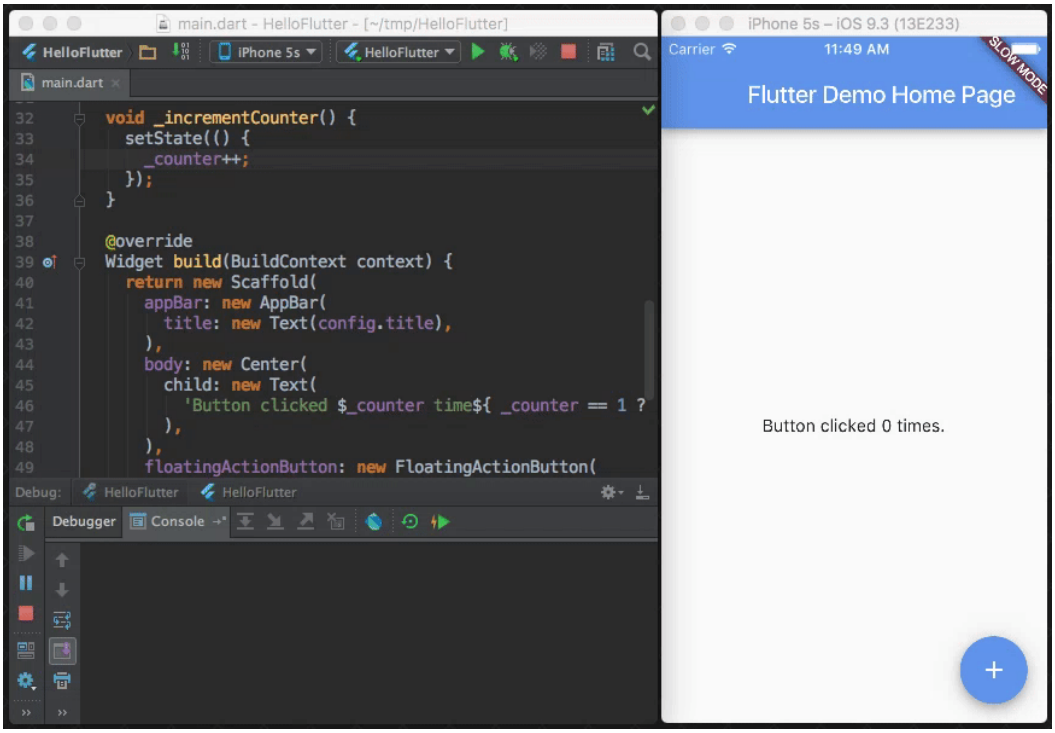
Joonis 2. Flutter käitumine iOSil

Kui minna teisele ekraanile, tuleb teine ekraan esimesel GIF-il (Androidil) alt üles, aga teisel (emuleeritud iOS) paremalt vasakule, ilma, et ma oleks pidanud midagi sellist kirjutama. Pange tähele ka tagsinupu erinevat väljanägemist teise ekraani üleval vasakul.

b. Flutter on moodne. Mobiilide algusajal, kui Java sai Androidi põhikeeleks (ingl k *Native language*), polnud telfoniäppides animatsioone, kuna telefonid polnud nii võimsad kui praegu. Seetõttu on Java keeles animatsioone teha raske ja kohmakas. Kuna Flutter on uus, on seal animatsioonidega arvestatud, nende tegemine on kerge, ja mõned animatsioonid on juba raamistikku sisse kirjutatud. Samuti näitab uudsust Hot Reload – võimalus äpi välimust ja koodi muuta infot kaotamata. Kuna seda võimalust on sõnadega raske edasi anda ilma oskuskeelt kasutamata, lisan siia GIF-faili, mis seda võimalust visuaalselt näitab.

Pange tähele numbrit (infot), mis suureneb, kui vajutatakse FAB-i (*Floating Action Button*, all paremal) ja jääb koodi muutmisel samaks.

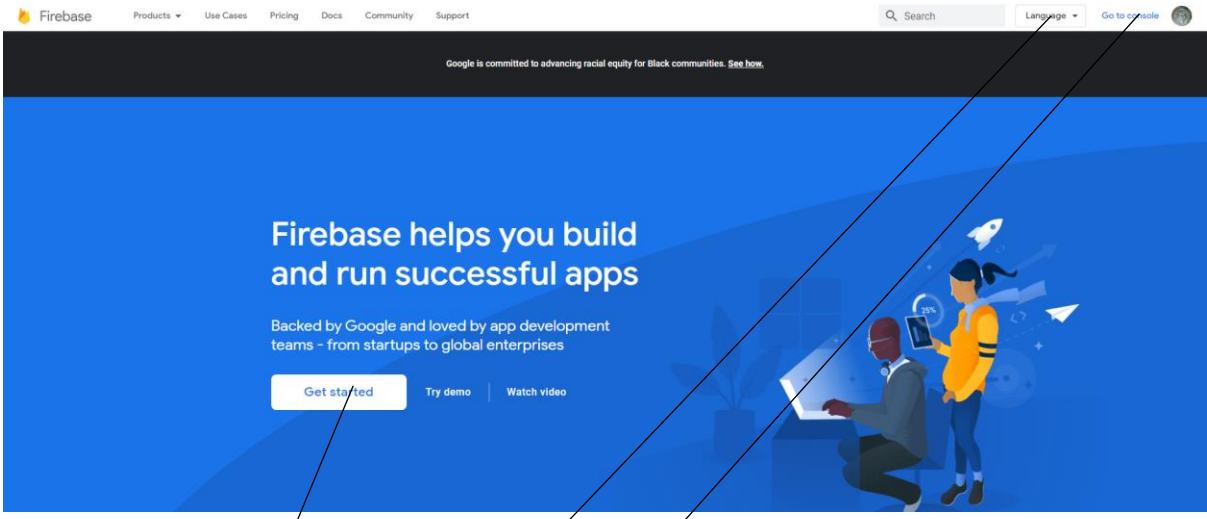
c.

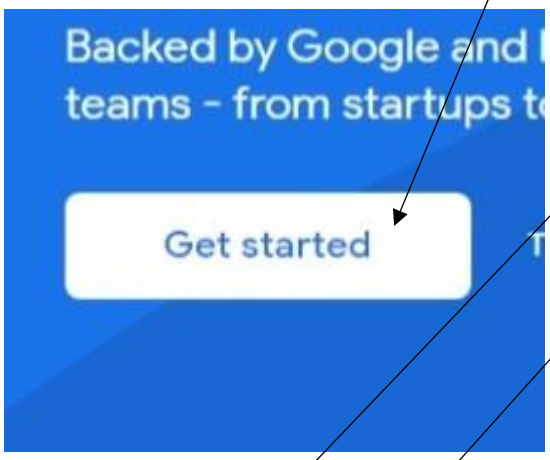


Joonis 3. Hot reload näide

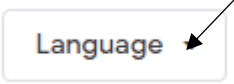
d. Flutter on avatud lähtekoodiga (ingl k *open-source*). Avatud lähtekood tähendab seda, et kogu raamistiku kood on avalikult ja tasuta kättesaadav. Peaaegu kõik avatud lähtekoodi projektid kasutavad Git-i ja GitHub-i. Ka Flutter on seal repositooriumina (uudissõnaga koodivaramu) olemas. Avatud lähtekoodil on mitmeid eeliseid. Esiteks, koodi saavad muuta kõik kasutajad. Kõige, ka sisseehitatud elementide (Flutter kutsub neid *Widgets*, vidinad), välimust ja käitumist saab muuta. Teiseks, kuna koodi saab üle vaadata, ei pea kasutajad muretsema, et omanikud on koodi pahavara lisanud, nii et see lisab turvalisust. Kolmandaks, kuna kõik saavad koodi muuta ja ideid anda, on projekt kõrgema kvaliteediga.

e. Flutter kasutab Material Design aluseid. Material Design on Google'i arendatud äpidisaini kogumik, mis sisaldab juhiseid, kuidas mingeid elemente (nt nupud, värvid, tekst, tume teema, FAB (*Floating Action Button*), ikoonid jne) kasutada, samuti on nad need elemendid kujundanud. Paljud Google teenused kasutavad Material Design'i elemente, näiteks Google Maps, Google Classroom, Google Photos jpt. Kõige äratuntavamad on näiteks nupud. Näiteks toon Google Firebase kodulehe:





See nupp on Contained Button. See väljendab suurt olulisust.



See nupp on Outlined Button. See väljendab keskmist olulisust.

See nupp on Text Button. See väljendab vähest olulisust.

Dart

Dart on Google'i loodud programmeerimiskeel, mis on eeskätt mõeldud kasutamiseks kasutajaliidestes (front-end), seega näiteks veebilehtedes ja äppides. Raamistiku Flutter kasutatakse peaaegu alati Dartiga. Dart on, sarnaselt Flutteriga, üsna moodne programmeerimiskeel. Just Dart abiga töötab Hot Reload (vt 1.b). Samuti on Dart keelt lihtne õppida. Süntaks sarnaneb väga nt JavaScriptile ja on väga lühike võrreldes teiste populaarsete

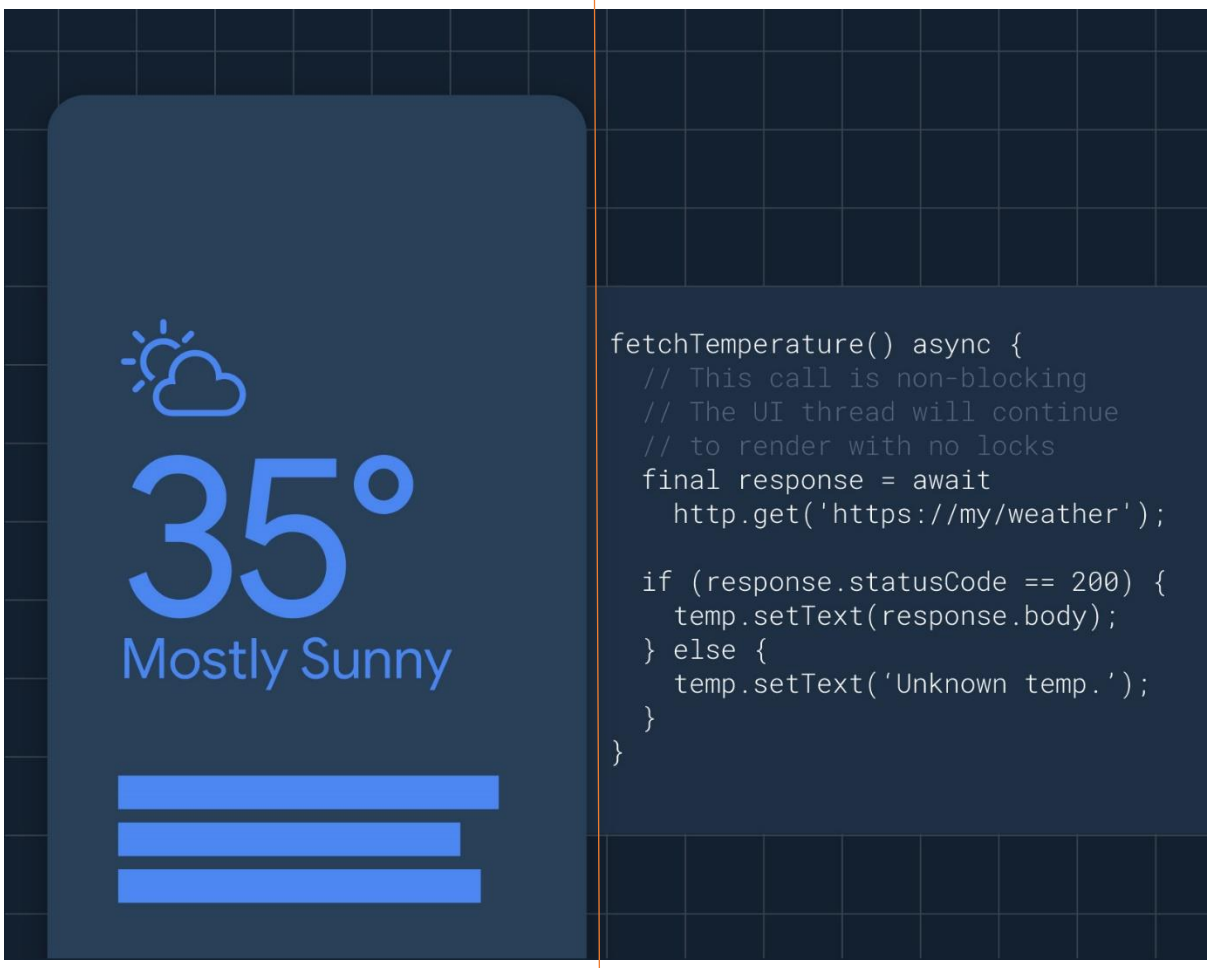
keeltega nagu Java. Kuna Dart on uus, saab see ära kasutada kogu arvuti võimsuse võimalikult efektiivselt.

Dart oli vahemikus 2018-2019 GitHubi andmetel kõige kiiremini kasvav programmeerimiskeel maailmas (532%) (allikas).

Kahe keele vahe

Oletame, et tahan kirjutada ilmaäppi. Ilma-andmete saamiseks kasutan API-t (API – Application Programming Interface on selles kontekstis lihtsustatult internetist saadav dokument, kus on kirjas infot ilma kohta kindlas asukohas, mille on arendaja valinud). Sellega, kuidas ma ilma-andmeid „tellin“ (request), tegeleb Dart'i kood, aga nende andmete ekraanile näitamisega (kujundus) tegeleb Flutter kood. Toon välja ka pildid koodidest ja nende erinevusest.

Teoreetiline vahe neil kahel keelel on selline.



Joonis 4. Veebilehelt dart.dev

Vasakul on Flutter kood, mis mõjutab kõike, mida kasutaja näeb. Paremal on Dart kood, mis tegeleb temperatuuri saamisega internetist (kasutades API-t).

Allolevatel piltidel esimesena on toodud valik Flutter (kasutajaliides) koodist, teisena Dart (andmed, back-end).

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Color(0xFF2A4058),
    body: SafeArea(
      child: Padding(
        padding: const EdgeInsets.symmetric(horizontal: 25, vertical: 15),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Row(...),  // Row
            SizedBox(height: 80,),
            Image.network(currentImage, scale: 0.5,),
            Text("$currentTemperature$unitPrefix", style: TextStyle(fontWei
            Text(currentDescription, style: TextStyle(fontWeight: FontWeigh



          ],
        ),  // Column
      ),  // Padding
    ),  // SafeArea
  );  // Scaffold
```

Joonis 5. Flutter kood (ilmaäpp)

```dart
Future<WeatherData> fetchWeatherDataForLocation(String location, [TemperatureUnit unit = TemperatureUnit.celsius])async{

  //Makes the request to OpenWeatherMap API
  http.Response response = await http.get(
      Uri.parse("https://api.openweathermap.org/data/2.5/weather?q=$location&appid=$apiKey&units=metric"),
  );

  //Checks statusCode, continues if everything went well
  if(response.statusCode == 200){
    Map data = jsonDecode(response.body);
    String cityName = data["name"];
    int temperature = double.parse(data["main"]["temp"].toString()).toInt();
    String description = data["weather"][0]["description"];
    String icon = "https://openweathermap.org/img/w/${data["weather"][0]["icon"]}.png";
    switch(unit){
      case TemperatureUnit.celsius:
        temperature *= 1;
        break;
      case TemperatureUnit.fahrenheit:
        temperature = ((temperature * 9/5) + 32).toInt();
        break;
      case TemperatureUnit.kelvin:
        temperature += 273;
        break;
      default:
        temperature *= 1;
        break;
    }
```

Joonis 6. Dart kood (ilmaäpp)

Kuigi esmapilgul võib tunduda, et tegu on sarnaste süsteemidega, on siiski näha palju erinevusi. Flutter (esimene) kood on silmapaistvalt modulaarne ja tomib laste/vanemate põhimõttel (kõige üleval on Scaffold, mis on allolevate Widgetite (kollaselt kirjutatud) vanem, st teised kuuluvad tema hulka). Flutter kood on suhteliselt lühike, selles lihtsas äpis vaid 50 rida.

Dart koodis on näha, et sellist hierarhiat pole. Seda koodi loeb arvuti ülalt-alla, tehes enne ära ülemised käsklused. Dart kood on pikem, sest sellega nii kirjutatakse back-end koodi (mis saab ilma-andmed), kui ka „ühendatakse“ need andmed kasutajaliidesesse. Selles äpis on Dart kood umbes 120 rida ehk 2-3 korda pikem.

See harjutusprojekt on nähtav siit.

Android Studio

Android Stuido on ametlik Androidile arendamise, IntelliJ IDEAd kasutav IDE (ingl k *Integrated Development Environment* ehk programm, kus saab programmeerida). Android Studio on Flutter projektide jaoks hea valik, sest sinna on mõned otseteed lisatud. Samuti on sinna lisatud populaarsemate terminalikäskude jaoks graafiline kasutajaliides. Lisaks sellele on Flutter'i tegijad Android Studiole kirjutanud Dart ja Flutter pluginid, mis võimaldavad näiteks IntelliJ süsteemi kasutada. Lisaks on Android Studiosse sisseehitatud paljud väga mugavad võimalused, näiteks versioonihaldus.

Git (GitHub)

Git ja GitHub'i võib vaadelda kui ühte tervikut. Nad on maailma suurim versioonihaldussüsteem. Mida see tähendab? Kui arendada tarkvara, näiteks mobiiliäppi, on Git võimalus oma lähtekoodi internetti üles laadida. See tähendab, et kui kellelgi tekib kahtlusi äpi üle (näiteks kahtlustab, et äpp jälgib teda), saab ta internetist failid üles otsida, ja (eeldades, et ta oskab seda programmeerimiskeelt/raamistikku lugeda) saab näha, kas kahtlused vastavad tõele. Tegelikult võimaldab see veel väga palju muid võimalusi, näiteks arendajad üle maailma saavad samal ajal ühe projekti kallal töötada. Git'i ja GitHub'i eesmärk on muuta maailma tarkvara läbipaistvamaks ehk suurendada avatud lähtekoodiga projekte, mistõttu pole ime, et süsteemi lõi avatud lähtekoodi suurteose Linux looja Linus Torvalds.

Ajakava

Kasutan Flutter/Darti õppimiseks kohati ka veebikursust. Tuginen ajakava koostamisel osaliselt sellele. Pärast kursuse lõppemist planeerin kavva harjutamisprojekte ja uute, keerukamate teadmiste omandamist. Tänu veebikursuse kasutamisele saan eeldatava ajakava suhteliselt hea täpsusega kirja panna. Järgnevalt toongi selle välja:

2020

4 – 10. mai – sissejuhatus, vajaminevate SDK-de allalaadimine ja ülesseadmine

11 – 17. mai – teooria (äpiarenduse põhialused, mõisted, Flutter spetsiifilised mõisted ja põhimõtted) õppimine

18 – 24. mai – teooria (Flutter äpi „koostis“, ülesehitus)

25 – 31. mai – kõige esmasema äpi loomine (1), põhitalade kinnistamine ja praktiline kasutamine

1 – 7. juuni – kõige esmasema äpi loomine (2)

8 – 14. juuni – äppide reaalsetele seadmetele jooksma saamine (ingl k *deploy*)

15 – 21. juuni – Flutter kasutajaliidese põhitõed (*UI layout*) (1)

22 – 28. juuni – Flutter kautajaliidese põhitõed (2) (Container vidin)

...

3 – 9. august – Eelnevate teadmiste kordamine

10 – 16. august – Flutter kasutajaliidese õppimine (paigutus vidinate Column ja Row abil)

17 – 23. august – Flutter vidinate omaduste (*properties*) muutmine

24 – 30. august – äppi piltide, erinevate fontide ja ikoonide lisamine, Google Fonts kasutamine

...

14 – 20. september – „paindlike“ ekraanipaigutuse (*Flexible layout*) põhitõed kasutades Expanded vidinat

21 – 27. september – nuppude lisamine Flutter äppi, funktsioonide käivitamine nupu vajutamisel

28. september – 11. oktoober – olekuga ja olekuta vidinad (*Stateful* ja *Stateless Widgets*)

12 – 18. oktoober – pooljuhuslike* arvude kasutamine Flutter äpis

* Kuna kõik juhuarvualgoritmid põhinevad matemaatilistel arvutustel, ei saa neid lugeda täisjuhuslikeks.

19 – 25. oktoober – Flutter ja Dart „koodipakkide“ (*packages*) kasutamine ja projekti importimine (YAML faili õppimine)

26. oktoober – 1. november – Audiofailide mängimine kasutades audioplayers „koodipakki“

2 – 8. november – koodi restruktureerimine (*refactoring*)

9 – 15. november – dünaamiliselt erinevate andmete näitamine kasutajale

16 – 29. november – ise klasside ja vidinate loomine

30. november – 13. detsember – OOP (objektorienteeritud programmeerimine, *object-oriented programming* on programmeerimiskeelte „stiil“, kus andmeid säilitatakse klassides, mis koosnevad meetoditest (funktsioonidest) ja andmeväljadest)

14 – 20. detsember – *enum* andmetüübi õppimine keeles Dart

...

2021

28. detsember – 3. jaanuar – Ternaarsed tehted keeles Dart (süntaks, kasutamine jne)

4 – 10. jaanuar – Flutter teemade (*themes*) kasutamine äpi välimuse ühtsustamiseks

11 – 17. jaanuar – Navigator vidina kasutamine erinevatele ekraanidele liikumiseks

18 – 31. jaanuar – telefoni asukohaandmete saamine kasutades *geolocator* „koodipakki“, tehted koordinaatidega (Haversine’ valem), asukohaandmete lubade lisamine äppi

1 – 7. veebruar – Asünkroonne programmeerimine Dart-is (süntaks, põhimõte)

* Asünkroonne programmeerimine tähendab, et programm ei jookse ülalt-alla, vaid asünkroonselt (märksõnaga *await*) märgitud ridade järgi oodatakse, enne kui järgmistele ridadele minna. Sellist võimalust kasutatakse näiteks internetist infot paludes (*request*), sest internetist ei tule need andmed koheselt.

8 – 14. veebruar – *Stateful* vidina elutsükkel

15 – 21. veebruar – API (aplikatsiooni programmeerimisliides, *application programming interface*) kutsed (*request*) keeles Dart kasutades *http* kogumit.

22 – 28. veebruar – kuidas uuendada ekraani andmete uuendamisel

1 – 7. märts – keerulisemate vidinate õppimine

8. märts – 11. aprill - **Google Firebase**

8 – 14. märts – sissejuhatus Google Firebase’i (pilvetehnoloogial andmebaas, autentimine jne)

15 – 21. märts – NoSQL andmebaasi ehitus *

Firebase’i pilveandmebaas Firestore on NoSQL põhimõttel (st andmeid ei talletata suhtelistes tabelites)

22 – 28. märts – Firestore õppimine (pilve salvestamine, andmete lugemine, muutmine, süntaks, integratsioon Flutter äppidega, turvalisusreeglid jne)

29. märts – 4. aprill – kasutajate autentimine Firebase Auth'iga (e-postiga, Google-ga, Apple-ga jne)

5 – 11. aprill – sissejuhatus Dart *stream* andmetüüpi *

* *Stream* andmetüüp sarnaneb tänapäeval populaarse striimimisega, aga on tehnilisem. *Stream* tüüpi uuendatakse ainult siis, kui andmed uuenevad, mistõttu saab seda võrrelda ojaga, millest nimi tuleb. Dart kui moodne keel on selle tüübi enesesse intregreerinud.

12 – 18. aprill – olekuhaldus Flutter äpis (*state management*)

19 – 25. aprill – *provider* koodikogumi õppimine

26. aprill – 23. mai – iseenda äpi tegemine (idee, programmeerimine, testimine, väljaandmine)

juuni 2021 – loovtöö kaitsmine

Planeeritud tegevused ja nende teostamine

Loovtööga pihta hakates läksid mul (algul) tegevused ajakavast palju kiiremini. Olin seda ette planeerinud, juhuks kui hilisemates faasides peaks aega väheks jääma. Teine põhjus selliseks otsuseks oli võimalus, et midagi lükkub mingil põhjusel edasi (õnneks koroonapandeemia minu loovtöö tegemist ei mõjutanud, vähemalt mitte otseselt) või ei tööta, millisel juhul oli vaja lisaaega. Eeldatavas ajakavas nägin ette, et vähem kui kuu äpi arenduseks on vähe aega. Allpool panin kirja enda umbkaudse tegeliku ajakava, kuhu olen tegelike kuupäevade taha kleepinud tegevused eeldatavast ajakavast:

2020

4 – 10. mai - sissejuhatus, vajaminevate SDK-de allalaadimine ja ülesseadmine; teooria (äpiarenduse põhialused, mõisted, Flutter spetsiifilised mõisted ja põhimõtted) õppimine; teooria (Flutter äpi „koostis“, ülesehitus)

11 – 17. mai - kõige esmasema äpi loomine, põhitalade kinnistamine ja praktiline kasutamine

18 – 24. mai - äppide reaalsetele seadmetele jooksma saamine (ingl k *deploy*)

25 – 31. mai - Flutter kasutajaliidese põhitõed (*UI layout*)

1 – 7. juuni – Flutter kasutajaliidese õppimine (paigutus vidinate Column ja Row abil)

8 – 14. juuni – Flutter vidinate omaduste (*properties*) muutmine

15 – 21. juuni – äppi piltide, erinevate fontide ja ikoonide lisamine, Google Fonts kasutamine

22 – 28. juuni – „paindlike“ ekraanipaigutuse (*Flexible layout*) põhitõed kasutades Expanded vidinat; nuppude lisamine Flutter äppi, funktsioonide käivitamine nupu vajutamisel

29. juuni – 5. juuli - olekuga ja olekuta vidinad (*Stateful* ja *Stateless Widgets*)

...

20 – 26. juuli – kordamine; pooljuhuslike arvude kasutamine Flutter äpis

27. juuli – 2. august - Flutter ja Dart „koodipakkide“ (*packages*) kasutamine ja projekti importimine (YAML faili õppimine); audiofailide mängimine kasutades audioplayers „koodipakki“;
3 – 9. august – koodi restruktureerimine (*refactoring*); dünaamiliselt erinevate andmete näitamine kasutajale; ise klasside ja vidinate loomine

10 – 16. august – OOP õppimine

17 – 23. august – *enum* andmetüübi õppimine keeles Dart

24 – 30. august – Ternaarsed tehted keeles Dart (süntaks, kasutamine jne)

...

14 – 20. september – Flutter teemade (*themes*) kasutamine äpi välimuse ühtsustamiseks; Navigator vidina kasutamine erinevatele ekraanidele liikumiseks

21 – 27. september – telefoni asukohaandmete saamine kasutades *geolocator* „koodipakki“, tehted koordinaatidega (Haversine’ valem), asukohaandmete lubade lisamine äppi

28. september – 4. oktoober – Asünkroonne programmeerimine Dart-is (süntaks, põhimõte)

5 – 11. oktoober – *Stateful* vidina elutsükkel

12 – 18. oktoober – API (aplikatsiooni programmeerimisliides, *application programming interface*) kutsed (*request*) keeles Dart kasutades *http* kogumit

19 – 25. oktoober – kuidas uuendada ekraani andmete uuendamisel; keerulisemate vidinate õppimine

26. oktoober – 29. november – Firebase

30. november – 20. detsember – Olekuhaldus

...

2021

...

4 – 10. jaanuar – eelnevalt õpitud teadmiste kordamine

11 – 17. jaanuar – *provider* õppimine, kasutamine

18 – 24. jaanuar – ideede kogumine enda äpi jaoks, parima idee valimine

25. jaanuar – 7. veebruar – parima idee valimine, äpi arhitektuuri, välimuse jne väljamõtlemine ja disainimine, vajaminevate teenuste (Firebase) aluse loomine, äpiprojekti loomine, Firebase konfiguratsioon Flutter äppi

8 – 14. veebruar – äpi põhiliste võimaluste planeerimine, kõige põhilisema kasutajaliidese tegemine disainikavade abil

15. veebruar – 7. märts – põhilisemate võimaluste implementatsioon äppi *

* Detailid peatükis Loovtöö Tulemus

8 – 21. märts – põhilisemate andmete lisamine äppi, mõnede võimaluste lisamine, koodi restruktureerimine *

* Detailid peatükis Loovtöö Tulemus

22. märts – 4. aprill – Kasutajate autentimine Firebase Auth’iga, pilve-andmebaasi toe lisamine

5 – 18. aprill – vigade otsimine ja parandamine (*debugging*), väiksemate, äppi tervikumaks tegevate võimaluste lisamine, kasutajakogemuse parandamine ja lihtsustamine (ingl k *UX*)

19. aprill – 2. mai – viimaste võimaluste lisamine, vigade parandamine (*debugging*), kasutajaliidese parandamine

3 – 9. mai – Google Play Store’i ja Apple App Store’i avaldamine *

10 – 16. mai – loovtöö päeviku lõplik korrastamine, esitluse loomine

19. mai – **loovtöö kaitsmine**

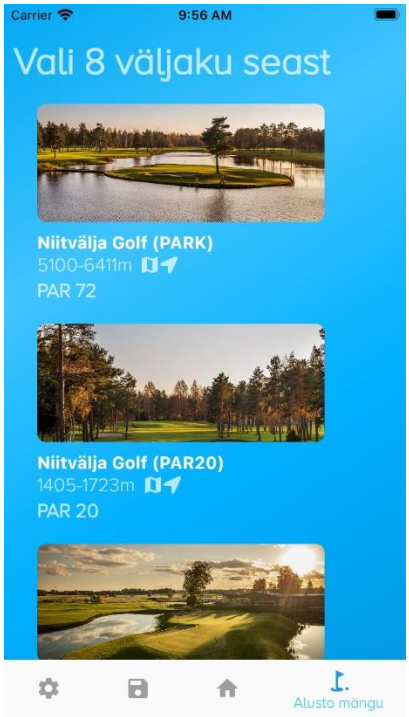
Töö tulemus

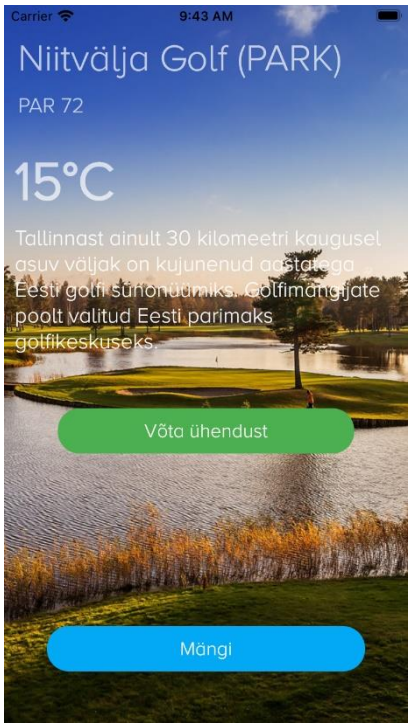
Otsustasin töö tulemuseks luua õpitud teadmisi kasutades mobiiliäpp. See näitaks, et valdan teemat ja oskan erinevaid tehnikaid ja õppetükke omavahel kombineerides luua terviku. Pärast eeldatava ajakava kirjapanekut lisasin enda äpile mõned nõuded:

- Äpp peab olema mulle relevantne – ma ei taha luua äppi, mida mul poleks vaja
- Äpp peab olema intuitiivse kasutajaliidese ja -kogemusega – ka teised, kes äppi kasutada tahavad, peavad seda oskama kasutada õpetuseta
- Äpp peab olema avatud lähtekoodiga – projekt peab olema saadaval GitHubis
- Äpp peab olema eriline – mingid võimalused/andmed/keeled peavad olema unikaalsed
- Äpp peab vähemalt osaliselt kasutama Firebase'i – kuna õppisin Firebase'i suhteliselt pikalt, peaks see olema äpis esindatud
- Äpp peab olema Androidi ja iOSi äpipoodides saadaval
- Äpp peab olema mõlema platvormi tunnustega (Androidil ja iOS-il erineb kasutajaliides natuke, võttes arvesse selle plavormi standardeid)

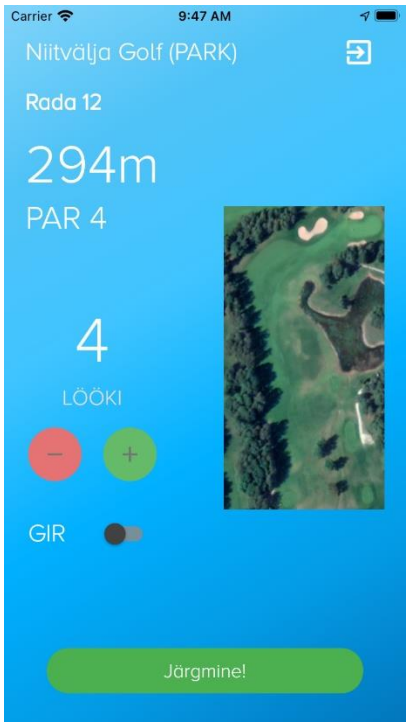
Minu äpp

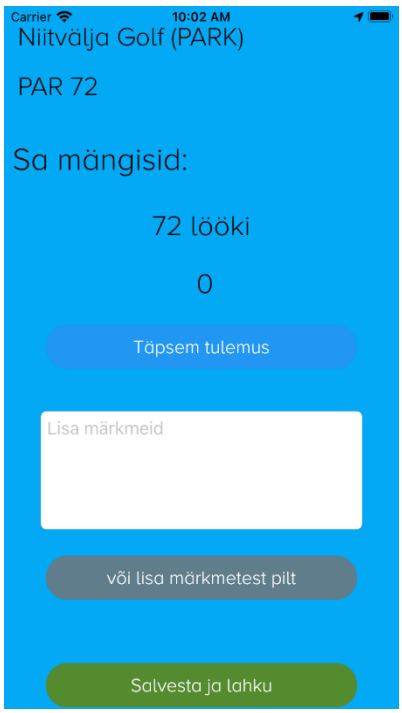
Valisin enda äpiks golfäpi tegemise. Valisin selle, kuna mulle väga meeldib golfi mängida. Pärast äpi tegemiseks kulunud aega (vt ajakavast), valmiski äpp. Toon siin dokumendis välja mõned kuvatõmmised. Terve äpp on muidugi komplekssem kui kuvatõmmised.






Carrier
9:45 AM
Niitvälja Golf (PARK)
Rada 1
107m
PAR 4
2
LÖÖKI
GIR
Järgmine!


Carrier
9:47 AM
Niitvälja Golf (PARK)
Rada 12
294m
PAR 4
4
LÖÖKI
GIR
Järgmine!


Carrier
10:02 AM
Niitvälja Golf (PARK)
PAR 72
Sa mängisid:
72 lööki
0
Täpsem tulemus
Lisa märkmeid
või lisa märkmetest pilt
Salvesta ja lahku


TELE2
09:27
Minu ringid
Laadi enda ringid siia seadmesse
103,2
Keskmine tulemus 18-rajal
13.05.2021 väljakul Niitvälja Golf (PAR20)
25
08.05.2021 väljakul Niitvälja Golf (PAR20)
31
08.05.2021 väljakul Niitvälja Golf (PAR20)
31
01.05.2021 väljakul Niitvälja Golf (PAR20)
29
01.05.2021 väljakul Niitvälja Golf (PAR20)
27
Minu ringid

Äpi vastavus seatud tingimustele

Selles osas räägin, kuidas täitsin kõik punktid enda seatud eesmärkidest.

- **Äpp peab olema mulle relevantne – ma ei taha luua äppi, mida mul poleks vaja**
  Kuna golf on minu üks lemmikhobisid, on äpp mulle relevantne, saan seda kasutada, kui lähen mängima
- **Äpp peab olema intuitiivse kasutajaliidese ja -kogemusega – ka teised, kes äppi kasutada tahavad, peavad seda oskama kasutada õpetuseta**
  Selle punkti täitmiseks, tegin mitu korda arendusperioodi jooksul „katset“. Andsin äpi enda õele kasutada ja ei öelnud talle midagi. Sellest sain huvitavaid andmeid, kuidas kasutajakogemust parandada. Äpi viimast versiooni oskab ta õigesti kasutada, kasutades äpis olevaid tekste ja kasutajaliidest.
- **Äpp peab olema avatud lähtekoodiga – projekt peab olema saadaval GitHub’is**
  Äpi kasutajaliidese ja Dart kood on saadaval GitHub’is
- **Äpp peab olema eriline – mingid võimalused/andmed/keeled peavad olema unikaalsed**
  Kuigi golfiäpid on suhteliselt tavalised, ei ole ma leidnud ühtegi, mis oleks eesti keeles ja milles oleks täpsed andmed Eesti väljakute jaoks. Kõikidel äppidel esineb nt radade pikkustega vigasid.
- **Äpp peab vähemalt osaliselt kasutama Firebase’i – kuna õppisin Firebase’i suhteliselt pikalt, peaks see olema äpis esindatud**
  Äpp kasutab Firebase’i selleks, et kasutaja mängtud ringid pilve laadida. See on vajalik, kui vahetad telefoni või kui tahad, et ringid oleksid turvaliselt varundatud. Pilveandmebaas on Cloud Firestore’iga tehtud. Kasutaja autentimine (ainult sisse logitud kasutajad saavad turvalisuse põhjustel ringe üles/alla laadida) toimub Firebase Auth’iga.
- **Äpp peab olema Androidi ja iOSi äpipoodides saadaval**
  Äpp on loovtöö kaitsmise hetkeks olemas nii Google Play poes kui App Store’is.
- **Äpp peab olema mõlema platvormi tunnustega (Androidil ja iOS-il erineb kasutajaliides natuke, võttes arvesse selle plavormi standardeid)**
  Minu äpp käitub vastavalt platvormile. See hõlmab väikeseid erinevusi, mida kirjeldasin siin (joonised 1, 2), aga ka kasutajaliidese silmnähtavaid erinevusi.
  Näiteks, kui soovite oma mängitud ringi kustutada, tekib iOS’is selline kinnitusvõimalus, mida nimetatakse Action Sheet:

Androidil aga tekib hoopis hüpikaken, mille nimi on AlertDialog ja mis kasutab Material Design’i:

Kokkuvõte

Loovtööst ja selle protsessist õppisin palju. Üks suurimaid õppetunde oli aja planeerimine. Loovtööd tuleb teha aegsasti ja päevikut pidevalt täita. See oli ka üks minu loovtöö puudujääkidest – lõpetasin päeviku kirjutamise vaid nädal enne kaitsmist.

Minu loovtöös oli palju positiivseid tulemeid. Õppisin uusi teadmisi ja oskusi, leidsin uue hobi. Samuti õppisin pidama ajakava, mida pole varem teinud. Aga võimalik et kõige olulisem positiivne külg töö juures on arusaam, et kui üritada ja õppida, olen võimeline hakkama saada nii raskete (nt programmeerimine, äpidisain) kui ka põhjalike (loovtöö kui protsess) ülesannetega.

Siiski leidub ka asju, mida oleksin võinud teha teisiti või paremini. Üks suurimaid neist ongi ajaplaneerimine. Oleksin võinud alustada varem, et loovtöö lõpetamiseks (päeviku viimistluseks jne) oleks jäänud rohkem aega. Samuti oleksin võinud teha loovtööd koos teistega, kas klassikaaslaste või sõpradega, mis oleks võimaldanud rohkem abi küsida/anda ja oleks olnud suhtlusrohkem valik.

Kokkuvõtteks loovtööle võin öelda, et jäin sellega kui protsessiga rahule. Kuigi esines probleeme või ebatäpsusi, sain nendega kokkuvõttes hakkama ja õppisin töö käigus mitmeid väärtuslikke oskusi ja õppetunde.