Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap
Institutt for datateknologi, elektroteknologi og realfag

Fotspor 2.0

Systemdokumentasjon

Versjon 3.0

Dokumentet er basert på Systemdokumentasjon utarbeidet ved NTNU. Revisjon og tilpasninger til bruk ved IDER, DATA-INF utført av Carsten Gunnar Helgesen, Svein-Ivar Lillehaug og Per Christian Engdal. Dokumentet finnes også i engelsk utgave.

REVISJONSHISTORIE

Dato	Versjon	Beskrivelse	Forfatter
25.04.22	1.0	Første iterasjon av dokumentet	Thomas Eikhaug, Håkon Herrevold og Filmon Fisseha Weldeyohannes
05.05.22	2.0	Andre iterasjon av dokumentet	Thomas Eikhaug, Håkon Herrevold og Filmon Fisseha Weldeyohannes
22.05.22	3.0	Tredje iterasjon av dokumentet	Thomas Eikhaug, Håkon Herrevold og Filmon Fisseha Weldeyohannes

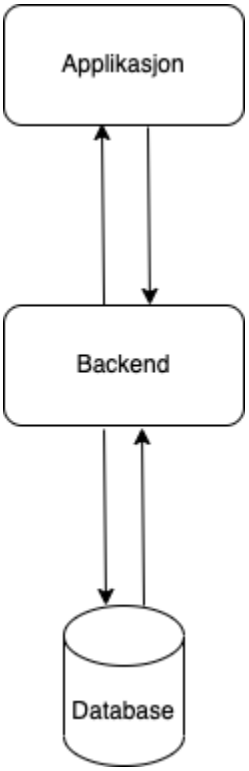
INNHOLDSFORTEGNELSE

1 INNLEDNING

Denne rapporten inneholder systemdokumentasjonen til Fotspor 2.0. Dokumentet har oversikt over tekniske valg gruppen har besluttet. Studentene, oppdragsgiver og veileder har hatt jevnlige møter under prosjektperioden for å komme frem til den systemspesifikke oversikten for Fotspor 2.0.

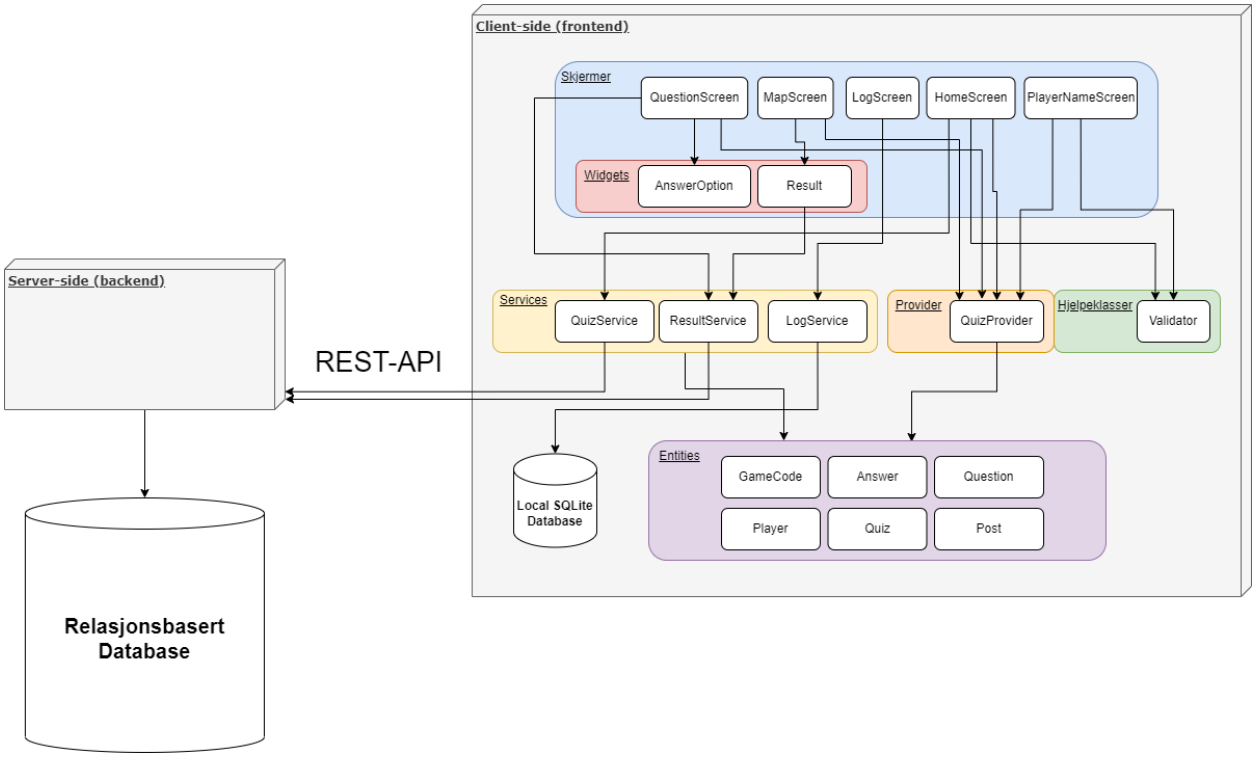
2 ARKITEKTUR

Prosjektets fokus har kun vært på klient-side delen av løsningen, i form av Fotspor 2.0-appen. Database og server-tjeneste skal ikke lages av gruppen, og er opp til Medielab, HVL selv å finne en løsning til. Den tenkte løsningen fra gruppen er at det vil være en backend (server-tjeneste) som appen snakker til, og vil være mellommannen mellom database og applikasjon. Som illustrert i Figur 1, vil de ulike delene kommunisere via et REST-api der data blir sendt i JSON format frem og tilbake etter behov.



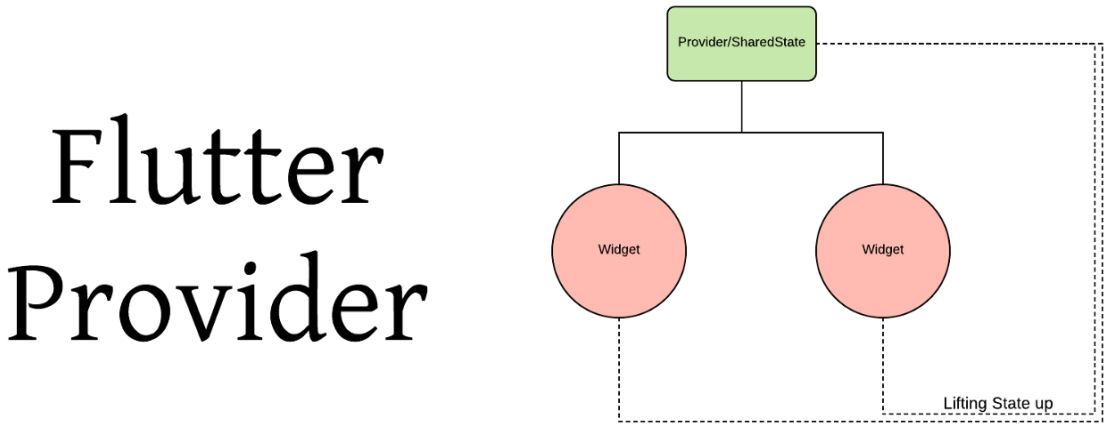
Figur 1: Hvordan løsningens deler kommuniserer.

Appen sin arkitektur bygger på Flutter sin bruk av screens, som en form av side eller “pages” som det ville vært på en web-app. Hver skjerm har logikk, tilstand (state) og byggemetode, som bygger det visuelle på telefonens skjerm. En screen er hovedsakelig en stor widget, som inneholder flere mindre widgets. Dette er et sentralt konsept i Flutter, der widgets blir brukt som byggeklosser til å lage større widgets. Det finnes lite grenser for hva som er mulig å skape. Figur 2 er en mer detaljert versjon av Figur 1, og viser hvordan klient-siden er bygd opp med screens, widgets, services, hjelpeklasser, provider og entities.



Figur 2: Klient-side arkitektur

Når en har et større prosjekt er det enda viktigere å holde styr på verdier og tilstander (states) til de mange widgetsene. Derfor har det blitt tatt i bruk state management som gjør det enkelt for hele appen å holde styr på tilstand og verdier gjennom kjøringen. Dette blir gjort gjennom Flutter sitt eget provider bibliotek, som er enkelt å sette opp. Provideren i appen kan bli sett på som en lagringsplass av verdier og tilstander som alle widgets i widget-treet kan hente fra. Alle verdier som hentes derfra vil alltid stemme overens med hverandre, ettersom en endring av tilstand i provideren vil varsle alle widgets om endringene. Når en varsling skjer vil alle widgets som bruker verdiene automatisk oppdatere seg, nå med de nye verdiene. Figur 3 viser en enkel illustrasjon av hvordan provider hever tilstanden over widget-treet.



Figur 3: Illustrasjon av Flutter provider, hentet fra

https://medium.com/flutter-community/flutter-pragmatic-state-management-using-provider-5c1129f9b5bb

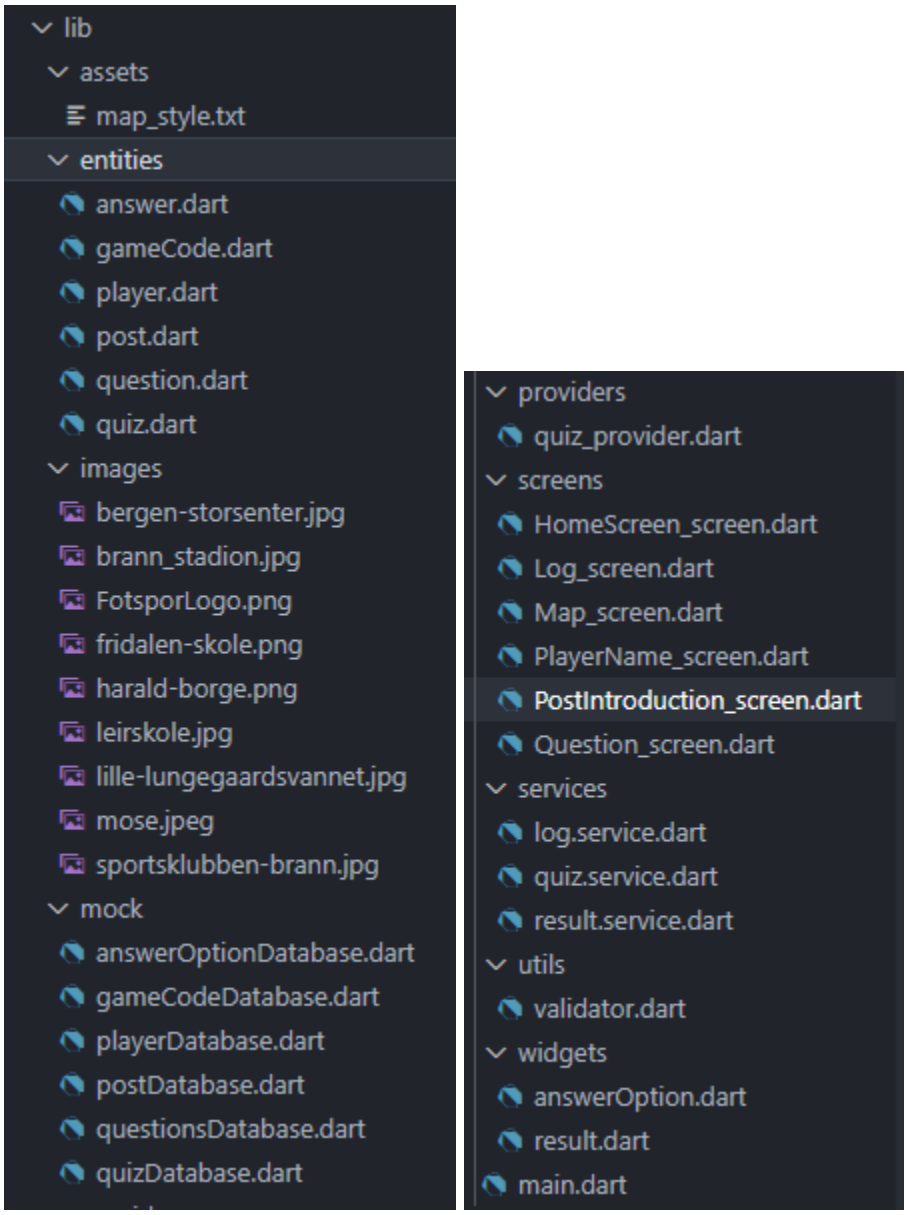
(05.05.2022)

3 PROSJEKTSTRUKTUR

Applikasjonen er bygd med rammeverket Flutter utviklet av Google, som tar i bruk Dart som programmeringsspråk. Hver applikasjon i Flutter har en egen Android og iOS mappe der kode automatisk blir generert fra dart filene i lib mappen. Android bruker henholdsvis enten Java eller Kotlin, mens iOS bruker Swift som programmeringsspråk.

Prosjektstrukturen vises i sin helhet i Figur 4. I Entities-mappen finner en alle objektklasser som er tatt i bruk i appen, som for eksempel Quiz og Player. I images-mappen ligger det bilder som blir tatt i bruk. Flere av disse er mock-bilder, ettersom appen ikke er tilkoblet en database. I mock-mappen ligger mockdata for spørsmål, poster etc. Providers-mappen har en provider som administered livssyklusen, og er state management løsningen til applikasjonen. Screens-mappen er alle skjermene som vises på applikasjonen. Services-mappen inneholder hjelpeklasser for henting av data, og utils-mappen hjelpeklasser for validering, som blant annet screens-klassene tar i

bruk. Til slutt inneholder Widgets-mappen selvskapte widgets, som de større screen-widgetsene tar i bruk.



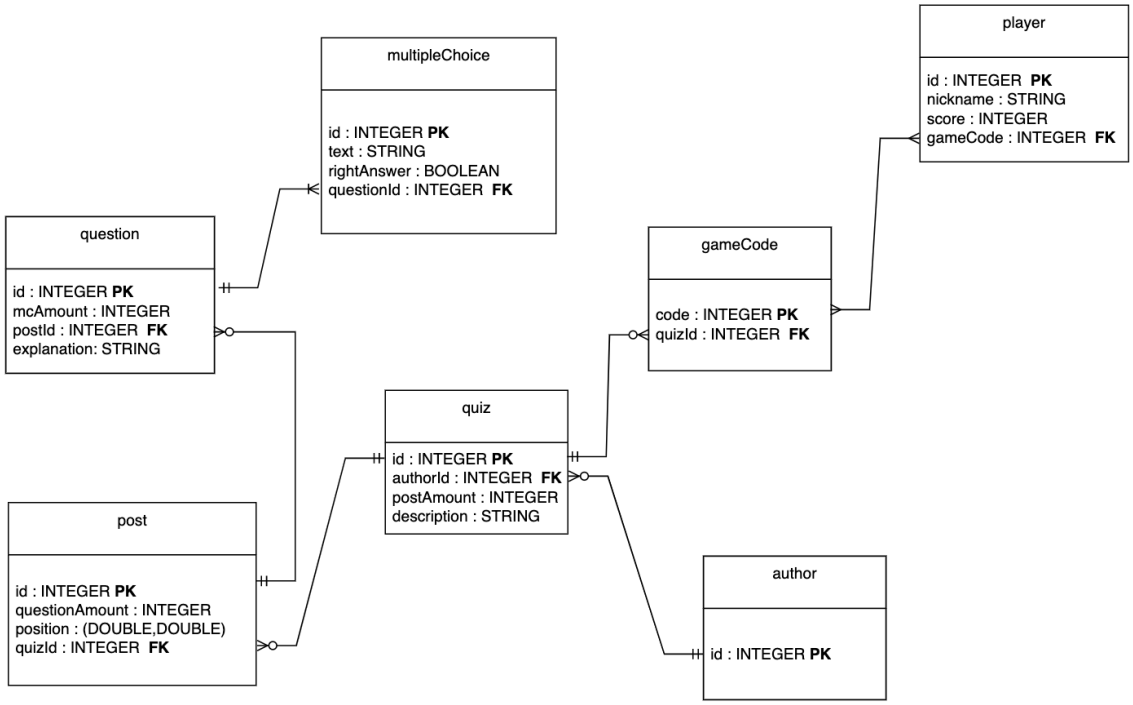
Figur 4: Fil- og katalogstruktur

4 KLASSEDIAGRAM

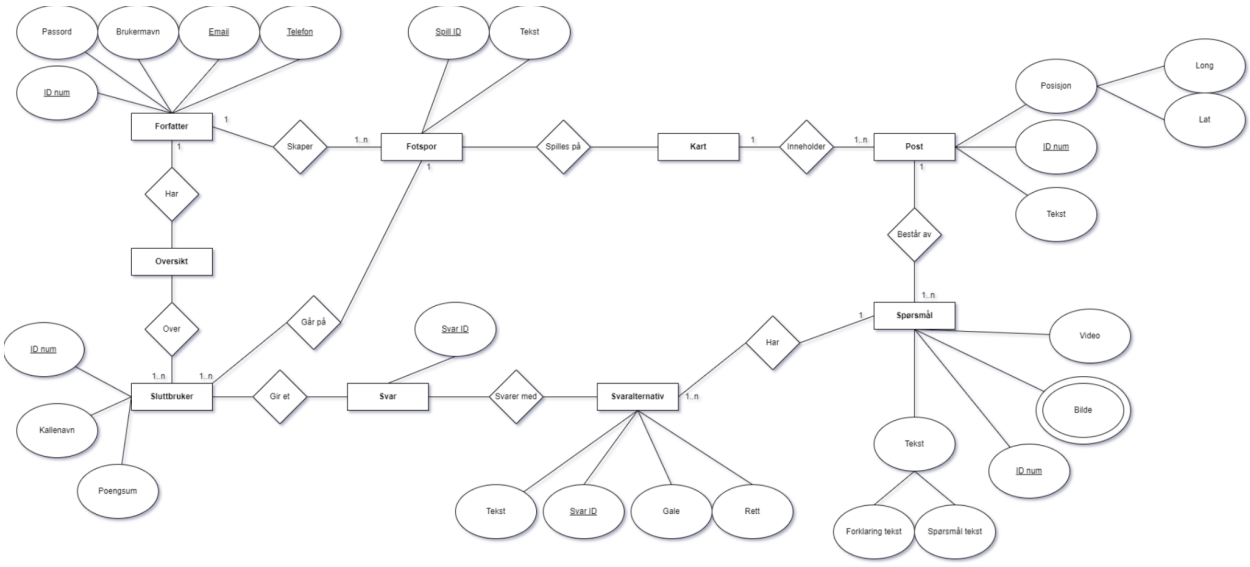
Klassediagram har ikke blitt lagd for dette prosjektet, ettersom arkitekturdiagrammet gir en bedre oversikt over hvordan applikasjonen er satt sammen. Flutter sitt programmeringsspråk Dart, er et objektorientert språk, men fokuserer ikke like mye på objektorientert programmerings (OOP) aspekter som for eksempel et Java program ville gjort. For et Java program er det mer naturlig å lage et klassediagram.

5 DATABASEMODELL

Gruppen har laget en databasemodell (Figur 5), som er visualisert ved bruk av standarden Unified Modeling Language (UML). Oppgavens fokus var bare å skape en frontend-løsning i form av en app, som medfører at en database ikke har vært en del av oppgaven. Gruppen utformet likevel en eventuell relasjonsbasert databaseløsning til Fotspor 2.0. Gruppen, veileder og en av utviklerne av Fotspor 1.0, diskuterte eventuelle andre løsninger som også ville fungere, og da ville det eventuelt vært en dokumentbasert database. Konklusjonen på den diskusjonen endte med at det ikke ville spilt mye rolle hvilken løsning som blir valgt. Logikken i appen vil stort sett være lik uansett hva som blir valgt, valget vil være opp til Medielab, HVL når de overtar prototypen til Fotspor 2.0. Figur 6 viser en ER-modell av Fotspor 2.0, som i kombinasjon med databasemodellen gir et godt innblikk i hvordan de ulike entitetene henger sammen i systemet.



Figur 5: Database UML modell



Figur 6: ER-modell for Fotspor 2.0

6 SERVER-TJENESTER

Det er nevnt tidligere i kapittel 2 og kapittel 5 at oppgaven kun gikk ut på en frontend-applikasjon, og gruppen jobbet derfor ikke med server-tjeneste eller database. Dersom det skulle blitt implementert ville et REST-api blitt tatt i bruk for at de ulike delene skal kommunisere med hverandre, som sett på Figur 1 og 2 i kapittel 2.

7 SIKKERHET

Fotspor 2.0 er laget på en måte, slik at det vil være så lite som mulig lagring av persondata. Det som blir lagret av sluttbrukerne, er bare selvdefinerte kallenavn og resultater, og det er ingen innlogging for å spille. Det vil derfor ha minimale påvirkninger på GDPR, som var et tema Medielab, HVL var strenge på at det skulle være lite påvirkning. Det er heller ikke behov for kryptering av data som går mellom appen og database. Gruppen har tatt i brukt valideringsmetoder for alle innskrivingsfelt, for å beskytte databasen for SQL-injection. Om en backend skulle blitt laget på et senere tidspunkt, ville det vært lurt å gjøre en validering på server-siden også, ettersom å bare gjøre det på klient-siden ikke er sikkert nok i seg selv.

8 INSTALLASJON OG KJØRING

- Flutter Provider[1] blir brukt som state management løsning for appen.
- Google_maps_flutter[2] blir brukt for å hente kart.
- Geolocator[3] brukes for å finne posisjon med GPS.

[1] https://pub.dev/packages/provider
[2] https://pub.dev/packages/google_maps_flutter
[3] https://pub.dev/packages/geolocator

Om en skal laste ned appen for testing før en eventuell lansering, så ville det skjedd på to ulike måter:

1. Den første og mest tungvinte, ville vært å nedlaste gjennom å sette telefonen i en utviklermodus. Dette kan enkelt bli gjort på Android-telefoner ved å trykke på telefonens “Build number” inne på innstillinger, syv ganger (Syv ganger er standarden på Android-telefoner, men enkelte Android-telefoner kan ha et annet antall ganger). Etter dette må en inn i utviklerinnstillingene og ta på USB-feilsøking. Til slutt må en koble telefonen til en pc som har appen sin kode inne i en IDE, som for eksempel Visual Studio Code. Deretter velger en å debugge koden på den tilkoblede telefonen, og appen vil så bli nedlastet.
2. Den andre måten appen kan nedlastes på ville vært å registrere appen i Google Play eller App Store sine egne løsninger for å publisere apper for testing. Dette ville gjort det betydelig lettere for testere å laste ned appen.

Om appen var lansert, så ville det fungert som de fleste apper, at den blir lastet ned via hver plattform sine appkatalog-løsninger, som Google Play og App Store.

9 DOKUMENTASJON AV KILDEKODE

Dokumentasjon av kode har skjedd gjennom kommentarer lagd av gruppen selv. Det er fokus på at kommentarene skal være korte og presise, som vist på første bildet, Figur 7. Språket på variabler, metoder og funksjoner har vært på engelsk, ettersom alle eksterne biblioteker bruker det som en standard, og det er viktig at alt er uniformt.

```dart
// Check if all posts are visited
bool outOfPosts(Quiz quiz) {
  bool isDone = true;
  for (Post p in quiz.posts) {
    if (!p.isAnswered) {
      isDone = false;
    }
  }
  return isDone;
}

// Returns to the home screen and opens all posts to reset the game
void returnToHomeScreen(GameProvider p) {
  p.reset();
  timer?.cancel();
  Navigator.pushReplacement(
    context,
    MaterialPageRoute(
      builder: (context) => HomeScreen(),
    ), // MaterialPageRoute
  );

  for (Post p in p.quiz.posts) {
    p.isAnswered = false;
  }
```

```dart
IconButton(
  onPressed: (() {
    showDialog(
      context: context,
      builder: (context) => AlertDialog(
        title: const Text("Introduksjon til rute:"),
        content: Text(gameProvider.quiz.description),
        actions: [
          TextButton(
            onPressed: () {
              Navigator.pop(context);
            },
            child: const Text("Lukk"),
          ), // TextButton
        ],
      ), // AlertDialog
    );
  }),
  icon: const Icon(Icons.info),
), // IconButton
```

Figur 7: Manuell, og automatisk kommentering av kode

Flutter kommenterer også kode automatisk selv i build-metoden, for å gjøre hierarki strukturen tatt i bruk mer leselig. Et eksempel på dette er Figur 7, der de røde pilene peker. Alle widgets sine sluttpunkter i koden blir automatisk kommentert.

10 KONTINUERLIG INTEGRASJON OG TESTING

Fotspor 2.0 ble ikke utviklet ved bruk av testdrevet utvikling (TDD), men ble likevel testet regelmessig gjennom utviklingen. Dette ble gjort ved å lage testvandringer som kunne bli spilt gjennom uten å måtte gå utendørs. Appen ble også lastet ned på egen telefon for å teste hvordan appen ville fungerte utenfor emulator på pc.

11 REFERANSER

Google, 2022, *Provider,* tilgjengelig fra:

https://pub.dev/packages/provider (hentet 05.05.2022)

Google, 2022, *google_maps_flutter,* tilgjengelig fra:

https://pub.dev/packages/google_maps_flutter (hentet 05.05.2022)

Google, 2022, *Geolocator,* tilgjengelig fra:

https://pub.dev/packages/geolocator (hentet 05.05.2022)

Siddiqui, Atif., 2019, *Pragmatic State Management Using Provider*, tilgjengelig fra:

Pragmatic State Management Using Provider. | by Atif Siddiqui | Flutter Community | Medium (hentet 05.05.2022)