Apputvikling - Espen Zaal - S198599

Rapport for Mappe 3 - Skisteder

Introduksjon

Appen er basert på tjenester levert av et nettsted kalt fnugg.no. Dette er et nettsted som alle de store skitsentre i Norge står bak, og skal vise relevant informasjon om de ulike skisentrene, løypene, heisene, priser, vær og snøforhold, blant annet.

Det finnes et åpent API http://fnuggapi.cloudapp.net/ som jeg da benytter et utdrag av informasjon fra.

Formål og Idé

Appen er ment å være et hjelpemiddel for skiinteresserte som ønsker inspirasjon på hvor de eksempelvis skal stå på ski til helgen.

Bachelorprosjektet jeg skal i gang med etter nyttår skal også jobbe med dette API'et, i en annen sammenheng, og ikke som android app, men det å sette seg inn i API'et og jobbe med store Json-objekter er et formål i seg selv.

Ønsket med appen er at det skal være enkelt å bruke den og finne relevant informasjon om både skistedene og hvor langt det er til det enkelte skisted.

Avgrensning av prosjektet

Det finnes veldig mye informasjon i API'et og mye av det er veldig interessant å kunne vise i en app. Min app vil i utgangspunktet laste ned Id, navn og GPS-data om hvert skisenter, og resultatet skal vises i Google Maps. Hvis en klikker seg inn på en "pin" i kartet så skal det hentes mer utfyllende informasjon om skisenteret og kjøretid og distanse dit. Det kommer mer informasjon om dette senere.

Krav til testoppsett

Appen er laget med bruk av GPS og Google Maps blant annet. Dette gjør at det ikke har vært mulig å teste appen i emulatoren med API 19 og oppløsningen som Nexus-telefonen har.

Jeg testet appen på din telefon og det så greit ut, men jeg vet ikke om alt vil se like bra ut som det gjør på min telefon.

Hvis du vil kan jeg komme til deg og vise hvordan det eventuelt ser ut på min telefon.

For kunne kjøre appen må man ha GPS slått på, samt internettilgang.

Teknisk dokumentasjon

Bruk av teknologi

Apache Commons-IO

IOUtils er et utility-bibliotek som har gode metoder for å jobbe med URL'er, input/output-streams osv. Her kunne jeg like godt bruk Java sine InputStreams, men følte at det er greit å bruke noe som også er Open Source og mye brukt ellers i verden. Feks så blir kall på et API gjort så enkelt: `jsonResult = IOUtils.toString(url);`

GSON

Det har vært forsøkt veldig mye i forbindelse med å jobbe mot API'et til fnugg.no. Det finnes utallige måter å jobbe med Json-data på. GSON er utviklet av Google, og gir veldig god arbeidsflyt når en skal traversere dype Json-objekter.

Man importerer data via Apache Commons-IO bilbioteket som nevnt ovenfor og traverserer Json-objektet på denne måten:

```
jsonResult = IOUtils.toString(url);

parser = new JsonParser();

root = parser.parse(jsonResult);

outerArray = root.getAsJsonObject().get("hits")

    .getAsJsonObject().getAsJsonArray("hits");
```

Her ser man hvordan man kan gå fra "root" og nedover i strukturen ved å bruke "getAsJsonObject" og lignende metoder for å traversere hvert gren i Json-objektet som enten en verdi som man velger via ".get("name").asString()" eller som objekter eller arrays.

GSON tilbyr noe annet som ikke ble implementert, men som opprinnelig var ønskelig. Det er at man mapper Java-klasser mot JsonObjects. Man kan sette Annotations på javaklasene, som motsvarer json-objektenes "key", og GSON importerer og traversere da hele resultatet automatisk.

Tekniske detaljer

Repoistory - Singleton pattern

Ettersom dataene som brukes i stor grad er “live data”, som avhenger av hvor man befinner seg, så har det ikke vært aktuelt med permanent lagring til database. I stedet er det brukt et Repository i form av et Singleton-objekt. Repository.java er et slikt objekt, og som har statiske medlemmer. Blant annet én instans av seg selv, og en privat konstruktør.

Dette medfører at man bruker en statisk metode “getInstance” for å få tak i en instans av repository, og at det aldri vil kunne lages flere versjoner. At listen med Resorts også er statisk betyr at disse dataene vil leve like lenge som applikasjonen kjører, uavhengig av aktiviteter som startes på nytt.

UrlBuilder - Builder pattern

Det å bygge en lang Url/Uri dynamisk basert på forskjellige parametre er litt omstendelig. Det er derfor laget en klasse som heter FnuggUrlBuilder.java.

Denne klassen har en statis instans av seg selv og diverse andre variabler. Man starter en “url-bygging” med å opprette et objekt av klassen og så kjøre metoden “startUrl()”.

Hver metode returnerer en instans av seg selv (den statiske), og bygger så litt av Url-strengen i prosessen. Man kan på denne måten joine/skjøte sammen flere metodekall før man til slutt velger metoden “build()”, som returnerer den ferdige strengen.

Bruk av Google Distance Matrix Api

For å kunne regne ut hvor langt det er fra “min” lokasjon til hvert av skisenterne brukes Google sitt Distance Matrix Api. For hver Resort som lastes ned via FnuggApi vil GPS-dataene legges inn i URI’en for Google Api’et og de relevante data derfra lastes ned og legges inn i Repository.

Redusering av båndbreddebehov

Dataene som brukes er “live data”, men det er ikke så kritisk at man trenger å laste ned alle Resorts og alle distanser på nytt og på nytt.

Det er derfor en boolean “isLoaded” i Repository som settes i det man har lastet alt én gang, og som forhindrer at man ikke laster ned alle dataene på nytt hver gang man vender på telefonen og aktiviteten starter på nytt.

Designvalg og UX-funksjonalitet

Design

Det er ikke så mye å si om design av appen. Det er prøvd å bruke farger som står godt sammen, samt at alle viktige egenskaper ved appen er tilgjengelige via ActionBar i form av ikoner som er enhetlig i utforming og farge.

Det er mange begrensninger i Android på å endre Themes/Styles i layoutene generelt, og i ActionBar og PreferenceFragment spesielt. Grunnet dette er det ikke brukt i stor grad i appen.

Brukeropplevelse

Det er lagt mye arbeid inn i funksjonaliteten for å gi brukeren mulighet til å bruke appen på sin måte. Det finnes selvsagt mye mer, av både data tilgjengelig i APi’ene og i Android til å legge til enda mer, men et sted må man stoppe.

Google Map

En GoogleMap-funksjon i Android gir mye funksjonaitet. I MainActivity er det en privat klasse om implementerer OnMapReadyCallback-interfacet og onMapReady(GoogleMap googleMap).

Selve kartet blir plottet og sentrert rundt “min posisjon” som er siste avleste GPS-posisjon gjort av android. Alle skistedene blir så plottet inn i karter og er klikkbar, nor som medfører overgang til ny aktivitet der det vises informasjon om det aktuelle skistedet.

I kartet kan man også gjøre “long click” på et sted der man ønsker å flytte “min posisjon” til. I det dette registrerers som ny poisjon vil Google Api’et kalles på nytt og alle beregninger av avstand og kjøretid blir utført på nytt. Man kan flytte “min posisjon” flere ganger, og nye beregninger blir gjort hver gang.

Hvis man plasserer markøren på en lokasjon som ikke er kjent av Google, altså der man ikke kan komme til med bil eller andre fremkomstmidler så vil appen ikke registrere dette som ny posisjon, og man vil få en beskjed på skjermen om dette. Appen vil krasje om man ikke tar dette forbeholdet.

Ønsker man å resette appen og bli kvitt “my custom position” så må man trykke på “refresh”-ikonet for å laste alt inn på på nytt.

Listevisning av skisteder

Listevisningen er enkelt i sin visning og ikke ment for å vise informasjon om annet enn navn på skistedet og kjøretid og avstand. Trykker man på et skisted vil man som i kartet bli sendt til ny visning for utfyllende informasjon om skistedet.

Man kan sortere listen basert på avstand og kjøretid.

Innstillinger

Her er det brukt en del tid. I det man gjør endringer i denne visningen og trykker på “tilbake” så startes MainActivity på nytt og FnuggApi leser av hva som er markert i Innstillinger og gjør sitt API-kall basert på det.

Det ene aspektet under Innstillinger er Zoom-nivået i kartet i det det lastes. Det finnes ingen fasit på hva som er rett zoom og dette gir da brukeren mulighet til å justere dette etter eget behov. Man kan selvsagt zoome i kartet, som i Google Maps, men det er tungvindt om det hele tiden er for mye eller for lite zoom i utgangspunktet.

Det andre aspektet som dekkes er filtrering av søkeresultater. Man trenger kanskje ikke å vite som skistedene i Nord-Norge om man ikke skal dit på ski den nærmeste tiden. Dette kan filtreres under Regioner. Om ingenting er markert så vil det ikke være noe regionfilter, men om man markerer en eller flere så vil bare disse vises i resultatet.

Samtidig kan det hende at man bare vil vite om skisteder som har minst 5-10-15-20-25 skibakker eller skiheiser. Dette kan man sette under de to filterne som omhandler skibakker og skiheiser.

Det siste aspektet som dekkes under Innstillinger er Debug. Dette er en funksjonalitet som har vært i bruk under testing. Det å laste ned 90 skisteder hver gang man laster appen eller gjør noen endringer gjør at Google sin API-kvote fylles ganske fort. Hvis man har på Debug-modus så vil man bare få ned de 15 første skistedene fra API'et.

Er denne modusen deaktivert så vil alle skistedene lastes ned.

Utfordringer

Begrensninger i InputStream

Det har vært utfordrene å jobbe med stå store Json-objekter som et fullt API-kall her vil gi som resultat. Uansett om jeg har prøvd med Java sin InputStream, BufferedReader, BufferedInputStream eller med Apache Common-IO sine løsninger så får jeg ikke lastet ned en Json-string større enn 4kb.

Det har vært gjort mange forsøk, og jeg har endt opp med å dele opp i mindre spørringer i stedet. Det kan for så vidt være en god ting i mobilverdenen uansett.

Det som skjer ved større objekter er at ved ca 4kb data så kutter strømmen. Hva de skyldes er ukjent, og det er ikke klart hvor det stopper, men ingen Exceptions blir kastet.

Google Distance Matrix API

En gratis Google Developer-konto har begrensninger på hvor mange kall man kan gjøre på én dag, samt hvor mange på kort tid, hvor mye data totalt i løpet av en dag osv.

Dette gjør at det ikke er hensiktsmessig å laste ned alle 90 skisteder hver gang man skal teste appen. Ved flere anledninger har jeg overskredet kvoten og blitt stengt ute for resten av dagen. Idag, fredag kl 0900 fikk jeg senest denne beskjeden, og kan dermed ikke gjøre siste finish på layout.

For å løse dette har jeg implementert en “debugmodus” under innstillinger. Så lenge den er “checked” så vil det bare lastes ned 15 skisteder. Slå man denne funksjonaliteten av så vil alle 90 skistedene lastes ned.

Applikasjonen vil ikke avsluttes

Uansett hva som prøves så vil ikke applikasjonen avsluttes når det trykkes på Avslutt eller på “back”-knappen til telefonen. Det ligger ingenting på backstack.

Det som er prøvd:

- android.os.Process.killProcess(android.os.Process.myPid());
- System.exit(0)
- finish()
- finishAffinity()
- onBackPressed()

Videre har jeg lest om Exit-knapp og det anbefales faktisk å ikke ha det. Man kan lese her How do I "Quit" an Android App?. Den siden har også en link til en Google Android Developer forum, der det første svaret er fra en Google ansatt

Det får vel stå som uferdig funkjsonalitet, selv om det antageligvis ikke er behov for en slik knapp.

Layout

Bilder

I henhold til Fnugg sitt Api skal alle bilder være like store, innenfor den sammen Uri. Feks bildet "image_16_9_m" være 600 x 338 pixler. Ikke alle er det, og dermed er det litt ujevn kvalitet på bildene både i Portrait og Landscape. Det har ikke vært mulig å lage noe automatisk skalering og justering av størrelse ettersom noen bilder er helt ubrukelige i annet format enn det de er tenkt som.

Farger - Styles

Spesielt PreferenceFragment er vanskelig å formatere. Det samme gjelder ActionBar. Man må kjenne til Android sitt "Style-system" og hvordan man skal overstyre de enkelte komponenter.

Selve utseende blir ikke helt gjennomført når man ikke gjør hele den jobben, og derfor er det noen småting som ikke blir bra. Blant annet så er teksten i en "Toast.makeText" omgitt av en padding, og man kan se at fargepaletten ikke stemmer helt over ens.

Egenvurdering av resultatet

Teknisk sett føles dette som den klart beste appen jeg har laget. Jeg har hatt frihet til å gjøre akkurat som jeg vil, og det har vært interessant å sette seg inn i både Google og Fnugg sitt API, samt det å håndtere og sammenstille flere data til ett produkt.

Det ble tidlig bestemt å ikke bruke database for å ta vare på data, men det var før Google sitt Distance Matrix API ble tatt i bruk. Det at man må kalkulere distanser på nytt hver gang man endrer "min posisjon" er selvsagt, men man blir veldig oppmerksom på hvor mye data som sendes frem og tilbake hele tiden.

Samtidig ville håndtering av bilder vært lettere ved å ha en database. Man kan feks da ha et primærbilde man ønsker å bruke, og om det ikke passer med oppløsningen så kan man falle tilbake på et "reservebilde". Dette kunne også gjøres via vanlige API-kall, men det har ikke blitt tid til det.

Ellers er utseende og layout ikke min sterkeste side og ettersom alt som skjer i kulissene likevel er min sterkeste side så ender det alltid opp med at selve utseende ikke blir så

bra som jeg skulle ønske. Noen har det omvendt og dermed lider av samme, men motsatte problem. Fin app med lite funksjonalitet..