



s114143 - Alexander Skjolden

2015-11-27

Sammendrag

Til mappe 3 i faget DAVE3600 (Applikasjonsutvikling) 2015, ønsket jeg å lage en app som melder fra om avvik i Ruter sine rutetider. Eksempel på avvik kan være «buss-for-tog», forsinkelser m.m. Som pendler fra Ski til Høyskolen ser jeg ikke behov for en app som bare forteller meg når toget skulle gå. Dette er informasjon jeg mener enhver pendler lærer utenat i løpet av veldig kort tid (uansett reiserute). En slik app kan dermed fort bli liggende ubrukt.

En app som derimot kan gi et konsentrert oversiktsbilde av nåværende avvik vil kunne brukes daglig. Jeg ser for meg muligheten til å abonnere på steder (holdeplasser) og liner (tog, trikk, buss m.m). Et slik abonnement vil kunne brukes som snarveier (favoritter) for å hente ut informasjon som er relevant for brukeren raskere.

Innhold

| | |
|--|-----------|
| I Prebetingelser | 5 |
| 1 Google Maps | 5 |
| 1.1 Emulator | 5 |
| 1.2 API nøkkelen | 5 |
| II Brukermanual | 6 |
| 2 Navigasjonsskuff | 6 |
| 3 Søk | 6 |
| 3.1 Linjenummer | 6 |
| 3.2 Lokasjon | 7 |
| 3.3 Kart | 7 |
| 3.3.1 ClusterManager | 8 |
| 4 Favoritter | 8 |
| 5 Stasjonsvisning | 9 |
| 5.1 Stasjonsfilter | 9 |
| 5.2 Stasjonsstatistikk | 10 |
| 6 Instillinger | 10 |
| 6.1 Generelt | 11 |
| 6.2 Stasjonsvisning | 11 |
| 7 Om | 12 |
| 7.1 Mellomlagring | 12 |
| 8 SwipeRefreshLayout | 13 |
| 9 Autocomplete | 13 |
| III Bruksmønster | 13 |
| 10 Daglig bruk | 13 |
| 11 Engang reise | 13 |
| IV Teknisk del | 14 |
| 12 Ruter Reis API | 14 |
| 12.1 Dokumentasjon | 14 |
| 12.2 Utfordringer | 14 |
| 13 RuterSugar | 14 |
| 13.1 Gson | 14 |
| 13.2 Retrofit | 14 |
| 13.2.1 Callback vs AsyncTask | 15 |
| 13.2.2 AsyncTask eksempel: | 15 |
| 13.2.3 Callback eksempel: | 15 |
| 14 ClusterManager | 16 |
| 15 Feature Creep | 16 |
| V Koden | 16 |

| | | |
|-----------|---------------------------------|-----------|
| 16 | Fragmenter | 16 |
| 17 | Navigasjonsbaren | 16 |
| 18 | Adapterene | 17 |
| 19 | Datalagring | 17 |
| 19.1 | Settings | 17 |
| 19.2 | PersistentCache | 17 |
| 20 | Helper pakken | 17 |
| 20.1 | UTM vs Lat/Lng | 17 |
| 21 | Cluster pakken | 17 |
| 21.1 | CustomClusterRenderer | 17 |

Del I

Prebetingelser

1 Google Maps

1.1 Emulator

Standard emulatoren «Nexus S - API 19» vil ikke uten videre fungere med Google Maps funksjonaliteten til applikasjonen, så mye så at den ikke engang lar seg installere. Løsningen på dette vil være å sette opp en ny emulator. Under utviklingen av applikasjonen har jeg hatt gode erfaringer med «Nexus 5 - API 21 (Google APIs)». Alle skjermbildene i dette dokumentet er tatt med nevnte oppsett. Nexus 5 emulatoren har jeg velykket kjørt på både en kraftig stasjonær arbeidsstasjon, såvel som en svakere bærbar datamaskin fra 2012. Det bør med andre ord ikke by på store utfordringer. Kravet for dette er såklart at «x86» eller «x86_64» bildene velges, og ikke «ARM» bildene.

1.2 API nøkkelen

Siden API nøkkelen som er satt i AndroidManifest.xml¹ ikke lar seg overføre grunnet tilhørigheten til den lokale `~/.android/debug.keystore` filen, bør en ny nøkkel genereres. Alternativt kan en laste opp den ferdigkomplilerte pakken `ruteravvik.1.0.apk` til emulator instansen.

- Generere en ny Google Maps Api nøkkel, og legge den inn i `AndroidManifest.xml`²
- Eller; installere vedlagt apk pakke:

```
$ adb install app/app-release.apk
```

¹ `<meta-data android:name="com.google.android.maps.v2.API_KEY">`

² pakke strukturen er `<net.plastboks.s114143.ruteravvik>`

Del II

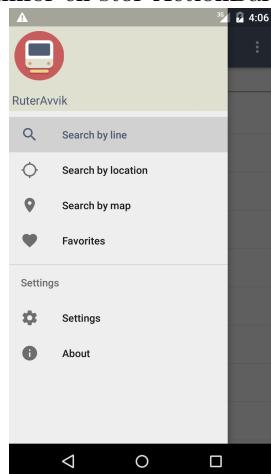
Brukermanual

Under er applikasjonen beskrevet funksjon for funksjon med skjermbilder. Alle skjermbildene er tatt med engelsk språk aktivert på enheten.

Når applikasjonen starter for første gang vil den gjøre to kall på «Ruters Reise Api». Det første er «GetRuterStops» og det andre er «GetLines». Dataene fra disse kallene lagres så i et «mellomlagringssystem» slik at de er tilgjengelig raskere for brukeren neste gang det er behov for dem. Grunnen til dette er at begge kallene returnerer datasett på over 1500 objekter. Denne lastingen av data vises på en «Load Screen», men med rask internett kan denne load screenen forsvinne raskt.

2 Navigasjonsskuff

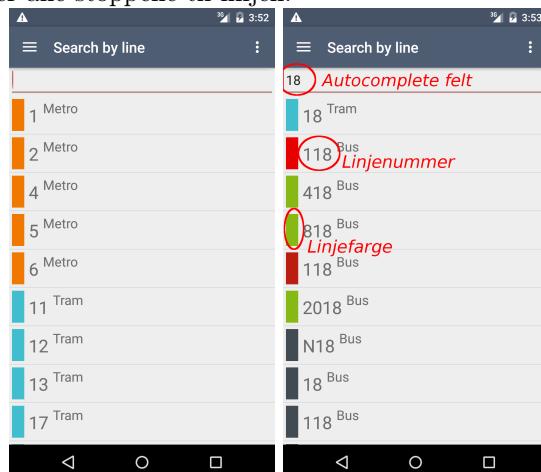
Navigasjonsskuffen kan aktiveres ved å dra den inn fra venstre, eller trykke på ikonet til venstre i actionbaren. Dette er hovednavigasjonsmetoden til applikasjonene. Den er valgt framfor en stor ActionBarMenu.



3 Søk

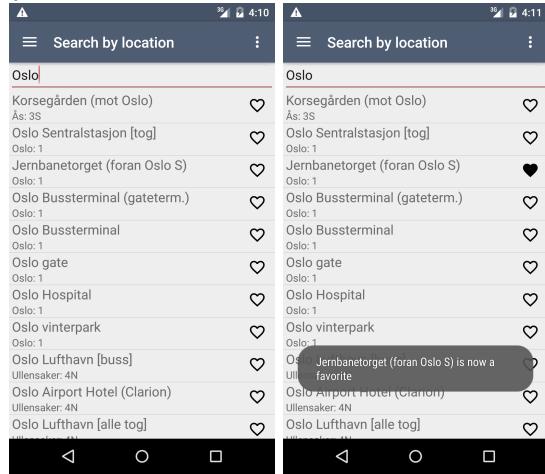
3.1 Linjenummer

Søkesiden for linjenummer er standard landingside for applikasjonen (dette kan endres i «Instillinger»). Her kan brukeren søke etter ønsket linjenummer (f.eks 18 trikken som vist under). Alle elementene i denne listen er mellomlagret, og hentes ikke direkte fra Ruter hver gang de benyttes. Alle elementene i listen kan trykkes på for å føre brukeren til en visning over alle stoppene til linjen.



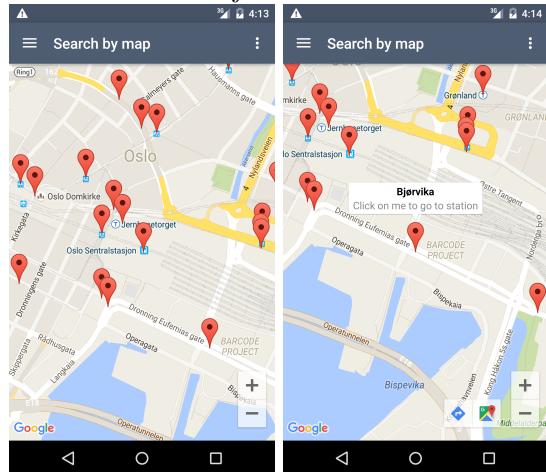
3.2 Lokasjon

Lokasjonssøket er et søk i Ruters «GetRuterStops» database, elementene i dette søker etter «tittel» til elementene. I denne listen kan hjertene trykkes for å sette lokasjonen som favoritt, og elementene kan velges for å føre brukeren til informasjon over alle linjene som går fra den stasjonen nå.



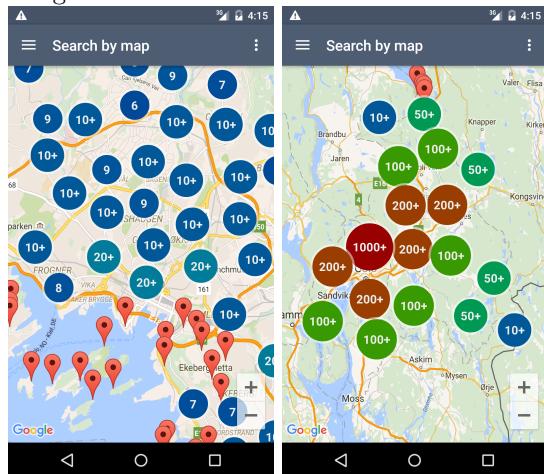
3.3 Kart

Det er mulig å søke i kart ved å navigere rundt i et Google Maps. Det er ikke implementert noe tekstsøk i dette fragmentet, og standardlokasjonen er Oslo sentrum. Alle markørene kan trykkes på for å vise stasjonsnavn. Infovinduet som så dukker opp kan videre trykkes på for å føre brukeren til stasjonen.



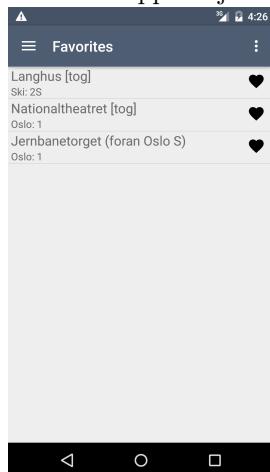
3.3.1 ClusterManager

Når brukeren zoomer ut vil nærliggende markører gruppес sammen i «clustere» for å lette rendringsarbeidet til enheten.



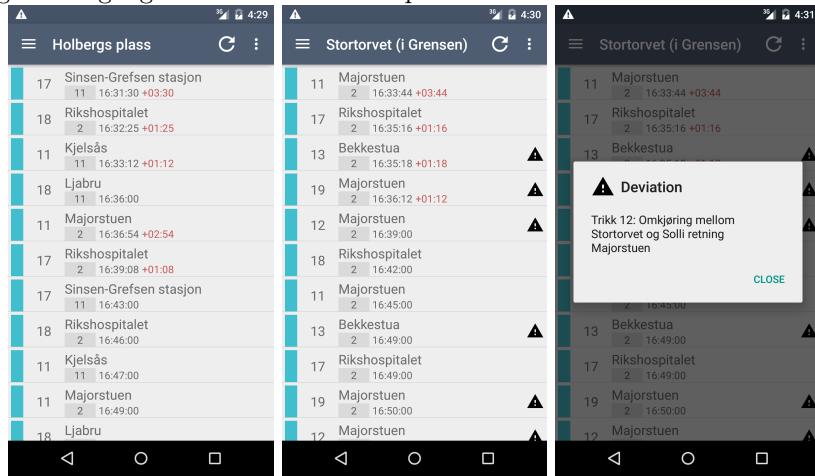
4 Favoritter

Favorittene i applikasjonen kan enkelt vises ved å navigere til «Favoritter»



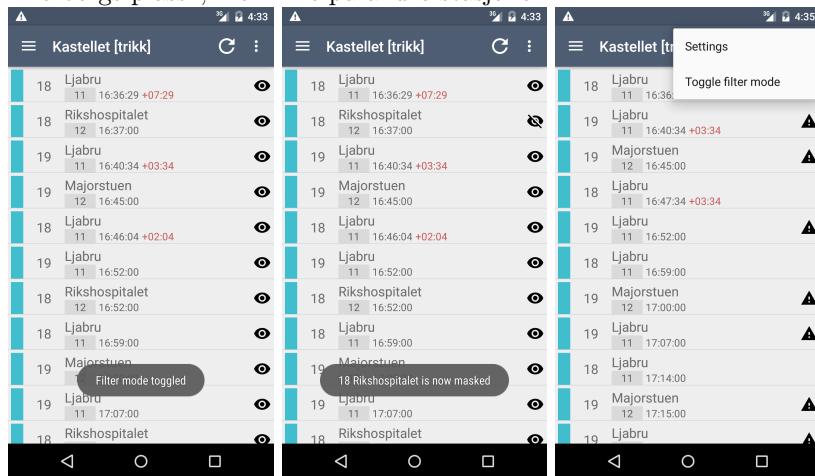
5 Stasjonsvisning

Stasjonvisningen inneholder en del data om de forskjellige linjene som stopper på stasjonen. Fra høyre mot venstre er; linjens farge tegnet opp, deretter er linjenummeret vist sentrert. I den grå boksen under linjetittelen er platformnummeret, og til høyre for den er forventet avgangstid samt eventuell forsinkelse i rødt. Alle dataene i denne visningen er sortert på stigende avgangstid fra nåværende tidspunkt.



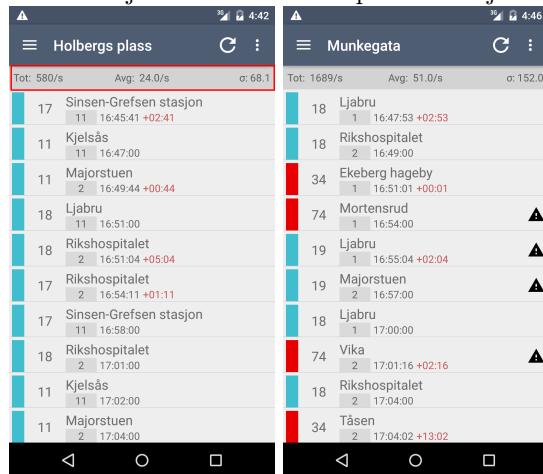
5.1 Stasjonsfilter

Enkelte holdeplasser/stasjoner har veldig mange forskjellige linjer og platformer. For en pendler som til daglig ønsker å skjekke en slik holdeplass kan det fort bli mye uninteressant data. Et filter er derfor implementert for å sjule linjer som er uinteressante. Dette filteret er holdeplassbestemt, og vil dermed ikke påvirke andre stasjoner. En kan f.eks sjule 18 trikken på «Holbergs plass», men ikke på andre stasjoner.



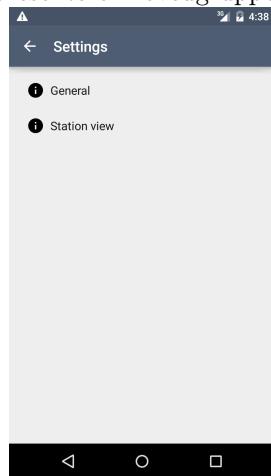
5.2 Stasjonsstatistikk

Om stasjonsstatistikk er valgt i «Instillinger», vil et ekstra grått felt over listen dukke opp, med informasjon om forsikelsene på den stasjonen i det datasettet.



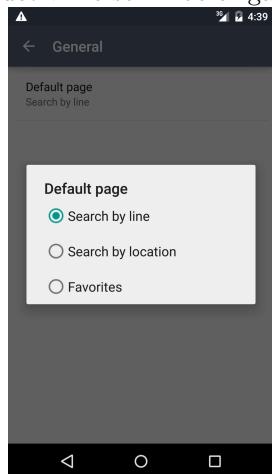
6 Instillinger

Det er benyttet en tolagstopologi for instillinger funksjonalitetene til applikasjonen. Denne topologien er tiltenkt for ekspansjon med flere instillinger (videre utvikling). Det første laget representerer hovedgruppene.



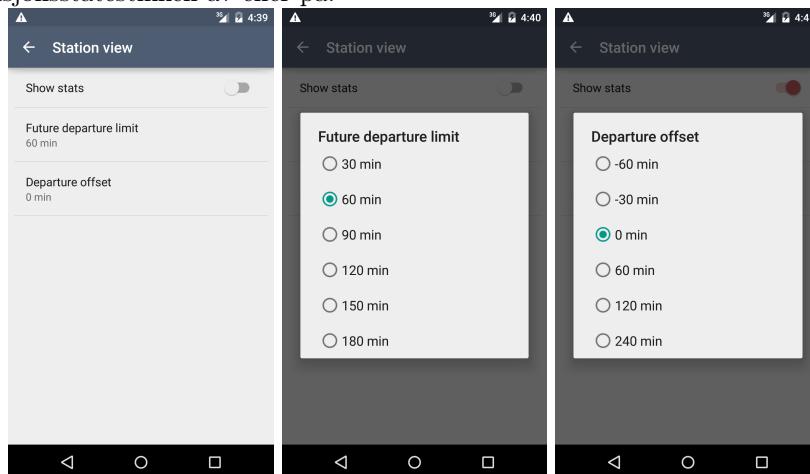
6.1 Generelt

Her kan landingssiden settes for applikasjonen. Det er denne siden som vil vises når applikasjonen startes. Kartvisningen er tatt ut av denne listen da den er såpass ressurskrevende at det virke som noe er galt når en starter applikasjonen med den som landingsside.



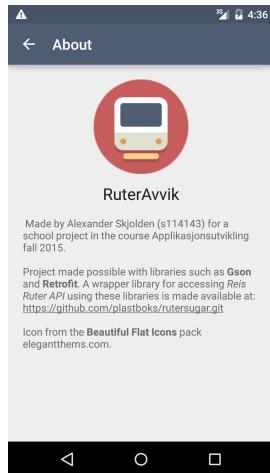
6.2 Stasjonsvisning

Her kan brukeren sette et «fram i tid / tidsbegrensning» filter. Ruters Reise Api leverer vanligvis ut 3-4 timer med data fram i tid. Dette filteret er ment for å begrense datamengden som blir presentert i stasjonsvisningen. Brukeren kan også sette et «offset» for å manipulere datasettet fram og bakover i tid. Dette filteret samkjører med «fram i tid» filteret, slik at man f.eks kan se data to timer fram i tid, og deretter kun i 30 minutter. Her settes også stasjonsstatistikken av eller på.



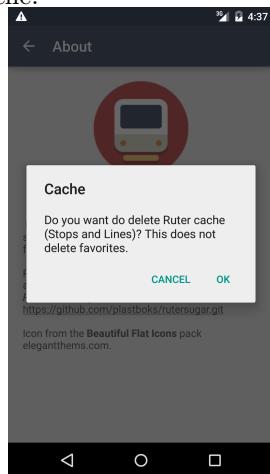
7 Om

En enkel om side som beskriver diverse hjelpebiblioteker og ikoner.



7.1 Mellomlagring

Det er implementert et lite «påskeegg» i applikasjonen. Siden dataene fra Ruter mellomlagres (så langt det lar seg gjøre), og siden disse dataene ikke har noe rutine for å oppdatere seg selv, er det implementert en manuell løsning for å slette de mellomlagrede dataene. Ved å trykke fem ganger på ikonet vil en dialogboks dukke opp og spørre om en ønsker å tømme cache.



8 SwipeRefreshLayout

Sider som ikke har et søkefelt (søke sidene) kan oppdateres via å «dra» ned listen til et «loader» ikon dukker opp. Alternativt kan oppdaterings ikonet i actionbaren benyttes for samme funksjonalitet. Mer om SwipeRefreshLayout her: <https://www.google.com/design/spec/patterns/swipe-to-refresh.html#swipe-to-refresh-swipe-to-refresh>.

9 Autocomplete

Søkesidene «Linjenummer» og «Lokasjonssøk» benytter er seg av «Autocomplete» funksjonalitet. Det betyr at listen dynamisk oppdateres så fort brukeren har tastet inn over to karakterer i søkefeltet. Søkefunksjonen fungerer som en «inneholder» metode (type ‘like %string%‘ sql sok). Mer om dette her: <https://www.google.com/design/spec/components/text-fields.html#text-fields-auto-complete-text-field>.

Del III

Bruksmønster

Under utviklingen av applikasjonen har bruksmønsteret sklidd litt til siden av hva som var opprinnelig tiltenkt. Applikasjonen er nå blitt mer generell i den forstand at den kan brukes som et oppslagsverk for bruk ved behov, og som daglig oversikt over forsinkelser m.m

10 Daglig bruk

Til daglig bruk ser jeg for meg at «Favorittsiden» vil være brukt mest som en veiviser til de lokasjonene man ønsker. Videre vil dataene pr lokasjon kunne gi en rask og enkel tilbakemelding over forsinkelser m.m. Dermed vil applikasjonen ikke kreve mer enn ett intraksjonsklikk fra brukeren (gitt at favoritter er satt til standard landingsside).

11 Engang reise

Kartdelen, og Linjeoversiktssidene kan bedre benyttes som oppslagsverk og oversikt over nye reisemål. Tanken bak disse visningene har vært å skape et brukervennlig overblikk og raskt kunne se hvor diverse linjer stopper, og hvor enkelte stop er.

Del IV

Teknisk del

12 Ruter Reis API

Som datakilde til applikasjonen ble Ruters Reise API valgt. I forprosjektrapporten skrev jeg om «Dit.no» som et alternativ, men det ble raskt valgt bort. Jernbaneverket har og et data API som kunne vært aktuelt, men dette APIet gir kun informasjon om togstidene. For å gi applikasjonen en bedre brukeropplevelse kunne det vært interessant å samkjørt disse APIene (Ruter Reise og Jernbaneverket), men det ga tidsrammene ikke mulighet for.

12.1 Dokumentasjon

All dokumentasjon for Ruter Reise API har vært å finne på denne siden: <http://reisapi.ruter.no/help>. Ut i fra den siden har jeg manuelt bygget opp klasser som matcher for forskjellige typene Ruter opererer med («Place», «Stop», «MonitoredStopVisit», m.m). Se RuterSugar under.

12.2 Utfordringer

Under utviklingen av applikasjonen har jeg til stadighet opplevd nedetid på Reise APIet. Dette har vært nedetid fra få minutter til lengre tider. Utfordringen har da vært å få applikasjonen til å oppføre seg forutsigbart, og ikke minst unngå å «kræsje». Jeg tror jeg har klart å «hoppe bukk» over de fleste av disse problemene, men slik feilsøking er svært tidkrevende og vanskelig å få til skikkelig.

13 RuterSugar

I forbindelse med forprosjektet til mappe 3 valgte jeg å utviklet et selvstendig (wrapper) bibliotek for å gjøre kommunikasjonen med Ruters Reise API så strømlinjeformet og uforstyrrende som mulig. I dette biblioteket valgte jeg å benytte meg av «Gson» og «Retrofit» som avhengigheter. Biblioteket er åpent og tilgjengelig på <https://github.com/plastboks/rutersugar> og kan pakkes ned enkelt med ‘gradle build’. Dette skaper en jar fil som er lagt ved i «app/libs» i android prosjektet. Hele kildekoden ligger og med som en referanse, men det er jar filen som brukes i classpathen under kompilering av applikasjonen.

RuterSugar er bygget opp med tanke om å dekke mesteparten av funksjonaliteten til Ruters Reise API. Dette innebærer å dekke funksjonalitet som «RuterAvvik» ikke benytter.

13.1 Gson

Json parser utviklet av Google: <https://github.com/google/gson>

13.2 Retrofit

Http client for android og java: <https://square.github.io/retrofit/>

13.2.1 Callback vs AsyncTask

Under utviklingen og testing av applikasjonen ble det fort oppdaget at Ruters Reise API til tider kan være nede. For å gi en optimal brukeropplevelse må disse nedetidene presenteres og gi en forutsigbar tilbakemelding. Den store utfordringen med AsyncTask og direkte forespørslar til Ruter Reise Apie i den forbindelse var nettopp å kunne fange disse nedetidene på en ryddig måte. En typisk try/catch blokk kunne nok ha gjort jobben, men kodebasen ville blitt unødvendig kompleks og foreurenset. En nedetid hos Ruters Reise Api trenger nødvendigvis ikke være en 408 (Request Timeout), men kan godt være en 404, 500 osv error. Retrofit's Callback grensesnitt syr dette sammen på en smidig og pen måte, og gjør at hele AsyncTask objektet kan omgåes.

Begge eksemplene under oppnår samme resultat (thread'er ut en jobb, og kaller på «update()» når den er ferdig), men Callback eksempelet er vesentlig mer fleksibelt.

13.2.2 AsyncTask eksempel:

```
private class FetchStops extends AsyncTask<Object, Object, Object> {
    @Override
    protected void onPostExecute(Object o) {
        super.onPostExecute(o);
        update();
    }

    @Override
    protected Object doInBackground(Object[] params) {
        stops = MainActivity.ruter.fetch().getStopsRuter();
    }
}
```

13.2.3 Callback eksempel:

```
private void fetchStops() {
    MainActivity.ruter.callback()
        .getStopsRuter(new Callback<List<Stop>>() {
            @Override
            public void success(List<Stop> s, Response response) {
                stops = s;
                update();
            }

            @Override
            public void failure(RetrofitError error) {
                // Send ut en toast f.eks.
            }
        });
}
```

14 ClusterManager

Siden det eksisterer ca to tusen stop i ruter sin database på østlandet, ble det en betydelig utfordring å tegne alle på kartet samtidig. I diverse forsøk på dette ville telefonen eller emulatoren stoppe fullstendig opp til tider. Etter litt research dukket en løsning opp fra Google selv; ClusterManager³. Dette biblioteket ble lastet ned og lagt til applikasjonen. ClusterManager er heller ikke en perfekt løsning, da det er veldig ressurskrevende å tegne om, og «clustre» sammen markører ettersom brukeren zoomer ut, og splitte de opp når brukeren zoomer inn.

Med tanke på tidsrammene til prosjektet falt valget på å beholde ClusterManager, da alternativet ville være å droppe Maps funksjonaliteten fullstendig.

15 Feature Creep

En av de vanskelige utfordringene med utvikling av en applikasjon som dette er å sette ned foten for hva en har lyst til å gjøre, og hva en kan gjøre. Siden jeg har utviklet denne applikasjonen til personlig bruk, og siden jeg har brukt applikasjonen daglig den siste måneden, har mange ønsker og tanker dukket opp om hvordan den kan forbedres. Mange av disse funksjonene har blitt implementert (som: stasjonsfiltere, statistikk, tidsberesning m.m), men mange har dessverre og måtte bli utelatt.

Eksempler på dette er:

- Tekstsøk i kart.
- Hvor er jeg i kart (via GPS og eller GSM)
- Notifikasjoner basert på tidspunkt og favoritter (melding om forsinkeler m.m).
- Widgets for utlisting av kritisk informasjon, som f.eks store forsinkelser på gitt reise rute m.m.
- Lagring av data over tid for å sammenligne forsinkelsesgrad med historiske forsinkelser (f.eks når var siste store togstans på Oslo S).
- Mer ikoner for å visuelt skille bus, trikk, tog, båt m.m.
- Integrasjon av Jernbaneverket sine data for mer presise togdata (og for større dekningsradius over østlandet/Norge).

Del V Koden

16 Fragmenter

Til denne oppgaven har jeg valgt å bruke fragmenter for mesteparten av visningssidene. Nesten alle visningene kjøres fra MainActivity.java som også er hovedaktiviteten til applikasjonen. På den måten vil data kunne møtes i MainActivity.java og sendes videre som i et Mediator pattern. Skulle applikasjonene bygges større kan det være at dette designet ikke skalerer godt nok opp. Måten «EventCallback» designet til alle fragmentene mot MainActivity.java er implementert er en del av Goole retningslinjer⁴.

17 Navigasjonsbaren

Navigasjonsbaren som står for presentasjon av de sentrale navigasjonspunktene er i seg selv et fragment. Dette designet er et velkjent og anbefalt design fra Googles Android⁵ team i samspill med Material Design⁶.

³<https://developers.google.com/maps/articles/toomanymarkers#distancebasedclustering>

⁴<https://developer.android.com/guide/components/fragments.html#EventCallbacks>

⁵<https://www.google.com/design/spec/patterns/navigation-drawer.html>

⁶<https://www.google.com/design/spec/material-design/introduction.html>

18 Adapterene

Alle listefragmentene bruker spesiallagede adaptere for visning av data. Dette for å kunne skape helt egne visninger som oppfylte de ønskene jeg måtte ha, og som standard adapterene ikke kunne gi. Alle adapterene er samlet i «adapter» pakken, og deres stilark er samlet i «res/layout/list_item_*».

19 Datalagring

Datalagring til applikasjonen er gjort alene igjennom «SharedPreferences» biblioteket til Android. En SQLite database hadde kanskje kunne vært litt mer spennende, men hadde ikke gitt noen praktiske fordeler framfor SharedPreferences. Siden SharedPreferences ikke har støtte for komplekse objekter, er Gson (som nevnt tidligere en avhengighet til RuterSugar) benyttet for å encode dataobjektene ned til JSON strenger ved lagring, og dekode de tilbake til Java objekter ved uthenting.

19.1 Settings

I denne klassen «Settings.java» lagres små informasjon som favoritter og filtermasker (stasjonsfilter) m.m.

19.2 PersistentCache

Dette er lagringsklassen til Ruter dataene som blir mellomlagret.

20 Helper pakken

20.1 UTM vs Lat/Lng

En utfordring jeg støtte på med dataene fra Ruter var deres valg av koordinatprosjeksjon: UTM⁷. Som kjent bruker Google Maps et «Geografisk koordinat system» (Lat / Lng)⁸ og det skulle vise seg at det å konvertere mellom disse systemene ikke nødvendigvis var enkelt. Etter endel research på området falt valget på å «låne» IBMs kode (hvis lisens ikke var veldig tydelig). Klasse «CoordinateConversion.java» er en privat klasse i hjelpepakken og er klassen som er kopiert fra IBMs nettsider. Klassen «Coordinates.java» står foran som et abstraksjonslag for å forenkle koden ute i adapterene.

21 Cluster pakken

21.1 CustomClusterRenderer

Som en liten bakside av ClusterManager⁹ integrasjonen gjort i kartfragmentet, var kompleksiteten rundt det å tilpasse markørene. Måten å gjøre det på var å skrive en ClusterRenderer klasse som tok hånd om de ønskede endringene.

⁷https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system

⁸https://en.wikipedia.org/wiki/Geographic_coordinate_system

⁹<https://developers.google.com/maps/documentation/android-api/utility/marker-clustering>