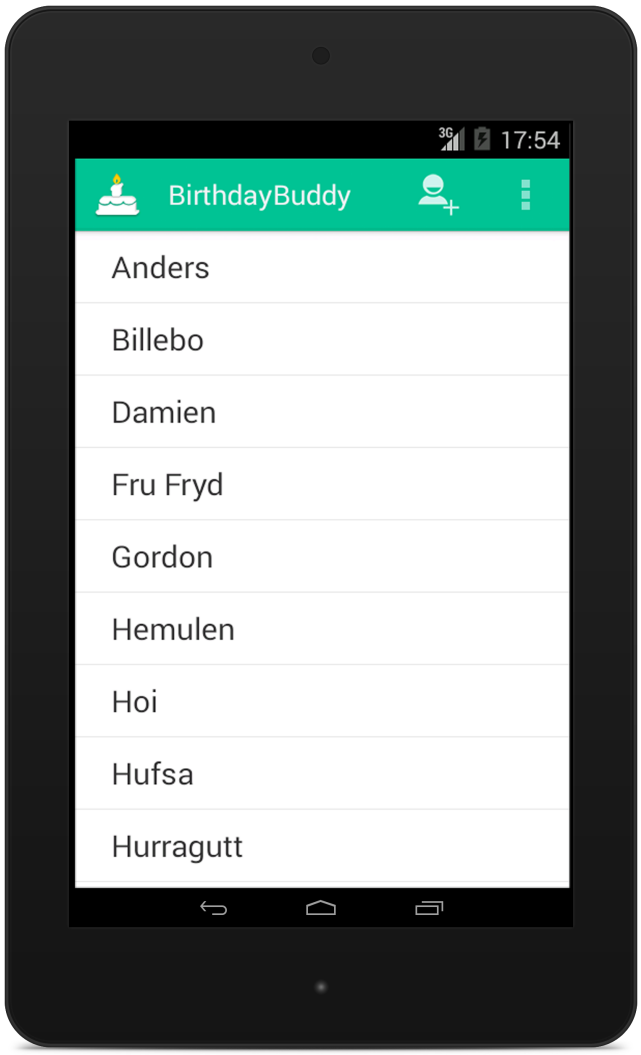
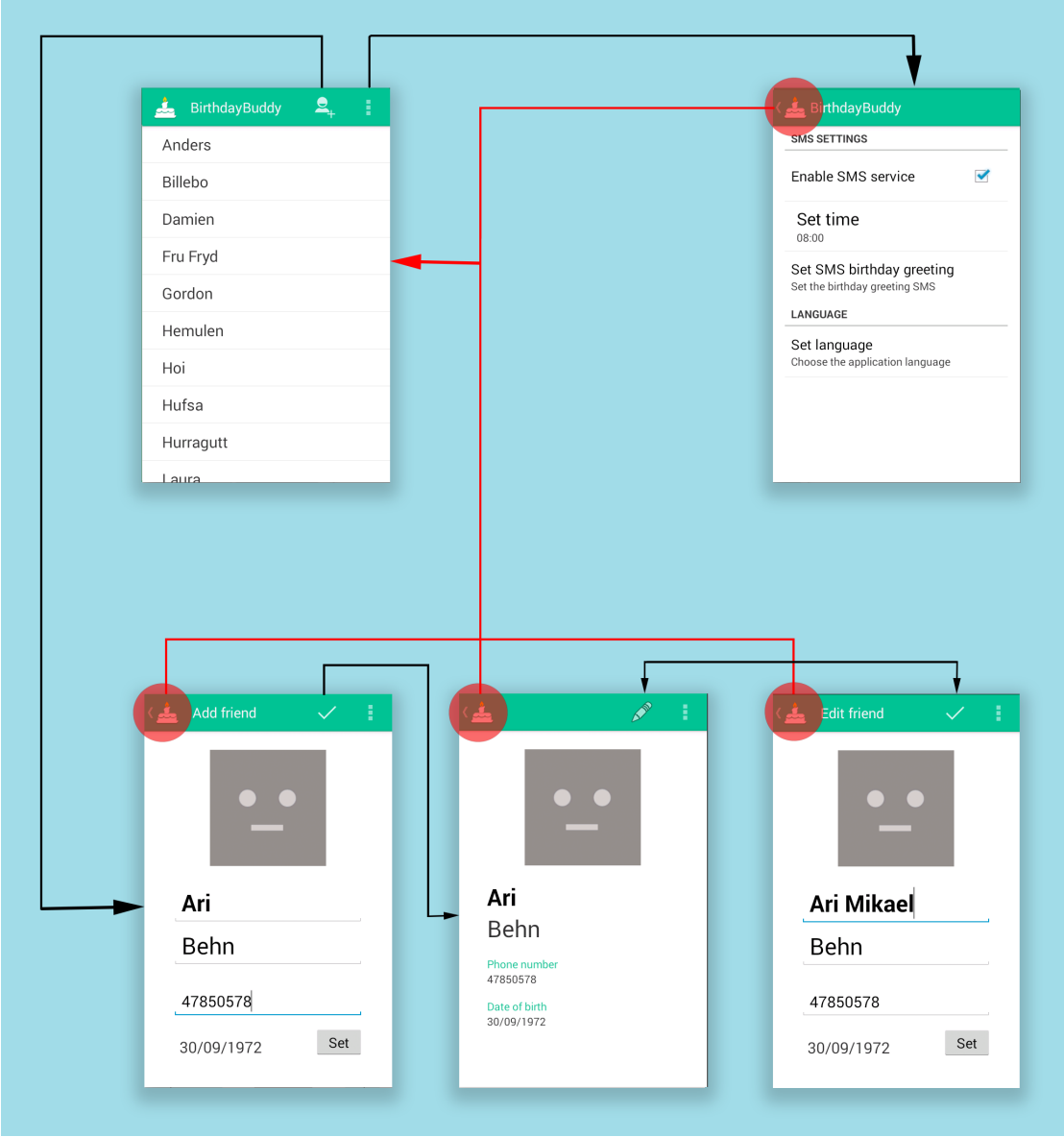
APPLIKASJONSUTVIKLING

Andre mappeinnlevering

Martine Eklund S156960 / Audun Karlsrud Larsen S172860





Illustrasjon 1: Navigasjonsflyt for funksjonaliteten ”Add friend” samt innstillinger. Sorte piler viser flyt ”fremover”, mens rødt er tilbake-navigering. Ved trykk på en oppføring i lista kommer man til skjermbilde nederst i midten (detaljvisning av venn). En mer inngående beskrivelse av strukturen og de ulike klassene finnes på s. 5-6, under ”Struktur”.

Designprosess og antagelser

Basert på erfaring fra arbeidet med mappeinnlevering 1, visste vi at dette kom til å bli en tidskrevende prosess. Fagområdet og introduksjonen til de nye funksjonene gjorde at vi valgte å arbeide sammen, for å opparbeide oss en bredere forståelse og muligheten til å utvikle en mer komplett applikasjon.

Første steg i prosessen var å kartlegge hvordan lignende applikasjoner hadde utviklet sitt design, i henhold til retningslinjene satt av *Android Developer* sin offisielle side. Vi arbeidet også med å kartlegge våre telefoners kontakt applikasjoner for å se etter gjenkjennbare elementer og konstante navigasjonsmønstre. Resultatet av disse undersøkelsene var et klart bilde av hvordan vi ønsket at navigasjonen i vår applikasjon skulle foregå. Vi vil komme tilbake til dette i den videre beskrivelsen.

Grunnleggende funksjonalitet

Applikasjonen støtter følgende funksjonalitet:

- Lar brukeren legge til venner og registrere navn, mobilnummer og fødselsdag for vedkommende.
- Visning av alle venner via ListFragment i en sortert liste
- Visning av detaljinformasjon om en venn, ved å trykke på vedkommende i listen. Dette fragmentet vises også umiddelbart etter at man har opprettet en venn.
- Mulighet til å redigere informasjon om en venn
    a) Umiddelbart etter at vedkommende er registrert, via ”edit”-symbol på menylinjen
    b) Ved å trykke på en venn i listen, slik at man får opp detaljvindu om vedkommende. Menyen i detalj-vinduet inneholder ”edit”-symbol som ved klikk tar deg videre til redigeringsvinduet.
- Mulighet til å slette en venn
    a) I detaljvisningen: Velg ”Delete friend” fra innstillingsmenyen.
    b) I ListViewet: Trykk ”lenge” (longClick) på vennen for å få opp en dialog som spør om du vil slette vedkommende
- Validering av input ved hjelp av java og standardiserte Android-løsninger
    - Tidspunkt og dato-funksjonalitet via time/date-pickers
- Bursdag-SMS-funksjonalitet. En bursdagshilsen sendes per SMS til alle venner som har bursdag. Koden kjører på et gitt tidspunkt hver dag.
- Notifikasjon gis til brukeren når SMS-hilsener er sendt ut.
- Tilgang til eget skjermbilde for innstillinger via menyen
    - Mulighet for å slå av og på bursdags-SMS funksjonaliteten.
    - Mulighet til å skrive en egendefinert bursdagshilsen.
    - Mulighet for å endre tidspunktet for SMS-utsendingen.
    - Støtte for valg av språk. Man kan velge mellom norsk og engelsk språk.
- Widget

    - Viser dagens dato og vennnene som har bursdag. Oppdateres når man klikker på den, og ellers hver halvtime.
- Mulighet til å avslutte applikasjonen via menyen fra ethvert skjermbilde.
- Lagring av data i SQLite database.
- Bruk av Content Provider.

Design

Vi utviklet våre designelementer basert på designprinsipper fra *Android Developer*, samt på bakgrunn av Android-grensesnitt vi har brukt selv og egne preferanser.

Kjente, fleksible navigasjonsmønstre

Vi har som et gjennomgående element i vår applikasjon valgt å implementere en navigasjonsmeny. Denne menyen opererer som en konstant, og er tilstede på alle våre skjermbilder. Dette er et bevisst valg for å gi brukere en god navigasjonsflyt, slik at man enkelt kan navigere mellom skjermbildene. Man kan argumentere for at dette passer med design-prinsippene ”Simplify My Life” og ” "I should always know where I am", siden denne plasseringen av en tilbakeknapp passer inn et i et allerede kjent navigasjonsmønster for brukeren. Slik minsker man risikoen for feilnavigasjon og frustrasjon.

Vi har konsekvent valgt å gi brukeren en mulighet til å navigere tilbake til forrige skjermbilde via app-ikonet til venstre på menyen, slik konvensjonen for Android-applikasjoner er.

Navigasjonsmenyen inneholder også et *Options* element, som alltid ligger lengst til høyre. Symbolet er representert i *Android Developer* ånd med tre vertikale rundinger. Denne menyen får man tilgang til fra hovedskjermbildet, og ved trykk kommer det frem en drop-down meny som gir deg en mulighet til å gå inn på innstillinger, men også muligheten til å avslutte applikasjonen fra ethvert skjermbilde. Vi har valgt å legge til denne funksjonaliteten her da vi mener det er god konvensjon og kunne avslutte applikasjonen uavhengig av hvor man måtte befinne seg i applikasjonens livssyklus.

Videre er all funksjonalitet knyttet til skjermbildet du er i, eksempelvis funksjonalitet for å lagre eller endre en venn, alltid plassert øverst til høyre i menyen, slik at man har denne funksjonaliteten godt synlig og lett tilgjengelig. Dette er et gjennomgående navigasjonsvalg i vår applikasjon.

Annen funksjonalitet har vi bevisst gjort mindre tilgjengelig som funksjonaliteten for å slette en venn. På Android-enheter vi er kjente med har

dette ligget gjemt i dropdown-menyen, slik at man ikke trykker på den ved et uhell, slik man kunne gjort om den lå ved siden av lagre-knappen.

Vi har valgt å følge prinsippet «Only show what I need when I need it» i vår applikasjon overføres dette til at brukere kan få for mye informasjon på en gang, vi har derfor implementert nødvendig informasjon på de respektive skjermbildene, og fjerner funksjonalitet som ikke er nødvendig for de respektive skjermbildene brukeren aksessere. Eksempelvis mener vi det er rotete om man kan endre SMS-tekst fra ”legg til ny venn”-skjermbildet, siden disse funksjonene ikke er direkte knyttet opp mot hverandre, så dette har vi forbeholdt hoved-skjermbildet (venne-listen).

Som en videreføring av prinsippet ”Simplify My Life” har vi fokusert på å gjøre grensesnittet fleksibelt og forhåpentligvis intuitivt. Eksempelvis får man mulighet til å redigere en venn umiddelbart etter at man har lagt til personen, uten at man er nødt til å gå tilbake til hovedskjermbildet (listen) og trykke på vedkommende der. Vi har valgt å bruke en knapp for lagring og en knapp for å gå tilbake, slik at man kan avbryte endringsprosessen dersom man ombestemmer seg etter å ha trykket på ”edit”-knappen.

Vi hadde tidlig et ønske om å gjøre navigasjonen enda mer effektiv ved å implementere ”onLongClickListener” for listViewet, slik at man kunne velge å slette en person direkte fra lista ved å klikke ”lenge” på vedkommende. Dette har vi lykkes med å implementere, men brukeren vil oppleve at applikasjonen blinker et sekund når man velger sletting, siden aktiviteten restarter for å få oppdatert listevisningen. Dette kunne sannsynligvis vært løst mer elegant rent kodemessig om vi hadde hatt mer tid.

Gjenkjennbarhet

Applikasjonen har implementert et minimalistisk design, for å holde funksjonaliteten til applikasjonen i fokus. Det er derfor viktig at de designelementene som vi har implementert er intuitive og lette å bruke. For å løse dette har vi valgt en høy grad av gjennbarhet på alle vår symboler. De er hentet fra *Android Developer* sine sider, og er synonymt med navigasjon i lignende applikasjoner. Dette valget er tatt å øke brukere sin mulighet til å gjenkjenne elementer og forstå funksjonaliteten til de respektive knappene.

Ikoner og fargevalg

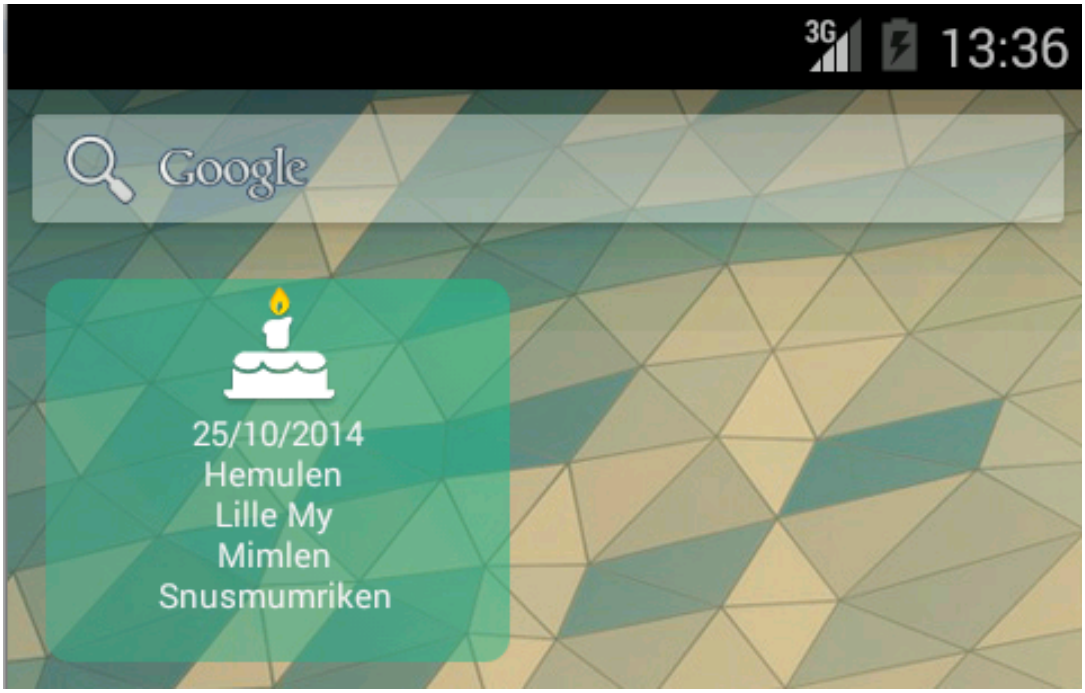
Ifølge design-prinsippene er ikoner og objekter mer brukervennlige enn knapper med tekst, så vi har basert oss på å bruke kjente ikoner fra Android sin sdk-pakke, slik at brukeren skal gjenkjenne funksjonaliteten bak disse.

Android skriver på sine utviklersider at ”launcher icons” bør være tredimensjonale ”with a slight perspective as if viewed from abowe, so that users perceive some depth”. Videre skriver de at ikonet bør ha en distinkt silhuett og at man burde unngå firkant- og rundingformer og heller gå for en unik form. Dette har vi forsøkt å etterfølge og ikonet vårt ser altså slik ut:

Illustrasjon 2: Launcher icon utformet med Androids føringer i tankene.

Fargevalgene for applikasjonen for øvrig er basert på et ønske om å skape et design som appellerer til det estetiske hos applikasjonens brukere. Fargevalgene våre er tatt på bakgrunn av et ønske om å skape en ”glad” følelse når man bruker applikasjonen. Grønn er en behagelig og ”ren” farge som ikke brukes av så mange applikasjoner, og vår applikasjon kan sånn sett gjøre seg litt bemerket. Fargene vi har valgt komplimentere hverandre og er behagelige å se på, og danner et gjennomgående tema i vår applikasjonen.

Vi har valgt å videreføre applikasjonens farger i widget-utformingen, slik at det skal være tydelig at widgeten tilhører nettopp vår applikasjon.



Illustrasjon 3: Widget med samme fargebruk som den øvrige applikasjonen.

Struktur

Konvensjoner

Siden vi er ferske innen Android-programmering er det sikkert en flere konvensjoner vi burde fulgt som vi ikke vet om. Det er mange små detaljer, som for eksempel at nøkkelen man sender med en ny Intent bør være en public constant med pakke-navnet som prefix.

Struktur

Vi har forsøkt å holde følgende struktur i vår applikasjon:

1) En hovedklasse, **MainActivity**, som automatisk starter opp ”hovedskjermbildet” vårt, i form av fragmentet FriendListFragment, som viser listen av venner. Det eneste hovedklassen har ansvaret for i tillegg til å vise dette fragmentet, er å starte opp en barne-aktivitet, **FriendDetailActivity** dersom man trykker på en av vennene i listen i liste-fragmentet eller å starte opp barne-aktiviteten om man trykker på ”legg til ny venn”-ikonet i menyen. Klassen sender med en tekststreng til intenten, slik at FriendDetailActivity vet hvilket fragment den skal vise.

2) Barneaktiviteten til **MainActivity** er **FriendDetailActivity**. Denne aktiviteten har ansvaret for å behandle de ulike fragmentene som viser detaljer om en venn:
    a. **AddFriendFragment:** viser et view med input-felter slik at man kan skrive inn informasjon om en ny venn. Menyen i dette fragmentet består av en tilbakeknapp til hovedaktiviteten (som viser liste-fragmentet) og en ”lagre”-knapp.
    b. **ShowFriendDetailsFragment:** viser et view med tekstfelt som inneholder informasjonen til vennen vi nettopp opprettet. Menyen i dette fragmentet består av en tilbakeknapp til hovedaktiviteten og en ”rediger”-knapp.
    c. **EditFriendFragment:** viser et view med input-felter hvor man kan endre informasjon til vennen man nettopp så på i **showFriendDetailsFragment**. Har i likhet med **addFriendFragment** en ”lagre”-knapp.

    Vi har latt fragmentene ta seg av representasjonen av skjermbildet, mens aktiviteten fragmentene ble startet fra tar seg av logikken. Fragmentene gjør dermed ikke stort annet enn å kalle på passende metoder i aktiviteten når man interagerer med de ulike skjermbilde-elementene. Aktiviteten er ansvarlig for å passe på venneobjektet som er blitt opprettet eller trykket på i listen, og kalle på objektets valideringsmetoder eller innsetting/oppdatering-metoder når dette er nødvendig.
    Aktiviteten er videre ansvarlig for å ta imot eventuelle feilmeldinger som returneres fra venne-objektets valideringstester og vise disse til brukeren.

3) Klassen **Friend** er java-representasjonen av en venn, med tilhørende variabler. I objekt-orientert ånd, og fordi vi mener dette gir en mer fleksibel, gjenbrukbar kode, har vi overlatt til venne-objektet selv å kontrollere de variablene man forsøker å sette. Venne-objektet har også metoder for å sende variablene sine til databasen via **contentProvider-klassen.**

4) Klasser for tilpasning og visning av venne-objektene i listen:
FriendListLoader henter data asynkront til listViewet vårt.

CustomArrayAdapter er en tilpasning av ArrayAdapter som lar oss populere listViewet med venne-objekter.

5) Klasser for database-behandling: **FriendsContentProvider** er en klasse som den håndterer tilgangen til strukturert data, i vårt tilfellet er dette vår database, og gjør det mulig å få tilgang på vår databasestruktur ved hjelp av URI. **FriendsDB** er database klassen vår som inneholder selve database strukturen, samt metoder for å hente ut objekter av databasen.

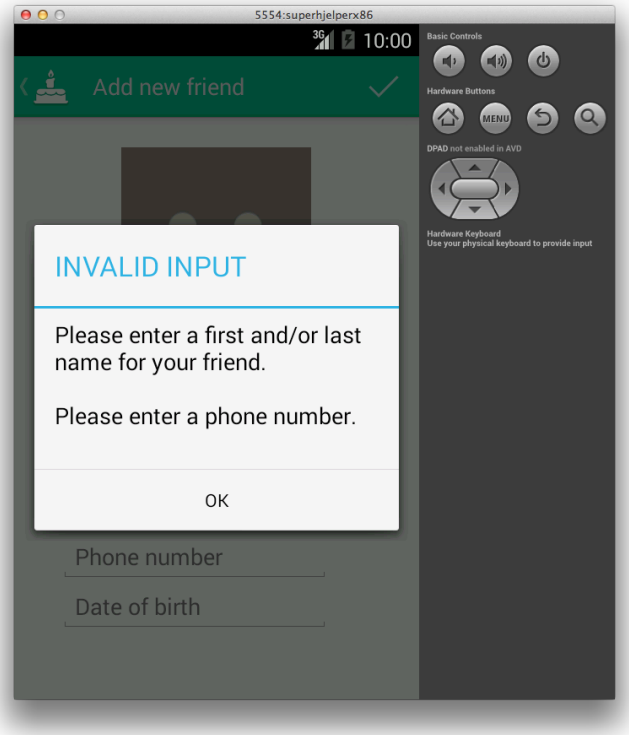
6) Klassen **SettingsActivity** er en utvidelse av Android sin egen PrefernceActivity-klasse. Denne gir direkte og automatisk adgang til å lagre innstillinger brukeren gjør i SharedPreferences. Klassen har også metoder for å lytte på endringer i SharedPreferences. Den baserer seg på en individuell xml-fil plassert i en egen XML folder. **SettingsActivity** sitt grafiske oppsett er synonymt med innstillinger i hoved-andelen av allerede eksisterende Android applikasjoner, og passer dermed godt med det allerede diskuterte prinsippet ”gjenkjennbarhet”. **SettingsActivity** klassen gir brukeren mulighet til å aktivere og deaktivtere SMS-tjenesten, legge inn egendefinert tekst for bursdagsmeldinger samt sette et tidspunkt for utsending av SMS.

7) **BroadcastReceiver** klassen fungerer ved at manifestfilen har en definert <uses-premission> ved BOOT_COMPLETED. Oppgaven til BroadcastReceiveren overvåker tilstanden til applikasjonen, og ved oppstart av telefonen starter den en ny intent som sender applikasjonen til:
    a. **SetRepeatingService:** Denne Serviceklassen har som oppgave å sette tidspunktet for sending av tekst meldinger fra SharedPrefrences. Basert på input gitt av brukeren i settings menyen. Den operer med en boolean verdi fra SharedPrefrences satt av bruker ved ønsket om å aktivere eller deaktivere SMS tjenesten. Hvis bruker ønsker at SMS tjenesten skal være aktivert sendes man på det gitte tidspunktet videre til:

b. **NotificationAlarmService** denne klassen har som hovedfunksjon å sende ut bursdagsmeldinger til venner som har bursdag på den gitte datoen, og sende ut Notifications til brukeren om hvilke venner som har fått tilsendt bursdagsmelding i dag.

8) Hjelpeklasser:
    a. **DatePickerFragment:** For input validering av dato har vi valgt å benytte oss av DatePicker. Å få DatePicker til å samhandle med et eksisterende fragment var en tidskrevende prosess, men den ferdig implementerte løsningen fører meg seg riktig dato input som ikke krever ekstra validering.
    b. **TimePickerFragment:** Vår løsning bruker som tidligere nevnt en **SettingsActivity** som utvider Androids PreferenceActivity-klasse. Dette grensesnittet har ikke egne standariserte pickers, men krever at man lager sin egen variant. Vi måtte derfor designe vår egen TimePicker, denne finnes i klassen **TimePrefrence** og fungere som de andre prefrenceActivity-klassene, ved at inputen blir validert og plassert i sharedPrefrences.

Validering av input-data



Illustrasjon 4: Input-validering.

Input-kontroll ved hjelp av xml

Ved å sette input-typen til EditText-feltene for fornavn og etternavn til ”textCapWords” får man automatisk stor forbokstav i alle ord. Det finnes noen unntak hvor dette kan være plagsomt, for eksempel ved ”van” i navnet ”Vincent van Gogh”, men i de aller fleste tilfeller mener vi dette er et positivt bidrag til applikasjonen. Fordi kontaktlisten blir estetiske penere og mer konsistent. Samtidig slipper man da å ta hensyn til dette i sorteringsmetoden som passer på hvilken rekkefølge elementene i arrayListen blir fremstilt.

Man kan velge å sette en makslengde på input-tekstene ved hjelp av android:maxLength. Siden SQLite-databasen ifølge dokumentasjonen ikke har noen makslengde på tekst, er ikke dette nødvendig å sette for vår applikasjon slik den er i dag. Vi velger heller å opprettholde fleksibilitet på dette punktet.

Telefonnummer-feltet har fått input-type ”phone”. Dermed tillates bare sifre og tegnet ’-’. Siden lengden på telefonnumre kan variere avhengig av om det er et utenlands nummer, eller et service nummer har vi ikke satt noen grense på maks antall tegn.

Input-kontroll ved hjelp av Java

Selv om XML-filene hjelper oss å kontrollere input-verdier, hadde vi i objekt orientert ånd lyst til at venne-objektene skulle få kontrollere sine egne input-verdier. Vi har derfor RegEx-baserte valideringsmetoder for fornavn, etternavn og telefonnummer.

Vi har valgt å gjøre det obligatorisk å legge inn telefonnummer, siden det å sende ut SMS er blant hoved funksjonaliteten til applikasjonen. Dette valget kan selvfølgelig diskuteres. Vi har også satt et minstekrav om at man må legge inn enten fornavn eller etternavn – forsøk på å legge inn person uten navn vil ikke passere valideringstestene, og man vil få beskjed om at dette er påkrevd.

Ved å bruke fleksibiliteten Java gir oss, kan vi også la Java samle opp de ulike feilmeldingene som kommer fra valideringsmetodene hvis man prøver å legge inn ugyldige verdier. Dermed kan vi presentere feilmeldingene samlet til brukeren når man forsøker å lagre objektet, slik at brukeren kan rette opp både fornavnet og legge til et telefonnummer, heller enn at brukeren først får melding om at fornavnet inneholder tall og deretter, etter å ha korrigert fornavnet, får beskjed om at telefonnummer er obligatorisk.

Navn kan kun inneholde bokstaver, mellomrom og tegnet ”-”. Telefonnumre får bare inneholde tall og tegnet +. Som tidligere nevnt forholder vi oss ikke til noen maks lengde ved validering av telefonnummeret. Det er mulig man burde sette en minimumsgrense for hvor kort telefonnummeret kan være, men vi har ikke gjort dette i vår applikasjon.

Samarbeid

Samarbeidet har fungert bra, vi startet prosessen med å opprette en felles plattform ved hjelp av Dropbox for deling av inspirasjonsdokumenter og utveksling av ideer. Videre i prosessen utarbeidet vi en mappe-struktur hvor vi tok backup av eksisterende kode, før vi implementerte og arkiverte løsninger vi hadde utarbeidet individuelt. Vi fordelte individuelle arbeidsoppgaver som vi fullførte til vi møttes til faste møter på tirsdagsforelesningene. Nærmere innlevering arbeidet vi i felleskap med å utvikle ønsket funksjonalitet i vår applikasjon.

Utfordringer

Oppgaven har bydd på en rekke utfordringer. Hovedutfordringen har vært å tilegne seg nok kunnskap til å bli komfortabel med å arbeide med en applikasjonsstruktur, som i stor grad involverte funksjonaliteter vi ikke har tilstrekkelig kompetanse på. Denne prosessen har vært meget lærerik da vi har

blitt presset til å få nødvendig kompetanse til å implementere og manipulerer funksjonalitet til å passe våre spesifikke krav.

Valget om å basere den grafiske funksjonaliteten på fragmenter har gitt oss en større forståelse for fragmenter som konsept, og hvordan det bedrer livssyklusen til applikasjonen. Det var en stor utfordring implementere, men det er etter vår forståelse blitt den aksepterte måten å styre en applikasjons sin livssyklus og en mer fleksibel måtte å bygge grensesnitt på.

Et annet utfordrende problem vi møtte på er knyttet til at bursdagsdatoen også inneholder år. Dermed vil en simpel sammenligning av dagens dato og en venns fødselsdag ikke være tilstrekkelig, siden denne bare vil matche dersom din venn er født i år (i dag). Dermed er man nødt til å fjerne informasjon om år fra dagens dato og også fra vennens fødselsdag før disse sammenlignes. Siden man bruker lokalt format på datoen, kan årstall i teorien være både aller først eller aller sist og bestå av fire eller to siffer. Skilletegnene i datoen kan også være forskjellige.

Vi tenkte mye på hvordan dette kunne løses. En mulighet er å tilpasse DatePickerFragment slik at man ikke får legge inn årstall. Dette ville vært det enkleste, men siden det er en bursdagsapplikasjon følte vi at årstall var en naturlig del av profilen til en venn. Ved en senere utvidelse kunne denne informasjonen også brukes til å vise alderen til vedkommende i for eksempel notifications eller ved individuelle bursdagsmeldinger.

Den valgte løsningen ble å bruke regulære uttrykk for å finne ut hvilken form vennens dato er lagret i, for så å fjerne dette årstallet fra fødselsdatoen ved sammenligning med dagens dato. Ulempen ved denne løsningen er at det er en lite fleksibel løsning, siden man kun vil klare å fjerne årstallet kun for akkurat de datoformatene man har programmert en løsning for. Det er fort gjort å snuble i regulære uttrykk og å dekke den uttømmende listen av formater er en tidkrevende jobb.

Datoformat vi har programmert for, slik at programmet klarer å fjerne informasjon om år før sammenligning av strenger:

- dd/mm/yyyy
- dd-mm-yyyy
- dd.mm.yyyy
- mm/dd/yyyy
- mm-dd-yyyy
- mm.dd.yyyy
- yyyy/mm/dd
- yyyy-mm-dd
- yyyy.mm.dd
- yyyy/dd/mm

- yyyy/mm/dd
- yyyy.mm.dd
- d/mm/yyyy
- m/dd/yyyy
- m.dd.yyyy
- dd/m/yyyy
- mm/d/yyyy
- dd.m.yyyy

Videre potensiale og mulige problemer

Vår applikasjon fyller alle kravene satt i oppgaveteksten, pluss implementering av ekstra funksjonalitet. Ved setting av tidspunkt for utsending av SMS har vi ved en anledning oppdaget at det kan forekomme en feilmelding: «BirthdayBuddy not responding, do you want to close?» denne feilen skyldes visstnok for mye arbeide på emulatoren, og kan fikses ved å slettet emulatorens «cache». Dette kan enkelt gjøres ved å starte emulatoren på nytt. Feilen forekommer så langt vi har sett ikke ved kjøring på mobile enheter.

Noe vi ønsket å implementere, men ikke rakk, var støtte for ”fast scroll” ved hjelp av en alfabetisk liste til høyre i ListViewet, slik at man lett kan bla selv om vennelisten var lang.
Videre hadde det vært ønskelig å legge til funksjonalitet slik at man kunne legge til et personlig bilde for hver enkelt venn.

Vi har dessverre måttet nedprioritere horisontal design siden det er meget tidkrevende å få dette til å se pent ut. De aller færreste applikasjoner vi har brukt har støtte for dette, så brukeren forventer antagelig ikke denne funksjonaliteten. Vi ønsket heller å fokusere på å jobbe med konsekvente navigasjonsmønstre og funksjonalitet vi ikke har prøvd oss på før.

Et mulig problem vi ikke har hatt tid til å prioritere, men som man kanskje bør ta høyde for, er det tilfellet hvor man legger til en venn med fødselsdato, for senere å endre de lokale innstillingene for dato-format (noe som blant annet skjer når man skifter språk). Siden applikasjonen i seg selv alltid vil bruke de lokale innstillingene, risikerer man at venner som er lagt til i databasen med ”gamle” lokale innstillinger ikke vil få noen bursdagsmeldinger, siden deres dato ikke vil matche dagens dato hvis disse er formatert ulikt. En mulig løsning på dette er å oppdatere bursdagsdato for alle vennene i databasen ved endring av lokalt datoformat.