NTNU
Det skapende universitet

Innendørs kart og navigering

3D visualisering og relasjoner til eksterne data

Jon Villy Selnes Meidell

Master i datateknikk

Oppgaven levert: August 2011

Hovedveileder: Alf Inge Wang, IDI

Biveileder(e): Gunnar Rangøy, Trådløse Trondheim

Norges teknisk-naturvitenskapelige universitet

Sammendrag

Målet med denne oppgaven var å demonstrere et konsept om innendørs navigering med 3D kart-visning på android-platformen.

Basert på tilgjengelig dokumentasjon, hjelp av noen utvalgte kode og applikasjonseksempler, samt utforsking av en android-enhet ble det gjort observasjoner som gjorde det mulig å selv utvikle en applikasjon for denne platformen.

Prosjektet har gitt en god innsikt i android-platformen og programmering med OpenGL ES, som har gitt meg mye ny kunnskap.

Forord

Denne rapporten er utarbeidet i forbindelse med min masteroppgave ved Norges teknisk-naturvitenskapelige universitet (NTNU), Trondheim, der Alf Inge Wang har vært hovedveileder og faglærer.

Oppgaven har blitt gjort i samarbeide med Trådløse Trondheim, der Gunnar Rangøy har vært min veileder. Jeg takker for innblikk i et av deres prosjekter, Campus Guiden, som har gitt inspirasjon i arbeidet med denne oppgaven. Håper jeg har bidratt til nye ideer i deres prosjekt.

Jeg vil også rette en takk til Mikael Valen-Sendstad, for tips og råd.

Til slutt vil jeg takke Gud min skaper, og Jesus Kristus min frelser, for støtte og velsignelse i medgang og motgang.

Dokumentstruktur

Denne rapporten er inndelt i tre deler:

Del 1: Introduksjon
Denne delen starter med en introduksjon til prosjektet i form av en prosjektbakgrunn. Kapittel 2 tar for seg et utvalg forskningsmetoder, samt denne oppgavens forskningsmetode, mens kapittel 3 gjengir kort fra mitt fordypningsprosjekt om samme emne.

Del 2: Forstudie
Denne delen begynner med et kapittel som tar for seg android-platformen, først med en innledning for så å ta for seg begreper og konsepter på platformen som har vært nødvendige for denne oppgaven. Kapittel 5 tar for seg noen kode og applikasjonseksempler som gjorde innstuderingen til platformen enklere.

Kapittel 6 tar for seg emnet 3D-grafikk samt OpenGL ES, som er grafikkbilioteket på android-platformen, men kapittel 7 tar for seg andre emner som har en betydning for denne oppgaven.

Del 3: Eget arbeid
Denne delen tar for seg det arbeidet jeg har lagt ned i denne oppgaven. Det første kapittelet definerer krav til applikasjonen, mens kapittel 9 gir en gjennomgang av arkitekturen for applikasjonen. Kapittel 10 gir innblikk i implementeringsfasen. Avsluttningsvis konklusjoner for oppgaven.

Innhold

Tabeller

Figurer

Del I

Introduksjon

Kapittel 1

Prosjektbakgrunn

1.1 Oppgavetekst

Med hensyn til interesseområdet for oppgaven har man kommet frem til følgende oppgaveformulering:

Oppgaven vil undersøke det å fornye tradisjonelle plantegninger til muligheten for 3D visualisering for navigering og å gjøre seg kjent i bygninger, der det er ønskelig å kommunisere med systemer slik som romreservering, inventarliste osv. i forbindelse med lokalisering. Prosjektet utvikles mot den mobile platform, nærmere bestemt Android.

1.2 Motivasjon

I dag finnes det mange produkter og løsninger for å navigere i trafikken med hjelp av GPS. Verktøyet navigasjon har gjort det enklere å komme seg fra A til B. Dette er ikke nødvendigvis samme opplevelse når man skal finne frem på store offentlige bygg, i den skala NTNU Gløshaugen representerer.

Mitt fordypningsprosjekt [16] startet arbeidet med å få definert ideer og tanker om løsninger for denne konteksten. Det arbeidet kom ikke så langt i implementasjon av konseptet, noe jeg ønsker å gjøre denne gangen.

På mobilplatformen er Android en stor aktør i vekst[17]. Siden et lite firma med samme navn ble oppkjøpt av Google i 2005 [12] har de klart å bli en de ledende operativsystemene på mobilplatformen. Det ses derfor på som en nyttig investering å lære seg å utvikle på denne platformen.

1.3 Prosjektmål

Målet med prosjektet er å utvikle en applikasjon på android-platformen som gir en kartvisning som viser en 3D representasjon av bygninger på Gløshaugen. Et annet mål er å kunne komme lengre med konseptet jeg definerte i mitt fordypningsprosjekt og ende opp med en applikasjon som kan demonstrere disse definerte ideene, og eventuelt i fremtiden bli en inspirasjon og på forhånd kunne slippe de problemene som måtte oppstå i denne oppgaven, i form av en formiddling av de erfaringene jeg gjør med dette prosjektet.

1.4 Spørsmålsstillinger

Jeg har her stilt noen spørsmål som jeg ønsker å finne svar på med denne oppgaven.

- Hva er android og hvordan utvikle applikasjoner på denne platformen?
- Hvordan utvikle 3D-grafikk-applikasjoner på android-platformen? Skiller dette seg fra slik programmering på datamaskiner?
- Finnes det grafikkrepresentasjon av bygninger på NTNU gløshaugen som lar seg bruke som modeller?
- Hvordan koble til andre datakilder og kobler disse til objekter i kartet?

1.5 Interessenter

The stakeholders in this project are the project group of one person and the course staff.

- Jon Villy Meidell - Forfatter
- Trådløse Trondheim (Gunnar Rangøy) - Veileder
- Alf Inge Wang - Faglærer

Kapittel 2

Forskning og metodevalg

Dette kapittelet tar for seg forskning ved programvareutvikling, samt å diskutere forskningsmetode for denne oppgaven.

2.1 Forskningsmetoder

Ved forskning i programvareutvikling er det ønskelig å finne verktøy, utviklingemetoder og modeller som gjør det lettere, mer effektivt og billigere kostnademessig å utvikle programvare med tanke på tidskostnader og ressurskostnader.

Basili [2] beskriver forskningsmetoder ved programvareutvikling med følgende metoder:

Empirisk metode
En statisktisk tilnærming er brukt for å validere en gitt hypotese. Data samles inn for å enten vertifisere eller falsifisere hypotesen. Med denne metoden kan man se om ny teknologi er værre eller bedre enn for eksisterende løsninger.

Matematisk metode
Denne metoden baserer seg på den formelle måten å gjøre eksperimenter på, ved å utvikle en formell teori og avvikende resulteter sammenlignes med empiriske observasjoner.

Utvikler metoden
Et system utvikles og testes for å se om det virker slik man har hatt en hypotese om. Programvareuvikling er en kreativ prosess, der begrensingene gis ved gitte begrensinger for det ferdige produktet.

En annen metode er å tilnærme seg et emne ved å sette seg inn i tilgjengelig litteratur.

2.2 Metodevalg for denne oppgaven

Dette prosjektet innledes med en forskningsfase i form av å studere litteratur og programeksempler for android-platformen. Jeg har ingen forkunnskaper om emnet, eller hva android-enheter tilbyr, annet enn at det er en smart-telefon. En viktig del av den innledende forskningen vil da også være å prøve ut en android-enhet for å observere og la seg inspirere hva tilgjengelige funksjoner og brukerglede angår.

Etter å ha etablert en forståelse og innsikt i hva platformen har å tilby, er det mulig å definere hva denne platformen gir som kan utnyttes i min utviklingsprosess. Dette vil også påvirke hvordan applikasjonens arkitektur blir.

Under utviklingsfasen vil implementerte funksjoner testes ved vanlig brukerinteraksjon, men også ved å validere loggutskrifter (som forklart i seksjon 4.1.1.3). Med resultatene fra dette kan hypoteser for hvordan den implementerte koden burde oppføre seg kunne valideres eller falsifiseres, der man enten da går vidre til implementasjon av annen funksjonalitet med nye hypoteser, eller må revurdere og utarbeide en ny hypotese om hva som vil korrigere avikene fra forrige hypotese og gi ønsket oppførsel.

Kapittel 3

Tidligere arbeid

Forrige semester jobbet jeg med en definisjon av konseptet ved innendørs navigering i mitt fordypningsprosjekt [16]. Implementeringensfasen ble ganske kort, noe som førte til at flere av konseptene og hypotesene ikke ble prøvet ut som applikasjon.

Dette kapittelet oppsummerer konseptet slik det ble presentert i fordypningsprosjektet. Rammene for denne oppgaven er noe anderledes, men grunninspirasjonen har vært den samme. Det kan også bli mulig å se om man har funnet ut noe mer eller fått tatt konseptet videre eller i en ny retning.

3.1 Scenarioer

Fire scenarioer ble gitt for å belyse situasjoner der en slik applikasjon ville løst situasjonen eller gjort hverdagen enklere for de beskrevede karakterene.

Scenario 1

En student ønsker å møte en professor for i IT-vest bygningen, for å diskutere strukturen for et fordypningsprosjekt. Rommene i etasjen viser seg ikke å være i en kronologisk rekkefølge. Studenten oppholder seg i midten av en lang gang, og ønsker ikke å kaste bort dyrebar tid på å skulle gå i feil retning.

Scenario 2

En gjesteforeleser ankommer NTNU Gløshaugen med taxi. Foreleseren har ikke vært på campus før, og er allerede sent ute på grunn av et forsinket flyvning og stor trafikk fra flyplassen. Han ønsker kaffe og noe å bite i på veien til forelesningssalen han ikke vet hvor er.

Scenario 3

En vaktmester finner møbler på utsiden av et grupperom. Innsiden ser ikke ut. Vaktmesteren ønsker å få oversikt på hvilke møbler som egentlig skal være på rommet, men også finne ut mer om årsaken til denne tilstanden.

Scenario 4

En gruppe studenter skal møtes for å jobbe med et prosjekt. De har reservert et grupperom, men vet ikke hvor det er. Under en pause oppdager de et annet rom, som er mindre utsatt for støy fra omgivelsene, og lurer på om dette rommet er tilgjengelig for å reserveres.

3.2 Konseptet

Basert på scenarioene ble det reflektert over tre hovedtemaer:

- Innendørs lokalisering.
- Eksterne datakilder.
- Teknologiplatformer.

Det ble presentert en hypotese i form av en måte å finne en sannsynlig posisjon til gitte navigasjonpunkter, og ikke absoluttposisjon, basert på observerte basestasjoner og signalstyrke til disse navigasjonspunktene. Signalavlesning til nærliggende tilgangspunkter ville på en tidligere anledning registeres for navigasjonspunkter, slik at man i nåtid sammenligner tidligere registerte profil og finner det navigasjonspunktet som gir best sammenligning og sannsynlig nærliggende lokalitet.

En annen ide var å kunne sammarbeide med andre systemer, som holder på data som kan kobles til elementer i kartet.

Det ble og drøftet noen momenter for applikasjonstilpasning for en mobil eller stasjonær platform.

Del II

Forstudie

Kapittel 4

Android

I dette kapittelet vil jeg først gi en oppsummert introduksjon til android-konseptet, før fokus går over på utviklingsmiljøet og nødvendige begrep og elementer hos android-platformen sett fra en utviklers synspunkt.

Android-versjonen som omtales er versjon 2.2, eller API versjon 8, som er androidversjonen som kjører på Galaxytab GT-P1000, og som har vært utviklingsplatformen i dette prosjektet.

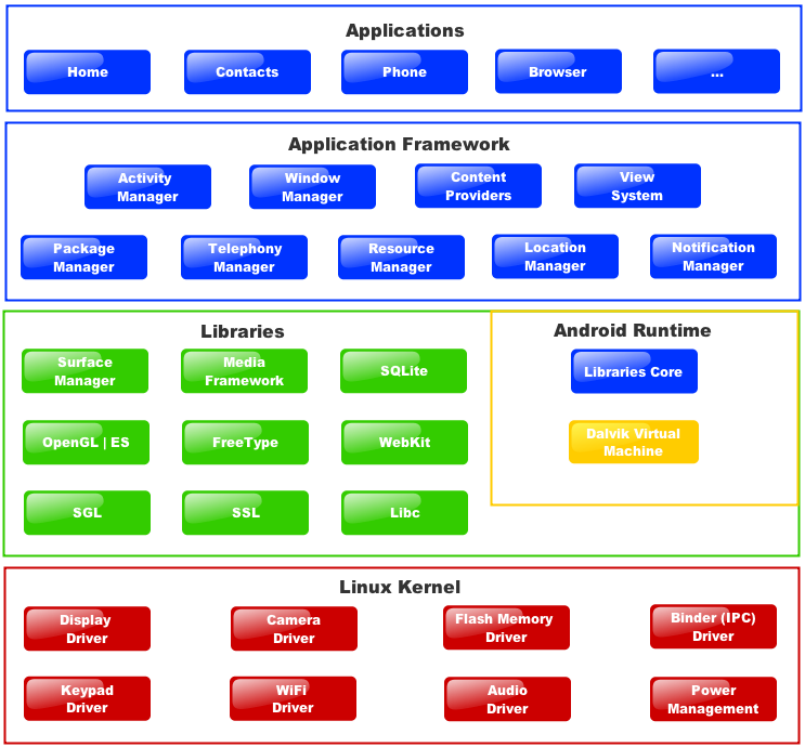
Sommeren 2005 tok Google til seg det til da ukjente mobilprogramvarefirmaet Android Inc [12]. Android så et stort potensiale i utviklingen av smartere telefoner som er bevisst på omgivelsene. Google er i dag den ledende aktøren i dette prosjektet, basert på åpen kildekode. Google begrunner valget av åpen kildekode med at de ikke ønsker at noen industri aktør skal kunne sette begrensninger eller kontrollere hverandres innovasjon. [18]

En allianse forbundet med Android er Open Handset Alliance [1], der 83 firmaer knyttet til programvare, telefoner, tjenesteleverandører og andre teknologifirmaer. Disse har som mål å sette en standard i industrien, og tilby telefoner og tjenester som utnytter android-plattformen.

Den første android enheten kom i oktober 2008 [13]. I dag aktiveres over 500 000 android-enheter hver dag. [17]

4.1 Android - programvarekonseptet

Android er et operativsystem beregnet på mobilplatformen bygd opp i flere lag. Hovedtankegangen på platformen er at alle applikasjoner har samme rettigheter, og definerer selv hvilke ressurser applikasjonen trenger tilgang til. Brukeren blir også gjort oppmerksom på hvilke ressurser applikasjonen trenger under instalasjonen.



Figur 4.1: Android har fire programvarelag: applikasjon, applikasjonsrammever, biblioteker og kjerne.

Applikasjoner kan også gjøre sine data tilgjengelig for andre applikasjoner. Ifølge [11] og [15] har Android-programvarearkitekturen fire lag:

Kjernelag
Android bruker linux som kjernelag. Linux tilbyr virtuelt minne, nettverk, drivere og strømforbrukhåndtering.

Native biliotek - lag
Dette laget består av en hovedsaklig SGL (underliggende 2D grafikkrenderer i Android), OpenGL ES (3D grafikkrenderer, se 6.4), SQLite

(databaselager for internlagring, se 4.5.1), media bibliotek (støtte for forskjellige video, bilde og lydformater) WebKIt (WEB-renderer).

En annen viktig del av dette laget er Dalvik [6], en virtuell maskin, på samme måte som Java VM, men optimalisert for den mobile platformen med tanke på effektivitet og at den skal kjøre på mindre prosessorer. Kjernebiblotekene er implementert i Java, og tilbyr de samme hovedklassene som vanligvis er å finne i Java VM.

Applikasjonrammeverkslag
Dette laget er implementert i Java og gir et applikasjonsprogrammeringsgrensesnitt som gir tilgang på funksjontalitet slik som deling av data, telefonmodulen, kjøre bakgrunnstjenester, sette varsler til statuslinjen osv. Noen hovedelementer er Content Providers (se 4.5.2), Resourcee Manager (se 4.6.1) og Activity Manager (se 4.4).

Applikasjonslag
Applikasjoner for Android skrives i Java, som på enheten kjøres på den virtuelle Dalvik maskinen. Selv hovedfunksjonalitet for telefon og kontakter ligger i dette laget. Tredjeparts applikasjoner har tilgang til de samme resursene som applikasjonene Android utviklerene selv har utviklet. Brukeren kan selv velge hvilken applikasjon som skal håndtere en hendelse, eller kall for åpning av url eller fil. Hver applikasjon kjører sin egen prosess på den virtuelle maskinen, noe som sikrer bedre systemstabilitet selv om en applikasjon skulle krasje. [8]

4.1.1 Utviklingsmiljø

Utviklingsmiljøet består av Android SDK (Software Development Kit), ADT Plugin (Android Development Tools) for Eclipse og en instalasjon av Eclipse selv. Disse lastes ned seperat.

4.1.1.1 SDK

Utviklingsverktøypakken består av:

- Androidplatformens bibliotek (android.jar)
- Dokumentasjon for biblioteket
- Platformverktøy.
- Eksempler

Man kan i utgangspunktet velge selv hva man laster ned av feks. dokumentasjon og android-platformversjoner.

Biblioteket består av pakker implementert spesifik for android i form av android.*, men også standardklasser og funksjonalitet fra java.*, javax.* og organisasjoner som org.apache.http og xmlsøtte org.xml*. Eclipse ser pakkene, deres klasser og funksjoner, noe som gjør selve utviklingen raskere enn ved fravær av denne funksjonaliteten.

Videre kan man med dokumentasjonen ha tilgang til JavaDoc dokumentasjonen av bibliotekene lokalt, noe som også er tilgjengelig i sanntid i Eclipse, nyttig måte å kunne lese seg opp på funksjonalitet, uten å på forhånd måtte ha noe oversikt på bibliotekene.

En annen komponent er platformverktøy i form av bla. en konfigurerbar emulator (kan opprettes og startes enkelt fra eclipse) og ADB (Android Debug Bridge [4]). Sistnevnte gir tilgang en kommandolinjebasrt verktøy som kommuniserer med en emulator eller android-basert enhet, der man på emulatoren får full root-tilgang, mens man på en android-enhet for noe begrenset tilgang til filer, men fortsatt mulighet til loggen.

Ved hjelp av loggen, som aksesseres med logcat og mulig filtrering for applikasjon og grad av feil, kan man lese informasjon som applikasjonene gir. Av loggen vil også den virtuelle maskinen gi stakksporing og feilmeldinger om et program får en alvorlig feil ect.

4.1.1.2 ADT Plugin

Dette tillegget til Eclipse gir tilgang til vedlikeholdsprogrammet til SDKen (SDK Package Manager), men gir også Eclipse støtte for Androidprosjekter, applikasjoneksempler og et redigeringsprogram for brukergrensesnitt (se 4.6.1.

4.1.1.3 Logging/avlusing

Et viktig verktøy i utviklingsprosessen er å kunne oppdage feil og få en form for tilbakemelding på hva som forårsaket feilen. Ved å benytte seg av ADB og logcat kan man ved uhåndterte avbrudd (exceptions) få full stakksporing fra AndroidRuntime, men også ved å logge fremgangen i sin egen kode og funksjoner vertifisere og kontrollere rett adferd. Logging er tilgjengelig via android.util.Log.

Ved bruk av logcat, kan man definere filtere for applikasjon og feiltype, av typen <applikasjon>:<feil>.

<feil> i form av en bokstav:

- V - Verbose (laveste prioritet)

- D - Debug
- I - Info
- W - Warning
- E - Error
- F - Fatal (høyest tilgjengelig for bruker/utvikler)

<feil> definerer laveste grense for hva som skal vises fra loggen, dvs bruker man I, vil kun V og D filtreres bort, da W, E og F har høyere prioritet.

Man kan definere generelle filtre med hjelp av stjerne-tegn, feks: '*:E Applikasjon:D' der alt fra debugnivå og opp vil vises for gitte applikasjon, mens man ellers lister error-nivå eller værre.

Android tilbyr også en implementering av Junit og monkeyrunner for testing og vertifisering av kode [10], men dette har jeg ikke benyttet meg av, og vil derfor ikke kommentere dette ytterligere.

4.2 Applikasjon

En android-applikasjon er bygget opp av en eller flere aktiviteter (activity), der overordnet informasjon om applikasjonen defineres i en xml-fil, AndroidManifest.xml [9]. I denne filen defineres og deklareres bla.:

Overordnet info
Applikasjonens pakkenavn, applikasjonsversjon og android-sdk-minimumsversjon.

Aktiviteter
Alle aktiviteter i applikasjonen listes med relativbane i klassen og klassenavn. For hver aktivitet defineres intent-filtre (se 4.4.1) for hvilke handlinger, kategori, dataformat (mimetype), url eller filbane de tjener. MIME vil si mediatype [25].

Rettigheter til bruk
Skal en applikasjon ha tilgang til sensorer eller andre interne komponenter på android-enheten, må det defineres i manifestet hva applikasjonen trenger tilgang til. Slik funksjonalitet er feks. internettilgang, eksternt lager, trådløskort-adgang ect.

Leverandør (Provider)
Inneholder applikasjonen en dataleverandør må dette reigstreres i manifestfilen. (se 4.5.2)

Tjeneste (Service)
Gir muligheten til å definere funksjonalitet som skal kjøre i bakgrunnen feks. som server, som i seg selv ikke har noe brukergrensesnitt.

En applikasjon har en egen datastruktur i form av en *Context*, som er et objekt som holder på visningselementer, referanser til systemet, tilbydere osv.

4.3 Intentensjon (Intent)

Et sentralt begrep ved starting av aktiviteter, men også å sende og motta data mellom applikasjoner og aktiviteter som samarbeider, er en Intent. Den limer sammen aktiviteter.

En Intent har to primærattributter i form av handling (action) og data. Handlingen definerer hva man vil som ønskes gjort med dataene, der noen eksempler på handling er *ACTION_VIEW*, *ACTION_EDIT* og *ACTION_MAIN* som henholdsvis vil vise dataene, redigere dataene, og starte standardaktiviteten for gitte data.

Dataene består av en URI (Uniform Resource Identifiers, RFC 2396 [24]), eller en sammensmelting av *URL* og *URN* [32]. En URI kan være en web-adresse på formen *http://www.webside.com/bane*, eller *file://bane/enfil.type*, mens man på android ofte opererer med *content://leverandør/hva/id* (se provider 4.5.2).

Når en handling og data kombineres til en intent, vil systemet selv kunne finne activiteter og applikasjoner som støtter gitte data og ønskede handling. Dette defineres i en applikasjonens manifestfil (se 4.2) i form av intent-filtre, som vil bli nærmere beskrevet i 4.4.1.

En Intent kan også utstyres med sekundærattributter. Disse er kategori, type (Mime-type), komponent eller ekstra. En kategori utdyper definerte handling og kan feks. be om topp-nivå applikasjonen skal startes, eller alternative behandlinger av dataene.

Ved å definere type, vil systemets tolkning av hva slags data det er snakk om overstyres. En komponent er en eksakt aktivitet man ønsker skal åpne dataene, mens ekstra (Extras) gjør det mulig å sette nøkler og verdier i form av en innpakning (Bundle).

Når kun primærattributter er definert, vil dette utgjøre en implisitt Intent, der det er opp til systemet selv å finne egnet aktivitet til å behandle dataene. Finnes det flere aktiviteter som tilbyr samme handling for dataene, vil brukeren typisk kunne velge hvilken aktivitet som skal behandle dataene.

Ved definert komponent, vil intensjonen være eksplisitt, da det er eksakt denne aktiviteten som ønskes til å behandle dataene.

Når en aktivitet starter en annen aktivitet, kan den be om å få et resultat fra denne aktiviteten. Svaret vil også være i form av en Intent.

4.4 Aktivitet

En aktivitet tilbyr brukeren et skjermbilde som lar brukeren gjøre ett eller annet. En aktivitet er ikke det samme som et program/prosess på en datamaskin, da man starter aktiviteter, men avslutter de ikke. Aktivitetene legger seg i en stakk, der aktiviteten som ligger øverst i stakken er det brukeren ser på skjermen. Om brukeren trykker tilbakeknappen, vil den øverste aktiviteten avbrytes og fjernes fra stakken, og forrige aktivitet blir øverste aktivitet på stakken og vises for brukeren. Om aktiviteten som ble avbrutt ble startet av aktiviteten under på stakken, vil den gjenopprettede aktiviteten motta etterspurte data i form av en Inten fra den avbrutte aktiviteten. Dette vil også skje om den øverste aktiviteten fullfører etterspurte handling, og sier seg ferdig og gir nevnte resultat.

På grunn av denne anderledese tankegangen, har aktivitetene hver sin livssykel (se figur 4.2) i form av metodene:

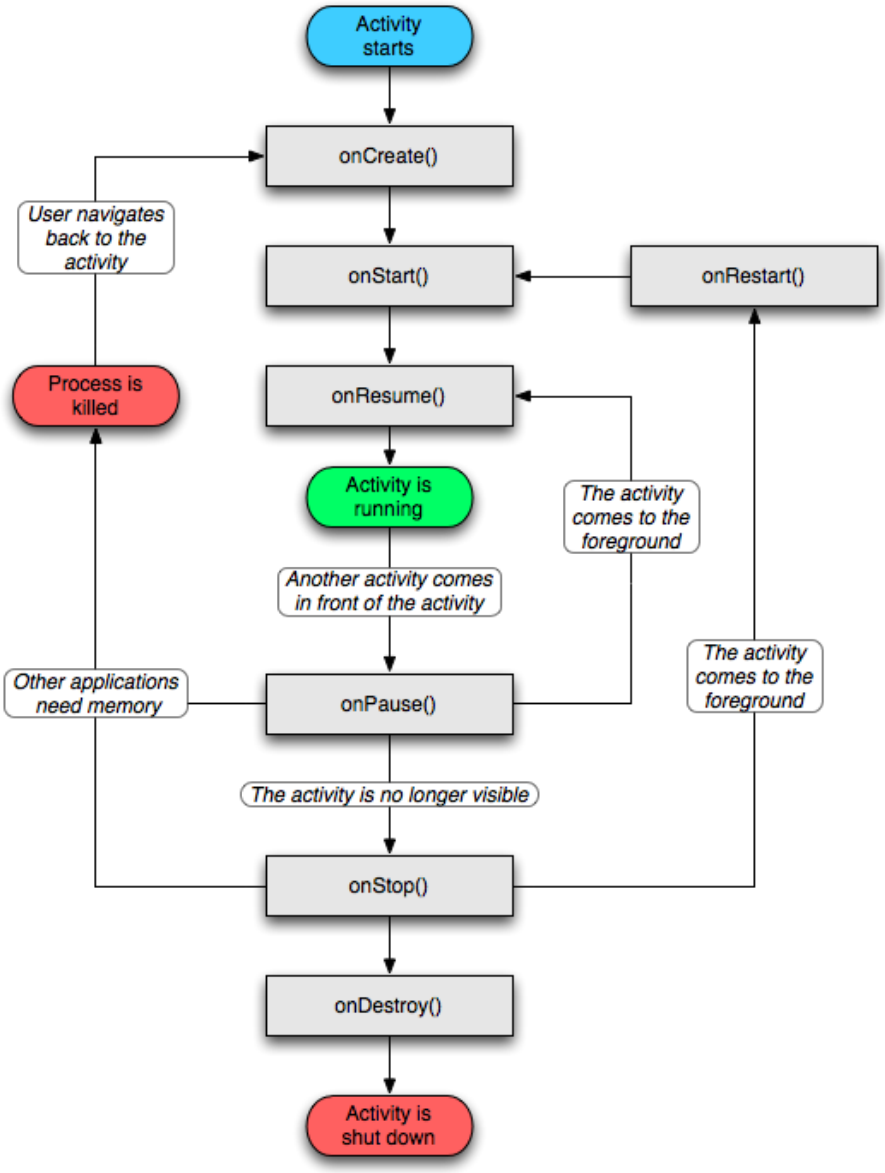
onCreate
: Kalles i det aktiviteten startes, der nødvendige komponenter og brukergrensesnittelementer initialiseres.

onStart
: Aktiviteten er i ferd med å bli vist på skjermen.

onResume
: Kalles i det aktiviteten vises på skjerm, får fokus og hva brukere har oppe. Denne metoden kalles også når bruker kommer tilbake til en aktivitet, som ikke har blitt avsluttet, etter å ha vært innom en annen aktivitet.

onPause
: Kalles i det brukeren er i ferd med å forlate en aktivitet til fordel for en annen. Den andre aktiviteten er over denne aktiviteten, men denne aktiviteten kjører fortsatt.

onStop
: Kalles i det en aktivitet tar overhånd og denne aktiviteten legges i bakgrunnen. Denne aktiviteten er fortsatt i live og ligger i minnet, men er ikke knyttet til noe skjermvindu lengre, og ikke synlig for brukeren.



Figur 4.2: Aktivitet for å redigere og behandle noder..

onDestroy

Aktiviteten avsluttes, og lastes ut av minnet.

Disse metodene utgjør tre nivåer av levetid:

Totallevetid

Startes med *onCreate* og avsluttes med *onDestory*. Hovedtilstander og

lignende bør opprettes og avsluttes med disse metodene.

Synlig levetid
Startes med *onStart* og avsluttes med *onStop*. Resurser som påvirker brukergrensesnittet bør lastes og avsluttes her. Disse metodene kan kalles flere ganger under en aktivitets totalelevetid ettersom andre aktiviteter får forgrunnen og denne aktiviteten blir usynlig.

Forgrunn levetid
Startes med *onResume* og avsluttes med *onPause*.

En aktivitet kan aksessere data direkte, men det vanligeste er å gå gjennom en tilbyder (Content Provider, 4.5.2) som gir et grensesnitt i form av en *URI* og returnerer en *Cursor* 4.5.2.1. En tilbyder gir alle aktivitetene i en applikasjon tilgang de samme dataene. Andre applikasjoner kan også aksessere en tilbyder. På den måten kan feks. hvilken som helst aktivitet kunne lese adresseboken, om applikasjonen ber om tillatelse til å aksessere disse dataene, som brukeren igjen må bekrefte ved installering, som nevnt i 4.1. Mer om lagring i 4.5.

4.4.1 Intent filter

En aktivitets handlinger og evnte til å behandle gitte data defineres av et intent-filter, i en applikasjons manifestfil (se 4.2). En aktivitet kan ha flere intent-filtre.

Et intentfilter definerer i hovedsak hva slags handling aktiviteten skal respondere til, men det er også mulig å definere hva slags kategori eller hvilke data aktiviteten kan håndtere. Data vil si en *URI* og/eller Mime-type [25]. Skal aktiviteten håndere forskjellige datatyper, der en skal kunne redigeres og vises, mens den andre bare skal vises, må separate filtre defineres.

Systemet har oversikt på alle installerte applikasjoner, deres aktiviteter og filtere, slik at det ved en implisitt intensjon kan finne ut hvilken eller hvilke aktiviteter som egner seg til å behandle eventuelt gitte data.

4.5 Lagring

Android platformen tilbyr forskjellige måter å lagre data på, litt avhengig av hvilke behov man har, også mulighet for å gi andre applikasjoner adgang til dataene eller ikke. Alternativene er som følger:

Delte preferanser (Shared preferences)
Denne formen for lagring gir muligheten til å lagre primitive data på

nøkkel-verdi format, på lignende vis som en *Bundle*. Det vanlige er å lese inn dataene eller benytte standardverdier (ved førstegangs start) når aktiviteten starter, og lagre ved avsluttning.

Database
Android platformen tilbyr SQLite - en simpel og enkel SQL-database. Denne kommenteres ytterligere i 4.5.1

Internlager
Aktiviteten kan lagre data på intern-SD-kortet til Android-enheten, der man kan velge om filene skal være private, eller også gi lese/skrivetilgang til andre applikasjoner ect. Man får tilgang på internlageret enten via *openFileInput* som gir en *FileInputStream*, eller *openFileOutput* som gir en *FileOutputStream*. Parametrene er filnavn og lese/skrive flagg. Det er også mulig å definere egne cache-filer.

Eksternlager
Eksternlageret vil si minnekortet brukeren kan sette i android-enheten, om dette finnes på enheten. Ved bruk av dette, kan det tenkes at lageret kan bli uttilgjengelig om minnekortet tas ut, eller lageret gjøres tilgjengelig for en tilkoblet datamaskin. Her benyttes *File* som er en implementasjon på av samme klasse lik i *java.io.File*.

Nettverkslagring
Android tilbyr *java.net.** og *android.net.** pakkene, som kan brukes til å implementere nettverkskommunikasjon, og potensiell lagring på en server. Kan også nevnes at *org.apache.http.** finnes i biblioteket, slik at man kan benytte seg av en webserver som grensesnitt til lagring.

4.5.1 SQLite

SQLite er et enkelt og kompakt database-bibliotek designet for lav minnebruk, som en erstatter for *fopen* og egne filformat man ellers måtte ha kommet opp med for sin datastruktur. Den kjører på innsiden av applikasjoner, uten behov for noe databaseserver og behøver ingen konfigurasjon for å virke [22].

I motsetning til databaseserverløsninger som gir en klient-server modell som kjører i en egen tjener-prosess med innebygget brukerhåndtering osv, kjører SQLite som en del av applkasjonen som trenger et datalager, med de filrettighetene applikasjonen har i filsystemet, uten noen brukerstøtte [19].

SQLite lagrer en komplett database med flere tabeller, avtrekkere (triggers) og visninger (views, kun lesbare) i en eneste fil. Den største fordelen er databibliotekets kompakthet, som gjør den også mulig å bruke i miljøer med begrenset minnekapasitet slik som bærbare enheter i form av mobiltelefoner, mp3-spillere ect. Databasen er også lett å flytte eller ta sikkerhetskopi av siden alt ligger i en fil, der alle referanser er lagret i denne filen.

SQLite søtter stort sett det meste av SQL92 [22], med noen unntak. En av forskjellene er datatyper. I fullskala databaseservere har man langt flere datatyper, også med tanke på hvor mange bit som trengs for et felt, feks. *smallint* - *int* eller *varchar* - *text*. Datatypene, eller dataklasser[23] , som tilbys på SQLite er:

- NULL - null-verdi.
- INTEGER - heltall med 1-4,6 eller 8 bytes. SQLite skiller har ikke eget format for kun positive heltall.
- REAL - Flytall, lagret som 8-bytes IEEE flytall.
- TEXT - Tekststreng lagret som UTF-8 eller UTF-16
- BLOB - Data, lagret slik som levert.

Man ser at det mangler klasser for dato og tid[23]. SQLite tilbyr de vanlige dato og tid-funksjonene, men dette lagres i databasen enten som *TEXT* som "YYYY-MM-DD HH:MM:SS.SSS", *INTEGER* som unixtid (sekunder siden 1.1.1970 midnatt UTC) eller *REAL* Juliansk dato [27].

I en SQLite tabell, er det definert en anbefalt affinitet for hver kolonne, men dette betyr ikke at dataene der må ha denne typen affinitet. SQLite3 har disse affinitetene:

- TEXT
- NUMERIC
- INTEGER
- REAL
- NONE

Som eksempel på hva en affinitet er, kan man se på *TEXT*: En kolonne med *TEXT*-affinitet, kan bruke datatypene/klassene *NULL*, *TEXT* eller *BLOB*. Det gjør også at alle data deklarert som *CHAR*, *VARCHAR*, *TEXT* osv, vil få *TEXT* affinitet.

4.5.2 Content Provider

Tanken bak en tilbyder er gjøre lagering og henting av data tilgjengelig for alle applikasjoner. Det er eneste måten å dele data mellom applikasjoner på, uten å kalle andre aktiviteter til å håndere dataene man sitter med. Tilbyderen gir også beskjed om det skjer endringer på dataene til de applikasjonene som registrer seg som interesserte, på samme måte som tankegangen i Java om event og event-listeners, der en event-listener får beskjed når noe skjer.

Android har i utgangpunktet flere innebyggede tilbydere, listet i *android.provider*-pakken. For å gjøre sine data tilgjengelig for andre må man enten implementere sin egen tilbyder, eller benytte allerede tilgjengelige tilbydere.

En tilbyders datamodell basererer seg på databasemodellen, der alle oppføringer har en unik _ *id* - en oppfordring til å benytte seg av SQLite. En tilbyder har en egen *URI* (se 4.3) på formen

content://<tilbyder>/<entitet>[/<id>].

Hovedfunksjonene til en tilbyder kan aksesseres fra applikasjonen gjennom *getContenResolver()* for deretter å kalle en av tilbyderens metoder. (se tabell 4.1)

Tabell 4.1: Metodene til en *ContentProvider*

Metodenavn	Beskrivelse
query	Spørring for å lese ut data. Resulterer i en *Cursor*.
insert	Legge til ny oppføring. Trenger *ContentValues*.
update	Oppdatere eksisterende oppføring. Trenger *ContentValues*.
delete	Slette en oppføring.
getType	Få datatype / MIME for gitt *URI*.
onCreate	Opprette lageret om det ikke eksisterer.

For alle metodene, bortsett fra *onCreate* har *URI* en sentral rolle, enten for å definere hvilke data som skal leses, eller en måte å kvittere og gi

interesserte beskjed om nye eller endrede data, i form av en *URI*

4.5.2.1 Cursor

En fordel av å bruke en tilbyder, er at brukergrensesnittet har en del ferdigfunksjonalitet som enklet gjør det mulig å vise dataene. Når en tilbyder spørres etter data, vil resultatet gir som en *Cursor*. En *Cursor* er en datastruktur bestående av rader og kolonner med data, der man flytter en peker fritt mellom radene etter behov, for så å hente ut felt gitt feltnummer eller feltnavn.

Brukergrensesnittet har funksjonalitet som gjør det enkelt og liste data fra en *Cursor* direkte. Mer om dette i seksjon 4.6.

4.5.2.2 ContentValues

Ved innsetting eller oppdatering av data benyttes et *ContentValues* objekt. Dette holder en datastrukur, noe lignende en rad i en *Cursor*, der man setter kollonner gitt kolonnenavn. Datastrukturen gi mulighet for å sjekke om nøkler er definert/finnes, slik at tilbyderen selv eventuelt kan fylle ut manglende data, eller avvise kallet, alt ettersom.

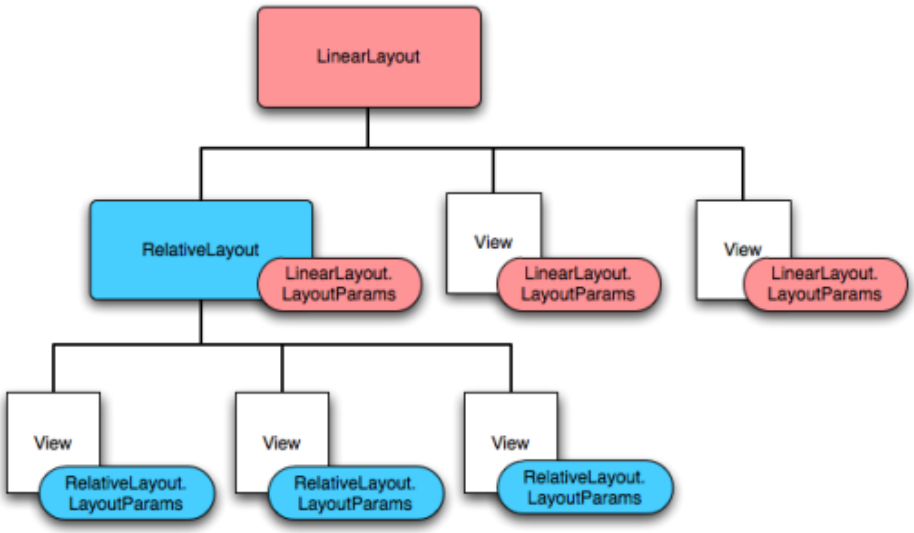
4.6 Layout/Brukergrensesnitt

Brukergrensesnittet på Android-platformen er bygget opp av visninger (views) og grupperinger av disse. Visninger fungerer som løvnoder, mens visningsgruppene fungerer som grener, der dette til sammen utgjør visningen. Rotnoden settes til en aktivitet med metoden *setContentView*.

Hovedgrupperingene kalles layouts. Hver layout har egne parametre for hvordan barnekomponenter plasseres, utnytter tilgjengelig skjermplass som forelderen har og plasserer seg i forhold til hverandre. Barnevisninger eller barnegrupperinger vil derfor få spesieller parametre (se figur 4.3) gitt hvilken type layout de befinner seg i. Slike parametre kalles *LayoutParams*. De vanligeste parameterene er hvilken høyden og bredden en komponent skal bruke av den tilgjengelige plassen foreldrekomponenten har. Dette kan settes med *wrap_content* (komponenten bruker ikke mer plass enn den trenger), *match_parent* (komponenten bruker all plassen foreldrekomponenten har i denne retningen) eller et eksakt oppgitt tall.

Vanlige layouter er:

FrameLayout



Figur 4.3: Layoutparametre. Viser hvordan *LinearLayout* gir barnekomponentene egne attributter på grunn av forelderens type.

Er den enkleste layouten. Denne kan i utgangspunktet fylles med et objekt. Legges det til flere objekter, vil disse tegnes over det forrige. Er den nyeste komponenten til dels gjennomsiktig, kan flere komponenter legger oppå hverandre.

LinearLayout

Denne layouten plasserer barnekomponenten enten vertialt eller horisontalt etter hverandre ved å definere dette. Om det finnes ledig plass i denne layouten, kan barnenodene definerers med vekter (*weight*) som definerer hvilken barnekomponent som har størst rett til å benytte seg av eventuelt ledig plass innenfor foreldrekomponenten, LinearLayout.

TableLayout

TableLayout plasserer barnekomponentene i rader og kolonner, der radene plasseres i en *TableRow*. En rad vil ha like mange klonner som den raden med flest kolonner. Det er ikke mulig å strekke en kolonne over flere kolonner slik man kan i HTML.

RelativeLayout

Med denne layouten plasseres barneelemntene relativt i foreldrekomponentens ramme og til hverandre. Komponentene plasseres over, under, til siden for hverandre, og/eller øverst, nederst, til siden eller i midten vertikalt eller horisontalt av foreldrekomponenten, RelativeLayout.

Denne layouten gir mulighet for de mest avanserte oppsettene, men er også den mest kompliserte.

Andre visningsgrupper er:

Gallery
Et horisontalt blabar visning for bilder.

GridView
En blabar visning på m kolonner og n rader.

ListView
En blabar visning med en enkel kolonne. (se Adapter 4.6.4)

ScrollView
En *FrameLayout* som gjør det mulig å bla i et større element enn hva det er plass til på tilgjengelig plass.

Spinner
En visning som lar brukeren velge en barnekomponent, der den valgte komponenten er det som vises. (se Adapter 4.6.4)

SurfaceView
Gir en tegnbar overflate inne i et visningshieraki.

TabHost
Gir en fanevisning og en *FrameLayout* der visningen knyttet til valgte fane blir vist.

ViewFlipper / ViewSwitcher
Skifter mellom barnekomponentene med fast tidsintervall . En barnekomponent vises av gangen.

Vanlige visninger, barnekomponenter:

- TextView - Tekstvisningsboks
- EditText - Utfyllingsboks. Her kan det defineres hva som skal skrives inn, med tanke på tall, desimaltall, dato, tekst osv.
- Button - Knapp.
- CheckBox - Avkryssningsboks.

- ProgressBar - Fremdriftsstatus.
- SeekBar - Noe lignende av ProgressBar men som gir brukeren mulighet til å velge en verdi.
- RatingBar - Terningkast / Stjerner.
- ImageView - Bildevisning
- VideoView - Videovisning. Kan og kombineres med en *MediaController*.
- DatePicker/TimePicker
- AnalogClock/DigitalClock

4.6.1 Bruk av XML

På androidplattformen blir skjermvisninger stort sett definert i xml-filer. I Eclipse får man tilgang til et redigeringsprogram som gjør det enklere å lage skjermvisningene, når de defineres i en xml-fil. Tekstsnutter, og andre verdier kan også defineres i xml.

For å benytte xml-filene tilbyr applikasjonsrammeverket det som kalles en *ResourceManager*. Når en visning er lastet, kan man aksessere dens komponenter ved definert unik identifikasjon av komponenten. Dette for å endre tekst og lese av manipuleringer gjort av brukeren under kjøring.

4.6.2 Meny

En viktig del av brukergrensesnittet er menyer. Android har to menyer i form av alternativ-meny (options menu, se figur 4.4) og kontekstmeny (context menu). Førstnevnte er en hovedmeny for applikasjonen, der man finner innstillinger og viktig funksjonalitet. Sistnevnte gir en meny for et valgt element i en liste, gjerne *ListView* og gir alternativer for hva man vil gjøre for et gitt element. Alternativ menyen har ikoner. Overskrider menyen et gitt antall elementer, vil de resterende elementene vises på en "mer"(more) meny, uten ikon. Elementene på en kontekstmeny har ikke ikoner.

Menysystemet benytter seg også av xml-filer. Når en meny blir kalt av brukeren, kalles *onCreateOptionsMenu* og *onCreateContextMenu* for henholdsvis alternativmeny eller konteksymeny, der xml-filen leses inn og genererer en visning ved hjelp av en *MenuInflater*. Det her også mulig å legge til eller deaktivere menyalternativer etter behov gitt applikasjonens tilstand.

Når brukeren velger et menyalternativ, kalles henholdsvis *onOptionsItemSelected* og *onContextItemSelected* for alternativmeny og kontekstmeny.

Figur 4.4: Alternativmeny (Optionsmenu).

I disse metodene kan man identifisere hvilke menyalternativ som er valgt, og utføre tilhørende funksjonalitet.

Det er også mulig å definere undermenyer, som åpner seg i det elementet på hovedmenyen som inneholder undermenyen, blir åpnet. En annen funksjon er å gruppere menyelementer og kunne gi dem avkryssningsbokser ect.

4.6.3 Tilbakemelding til brukeren

En annen komponent i brukergrensesnittet er dialogbokser/vinduer, som brukes til å informere brukeren om hva applikasjonen foretar seg, eller avgjørelser brukeren må gjøre for gitte funksjon ect.
Det er tre måter å gi brukeren slike tilbakemeldinger på:

Toast

En tekstboks som gir brukeren en informativ tilbakemelding på en handling som har blitt utført, som vises en kortere på skjermen.

Statuslinje

Det genereres en oppføring i statuslinjen varsler(notifications). Dette alternativet egner seg best for aktiviteter/tjenester som også lever i bakgrunnen.

Dialog
En dialog er et vindu som dukker opp forran aktiviteten på skjermbildet. Her kan brukeren varslet om en laste-prosess, stilles avgjørene spørsmål, ofte besvart med ja/nei eller ok/avslutt, ect. En dialog kan også presentere en liste der brukeren må velge en eller flere elementer på listen. Innholdet i en dialogboks kan defineres med xml-fil, og visningskomponenter og grupper som beskrevet tidligere.

4.6.4 Adapter

Et adapter brukes som en komponent til å binde sammen elementer i en liste og en datastruktur. Det sørger også for å hvordan elementene i en liste skal vises. Komponenter som bruker dette er feks. *ListView* og *Spinner*. Eksempel på adaptere er:

SimpleCursorAdapter
En komponent som kobler et resultat fra en spørring mot leverandør/-database (*Cursor*) med en visningressurs for hvert listeelement. Gir dermed en kort vei fra database og til skjermvisning. Et elements unike _ *id* er tilgjengelig når et element i listen velges.

ArrayAdapter
Denne adaptereren genereren en en-kolonnet liste som viser resultetet fra et kall til*toString* metode på elementene i en array, *ArrayList*, *List* eller elementene i en array xml-ressurs.

Ved spesiellte listevisninger må man selv implementere et adapter for å få ønsket visning. Man implementerer da datastrukteren for listen og metoden som genererer visningen for hvert element i listen.

Kapittel 5

Kode-eksempler

Som en del av innstuderingsfasen var det naturlig å studere noen kodeeksempler, i form av små applikasjoner som benytter seg av funksjonaliteten og elementer fra bibliotekene, for å se hvordan programmering på plattformen skjer i praksis. Noe av android-plattformen refererer også til noen av disse eksemplene og gir der også tips til hvordan implementeringen kan angripes.

Eksemplene har bidratt med inspirasjon til hvordan enkelte problemstillinger kan løses, med tanke på brukergrensesnitt og lager. Da jeg ikke før prosjektet hadde noe bruker eller utviklererfaring med android-platformen, var det naturlig å bruke en del tid på å utforske platformen i form av en android-enhet som jeg fikk lånt. Utforskingen skjedde i form av utprøving av forskjellige applikasjoner, trykke og prøve seg frem. Eksemplene gjorde det mulig å se hvordan enkelte funksjoner og brukergrensesnitt var implementert, der dokumentasjonen deretter kunne oppsøkes for å gi bedre dybde i forståelsen.
Jeg har tatt utgangspunkt i fem applikasjoner/kodeeksempler:

Hello Android Grunnleggende kodeeksempel.

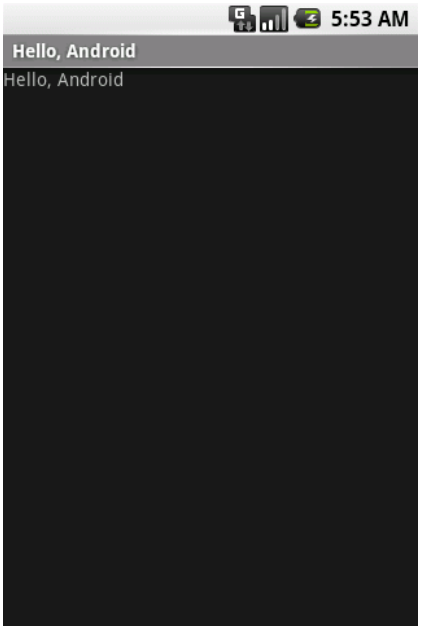
BlueTooth Chat En enkel prateapplikasjon via blåtann.

Notepad En applikasjon som demonstrerer bruk av *ContentProvider* (se seksjon 4.5.2).

Kube En 3D-grafikk-demo som benytter OpenGLES. Viser en rubik's kube [31], med tilfeldig rotasjoner.

Triang Test En enkel grafikk-demo, som også demonstrerer enkel skjermberøringsfunksjonalitet.

5.1 Hello Android



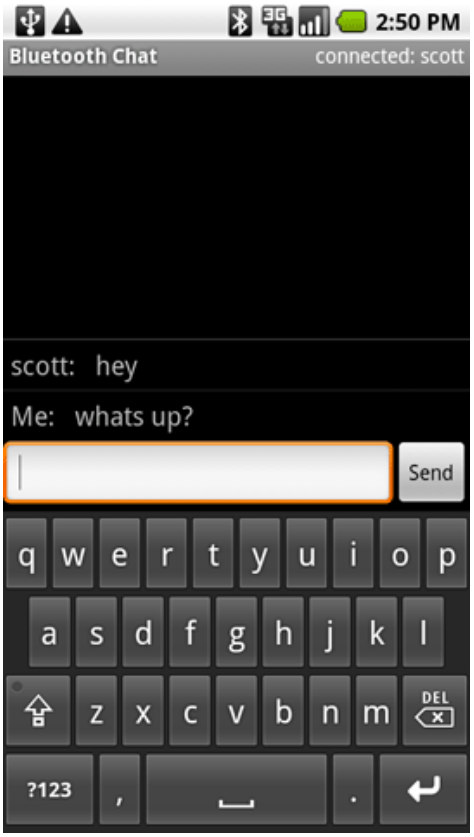
Hello Android er et kodeeksemplen i forbindelse med en grunnleggende guide for nybegynnere på android-platformen. Guiden tar for det første for seg instalasjonen av SDK-en, Eclipse og ADT-plugin (se 4.1.1). Videre beskriver guiden hvordan man oppretter en emulator og oppretter et Android-prosjekt i Eclipse.

Neste steg tar for seg en enkel kodesnutt og hvordan denne kan kjøres på emulatoren, som resulterer i at man får teksten "Hello Android"opp på skjermen.

Guiden gir også en enkel introduksjon til bruk av *ResourceManager* og xml-filer for brukergrensenittet (se seksjon 4.6.1), før den avslutter med en metode for å avluse koden med hjelp av emulatoren.

Guiden er tilgjengelig på `"http://developer.android.com/resources/tutorials/hello-world.html"`.

5.2 BlueTooth Chat



BlueTooth Chat er en enkel blåtann-prateklient, som benytter seg av peer-to-peer oppkobling direkte til en annen enhet. Applikasjonen benytter seg ikke av noe lager, men har en del brukergrensesnitt elementer, der man av koden kan se hvordan disse gis innhold og reagerer på brukerens iteraksjon.

Applikasjonen har også en semi-tjener som ved hjelp av tråder kommuniserer med blåtanngrensesnittet og behandler tilkobling og sending av meldinger mellom to androidenheter. Denne tjeneren er ikke en fullverdig systemtjener som kjører i bakgrunnen på androidplatformen, men instansieres og kjører som egne tråder i bakgrunnen av applikasjonen.

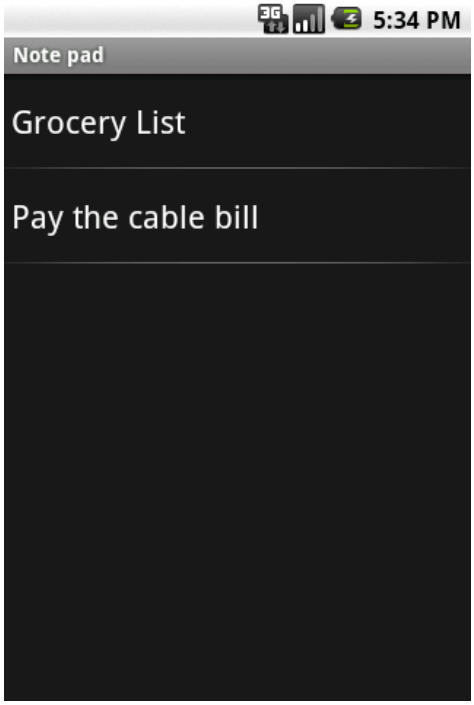
Prateklienten har to aktiviteter, i form av pratevinduet som er hovedaktivitet, samt en egen aktivitet for å velge blåtannenhet det skal kommuniseres med. Hovedaktiviteten har derfor funksjonalitet for å motta resultat fra underaktiviteten, dermed også hvordan resultatet pakkes inn og sendes via applikasjonkonteksten tilbake til hovedaktiviteten.

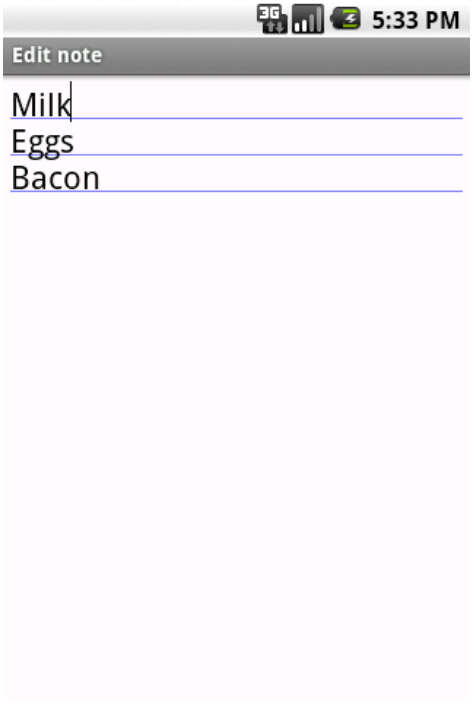
Applikasjonen benytter seg også av *BroadcastReceiver*, som er en klasse som håndterer mottak av informasjon fra sensorer og grensesnitt som er tilgjengelig på Anroid-enheten.

Konfigurasjonsfilen (AndroidManifest.xml, 4.2) viser også hvordan applikasjonen gis tilgang til å motta data og bruke blåtannenheten.

Applikasjonen er tilgjengelig på `"http://developer.android.com/resources/samples/BluetoothChat/index.html"`.

5.3 Notepad





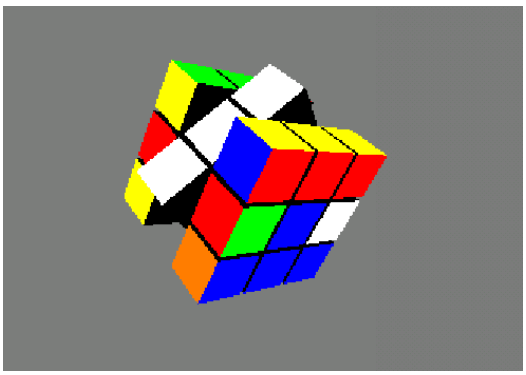
NotePad ser ut som en veldig enkel applikasjon, men tilbyr implementering av en tilbyder (ContentProvider, 4.5.2) som benytter seg av en SQLite-database (se 4.5.1 lokalt på enheten. Med dette demonstreres også funksjonalitet for å behandle Uri-er i tilbyderen.

Applikasjonen viser også hvordan en underaktivitet blir gitt en Intent (se 4.3) og laster gitt element fra databasen med hjelp av tilbyderen, samt flere eksempel på bruk av brukergrensesnittet.

Denne applikasjonen som en del av kommunikasjonen mellom applikasjonens aktiviteter, er det i konfigurasjonsfilen definert flere intent-filtere (se 4.4.1) samt hvordan tilbyderen må deklareres.

Applikasjonen er tilgjengelig på "http://developer.android.com/resources/samples/NotePad/index.html".

5.4 Kube

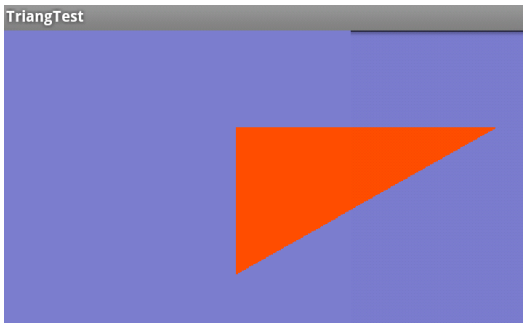


Denne grafikkdemoen benytter seg av OpenGLES rammeverket, samt en *GLSurfaceView* og renderer for å gi en overflate å vise grafikken på. Demoen er relativt komplisert, med tanke på at den for transformasjoner ikke benytter funksjonaliteten i OpenGLES, men heller har implementert sin egen, med matrisemultiplikasjon, transformer og grafikk-elementer-struktur.

Eksemplet viser hvordan man kan endre de delene av en vertexbuffer som faktisk er blitt endret.

Applikasjonen er tilgjengelig som en del av android-eksemplene på "http://developer.android.com/resources/samples/ApiDemos/src/com/example/android/apis/graphics/kube/index.html".

5.5 Triang Test



Dette er et veldig enkelt kodeeksempel som ved hjelp av OpenGLES tegner en trekant på skjermen, som man kan flytter rundt på skjermen ved hjelp av berøringsskjermen.

På grunn av enkelheten, og at koden kun er det som er nødvendig for å tegne en enkel trekant til skjermen, så gav dette en veldig god oversikt på hvordan å bruke OpenGLES på android-platformen.

Kodeeksempelet er hentet fra "http://ruibm.com/?p=263".

Kapittel 6

Tredimensjonell grafikk

Dette kapittelet tar for seg sentrale teknikker og begreper i forbindelse med 3D-programmering. Siste del av kapittelet går også mer inn i detalj på OpenGL ES

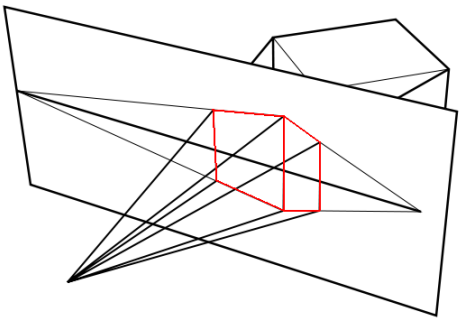
6.1 Projeksjon

Prosjektsjon er teknikk for å gjøre en todimensjonell visning fra en tredimensjonell verden. Et fotografiaparat gjør nettopp dette, ved å avbilde perspektiv til et todimensjonel gjengivning. Figur 6.2 viser hvordan en kube projekteters til en overflate. De røde linjene reprensenterer kuben som et todimensjonelt bilde. I figur 6.1 vises det ferdige resultatet.

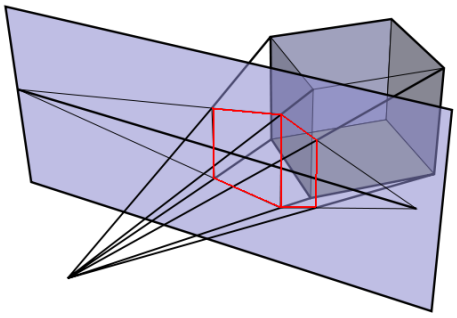
Når en tredimensjonell modell projekteres til en todimensjonell representasjon, i form av en skjermvisning, blir det definert et visningsvolum som setter grenser for hvor nært og hvor fjernt elementer i modellen skal la seg avbilde til den todimensjonelle representasjonen. Prosessen som utfører prosjeksjonen kalles pipeline, se seksjon 6.2.
En projeksjon kan gjøres på to forskjellige måter som skjermgrafikk:

Perspektiv projeksjon
Denne projeksjonen fungerer på samme måten som det menneskelige øyet. Ojekter på stor avstand oppfattes mindre enn objekter som er nære.

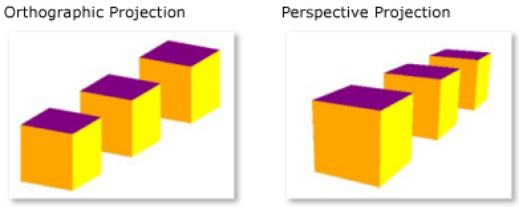
Ortogonal projeksjon
I motsetning til forrige projeksjon, gir ortogonal projeksjon korrekte proporsjoner og muligheten for målbarhet.



Figur 6.1: En projeksjon av et 3D-objekt til en todimensjonell overflate. Avbildningen.



Figur 6.2: En projeksjon av et 3D-objekt til en todimensjonell overflate. Hvordan projeksjonen foregår.



Figur 6.3: Perspektiv- og ortogonalprojeksjon.

6.2 Pipeline

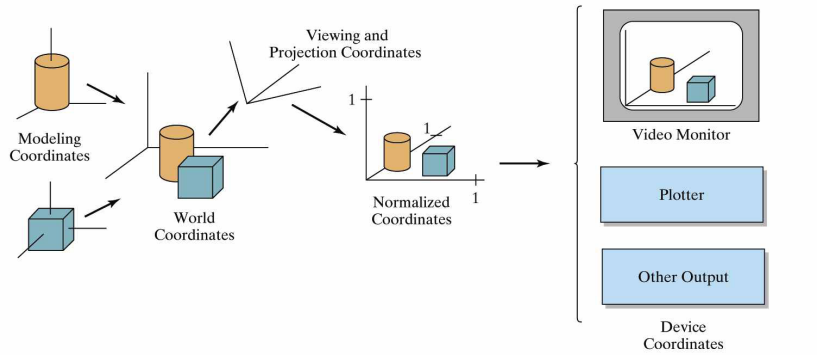
Denne prosessen utfører en serie av transformasjoner som tar et objekt inn i en scene og videre til å bli en projeksjon på en skjerm. På første steg i prosessen ses et objekt på i et lokalt modell-koordinatsystem der objektets overflater gitt ved hjørner blir definert.

Neste steg er å konvertere modell-koordinatene til verdenskooridinater, ved å flytte, rotere og skalere ojektet. Ved å etterpå flytte, rotere og skalere objektene i verdenskoordinatsystemet, blir resultatet en scene med visning og projeksjonskoordinater. Her blir de delene av ojekter og alle objekter som ligger utenfor visningsvolumet klippet bort samt å ignorere objekter som er skjult bak andre objekter.

Deretter normaliseres koordinatene og visningen blir skannet inn med hjelp av en rasterisering til en visning på skjerm.

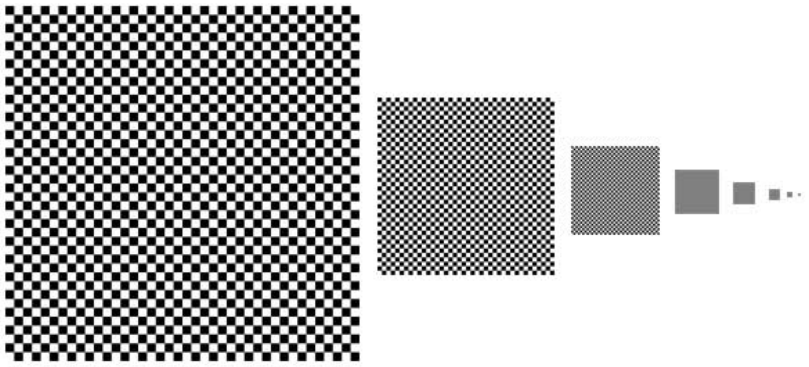
6.3 Texture Mapping

Texture-mapping er en teknikk for å knytte et todimmensjonelt bilde til en overflate hos et 3D-objekt. Teksturbildet blir lagret med forskjellige



Figur 6.4: De forskjellige koordinatsystemene under prosessen med gjøre om en 3D modell til en skjermvisning

oppløsninger, for bruk på nære og fjerne objekter, med en teknikk kalt mipmapping (se figur 6.5). Objekter i det fjerne trenger ikke så detaljerte gjengivelse som orginalbildet for teksturen, og bruker derfor en mindre mipmap. Mipmappene ferdigprosseseres før de lagres, da med tanke på antialiasering, fjerning av falske detaljer slik som svart og hvit støy, noe som kan oppstå ved nedskalering av bilder. Denne forhåndsprosesseringen sparer resurser når skjermbildet skal genereres, noe som gir bedre ytelse.



Figur 6.5: Eksempel på en antialiasert mipmap.

6.4 OpenGL ES

På den mobile platformen, deriblant android, er det støtte for 3D-grafikk ved hjelp av OpenGL ES. Dette er en forenklet implementering av av OpenGl, som er tilpasset systemer med begrensede resurser. Hovedforskjellen

er at OpenGL ES ikke metodene glBegin() og glEnd() [33], som er mye brukt på OpenGL. OpenGL ES versjon 1 er det offesielle 3D-grafikk biblioteket på android.

Grunnen til dette er i følge [20] at disse metodene og måten OpenGL ellers behandler og definerer objekter på er ganske komplisert og krever mye overliggende prosessering. Mellom kallene til de nevnte metoder blir farger, hjørneplassering, teksturkoordinater og overflatenormaler gitt, vanligvis flere ganger for hvert objekt. Et annet moment er også at kallene som skjer for å definere parametre kan skje på forskjellig måte, noe som gir utfordringer i å lage optimal implementering for de nevnt metoder, med tanke på kompleksitet for å takle noe tilfeldig og forskjellig parametersetting.

OpenGL ES sparer mye prosessering under tegningen ved å heller benytte buffere for hjørner, farger, teksturer og normaler. Dette fjerner behovet for mye overliggende prosessering, men gjør det også mulig å laste opp ferdige buffere til en grafikkenhet, uten at internbuser og minne blir en flaskehals for innlastingen av bufferene til grafikkenheten.
Ved generering av bufferene finnes det noen forskjellige tilgjengelige datatyper:

Flytall
: Vanlig 32-biters flytall. Ved eldre androidenheter finnes det ikke nødvendigvis en egen grafikenhet, heller ikke støtte for flytallsregning. I det tilfellet denne datatypen et dårlig valg.

Fixed
: Denne datatypen er et slags flytall, men lagret i form av en 32-biter heltall. 16bit før og etter komma. Dette gjør det også mulig med grafikk på enheter uten flytallsstøtte.

Short
: 16-biters heltall.

6.4.1 VBO - Vector Buffer Object

Fra og med OpenGL ES versjon 1.1 har det vært støtte for bruk av VBO [33]. Dette er en ny forenkling i prosessen med å tegne en skjermvisning fra en 3D-modell, der buffere lastes opp til grafikkenheten for hvert skjermbilde i versjon 1.0. Tanken er at man laster opp bufferene første gang de skal tegnes, og at man senere i prosessen med å tegne ut objektene refererer til den opplastede bufferen. Om en buffer endres, lastes endingen opp til grafikkenheten.

Dermed er det kun flytting, rotering og skalering som i hovedsak gjøres under generering av et skjermbilde, mens bufferene allerede ligger hos grafikkenheten, som tar seg av utregninger og prosesseringen beskrevet i seksjon 6.2.

Dette reduserer belastningen på internbus og systemminne.

Kapittel 7

Annet

Dette kapittelet omhandler noen mindre temaer som har hatt betydning for prosjektet. Dette er DXF – et filformat, og QR-kode – en trend.

7.1 DXF

DXF (Drawing Interchange Format) er et format for design på datamaskiner (CAD) utviklet av Autodesk [26]. Formatet ble introdusert i 1982 som en del av AutoCAD v1.0. Formatet hadde lenge ikke tilgjengelig dokumentasjon for implementasjon for andre aktører, men er nå tilgjengelig hos Autodesk. Det er implementert et tilgjengelig bibliotek for C++, "http://www.qcad.org/dxflib.html".

Etter hvert som AutoCAD har blitt et større og mer omfattende verktøy, har DXF blitt et mindre nyttig format.

Min innfallsvinkel til formatet er at plantegningene til NTNU er tilgjengelige i dette formatet på "http://lydiaweb.itea.ntnu.no/plantegninger/".

Formatet er lesbart i en vanlig tekstredigeringsverktøy.

7.2 QR-kode

QR-kode (Quick Response code) er en matrisebasert strekkode, bestående av et rutemønster, med forgrunn og bakgrunnsfarge, utviklet i 1994 [30]. Intensjonen var en strekkode som raskt lot seg avlese og tolke. Den tilbyr også forskjellige nivå av feilretting.

Bruken av QR-koder er hyppig brukt enkelte steder, men er en trend som oppdages av stadig flere, deriblant meg.

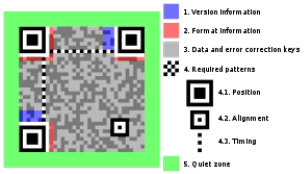
Det er ingen begrensing for hva slags data som kan kodes til en slik

strekkode, se figur 7.1. Den eneste begrensingen gis som en kombinasjon av feilkorrigeringsmulighet, og en øvre grense for antall tegn dataene kan ha, ofte begrenset av programvaren som skal lese av strekkoden.

Noen kjente bruksområder er web-adresser i form av *URL*, tilkoblingsinformasjon for trådløsnettverk eller kontaktinformasjon i form av *vCard*.

Avlesing av dataene lagret i strekkoden kan gjøres med kamera og programvare for å behandle bildet. På Android-platformen er en tilgjengelig applikasjon for å tolke bildet *Barcode Scanner* [5]. Denne applikasjonen benytter kameraet til å lese av strekkoden, for så å vise resultatet og tilby å åpne en applikasjon som har støtte for dataene, om det finnes på systemet.

Figur 7.1: Et eksempel på QR-kode. På androidplatformen kan bla. QR-koder lese med applikasjonen *Barcode Scanner* [5]



Figur 7.2: QR-kode. Forklaring av oppbyggingen av et QR-kode-bilde.

Det finnes flere QR-kode generatorer på internett. En av de er "http://invx.com/"

Del III

Eget Bidrag

Kapittel 8

Krav til applikasjonen

Dette kapittelet definerer noen konkrete krav til applikasjonen som skal utvikles i løpet av prosjektet. Hovedkravet er selvsagt, at applikasjonen skal utvikles på Android-platformen.

Mange av kravene definert i dette kapittelet gjenspeiler krav for fordypningsprosjektet, presentert i kapittel 3, da konseptet er det samme. Den største forskjellen er det nå jobbes med en konkret applikasjon til android-platformen, og at all tid for dette semesteret er tilgjengelig for å jobbe med denne oppgaven.

8.1 Intensjon

En overordnet intensjon er en applikasjon som får tilgang til dataene fra et sentralt distribusjonspunkt. En form for database virker som et fornuftig valg. Da grafikkobjektene mine skal forandre seg under kjøretid og gitt en annen tankegang for OpenGL ES enn hva jeg var vant til fra OpenGL (se seksjon 6.4) kan grafikkelementene ferdiggenereres eksternt for så å lastes ned til enheten og lagres internt der.

Med utgangspunkt i at NTNU har plantegningene side tilgjengelig som DXF-filer (se seksjon 7.1) er det ønskelig med et verktøy for å konvertere disse plantegningene til tredimensjonell grafikk som lagres i eksterndatabasen, tilgjengelig for nedlastning for brukere av applikasjonen. Et slikt verktøy er likevell prioritert lavere enn selve utviklingen på android-platformen, siden det er en av hovedrammene for oppgaven.

8.2 Funksjonelle krav

Denne seksjonen lister funksjonelle krav som jeg har indentifisert som esensielle for applikasjonen. Kravene er delt inn etter tre hovedkategorier:

- **Skjermbilde:** Hva presenteres til brukeren av brukergrensesnitt og grafikk.
- **Datamodell/lager:** Hvordan data skal behandles og lagres.
- **Egne krav til navigasjonspunkter:** Navigasjonspunkter, omtalt som *Node(r)*.

Kravene er vektet etter hvile grad de regnes viktigere enn andre krav, med vektene *Høy*, *Middels* og *Lav*, der de med høy prioritet ses på som viktigst.

Tabell 8.1: Krav til brukergrensesnitt (G). Krav G10 og G11 ble lagt til under implementeringen etter behov.

ID	Kravbeskrivelse	Prioritet
G01	Førstepersonssynsvinkel for kameravisning	Høy
G02	Visning lignende tradisjonell plantegningvisning	Middels
G03	Få objekter til å skille seg ut i visningen sammenlignet med andre objekter.	Middels
G04	Visning av noder	Middels
G05	Vise en rute i kartvisningen	Middels
G06	Tekstlig rutebeskrivelse for en rute	Lav
G07	Menysystem for å vise og aksessere eksterne kilder	Lav
G08	Åpne og vise eksterne data ved hjelp av annen applikasjon	Lav
G10	Algoritme for generering av gulvoverflate	Høy
G11	Algoritme for generering av veggoverflater	Høy
G12	Applikasjonen skal kunne åpnes med hjelp av QR-koder og tilpasse kartvisningen til dette	Middels

Tabell 8.2: Krav for navigasjonspunkter, noder (N)

ID	Kravbeskrivelse	Prioritet
N01	Generere en liste over nærliggene tilgangspunkter	Høy
N02	Knytte en liste tilgangpunkter til en node	Middels
N03	Søke etter rute fra, via og til node	Middels
N04	Redigeringsverktøy for noder	Middels
N05	Detektere beste node gitt tilgangspunkter	Lav

Tabell 8.3: Krav til datamodell og lager(D). Krav D06 ble lagt til underveis i implementasjonen.

ID	Kravbeskrivelse	Prioritet
D01	Data skal kunne lagres i en mysql-database, som sentrallager	Høy
D02	Intern lagring av datamodell, uavhengig av ekstern lager	Høy
D03	Datastruktur for bygning, etasje, rom, rute, node og tilgangspunkt	Høy
D04	Caching av nødvendige objekter som er sannsynlig for å vises på skjermen	Lav
D05	Verktøy for å lese inn DXF-filer, plassere og skallere i verdenen	Lav
D06	Mulighet for å lagre en databaserepresentasjon av plantegninger, der enheten selv kan hente denne representasjonen ved mangel på ferdiggenerert 3D i eksterndatabasen og produsere 3D grafikk av dette.	Høy

8.3 Ikkefunksjonelle krav

I denne seksjonen listes noen ikkefunksjonelle krav for applikasjonen. Det understreks med krav *I01* at android-platformen er hovedrammen for oppgaven.

Kravene prioriteres på samme måte som for de funksjonelle kravene, som beskrevet i seksjon 8.2.

Tabell 8.4: Ikkefunksjonelle krav (I)

ID	Kravbeskrivelse	Prioritet
I01	Applikasjonen skal være implementert for Android-platformen	Høy
I02	Applikasjonen skal bruke OpenGL ES for 3D grafikk	Høy
I03	Datamodellen skal kunne gjenbrukes på andre platformer med Java-støtte	Middels
I04	Enkelt og tilgjengelig brukergrensesnitt	Middels
I05	Tilstrekkelig skjermoppdateringsfrekvens og flyt	Høy
I06	Innlasting i bakgrunnen må ikke fryse grafikken	Middels

Kapittel 9

Arkitektur

Dette kapittelet tar for seg arkitekturen for applikasjonen.

9.1 Mål

En grunnleggende tanke for strukturen var å gjøre denne tredelt i form av en datamodell, et lag for lagring og et visningslag. Videre var det også et mål å la funksjonalitet knyttet til en entitet skulle ligge nært entiteten og der den ble brukt. Et siste mål var at datastrukturen skulle være plattformuavhengig (som gitt i krav *I03* i tabell 8.4), med tanke på behovet for et eksternt verktøy for å redigere og lagre grafikkdata for forskjellige bygninger til eksterndatabasen. Da man et stykke ut i implementasjonsfasen likevel så behoved for plattformspesifikke funksjoner eller bruk av klasser fra platformens bilioteker i en entitet, ble sistenevnte mål nedprioritert til fordel for at funksjonaliteten skulle kunne ligge der den ble brukt. Lokalitet av funksjonalitet ble med andre ord prioritert fremfor platformuavhengighet.

9.2 Biblioteker

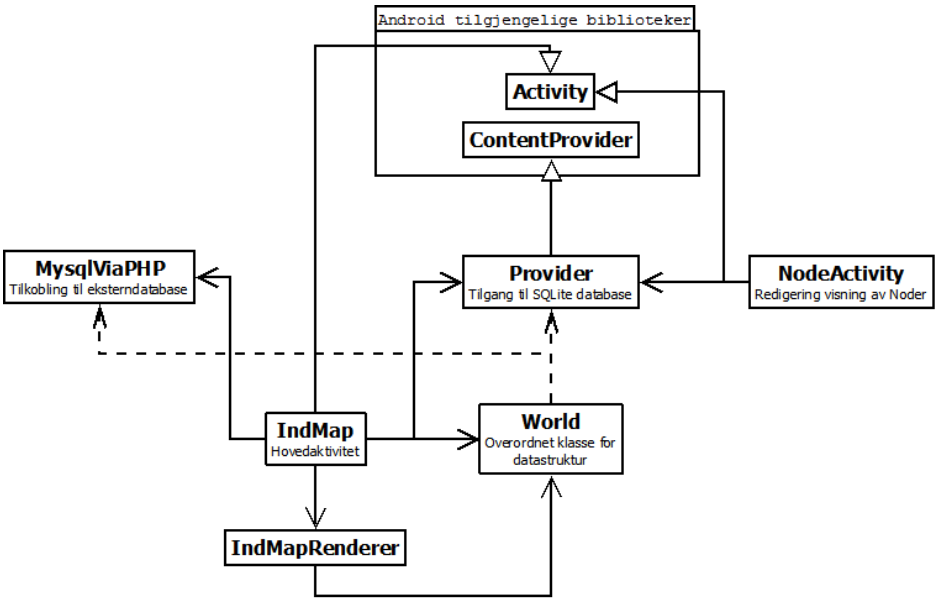
Det benyttes noen hovedbilioteker i implementasjonen. Disse er:

Android
Biblioteker og rammeverk behøvelig for å lage android-applikasjoner.

JSON
Et bibliotek for en tolkbar ren-tekst datastruktur, litt på samme måte som xml. Implementert for de fleste programmeringsspråk og skriptspråk, derfor en fleksibel datastruktur for å kommunisere mellom platformer og utviklingsmiljøer [14].

org.apache.http
: Et rammeverk for å kommunisere med en web-server, i form av http og https protokollene.

9.3 Overordnet oversikt



Figur 9.1: Overordnet oversikt for applikasjonen. Se beskrivele i tabell 9.1

Overordnet sett er det blitt implementert to aktiviteter, *IndMap* og *NodeActivity*, der førstnevnte er hovedaktivitet i applikasjonen og gir kartvisningen, mens sistnevnte er et verktøy for behandling av noder. Sistnevntet blir beskrevet i seksjon 9.8.

I hovedaktiviteten vises en grafisk fremstilling av datastruktturen i form av en tre-dimensjonell kartvisning, der iteraksjon fra brukeren behandles av kontrollfunksjonalitet plassert i *IndMap*, men hovedsaklig i *World*, som også er bindeleddet mellom hovedaktiviteten, renderen (*IndMapRenderer* og datastrukturen. Dette beskrives mer i detalj i seksjon 9.6.

Renderen setter opp OpenGLES miljøet og har overordnet funksjonalitet for hver frame (skjermbilde-visning) ved å aksessere datastrukturen via *World*. Hvordan tegningen skjer forklares mer detaljert i seksjon 9.7.

Som lager vil det ved en førstegangskjøring være nødvendig å laste inn data fra et eksternlager (*MysqlViaPHP*), som laster inn til modellen, men som også lagrer en kopi til det interne lageret i form av en ContentProvider,

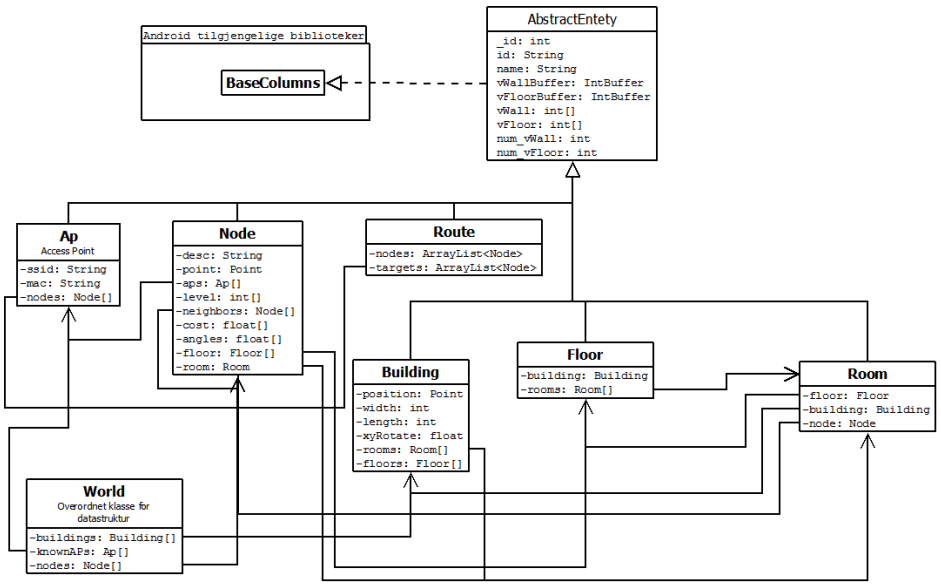
her kalt *Provider* (se 4.5.2).

Med dette sagt, kan man konkludere med at dette er en modell-kontroller-visning arkitektur (MVC [29]).

Tabell 9.1: Overordnet arkitektur, oppsummering

Klasse navn	Beskrivelse
IndMap	Hovedaktivitet. Tilbyr kartvisning,informasjon om nåværende posisjon og lokalisering av nåværende posisjon.
World	Bindeledd mellom visning og datastruktur. Her ligger det meste av kontollfunksjonalitet og styring av verdenen.
IndMapRenderer	Render som setter opp 3D-visningen, og tegner grafikken.
MysqlViaPHP	Innlasting av data i form av spørringen via et php-web-grensesnitt.
Provider	Implementert som *ContentProvider* som gir *World* (via *IndMap*) og *NodeActivity* adgang til internlageret.
NodeActivity	Underaktivitet som tilbyr et redigering og behandlignsverktøy for noder.

9.4 Datastruktur



Figur 9.2: Oversikt datastruktur. Alle entitetene baserer seg på en *AbstractEntety*.

Som et grunnelement i datastrukturen er den abstrakte klassen *AbstractEntety*, som implementerer grensesnittet (interface) *BaseColumns*, noe som gir hovedkolonnen *_ID* som er vanlig i *ContentProvider*-sammenheng (se 4.5.2). Denne abstrakte klassen definerer hovedfelter og funksjonalitet som er lik for alle entitetene. Felt knyttet til grafikk kommenteres i seksjon 9.7.

Klassen *World* rammer inn datastrukturen ved å liste hovedentitetene, der disse igjen har referanser til andre entiteter dypere inn i datastrukturen. Hovedentitene er bygninger, noder og kjente tilkoblingspunkter (Access Points). Nåværende implementasjon behandler ikke en rute som en overordnet entitet, men mer i form av en hendelse eller midlertidig tilstand om brukeren er på en rute (se seksjon 9.6 og 9.7).

En bygning har en eller flere etasjer som igjen består av rom. Legg merke til at elementer dypere inn i datastrukturen har en kobling til overliggende elementer, dette for å gi raskere oppslag, enn om man måtte iverksette et søk. Et rom har også en kobling til en node, om det finnes en node i et rom. Om et rom har flere noder, vil en hovednode være definert for rommet.

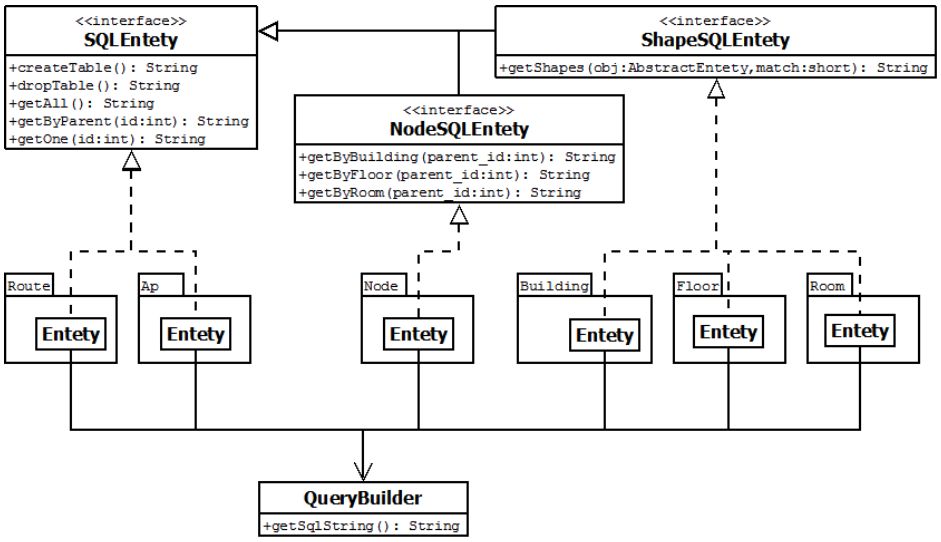
Noder har en kobling til sine naboer sammen meg kostnad og vinkel til naboen der positiv x-akse gir nullpunkt. Det vil si at ligger en nabonode kun lags positiv x-akse fra en node, vil vinkelen være 0 grader. En node holder også informasjon om hvilke tilkoblingspunkter og registrert signalstyrke hvert tilkoblingspunkt har i noden. Et tilkoblingspunkt har også oversikt på hvilke noder det er registrert i. Bruk av dette beskrives i seksjon 10.2.4.

En rute holder en liste på noder den skal fra, via og til, samt en generert liste med noder for å komme seg fra første til siste node.

9.5 Lagring

Siden intenternlageret benytter en *ContentProvider* og den igjen bruker *SQLite* trenger denne SQL-spørringer. Seksjon 9.5.3 forklarer internlageret mer detaljert. Eksternlageret er i grunn en veldig forenklet måte å gir applikasjonen tilgang til en mysql-database som ligger på en server ved at SQL-spørringene sendes fra applikasjonen selv. Seksjon 9.5.2 forklarer eksternlageret mer detaljert.

Da begge lagre benytter SQL-spørringer, var det naturlig å ha en strukturert måte å utføre spørringene på. Hver entitet har derfor en indre klasse, *Entety*, som implementerer et *SQLEntety*-grensesnitt, eller en utvidelse av dette grensesnittet, se figur 9.3. Dette gir hovedfunksjoner for administrasjon og uthenting av data. Da eksternlageret kun blir lest, og en *ContentProvider* opererer med *ContentValues* (se seksjon4.5.2), var det ikke nødvendig å



Figur 9.3: Overordnet lager-oversikt. Entiteten har hver sin innerklasse som implementerer hovedfunksjoner som er nødvendig i et SQL-lagermiljø.

definere funksjoner for å sette inn eller oppdatere data.

For å forenkle og forhindre syntaksfeil i spørringer, er det omplementert en hjelpeklasse, *QueryBuilder* som gir funksjonalitet for å legge til felt, tabeller, filtre, sortering osv, alt hvor en instans av en slik klasse settes som *CREATE*, *SELECT*, osv ved instansiering.

En fordel med *SQLEntety*, er at metoden som oppretter SQLite databasen kan iterere gjennom entitetene og kalle tilhørende *createTable*-metode. Koden for å opprette tabellen er lagret som innerklasse i entitet-klassa, og er dermed samlet på et sted, noe forenklende om det skjer oppdatering på entitetens databasestruktur, da det som må endres ligger hos entiteten selv, i stedet for på forskjellige steder.

Tabell 9.2: SQLEntety. Metodene returnerer en tekstreng som utgjøre SQL-spørringen, ved hjelp av *QueryBuilder*

Metodenavn	Beskrivelse
createTable()	Spørring for å opprette entitetens tabell i SQLite
dropTable()	Spørring for å fjerne tabellen og dens innhold
getAll()	En spørring som gir alle oppføringer i entitetens tabell
getByParent(id)	En spørring som gir alle oppføringer gitt en foreldre-entitet
getOne(id)	En spørring som gir en eksakt oppføring, gitt oppføringens identifikasjon

Da man ble kjent med at de tilgjengelige plantegningene for NTNU ikke akkurat hadde en hendig datastruktur, ble det nødvendig med en støtte for å laste inn 2D-grafikkdata fra ekserndatabasen, gitt i krav *D06* i tabell 8.3. Mer om årsakene til denne endringen utdypes i seksjon 10.1.1.

Tabell 9.3: ShapeSQLEntety. En utvidelse av *SQLEntety* siden entitetene bygning, etasje og rom har tilgang til 2D data i eksternlageret.

Metodenavn	Beskrivelse
getShapes(obj, match)	En spørring for å liste alle fasonger knyttet til en entitet. Parameteren obj er oppføringen som trenger fasonger, og match definerer hvilken type fasong eller fasonger som ønskes.

Tabell 9.4: NodeSQLEntety. En utvidelse av *SQLEntety* siden node-entiteten må kunne listes etter foreldrenode (finne naboer), bygningen den er del av, etasjen den er del av, eller hvilket rom den er del av. Den finner nabonoder med *SQLEnety* sin *getByParent(id)* metode.

Metodenavn	Beskrivelse
getByBuilding	En spørring for å liste alle noder knyttet til en bygning.
getByFloor	En spørring for å liste alle noder knyttet til en etasje.
getByRoom	En spørring for å liste alle noder knyttet til et rom.

9.5.1 Innlasting til datamodellen

Ved oppstart av applikasjonen vil dataene i interndatabasen lastes inn til datastrukturen fra internlageret. Inneholder ikke internlageret noen data, vil brukeren bli anmodet om å koble til det eksterne lageret.

Innlastingen fra de forskjellige lagrene er nokså like, der forskjellene blir redegjort for i de følgende seksjonene 9.5.2 og 9.5.3.

Under innlastingen blir alle bygninger, tilgangspunkter og noder lest inn fra valgte lageret og lagret i et tilhørende objekt av entitetstypen det har. Deretter vil hver bygning laste inn tilhørende etasjer, og etasjene de tilhørende rom. Deretter kobles nabonoder og nodene som tilhører et rom. De innlastede nodene er sortert kronologisk, slik at det er mulig å finne igjen objektet til en node ved å gjøre binærsøk på identiteten til noden. Slik blir nabonode-objekter og node-objekter koblet til sine tilhørende noder og rom. Aktuelle metoder vises i figurene 9.5 for eksternlageret og 9.6 for internlageret.

Nabonodene til en node sorteres vinkel til plasseringen i gulv planet, på samme måten som et kompass der nord er 0°. Nord er i dette kartet definert som positiv x-akse, i datastrukturen. Gulvplanet ligger i x- og y-akse, mens høyden gis langs z-aksen. Nabonodene sorteres etter hvor mange grader rotert mot høyre.

Det må poengteres at OpenGL opererer med et annet koordinatsystem. Her er oppover på skjermen y-akse, sidelengs på skjermen x-akse, og innover i skjermen negativ z-akse. Dette forklares grundigere i seksjon 9.7.

Ved kobling til tilgangspunktene brukes tilsvarende teknikk, bare at tilgangspunktene er sortert på en numerisk representasjon av tilgangspunktenes bssid(MAC-adresse [28]).

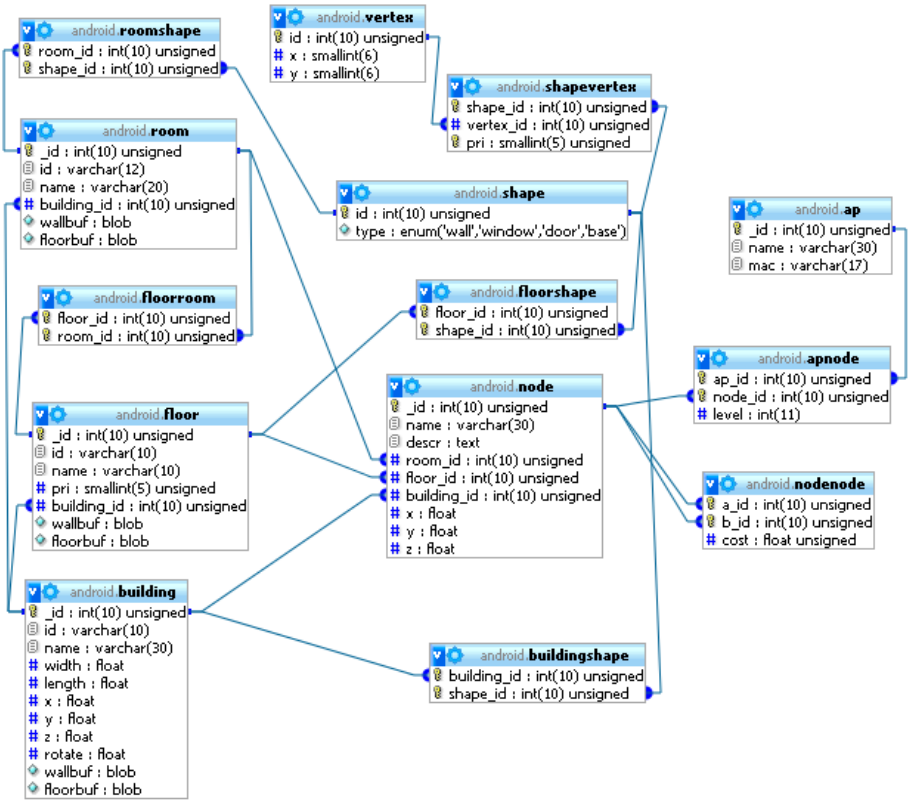
En annen oppgave til innlastingen er å konvertere grafikkdatane i lagrene til grafikkbufre som brukes av OpenGLES. Dette ligger lagret som gulv-grafikk eller vegg-grafikk i form av en *BLOB*, en for hver av grafikk-typene, som i kodemiljøet er tilgjengelig som en byte-array. Disse konverteres til en *integer-array*, som tilordnes til hver sin *IntBuffer*. En mer detaljert forklaring av slike buffere i seksjon 9.7.

9.5.2 Eksternlager

9.5.2.1 Tilgang til dataene

I utgangspuntket var behovet å enkelt laste inn data fra en mysql-database til applikasjonen. Da java-biblioteket som er tilgjengelig på android viste seg å ikke ha en implementering av en mysql JDBC-driver, som har vært måten å koble til en database ved tidligere erfaring med java, ble det behov for å en alternativ løsning som jobbet seg rundt dette problemet. Etter en del prøving og feiling og tid som bare på mystisk vis ble borte, ble en mulig og grei løsning å gi applikasjonen tilgang til databasen via en server i en applikasjon-tjener-database modell.

Tjenerlaget er et php-skript som kjører på en apache-tjener. Mysql-databasen aksesseres fra skriptet, der spørringene i seg selv genereres og sendes til web-tjeneren fra applikasjonen. Applikasjonen mottar en JSON-datastruktur. Dette er ikke en spesielt sikker implementasjon, men var ikke ment som en permanent løsning, men mer som en enkel og rask måte å få tilgang til dataene i mysql-databasen på. Sett i et annet lys, kan tjenersiden vidreutvikles til et samlingspunkt for brukere av applikasjonen, eventuelt også med adgangsbegrensing, der også kryptering av dataforbindelsene kan anbefales. Konseptet bak applikasjonen legger opp til et sentralt distribusjonspunkt, der oppdatering av kart, noder og grafikk tilbys brukeren. Figur 9.4 viser strukturen i databasen.



Figur 9.4: Eksternlager i form av en ER-modell. Viser eksterndatabasen struktur, og relasjoner. (Generert med designer"i PHPmyAdmin)

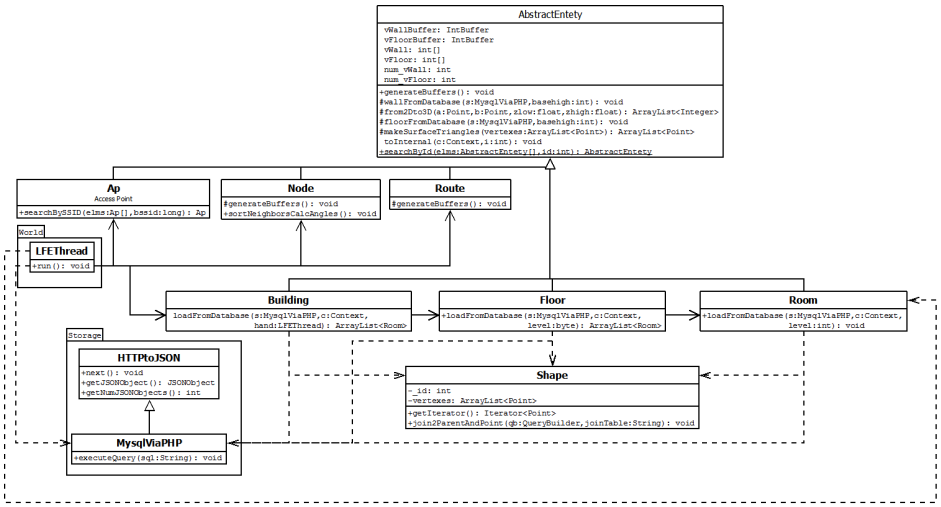
9.5.2.2 Innlasting av data

I seksjon 9.5.1 beskrives hovedfunksjonaliteten ved innlasting både fra intern- og eksternlager. Innlastingen fra eksternlageret har noen flere momenter enn internlageret, i hovedsak å blåse opp 2D grafikkdata til 3D grafikkdata, og å sørge for å lagre dataene fra eksernlageret også lokalt i internlageret.

Øverst i innlastingsprosessen fra eksternlageret, er det implementert en generell klasse (*HTTPtoJSON*) som sender http-forespørsler til en webside, og behandler svaret fra websiden som en JSON-datastruktur. Denne utvides av *MysqlViaPHP* som pakker inn SQL-spørringer som en http-forespørsel. Disse har noe lignende funksjonalitet som det man ville hatt ved en mysql-jdbc driver, med funksjoner for å gjøre kall, og iterere gjennom radene i resultatet.

Innlastingen styres av en egen tråd, kalt *LFEThread* (Load From External Thread). Ved å implementere innlastingsprosessen som en tråd, kunne

brukergrensesnittet informeres med beskjeder til brukeren og status på progresjonen under innlasting.



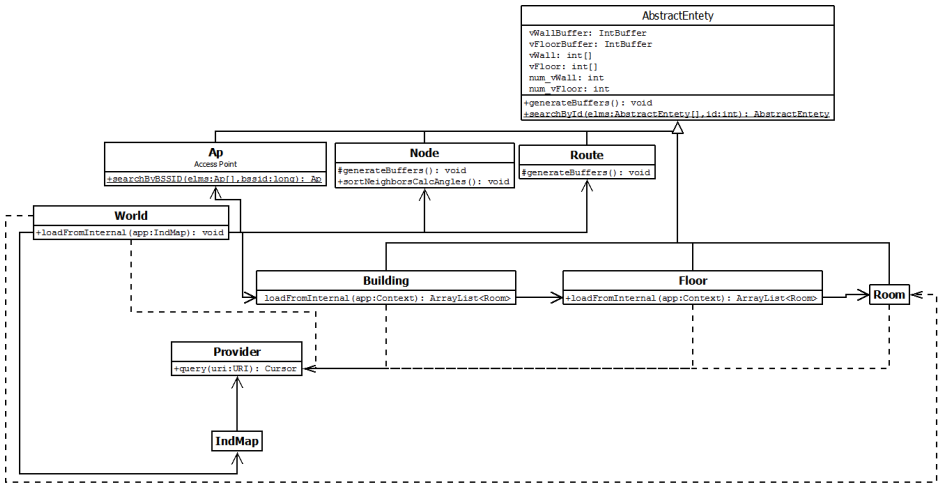
Figur 9.5: Eksternlager.

Hvis en entitet, som har fasonger (Shapes) ikke finner noe ferdig 3D grafikk i databasen (gulv- og veggkantbuffere) benytter den seg av tilgjengelig informasjon lagret som *shape* i databasen (se 9.4). Metodene *wallFromDatabase* og *floorFromDatabase* benytter seg henholdsvis av metodene *from2Dto3D* og *makeSurfaceTriangles*. Fasongene i eksternlageret er hjørner lest inn fra vanlige plantegninger, der sistnevnte metoder henholdsvis blåser opp en vegg, og tegner trekanter mellom gulvpunktene som tilsammen utgjør gulvet. En mer detaljert beskrivelse av disse i seksjon 10.2.3 og 10.2.2, i neste kapittel.

Ferdiggenerert 3D-grafikkdata lagres sammen med sine respektive entiteter til interndatabasen og grafikkbufferene genereres henholdsvis metodene *toInternal* og *generateBuffers()*. *Node* og *Route* har egne implementasjoner av *generateBuffers()* siden de ikke har noe grafikkdata hverken som ferdiggenerert 3D data eller plantegninggrafikkdata. Grafikken for disse entitetene genereres av applikasjonen selv.

9.5.3 Internlager

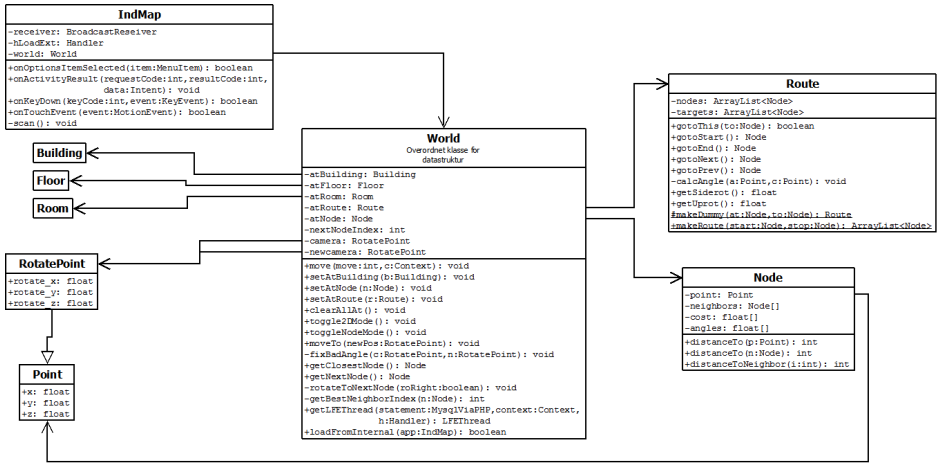
Internlageret er en SQLite-database med tilsvarende modell som eksterndatabasen (se figur 9.4) med har ikke tabeller for *shape* og *vertex* da internlageret opererer med en lagring av ferdiggenerert 3D-grafikkdata. Datatypene er også noe anderledes, som forklart i seksjon 4.5.1, men dette har ikke noen betydning for dataene.



Figur 9.6: Internlager.

Internlageret er implementert i form av en tilbyder (se seksjon 4.5.2), der tilbyderen gjøres tilgjengelig via en aktivitet, i dette tilfellet *IndMap*. En referanse til aktiviteten er derfor gjort tilgjengelig i innlastingsrutinene og underliggende entiteter.

9.6 Kontroll



Figur 9.7: Kontrollfunksjonalitet.

Som et knutepunkt i applikasjonen er klassen *World*. Den holder på datamodellen, kameraposisjoner og tilstandene til applikasjonen. Brukeriter-

aksjon blir derfor rapportert fra *IndMap*-aktiviteten slik at *World* kan oppdatere kameraposisjoner, eller hvilke rom, etasje, bygning, og node brukeren befinner seg ved og i. Knutepunktet innhenter også informasjon fra entitetene for å plassere kameraet riktig, eller oppdatere nåtilstand. Se forklaring til kontrollfunksjoner for kartaktiviteten og knuntepunktet i henholdsvis tabell 9.8 og 9.5.

Tabell 9.5: Kontrollfunksjoner i klassen *World*, se fortsettelse 9.6

Navn	Beskrivelse
move(move)	Mottar informasjon om hva brukeren vil gjøre, ved hjelp av parameteren move. Dette er hendelser som å gå frem/tilbake, opp/ned, venstre/høyre og å rotere.
setAtBuiling(building)	Oppdaterer tilstanden for hvilken bygning brukeren befinner seg i. Definerer også innsynspunkt kameraet må plasseres i for å gi en visning ovenfra av bygningen.
setAtNode(node)	Oppdaterer tilstanden for hvilken node brukeren befinner seg i. Kameraet vil også bli flyttet til posisjonen til gitte node.
setAtRoute(route)	Setter applikasjonen i rute-modus ved gitt rute. Se tabell 9.7 for kontrollfunksjonalitet ved rute-modus.
clearAllAt()	Nullstiller tilstander for bygning, etasje, rom, node og rute.
toggle2DMode	Aktiverer eller deaktiverer om verdenen eller en bygning skal vises ovenfra eller ikke.
toggleNodeMode	Aktiverer eller deaktiver om kameraet skal flyttes mellom noder, eller kunne flyttes fritt i rommet.

Tabell 9.6: Fortsettelse.. Kontrollfunksjoner i klassen *World*

Navn	Beskrivelse
moveTo(newPos)	Brukes til en glidene forflytting av kameraet fra nåværende posisjon til en ny posisjon, der også kameraretning gis.
fixBadAngle()	Forebygger at kameraet roterer over 360 grader unødvendig.
getClosestNode()	Gir nærmeste node til nåværende posisjon
camera	Nåværende kameraposisjon.
newcamera	Nåværende sanntidsposisjon til kameraet, også under veis i en glidende flytting fra et punkt til ett annet.
getNextNode()	Gir neste nabonode ved nodemodus.
nextNodeIndex	Holder indeks til nåværende neste nabonode.
rotateToNextNode()	Roterer kameraet til neste nabonode, som også blir neste node *getNextNode* oppgir. Her benyttes sorteringen av nabonodene som beskrevet i seksjon 9.5.1, for å kunne rotere til neste høyre- eller venstreliggende nabonode.
getBestNeighborIndex()	Gir indeks til beste nabonode i en node kameraet beveger seg til, gitt vinkelen kameraet har. Den nabonoden som er mest mulig rett frem i forhold til kameraets rotasjon, blir valgt.
getLFEThread()	Gir tilgang til innlastingen fra eksternlageret om brukeren ber om det. Progresjonen i innlastingen rapporteres til *IndMap* aktiviteten gjennom dens *hLoadExt*-håndterer.
loadFromInternal()	Metoden som laster inn datastrukturen fra internlageret. Kalles om noderedigeringsaktiviteten har gjort noen endringer.

Når applikasjonen er i nodemodus tar knutepunktet seg oppdaterer nåtilstanden til en ny node, samt flytte kameraet i en glidende bevegelse til neste node. Metoden *rotateToNextNode()* (se tabell 9.5) kontrollerer hvilken node som vil bli den neste kameraet flytter seg til, mens *getNextNode()* leder til en faktisk bytte av nåværende node.

Er en rute aktivert blir en rutes egne kontrollfunksjoner brukt for å gi noden kameraet skal beveges til (se tabell 9.7). En rute tilbyr funksjonalitet for å få riktig rotasjon ved flytting mellom to punkter, noe som også utnyttes ved hjelp av *makeDummy* når ikke noe rute er aktivert og det navigeres

mellom noder.

Tabell 9.7: Kontrollfunksjoner i klassen *Route*

Metodenavn	Beskrivelse
gotoThis(node)	Bekrefter om en node finnes i ruta og setter dette som nåværende punkt i ruten.
gotoStart()	Setter ruten til startnoden og returnerer denne.
gotoEnd()	Tilsvarende som *gotoStart* men med siste node i ruten.
gotoNext()	Går til neste node på ruten.
gotoPrev()	Går til forrige node på ruten.
calcAngle()	Regner ut vinkler for neste node på ruten
getSiderot()	Gir siderotasjon funnet med *calcAngle* som trengs for å rotere kameraet riktig horisontalt.
getUprot()	Gir vertikal rotasjon funnet med *calcAngle* som trengs for å rotere kameraet riktig vertikalt.
makeDummy()	Utnytter funksjonalitet i rute-klassen ved flytting mellom noder, når det ikke er aktivert noe rute.
makeRoute()	Finner en rute fra en node, eventuelt via en eller flere noder og til en sluttnode.

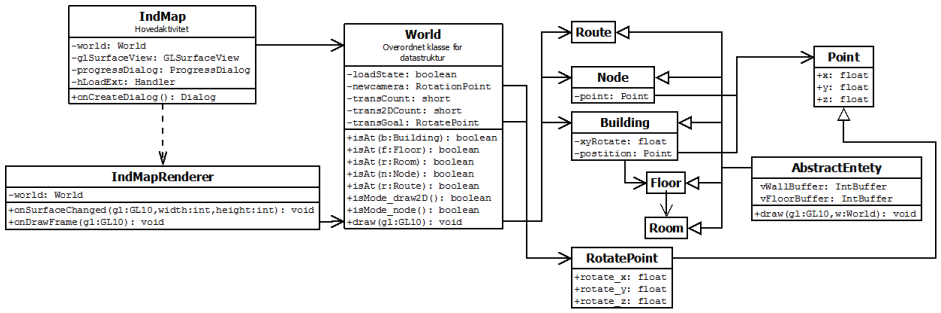
Noe kontrollfunksjonalitet finnes det også i kartvisningen, *IndMap*-aktiviteten, men dette kalles stort sett kontrollfunksjonalitet i *World* klassen, så lenge det ikke er snakk om å åpne noderedigerings-aktiviteten og endringer på brukergrensesnittet som ikke har noe med kartvisningen og datastrukturen å gjøre.

Om brukeren ber om en lokalisering, vil aktiviteten selv ta seg av dette og vise resultat fra operasjonen, helt frem til en node velges og et kall sendes til *World*. En beskrivelse av lokaliseringen finnes i seksjon 10.2.4.

Tabell 9.8: Kontrollfunksjoner i klassen *IndMap*

Metodenavn	Beskrivelse
onOptionsItemSelected()	Behandler brukerens valg på alternativer-menyen. Flere av disse alternativene kaller funksjoner i *World*-klassen.
onActivityResult()	Behandler responser fra noderedigeringsprogrammet, enten ved valg av node som skal vises, generert rute eller om endringer har blitt gjort slit at datamodellen er invalid og må lastes inn på nytt.
onKeyDown()	Registrerer knappetrykk og ber *World* om å oppdatere kameraposisjon osv.
onTouchEvent()	Registrerer berøringer av skjermen.
onClick()	Registerer trykk på knapper i brukergrensesnittet.
scan()	Begynner et nytt søk etter tilgangspunkter.

9.7 Grafikk



Figur 9.8: Metoder og felt knyttet til tegning av kart-visningen. Utdypende forklaring av funksjonene gis i tabellene 9.9, 9.10 og 9.11.

Grafikken til kartvisningen vises på en overflate av typen *GLSurfaceView*, der det defineres en rederer for denne overflaten, her i form av *IndMapRenderer*. Renderen initialiserer OpenGLES miljøet, setter opp visning, projeksjon osv. For hver skjermbilde gjøres de nødvendige nullstillinger i renderen, før *draw(GL10 gl)* kalles i *World*-klassen. Her blir kameraet plassert i verdenen, før entitetenes kalles til å tegne seg selv.

Alle entitetene arver draw-metoden fra *AbstractEntety*, som også gir vertexbufferene *vWallBuffer* og *vFloorBuffer*, se figur 9.8. Ved kall til entitetenes draw-metode, vil entitetene sjekke sin entitets *isAt(<entitetstype>)*, og *isMode..()*-metoder (se tabell 9.11) siden dette avgjør fargevalg og om en gulv eller vegger for entiteten skal tegnes eller ikke.

Tabell 9.9: Metoder og felt i forbindelse med grafikk i klassen *IndMap*

Navn	Beskrivelse
world	Referanse til knutepunktet *World*
glSurfaceView	Referanse til kartvisningen, der renderen *IndMapRenderer* virker.
progressDialog	Referanse til dialogen for innlastingen fra eksternlageret.
hLoadExt	Koblingspunkt til innlastingsfunksjonaliteten, der tekstbeskjeder og progresjon rapporterestilbake fra innlastingsprosessen.
onCreateDialog	Tegning av innlastingsdialogen.

Tabell 9.10: Metoder og felt i forbindelse med grafikk i klassen *IndMapRenderer*

Navn	Beskrivelse
world	Referanse til knutepunktet *World*
onSurfaceChanged	Kalles ved start og når overflaten det tegnes til endres, feks når android-enheten fysisk vris 90 grader.
onDrawFrame	Kalles for hver skjermoppdatering.

Når kameraet flytter seg vil posisjon og rotasjon gis ved

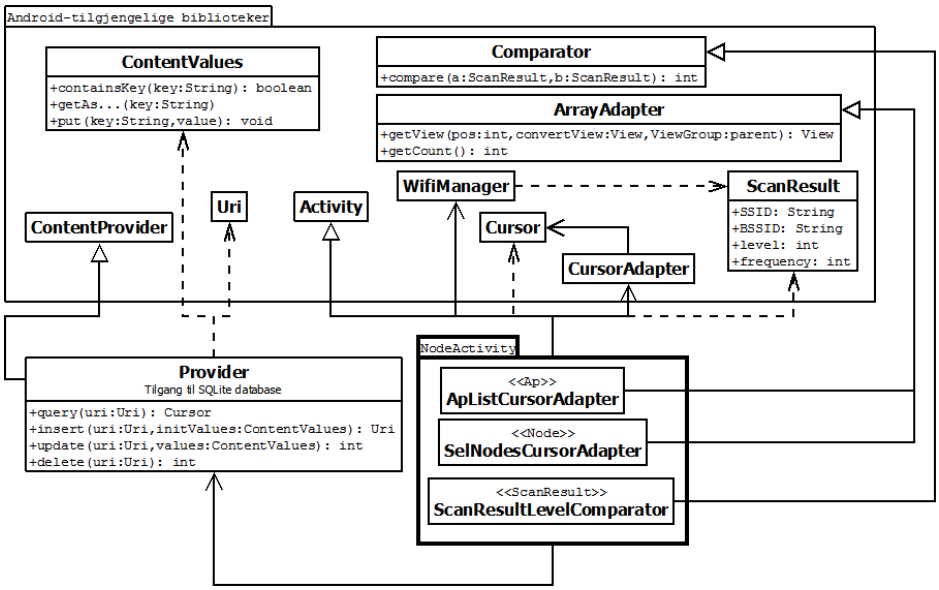
$$newcamera = camera\ \frac{max - i}{max} + transGoal\ \frac{i}{max}$$

der *newcamera* til enhver tid har nåtids posisjonen og rotasjon til kameraet, mens *camera* holder posisjon og rotasjon for forrige punkt, og *transGoal* oppgir posisjon og rotasjon dit kameraet skal flyttes til.

Tabell 9.11: Metoder og felt i forbindelse med grafikk i klassen *IndMap*

Navn	Beskrivelse
loadState	Et flagg for om applikasjonen laster data slik at ikke noe bør tegnes.
newcamera	Til en hver tid nåposisjonen og rotasjon til kameraet.
transGoal	Ny posisjon og rotasjon når kameraet er ferdig med å gli fra et punkt til et annet.
transCount	Teller for progresjon med å gli fra et punkt til et annet. Går fra 0 til *TRANS_LENGTH*.
trans2DCount	Tilsvarende som *transCount*, men gir progresjon til og fra visning ovenfra.
isAt()	Gir entitetene mulighet til å kontrollere om man befinner seg hos dem eller ikke.
isMode_draw2D()	Gir entitetene tilstanden for visning ovenfra, slik at fargevalg og eniteten skal tegnes tilpasses.
isMode_node()	Samme som for *isMode_draw2D()*, men gir tilstanden for fri bevegelse eller flytting mellom noder.
draw()	Plasserer kameraet i verdenen og kaller tegning (*draw()*) for alle entiteter.

9.8 Node redigerer



Figur 9.9: Aktivitet for å redigere og behandle noder..

Kapittel 10

Implementasjonen

Dette kapittelet omtaler uviklingsfasen, utfordringer man møtte på under implementeringen, spesielle algoritmer som fortjener mer omtale, og til slutt litt om brukergrensesnittet.

10.1 Utviklingsfasen

En del av utviklingsfasen er de definerte kravene. En helt annen side er hvordan implementasjonen skal angripes og faktisk blir gjort. Under implementasjonsfasen ble det ført en liste, der nye elementer ble valgt fra kravene og lagt til listen. Dette gir et godt bilde av rekkefølgen for implementasjonen. Oppføringen satt i kursiv viser planlagte nye elementer, men som det ikke ble tid til å implementere.

1. Datastruktur, Overordnet design klasser og objekter.
2. Testdata ekstern database.
3. Laste inn data fra eksterndatabase.
   (a) Snakke med database.
   (b) Laste inn til datastrukturen.
   (c) Opprette internlager.
   (d) Lagre og laste inn fra interndatabase.
4. Laste inn data fra database og lagre til intern lagring.
   (a) Opprette internlager.
   (b) Importere fra eksternlager.

   (c) Laste inn til datastrukturen.

5. Generer OpenGL ES grafikk.

   (a) Tegne kube som kan brukes som markør for noder.

   (b) Tegne fra datastruktur.

   (c) Gjør om last inn som egen funksjon, som nullstiller intern lager og importerer på nytt. Krever også mulighet til å laste fra intern til datastruktur, uten behov for ekstern.

6. AccessPunkt, legge til Node

   (a) NodeActivity. Legg til ny node, høyrekant:

      - Accesspunkt liste.
      - Bruk av Provider

   (b) Lese Accesspunkter, lagre til node.

   (c) Sammenligne scanresult med noder.

7. Rute

   (a) Tegne link til noder

   (b) Navigere langs rute

   (c) Navigere ved hjelp av link mellom noder

8. Oppgradert grafikk

   (a) Sette farger på valgte elementer/gjennomsiktighet

   (b) 2D Mode, visning ovenfra

   (c) *Støtte for både landscape og portrait skjermvisning (ved vridning på skjerm)*

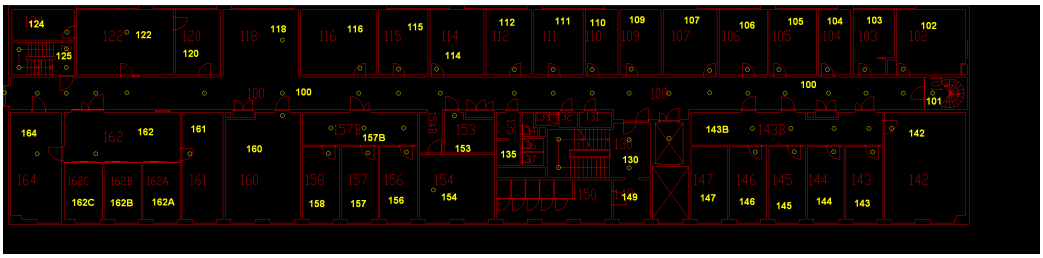
   (d) *Utnytte enhetens egen grafikkenhet bedre(laste opp vertexbuffere en gang)*

   (e) *Settings for at brukeren kan velge farger selv (egne fargeoppsette)*

9. *Eksterne datakilder*

10. *Verktøy lese inn vektorgrafikk, scalere i forhold til eksisterende bygninger/objekter, lagre til database.*

Utviklingen skjedde på en Galaxytab, GT-P1000.



Figur 10.1: Plantegning for 1.etasje IT-vest. Noder er markert med gul, hørner er markert som blå-piksler.

10.1.1 Skuffende plantegninger

Den første utfordringen i implementasjonsfasen var å kunne ta i bruk plantegningene for NTNU sine bygg. Jeg valgte å ta et utgangspunkt i it-vest bygget og la dette bygget være en start, blant annet på grunn av en nokså systemmatisk bygningsmasse, uten masse rariteter og duppeditter, noe arkitekter har sine egne ord for.

Filformatet *DXF* (se seksjon 7.1) virker som en oversiktlig og greit format, der selv uten et rammeverk for å akessere filen kan implementere metoder som leser filstrukturen og laster det inn til datastrukturen min. Etter en nærmere betrakting av hva plantegningene faktisk hadde å tilby, ble det oppdaget at disse seg i mellom hadde helt forkjellig inndelinger og skaleringer, men aller værst det at veggene til alle rom lå i samme lag.

Konklusjonen ble at en implementasjon av et verktøy for å lese inn filene, og få delt opp grafikkdatane til å passe min datastruktur ville bli en tidkrevene oppgave. På grunn av forskjellig skalering og navngiving, ville det også vær nødvendig med et brukergrensesnitt med mye funksjonalitet for å vise frem de importerte dataene, og å sortere de til ønsket entitet. Et slik verktøy ville eventuelt måtte implementeres for pc-platformen, men mulighet til å lagre til databasen som android-applikasjonen kobler seg til. Dette brøt med oppgavens hovedfokus om android-platformen, og det virket åpenbart at tidsforbruket ville være for omfattende.

Løsningen ble modifisert eksterndatabase, men mulighet til å legge til 2D-data fra plantegningene manuelt. Førsteetasje for it-bygget ble lagt inn i databasen, i form av 620 hjørner(x og y koordinat), som igjen ble linket til 347 fasonger (shape). Arbeidet ble regnet som nødvendig for å få tilgjengelige data å kunne vise frem på applikasjonen. Som en del av dette arbeidet ble det skrevet flere SQL-skript og spørringer som forenklet og akselrerte prosessen betydelig.

Figur 10.1 viser den behandlede plantegningen.

10.1.2 Manglende Mysql-driver.

Android sine Java-biblioteker er ikke komplette. At de ikke hadde en mysql-driver, som ville gjort det enkelt å laste inn data til applikasjonen fra en eksterndatabase, er forståelig med argumenter som klient-server-database – arkitekturen, og det at en mobil enhet har en trådløs, potensiell usikret tilkobling.

Løsningen ble *JSON* og noe php-kode som beskrevet i seksjon 9.5.2.1.

10.1.3 Annen tankegang i OpenGL ES

Metoder og funksjonalitet man var vant til fra OpenGL viste seg å ikke eksistere i OpenGL ES, som beskrevet i 6.4. Løsningen ble ganske enkel og elegant. En vegg- og en gulv-grafikbuffer for hver entitet, som vist i seksjon 9.4.

10.2 Spesielle funksjoner

Denne seksjonen tar for seg funksjonalitet som fortjener noe omtale.

10.2.1 Bruk av QR-kode

QR-kode ble en inoativ forfriskning for applikasjonen, og god gadget-faktor i en demosammenheng. Det kan også kompansere for manglende utprøving, justeringer og kalibreringer for tilgangspunkt/node-lokaliseringsfunksjonaliteten, ved at plassering kan identifseres ved hjelp av slike strekkoder.

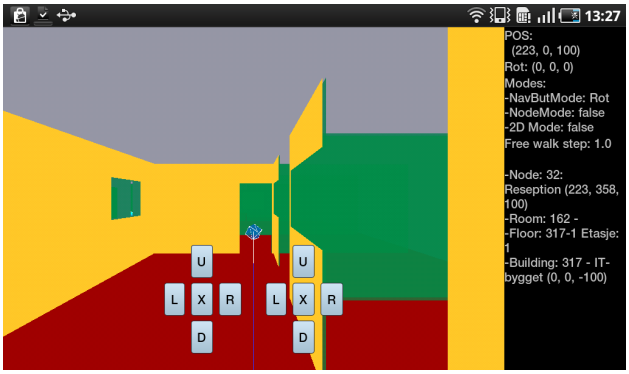
10.2.2 Gulvgenereringsalgoritme

Da OpenGL ES ikke tegner polygoner, men kun linjer og treangler, var det nødvendig med en algoritme som kunne jobbe rundt dette.

Løsningen ble en algoritme som går langs hjørnene $x_0, x_1, \ldots, x_n$ til en overflaten, der tre og tre hjørner undersøkes av gangen. Er vinkelen mellom vinkelbenene $x_i\vec{x}_{i+1}$ og $x_{i+1}\vec{x}_{i+2}$ under 180°(konveks), kan det tegnes en trekant, dersom det ikke ligger et fjerde punkt x_j innen for trekanten, gitt at $x_j \neq x_i, x_j \neq x_{i+1}, x_j \neq x_{i+2}$. En konvekst vinkel finnes med kryssprodukt, som vist i figur 10.3 gitt

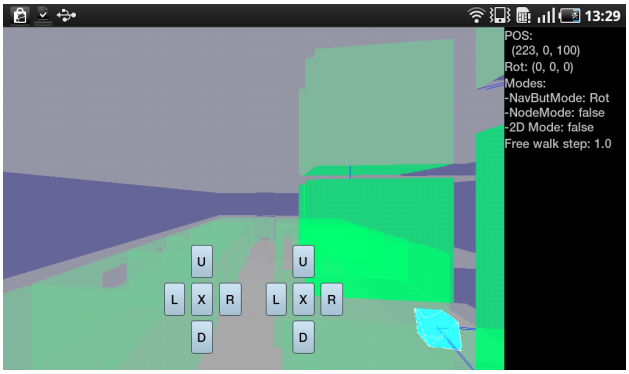
$$\begin{aligned} \vec{n} &= \vec{BA} \times \vec{BC} \\ &= \vec{u} \times \vec{v} = [y_u z_v - z_u y_v,\ z_u x_v - x_u z_v,\ x_u y_v - y_u x_v] \end{aligned}$$

(a) QR-kode for en node



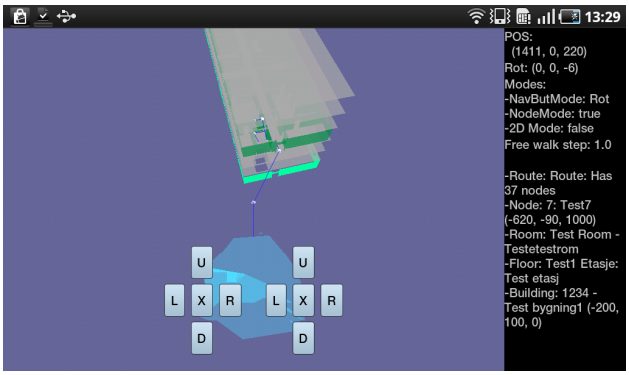
(b) Visning av noden, der QR-koden gir `indmap:node:32`

(c) QR-kode for et punkt og synsvinkel



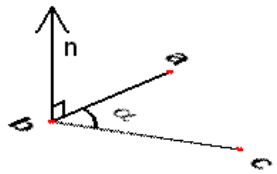
(d) Visning av punktet, der QR-koden gir `indmap:goto:1411:253:220:0:0:-6`

(e) QR-kode for en rute



(f) Visning av ruten, der QR-koden gir `indmap:asRoute:7&1&8&10`

Figur 10.2: Støtten for QR-kode som implementasjonen gir.



Figur 10.3: Kryssprodukt. Om vinkelen er konveks er kryssproduktet (normalt på $\triangle$**abc**-planet) i positivt retning. En konkav vinkel gir negativt retning

Da man betrakter vinkelen i gulvplanet vil z-koordinaten for $\vec{BA} = \vec{u}$ og $\vec{BC} = \vec{v}$ kunne ses på som 0. Dermed får man:

$$\begin{aligned} \vec{n} &= \vec{BA} \times \vec{BC} \\ &= \vec{u} \times \vec{v} = [0,\, 0,\, x_u y_v - y_u x_v] \end{aligned}$$

Med dette kan man utnytte z-delen av kryssproduktet til å klassifisere om vinkelen er konvekst eller ikke:

$$x_u y_v - y_u x_v \begin{cases} > 0 & \text{hvis } 0° < \alpha < 180° \\ \leq 0 & \text{hvis } 180° < \alpha \end{cases}$$

Et hjørne som er ferdigtegnet merkes som ferdigtegnet, mens hjører som er passert, men som ikke er ferdigtegnet, vil kombineres med andre hjørner som ikke er ferdigtegnet, for også der generere trekanter. Se figur 10.4 som viser hvordan algoritmen går frem.

Sjekken som finner ut om det finnes et fjerde punkt som ligger i trekanten, har jeg latt meg inspirere av [3].

10.2.3 Veggenereringsalgoritme

Veggene genereres ved å gå lang gulvlinjen til veggen, og blåse den opp med definerte høyder for vegger, vindu og dører. Skaleringen applikasjonen bruker definerer en meter som 38, som oppleves som normal proporsjon til dataene lastet inn fra plantegningen til 1. etasje, IT-vest-bygget. Figur 10.5 viser hvordan trekanter for veggene blir funnet.

10.2.4 Lokaliseringsalgoritme

Lokaliseringsalgoritmen finner nærtliggende tilgangspunkter som er godkjente. Deretter søker den opp noder som har registrerte signalnivåer for disse

tilgangspunktene, og regner ut en verdi for hver node som antyder hvor godt de passer til nåværende posisjon. Denne verdien regnes ut ved å sammenligne registerte signalnivåer til alle tilgangspunktene som er registert i en node (der er satt et tak på hvor mange tilgangspunkter en node registrerer). Verdien er gitt ved

$$\text{verdi}(node_i) = \sum_{j=0}^{n} (f(j) - h(i,j))^2$$

der i er en viss node, j et tilgangspunkt, $f(j)$ gir nåværende signalstyrke, og $h(i,j)$ gir tidligere signalstyrke som er registrert.

Som en vidreføring er det også vært tenkt på å tilføre straff til utregnet verdi, om et tilgangspunkt en node er registrert på ikke blir funnet blant tilgangspunktene som ble funnet for nåværende posisjon.

10.3 Brukergrensesnitt

10.3.1 Kartvisning

Hovedaktiviteten i applikasjonen er en kartvisning med mulighet for å få vist informasjon om nåværende posisjon, med tanke på bygning, etasje og rom. Visningen gir et sett navigeringsknapper, der knappene på venstre side gir mulighet til bevegelse forrover, bakover og til sidene. Knappene på høyre side styrer rotasjon av kameraet.

Midtknappen blant knappene til venstre, aktiverer og deaktiverer node-modus. Det vil si at flyttingen skjer langs kartets utplasserte noder, eller langs nodene i en rute, om en rute er valgt. Frem-knappen flytter kameraet til neste node, mens sideknappene gjør det mulig å velge andre noder å gå til, enn den som er valgt. Kameraet roterer til valgte neste-node. Bakoverknappen tar kameraet tilbake til forrige node. Bakoverfunksjonen husker bare forrige node, altså ikke node før forrige node. Ved aktivering av node-modus vil kameraet flyttes til nærmeste node.

En annen visning får ved å aktivere 2D-modus. Dette gir en visning ovenifra, noe som kan minne om en tradisjonell plantegningsvisning. I denne modusen blir personen reprensentert av en svart figur, som reagerer på de samme kommandoene fra navigeringsknappene som ved vanlig visning. Visningen viser kun den etasjen personen befinner seg på.

Ved vanlig visning, det vil si ikke 2D-modus eller node-modus, kan knappenes funksjon byttes. Aktivert funksjon vises på side-oversikten som *Rot*-rotasjon, eller *Z*-bevegelse i høyderetning. Tilstanden byttes ved høyre midt-knapp. 2D-modus aktiveres fra alternativ-menyen.

Om *Z* er aktiv kan ganghastigheten ved fri-bevegelse endres med venstre og høyre knapp på høyre side, og opp og nedknappen vil flytte kameraet oppover eller nedover. I denne tilstanden vil venstre midtknapp nullstille valg av bygning, etasje og rom.

Alternativet "My position"vil gi en liste over tilgangspunkter som er lagt til som godkjente tilgangspunkter i applikasjonen og deres signalstyrke, samt gi en liste over noder som passert til de tilgangspunkter som er funnet og deres signalnivå, som beskrevet i seksjon 10.2.4.

Figur 10.2b viser visning i node-modus, figur 10.2d viser vanlig visning, mens figur 10.2f viser node-modus med aktivert rute. Figur 10.6 viser 2D-modus.

10.3.2 Noderedigeringsvertøy

Denne aktiviteten ble implementert for å hovedsaklig kunne tilordne noder en registrering for nærliggende tilkoblingspunkter. Se figur 10.7.

Aktiviteten gir til høyre mulighet til å redigere lagret informasjon om en node. Oppe til venstre er en liste der noder kan legges til og kobles sammen som nabonoder.

Nede til venstre er en liste over alle noder i systemet. Man kan liste noder som ikke er tilknyttet noen bygning, eller vise noder etter bygning, etasje og rom.

Det er mulig å velge en node som blir returnert tilbake til kartvisningen, som straks flytter seg til valgte node. Valgte noder-listen gir også mulighet til å vise de valgte nodene som en rute i kartvisningen.

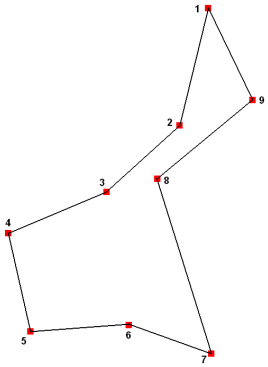
10.4 Gjenstående

Alle høyt og middels prioriterte funksjonelle krav og ble implementert. Krav N03 ble implementert, men ble ikke tilstrekkelig testet til få regnes som helt ferdig. Den mest omfattende gjenstående komponenten, er innhenting av data fra andre eksterne systemer, og vise dette satt i en sammenheng med objekter i kartet. Klassen *HTTPtoJSON* gir funksjonalitet som kunne ha blitt brukt til dette formålet, men ingen datastruktur, håndering eller brukergrensesnitt er implementert for dette. Tankeprosessen var i gang, noe arbeidsprogresjonen vist i seksjon 10.1 viser.

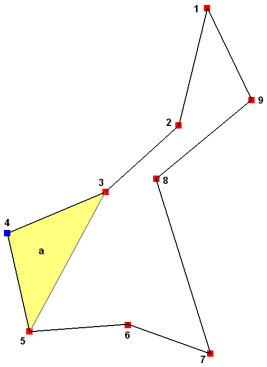
Jeg må innrømme at en del grafikk-relatert funksjonalitet som styrker demo-effekten har blitt prioritert noe høyere under implementasjonen, enn opprinnelig tenk.

Tabell 10.1: Krav til applikasjonen som ikke har blitt implementert

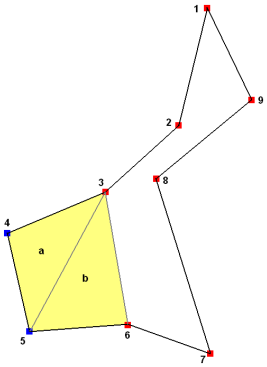
ID	Kravbeskrivelse	Prioritet
G06	Tekstlig rutebeskrivelse for en rute	Lav
G07	Menysystem for å vise og aksessere eksterne kilder	Lav
G08	Åpne og vise eksterne data ved hjelp av annen applikasjon	Lav
N03	Søke etter rute fra, via og til node	Middels
N05	Detektere beste node gitt tilgangspunkter	Lav
D04	Caching av nødvendige objekter som er sannsynlig for å vises på skjermen	Lav
D05	Verktøy for å lese inn DXF-filer, plassere og skallere i verdenen	Lav



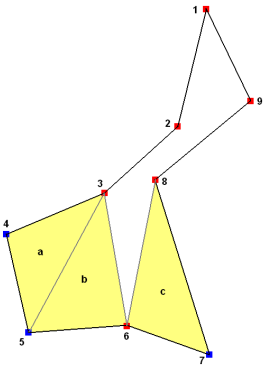
(a) Søker fremover etter konveks vinkel.



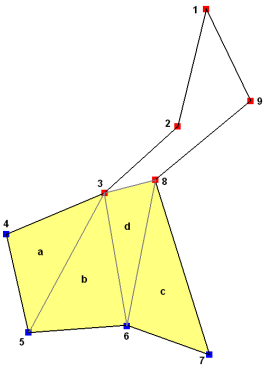
(b) Funn! Hjørne 4 er ferdigbehandlet



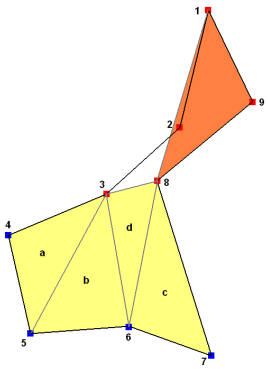
(c) Tilbakesporer hjørnene 6,5,3. Hjørne 5 ferdigbehandlet



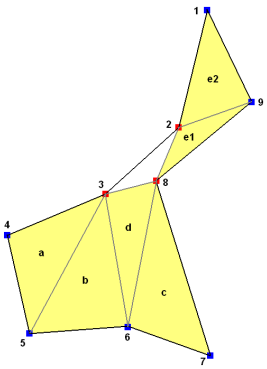
(d) Finner ny trekant. Hjørne 7 er ferdig.



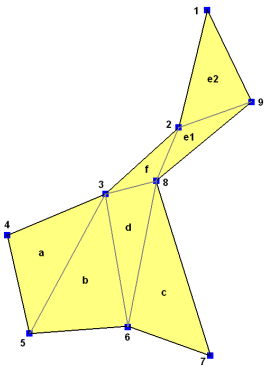
(e) Tilbakesporer hjørnene 8,6,3. Hjørne 6 er ferdig.



(f) Finner ny trekant, men hjørne 2 befinner seg innen-for trekanten. Avbryter..

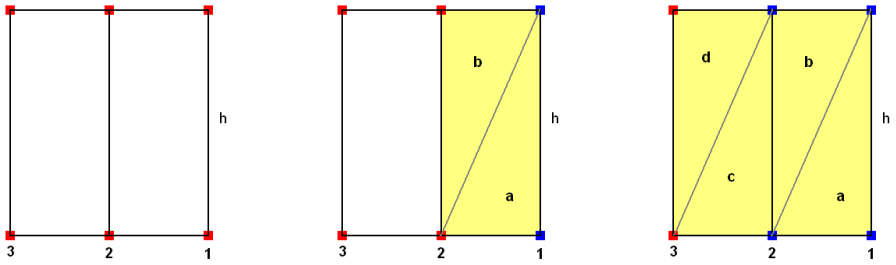


(g) ..tegner heller to mindtre trekanter.



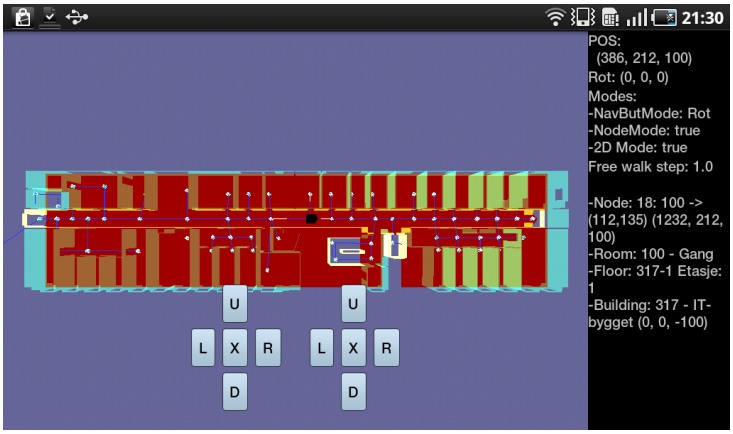
(h) Tilbakesporer hjørnene 8,3,2. Ferdig.

Figur 10.4: Gulvtegningsalgoritme. Legg merke til 10.4f og alternativet 10.4g

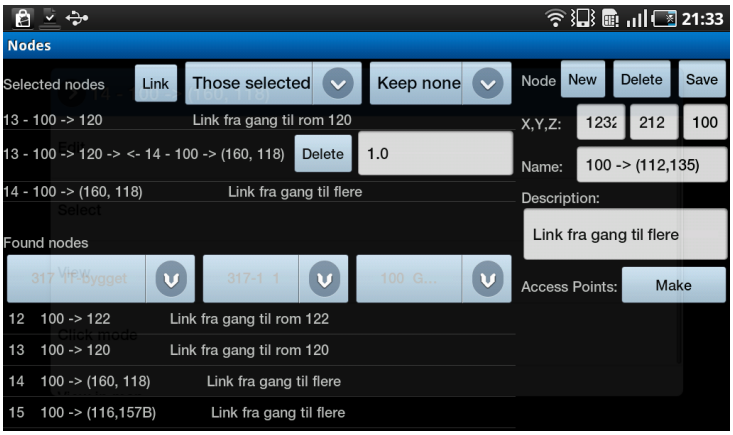


(a) Før start (b) Ferdig med hjørne 1. (c) Ferdig med hjørne 2.

Figur 10.5: Veggtegningsalgoritme. Hjørnene 1, 2 og 3 ligger i gulvplanet, mens h gir høyden langs z-aksen.



Figur 10.6: 2D-modus – Kartvisning ovenfra



Figur 10.7: Noderedigeringsverktøy.

Kapittel 11

Evaluering

For å evaluere sitt eget arbeid, kan man ta utgangspunkt i spørsmålene jeg ville finne svar på, gitt i seksjon 1.4.

Jeg har lykkes i å sette meg inn i androidplattformen, noe mengden produsert kode og grad av fullført implementering understreker.

Applikasjonen har tross alt en ny plattform, kommet mye lengre i å realiseres som en ferdigimplementasjon enn jeg klarte på en kjent platform i fordypningsprosjektet [16].

Jeg klarte ikke å svare på spørsmålet angående eksterne bruk av andre eksterne datakilder, i hovedsak på grunn av at tiden tok slutt og det som hadde blitt implementert også trengte tid til å bli dokumentert.

Kapittel 12

Konklusjon

En av hovedmålene med dette prosjektet var å sette seg inn i android-platformen og med det dra nytte av platformen til å utvikle en noe omfattende applikasjon. Flere kodeeksempler og guider gav utfyllende observasjoner med tanke å identifisere funksjonalitet observert ved utforsking av en android-enhet og knytte dette til hvordan det kan implementeres.

Jeg er svært fornøyd med hva implementasjonen har resultert i, da dette overgår andre utviklingsprosjekter jeg har jobbet med tidligere, i stor grad. Jeg har og fått oppleve å få implementert funksjonalitet og se det fungere i henhold til tankene fra fordypningsprosjektet [16].

Android-platformen i seg selv kunne først virke veldig godt gjennomtenkt og lett å utvikle på, men har jeg har observert tilfeller som krevde stor innsats og kreativ problemløsning for å jobbe rundt problemet, noe som viser at også android-platformen er i en utviklende prosess.

Applikasjonen ble ikke ferdig implementert i henhold til de funksjonelle kravene gitt i seksjon 8.2, men det som til nå gjort virker lovende, og antyder at en slik applikasjon lar seg implementere på en smart-telefon, selv med behov for god grafikkprosessering.

Del IV

Tillegg

Tillegg A

Kildekode

Da jeg har produsert 6300 kodelinjer Java, 550 linjer XML og 22 linjer PHP, er det ikke plass til å legge ved kildekoden. Vil derfor liste alle filer, og gi en beskrivelse av de.

A.1 PHP-fil

For å få importert data fra den eksterne databasen og inn på androidenheten ble det nødvenlig med et php-webbasert grensesnitt som kunne motta spørringer om ønskede data og returnere resultatene i en form for datastruktur. Som en enkel løsning ble JSON valgt (se seksjon 9.2). Dette er et rammeverk som er støttet av de fleste programmering- og skriptspråk, der data aksesseres litt på samme måten som ved en Bundle eller Cursor(se 4.5.2.1, ved å oppgi nøkkel, for så å få en verdi i svar. Hver rad representeres av et JSONObject, lagret i en JSONArray. Jeg laget meg en klasse som tok seg av å oppkobling, motta og avlesing av rad for rad. Serverside PHP-skriptet ser slik ut:

```
<?php
header('Content-Type: text/plain; charset=ISO-8859-1');
$debug=false;
if(isset($_GET["debug"]))$debug=true;
if(isset($_POST['query'])||$debug)
{
        $query=($debug?$_GET['query']:$_POST['query']);
        mysql_connect('localhost','android','android')
          or die('!ERROR|Sorry, the database is down');
```

```
        mysql_select_db('android');
        $rs=mysql_query($query);
        if(mysql_num_rows($rs)<1) echo '[]';
        else
        {
                while($row=mysql_fetch_assoc($rs))
                $output[]=$row;
                echo(json_encode($output));
                mysql_close();
        }
}
else print('[]');

?>
```

A.2 XML-filer

AndroidManifest.xml
Konfigurasjonsfil for applikasjonen

ap_list.xml
Enkel visning med liste for tilgangspunkter, samt en lagre- og avbryt-knapp.

aplist_stt_item.xml
En listeoppføring for tilgangspunkter med avkrysningsboks.

aplist_sttp_item.xml
En listeoppføring for tilgangspunkter med avkrysningsboks, samt bar for signalstyrke.

aplist_ttlp_item.xml
En listeoppføring for tilgangspunkter med lang bar for signalstyrke.

aplist_ttp_item.xml
En listeoppføring for tilgangspunkter med kort bar for signalstyrke.

apnode_list.xml
En visning med en liste for tilgangspunkter, og en liste for noder.

list_3tv_item.xml
En listeoppføring med tre tekstbokser.

list_id_name_item.xml
En listeoppføring med to tekstfelt, id og navn.

list_tv_item.xml
En listeoppføring med en enslig tekstboks.

main_setting.xml
Innstillinger-visningen for hovedaktiviteten.

main.xml
Skjermoppsett for hovedaktiviteten.

nav_but_hor.xml
Navigasjonsknapper, hovedaktivitet.

node_main.xml
Skjermoppsett for noderedigeringsaktiviteten.

node_sel_par_item.xml
En node-listeoppføring for føring av kostnad mot noden som redigeres.

node_sel_rou_et.xml
En node-listeoppføring, ved rute-modus, partallslinje, 1-indeksert. EditText-element.

node_sel_rou_item.xml
En node-listeoppføring, ved rute-modus, oddetallslinje, 1-indeksert. Node-visning.

A.3 Java-filer

AbstractEntety.java
Overordnet klasse for entitetene i datamodellen.

Ap.java
Tilgangspunkt-entitetklassen.

Building.java
Bygnings-entitetenklassen.

DataSource.java
Tiltenkt hånderingsklasse for eksterne datakilder.

Floot.java
Etasje-etitetsklasse.

IndMap.java
Hovedaktivitet. Kartvisning.

IndMapRenderer.java
Renderer for kartvisningen.

Node.java
Node-entitetsklasse

NodeActivity.java
Underaktivitet, og redigeringsprogram for noder.

Point.java
Et punkt i 3D-rommet. Holder også på utvidelsen *RotatePoint* som i tillegg også har rotasjon for hver akse.

Provider.java
En implementasjon av *ContentProvider*. Indernlager.

Room.java
Rom-entitetsklasse.

Route.java
Rute-entitetsklasse.

Storage.java]
Klasse med interface-er for lagring, MysqlViaPHP (tilgang til eksternlager), HTTPtoJSON, og QueryBuilder

World.java
Knytepunkt mellom hovedaktivitet og datamodell, og senter for kontrollfunksjonalitet.

Del V

Kildeliste

Bibliografi

[1] Open Headset Alliance. Faq. "http://www.openhandsetalliance.com/oha_faq.html".

[2] Victor R. Basili. The experomental paradigm in software engineering, 1992.

[3] blackpawn.com. Point in triangle test. "http://www.blackpawn.com/texts/pointinpoly/default.html".

[4] Android Debyg Bridge. Android developers. "http://developer.android.com/guide/developing/tools/adb.html".

[5] code.google.com. Zxing ("zebra crossing"). "http://code.google.com/p/zxing/".

[6] Dalvikvm.com. Dalvik virtual machine. "http://www.dalvikvm.com/".

[7] Android Developers. Activities. "http://developer.android.com/guide/topics/fundamentals/activities.html", .

[8] Android Developers. Application fundamentals. "http://developer.android.com/guide/topics/fundamentals.html", .

[9] Android Developers. The androidmanifest.xml file. "http://developer.android.com/guide/topics/manifest/manifest-intro.html", .

[10] Android Developers. Testing fundamentals. "http://developer.android.com/guide/topics/testing/testing_android.html", .

[11] developler.android.com. What is android? "http://developer.android.com/guide/basics/what-is-android.html".

[12] Ben Elgin. Google buys android fro its mobile arsenal. "http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm", 2005.

[13] Kent German. A brief history of android phones. "http://www.cnet.com/8301-19736_1-20016542-251.html".

[14] json.org. Introducing json. "http://www.json.org/".

[15] Paul Michael Kilgo. Android os: A robust, free, open-source operating system for mobile devices.

[16] Jon Villy Meidell. 3d indoor maps and indoor navigation - specialization project, 2011.

[17] Jeffrey Van Camp Yahoo News. Google activates 500.000 android devices a day, may reach 1 million in october. "http://old.news.yahoo.com/s/digitaltrends/20110628/tc_digitaltrends/googleactivates500000androiddevicesadaymayreach1millinbyoctober", 2011.

[18] Android Open Source Project. Welcome to android. "http://source.android.com".

[19] sobbayi. Sqlite vs mysql: How to decide which to use. "http://blog.sobbayi.com/2010/05/sqlite-vs-mysql-how-to-decide-which-to-use/", 2010.

[20] PlayControl Software. Opengl vertex buffer objects (vbos): A simple tutorial. "http://playcontrol.net/ewing/jibberjabber/opengl_vertex_buffer_object.html", 2008.

[21] Benjamin Speckmann. The android mobile platform - a review paper submitted to the eastern michigan university department of computer science in partial fulfillment of the requirements for the master of science in computer science, 2008.

[22] SQLite.org. About sqlite. "http://www.sqlite.org/about.html", .

[23] SQLite.org. Datatypes in sqlite version 3. "http://www.sqlite.org/datatype3.html", .

[24] U.C. Irvine T. Berners-Lee, R. Fielding and L. Masinter. Rfc2396 - uniform resource identifiers (uri): Generic syntax. "http://www.faqs.org/rfcs/rfc2396.html", 1998.

[25] Wikipedia. Internet media type. "http://en.wikipedia.org/wiki/Internet_media_type", .

[26] Wikipedia. Autocad dxf. "http://en.wikipedia.org/wiki/AutoCAD_DXF", .

[27] Wikipedia. Julian day. "http://en.wikipedia.org/wiki/Julian_Day_Numbers", .

[28] Wikipedia. Mac address. "http://en.wikipedia.org/wiki/MAC_address", .

[29] Wikipedia. Model-view-controller. "http://en.wikipedia.org/wiki/Model_view_controller", .

[30] Wikipedia. Qr code. "http://en.wikipedia.org/wiki/QR_code", .

[31] Wikipedia. Rubik's cube. "http://en.wikipedia.org/wiki/Rubik%27s_Cube", .

[32] Wikipedia. Uniform resource identifier. "http://en.wikipedia.org/wiki/Uniform_Resource_vIdentifier", .

[33] Wikipedia. Vertex buffer object. "http://en.wikipedia.org/wiki/Vertex_Buffer_Object", .