Luleå tekniska
universitet

Projektrapport

Android applikation DicNix

Kurs: D0016E, Digitalt projekt

Deltagare: Jim Gunnarsson, di98jgu@gmail.com

Viktor Stärn, viktor.starn@gmail.com

Källkod: https://github.com/di98jgu/D0016E---Digitalt-projekt.git

Datum: 2013-03-26

Innehåll

Problembeskrivning

Bakgrund

Skellefteå har som mål att vara en "smart-city". Det innebär att man vill förbättra service till medborgarna och utnyttja resurser mer effektivt med hjälp av teknik. Rent praktiskt så innebär det att sensorer används för att samla in och sprida information om energi, vatten, trafik, avfall och väder. Denna satsning går under namnet Sense Smart City[1].

Med ett snödjup upp till 120 mm i snitt så är snö en real fara för byggnader och människor. Flera faktorer inverkar snötryck, luftfuktighet, snödjup och temperatur. Ett etablerat system med sensorer finns för att samla in mätdata i databas. Databasen nås via en RESTful server.

Som en del i ett pilotprojekt vill man nu skapa en Android app för att presentera insamlad data. Avsikten är att användaren, till exempel en fastighetsägare, prenumerera på ett antal mätpunkter. Han kan se insamlad data samt så ska appen larma om vissa riktvärden överskrids.

Krav och målsättning

De punkter som finns från beställaren är:

- Att hämta och presentera data från servern för användaren. Varje användare prenumerera på ett vist antal mätpunkter. Inloggning krävs för att hämta data.
- Att larma om data överskrider vissa värden.

Dessa vaga punkter är vi fria att utveckla. Inga krav ställs på gränssnittet.

Att hämta och presentera data är två skilda delar, databas och gränssnitt. Dessa behöver kommunicera med varandra. För att reducera datatrafiken mot servern så behöver datat sparas lokalt och uppdateras vid behov.

Presentationen kan ske på flera sätt, enklast är en lista. Det finns önskemål om att representera data på en karta, detta är dock mindre viktigt och kan därför ses som extra.

Det är rimligt att användaren vill göra vissa inställningar. Det kan vara att namnge mätpunkterna. Att organisera listan med mätpunkter om de är många. En profil behövs. Att larma är central viktigt, en av målsättningarna med detta projekt. Den bör hamna så högt som möjligt.

Detta ger följande strategi för detta projekt:

1 *Sense Smart City (2013-03-25) http://sensesmartcity.org/index.htm*

- Lista med mätpunkter. Detta är en lista med mätpunkter, en mätpunkt är en specifik sensor och data kopplad till denna. Användaren väljer en mätpunkt ur listan och data för just den mätpunkten visas för användaren.
- Att kontakta servern logga in och hämta data. Här är Yamba och JTwitter en viktig källa till kunskap och inspiration[2] [3]. Twitter använder precis som Sense Smart City en RESTful server.
- Att skapa en lokal databas för datat. En innehållsleverantör, content provider, skulle vara trevligt men är tekniskt mer krävande.
- Ställa samman appen och redovisa faktisk mätdata. Applikationen ska fungera för en *vanlig* användare. Detta är en milstolpe, att se till att allt är gjort och verkligen fungera.
- Larm då data överskrider vissa värden. Vi har nu en färdig produkt med ett minimum av funktionalitet.
- Karta med mätpunkter. Användarens mätpunkter redovisas på en karta. Väljer användaren en viss mätpunkt visas all data kopplad till den.
- Att hantera användarens inställningar.

Plattform

Användaren förutsätts veta hur man använder en Android enhet. Typ av enhet vet vi inget om men då appen ska larma kan vi anta att primär enhet är mobiltelefon. Den har användaren alltid med sig.

Andelen användare som har Gingerbread eller högre är närmare 90% där Gingerbread svara för 45% av samtliga användare[4]. Varje version av Android är namngiven, Gingerbread är API version 9 och 10.

Skärmen är typiskt av normal storlek[5], normal. Det finns tre andra men normal täcker 90% av alla användare. Det finns fyra olika upplösningar dpi, mdpi, hdpi och xhdpi. Den minsta av dessa dpi används mycket lite, mindre än 1% och är därför av mindre vikt.

Appen bör därför riktas mot Gingerbread med normal skärm och de tre upplösningarna mdpi, hdpi och xhdpi. Detta når flest användare.

2 *JTwitter - the Java library for the Twitter API (2013-03-19) http://www.winterwell.com/software/jtwitter.php*
3 *LearningAndroidYamba, marakana (2013-03-19) https://github.com/marakana/LearningAndroidYamba.git*
4 *Dashboards, Android Developers (2013-02-06) developer.android.com/about/dashboards/index.html*
5 *Supporting Multiple Screens, Android Developers (2013-02-06) developer.android.com/guide/practices/screens_support.html*

Systembeskrivning

Gränssnitt

Gränssnittet för applikationen är framtaget med tydlighet, enkelhet och användarvänlighet i fokus. Utgångspunkten är Google I/O 2010 Android UI design patterns[6]. Förhoppningen är att central funktionalitet skall nås på ett smidigt och intuitivt sätt. Bilder och ikoner som använts i applikationen har antingen skapats inom projektet eller är fria att använda.

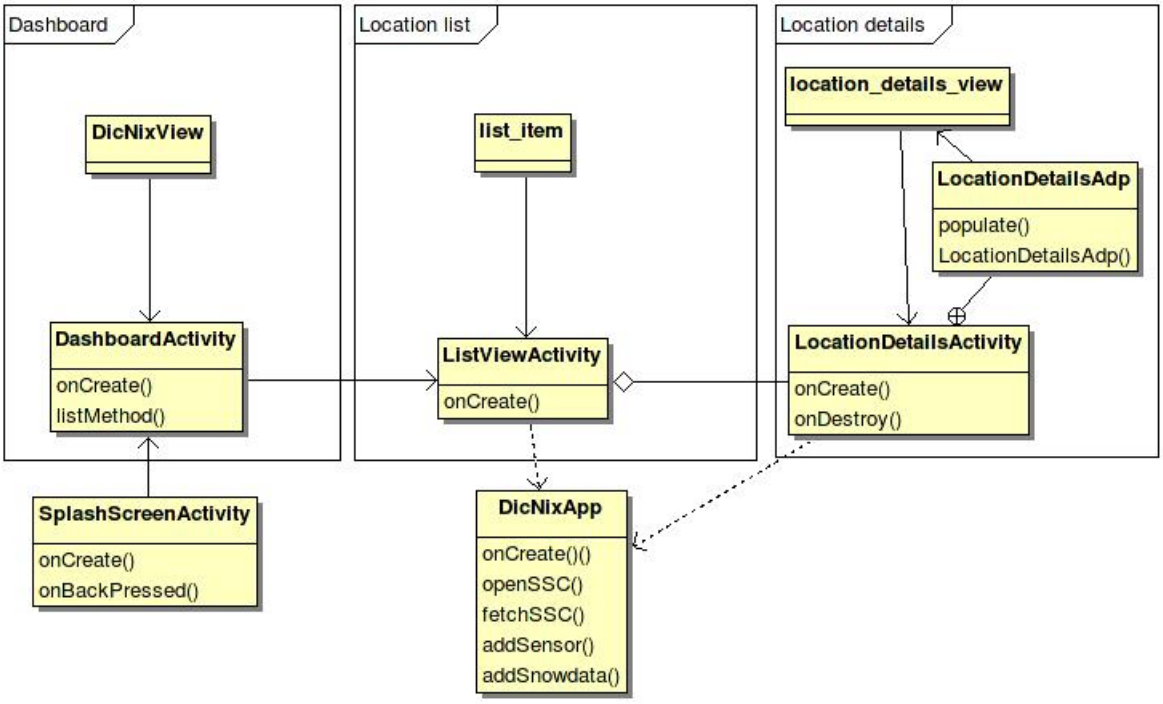

Bild 1: Aktiviteter och vyer.

Gränssnittet är skalbart, bibehållet utseende oberoende av skärmstorlek. Men eftersom välkomstskärm och övriga bakgrundsbilder är skapade för en upplösning i storleksordningen 480 x 800 pixlar kan estetiken bli lidande när applikationen körs på enheter vars skärmupplösning skiljer sig mycket från ovan nämnda.

6 *Google I/O 2010 - Android UI design patterns (2013-03-25)*
http://www.google.com/events/io/2010/sessions/android-ui-design-patterns.html

Aktiviteter

Varje skärm, välkomstskärm, inloggning, instrumentpanel, sensorlista och sensordata representeras i applikationens källkod som en subklass av `Activity`. Detta är ett Android-specifikt begrepp som kan sägas beskriva en klass som hanterar en skärm i applikationens gränssnitt.

Dessa aktiviteter länkas sedan samman via något som kallas `Intent`, en avsikt. den aktuella aktiviteten startar den önskade aktiviteten via `Intent`. Uttryckt med andra ord så är avsikter limmet som håller samman aktiviteterna.

Varje aktivitet har i regel en tillhörande XML-fil exempelvis `splash_screen.xml` som ansvarar för aktivitetens layout.

Lokal databas

Databasen SQLite är en inbyggd del av Android och är ett lättviktsverktyg för hantering av relationsdatabaser[7]. Den bäddas in i applikationen kör på samma tråd som denna. Databaser kan delas mellan applikationer med `Content providers`, innehållsleverantör, Att skapa en innehållsleverantör för snödatat var ett önskemål men så långt kom inte projektet.

Vår databas heter Snowflake, `snowflake.db` och skapas vid första inloggningen till Sense Smart City. SSC beskrivs mer detalj senare. Snowflake består i nuläget av två tabeller, `Sensordata` och `Snowdata`. Kolumnerna följer så nära som möjligt de i SSC. Aktiviteterna arbetar alltid mot den lokala databasen och känner inte alls till SSC. Detta underlättar implementation och gör koden lättare att underhålla.

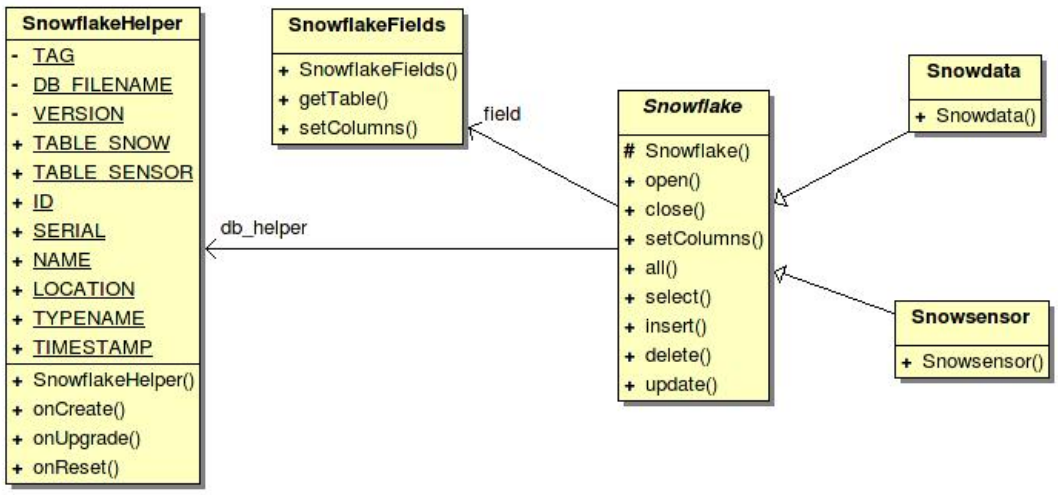

Bild 2: Databasen Snowflake, de flesta attributen har utlämnats. Klassen Snowflake är abstrakt.

7 *SQLite (2013-03-25) http://www.sqlite.org*

Tabellklasserna `database.Snowsensor` och `database.Snowdata` ärver av `database.Snowflake` och håller reda på kolumnerna i respektive tabell. Klassen `Snowflake` i sin tur ansvara för hanterandet av databasen med funktioner som `select`, `insert`, `update` och `delete`, alltså CRUD. För att skapa själva databasen används hjälpklassen `database.SnowflakeHelper` som också listar samtliga tabeller och kolumner.

Den lokala databasen länkas samman med fjärrdatabasen Sense Smart City i applikationsobjektet `DicNixApp`. Ett applikationsobjekt, `Application object`, lever under applikationens hela livsspann och är oberoende av aktiviteterna. Här finns referenser till den lokala databasens och fjärrdatabasen. En aktivitet behöver därför aldrig skapa en ny referens och det ger ökad säkerhet vid trådning.

Det kan tilläggas att det inte finns något väletablerat sätt att hantera SQLite under Android. Något som gett upphov till mycket tyckande och tänkande Några exempel är Handling some SQLite issues, Correctly Managing Your SQLite Database och Database pitfalls[8] [9] [10].

Sense Smart City

Kommunikation med fjärrdatabasen Sense Smart City sker med ett fristående bibliotek SSC. Biblioteket är fristående och utvecklat utanför Android. Det gör det enkelt att testa och utveckla. Gränssnittet mot servern finns beskrivet i bilagan API Sense Smart City.

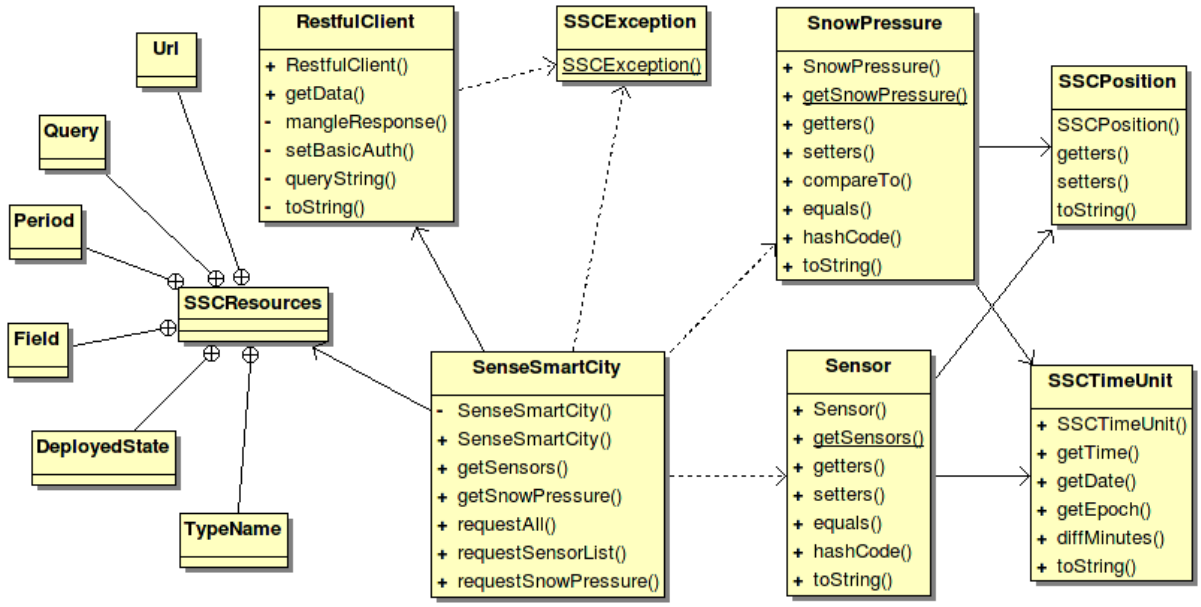

Bild 3: Biblioteket SSC, attribut har utelämnats och vissa privata metoder.

8 *Android. Handling some SQLite issues (2013-03-25) http://www.enterra-inc.com/techzone/handling_sql_issues*

9 *Correctly Managing Your SQLite Database (2013-03-25) http://www.androiddesignpatterns.com/2012/05/correctly-managing-your-sqlite-database.html*

10 *Database pitfalls (2013-03-25) http://www.ragtag.info/2011/feb/1/database-pitfalls/*

Relationen mellan sensor och sensor data är en till många, en sensor flera avläsningar. Sensor och avläsning kopplas samman med ett serienummer som är unikt för varje sensor. Endast en typ av sensor är implementerad, SnowPressure. Biblioteket är så strukturerat för att möjligöra en fortsatt utveckling.

Huvudklassen är `SenseSmartCity`. Här finns metoder för att hämta data från fjärrdatabasen. För att hämta data krävs användarnamn och lösenord. Det finns två typer av anrop, ett för data om sensorerna och ett för snödata. Data från fjärrdatabasen kapslas in i klasserna `Sensor` och `SnowPressure`.

Klassen `SSCResources` håller en lista över all statisk data som adresser nycklar och kolumner. Den fungera också som dokumentation över alla giltiga fält. Tanken är att den ska användas tillsammans med `SenseSmartCity` vid anrop till fjärrdatabas.

Som sagts tidigare så länkas biblioteket SSC samman med den lokala databasen i `DicNixApp`. Databasen SQLite i Android använder sig av en skräddarsydd hashtabell `ContentValues`. Datat från SSC måste flyttas över värde för värde. Poängen med klasserna `Sensor` och `SnowPressure` är att `DicNixApp` varken vet eller behöver veta något om fjärrdatabasen. Aktiviteterna i sin tur vet inget om biblioteket SSC.

Applikation DicNix

DicNix

DicNix är hämtat från latinet och kan grovt översättas med berätta snö. Ett ord som är unikt, enkelt och lätt att uttala. Detta blev namnet på applikationen.

Vid uppstart möts användaren av en välkomstskärm som förmedlar applikationens namn och logotyp. Välkomstskärmen försvinner strax för att ersättas av en instrumentpanel som erbjuder användaren möjlighet att lista sina registrerade sensorer, synkronisera informationen som finns lagrad i fjärrdatabasen med den som finns sparad i den lokala databasen samt redigera användarinställningar. Själva sensordatan nås via sensorlistan på instrumentpanelen.

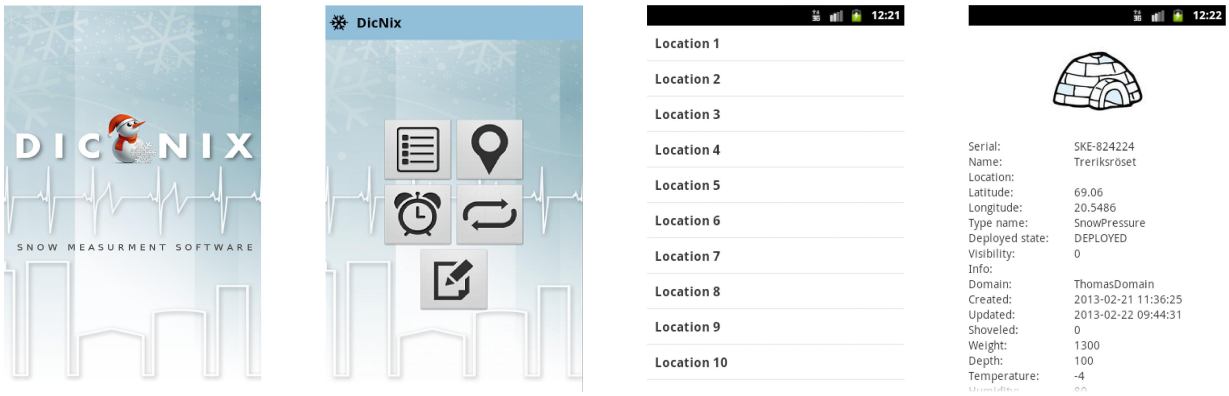

Bild 4: Välkomstskärm, instrumentpanel, sensorlista och sensordata, notera att appen är under utveckling.

Källkod

Projektet finns i sin helhet tillgängligt på GitHub, D0016E---Digitalt-projekt[11]. GitHub är utvecklat för samarbete projektledning och utveckling av mjukvara. Projektet i sin helhet med all kod kan hämtas med kommandot:

```
git clone https://github.com/di98jgu/D0016E---Digitalt-projekt.git
```

Detta under förutsättning att git finns installerat[12].

11 *D0016E---Digitalt-projekt (2013-03-19) https://github.com/di98jgu/D0016E---Digitalt-projekt.git*
12 *git (2013-03-19) http://git-scm.com*

Installation

Det förutsätts här att en utvecklingsmiljö med Eclipse och SDK finns tillgänglig[13]. Det finns inget färdigt packet för installation utan projektet måste importeras som existerande Android kod. Applikationen finns i katalogen DicNix.

Det är lite omständligt att importera ett projekt då en bug i Eclipse gör att det inte går att importera en projektkatalog utan att flytta den. Det går alltså inte att öppna ett projekt på plats. Så DicNix måste flyttas till annan plats innan den kan importeras till Eclipse.

För att köra koden behöves biblioteket för Sense Smart City. Det är en fristående klient för fjärrdatabasen. Biblioteket finns i katalogen lib/ssc och länkas in som extern källa.

Nu ska projektet vara körklart. Lägsta API nivå är 9 och målnivå är 10, alltså Gingerbread.

13 *Get the Android SDK (2013-03-19) http://developer.android.com/sdk/index.html*

Diskussion

Målsättning

Vi hade en ambitiös lista med punkter som vi önskade uppnå. Det ena av dessa löd:

> *”Ställa samman appen och redovisa faktisk mätdata. Applikationen ska fungera för en vanlig användare. Detta är en milstolpe, att se till att allt är gjort och verkligen fungera.”*

Vi kom fram till första meningen. Appen DicNix fungera och visar att all huvuddelar fungera som de ska. Det innebär att vi har ett bibliotek Sense Smart City för att hämta data. En lokal databas för spara datat och gränssnitt för att redovisa det.

Appen fungera inte för en vanlig användare. Den saknar många kritiska komponenter som hantering av synkronisering mot fjärrdatabas. Det saknas möjlighet att logga in. Vi har för varje senor en lista med avläsningar och inte bara en enda avläsning som gränssnittet ger ett intryck av.

Men det ska påpekas att när listan sattes upp så var vi tre i det här projektet. Den var ambitiös målsättning för tre personer. För två så var den inte möjlig att genomföra.

GitHub

För att dela och hantera kod och övrigt material för det här projektet så används GitHub. Det fungerande utmärkt under Linux men mindre bra under Windows. Eclipse har stöd för GitHub vilket testades under Windows. Det fungerade mindre bra. Vid någon tidpunkt så upphörde systemet att fungera alls med det kryptiska meddelandet ”git-upload-pack not found...”. Att använda git direkt utan Eclipse som mellanhand fungerade bättre[14]. Det körs i terminal.

När git fungera som det är tänkt så är det verkligen ett trevligt och effektivt verktyg. Det gör det mycket lättare att hantera kod och annat material. Trots besväret med att få git att fungera under Windows så väl värt mödan. Det tar cirka en vecka att lära sig det som behövs för ett projekt.

14 *Welcome to the home page of Git for Windows (2013-03-19) http://msysgit.github.com*

Digitalt projekt

Det verkligt intressanta och drivande med detta projekt är att det är hämtat från verkligheten. De flesta projekt brukar antingen vara en laboration eller ett tänkt problem. Omfattningen var rimlig, projektet var egentligen utformat för 4 –5 personer. Då hade vi haft en komplett och fungerande app. Vi var från början tre personer men blev snabbt två. Det var oförutsett och lite kunde göras åt saken.

Det hade varit till stor hjälp om det hade funnits en stödperson som verkligen kunde Android. Android har vad man brukar kalla brant inlärningskurva. Det är omfattande ramverk och tar mycket tid att lära sig på egen hand.

Framtida utveckling

Vi har en fungerande app där data kan hämtas från fjärrdatabasen. En fortsatt utveckling av DicNix med den kunskap vi har idag ser ut som följande:

- Gränssnittet skapades innan vi hade någon kunskap om Sense Smart City. En ny och mer utarbetad plan behövs för gränssnittet
- Inloggning
- System för hantering av synkronisering. Detta ska ske automatiskt.
- Larm då data överskrider vissa värden. Vi har nu en färdig produkt med ett minimum av funktionalitet. Applikationen ska nu fungera för en vanlig användare.
- Utveckling av Sense Smart City samt att skapa en innehållsleverantör.
- Karta med mätpunkter. Väljer användaren en viss mätpunkt visas all data kopplad till den.

Detta skiljer en del från den tidigare målsättningen. Främst då vi nu kan precisera vad som behövs på helt annat sätt än tidigare.

Referenser

Sense Smart City (2013-03-25) http://sensesmartcity.org/index.htm

JTwitter - the Java library for the Twitter API (2013-03-19)
http://www.winterwell.com/software/jtwitter.php

LearningAndroidYamba, marakana (2013-03-19)
https://github.com/marakana/LearningAndroidYamba.git

Dashboards, Android Developers (2013-02-06)
developer.android.com/about/dashboards/index.html

Supporting Multiple Screens, Android Developers (2013-02-06)
developer.android.com/guide/practices/screens_support.html

Google I/O 2010 - Android UI design patterns (2013-03-25)
http://www.google.com/events/io/2010/sessions/android-ui-design-patterns.html

SQLite (2013-03-25) http://www.sqlite.org

Android. Handling some SQLite issues (2013-03-25) http://www.enterra-inc.com/techzone/handling_sql_issues

Correctly Managing Your SQLite Database (2013-03-25)
http://www.androiddesignpatterns.com/2012/05/correctly-managing-your-sqlite-database.html

Database pitfalls (2013-03-25) http://www.ragtag.info/2011/feb/1/database-pitfalls/

D0016E---Digitalt-projekt (2013-03-19) https://github.com/di98jgu/D0016E---Digitalt-projekt.git

git (2013-03-19) http://git-scm.com

Get the Android SDK (2013-03-19) http://developer.android.com/sdk/index.html

Welcome to the home page of Git for Windows (2013-03-19) http://msysgit.github.com

Malm Thomas (feburari 2013) Brevkorrespondens

Bilaga
API Sense Smart City

Dokumentation

Fjärrdatabasens API är endast beskrivet med ett antal exempel i Python. Strukturen för anrop till fjärrdatabasen är därför känt men inte respons. För bättre förståelse av fjärrdatabasen gjordes ett antal anrop från Python och dess respons analyserades. Detta är grunden för biblioteket SSC och återges här som bilaga.

Kod som ligger till grund för analys av fjärrdatabasen:

- ListSensor.py
- RequestSensor.py
- AddSensor.py
- SubmitSnowPressureData.py
- ListSensor.java

Kod exemplen kompletterades med brevkorrespondens med Thomas Malm[15], ej återgiven.

Anrop till fjärrdatabas

Adress

Basadress till fjärrdatabasen:

https://ip30.csse.tt.ltu.se/ssc/api/basic/

Relativa adresser, som alltså följer efter basadressen:

- **sensor/list:** Lista över alla sensorer, mätpunkter.
- **snow_pressure/request:** All snödata

15 *Malm Thomas (feburari 2013) Brevkorrespondens*

- **geo_location/request:** Position
- **free_text/request:** Allmän information, om sådan finns
- **temperature/request:** Temperatur

Av dessa är **sensor/list** samt **snow_pressure/request** de som är av intresse. De är också väldokumenterade vad gäller giltiga inparametrar. För att hämta information används SSL tillsammans med autentisering.
Data hämtas med GET och lämnas med POST, den relativa adressen skiljer sig åt. Här redovisas endast GET ty vi är endast intresserade av att hämta data.

Parameter sensor/list

Parametrar för **sensor/list**:

- **sensors:** JSON formaterad lista med sensor id.
- **public:** Visar sensorer från alla domäner ”**yes**” eller endast vår ”**no**”.
- **format:** Returnera data i form av ”**json**” eller ”**xml**”.

Samtliga parametrar är valfria. Utelämnas **sensors** så fås en lista över samtliga sensorer.

Parameter snow_pressure/request

Parametrar för **snow_pressure/request**:

- **sensor:** JSON formaterad lista med sensor id, ”**serial**”.
- **fields:** JSON formaterad lista med data fält, kolumner, som önskas.
- **period:** Tidsperiod, mätvärden för ”**last-hour**”, ”**last-day**”, ”**last-week**”, ”**last-month**” eller ”**last-year**”.
- **start:** Startpunkt för tidsperiod, (YYYY-MM-DD HH:MM:SS)
- **end:** Slutpunkt för tidsperiod, (YYYY-MM-DD HH:MM:SS)
- **limit:** Begränsa resultat till så här många poster.
- **offset:** Returnera poster från och med den här positionen.
- **sort:** Sortering av rader ”**desc**” eller ”**asc**”.

Med undantag av sensor så är samtliga parametrar valfria. En tidsperiod kan anges på två sätt under vilken vi har ett okänt antal utförda mätningar. Vad som händer om vi blandar **period**, **start** och **end** är ej dokumenterat.

Svar från fjärrdatabas

Svar från databasen är mindre väl dokumenterat.

Resultat sensor/list

För **sensor/list** kan vi får resultat i form av XML eller JSON.
I JSON så fås följande resultat:


```
{"response":[{...},{...},{"serial":"SKE-472478","name":"LTU
kontor","location":"","latitude":"64.7443","longitude":"20.9558","type_name":"SnowPress
ure","deployed_state":"DEPLOYED","visibility":"0","info":"","domain":"ThomasDomain","cr
eated":"2013-02-20 15:06:54","updated":"2013-02-21 09:01:40"},{...}]}
```


Vi har alltså för varje sensor följande information:

- **serial:** Detta är unikt id för sensorn, motsvars av ”sensor” ovan.
- **name:** Ett mer läsbart namn för oss människor, max 50 tecken.
- **location:** Kort platsbeskrivning, max 128 tecken.
- **latitude:** I grader, decimalt.
- **longitude:** I grader, decimalt.
- **type_name:** Typ av sensor, ”**FreeText**”, ”**Temperature**”, ”**Motion**”, ”**GeoLocation**”, ”**SnowPressure**”. Bör i vårt fall endast vara ”**SnowPressure**”.
- **deployed_state:** Status kan vara ”**DEPLOYED**” eller ”**NOT_DEPLOYED**”.
- **visibility:** Synlig för andra domänanvändare ”**1**” eller inte ”**0**”.
- **info:** Beskrivning av sensorn, max 255 tecken.
- **domain:** Domän som sensorn tillhör.
- **created:** Registreringsdatum (YYYY-MM-DD HH:MM:SS)
- **updated:** Senast uppdaterad (YYYY-MM-DD HH:MM:SS)

Alla värden oavsett typ anges som strängar, saknas värde är strängen tom. Nycklar anges alltid med germaner.

Resultat snow_pressure/request

För **snow_pressure/request** så är resultatet alltid i JSON. Detta är sant för Python men inte för Java. Vad som skiljer i sättet på vilket fjärrdatabasen anropas är inte känt. På försök angavs precis som för **sensor/list** parametern **format** satt till JSON, vilket löste problemet.

Anges endast en lista med sensorer så fås fälten **data_time**, **weight**, **depth**. Det framgår inte hur många mättillfällen som fås tillbaka, mest troligt är samtliga. En lämplig period kan därför vara lämplig om mätning har pågått under lång tid.

Med fälten, **field,** satta till **shoveld**, **weight**, **depth**, **temperature**, **humidity**, **data_time** och **info** fås följande:


```
{"response":{"12345":
[{"info":null,"shoveld":"1","weight":"1100","depth":"488","temperature":"-
1","humidity":"83","data_time":"2013-02-06 15:21:38"},{"info":"Ett
test","shoveld":"1","weight":"12","depth":"23","temperature":"-
7","humidity":"80","data_time":"2013-02-19 17:36:53"}],"SKE-472478":[...]}}
```


Vi har alltså:

- **12 345:** Detta är unikt id för sensorn.
- **info:** Beskrivning av aktuell registrering, max 100 tecken.
- **shoveld:** Ange om platsen gjorts ren på snö '1' eller inte '0'.
- **weight:** Snövikt i gram.
- **depth:** Snödjup i centimeter.
- **temperature:** Temperatur i grader Celsius.
- **humidity:** Relativ fuktighet RH i procent.
- **data_time:** Registrering av mätdata (YYYY-MM-DD HH:MM:SS)

Varje mätning märks med ett id unikt för varje sensor. Detta id benämns omväxlade **serial** eller **sensor**. Fältet **Shoveld** är korrekt återgivet vad gäller stavning, stavas normalt annars ”shoveled”. Notera att vi här har null för info fältet. Som tidigare så är samtliga nycklar germaner.

Dataformat

Nycklar ska alltid anges i germaner. Värden är undantagslöst strängar. Saknas värden så anges detta som en tom sträng. Från databasen kan null värden förekomma.

Tid är undantagslöst angivet i formatet 'YYYY-MM-DD HH:MM:SS'. Kan därför lätt läsa av med reguljära uttryck.

Med undantag för position så är tal angivarna som heltal. Position är angiven i grader med decimal precision, minst 4 siffror.

Temperatur anges med tecken i hela grader, '-7'. Boolean är '0' för falskt och '1' för sant.

Till några av nycklarna finns en bestämd uppsättning med konstanter, **type_name** till exempel, dessa är rena strängvärden.