Utveckling av en GIS-bildbank

Filip Holmberg

Examensarbete för Tradenom-examen

Utbildningsprogrammet i Informationsbehandling

Raseborg 2012

EXAMENSARBETE

Författare: Filip Holmberg
Utbildningsprogram och ort: Tradenom, Raseborg
Inriktningsalternativ/Fördjupning: Informationsbehandling
Handledare: Tomas Hammar

Titel: Utveckling av en GIS-bildbank

Datum 1.5.2012 Sidantal 35 Bilagor 3

Sammanfattning

Mitt examensarbete är ett utvecklingsprojekt där jag utvecklade en bildtjänst för hantering av bilder med GIS (Geographical Information System) -information, dvs. dess anknytning till en viss position. Tjänstens viktigaste funktionella krav var att kunna visa bilder och exakta positioner på en karta. Tjänsten utvecklades som en separat del i en större helhet i företaget Lounaispaikkas karttjänst.

Arbetet går igenom hela processen från planeringen till överlämnandet till kunden. Fokus ligger på de moderna, så kallade web 2.0 -teknikerna med Javascript i en central roll.

Projektet resulterade i en tjänst som båda parter var nöjda med. Tjänsten är klar att tas i bruk inom en snar framtid.

Språk: Svenska Nyckelord: bildbank, gis, php, javascript

OPINNÄYTETYÖ

Tekijä: Filip Holmberg
Koulutusohjelma ja paikkakunta: Tradenomi, Raasepori
Suuntautumisvaihtoehto/Syventävät opinnot: Tietojenkäsittely
Ohjaajat: Tomas Hammar

Nimike: GIS-kuvapankin kehittäminen

Päivämäärä 1.5.2012 Sivumäärä 35 Liitteet 3

Tiivistelmä

Opinnäytetyöni on kehitysprojekti missä kehitin kuvapalvelun GIS(Geographical Information System) kuvien hallintaan. Palvelun tärkein toiminnallinen vaatimus oli kuvien ja niitten sijainnin näyttäminen kartalla. Palvelu on osa Lounaispaikka yrityksen karttapalvelua ja keskittyy kuvien GIS tietoihin, kuten tarkka sijainti ja geograafiset tiedot.

Työ käsittelee kehitysprosessia suunnitteluvaiheesta asiakkaalle luovuttamiseen asti. Painopiste on moderneissa, niin sanotuissa web 2.0, tekniikoissa missä Javascript on keskeisessä roolissa.

Projektin päättyessä molemmat osapuolet olivat tyytyväisiä ja vakuuttuneita siitä, että palvelu on julkaisukelpoinen lähitulevaisuudessa.

Kieli: Ruotsi Avainsanat: kuvapankki, gis, php, javascript

BACHELOR'S THESIS

Author: Filip Holmberg
Degree Programme: Bachelor of Business, Raseborg
Specialization: Information processing
Supervisors: Tomas Hammar

Title: Development of a GIS-image service

Date 1.5.2012 Number of pages 35 Appendices 3

Summary

My thesis is a development project where I developed an online image service for GIS(Geographical Information System) related images. The most important funtionality requirement was the ability to show images and their exact location on a map. The image service was developed for Lounaispaikka as a future part of their existing image service.

The whole process from planning to customer delivery is described in the thesis. Most focus is put on modern web techniques with Javascript in a major role.

My work resulted in a service which is ready to be integrated and released as a part of Lounaispaikka's map service.

Language: Swedish Key words: photo service, gis, php, javascript

---

Innehållsförteckning

1 Inledning

Detta examensarbete baserar sig på ett projekt utfört åt Sydvästra Finlands Förbund och dess GIS-organisation Lounaispaikka. Arbetet är utfört som ett tidsbundet projekt och utfördes under perioden december 2009 till augusti 2010. Projektet utfördes som ett normalt avlönat arbete och beslutet om att använda projektet för mitt lärdomsprov kom först i ett senare skede. Yrkeshögskolan Novia är en av aktörerna i Lounaispaikkas verksamhet, vilket ledde till att projektet har utförst i Novias utrymmen i Raseborg.

Projektet har gått under namnet "Kuvapankki" och jag använder mig av översättningen bildbank i detta arbete. Bildbanken är en del av Lounaispaikkas förnyade webbtjänst, som jag går närmare in på i nästa kapitel. Trots att bildbanken integreras i Lounaispaikkas tjänst har den utvecklats helt fristående på en egen server under hela projektet.

Examensarbetet beskriver bildbankens utvecklingsprocess. Arbetet är uppbyggt så att jag först går igenom teorin och tekniken för den redan existerande tjänsten, som min bildbank baserar sig på, och sedan går jag mera detaljerat in på hur och varför jag utvecklat bildbanken som jag gjort.

Trots att arbetet inte utfördes med ett examensarbete i åtanke, anser jag att det lämpade sig bra som examensarbete. Jag samlade på mig en mängd nya kunskaper under projektets gång, inte minst om hur man genomför ett projekt från kravspecifikation till slutlig produkt.

1.1 Krav

Lounaispaikka hade för projektet producerat en detaljerad kravspecifikation(se bilaga 1.), som avsevärt underlättade arbetet med att komma igång. Eftersom tjänstens viktigaste funktioner var detaljerat specificerade var det lätt att fokusera på rätt saker redan i planeringsfasen. Kravspecifikationen delar in projektet i tre huvudfunktioner: uppladdning av bilder, modifiering av bilddata samt visning av bilder.

Den kanske mest elementära funktionen för en bildtjänst rent generellt sett är möjligheten till uppladdning av bildmaterial. Användaren skall från sin egen hårdskiva enkelt och effektivt kunna ladda upp bilder till tjänstens server. Lounaispaikkas krav kring denna funktion var, att uppladdningen skall lyckas med samma komponent i samtliga webbläsare, som tjänsten stöder. Möjligheten att ladda upp flera bilder på en gång var även ett tydligt krav. I samband med uppladdningen skall bilden tilldelas koordinater, antingen genom manuell placering på kartan eller direkt från bildens EXIF-koordinater i det fallet att bilden är tagen med en GPS-kamera. För att strukturera bilderna bör varje bild ingå i ett album. Album bör även gå att skapa i samband med uppladdningen.

Bildernas metadata såsom namn, datum, fotograf samt beskrivning bör gå att modifiera via ett enkelt gränssnitt såväl vid uppladdningen som vid ett senare tillfälle, då bilden redan ingår i bildtjänsten. Editering av många bilder på en gång framgår också som ett krav. Ett viktigt krav är även att kunna kommentera bilderna.

Som sista stora funktionellt krav har vi visning av bilder. Bilderna bör kunna visas såväl enskilt som såsom ett galleri, där man ser bilderna samt deras placering på kartan.

1.2 Mål

Tjänstens mål är att erbjuda en användarvänlig men innehållsrik bildtjänst för såväl professionella som amatörer inom GIS. Lounaispaikkas nuvarande användare skall få möjligheten att utvidga sina material med bilder, men man hoppas även kunna locka till sig nya användare. Bildbankens funktionella mål är att möjliggöra uppladdning av bilder, som sedan kan kopplas samman via koordinater med de övriga material, som Lounaispaikkas tjänst erbjuder.

Bildbanken är en viktig del av Lounaispaikkas satsning då det gäller att nå ut till en bredare publik. Public Participatory GIS (PPGIS) är en starkt kommande trend inom GIS och går ut på att involvera användaren i skapandet av GIS-materialet. Bildbanken består enbart av användardata så det är i högsta grad fråga om PPGIS. Bildtjänster med till koordinater kopplade bilder blir allt vanligare och många stora aktörer har redan publicerat dylika tjänster. Som exempel kan Google Maps, Yahoo Maps och Picasa nämnas.

Lounaispaikka vill således också visa med sin tjänst, att man ligger långt fram när det gäller utvecklingen av GIS.

Jag hade ingen erfarenhet av vare sig GIS eller Web 2.0 vilket ledde till att mitt mål med arbetet, förutom att leverera en välfungerande produkt, även blev att lära mig någonting nytt och utvecklas som programutvecklare.

1.3 Problemprecisering

För att uppnå kundens mål fanns det ett antal centrala frågor som krävde svar. Med vilka tekniker och med vilken typ av gränssnitt skapar man en optimal lösning för uppladdning av bilder till en server? Hur visualiserar man bilders geografiska position på bästa sätt? Hur kan man utnyttja Web 2.0-tekniker för att öka användarvänligheten? Hur bör man disponera arbetet? Vilka funktionaliteter bör skapas först?

För att kunna besvara dessa frågor på ett så bra sätt som möjligt, valde jag att som första steg omsorgsfullt studera teorin, dvs. de olika teknikerna som kunde användas för respektive lösning. Eftersom jag inte hade någon erfarenhet av Web 2.0, satte jag fokus på att fördjupa mig i dess grunder, som en inledning på projektet. När jag breddat mina kunskaper inom området fortsatte jag med att studera mera specifika delar av projektet, såsom t.ex. filuppladdningen. Efter studiefasen övergick jag till planeringen av mina egna lösningar. Jag bestämde mig redan i ett tidigt skede att jag skulle tackla problemen som små helheter, dvs. studera, planera och utveckla helheterna enskillt, för att seda upprepa processen för alla delproblem.

2 Lounaispaikka

2.1 Verksamhet

Lounaispaikka är en icke vinstdrivande organisation, som samordnar sydvästra Finlands aktörer inom geografisk information(GIS). Organisationen strävar efter, att förbättra utbudet av GIS-tjänster samt samarbetet mellan aktörerna. Lounaispaikkas tjänster är under ständig utveckling och man försöker följa med den internationella utvecklingen så bra som möjligt. Lounaispaikkas huvudtjänst går under namnet LPII och beskrivs närmare i nästa kapitel. Förutom LPII erbjuder Lounaispaikka ett flertal tillämpade tjänster såsom Paikkaoppi (www.paikkaoppi.fi) som riktar sig till undervisning inom GIS. Alla tjänster är gratis för användaren och de flesta erbjuder även en helt öppen källkod.

Lounaispaikkas projekt finansieras av områdets aktörer inom GIS. I Lounaispaikka nätverket ingår följande aktörer: Egentliga Finlands Förbund, Sydvästra Finlands Miljöcentral, Åbo Akademi, Yrkeshögskolan Novia, Åbo stad, Satakuntaliitto och Turun Yliopisto.

2.2 Tjänster

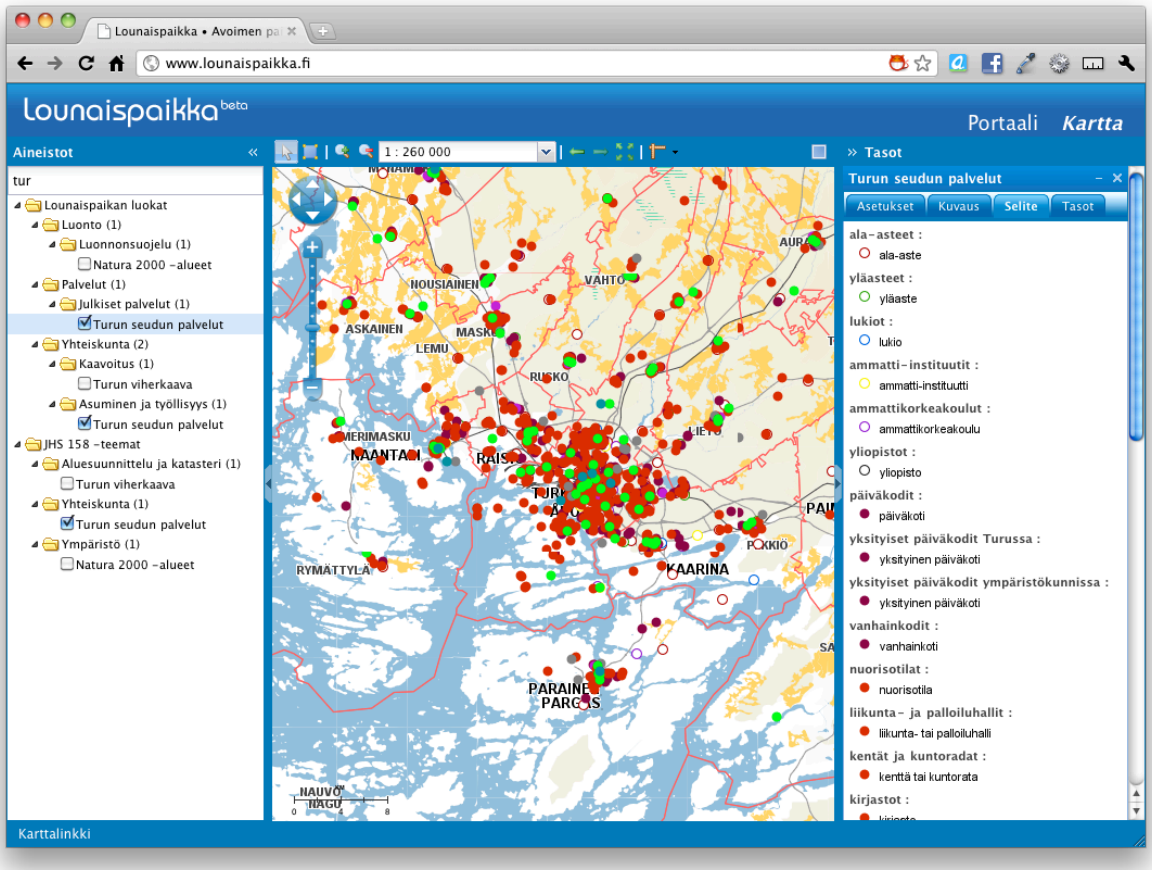
Lounaispaikka erbjuder en rad olika tjänster men jag skall fokusera på den nya kartportalen, LP II, och dess delprojekt i detta arbete. LP II står för Lounaispaikka II och hänvisar till att det är den andra versionen Lounaispaikka har utvecklat av sin karttjänst.

LP II är den tjänst som ligger som grund för hela mitt projekt. LP II är en förnyad version av Lounaispaikkas ursprungliga karttjänst, som byggde på gammal teknik och byttes därför ut under slutet av år 2009. LP II är i grund och botten en karttjänst, där regionens GIS-material finns samlat. Totalt har tjänsten idag över 200 olika GIS-material och antalet växer kontinuerligt. Materialet är av blandad natur med allt från olika fiskarters fortplantningsområden till regionens bredbandsutbredning. Till skillnad från liknande karttjänster erbjuder LP II sina användare möjligheten att redigera och skapa material direkt från webbläsaren. I skrivande stund erbjuder LP II följande grundläggande funktioner:

- Visualisering av obegränsat antal GIS-material samtidigt
- Visualisering av utvalda lager per material
- Möjlighet till rangordning samt justering av transparensen för respektive material
- Åtkomst till materialets metadata, beskrivning samt symbolförteckning

LPII består av fyra integrerade tjänster: karttjänsten, portalen, Graphical Survey Tool (GST) frågetjänsten samt den bildbank jag kommer gå in närmare på i senare kapitel. I dagsläget är det enbart de två första tjänsterna som är tillgängliga för allmänheten men under år 2012 ämnar LPII publiceras utan betastämpel med alla fyra tjänster tillgängliga.

Karttjänsten är, såsom namnet säger, kartdelen av tjänsten där man kan skapa och se på GIS-material. Karttjänsten är ganska naturligt den mest centrala tjänsten och alla andra delar är på ett eller annat sätt kopplade till karttjänsten. I figur 1 ser man kartan med "Turun seudun palvelut" som valt material i form av röda och gröna markörer. I den vänstra panelen, syns en trädmeny med alla tillgängliga material. I den högra panelen syns mera detlajerad information om valt material.



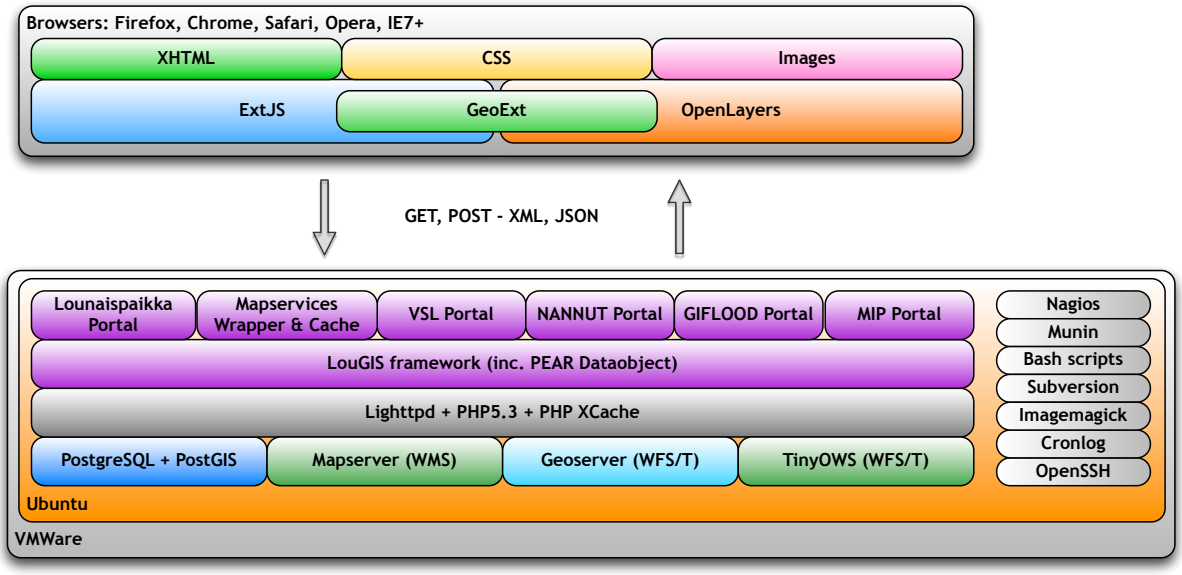
Figur 1. Lounaispaikkas karttjänst LPII.

Portalen är i princip en kopia av Lounaispaikkas tidigare hemsida, där man hittar grundinformation som t.ex. kontaktuppgifter och projektbeskrivningar. Portalen fungerar också som en informationskanal för aktuella GIS-händelser och liknande i området.

GST är en tjänst där man kan skapa frågor och undersökningar anknutna till ett visst material och en viss position. GST utvecklades under samma period som bildbanken och likt bildbanken utfördes all utveckling på en egen server. Därför syns det ingenting av denna tjänst på alla skärmdumpar som finns i detta arbete.

3 Tekniken bakom LP II

Grundtjänsten LP II är uppbyggd av en mängd olika bibliotek och program och i detta stycke går jag igenom de mest elementära delarna för att ge en bättre förståelse hur tjänsten är uppbyggd. LP II baserar sig på en modifierad version av LAMP(Linux, Apache, MySQL och PHP), där MySQL är utbytt mot databashanteraren PostgreSQL. Figuren nedan är ritad av Lounaispaikkas huvudplanerare och illustrerar hur de olika delarna är relaterade.



Figur 2. Strukturen för LPII. Pyry Liukas, Lounaispaikka 2010

Systemet är uppbyggt så, att det lätt skall gå att skapa modifierade versioner av tjänsten, som alla bygger på samma IT-struktur. Det är främst fråga om att erbjuda olika anpassade versioner av karttjänsten med egen design och funktionalitet. Kunden får således relativt lätt tillgång till en skräddarsydd version av tjänsten. Exempel på olika utföranden av tjänsten kan ses i det översta lagret av den nedre bilden i figur 2.

3.1 PHP

PHP är ett skriptspråk som används för att hantera dynamiskt innehåll på en webbsida. PHP körs på en webserver och används bl.a. för att kommunicera med en databas eller för att hantera input och output. PHP går att bädda in i vanlig HTML-kod vilket gör att det lämpar sig för allt från enkla hemsidor till avancerade applikationer.

Man kan antingen bädda in HTML i PHP-kod eller tvärtom, dvs. bädda in PHP-kod i HTML. Båda teknikerna, som demonstrerats i kodblock 1 respektive kodblock 2, är accepterade. Jag baserar mitt val av teknik på mängden HTML jämfört med PHP. Är det fråga om en PHP-klass, vars främsta funktion är att genererar HTML, använder jag den senare tekniken där HTML-koden kommer fram tydligare. Är det däremot fråga om en PHP-klass vars viktigaste funktion är att göra beräkningar är den första tekniken att föredra.

Nedan följer ett exempel där där två olika PHP-skript genererar samma HTML-resultat. Skriptet skapar en datastruktur av typen array och skriver sedan ut elementen i en HTML-paragraftag.

Kodblock 1. Ett PHP skript med HTML koden inbäddad.

```
<?
$demoArray = array('item1', 'item2', 'item3');
foreach($demoArray as $item) {
    echo '<p>' . $item . '</p>';
}
?>
```

Kodblock 2. Ett PHP skript med HTML koden utanför php koden.

```
<?
$demoArray = array('item1', 'item2', 'item3');
foreach($demoArray as $item) {
    ?><p><?=$item?></p><?
}
?>
```

3.2 PostgreSQL och PostGIS

PostgreSQL är en gratis databashanterare baserad på öppen källkod. PostgreSQL syntax, som följer SQL-standarden i stor utsträckning, liknar MySQL och Microsofts SQL Server.

PostgreSQL lämpade sig utmärkt för LPII främst därför att det finns ett likaså gratis tillägg PostGIS som erbjuder avancerad GIS-funktionalitet. PostGIS möjliggör t.ex. lagring av koordinater av olika format direkt till databasen. Databashanteraren stöder som standard även relationer, vilket är en stor fördel jämfört med MySQL, som kräver databasmotorn InnoDB för att möjliggöra användningen av relationer.

3.3 MapServer

MapServer är ett verktyg för att visa kartor på webben. Projektet har öppen källkod och startade som ett projekt i University of Minnesota i sammarbete med NASA. MapServer är flexibelt och stöder de flesta plattformer inklusive Windows, Linux och Mac OS X. Förutom det breda plattformstödet kan MapServer utvecklas i en rad olika utvecklingsmiljöer: PHP, Python, Perl, Ruby, Java och .NET. (Mapserver 2010).

3.4 OpenLayers

OpenLayers är ett öppet källkodbibliotek skrivet i JavaScript, som erbjuder kartfunktionalitet. Med hjälp av OpenLayers går det smidigt, att få en karta ut på webben. OpenLayers erbjuder i sitt grundutförande elementära funktioner som förstoring, förminskning, förflyttning samt diverse verktyg för att mäta avstånd och areal samt visualisering av t.ex. punkter och områden.

OpenLayers stöder en mängd olika kartkällor, bl.a. Google Maps, OpenStreetMap, Yahoo Maps, Web Map Service(WMS) samt Web Feature Service(WFS). Lounaispaikka använder sig av WMS för att visualisera kartorna som finns på servern medan WFS sköter visualiseringen av materialen. (OpenLayers Documentation 2010).

3.5 LouGIS

LouGIS är namnet på det internt utvecklade ramverk, som fungerar som grund för alla delprojekt. Ramverket är uppbyggt som en kraftigt modifierad version av MVC arkitekturen. MVC står för Model-View-Controller och går ut på, att dela in koden enligt ansvarsområde och således göra koden mera flexibel och återanvändbar. Model sköter om hanteringen av data, Controllern hanterar input och View sköter först och främst det synliga, dvs. användargränssnittet.

Ramverket är uppbyggt av olika moduler, som ansvara för ett respektive område. Jämför man med standard-MVC kan man säga, att modulerna motsvarar Controllern. Modulerna kommunicerar med databasmodellerna, som ligger i databaslagret och motsvarar således Model i MVC. För databasmodellerna används Pear Data Object, som genererar PHP-objekt av databasens relationer. Gränssnittet drivs så gott som enbart av Ext JS javascript-biblioteket med vissa undantag där modulerna skriver ut ren HTML. I normala fall returnerar modulen data i JSON-format och HTML-koden genereras först av klienten.

Modulerna anropas genom, att t.ex. kalla på http://lounaispaikka.fi/image/getImageById/, där image är modulen och getImageById är funktionen eller action, som det heter inom LouGIS-ramverket. Denna metod är både enklare och snyggare än den traditionella, där adressen hade varit http://lounaispaikka.fi?m=image&a=getImageById.

4 Web 2.0

Termen Web 2.0 är en vida använd term, vars betydelse inte är så lätt att slå fast exakt. Det är fråga om en ganska generell beskrivning av webplatser utvecklade med moderna verktyg och tekniker. Det är fråga om tjänster, som utnyttjar AJAX (Asynchronous Javascript And XML) för att uppdatera innehållet dynamiskt eller tjänster som överlag använder sig av tekniker, som endast funnits i applikationer tidigare.

4.1 Asynchronous Javascript and XML (Ajax)

Eftersom Lounaispaikkas webtjänst helt bygger på Ajax-teknik, redogör jag kort för vad Ajax är och hur det används i praktiken i detta projekt. Ajax förknippas ofta med Web 2.0-utveckling och Ajax är mycket riktigt en viktig del av hela det konceptet.

Som namnet säger, är det fråga om en asynkron Javascript-process, som involverar XML(Extensible Markup Language). Det är förenklat sagt en teknik för att utbyta data mellan klienten och servern. Den största skillnaden till traditionell Web 1.0-utveckling är, att innehållet kan uppdateras med data från servern utan att sidan laddas om. Ett traditionellt formulär, som skickar uppgifter till servern medför, att sidan laddas om och eventuella meddelanden till användaren visas först efter omladdningen. Använder vi oss av Ajax i samma formulär behöver vi inte ladda om sidan utan alla data från formuläret skickas via Ajax i bakgrunden till servern varifrån vi får ett svar, som vi sedan kan informera användaren om. (Brinzarea-lamandi & Darie & Hendrix 2009, s. 16).

XML är en del av Ajax eftersom trafiken mellan klienten och serven sker i form av XML data. I dagens läge är dock XML i många fall utbytt mot JSON(JavaScript Object Notation) eftersom det är bättre anpassat för bearbetning med Javascript. JSON är lättare att producera på servern och går att använda direkt utan att konvertera i Javascriptkoden. Med PHP kodar man om en räcka till ett JSON-objekt med ett simpelt kommando: json_encode($array).

Att processen är asynkron innebär, att efterföljande kod efter ett Ajax-anrop, fortsätter exekvera omedelbart utan att vänta på att anropet är klart. Detta resulterar i att vi kan ha

flera aktiva Ajax anrop och således t.ex. uppdatera olika delar av innehållet oberoende av varandra.

Ett ajax-anrop är i sin enklaste form uppbyggt av 4 delar: adressen där svarsfunktionen finns, en samling argument som skickas med till funktion samt en funktion för lyckat respektive misslyckat anrop. I kodblock 3 nedan görs ett anrop till getData.php skriptet med argumentet foo innehållandet textstringen "bar".

Kodblock 3. Syntaxen för ett Ajax anrop med ExtJS.

```
Ext.Ajax.request({
    url: 'getData.php', // url till server skript
    success: function(r) { // funktion vid lyckat anrop
        // bearbeta serverns svar här
    },
    failure: function(r) { // funktion vid misslyckat anrop
        alert(r.error);
    }
    params: {
        foo: 'bar'
    }
});
```

4.2 Ext JS

ExtJS är ett JavaScript-bibliotek utvecklat av Sencha, som erbjuder ett brett utbud av komponenter och funktionalitet som behövs vid modern webbutveckling. I detta projekt var det länge oklart ifall utvecklingen skulle ske med hjälp av ExtJS eller Mootools. Avgörande faktor vid valet blev det, att det vid sidan av ExtJS finns ett bibliotek för karthantering, som bygger direkt på ExtJS, nämligen GeoExt. GeoExt kan ses som ett tillägsbibliotek till ExtJS som erbjuder komponenter speciellt anpassade för karttjänster. ExtJS erbjuder ett stort utbud grundkomponenter, såsom fönster och paneler, vilket gör att det relativt snabbt går att skapa grundläggande gränssnitt med tillhörande funktionalitet. Mootools är ett Javascript-bibliotek som erbjuder liknande funktionalitet som ExtJS.

Man kan säga, att komponenterna är uppbyggda stegvis på ett sätt så att alla komponenter bygger vidare på en liknande, men mera generell föregångare. Som ett exempel på detta kan vi ta oss en närmare genomgång av en av de mest elementära komponenterna, nämligen Ext.Window. Som ses i figur 3, ärver Ext.Window klassen egenskaper från 5 andra komponenter.

Figur 3. Alla de klasser Ext.Window ärver i tur och ordning.

Ext.Window är en klass, som erbjuder grundfunktionalitet som man förväntar sig av ett fönster. Ett fönster är en av de yttersta klasserna, dvs. det ligger längst ut om man ser på komponentklasserna som en kedja, där alla egenskaper ärvs vidare och egenskaperna blir alltmer specifika ju längre ut man kommer. Som exempel på detta kan vi ta Ext.Panel som ligger närmast Ext.Window och således har mest liknande egenskaper. I praktiken har Ext.Window och Ext.Panel identiska egenskaper förutom några specifika egenskaper som t.ex. att man kan stänga, flytta och ändra storlek på fönstret. De flesta egenskaperna, som används för ett fönster, ligger alltså i Ext.Panel-klassen. Elementära egenskaper, som t.ex. händelsehantering härstammar ändå från Ext.Observable-klassen. Att skapa en komponent är trivialt och kräver i de flesta fall endast ett fåtal obligatoriska parametrar, som ses i kodblock 4 nedan. Koden resulterar i fönstrer som ses i figur 4. (Sencha Docs 2010, Ext.Window)

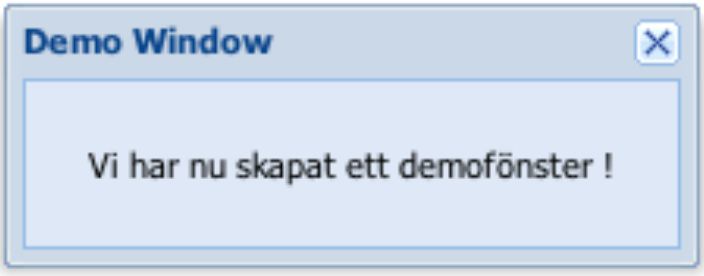
Kodblock 4. Ett konfigurationsobjekt som används för att skapa ett fönster av klassen Ext.Window.

```
// Körs när ExtJS biblioteket är laddat
Ext.onReady(function() {

    // skapar en instans av klassen Ext.Window
    var demoWin = new Ext.Window({
            id: 'demoWin',
            title: 'Demo Window',
            autoWidth: true,
            autoHeight: true,
            padding: '20 20 20 20',
            html: 'Vi har nu skapat ett demofönster !'
    });

    demoWin.show();

});
```



Figur 4. Komponenten vi skapade med kodblocket ovan.

5 Implementering av bildbanken

Som grund för hela bildbankprojektet ligger ett gammalt projekt EyeGIS som utförts på Yrkeshögskolan Novia. EyeGIS ingår i Novias karttjänst och används ännu i skrivande stund av vissa GIS-projekt i Raseborg. Grundidén är den samma i Lounaispaikkas bildbank men tekniken och utförandet avviker på de flesta punkter.

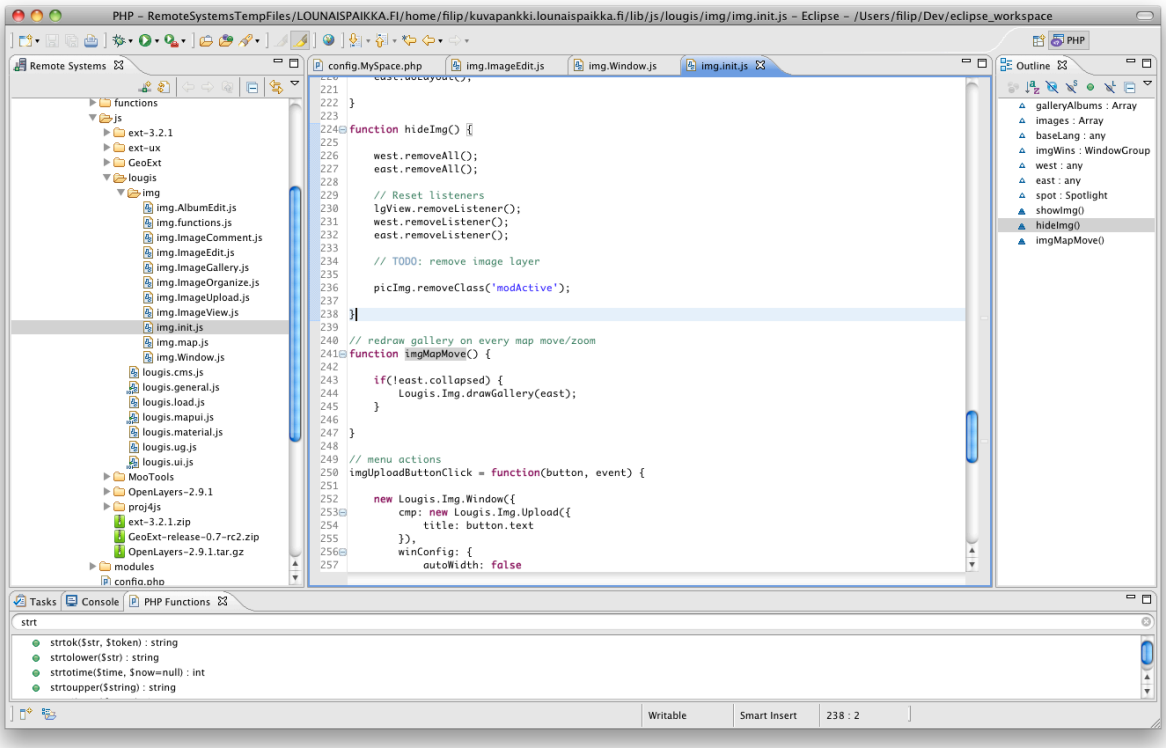
5.1 Verktyg

För att effektivera och underlätta utvecklandet finns det en mängd program och verktyg att välja mellan. Jag går nedan igenom de verktyg, som jag anser varit de viktigaste i detta projekt.

5.1.1 Eclipse

Under projektets gång har Eclipse använts som miljö för all utveckling. Valet föll på Eclipse främst p.g.a. att den på ett utmärkt sätt stöder editering av filer direkt på servern. Eftersom utvecklingen endast skett på en testserver, underlättar detta tillvägagångssätt avsevärt, då man inte behöver flytta filer via t.ex. en ftp-klient. Eclipse bygger på öppen källkod och har ett stor antal tillägg med t.ex. syntaxhjälp för alla populära programmeringsspråk. I detta projekt använde jag mig av ett tillägg som erbjuder automatisk komplettering av ExtJS-kod. Största nyttan med t.ex. detta tillägg är, att man lätt upptäcker triviala fel i koden, som annars kan slinka igenom och inte nödvändigtvis vara så lätta att hitta i efterhand.

Eclipse är ett flexibelt verktyg som går att anpassa efter ens egna behov. I figur 5 nedan ser vi t.ex. att jag delat in verktyget i 4 separata fönster. Filstrukturen till vänster och klassstrukturen till höger. I det nedersta fönstret syns dokumentation för aktuellt utvecklingsspråk.

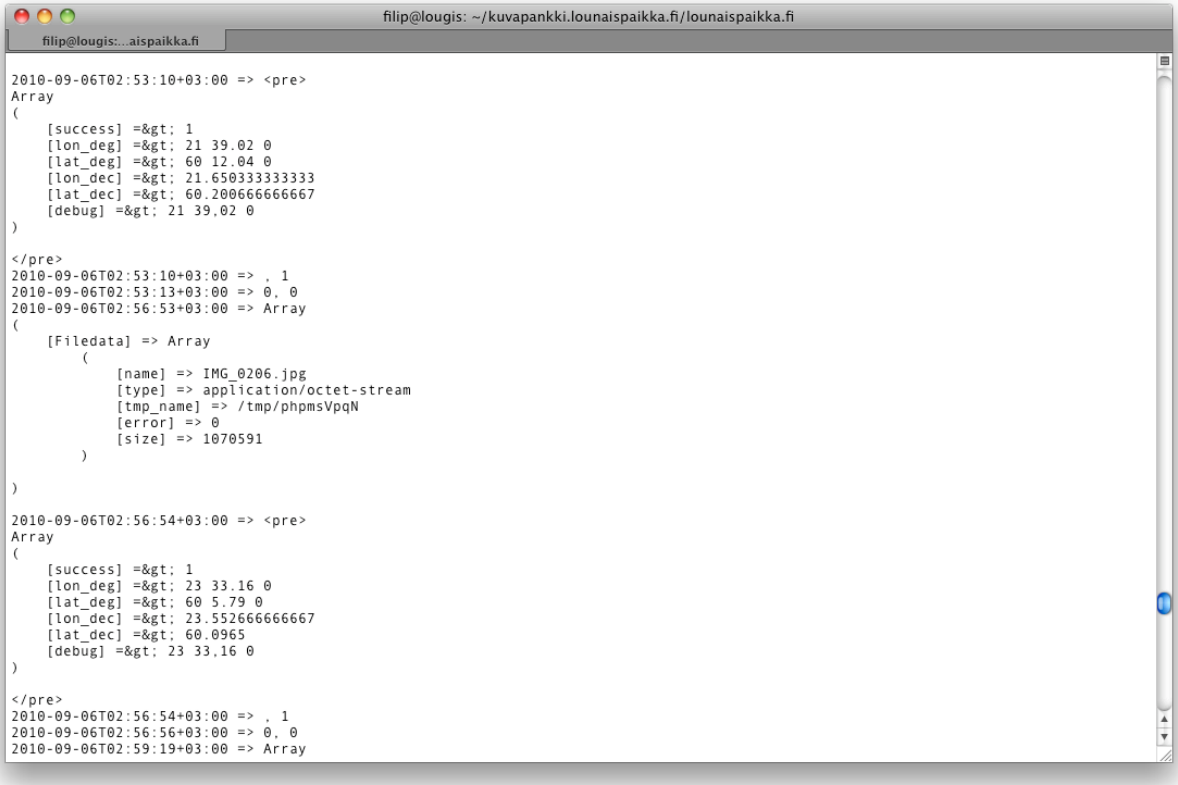


Figur 5. Utvecklingsverktyget Eclipse.

Generellt sett är utveckling direkt på servern inte något att rekommendera, men i detta fall var det oundvikligt, eftersom det hade varit för resurskrävande att köra alla nödvändiga processer för LPII. Eftersom jag hade min egen testserver och ingen annan deltog i utevecklandet förorsakade utveckling direkt på servern inga problem. Är man flera utvecklare inom ett projekt är det svårt att undvika filkonflikter och överskrivningar ifall man jobbar direkt på en server.

5.1.2 Terminalen

Terminalen är ett oundvikligt verktyg när man arbetar mot en server. Genom att ta kontakt till servern via SSH kommer man åt att läsa av och modifiera data. Förutom hantering av filer och dess rättigheter använde jag terminalen för att logga data. Denna logg hade jag stor användning av främst för att hitta fel. Genom att ange kommandot tail -f samt loggens filnamn uppdateras loggens innehåll i realtid. I figur 6 ses ett exempel från loggen där data om en bilduppladdningn samt tillhörande koordinater syns.



Figur 6. Serverlogen som visar det loggade värdet samt tidpunkt.

5.2 Planering

Som ett naturligt första steg av projektet började jag med planering. Planeringen indelades mer eller mindre medvetet i tre faser: undersöknings-, dokumenterings- och extern design-fasen.

Det första steget av planering var alltså att undersöka dels liknande tjänster men även de verktyg som skulle användas. Det finns en hel del liknande tjänster ute på nätet, som

erbjuder geotaggning av foton. Jag studerade och dokumentera totalt över 10 tjänster för att skapa mig en så bra uppfattning som möjligt av vad som gör en bra GIS-bildtjänst. Bland de tjänsterna jag undersökte kan man nämna Google Panoramio, Flickr samt Locr, som jag kanske fick mest idéer och inspiration av. I tabell 1 ses ett sammandrag av undersökningen. Eftersom ett av de viktigaste kraven på tjänsten var ett användarvänligt gränssnitt, satte jag stor vikt på detta i det här skedet. Hur är alla bildfunktioner lösta med tanke på användaren? Olika lösningar vad gäller uppladdning av bilder var också ett av de områden jag fokuserade på.

Tjänst	Positivt	Negativt
Google Panoramio	Möjlighet för användarna att föreslå ändring av koordinater. Bra stöd för kommentering av bilderna.	Inget stöd för uppladdning av flera bilder samtidigt. Inget stöd för att ange koordinater för många bilder samtidigt.
Locr	Bra gallerifunktion. Kartikonen och galleribilden visuellt sammankopplade. Bildens riktning anges visuellt på kartan.	Onödigt små versioner av bilderna tillgängliga.

Tabell 1. Analys av liknande tjänster.

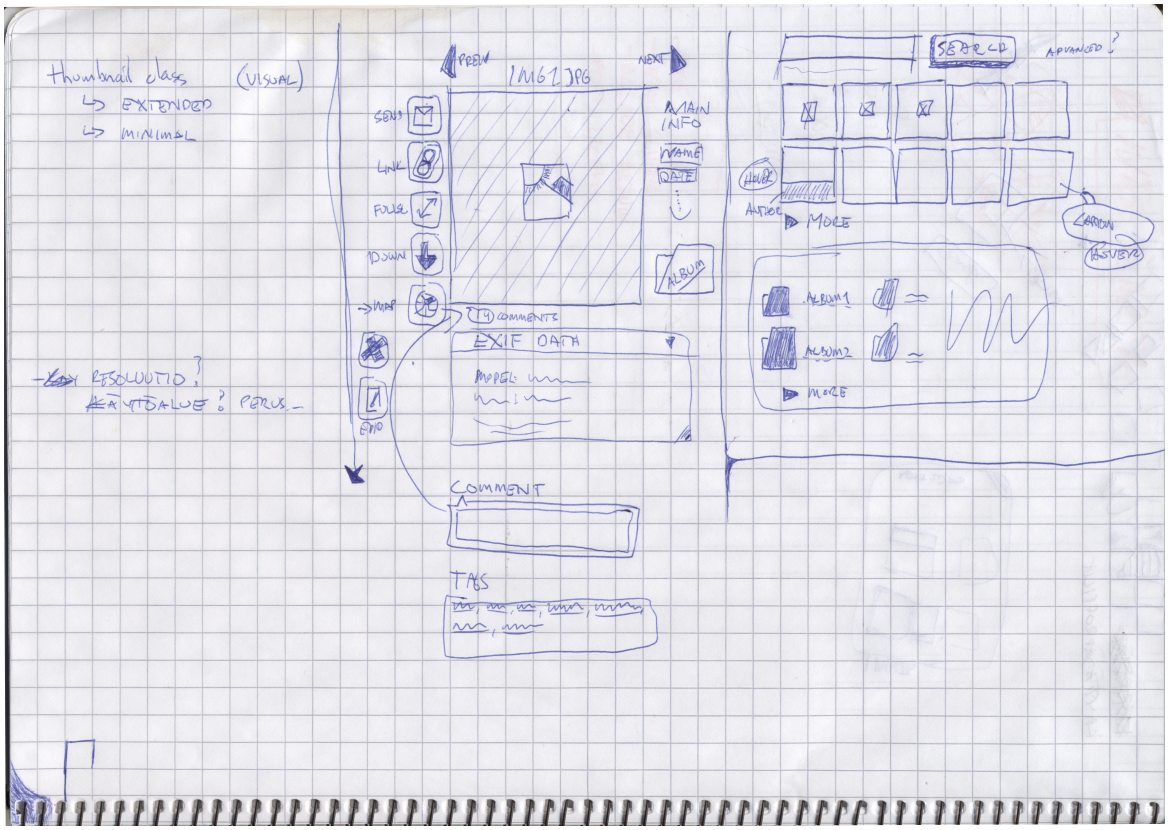
Förutom existerande tjänster bekantade jag mig även med de verktyg jag hade att jobba med för att lättare kunna relatera till de tjänster jag studerade. Med verktyg syftar jag då främst på ExtJS, som i stor grad styr utseendet av användargränssnittet.

Dokumenteringen bestod i det här skedet av anteckningar och en stor lista med plus och minus. Dessa plus och minus bestod av egna och andras idéer, observationer samt krav från kunden. Kundens krav var inte så strikta, så i det här skedet fanns det ännu möjlighet att stryka eller lägga till detaljer på projektplanen. I det här skedet var kanske det svåraste, att veta vilka funktioner den kommande användargruppen ville ha, eftersom hela Lounaispaikkas verksamhet var mer eller mindre okänd för mig.

Externa designen bestod sedan i sin tur av att få ner de idéer och visioner man kommit fram till på papper. Användargränssnittet skissades ner ett flertal gånger och bearbetades

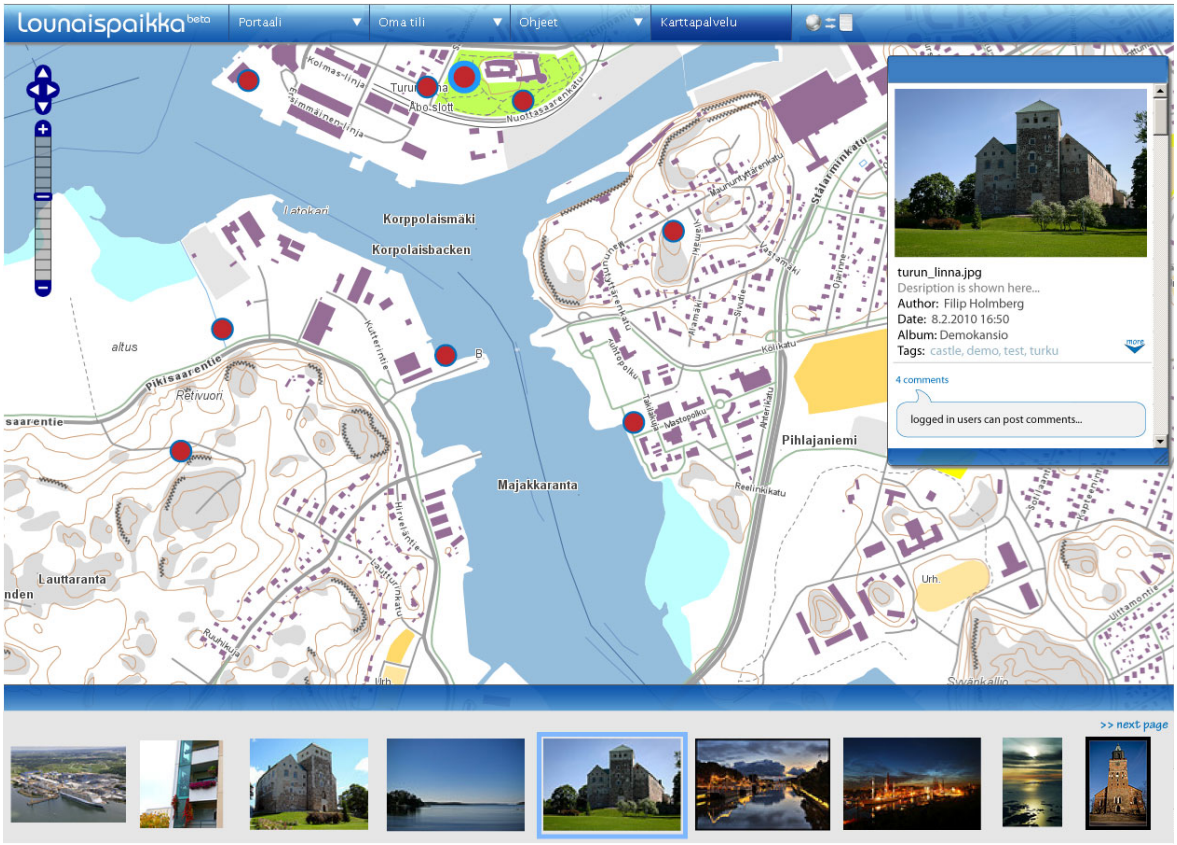
vidare utgående från kommentarer från kunden. Med facit på hand skiljer sig slutresultatet ganska radikalt från det, som skissades upp i i början av planeringsskedet. Detta beror främst på att utseendet på LPII ändrade en bit in på projektet. Men trots att användargränssnittet blev att se annorlunda ut, anser jag inte att någonting gick till spillo, utan alla idéer från detta stadium inverkade garanterat på slutresultatet på ett positivt sätt.

Nedan följer två figurer från olika faser av planeringen. Den handskissade versionen i figur 7 är det första jag konkret ritade ner på papper medan figur 8 är bland de sista.
Figur 7 illustrerar en fas där det är fråga om att få ut idéer till något konkret för att sedan därifrån kunna arbeta vidare. Syftet är inte ens att få fram någon realistisk prototyp skiss utan endast att "tänka konkret" med penna och papper.



Figur 7. En handgjord skiss över hur redigeringen av bildernas data kunde se ut.

Figur 8 är en tidig version av bildläget, där jag använt mig av en horisontell komponent för att visa bilderna, istället för den vertikala som slutligen blev använd.

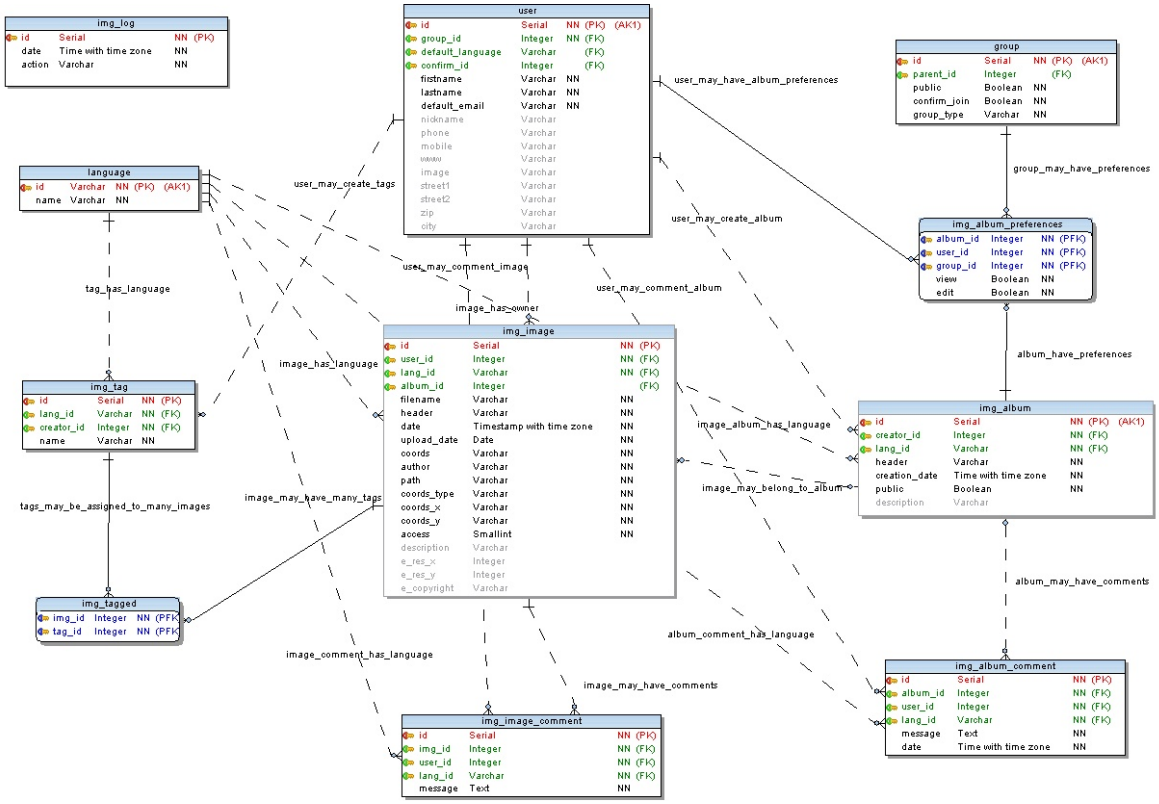


Figur 8. En tidig version av bildläget.

5.3 Databasstruktur

För att få en bra grund för ett projekt är databasdesignen av stor vikt. Jag skissade upp min första databasdesign på papper med enbart relationerna synliga, för att få en bild av hur det kunde fungera. Först senare, när allt planeringsarbete var klart, överförde jag skissen till en verklig databasmodell.

Databasen är planerad och ritad i programmet Toad Data Modeller. Figuren 8 nedan är en vy från programmet med den slutgiltiga databasstrukturen.



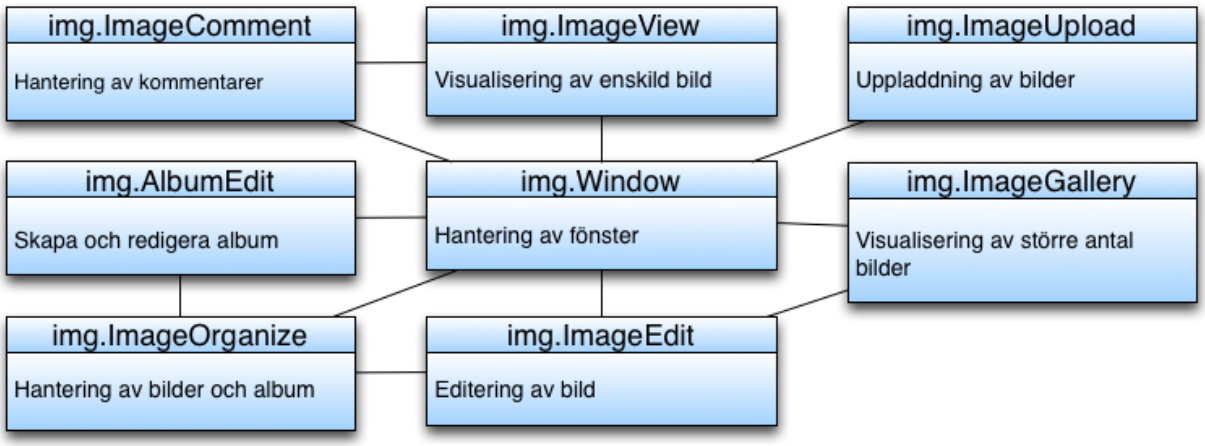
Figur 8. Bildbankens databasstruktur i sin helhet.

Den mest centrala relationen är naturligt nog img_images, där alla data om bilderna finns lagrade. Enligt kravspecifikation kan en bild endast höra till ett album, vilket underlättar både databasstrukturen och användargränssnittet.

Jag använde mig av referensintegriteter för att validera innehållet, men även för att hålla databasen ren från onödiga och felaktiga data. Om referensintegriteterna är korrekta, kan man t.ex. genom att radera alla användare, tömma alla användarrelaterade data utan att skilt behöva gå igenom respektive relation. En refenresintegritet innehåller information om hur databashanteraren skall reagera då data modifieras. För att få en funktionalitet enligt exemplet ovan, måste referensintegriteten vara definierad så, att relaterade data även raderas när huvudvärdet raderas. I bildbankens databas används denna typ av referensintegriteter mellan t.ex. användare och kommentarer, vilket resulterar i, att när en användare raderas så raderas automatiskt alla kommentarer gjorda av användaren. (Padron-McCarthy 2005, s. 75).

5.4 ExtJS struktur

Jag har valt att dela in de olika gränssnittsmodulerna i egna Javascript-klasser. Dels för att detta följer ExtJS rekommendationer, men främst för att de gör modulerna återanvändbara. Dessa klasser är alla utvidgade versioner av ExtJS-komponenter.



Figur 9. Simplifierat klassdiagram över bildbankens ExtJS klasser.

5.4.1 Utvidgning av ExtJS-klass

Jag har använt mig av metoden Ext.extend för att utvidga ExtJS-bibliotekets klasser. Genom att använda Ext.extend kan man skriva över existerande klassmetoder. I praktiken fungerar Ext.extend så, att man som namnet säger, utvidgar en av bibliotekets existerande klasser. Fördelen med detta är, att man således inte behöver skapa allting från grunden utan kan använda sig av en liknande klass som grund. Nedan följer ett exempel på utvidgning av en ExtJS-klass. (Ramon 2009, s. 31).

5.4.2 img.Window-klassen

För att ge en bättre bild av hur man utvidgar ExtJS-klasser har jag valt img.Window som exempel. Denna klass är central i hela bildbanken, eftersom de flesta funktioner fungerar inne i ett fönster av klassen img.Window. Klassen är alltså en utvidgad version av ExtJS standard-fönsterklass. Dvs. klassen har alla grundläggande egenskaper som ett standard-

fönster har, men med ett antal tillägg som img.Window tillför. Klassen finns i sin helhet i bilaga 2.

Ett standardfönster går att förminska så att enbart rubriken blir synlig. För att förbättra denna funktionalitet, har img.Window-klassen en mera avancerad förminskningsmetod, som organiserar alla förminskade fönster längst ner på rutan med en fade animation.

I kodblock 5 nedan ser vi funktionen för hanteringen av förminskade fönster i sin helhet. I funktionens övre del deklareras värden, som används i funktionen, för att optimera prestandan. Som exempel på detta kan vi ta variabeln destPos, som innehåller kartkomponentens position. För att inte behöva beräkna värdet flera gången i funktionen, lagrar vi det värdet i en intern variabel. Funktionen itererar sedan igenom alla fönster för att beräkna hur långt från vänstra kanten det aktuella fönstret skall placeras. I slutet anger vi fönstrets position och visar fönstret genom en 'slideIn' animation som hör till ExtJS-bibliotekets standardanimationer.

Kodblock 5. Funktionen som placerar de minimerade fönstren vid kartans nedre kant.

```
/**
 * Docks window to the mappanel bottom
 */
dock: function() {

    var dockEl = Ext.get(this.id); // element

    var destPos = Ext.getCmp('mappanel').getPosition(); // position of "dock" component
    var destSize = Ext.getCmp('mappanel').getSize(); // comp size
    var marginX = 10; // margin from left edge
    var marginY = destSize.height; // margin from top edge
    var winX = 0;
    var collapsedWins = 0;

    // loops imgWins window collection
    imgWins.each(function(win) {

        if(win.id.substr(0,4) != 'ext-') {  // exclude autogenerated windows(alert etc.)
            if(win.collapsed && win.id !== this.id) {
                if(winX < win.getPosition()[0]) { winX = win.getPosition()[0]+win.getWidth(); }
                collapsedWins++;
            }
        }

    }, this);

    if(collapsedWins === 0) {
        winX = destPos[0]+10;
    }

    this.setPagePosition(marginX+winX, destPos[1]+marginY-this.getHeight());
    dockEl.slideIn('b', {
        easing: 'easeIn',
        duration: .3
    });
    dockEl.frame();

}
```

Dock-funktionen anropas när fönstrets interna collapse-händelse inträffar som vi ser i kodblock 6. Collapse-händelsen inträffar när användaren klickar på ikonen som indikerar minimering i fönstrets högra hörn. Resultatet av funktionen ses i figur 10 där två fönster är minimerade och placerade längs med sidans nedre kant.

Kodblock 6. Anropet till dock funktionen.

```
this.listeners = {
    'collapse' : {
        fn: function(cmp) {
            this.hide();
            this.dock();
        },
        scope: this
    }
```

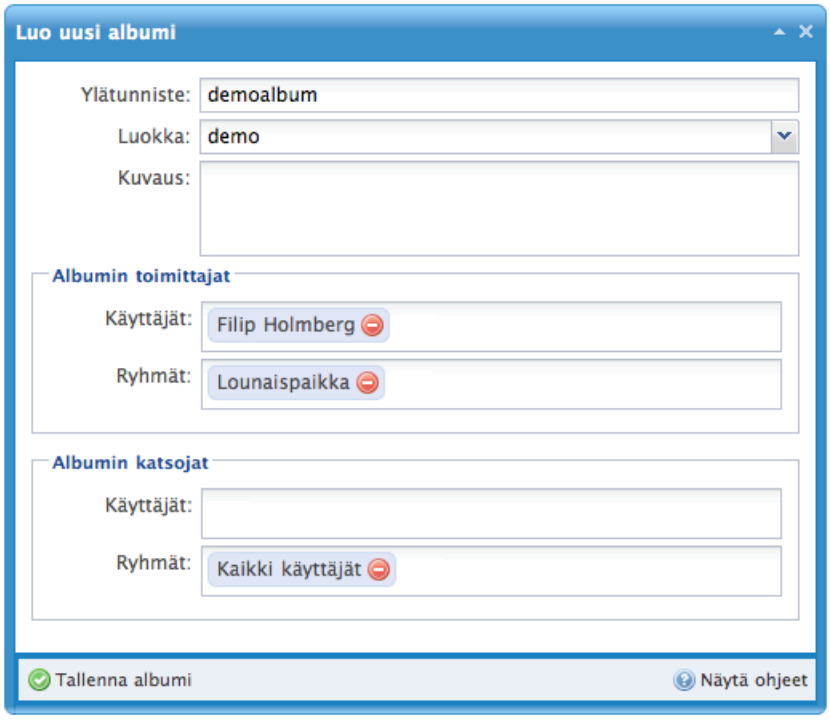
Figur 10. Två fönster som är minimerade med img.Window klassens dock() funktion.

Bildbankens fönsterklass har även en funktion för att visa information i en sidopanel. I dagsläget används funktionen för att visa instruktioner i en panel till höger, men funktionen är flexibel nog att visa extra paneler på alla fyra sidor. Det är fråga om separata fönster som placeras under huvudfönstret så att de upplevs höra ihop.

5.5 Hantering av bildalbum

För att organisera bildbankens bilder används album. Ett album är en samling bilder, som är relaterade antingen innehållsmässigt eller användarmässigt. När användaren skapar ett album, kan hon välja om albumet skall delas med andra användare/grupper eller enbart göras tillgängligt för visning åt andra användare/grupper. Album går också att göra helt publika, vilket i praktiken beyder att de är synliga för alla användare.

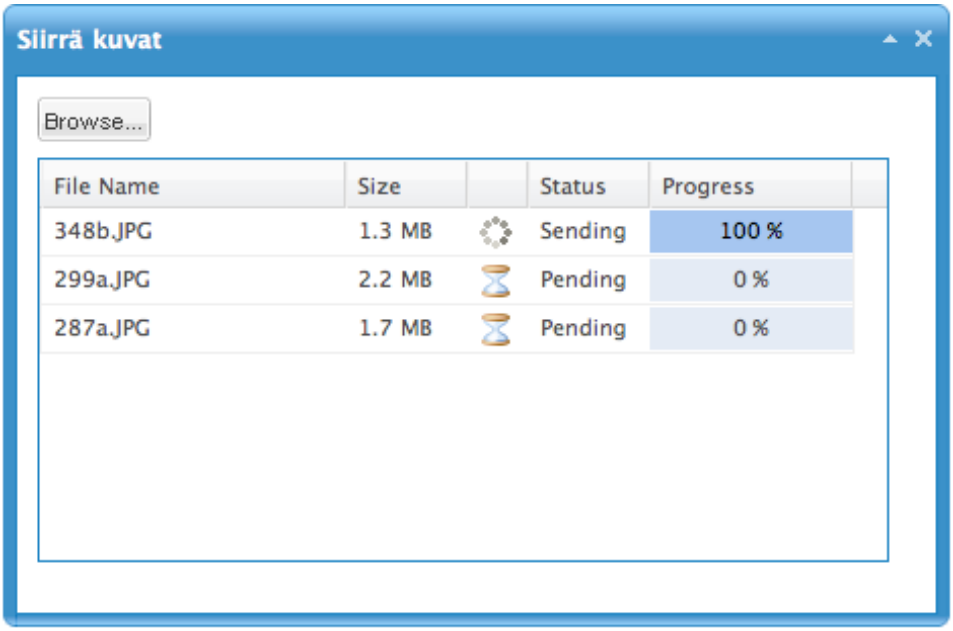
För att underlätta skapandet av album används ett sk. “auto-complete”-fält där användaren fyller i de första bokstäverna av önskad person eller grupp, varefter systemet visar vilka möjliga alternativ som finns tillgängliga. Denna teknik är vanligt förekommande i moderna webtjänster som t.ex. Facebook där tekniken är utnyttjad i meddelandefunktionen när man väljer mottagare. I figur 11 syns formuläret med två såväl användare som grupper valda.



Figur 11. Formuläret för att skapa ett nytt album.

5.6 Uppladdning av bilder

Tjänstens kanske viktigaste funktion, uppladdningen av bilder, var också den del av projektet, som tog längst tid att slutföra. Själva filöverföringen till servern fungerar med flash. Flash är den mest använda metoden, eftersom den är webbläsaroberoende till skillnad från t.ex. HTML5, som enbart fungerar i moderna webbläsare. För att flytta bilder till servern väljer användaren de önskade bilderna i den standarddialog som visas. Efter det, att bilderna är valda, börjar överföringen omedelbart och användaren kan följa med förloppet via en indikator enligt figur 12 nedan. När en bild är överförd visas en redigeringsmodul åt användaren. I modulen kan användaren ändra på titel, fotograf, datum & tid, koordinater samt beskrivning. Om användaren överför flera filer är det möjligt, att redigera de överförda filerna medan de resterande ännu överförs.



Figur 12. Dialogen för filöverföring.

När bilderna har nått servern, valideras filerna för att försäkra sig om, att de inte blivit korrupta under flytten. För att underlätta användningen av filerna skapas en mindre version av bilden, som går snabbare att ladda. Servern lagrar bilderna enligt användare, vilket resulterar i följande filstruktur: /cms/user/images/.

Det mest väsentliga med bilderna i denna tjänst är, att dess koordinater går att ange manuellt direkt från kartan, genom att klicka på önskad position. Om bilden har fått koordinater tilldelade av kameran, överförs dessa automatiskt och det framgår för användare att bilden redan har koordinater. Oftast är de koordinater, som bilden fått automatiskt mera exakta än de man anger manuellt. Ifall de är felaktiga av någon orsak, går de att ändra genom metoden beskriven ovan. När användaren anger koordinater för en bild via kartan, sparas även aktuell skala som en mätare på hur exakt bilden är placerad. Denna information används i sin tur när bilderna visas på kartan. Jag går närmare in på detta i kapitel 5.8 Visning av bilder.

För att underlätta redigeringen av ett större antal bilder, kan användaren välja att kopiera metadata till alla övriga bilder. Det naturliga är, att användaren laddar upp flera bilder från ett och samma ställe och då kan de underlätta avsevärt om t.ex. metadata som fotograf, datum och beskrivning går att kopiera till alla bilder.

5.7 Redigering av bilddata

En av bildbankens mest elementära funktioner är redigering av bilddata. För att möjliggöra en så flexibel grund för redigeringen som möjligt, har jag skapat en ExtJS-komponent, som kan användas globalt inom tjänsten.

Redigeringskomponenten byggs upp genom att kalla på funktionen extImageEdit() i img-Modulen. Funktionen tar bildens ID samt en modeparameter som input och returnerar ett JSON-objekt som ExtJS-klassen använder för att skapa själva redigeringsfönstret. Mode-parametern används för att ange ifall det är fråga om en enskild bild eller en hel kollektion, vilket är aktuellt vid t.ex. uppladdning av bilder. Modulfunktionen hämtar alla relevanta data för bilden och formaterar resultat som en array, som sedan konverteras till JSON.

JSON-objektet behandlas av ExtJS-klassen som ett konfigurationsobjekt av typen 'xtype'. Detta innebär, att man anger komponentens klass i samband med de övriga parametrarna och kan således skapa ett ExtJS-objekt utan att skapa en ny instans av klassen som i vanliga fall. Tekniken kallas för 'lazy initialization', dvs. lat initialisering och betyder i praktiken, att man överlåter initialiseringen åt ExtJS-biblioteket. Denna teknik är speciellt användbar då man gör flexibel funktionalitet, som på förhand inte känner till vilken typ av komponent som skall skapas. Alla AJAX-anrop i denna tjänst går via en hjälpfunktion, som behandlar resultatet baserat på de parametrar man gett. För redigeringskomponenten skickar vi med en createWin-parameter, som indikerar att vi vill skapa ett fönster med anropets komponentobjekt som innehåll. Eftersom hjälpfunktionen sköter behandligen av alla resultatdata behövs ingen anropsspecifik funktion för detta som ses i kodblock 7 nedan. I kodblock 8 ses funktionen som reagerar på createWin-parametern och skapar fönstret.

Kodblock 7. Ett AJAX anrop via hjälpfunktionen Lougis.Img.ajaxRequest.

```
Lougis.Img.ajaxRequest({
    url: '/run/img/extImageView/',
    method: 'POST',
    params: {
        imageId: record.data.id
    },
    destCmp: '',
    createWin: true,
    spot: false,
    animateTarget: node.id
});
```

Kodblock 8. Ett fönster skapas om paramentern createWin är sann.

```
if(config.createWin) {
    new Lougis.Img.Window({
        cmp:cmp,
        winConfig: config.winConfig || {}
    }).show();
}
```

Som man ser i koden ovan är komponenten för att redigera bilder lätt tillgänglig och i skrivande stund används komponenten redan i tre olika sammanhang, vilket visar, att det verkligen lönar sig att skriva flexibel kod. Komponenten anropas från galleriet, samt från fönstren för uppladdning och albumhantering.

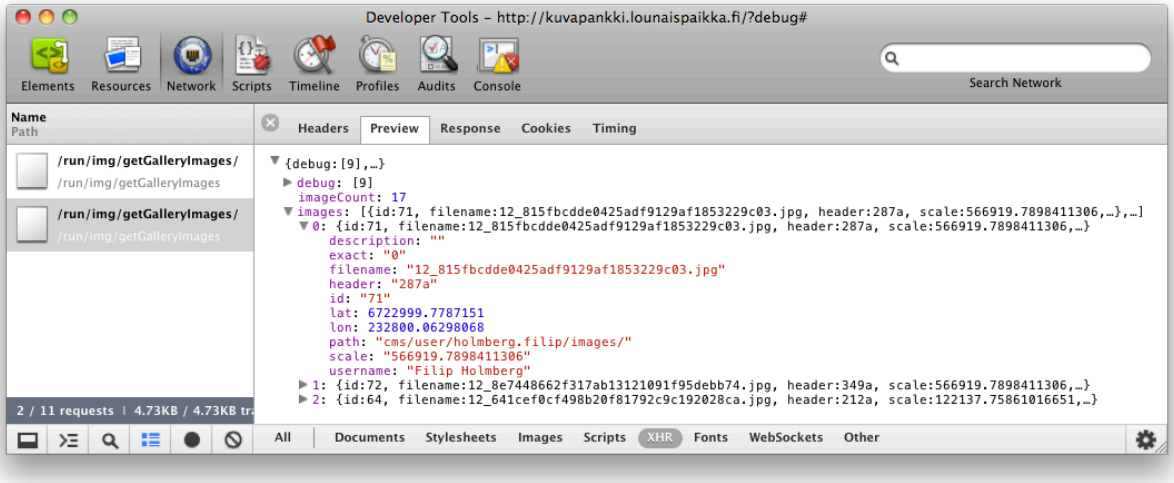
Komponenten för att redigera bildernas data består i sin enkelhet av ett formulär där uppgifterna går att ändra. Man kommer även åt hanteringen av album direkt från komponenten ifall man vill skapa ett nytt album för bilden.

5.8 Visning av bilder

Överförda bilder visualiseras åt användaren genom att visa ett galleri av bilderna samt markera dess position på kartan. Eftersom antalet bilder oftast är större än vad som ryms i galleriet på skärmen, markeras endast de bilder på kartan som visas på aktuell sida i galleriet. För att skapa ett visuellt samband mellan bilden och markören, reagerar båda när musen förs över dem genom att indikera vilken bild/markör som hör ihop. Genom att klicka på en bild får man upp en detaljerad vy av bilden enligt figuren nedan. Via den detaljerade vyn, som syns i figur 14, kan man få upp redigeringsmodulen, kommentera samt få upp EXIF(Exchangeable Image File Format)-data. EXIF är data som sparas i bildfilen av kameran och innehåller uppgifter som t.ex. datum och kamerainställningar.

Figur 14. En detaljerad vy av en bild i galleriet.

Galleriet uppdateras när användaren byter lokation på kartan för enbart vissa bilder, som är tagna inom det synliga kartområdet. Servern skickar information om vilka bilder som skall visas och var de är placerade i JSON-format som ses i figuren 15. Det totala antalet bilder finns även med i resultatet för att galleriet skall kunna organisera resultat på rätt sätt.



Figur 15. Visualisering av resultatet för ett serveranrop från galleriet.

Galleriet visar som standard två kolumner av bilder, men bredden går att justera manuellt för att bättre passa antalet bilder och skärmupplösning. Alternativt kan man även dölja galleriet helt och således enbart komma åt bilderna genom att klicka på indikatorerna på kartan.

5.9 Felsökning och testning

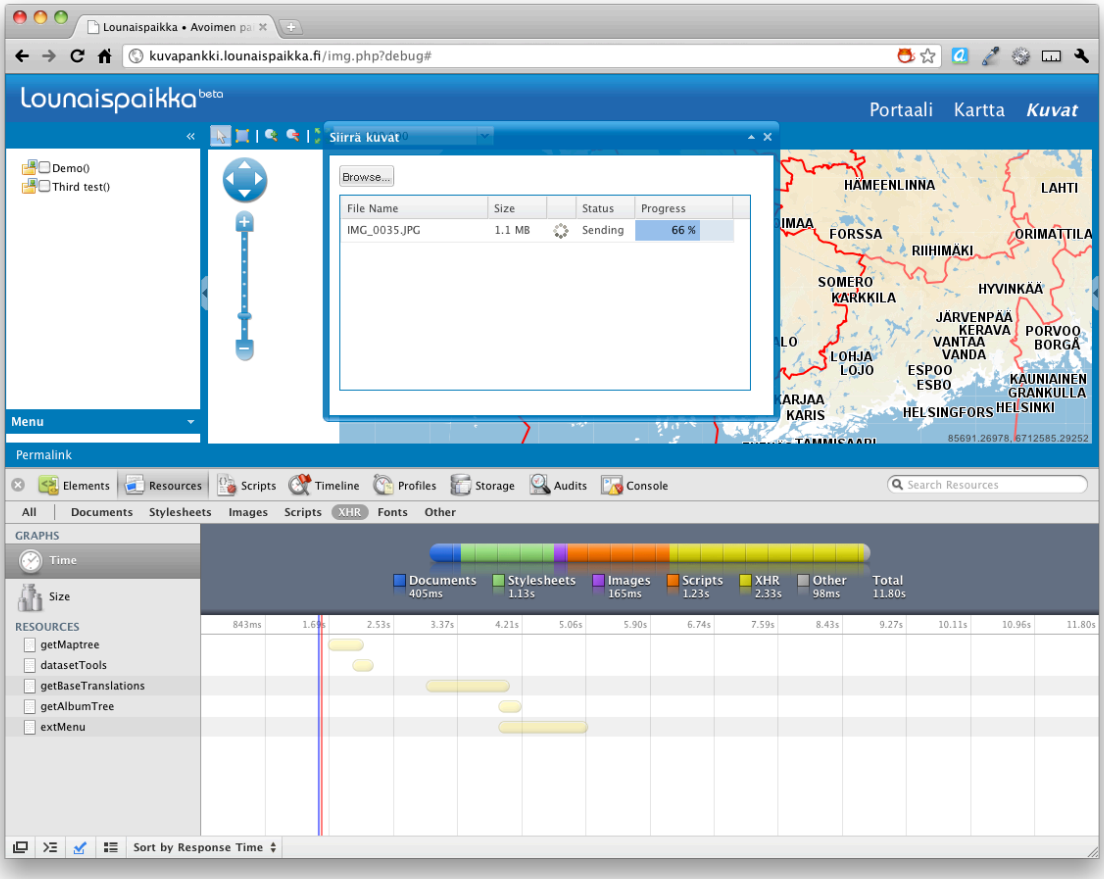
En stor del av tiden när man utvecklar går åt till felsökning, oberoende av språk, miljö och metoder. Utveckling med JavaScript och PHP är inget undantag. I ett storskaligt projekt som detta kan ett litet fel rendera timmar av felsökning om det vill sig illa.

När det kommer till felsökning inom webbutveckling är det nästan uteslutande ett verktyg som används, nämligen Firebug. Firebug är ett tillägg till den populära webbläsaren Mozilla Firefox, som erbjuder funktioner för att analysera en webbsida, vilket underlättar vid analys och felhantering. I Googles nya webbläsare Chrome, som jag använt i första hand i detta projekt, finns motsvarande verktyg som hos Firebug. Chrome och Firebug erbjuder följande funktioner direkt i webbläsaren:

- Inspektera och redigera HTML och CSS i realtid
- JavaScript debugger och konsol
- Analys av anrop till servern (AJAX)
- Analys av nätverksbelastning

Det jag haft mest användning för i detta projekt är helt klart JavaScript debuggern. Med hjälp av debuggern kan man följa med i koden när den körs och således upptäcka buggar. Detta kommer speciellt till pass när man arbetar med externa bibliotek som man inte känner till på samma sätt som sin egen kod. Med debuggern är det lätt att följa med koden in i biblioteken för att hitta fel och få en bättre förståelse för hur biblioteken fungerar. Webbläsarverktygen erbjuder även användningen av stopp i koden, sk. breakpoints, vilka kan användas för att analysera applikationens tillstånd vid ett förutbestämt läge.

En annan viktig del av dessa verktyg är analysen av AJAX-anrop. Eftersom hela tjänster bygger på AJAX, utförs det ett stort antal anrop till servern och här är det viktigt att kunna analysera såväl input- som outputdata. Chromes verktyg ger detaljerad information om anropen och t.ex. responstiden är väldigt viktigt. Figur 16 nedan visar aktuella AJAX-anrop samt när de startat och hur länge de tagit.



Figur 16. Google Chromes verktyg för felsökning.

Ett effektivt sätt att undvika eller hitta fel i en applikation är genom att använda loggar. Jag använde mig av en simpel textlog för att logga serverhändelser. Jag skapade helt enkelt en funktion som skriver till en logfil, som ligger på servern. Till loggen skrev jag allt från variabelvärden till timers för att effektivera koden. För att logga värde från JavaScript-koden använde jag de verktyg som webbläsarna erbjuder. Det fungerar så enkelt som att man t.ex. via kommandot console.log(myObject) skriver ut objektet i JavaScript-konsolen.

5.10 Utmaningar

Den kanske största utmaningen i det här projektet var att få igång utvecklingen av själva mjukvaran. Strukturen för LPII var länge oklar och eftersom detta projekt integreras i LPII föranledde oklarheterna vissa fördröjningar. Vilket JavaScript bibliotek som skulle användas var länge oklart, vilket i sin tur ledde till att jag kunde bekanta mig med det valda biblioteket betydligt senare än planerat. Som en lösning på denna fördröjning utfördes projektet nu som ett 9 månaders deltidsprojekt istället för 6 månaders heltidsprojekt, vilket var den ursprunliga planen.

Före projektet hade jag enbart erfarenhet av att utveckla Windows-applikationer samt traditionella Web 1.0-applikationer. Web 2.0 med JavaScript som det centrala utvecklingsspråket var således nytt för mig och krävde en hel del studier, men framför allt otaliga timmar av test och experiment.

6 Diskussion

Arbetet var väldigt givande för såväl mig som kunden. Trots att jag t.o.m. var aningen orolig i början av projektet när jag märkte hur omfattande och individuellt projektet var, kom jag bra igång och märkte snabbt att ämnet och tillvägagångssätten var intressanta. Någon fråga om motivationsbrist var det aldrig under projektets gång, det kändes snarare lite tråkigt, att projektet var tidsbundet och att jag inte själv hade möjligheteten att slutföra det i form av publicering som en del av LPII.

Om man ser tillbaka på projektet var det en ganska perfekt introduktion till modern webutveckling. ExtJS-biblioteket som användes, har enbart ökat i popularitet sedan projektet utfördes, och är fortfarande idag en av de största Javascript-biblioteken. Min förståelse för objektorienterad utveckling fördjupades avsevärt under projektets gång, speciellt inom PHP var objekthanteringen var en väsentlig del.

De problem jag specifierade i början av projektet, löste jag planenligt utan större problem. Uppladdningen av bilder lyckas genom ett användarvänligt gränssnitt och fungerar nära på identiskt i alla webbläsare som tjänsten stöder. Användarvänligheten har maximerats t.ex. genom att visa indikatorer varje gång tjänsten laddar för att hela tiden hålla användaren medveten om vad som händer.

Det klart mest negativa med projektet var det rent praktiska upplägget, då jag arbetade i Raseborg medan alla övriga, som var inblandade i projektet, arbetade i Åbo. Eftersom tjänsten inte var så fristående som det framgick i början, var en kontinuerlig kontakt med kollegorna i Åbo nödvändig i stort sett varje dag. Kommunikationen sköttes med Skype och mail, men jag tror att arbetet hade underlättats avsevärt ifall man fysiskt befunnit sig i samma utrymmen. Tröskeln för att ta kontakt via mail för att fråga eller kommentera någonting är trots allt betydligt högre, än om det går att sköta ansikte mot ansikte direkt när ärendet uppstår. Mot slutet av projektet införde vi veckomöten i Åbo för att säkerställa, att tjänsten följde kundens alla önskemål. Förutom att arbetet försvårades pga. den geografiska placeringen blev även den sociala biten lidande. Eftersom ingen annan från Novia Raseborg var direkt inkopplad i projektet blev min dagliga kontakt med kollegor minimal.

Den tidtabell, som sattes upp vid projektstarten höll inte, men det berodde till största delen på huvudprojektet LPII:s framskjutna tidsplan. T.ex. någon form av användarhantering, som är en grundläggande del av bildbanken, fanns inte ännu tillgänglig när projektet tog slut.

7 Avslutning

Projektet resulterade i att såväl jag som kunden var väldigt nöjda med slutprodukten. Samarbetet fungerade bra projektet igenom och trots smärre ändringar i tidsplanen kunde jag leverera en produkt som uppfyllde kundens önskemål.

Kunden uttryckte tydligt att de var nöjda med produkten och ansåg att den kunde vara klar för publicering relativt snart. Tjänsten fick speciellt beröm för användarvänligheten samt det okomplicerade gränssnittet. Nedan följer ett citat ur mötesprotokollet från LPII-projektets avslutningsmöte den 23.8.2010. Texten är översatt från orginalspråket finska till svenska av mig.

"Bildbanken fick mycket beröm och tjänsten ser väldigt lovande ut. Man kan skapa, organisera samt dela album och visa dess innehåll på kartan. Bilderna går även att kommentera. Bilder tagna med GPS-kamera får automatiskt rätt position medan övriga bilder får sin position genom att ange den på kartan. Bildindikatorerna som syns, visas beroende på vilken skala som är vald för kartan. Galleriet visar alla aktuella bilder för vald karta eller en detaljerad version av en enskild bild. Albumen vidareutvecklas ännu så att de får egna bildindikatorfärger. En sökfunktion, var man kan söka efter bilderna på ett liknande sätt som man söker material i huvudtjänsten, kommer ännu att utvecklas. Grupperna bildas organisatoriskt, dvs. baserat på användarna."

Källförteckning

Brinzarea-lamandi, B., Darie, C. & Hendrix, A. (2009). *Ajax and PHP Building Modern Web Applications - Second Edition*. Birmingham: Packt Publishing.

Extending Ext Class (u.å)
http://www.sencha.com/learn/legacy/Tutorial:Extending_Ext_Class (hämtat: 2.2.2011)

Garcia J. (2009). *Ext JS in Action*. (u.o) Manning Publications.

Mapserver (2010) http://mapserver.org/ (hämtat: 17.12.2010)

OpenLayers Documentation http://docs.openlayers.org/ (hämtat: 18.12.2010)

Padron-McCarthy T. (2005) Databasteknik. Lund: Studentlitteratur

Ramon, J. (2009). *Ext JS 3.0 Cookbook*. Birmingham: Packt Publishing

Ramsay, C., Frederick, S., Blades, S. (2008). *Learning Ext JS*. Birmingham: Packt Publishing

Sencha Docs (2010)
http://docs.sencha.com/ext-js/3-4/ (hämtat: 17.12.2010)

Zandstra, M. (2008) *PHP Objects, Patterns, and Practice Second Edition*. Berkeley: Apress

Bilaga 1. Kravspecifikation framställd av Lounaispaikka före projektets start

Kuvapankin vaatimusmäärittely:
Kuvien lataustyökalu toimii Lounaispaikan uudistuvan karttapalvelun yhteydessä osoitteessa www.lounaispaikka.fi. Työkalu integroidaan osaksi Lounaispaikan uudistuksessa kehitettyä frameworkia ja sen pitää olla käyttöliittymältään helppo ja soveltuva Lounaispaikan uuteen ulkoasuun.

Kuvien lataus:
Käyttäjä voi ladata yksittäisen kuvan tai useamman kuvan kerralla Lounaispaikan palvelimelle. Kuvien latauksen yhteydessä tehdään seuraavia toimenpiteitä:

- kuvien lataaja kirjautuu/rekisteröityy palveluun (Lounaispaikan käyttäjänhallinnan kautta)
- kuvien latauksen yhteydessä esitellään kuvien luovutussopimusteksti (vrt. facebook)
- kuvat viedään Lounaispaikan palvelimella lataustyökalun avulla, joka on yhteensopiva kaikkien selaimien kanssa (esim. swfupload)
    - kuvat tallennetaan palvelimen levylle selkeästi organisoituun hakemistoon
    - samalla niistä luodaan verkkopalveluun soveltuvat esikatselukuvat
    - latauksen yhteydessä kuville luodaan kuvakansio, joka sisältää seuraavia tietoja tietokannassa:
        - otsikko
        - kuvaus
        - kansion näkyvyys (käyttäjänhallinnan avulla) è määritellyt käyttäjät tai ryhmät voivat myös lisätä kuvia kansioon
        - alueellinen paikkatieto (esim. koordinaatti, kunnan nimi, seutukunnan nimi)
        - kommentit (kansiota voi kommentoida)
    - ladattavista kuvista tallennetaan seuraavia tietoja tietokantaan:
        - kuvateksti
        - päivämäärä & kuvausaika
        - paikkatieto
            - valmiin koordinaattitiedon perusteella (GPS-kamerat)
            - osoitteen tai paikan nimen perusteella (nimi/osoitetietokantaan yhteys) è pitää ratkaista vielä mittakaava-asia (esim. Turku: paikannimitietokannan perusteella kaikki Turkuun liitettävät kuvat tulevat Tuomiokirkkosillan kohdalle)
            - manuaalisesti kartalla klikkaamalla (kartalta tallentuu koordinaatti ja mittakaava, jolla kartalla on klikattu)
        - kameran tiedot yms. muut kuvaan liittyvät metatiedot (esim. exif, ipct ja xmp metatiedot säilytetään myös)
        - kuvan ottajan tiedot (huom! kuvan lataaja voi olla eri henkilö kuin sen ottaja), nimi näkyviin ulkopuolisille
        - avainsanat (mahdollistetaan useiden avainsanojen lisäys)
        - LP2 kategoriat (näin kuvat saadaan esiteltyä paikkatietoaineistojen kanssa kartalla)
        - kommentit (kuvia voi kommentoida)

Kuvatietojen muokkaus
Kuvien hallitsijan pitää voida muokata nopeasti ja helposti kuvien sisältämiä tietoja. Tätä varten luodaan kuvatietojen muokkausliittymä, jossa:

- kuvien metatietoja voi muokata yksitellen tai
- usean kuvan metatietoja voi muokata yhtäaikaisesti è esim. kaikille valituille kuville sama kuvanottaja tai avainsana tai kuvanottoajan muokkaus (esim. +1 tunti)
- kansion kuvien järjestyksen muuttaminen raahaamalla tai automaattisesti, esim. kuvan nimen tai ottohetken mukaan
- kuvien vieminen kansiosta toiseen

Kuvien esittäminen

- Hakutyökalu tulee sijaitsemaan Lounaispaikan aineistohaun yhteydessä (avainsanahaku, kategoriahaku jne.)
- Kuvat voidaan esittää kartalla ja kuvagalleriana
    - yhtä aikaa kartalla
    - yhtä aikaa erillisessä kuvagalleriassa
    - karttatasoina kategorian mukaan kartalla (samoin kuin paikkatietoaineistot)
    - kuvat voidaan esittää avainsanan tai muun hakutuloksen perustella kartalla/kuvagalleriassa
    - ajallisesti kartalla/kuvagalleriassa (kuvia selattaessa kartalle muodostuu kuvien ottajan ”reitti”, josta voidaan seurata ajallista kulkua maastossa)
- Kuvagalleriassa kuvien selailu pitää olla jouhevaa
  Kuvat esitellään kartalla kuvakkeina, joista hiiren ollessa kohteen yläpuolella tulee esiin ensin pikkukuva ja kohdetta klikkaamalla avautuu koko sisältötieto ja suurempi kuva

Bilaga 2. ExtJS klassen för fönster hantering

```
/**
 *
 * LouGIS Images ExtJS Library
 *
 * @namespace Lougis.Img
 * @class Lougis.Img.Window
 * @extends Ext.Window
 * @author Filip Holmberg <filip.holmberg@lounaispaikka.fi>
 *
 */
Lougis.Img.Window = Ext.extend(Ext.Window, {

	constructor: function(config) {

		this.cmp = config.cmp;
		var wc = config.winConfig || {};

		if(Ext.getCmp('win_'+this.cmp.id)) {
			Ext.getCmp('win_'+this.cmp.id).close();
		}

		this.id = 'win_'+this.cmp.id;
		this.autoHeight = wc.autoHeight || false;
		this.closable = wc.closable || true;
		this.collapsible = wc.collapsible || true;
		this.minimizable = false;
		this.maximizable = wc.maximizable || false;
		this.resizable = wc.resizable || false;
		this.shadow = wc.shadow || false;
		this.autoScroll = wc.autoScroll || true;
		this.modal = wc.modal || false;
		this.animateTarget = wc.animateTarget || 'lgSouth';
		this.autoDestroy = true;
		this.expandOnShow = true;
		this.manager = imgWins;

		this.scrollPadding = false;
		this.autoSize = wc.autoSize || true;

		this.contentPanel = {
			xtype: 'panel',
			id: 'win_content_'+this.id,
			autoWidth: true,
			autoHeight: true
		};

		/**
		 * Drawers sliding out of the parent window
		 *
		 * @example Ext.fly(winId).drawers.w.show();
		 */
		this.drawers = {
			w:	new Ext.ux.plugins.WindowDrawer({
					xtype: 'windowdrawer',
					id: 'drawerW_'+this.cmp.id,
					side: 'w',
					animate: true,
					resizable: true,
					autoHeight: true,
					width: 250
				}),
			n:	new Ext.ux.plugins.WindowDrawer({
					xtype: 'windowdrawer',
					id: 'drawerN_'+this.cmp.id,
					side: 'n',
					animate: true,
```

```
                        resizable: true,
                        autoHeight: true,
                        width: 400
                    }),
            e:      new Ext.ux.plugins.WindowDrawer({
                        xtype: 'windowdrawer',
                        id: 'drawerE_'+this.cmp.id,
                        side: 'e',
                        animate: true,
                        resizable: true,
                        autoHeight: true,
                        width: 250
                    }),
            s:      new Ext.ux.plugins.WindowDrawer({
                        xtype: 'windowdrawer',
                        id: 'drawerS_'+this.cmp.id,
                        side: 's',
                        animate: true,
                        resizable: true,
                        autoHeight: true,
                        width: 400
                    })
};
this.plugins = [        this.drawers.e,
                        this.drawers.s,
                        this.drawers.n,
                        this.drawers.w];

this.listeners = {
        'afterrender' : {
                fn: function(cmp) {
                        this.calculateSize();
                },
                scope: this
        },
        'beforecollapse': {
                fn: function(cmp) {
                        this.hide();
                        this.hideDrawers();
                },
                scope: this
        },
        'collapse' : {
                fn: function(cmp) {
                        this.hide();
                        this.dock();
                },
                scope: this
        },
        'beforeexpand': {
                fn: function(cmp) {
                        this.hide();

                        // throw the window beneath visible area to
                        // hide the expand action
                        this.setPosition(0,this.getPosition()[1]+100);
                },
                scope: this
        },
        'expand' : {
                fn: function(cmp) {
                        this.toFront();
                        this.calculateSize();
                        this.show();
                },
                scope: this
        }
};
```

```
        Lougis.Img.Window.superclass.constructor.call(this, config);

        if(this.cmp.title) {
                this.setTitle(this.cmp.title);
                this.cmp.title = '';
        }

        if(config.title) {
                this.setTitle(config.title);
        }

        if(this.cmp) {
                this.add(this.cmp);
        }

},

/**
 * Hide all drawers
 * @example when window is collapsed
 */
hideDrawers: function() {

        this.drawers.w.hide();
        this.drawers.n.hide();
        this.drawers.e.hide();
        this.drawers.s.hide();

},

/**
 * Docks window to the mappanel bottom
 */
dock: function() {

        var dockEl = Ext.get(this.id); // element

        // position of "dock" component
        var destPos = Ext.getCmp('mappanel').getPosition();
        // comp size
        var destSize = Ext.getCmp('mappanel').getSize();
        var marginX = 10; // margin from left edge
        var marginY = destSize.height; // margin from top edge
        var winX = 0;
        var collapsedWins = 0;

        // loops imgWins window collection
        imgWins.each(function(win) {

                // exclude autogenerated windows(alert etc.)
                if(win.id.substr(0,4) != 'ext-') {
                        if(win.collapsed && win.id !== this.id) {
                                if(winX < win.getPosition()[0]) {
                                        winX = win.getPosition()[0]+win.getWidth();
                                }
                                collapsedWins++;
                        }
                }

        }, this);

        if(collapsedWins === 0) {
                winX = destPos[0]+10;
        }

        this.setPagePosition(marginX+winX,
                destPos[1]+marginY-this.getHeight());
```

```
            dockEl.slideIn('b', {
               easing: 'easeIn',
               duration: .3
            });
            dockEl.frame();

},

/**
 * Recalculates position of "docked" tabs
 */
realignDockPosition: function() {

            var dockEl = Ext.get(this.id); // element

            // position of "dock" component
            var destPos = Ext.getCmp('mappanel').getPosition();
            // comp size
            var destSize = Ext.getCmp('mappanel').getSize();

            // margin from top edge
            var marginY = destSize.height;

            this.setPagePosition(this.getPosition()[0],
                        destPos[1]+marginY-this.getHeight());

},

/**
 * Calculates proper window size and resizes
 */
calculateSize: function() {

            var margin = 10;
            var cSize = this.calculateContentSize();
            var width = cSize.width;
            var height = cSize.height;

            if( (lgView.getHeight()-margin) < height) {

                        // set height based of resolution
                        this.setHeight(lgView.getHeight()-margin);

            } else {

                        // set height based on content
                        this.setHeight(height);

            }
            if( (lgView.getWidth()-margin <= this.getWidth()) ) {

                        // set width based on resolution
                        this.setWidth(lgView.getWidth()-margin);

            } else {

                        if(this.getHeight() < height) {

                                    // set width based on content and
                                    // add scrollbar padding
                                    if(!this.scrollPadding) {
                                                this.setAutoScroll(true);
                                                this.setWidth(width+Ext.getScrollBarWidth());
                                                this.scrollPadding = true;
                                    }

                        }
```

```
                else {

                        // set width based on content
                        this.setAutoScroll(false);
                        this.setWidth(width);
                        this.scrollPadding = false;

                }

        }

        this.doLayout();
        this.center();

    },

    /**
     * Calculate total width & height of window content(items)
     */
    calculateContentSize: function() {

        var width = this.getFrameWidth();
        var height = this.getFrameHeight();

        Ext.each(this.items.items, function(item, index) {

                if(width < item.getWidth()) {
                        width = this.getFrameWidth()+item.getWidth();
                }
                height += item.getHeight();

        }, this);

        return {
                width: width,
                height: height
        };

    },

    /**
     * Creates a new identical window thru config cloning
     */
    cloneWin: function() {

        var tmp = this.cloneConfig();
        this.close();
        new Lougis.Img.Window(tmp).show();

    }

});
```

Bilaga 3. Funktionen getGalleryImages() från Image databasklassen som returnerar alla bilder för aktuell gallerivy.

```
/**
    * get images for gallery view
    *
    * @param $params array
    * @return array
    * @author Filip Holmberg <filip@lounaispaikka.fi>
    */
   public static function getGalleryImages($params) {

            $result = array();

            $albumId = $params['albumId'];
                    $albumId = explode(',', $params['albumId']);
            $limit = $params['limit'];
            $start = $params['start'];

            $Img = new Lougis_img_image();
            $Img->selectAdd('ST_AsGeoJSON(location) as loc');
            $i = 0;
            $c = count($albumId);

            // loops album_id array
            while ($i < $c) {
                    $i == 0 ?
                            $Img->whereAdd("album_id = $albumId[$i]") :
                            $Img->whereAdd("album_id = $albumId[$i]", "OR");

                    $result['debug'][$i] = $albumId[$i];
                    $i++;
            }
            $Img->limit($start, $limit);
            $Img->find();

            $rootNode = 'images';
            $i=0;
            while( $Img->fetch() ) {

                    $path = Lougis_img_image::getPathByImageId($Img->id);
                    if(!$path['success']) throw new Exception($path['error']);

                    $result[$rootNode][$i]['id'] = $Img->id;
                    $result[$rootNode][$i]['filename'] = $Img->filename;
                    $result[$rootNode][$i]['header'] = $Img->header;
                    $result[$rootNode][$i]['scale'] = $Img->scale;
                    $result[$rootNode][$i]['description'] = $Img->description;
                    $result[$rootNode][$i]['username'] = Lougis_user::getUserNameByUserId($Img->user_id);
                    $result[$rootNode][$i]['path'] = $path['path'];

                    $loc = json_decode($Img->loc, true);
                            $lon = $loc['coordinates'][0];
                            $lat = $loc['coordinates'][1];
                            $result[$rootNode][$i]['lon'] = $lon;
                            $result[$rootNode][$i]['lat'] = $lat;
                            $result[$rootNode][$i]['exact'] = $Img->exact_loc;

                    $i++;
            }

            $result['imageCount'] = Lougis_img_image::getImageCount($params);

            return $result;

}
```