Crystallography Service Sample Database Administrator's Guide

J.P.Hagon
Computer Systems Support
School of Chemistry

April 7, 2012

Introduction

This guide is intended for staff who administer the Newcastle University Crystallography Service Sample Database. It includes both a general description of the web interface and associated administration procedures along with a more technical description of the software interface and the database itself so that administrators can recover from situations such as a forgotton administrator password.

Contents

1 General Description of the System

1.1 Introduction

The system consists of a *front-end* which is used by users and administrators to submit sample requests and upload analysis data. This front-end is implemented using the ruby[1] programming language and version 3 of *Ruby on Rails*.[2] The front-end is hosted on an *Apache*[3] server running on an *Ubuntu*[4] linux system. Technical details will be described elsewhere.

The *back-end* consists of a set of ruby programming libraries and a SQL database — in this case *SQLite3*.[5] More technical aspects of the back-end will be described elsewhere.

1.2 Users

The system has a relatively simple user setup with just one basic user type. However, there are three levels of authority that a user can have:

Standard Most users of the system will have a standard account which allows them to submit sample requests and view their own sample data.

Group Leader These users have the additional privilege of being able to see all of the sample data for their own group in addition to their own samples. A group of users may have more than one designated group leader.

Administrator An administrator, in addition to standard user privileges, can do many administration tasks. These include adding/deleting users, changing user privileges, submitting/updating/deleting samples, editing public web pages on the server and uploading files to the server.

Users can either self-register or be added by an administrator. An administrator can also disable a user account without actually deleting it. Only the most basic information about a user is stored in the database, namely first name, last name and email address. the email address serves as a login id. The user can set his own password. If the user forgets his password, the system can email him a secure link to the server via which the password can be reset.

1.3 Groups

All users must be associated with a group. Typically this will be a research group associated with a particular person. When a user self-registers, he must select an appropriate group. If such a group does not exist, an administrator must set one up for him. Usually one or more users will be designated *group leaders* and will have access to information about all the group's samples.

1.4 Samples

The primary purpose of the system is to track and keep a record of samples submitted to the crystallography service. A typical workflow is shown in Figure 1. Emails are sent automatically by the system when a sample status is updated by an administrator.

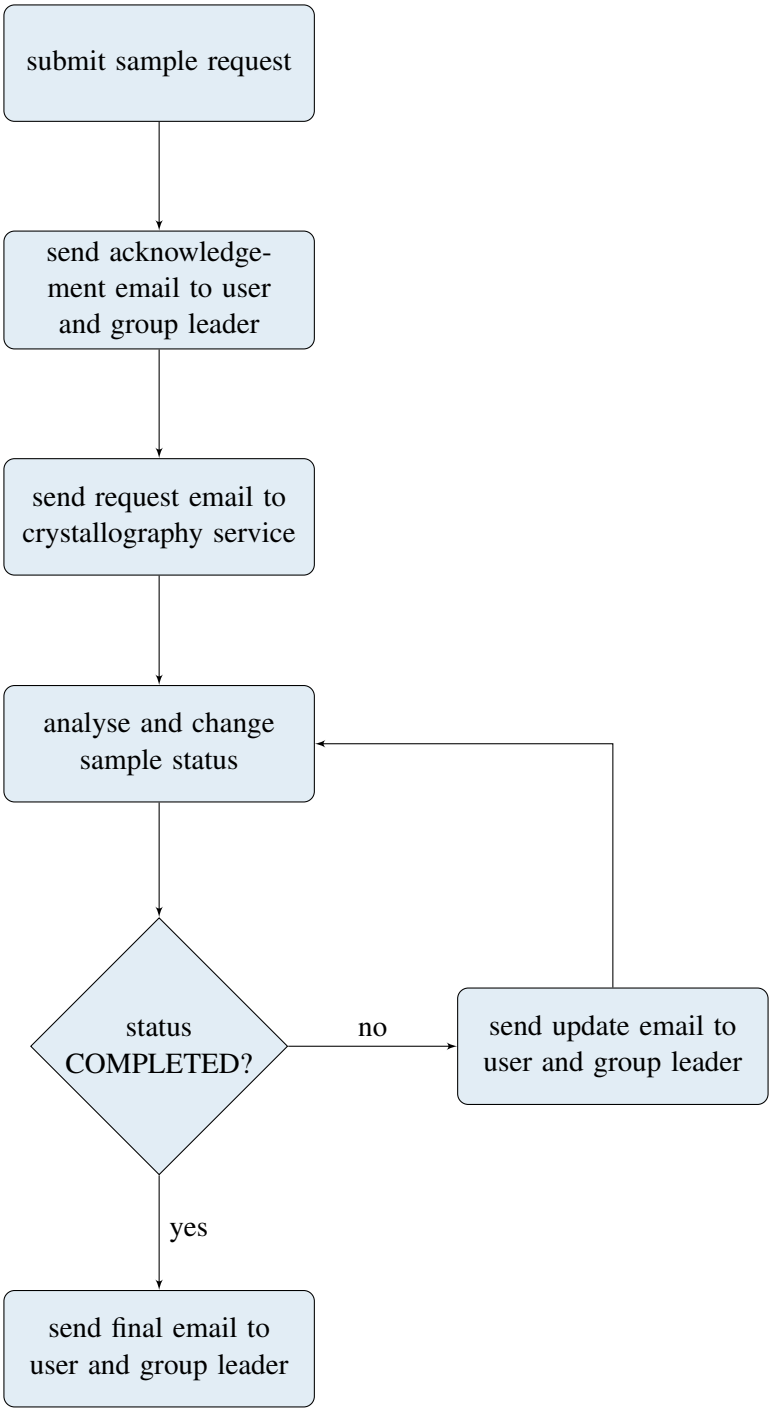


Figure 1: Typical workflow for sample processing cycle. Emails are sent automatically by the system whenever the status of a sample is updated.

1.5 Public Pages

Most information on the server can be viewed only by registered users. However, there are some pages which are more generally accessible. Such pages include the home page, general information pages and the sample queue. Public pages can be created and edited by an administrator using tools provided by the server software. rather than write pure HTML, an administrator can use a text-based markup language called *Textile*[6] which can produce sophisticated web pages with all the usual constructs such as headings, paragraphs, floating elements, tables and images.

2 Web Management Guide

2.1 Introduction

In this section we describe the web management interface to the sample tracking database. When a manager is logged-in, the home page of the system looks similar to that shown in Figure 2

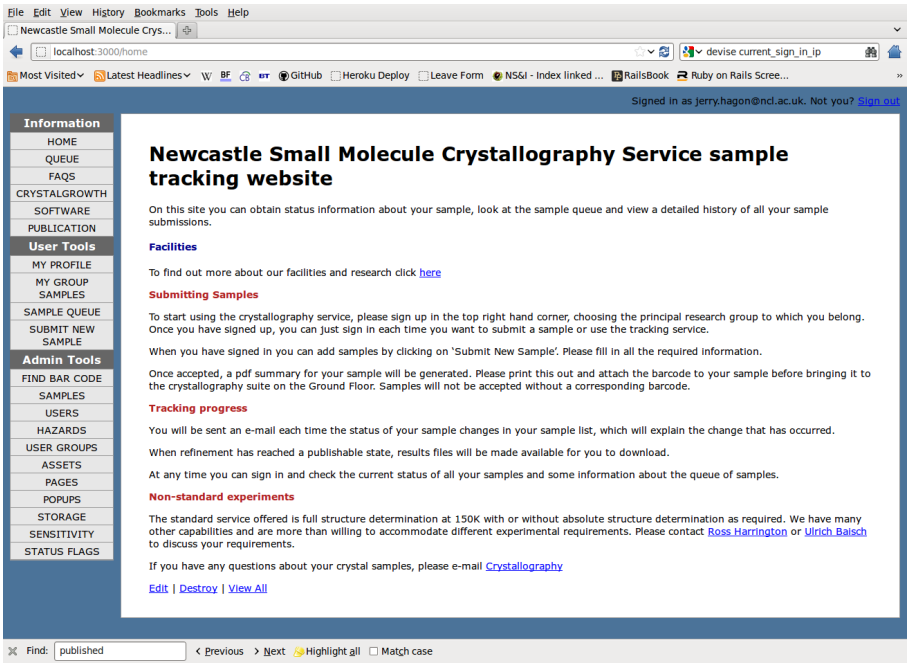


Figure 2: Administrator's view of home page.

There are three parts to this browser view:

(i) a main display showing the contents of a page of information;

(ii) a menu on the left side of the browser window;

(iii) login information and a `sign_out` link above the main display on the right.

The left side menu consists of three sections:

Information These links point to *static* pages which can be created by an administrator. the administrator can also add extra links to the information section. We describe how to do this in §2.2.

User Tools These tools allow a user to view his sample list, submit a new sample and view his profile information. Additionally, if a user is also a group leader, he will have access to the *My Group Samples* link for listing all samples in the user's group.

Admin Tools This is the main set of web-based tools for administrators. We will describe each of these in the next section.

2.2 Adding Static Pages

Clicking the *PAGES* link in the *Admin Tools* sub-menu produces the pages index shown in Figure 4. This shows a list of pages. For each of these pages is a set of buttons allowing the administrator to show, edit or delete the page as shown in Figure 3. These buttons are used throughout the database editing pages on the web server. At the bottom of the list is a link to create a new page.

Figure 3: The *show*, *edit* and *delete* buttons.

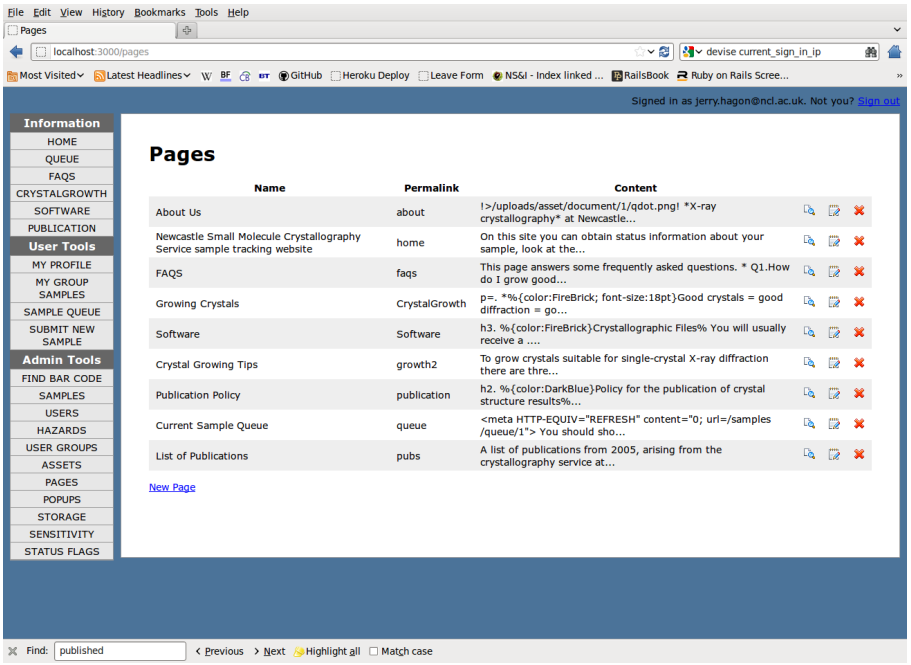


Figure 4: The pages index view.

Figure 5 shows the page editor — a very simple form for entering/changing text. This example shows the home page data. the content is entered in a markup language called Textile*. You can also specify the name of the page (this will be used to set the HTML title attribute and a permalink†. At the bottom are links to the Textile Reference Manual, the page view and the pages index. the page can be referred to via the URL:

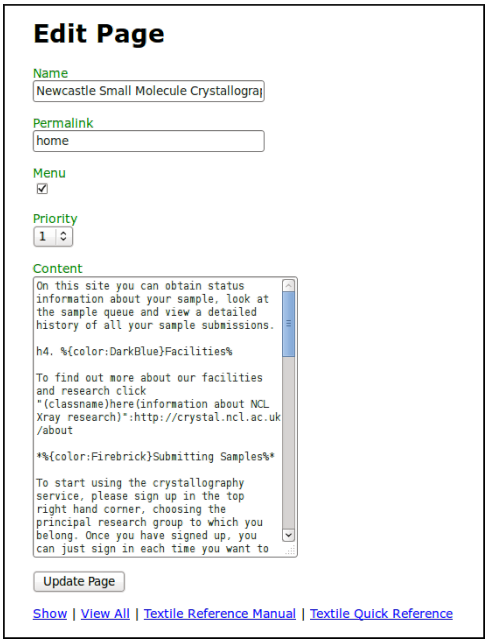


Figure 5: The page edit view.

```
<server name>/permalink
```

thus making static page addressing very simple. An alternative URL which can be generally used for any page is:

```
<server name>/pages/<id>
```

but the permalink-based URL is what you'd almost always use in practice. If, for some reason, you want to use the id-based URL, but don't know what the id is, then just click the 'show' icon in the pages index for the page you're interested in and look at the URL in the web browser window.

Note that there is also a checkbox labelled *Menu*. If this is checked then the page is added to the left hand *Information* menu of static pages. The *Priority* parameter is an integer that determines the order of the page in the menu. If two pages have the same priority their order is determined alphabetically. A good practice is to initially assign priorities in units of, say, 10. Subsequently, if a new page is created, there are then plenty of 'spare' priority numbers which can be used to add menu links in between existing links — otherwise existing priority numbers may need to be tweaked.

2.3 Uploading General Files to the Server

You can upload arbitrary files to the server. These files are referred to as 'assets' and can be uploaded via the *ASSETS* link in the *Admin Tools* menu. Clicking on this link will take you to the assets index page which looks very similar to the pages index described in the previous section. each index entry tells you the pathname of the file on the server, together with a description of what the file contains. Often these files will be images or documents (e.g. PDF files) that you want to link to on one of the static web pages created as described in the previous section.

At the bottom of the asset index list is a link to create a new asset. Clicking this takes you to a simple menu where you can browse for a file to upload to the server. Clicking the *Create Asset* button will then upload the file to the server. It will then be listed in the asset index with an entry under the *Document* column pointing to its location in the file system. This location has the general form:

```
/uploads/asset/document/<id>/<filename>
```

*You are allowed to mix HTML and Textile together.

†A tag which is used as a basis for a concise URL.

Here, `<filename>` is the actual name of the uploaded file as it was when it was uploaded. `<id>` is the database id the document has in the `assets` table described in detail in §4. The actual URL of the document is then:

`<servername>/uploads/asset/document/<id>/<filename>`

2.4 Creating Users and Groups

A user must belong to a group, so it is advisable to create a group for a user before the user is created. Creating a user group is straightforward via the *USER GROUPS* link in the *Admin Tools* menu. As usual, this will take you to an index of existing groups, with a link to create a new group at the bottom. Creating a new group merely requires that you enter two fields:

(i) a 3-letter group abbreviation (it *must* be three letters);

(ii) a longer group description.

Users can be created in two ways; they can self-register by clicking on the link top-right on the home page or they can be created by an administrator. In either case the form used to create and register a new user is the same.

2.5 Sample Management

In this section we give a complete description of the sample management workflow.

(i) First, a user fills out a sample submission form online. This form is shown in Figure 6. This figure shows the submission form from the point of view of both a non-administrative user and an administrator.

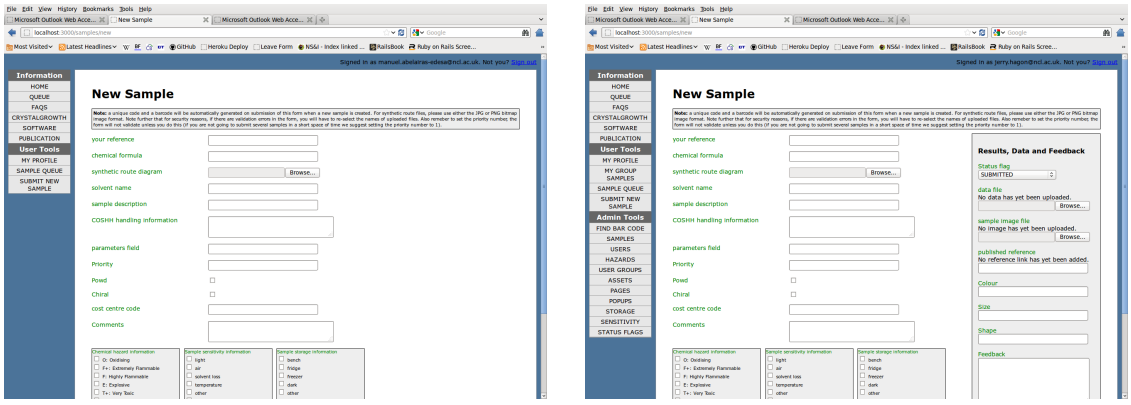


Figure 6: User's view (left) and administrator's view (right) of the sample submission form.

The bits that an ordinary user doesn't see are those parts of the sample fields that are subsequently filled in by an administrator in the course of sample processing.

There is a lot of validation built into the form making it very unlikely that a form will be submitted with incomplete information. Table **??** gives a complete list of validation checks. If a validation check fails on submission of the form, the user will be presented

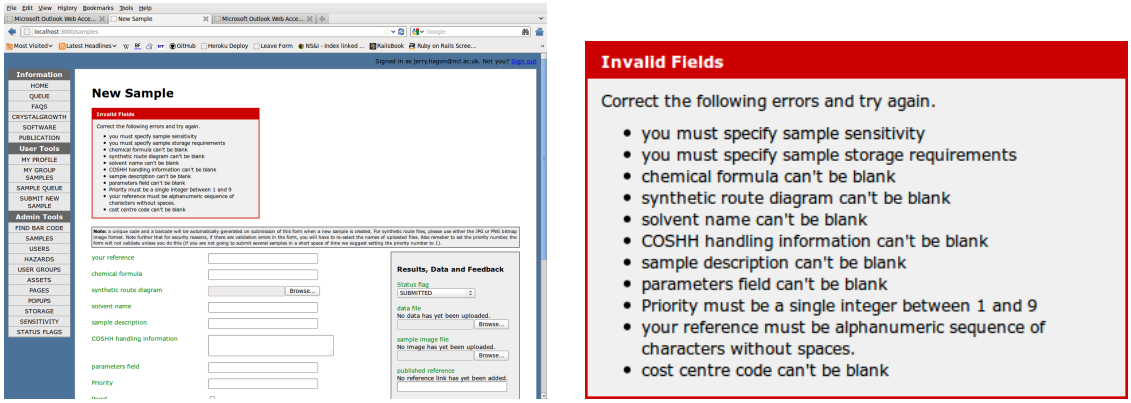


Figure 7: Illustrating what happens when an invalid form is submitted. The user sees an *Invalid Fields* box (shown enlarged on the right). In this case, an administrative user has failed to fill in any of the fields — hopefully a very rare occurrence!

with the *Invalid Fields* box which will tell him which fields have not been entered correctly and what is required. An extreme example of this is shown in Figure 7 which shows what happens when none of the fields in the form are filled-in.

(ii) On successful submission of the sample form, the user (and group leaders if the user is not a group leader himself*) will receive a confirmation email containing a link to the *Sample Receipt* — a PDF file containing the user input information as well as a unique code identifying the user, user group and sample and a unique barcode. The email has the following general form with the tags in angle brackets filled-in automatically:

```
Dear User

New Sample Submission Code: <SAMPLE CODE> (your ref <SAMPLE USERREF>)
Submitted By: <USER FULL NAME>


your sample analysis request has been received. Please download a
receipt using the link below. Please quote the sample code in any
correspondence.

There is a tear-off slip at the bottom
of the receipt which you should attach to your sample.
You will be informed via email of any
changes in the status of your sample.

<LINK TO SAMPLE RECEIPT>

Copies of this email are sent to both sample submitters and their
research group leaders (where different).

Newcastle Crystallography Service
```

A tear-off slip at the bottom of the sample receipt containing the barcode, sample code

*If there is more than one group leader then the other group leaders will also receive an email.

and provided user reference as well as essential COSHH information can be attached to the actual sample itself. Figure 8 shows a typical sample receipt.

The receipt is generated on-the-fly from the supplied sample information. To perform the generation of the PDF, the well-known Prawn ruby library is used. Once the sample has been submitted, a user can regenerate the sample receipt at any time via a PDF link button in the sample list on his profile page shown in Figure 9.

(iii) Next, the user brings his sample (with attached slip) for analysis and at this point he should also see it in the sample queue.

(iv) Initially the sample will have a status of `SUBMITTED`. At various points in the analysis, this status will be changed by crystallography staff. Whenever the status is changed, an email is sent to both the user and group leaders informing them of the change. The text of the email looks similar to this:

```
Dear User

This is to inform you that the status of sample
<SAMPLE CODE> (your ref <SAMPLE USERREF>) submitted by
<USER FULL NAME>
has changed as follows:

New Status: <NEW STATUS FLAG> <NEW STATUS FLAG DESCRIPTION>

Old Status: <OLD STATUS FLAG> <OLD STATUS FLAG DESCRIPTION>

<LINK TO FULL SAMPLE INFORMATION>

Copies of this email are sent to both sample submitters and their
research group leaders (where different).

Newcastle Crystallography Service
```

(v) When crystallography staff have uploaded all results files (docx, res and sample image) to the server and added any additional text feedback for the sample, they will flag the sample as `COMPLETED` and the process will end. The sample data will continue to be available to users (and administrators) indefinately after that.

Results files (docx and res) will usually be made available in a single zip file, with the sample image in a separate file. The files are uploaded by administrators using the sample edit form shown in Figure 10. Note that only administrators have access to the sample edit form. Users will see the information in a slightly different way, not as a form but as a standard 'show' page. This view is shown in Figure 10.Note that if administrators do not fill in the feedback section, then a default message 'No feedback given.' is seen on the relevant part of the sample show page. This is also illustrated in Figure 11.

2.6 Sample Search and Display Tools

It is important that both administrators, group leaders and users can find information about a particular sample or group of samples. In this section we describe the tools that are available to quickly find the sample data you need.

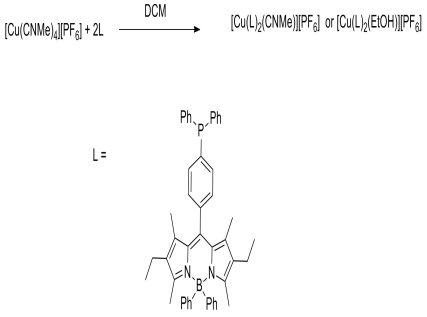
Newcastle Crystallography Service

Bedson Building ✦ Newcastle University ✦ NE1 7RU

Sample Code: LJH-LD-12-0002 **Your Ref: LD261**

Please check the details on this receipt. Changes can be made only by Crystallography staff. Please use the tear-off slip at the bottom of the page and attach it to your sample. You will be automatically informed via e-mail of any changes to your sample status.

Proposed Structure and Synthetic Route



Sample Details and Requirements

Chemical Formula: C96H95B2CuF6N5P3 OR C96H98B2CuF6N4OP3
Powder Diffraction Required? No
Chiral Structure? No
Your Priority Number: 2

User Details

Submission Date: 2012-03-02 15:29:54 UTC
Submitted By: Laura Davies
Research Group: Lee Higham Research Group
Contact E-Mail: l.h.davies@ncl.ac.uk
Cost Centre Code: n/a
Assigned Bar Code: FQYSY1J7L15

Supplied COSHH Information

Name of Solvent: Ethanol/Pentane
Description of Sample: Copper Bodipy phosphine complex
Handling Procedures: Flammable, Irritant
User Comments: Structure should be similar to LD260. As the crystals were grown in ethanol, the bound MeCN ligand may exchange with the ethanol.

Hazards:	Storage:	Sensitivity:
Highly Flammable (F) Harmful (Xn) Irritant (Xi)	bench dark	light

LJH-LD-12-0002(LD261)

FQYSY1J7L15

Supplied COSHH Information

Name of Solvent: Ethanol/Pentane
Description of Sample: Copper Bodipy phosphine complex
Handling Procedures: Flammable, Irritant
User Comments: Structure should be similar to LD260. As the crystals were grown in ethanol, the bound MeCN ligand may exchange with the ethanol.

Hazards:	Storage:	Sensitivity:
Highly Flammable (F) Harmful (Xn) Irritant (Xi)	bench dark	light

School of Chemistry Crystallography Service, Newcastle University

Figure 8: Example of a sample receipt (the actual size is A4). Note the tear-off slip at the bottom. The receipt is generated on-the-fly from the supplied sample information. The green border is not part of the PDF rendering, it is used here merely to show the A4 page border.

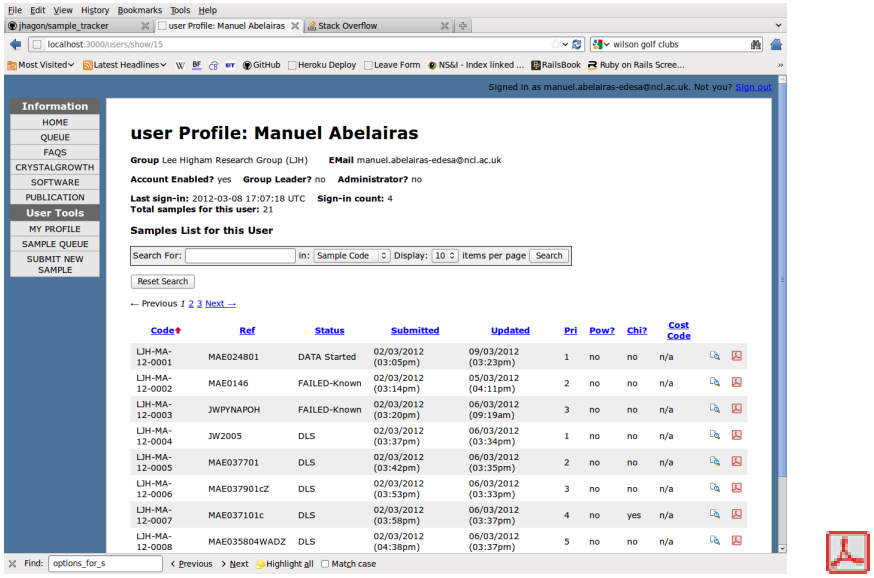


Figure 9: User profile page showing sample list with PDF icon (shown enlarged bottom right).

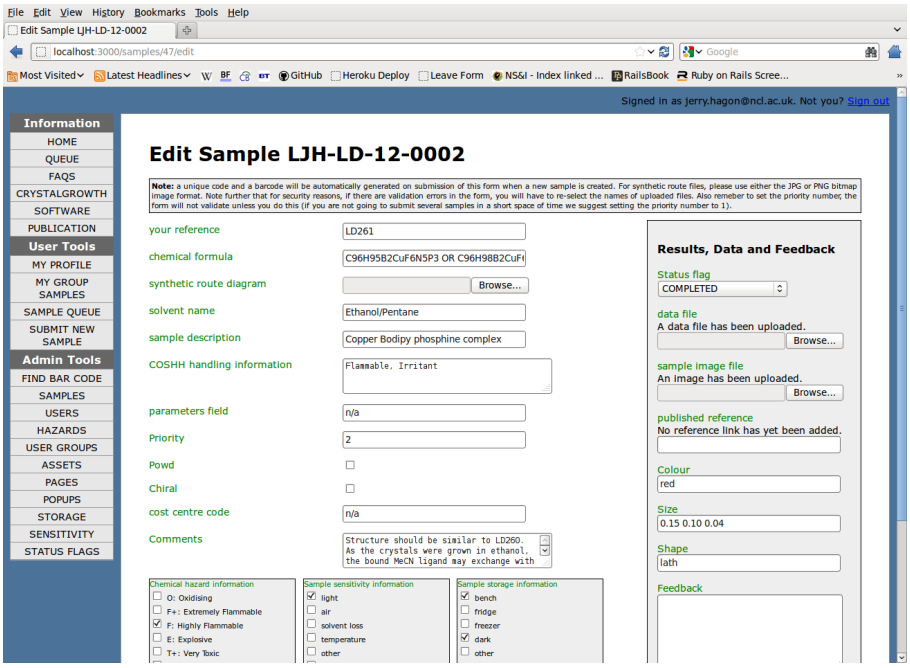


Figure 10: A fully complete sample edit form.

For administrators, the usual starting point will be the main sample index page shown in Figure 12. By default, this index is sorted by sample code in *ascending* order. This is indicated by a small red arrow pointing upwards next to the header text in the *Sample Code* column. Clicking the header text of the *Sample Code* column will reverse the order — i.e. it will now be *descending* order. This is indicated by a blue arrow pointing downwards. The list can be sorted on any other of the displayed columns simply by clicking the column header. repeated clicking on the same column header will toggle the sort order between ascending/descending.

The number of samples displayed per page can also be controlled by the user. By default,

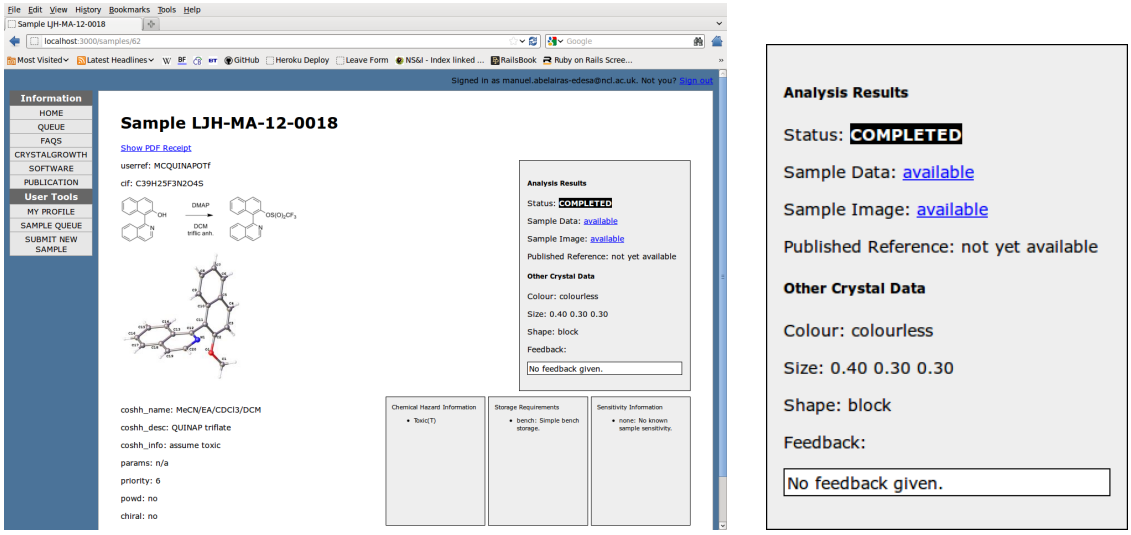


Figure 11: The user's view of a completed sample (left). On the right is an enlarged view of the results section. Note also that a thumbnail image of the sample is displayed. This thumbnail, when clicked, will show the full-size image which can then be downloaded if desired.

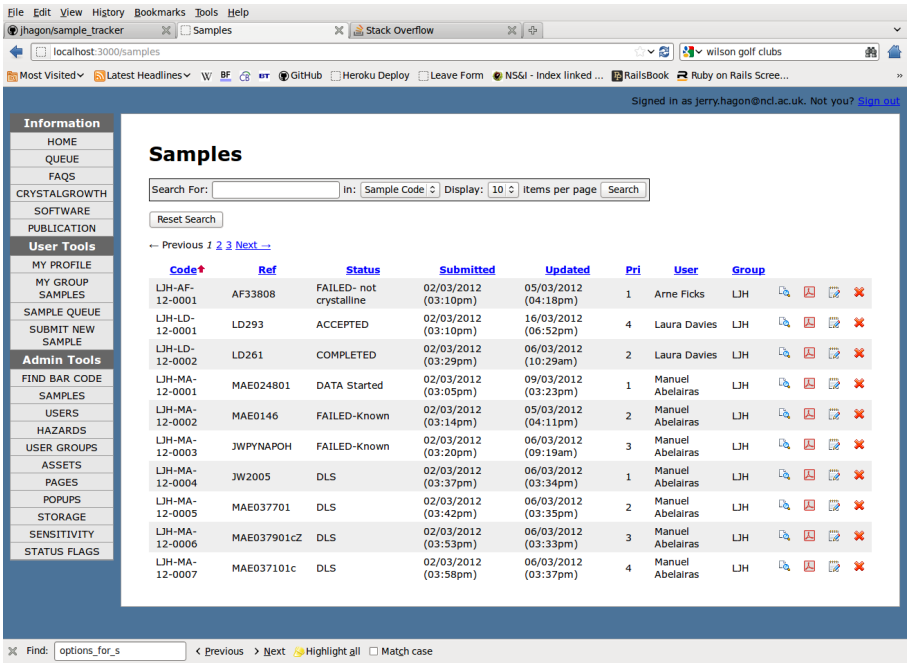


Figure 12: An administrator's view of the full sample index. In this case the red arrow next to the header in the *Code* column indicates that the list has been sorted by sample code in ascending order (the default sorting).

this number is set by a global variable, `ITEMS_PER_PAGE` defined in the `config/environment.rb` file*. However, it can be easily changed by selecting the desired value from a drop-down list in the search form above the sample list. The first entry in this list is always the value in the `ITEMS_PER_PAGE` variable. After the choice is made, the list will be re-paginated according

*See the *System Management* section for further details.

to the selected value. Note that for a full samples listing, the search box itself must be empty when you do this.

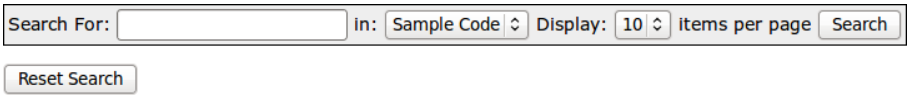


Figure 13: The sample index search form. This is usually found above the list of samples in most of the sample index pages.

To narrow down the list of samples you need to type something into the search box in the search form above the sample list. This search form is common to most of the sample listing pages and is shown in Figure 13. Currently You can search on three fields: the sample code, the user reference or the status. When you perform a search, the results will be paginated according to the setting of the pagination parameter described above. Note that search results can be sorted as before by clicking the header text of the column that you wish to sort by. Below the search form is a reset button which resets all the search parameters (but not the pagination) to their default values. You can also use the standard SQL % symbol as a 'wildcard' character.

Note that searches are *not* cumulative — they are always made with respect to the full set of samples. In other words if you perform a second search after an initial search, the results of the second search will be exactly the same as if it had been performed first.

2.6.1 BarCode Scanning

As mentioned earlier, each sample has associated with it a unique barcode and it may sometimes be convenient to scan a sample barcode and have the associated sample record displayed to the screen. To this end, the system has a very simple interface which allows a simple low-cost USB scanner such as the *Zebex* scanner shown in Figure 14 to be used to extract a barcode. The approach taken to facilitate this is brute force.

Figure 14: A Zebex scanner.

In the *Admin Tools* menu is a link called *Find Bar Code*. Clicking this takes the user to a very simple form with just a single entry field for a bar code. Now, assuming that the scanner is plugged in to the same PC, if the mouse is clicked in the search box, then when the scanner scans the barcode (usually a button needs to be pressed on the scanner) the actual code will magically appear in the box. Pressing the search button on the form should then produce the matching sample (see Figure 15. For this to work correctly, the barcode scanner needs to be put in *keyboard emulation* mode. Most scanners are capable of doing this, including the Zebex. Of course, you can also type in the bar code by hand if you don't have access to a scanner.

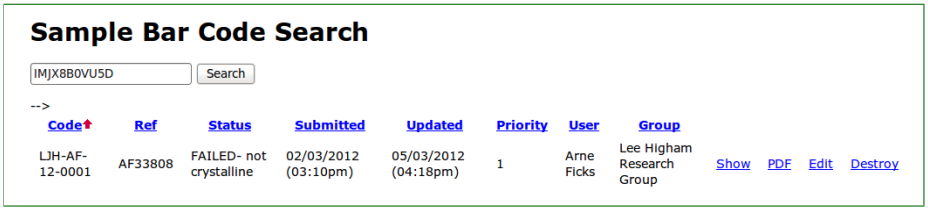


Figure 15: A successful search using the *Find Bar Code* form.

3 System Management and Internals

3.1 Introduction

In this section we will explain lower-level aspects of the system and its management. This includes the operating system, command shell, web server, database, ruby language, ruby gems, and the rails3 system which is used to do most of the programming.

3.2 Basic Components of the System

To begin with, we will describe the components of the system from the operating system right up to the *Rails3* software which is used to write the web interface to the sample database. Figure 16 summarizes the main components and their relationships.

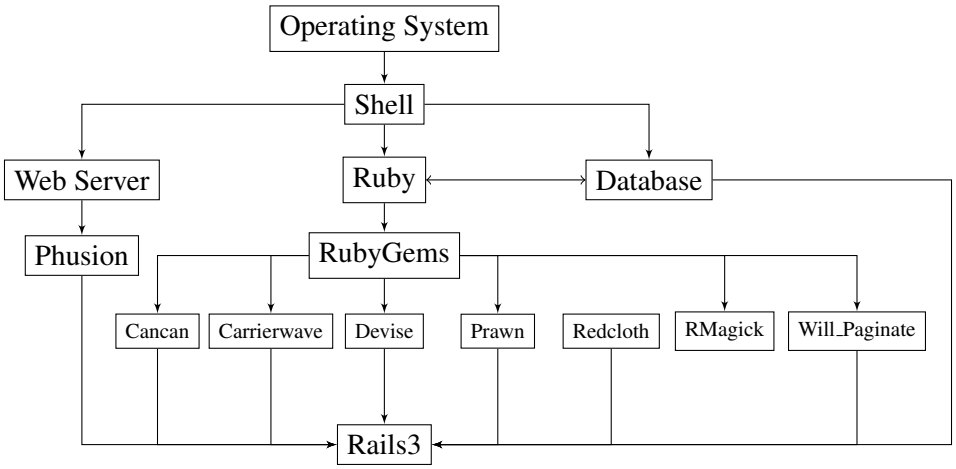


Figure 16: The main components of the sample tracking system and their relationships. This diagram takes a 'top-down' view starting with the operating system and moving through the various software layers down to the rails3 system itself on which the sample tracking software is based.

Let's now look at each of these components individually:

Operating System The system runs on a computer running the Ubuntu version of the linux operating system. The specific release used at present is *Ubuntu 10.04.3 LTS*. Release 10.04 is also known via the codename 'lucid'. LTS stands for 'Long Term Support'. Extended support for this version will last until 2014.

Shell Access to most software on linux is vis a command shell. There are many different shells available, the default being the *bash* shell. We will assume the use of bash throughout this guide, but other shells can be also be used with little, if any modification to the software itself.

Running under the shell are the three principal components of the system: a web server, database and the ruby programming language environment.

Web Server The web server we use is the ubiquitous *Apache*, specifically version 2.2.14.

Ruby The version of ruby that comes with Ubuntu 10.04.3 is 1.8.7. Unfortunately, this version is not recent enough to run a rails3 application. So, we use the *Ruby Version Manager* to run a more recent version of ruby (1.9.2).

Database Rails 3 requires a database application to store and retrieve sample data. We are using the *SQLite3* database. This is a 'lightweight' database which requires very little separate maintenance. In particular, it does not require a separate authentication process for acces, nor does it need to run a separate process in the background as with other database software such as *mysql*.

There are two other components of the system which sit between the web server, ruby and database layer:

Phusion Passenger is an Apache module – a 'plugin' – which interfaces a rails3 application to the Apache web server. It also supplies some debugging and error logging information which can be useful when things go wrong.

Ruby Gems are extension libraries to ruby. Some of them are specific to rails3, but most have wider applicability and can be used in more general applications. The ruby gems used in the sample tracking application are described in the next section.

3.2.1 Ruby Gems

The following Ruby Gems are used in the application:

cancan[7] written by well-known Rails programmer Ryan Bates is used for authorization. By this we mean controlling which parts of the application can be accessed by users. For example, some parts of the application can be accessed only by administrative users and others can be accessed by anyone — even those who have not registered to use the system.

carrierwave[8] is used to manage the upload of various files to the application including data files, image files and general asset files.

devise[9] is an authentication plugin to Rails which handles user data and user authentication. It provides related functionality such as forgotten password recovery (via email), user login statistics and more.

prawn[10] is a ruby library for producing PDF files. It is used to auto-generate the sample receipts.

redcloth[11] is a Ruby library used for parsing and display of Textile input.

rmagick[12] is a well-known software application consisting of both libraries and utility programs for manipulation of bitmap graphic files. This gem interfaces the ImageMagick libraries to Ruby allowing ImageMagick routines to be called from within a Ruby program.

will_paginate[13] is used to set up pagination of sample listings (and other things).

3.3 Web Server Management

Most of the time, you should need only two commands when managing the web server — the commands to switch it on and off. To do this use:

```
sudo apache2ctl start|stop
```

You would typically stop the server to do some maintenance such as editing a stylesheet or a major upgrade to the software. Occasionally you may need to change some server parameters in the web server configuration files. The main configuration file is `/etc/apache2/apache2.conf`. You will rarely, if ever, need to change anything here unless you want to tweak the server performance (in which case you'll definately need to know what you're doing). This file does contain some parameters for the apache *Phusion Passenger* module which is needed to interface Rails with the apache server:

```
LoadModule passenger_module /usr/local/rvm/gems/ruby-1.9.2-p290/
gems/passenger-3.0.9/ext/apache2/mod_passenger.so
   PassengerRoot /usr/local/rvm/gems/ruby-1.9.2-p290/gems/passenger-3.0.9
   PassengerRuby /usr/local/rvm/wrappers/ruby-1.9.2-p290/ruby
```

These configuration lines are added as part of the installation of *Phusion Passenger* and tell it where the ruby interpreter and passenger software reside in the file system.

The `apache2.conf` file also contains a line which refers to some other configuration files:

```
Include /etc/apache2/sites-enabled/
```

The above line tells the server to get some further configuartion details from the contents of the directory `/etc/apache2/sites-enabled`. `sites-enabled` contains a file called `000-default` which contains the descriptions of all the *virtual hosts**. The most important virtual host entry is the one for the host `crystal.ncl.ac.uk`. Here is the entry in full:

```
<VirtualHost *:80>
      ServerName crystal.ncl.ac.uk
      DocumentRoot /usr/local/share/sample_tracker/public
      <Directory "/usr/local/share/sample_tracker/public">
         AllowOverride all
         Options -MultiViews
      </Directory>
        ErrorLog /var/log/apache2/error-crystal.log
</VirtualHost>
```

*A single Apache web server can support multiple so-called virtual hosts which behave as separate web sites.

For virtual hosting to work, any name given to a virtual host (e.g. in this case `crystal.ncl.ac.uk`) must be an official alias for the actual apache web server (in this case `milkyway.ncl.ac.uk`). We will now describe in detail what these directives mean.

The first line is a tag which defines the beginning of a virtual host definition. It has the form `<VirtualHost *:80>`. The number `80` is the *port number**. The last line is a closing tag, `</VirtualHost>`. the lines in-between contain the directives which control the configuration of this host.

ServerName This directive sets the name of the virtual host, in this case `crystal.ncl.ac.uk`.

DocumentRoot Sets the absolute path of the directory on the server which contains all static documents which can be *directly* viewable on a web browser. No documents outside this directory can be viewed directly or downloaded. In this case, the document root is `/usr/local/share/sample_tracker/public` i.e. the `public` directory within the full sample tracking application. This is a standard rails convention.

ErrorLog Specifies the absolute path of the error log file for this virtual host. Here, it is set as `/var/log/apache2/error-crystal.log`.

There is also a `Directory` section in the virtual host definition. This has the form of an opening and closing tag with directives in-between. The opening tag has an argument which is the full path of the particular directory to which the directives apply. Note that the argument is in quotes. In this case the argument is the same directory as the document root. Note further that there may be more than one `Directory` section. The directives are as follows:

AllowOverride This directive controls whether directives contained in a file (conventionally called `.htaccess` contained in a particular directory can override earlier configuration directives at the server configuration level. The argument refers to which type of directive can be overridden — in this case `all`.

Options This sets a number of options when viewing files contained in the relevant directory. here, we have set the `-MultiViews` option which allows a single document to be displayed in different ways dependent on browser capabilities.

Location of the Sample Tracking Application

It should be clear from the above that if you want to change the location of the sample tracking application you must edit the virtual host directive file and change the `DocumentRoot` directive and `Directory` section argument appropriately.

We end this section on web management with a brief discussion about file permissions. Once you have uploaded the application and all its files to a suitable location you must make sure that the file permissions are correct. The correct permissions for all files are to set group ownership to `root` and user ownership to `nobody`†. This can be done with a simple one-line command:

```
sudo chown -R nobody.root <application document root>
```

*A port number defines a communications channel between computers on the Internet. Port 80 is usually used by the HTTP protocol for communication between web servers and browsers although other port numbers can be used.

†`nobody` is a special user account available on most linux systems.

3.4 File and Directory Management

In this section we describe the overall file and directory structure of the application. Some of the files contain parameters which can be changed to affect the behaviour of the application or the appearance of the views. The overall directory structure is shown in Figure 17.

The application root directory contains the following files:

```
config.ru Gemfile Gemfile.lock README.markdown
```

`README.markdown` is a brief summary of the application, written in *markdown*, a simple markup language similar to Textile. The contents of this file are automatically displayed when viewing the home page of the application on GitHub.

`config.ru` is a file used to initialize the application via a special software interface called *Rack**.

`Gemfile` contains a list of gems required to run the application. `Gemfile.lock` contains a list of all gems and their dependencies and is used to update or install additional gems for the application. In practice, only the `Gemfile` is actually edited.

There are 12 directories directly under the application root:

3.4.1 The app Directory

This directory contains the bulk of the application code and templates for most of the page views. It contains the following subdirectories:

controllers as its name implies contains code which *controls* the interface between the database and the browser. Each table in the database has its own controller code stored in a file in this directory. So, for example, the controller code for access to the samples table is contained in the file `samples_controller.rb`. The `.rb` extension indicates that this file contains ruby code.

helpers contains so-called 'helper' functions (again written in ruby) which are used to produce page views. Generally, there is a file of helper routines corresponding to each database table, for example the helper code containing routines to aid the display of sample data is stored in the file `samples_helper.rb`. Other files, such as `application_helper.rb` are more general-purpose and can be accessed more widely, not just to facilitate page display for particular database table items.

mailers contains cose used by the Rails mail subsystem. In the sample tracking application, when samples are updated, an email is sent to relevant users. The file `sample_mailer.rb` contains the ruby code which sends the email.

models is one of the most important directories because it contains the code which interacts with the database. In Rails, each database table defines a 'model' and the code to access this model is contained in a file in the `models` directory. In the case of the samples model for example, this file is called `sample.rb`.

pdfs contains code to generate PDF versions of model data. Only sample data is rendered to PDF form in this application. The rendering code (using the Prawn library) is contained in the file `sample_pdf.rb`.

*Rack is part of the interface between a rails application and a web server.

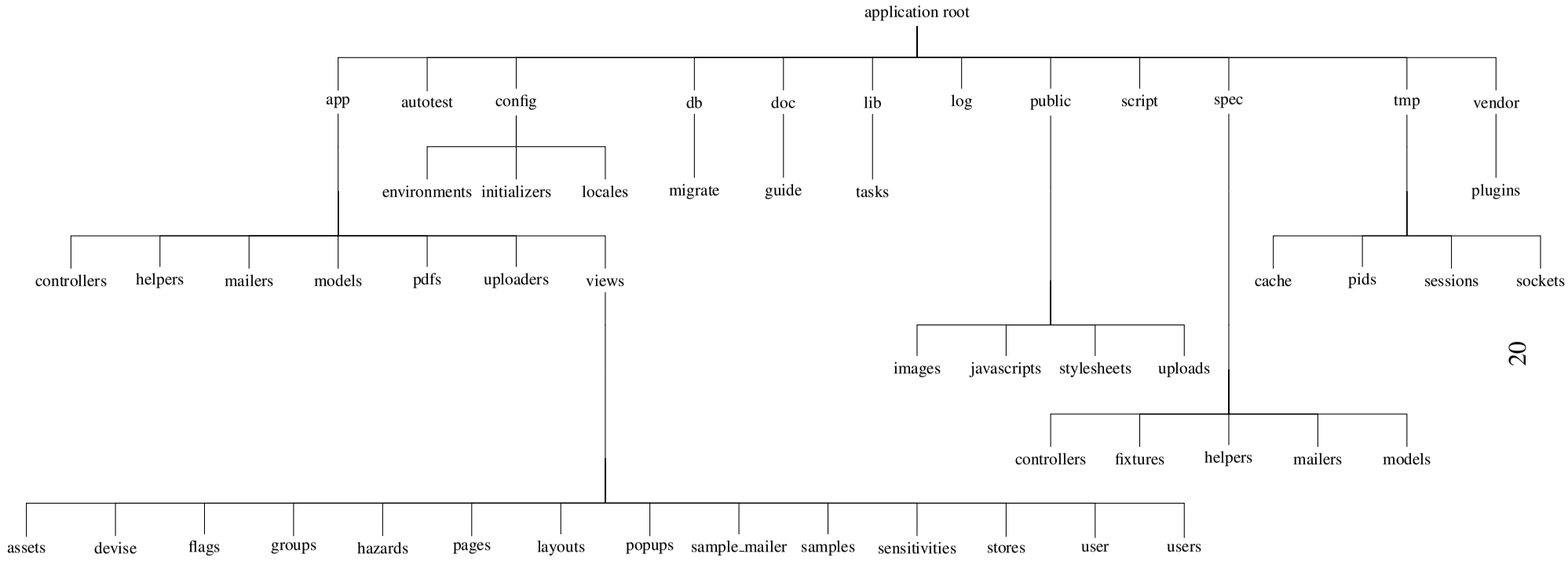


Figure 17: The application directory structure. Only directories are shown here. See text for a detailed explanation.

uploaders contains code for uploading files to the server. For each upload type, there is a file containing the appropriate code for that file type (although often the code will be almost identical in many cases). The standard naming convention for these files is `<uploader_name>_uploader.rb`. For example, in the case of an *asset* upload, the code to do this is stored in the file `document_uploader.rb` because the *asset* model has a field value called *document* which has been decalred in its model code as an uploader called *document**. Other uploader file types include zip data files, sample images and synthetic route diagrams.

views contains temmplate HTML code for displaying page views of data from the database. As can be seen from Figure 17, the views directory itself contains lots of subdirectories. each subdirectory corresponds to a model from the sample database and contains the view templates for various different types of view associated with that model. Let us look at the central model in the system, the *sample* model. A listing of the `sample` subdirectory shows the following files:

```
edit.html.erb          groupindex.html.erb  queue.html.erb
findbarcode.html.erb   index.html.erb       show.html.erb
_form.html.erb         new.html.erb         userindex.html.erb
```

These files all contain a mixture of traditional HTML and embedded ruby — hence the `.html.erb` extension. Some of the files are specific to the sample model but others are common to other models. In particular, the files (omitting the extension) `new`, `show` and `edit` are templates for the *create*, *read* and *edit* operations. Additionally, `_form` is a form template for editing or entering model data. This form template is actually included by the new and edit templates. To get an idea of how this works, here is the edit template contained in the file `edit.html.erb`:

```
<% title "Edit Sample #{@sample.code}" %>

<%= render 'form' %>

<p>
  <%= link_to "Show", @sample %> |
  <%= link_to "View All", samples_path %>
</p>
```

Note that there are familiar HTML elements such as the paragraph tags `<p>` and `</p>`. Text within the `<%` and `%>` tags is ruby code. Note in particular the `render 'form'` directive. This reads in the content of the file `_form.html.erb`.

3.4.2 The public Directory

This directory contains files that can be *directly* viewed by a web browser. For example, if you create a file called `info.html` in this directory then (provided its permissions are suitable) you will be able to view its contents simply by entering the following URL into your web browser:

*Note here that it is the uploader name which determines the name of the file containing the uploader code, not the field name. In this case the two are equal, but they don't have to be.

`http://crystal.ncl.ac.uk/info.html`

The public directory contains things such as stylesheets, error pages corresponding to the standard HTTP error codes* 404, 422 and 500, generic images (e.g. icons for buttons and logos), javascript files and any files which are uploaded to the server.

This last category includes all file uploads to do with samples and assets. These files are all placed in a directory called `uploads` within the public directory. Uploaded files are further organised depending on whether they are to do with samples or whether they are a generic asset. An example will best illustrate this. Suppose that a zip file of sample analysis data is uploaded by an administrator. He is free to give the zip file any name. Suppose he calls the file `results.zip`. Then, when the file is uploaded via the sample edit form, it will be placed in the following location:

`public/uploads/sample/zipdata/62/results.zip`

Similarly, a synthetic route image file called `synthroute.png` uploaded by a user when submitting a sample analysis request, will be placed in the file:

`public/uploads/sample/synth/<sample_id>/synthroute.png`

where `<sample_id>` is the id automatically assigned by the database engine to that particular sample†. To summarize, the general form of the file path for uploaded files is:

`<application root>/public/uploads/<model>/<uploader>/<id>/<file name>`

3.4.3 The config Directory

This directory contains configuration and initialization files for the rails application. You will rarely, if ever need to edit most of these. However, there is one file, `environment.rb`, which contains some parameters that you may want to change occasionally. Here is a listing of this file:

```
# Load the rails application
require File.expand_path('../application', __FILE__)

# Initialize the rails application
SampleTracker::Application.initialize!
ITEMS_PER_PAGE = 10 # default num items per page for will_paginate

###################  SPECIFIC GLOBAL VARS ####################
TEXTILE_REF_URL = "http://redcloth.org/textile/"
CRYS_EMAIL = "xray.cryst@ncl.ac.uk"
LOCAL_ADMIN_EMAIL = "crysadmin@milkyway.ncl.ac.uk"
TEXTILE_QUICK_REF_URL = "http://en.wikipedia.org/wiki/Textile_%28markup_language%29"
SAMPLE_INTRO_TEXT = "a unique code and a barcode will be
automatically generated on submission of this form when a new sample
is created. For synthetic route files, please use either the JPG or PNG
bitmap image format.
Note further that for security reasons, if there are validation errors
in the form, you will have to re-select the names of uploaded files.
Also remeber to set the priority number, the form will not validate unless
you do this (if you are not going to submit several samples in a short
space of time we suggest setting the priority number to 1)."
QUEUE_INTRO_TEXT = "The following gives an appoximate wait time before your
sample will be analysed. The actual time will vary depending on sample
quality and priority number."
```

There are several constants (those parameters in capitals) which define parameters which affect the way the application runs:

*404 and 500 refer to 'file not found' and 'internal server error' respectively. A 500 error often arises when a program script malfunctions or there is a database problem.

†In fact another file will be automatically created in the same directory, this file is a thumbnail version of the user's file and will be given a similar name to that which the user submitted except prefixed by the string `thumb_`.

ITEMS_PER_PAGE is the parameter that controls the *default* pagination of lists as mentioned elsewhere. Lists will use this parameter as a default, but in most cases users can change the pagination on-the-fly.

TEXTILE_REF_URL contains a URL pointing to a complete Textile reference — currently it refers to the RedCloth web site.

CRYS_EMAIL is the email account to which sample submission requests are sent.

LOCAL_ADMIN_EMAIL this is set to the email address which appears in the *From* part of all emails which are automatically sent to users by the sample tracking system.

TEXTILE_QUICK_REF_URL contains a URL to a Textile quick reference page. This is currently set to the Wikipedia entry for Textile and is used on the page edit screen to help administrators edit static pages as shown in Figure 5.

SAMPLE_INTRO_TEXT contains the text that appears in a box at the beginning of the sample submission form — see for example Figure 10.

QUEUE_INTRO_TEXT contains the text that appears in a box at the beginning of the sample queue page.

Note that if any of these parameters are changed, the server will need to be restarted before the changes will take effect.

4 The Database

4.1 Introduction

The core of the system is the database which holds information about users, samples etc. In this section we describe the whole database structure (or *schema* in database parlance). The easiest way to get an overall view of the database schema is to study Figure 18. This shows all the tables, fields and relationships in a single diagram. We now give a brief description of each table.

Note that all tables except join tables have an autoincremented integer field called `id` which servers as the unique primary key for each record in the table. The `id` field will not be listed explicitly in the description of each table. All non-join tables also have two other fields, `created_at` and `updated_at` in a datetime format. Again, we will not explicitly list these fields in the description of the tables which follows.

4.2 The Samples Table

The samples and users tables are the key parts of the database as is evident from Figure 18. They are related to each other via a *one-to-many* relationship, i.e. *one* user can have *many* samples. The samples table consists of the following fields:

code a string, automatically generated by the system having the general form `AAA-AA-YY-1111` where the `AAA` and `AA` represent 3-letter codes for group and submitter respectively; the `YY` represents the year and the `1111` represents a number which is incremented for that group but reset to zero at the start of each calendar year.

cif a string representing the chemical formula of the sample in cif format.

synth a string representing the file name of an image file specifying the details of the synthesis.

coshh_name a string representing the name of the solvent (if any).

coshh_info another string describing any procedures in case of contact with the sample.

coshh_desc a text field providing a brief description of the sample (e.g. organic amide).

params a string representing unit cell parameters or CSD/Newcastle code for possible by-products or previously obtained, unpublished results.

priority an integer between 1 and 9 to give an indication of priority.

powd a boolean parameter indicating if the sample requires powder diffraction (y/n).

chiral another boolean indicating whether the molecule is chiral (y/n).

costcode a string providing a cost centre code for charging if relevant.

barcode a string field for an automatically generated Code39 standard barcode.

user_id this integer holds the `id` field of the user who requested the sample analysis.

flag_id an integer holding the `id` field of the status flag of the sample.

userref a string for a user-defined reference. This is required to be an alphanumeric sequence of characters *without spaces*.

zipdata a string holding the name of a zipfile containing the results of the analysis.

sampleimage a string holding the name of an image file of the sample molecule after it has been identified by the analysis.

reference a text field for a published reference (typically in the form of a DOI).

comments a text field for any general comments the user wishes to make about the sample.

colour a string holding information about the colour of a sample after analysis.

size a string holding information about the size of a sample after analysis.

shape a string holding information about the shape of a sample after analysis.

feedback a text field containing any additional comments on the sample by crystallography staff.

4.3 The Users Table

The users table, in addition to maintaining a record of users and their samples, also serves as a key part of the authentication and authorization system which will be described later. the users table is related to the samples table via a *one-to-many* relationship, i.e. *one* user has *many* samples.

email a string holding the email address of the user. This serves also as the user login id.

encrypted_password a string holding the user's password in an encrypted form.

reset_password_token a string containing a special token used if the user has forgotten his password and needs to reset it.

reset_password_sent_at a datetime field recording the time a token enabling a user to reset his password was sent.

remember_created_at a datetime field specifying the time at which a user requested that his login id be remembered by the browser so he need not type in his credentials.

sign_in_count an integer holding the number of times a user has logged-in.

current_sign_in_at a datetime field holding the sign-in time for the current session.

last_sign_in_at a datetime field holding the last sign-in time for the user.

current_sign_in_ip a datetime field holding the user's ip address for the current session.

last_sign_in_ip a datetime field holding the previous login ip address for the user.

group_id an integer representing the `id` field of the group to which the user belongs.

admin a boolean field indicating whether the user is an administrator (y/n).

firstname a string holding the user's first name.

lastname a string holding the user's last name.

leader a boolean field indicating whether the user is a group leader (y/n).

enabled a boolean field indicating if the account is enabled (y/n).

4.4 The Stores, Hazards and Sensitivities Tables

These tables are each very similar and have the same basic structure. They are used to specify storage, hazard and sensitivity properties for a sample. They all have a *many-to-many* relationship with the samples table. This is because a sample can have, for example, *many* storage requirements, but also a single storage requirement can be associated with *many* samples. All these tables have essentially the same fields:

name a string defining a short name for the property.

description a text field describing the property at greater length.

For historical reasons, the hazards table uses the names `hazard_abbr` and `hazard_desc` for the `name` and `description` fields. Also the `hazard\_desc` field is a text field rather than a string.

Associated with these tables are three further *join tables* which facilitate the many-to-many relationship between a sample and its properties. These join tables are called `samples_stores`, `samples_hazards` and `samples_sensitivities`. They all contain two fields corresponding to the sample `id` field and the associated property `id` field. For example, `samples_stores` contains the fields `sample_id` and `store_id`. Both these fields are integers of course.

4.5 The Groups Table

This table represents groups of users, normally research groups but also perhaps external companies etc. It is a simple table, but important in the way the whole system works. It contains the following fields:

group_abbr a 3-letter string as an abbreviation for the group. Amongst other things this is used to form part of the sample code string mentioned earlier.

group_desc a string giving a more complete description of the group.

4.6 Other Tables

There are several other tables which are less important than the ones discussed so far in the sense that they are strictly not necessary for a working sample tracking system. However, they do assist in making the system much easier to manage and also help making the system much friendlier for users. These tables are the `assets`, `pages` and `popups` tables.

4.6.1 The Pages Table

The purpose of this table is to provide a means by which administrators can add 'static' content to the sample tracking web site. Each static page has its content stored in this table. The fields are:

name a string storing a name for the page. This is typically used to provide a title for the page in a web browser window.

permalink another string used to provide a short, quick URL for the page.

content a text field which contains the page content. This is expected to be written in Textile markup language (although a mixture of pure HTML and Textile can be used.

menu a boolean specifying whether this page should appear on the *Information Menu*.

priority an integer specifying a priority for ordering the page on the *Information Menu*.

4.6.2 The Assets Table

The assets table keeps a record of general files which have been uploaded to the server. These files are typically graphical images, pdf documents etc. and will usually be referenced in one of the static pages created by administrators which are stored in the `pages` table. An 'asset' is simply one of these uploaded documents and the `assets` table keeps a record of it. The fields are:

document the full path name of an uploaded document. This path name is ultimately assigned using the carrierwave file uploading plugin to ruby on rails.

description a text field giving a brief description of the document.

4.6.3 The Popups Table

This table stores descriptive information about the primary fields in the samples table. It has two fields:

name this string should have the same name as one of the sample fields for which a detailed description is required.

description a text field giving a detailed description of the associated sample field in the corresponding `name` field.

The `popups` table, as its name implies, provides descriptive text in popup boxes whenever a user hovers the mouse over the appropriate field in the sample submission form.

4.6.4 The Flags Table

This table stores a set of status flags together with a more verbose description of what the flag means.

name a string containing the name of the status flag, e.g. `SUBMITTED`, `COMPLETED` etc. There can be any number of flags but the aforementioned flags must be present because when the sample is originally submitted it is, by default, given the status `SUBMITTED`. Also, when analysis is finished, the sample queue list will omit all samples which have had the `COMPLETED` flag set or have a status flag which begins with the string `FAILED`.

description a text field giving a detailed description of the associated flag name.

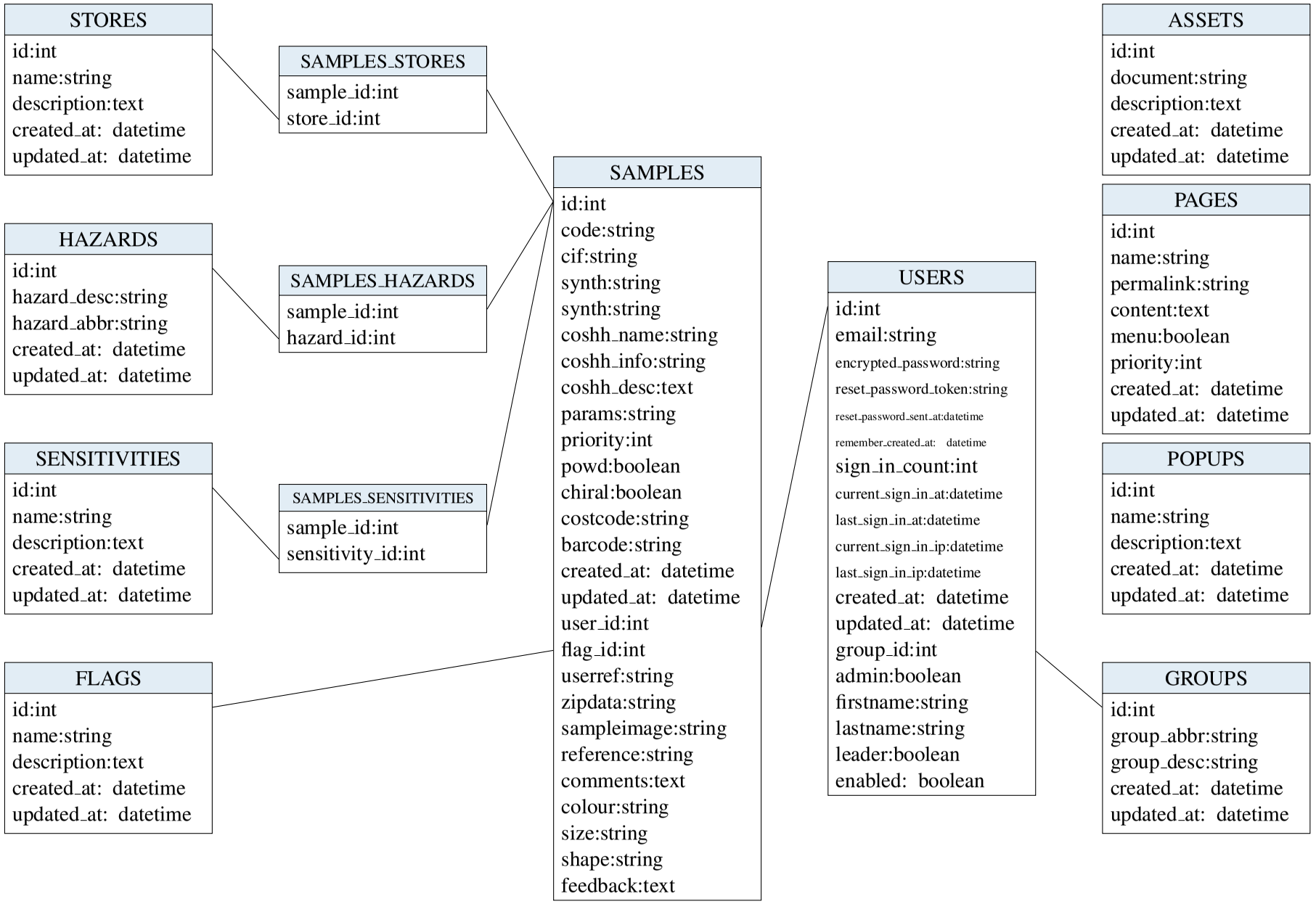


Figure 18: The overall database schema. Relationships between tables are indicated with lines joining the relevant fields. Note that the tables `SAMPLES_STORES`, `SAMPLES_HAZARDS` and `SAMPLES_SENSITIVITIES` are *join tables* which serve only to facilitate a many-to-many relationship between the tables they link.

References

[1] http://www.ruby-lang.org/

[2] http://rubyonrails.org/

[3] http://www.apache.org/

[4] http://www.ubuntu.com/

[5] http://www.sqlite.org/

[6] http://en.wikipedia.org/wiki/Textile_%28markup_language%29

[7] https://github.com/ryanb/cancan

[8] https://github.com/jnicklas/carrierwave

[9] https://github.com/plataformatec/devise/

[10] http://prawn.majesticseacreature.com/

[11] http://redcloth.org/

[12] http://rmagick.rubyforge.org/

[13] https://github.com/mislav/will_paginate