Cmput 391 - Radiology Project
Aaron Tse, Cody Ingram, Ondra Chan

Overview

In order to implement our system, our group opted to use the Django web framework. We made this decision for several reasons: for one, we wanted to use a more modern approach to developing web based systems in place of JSP or PHP and plain HTML, and Django provided a great framework that is popular in todays industry. Django also enabled us to develop our web app using python, which the majority of the group had much more expertise in compared to the alternatives. Lastly, Django still allowed us to use a SQL database, specifically MySQL. This enabled us to run the database from our laptops, as mySQL is easily available and powerful.

In order to fulfill the requirement of having a three tier system, we used mySQL as our database, Django as our business logic server, and Django using ngrok tunnelling to provide the client side available in any browser.

Because we used Django, there are many files in our directory that are automatically generated and not created by us. The files in which we implemented the project are the following: models.py, forms.py, urls.py, views.py, settings.py, and all the html files.

Tools

Django - https://www.djangoproject.com

Django is a free, open-source web framework for developing web applications in python. It allowed us to easily develop our website and focus on coding the actual project without worrying about small, low level problems. Django provides us the ability to easily connect with our mySQL database as well a run a virtual server in order to provide our client side over a web browser.

Ngrok - https://ngrok.com

Ngrok is a free program that allows you to expose a local web server over the internet. We used ngrok in order to relay Django's built in test server over the internet so that our

website is accessible via a web browser on a device other than the computer that the Django server is run on, thus providing the client side interface to our application.

MySQL - http://www.mysql.com/

The MySQL community edition (GPL) is one of the worlds most popular free, open-source database solutions. We chose mySQL to implement our database because of it's popularity, ease of use, and because of it's similarity to Oracle's database which we have used in this class and others.

Modules

Every Module is implemented with associated URLs in the 'urls.py' file, html files, and views (represented as functions) in the 'views.py' file. Because we used Django API for some queries to the database, rough translations of the mySQL statements are given.

Login Module - Cody

The login module allows users to login to their account(s) by simply entering their username and password. The module uses the 'myLogin(username, password)' function that queries the database for a matching username to the one entered and checks if the associated passwords match. If the passwords do not match or the query returns nothing then an error message is displayed and the user is not logged in. Once the user successfully logs in, they are given several options. Depending on the users class type, they will be able to view buttons to access different modules. All users are able to view the logout, change password, and change info buttons as a part of the login module. The logout button returns them to the login page so they are able to login as a different user. The change password button navigates them to a page where they can enter a new password. If they enter two different passwords, or they enter their current password, they are given an error message and their password is not changed. If they correctly enter a new password, then the users password is changed. The Change info button navigates to a page where the user can view their personal info (information in the persons table) and change any or all of their information (blank entries are not valid).

URLs: default index URL, home/, changePass/, changeInfo/

HTMLs: home.html, changeInfo.html, changePass.html

Views: index, change_info, change_pass, user_login

Example Queries:

1) SELECT *
   FROM Users
   WHERE user_name = username

2) UPDATE Users
   SET user_name = newUsername
   WHERE user_name = currentUsername

User Management Module - Cody

The User Management Module allows users who are admins to add new users, update an existing user, or add a new doctor/patient relationship. Adding/updating users includes all of the information in the users and persons table. If new user is selected, then the user is prompted to enter all necessary information. If any input is incorrect, a message will be displayed and a new user will not be created (date registered is automatically added as the current date). If the user selects update, they will then be prompted to enter the username of the user they wish to update. If a user enters an existing username, they will then be shown a form with the users current information, and will be able to enter new information much the same as a new user. If the user selects the update family doctor button, they will be taken to a page where they can select a doctor and a patient to create a relationship between.

URLs: userManagment/, newUser/, updateUser/, familyDoctorUpdate/

HTMLs: userManagment.html, newUser.html, updateUser.html, updateFamilyDoctor.html

Views: user_managment, new_user, update_user, update_family_doctor

Example Queries:

1) UPDATE Users, Persons
   SET first_name = newName, email = newEmail
   WHERE User.user_name = currentUsername

2) INSERT INTO Users(username, password, ‘a’, 1234, ‘2015-12-01’)
3) INSERT INTO Family_doctor(1234, 1221)

Report Generating Module - Aaron

The report generating module allows admins to generate a report containing a list of patients who were sent in to test for a certain test type. The admin will input a test type and a time frame (start date/end date) and receive a set of rows as output. Each row of this report will contain the patient’s first name, their address, their phone number and the date of the first record within the specified time frame. The submission requires that all of the fields be entered and will return a warning otherwise.

URLs: report/

HTMLs: report.html

Views: report

Queries:

```
(Select     p.first_name, p.address, p.phone, min(r.test_date)
from   RadiologySys_persons p, RadiologySys_radiology_record r
where  p.person_id = r.patient_id_id and
       r.test_type = %s and
       r.test_date >= %s and
       r.test_date <= %s
       group by p.first_name, p.address, p.phone', [diagnosis, tstart, tend])
```

where diagnosis is the diagnosis, tstart is the start date and tend is the end date

Uploading Module - Cody

The uploading module allows users who are radiologists to enter a radiology record and attach images to that record. The user will press the upload record button, and enter the necessary information on the next page, including choosing an existing doctor/patient/radiologist from the drop downs. The submission will not go through unless all fields are filled and the record ID is unique. To attach an image to a record, the radiologist presses the upload images button, then chooses an existing record to attach the images to, and selects three images (thumbnail, regular, large). The submit will not work until the user selects all three images (as well as an image ID and record).

URLs: uploadRecord/, uploadImages/

HTMLs: uploadRecord.html, uploadImages.html

Views: upload_images, upload_record

Example Queries:

1) INSERT INTO Radiology_record('11', '12', '13', 'Flu', datetime.date(2005, 8, 20), datetime.date(2005, 8, 20), 'Infected', 'Likely developed from sore throat')
2) INSERT INTO Pacs_images(12, 2, image_thumb.jpg, image_regular.jpg, image_full.jpg)

Search Module - Ondra

The Search Module allows all users to search the database for radiology reports. There are three input fields:

1) Keywords: The user can input a list of keywords and the Search Module will search the radiology records with each keyword and return any record that has that word or character pattern in it.
2) Start Time: The beginning of the time period the user wishes to search for.
3) End Time: The ending of the time period the user wishes to search for.

If a Start Time or End Time is selected without the other, a warning will prompt you to fill in the other field.
If both fields are filled in, the Search Module will show all records that fall within that time period, as well as satisfying the security constraints.
For the security of the records, the four user classes (Admin, Doctor, Patient, Radiologist) can all view different kinds of records. The Admins can view all records, Doctors can only view the records of their patients, Patients can only view their own records, and Radiologists can also only view their own records.
Four queries were created in order to reinforce the security restrictions, eg one query is for the Admins, and another is for Radiologists that restrict them to only viewing their own records.
To implement the keyword search, the list of keywords are broken down to singular words, and by using the LIKE comparator to compare the keyword to the first/last name of the patients, doctors, and radiologists, and the diagnosis, and finally the description. Then by using a for loop the query iterates through the list of keywords.
The reports do not include the medical images associated with them, and the records are always ordered the same. Also the records shown from searching multiple keywords

do have duplicates. All of these requirements were not completed because I failed to start working on this project ahead of time and ran out of time once I had began.

URLs: search/

HTMLs: search.html

Views: search

Queries:

SELECT Distinct p.first_name, d.first_name, i.first_name, r.test_type, r.prescribing_date, r.test_date, r.diagnosis, r.description
from (RadiologySys_persons p, RadiologySys_persons d, RadiologySys_persons i, RadiologySys_radiology_record r)
where (p.person_id = r.patient_id_id and d.person_id = r.doctor_id_id and i.person_id = r.radiologist_id_id) or p.first_name LIKE %s or p.last_name LIKE %s or d.first_name LIKE %s or d.last_name LIKE %s or i.first_name LIKE %s or i.last_name LIKE %s or r.test_type LIKE %s or r.diagnosis LIKE %s or r.description LIKE %s

Data Analysis Module - Aaron

The Data analysis module allows admins to get an OLAP report based on their specifications. There are 3 options (patient name, test type, time) in the data cube and the number of images is a function of those 3 options. This gives us the fact table:

Totals(patient_id, test_type, time, number of images)

By default, “time” is represented as “all”, that is it represents not selecting a time hierarchy.

To accomplish the queries to return the OLAP report, we have created a table, TIME(time_id, week, month, year) which represents the week, month, and year of a given day. The time_id for a record is dynamically generated on insert based on the record’s given test_date.

Next, using this, we create a temporary table to store all of the information that we need to use in our main query. We accomplish this with the following query:

```
Create temporary table if not exists temp as (
    Select person_id, week, month, year, test_type, image_id
    From RadiologySys_persons p
        Inner Join RadiologySys_radiology_record r
            on r.patient_id_id = p.person_id
        Inner Join RadiologySys_time t
            on t.time_id = r.time_id_id
        Inner Join RadiologySys_Pacs_images i
            on i.record_id_id = r.record_id)
```

which gives us a variant on the fact table presented above:

Temp(person_id, week, month, year, test_type, image_id)

where image_id is the id of an image that is linked to a record of type t and person p (and thus can be used to find the number of images with “count”)

A dynamic query string is built to query this Temp table to return the information requested.

For example:

if no options were selected (type == None, name == None, time == ‘all’):

```
Select count(image_id)
From temp
```

which simply returns the total number of images in the database

if 1 option is selected (For example, type):

```
Select count(image_id), test_type
From temp
Group by test_type
```

which returns the total number of images sorted by test type (ie each row represents the total number of images for that test)

if 2 options are selected (For example, name and time == “week”):

```
Select count(image_id), person_id, week
From temp
Group by person_id, week
```

which returns the total number of images sorted by patient and week (ie each row represents the total number of images a patient has taken each week)

And so on and so forth. Note: you cannot select week and month which respects the rules that are set by the data cube.

URLs: analysis/

Views: analysis

Queries: See above

Acknowledgments

As mentioned in the tools section, we used Django, ngrok, and mySQL in our project (links above). The official documentation (found at djangoproject.com) was our source of external help.