**Project Report**

The project is split into eight major component's which are:

* General
* Login
* User Management
* Report Generation
* Uploading
* Searching
* Data Analysis
* Scripts

Each component is described in more detail bellow and contains descriptions about the contents of the PHP and HTML files in each component including but not limited too; functions, sql queries and tasks.

**General**

These are general files and functions used throughout the web page.

**File:** PHPconnectionDB.php

**Description**: Contains all the necessary code and information to connect to the oracle database.

**Functions:**

* **connect()**: Connects to the oracle database

**File:** Date.php

**Description**: Prints a basic set of drop downs for use in picking a date. Inputs are very basic and do not contain any check’s for wrong date input.

**Functions:**

* **date\_picker($name, $startyear $endyear)**: Returns a string representing an html drop down for date selection
* **get\_month\_string($number)**: Returns the name for a specific date number.

**File:** DisplayIDs.php

**Description:** A file that contains functions to display the person\_id’s of doctors, radiologists, patients, and records.

**Functions:**

* **displayPatients():**
  + echos the person\_id’s of all patients in the database in between <option> tags, to be used for dropdown menus.
* **displayDoctors():**
  + echos the person\_id’s of all doctors in the database in between <option> tags, to be used for dropdown menus.
* **displayRecords():**
  + echos the record\_id’s of all radiology\_records in the database in between <option> tags, to be used for dropdown menus.
* **displayRadiologists():**
  + echos the person\_id’s of all Radiologists in the database in between <option> tags, to be used for dropdown menus.

**Login**

This section handles logging users into the system. Upon successful login a session instance of the user class is populated with corresponding users username/password/class. If an invalid username/password combination is set the user is redirected to the login page with an error message.

**File:** index.html

**Description**: Simple login html page. Contains the form that is populated then sent to the LoginModule.php file for processesing.

**File:** bad\_login.html

**Description**: Simple html page, similar to index.html, but also displays the login error. Only viewed if a user enters an invalid username/password combination.

**File:** LoginModule.php

**Description**: forms submitted from index.html & bad\_login.html pass through this PHP script. The scrip first checks if the forms were actually submitted, if not it redirects bag to the login page (index.html). If they have been properly submitted the script processes the information by making a query to the database to check if a row of the users table exists with the corresponding username/password combination. If they do then the login is successful and the user is redirected to the home.php page.

**Queries**:

Checks if a user exists and returns the person\_id and class:

SELECT person\_id, class FROM users WHERE user\_name = '" . $\_SESSION['user']->username . "' AND password = '" . $\_SESSION['user']->password . "'"

**File:** account\_settings.php

**Description:** This file displays the users current personal/user info and allows them to modify their username, password, address, phone number, email, etc. (not class, or id). They can do this by filling in the info in the displayed forms.

The users current info is retrieved and set into the session user class instance via the file getInfo.php

**File:** getInfo.php

**Description:** This file is used to retrieve the users current personal/user info from the database and set the session user instance variables accordingly.

**Functions:**

* **getUserInfo():**
  + Retrieves the up-to-date username and password from the database and sets the sessions user instance’s variables accordingly.
  + **Query:** "SELECT \* FROM users WHERE person\_id = " . $\_SESSION['user']->user\_id;
* **getPersonalInfo():**
  + Retrieves the up-to-date first names, last name, address, phone no, email of the current user and sets the session’s user instance’s variables accordingly.
  + **Query:** "SELECT \* FROM persons WHERE person\_id = " . $\_SESSION['user']->user\_id;

**File:** pass\_change.php

**Description:** This is brief script is called after the form for password changing is submitted. This script checks that the password entered in the old password area matches the user’s current password. It also checks that the two copies of new password that has been entered in the form match and are of correct form.

**Queries:**

* Checks if the old password is valid:

"SELECT user\_name FROM users WHERE password = '" . $oldPass . "' AND person\_id = " . $\_SESSION['user']->user\_id;

* Updates the user in the database with the new password:

"UPDATE users SET password='" . $newPass1 . "' " . "WHERE person\_id = " . $\_SESSION['user']->user\_id;

**File:** personalInfoUpdate.php

**Description:** This is a script that is called from a form in the account\_settings.php that updates the person in the database with the corresponding data that was entered in the form.

**Queries:**

* Updates the persons table with the new information:

"UPDATE persons SET first\_name='" . $\_SESSION['user']->first\_name .

"', last\_name = '" . $\_SESSION['user']->last\_name .

"', address = '" . $\_SESSION['user']->address .

"', email = '" . $\_SESSION['user']->email .

"', phone = '" . $\_SESSION['user']->phone .

"' WHERE person\_id = " . $\_SESSION['user']->user\_id;

**Functions:**

* **checkUsername($userName):**
  + This function checks the username which is a part of the users table (not the persons table like the other variables). It checks that the new username is of correct format and that it is unique.
  + **Queries:** (function specific)
    - Checks if the username is unique

"SELECT \* FROM users WHERE user\_name = '" . $username . "'"

* + - Updates the user in the database with the new username

"UPDATE users SET user\_name = '" . $username . "' WHERE person\_id = " .

$\_SESSION['user']->user\_id;

**User Management**

This section handles allowing a user to view, insert, and update into the SQL tables for users, persons, and family doctor. This module can only be viewed and used by users tagged as administrator.

**File:** user\_management.php

**Description**: Provides links for the three tables for the user to enter. Does not display anything else.

**File:** um\_persons.php

**Description**: Allows a user to view, insert and update into the persons table.

**File:** um\_users.php

**Description**: Allows a user to view, insert and update into the persons table.

**File:** um\_familyDoctor.php

**Description**: Allows a user to view, insert and update into the persons table.

**File:** changeUsers.php

**Description**: Hidden from the user, provides the back end of um\_users.php to insert or update into the users table.

**Functions:**

* **validateInsert()**: Checks post variables from insert form and throws necessary errors
* **validateUpdate()**: Checks post variables from update form and throws necessary errors

**Query:**

INSERT INTO users(user\_name, password, class, person\_id, date\_registered)

VALUES ($\_POST['iNewUserName'],

$\_POST['iNewPassword'],

$\_POST['iNewClass'],

$\_POST['iNewPersonID'] ,

$\_POST['iNewDateRegistered'])

OR

UPDATE users SET

**//If the field has been set, include that line in the query**

user\_name ='" . $\_POST['uNewUserName'] ,

password ='" . $\_POST['uNewPassword'] ,

class = '" . $\_POST['uNewClass'] ,

person\_id = '" . $\_POST['uNewPersonID'] ,

date\_registered = '" . $\_POST['uNewDateRegistered']

WHERE user\_name= $\_POST['uOldUserName']

**File:** changePersons.php

**Description**: Hidden from the user, provides the back end of um\_persons.php to insert or update into the persons table. Determines if an update value has been set, otherwise it inserts a new person with id 1 greater than the current maximum person id.

**Functions:**

* **validateForm()**: Checks table constraints on all variables in the form and throws necessary errors

**Query:**

UPDATE persons SET

//**if the variable is set, include that line in the query**

first\_name ='" . $\_POST['newFirstName'] ,

last\_name ='" . $\_POST['newLastName'],

address = '" . $\_POST['newAddress'] ,

email = '" . $\_POST['newEmail'] ,

phone = '" . $\_POST['newPhone'] ,

WHERE person\_id=" . $\_POST['oldPersonID']

OR

INSERT into persons(person\_id, first\_name, last\_name, address, email, phone)

VALUES ($nextid ,

$\_POST['newFirstName'] ,

$\_POST['newLastName'] ,

$\_POST['newAddress'] ,

$\_POST['newEmail'] ,

$\_POST['newPhone'])

**File:** changeFamilyDoctor.php

**Description**: Hidden from the user, provides the back end of um\_familyDoctor.php to insert or update into the family\_doctor table.

**Functions:**

* **validateInsert()**: Checks post variables from insert form and throws necessary errors
* **validateUpdate()**: Checks post variables from update form and throws necessary errors

**Query:**

INSERT INTO family\_doctor(doctor\_id, patient\_id)

VALUES( $\_POST['iDocID'] , $\_POST['iPatID'] )

OR

UPDATE family\_doctor SET doctor\_id = $newDoc , patient\_id = $newPat

WHERE doctor\_id = $\_POST['uOldDocID'] AND

patient\_id= . $\_POST['uOldPatID'];

**Report Generation**

This section handles generating a list of patients with a specific diagnosis for a specific time period. Requires both a diagnosis and a start – end date in order to return any information. This module can only be used by users tagged as administrator.

**File:** ReportForm.php

**Description**: Contains the code to create the form required in order to input the diagnosis and date information for generating a report.

**File:** ReportGenerator.php

**Description**: The main entry point for report generating. Gathers information from the form and the session then calls the function to generate the report.

**File:** ReportGeneratingModule.php

**Description**: Contains the functions necessary to generate the proper sql query, execute it and print the returning information to the screen.

**Functions:**

* **generateReport($diagnosis, $start\_month, $start\_day, $start\_year, $end\_month, $end\_day, $end\_year)**: Generates a sql query based on the input and prints the results to the screen in a table.

**Query:**

SELECT first\_name, last\_name, address, phone, MIN(test\_date), diagnosis

FROM radiology\_record r, persons p

WHERE contains(r.diagnosis, \'' . **$diagnosis** . '\', 1)**User Management**

> 0

AND

r.patient\_id = p.person\_id

AND

(r.test\_date BETWEEN

TO\_DATE(\'' . **$start\_month** . **$start\_day** . **$start\_year** . '\', \'MMDDYYYY\')

AND

TO\_DATE(\'' . **$end\_month** . **$end\_day** . **$end\_year** . '\', \'MMDDYYYY\'))

GROUP BY first\_name, last\_name, address, phone, diagnosis

**Uploading**

This section allows radiologists to upload new records to the radiology\_record table. It also allows radiologists to upload images to any radiology\_record that is currently in the database.

**File:** UploadPage.php

**Description:** This file is the home page for uploading records. It has two forms, one of which is used to upload a new radiology\_record. The other form is used to select an image to upload to an existing radiology\_record.

**File:** CheckRadiologyForm.php

**Description:** Checks the variables set by the radiology\_record form on UploadPage.php. It checks if the mandatory fields are set. It also checks that all the fields are valid.

**Functions:**

* **checkRecordID($recordID):**
  + Checks the database to see if the entered record\_id already exists. If it does, then it returns an error message stating that the record\_id must be unique.
  + **Query:**
    - "SELECT record\_id FROM radiology\_record WHERE record\_id = " . $recordID;
* **buildTestDateQuery():**
  + Builds a query to insert a radiology\_record where the test\_date field is entered but the Prescribe\_date field has not been entered.
  + **Query:**
    - INSERT INTO radiology\_record (record\_id, patient\_id, doctor\_id, radiologist\_id' .

', test\_type, diagnosis, description, prescribing\_date, test\_date) VALUES (' .

$\_POST['record\_id'] . ' ,' . $\_POST['patient\_id'] . ',' . $\_POST['doctor\_id'] .

' ,' . $\_POST['radiologist\_id'] . ', \'' . $\_POST['test\_type'] . '\' ' . ',\''.

$\_POST['diagnosis'] . '\', ' . '\'' . $\_POST['description'] . '\', TO\_DATE(\'' .

$testYear . $testMonth . $testDay . '\', \'YYYYMMDD\'))';

* + **Returns:** sql query string.
* **buildPrescribeDateQuery():**
  + Builds a query to insert a radiology\_record where the prescribe\_date field is entered but the test\_date field has not been entered.
  + **Query:**
    - 'INSERT INTO radiology\_record (record\_id, patient\_id, doctor\_id, radiologist\_id' . ', test\_type, diagnosis, description, prescribing\_date, test\_date) VALUES (' . $\_POST['record\_id'] . ' ,' . $\_POST['patient\_id'] . ',' . $\_POST['doctor\_id'] . ' ,' . $\_POST['radiologist\_id'] . ', \'' . $\_POST['test\_type'] . '\' ' . ',\''. $\_POST['diagnosis'] . '\', ' . '\'' . $\_POST['description'] . '\' TO\_DATE(\'' . $prscYear . $prscMonth . $prscDay .'\', \'YYYYMMDD\'))';
  + **Returns:** sql query string.
* **buildFullDateQuery():**
  + Builds a query to insert a radiology\_record where both the prescribe\_date and test\_date fields have been entered.
  + **Query:**
    - 'INSERT INTO radiology\_record (record\_id, patient\_id, doctor\_id, radiologist\_id' . ', test\_type, diagnosis, description, prescribing\_date, test\_date) VALUES (' . $\_POST['record\_id'] . ' ,' . $\_POST['patient\_id'] . ',' . $\_POST['doctor\_id'] . ' ,' . $\_POST['radiologist\_id'] . ', \'' . $\_POST['test\_type'] . '\' ' . ',\''. $\_POST['diagnosis'] . '\', ' . '\'' . $\_POST['description'] . '\', TO\_DATE(\'' . $prscYear . $prscMonth . $prscDay .'\', \'YYYYMMDD\'), TO\_DATE(\'' . $testYear . $testMonth . $testDay . '\', \'YYYYMMDD\'))';
  + **Returns:** sql query string.
* **buildNoDateQuery():**
  + Builds a query to insert a radiology\_record where both the prescribe\_date and test\_date fields have not been entered.
  + **Query:**
    - "INSERT INTO radiology\_record (record\_id, patient\_id, doctor\_id, radiologist\_id" . ", test\_type, diagnosis, description) VALUES (" . $\_POST['record\_id'] . " ," . $\_POST['patient\_id'] . "," . $\_POST['doctor\_id'] . " ," . $\_POST['radiologist\_id'] . ", '" . $\_POST['test\_type'] . "' " . ",'". $\_POST['diagnosis'] . "', " . "'" . $\_POST['description'] . "')";
  + **Returns:** sql query string.
* **insertRadiologyRecord($sql):**
  + Takes the sql query, $sql, that was generated by the above methods and executes it.
* **checkPrscDate():**
  + Checks that all the parts of the prescribe\_date are entered. Only is called if at least one of the parts has been entered.
  + **Returns:** True if all parts of the date has been filled, else false.
* **checkTestDate():**
  + Checks that all the parts of the Test\_date are entered. Only is called if at least one of the parts has been entered.
  + **Returns:** True if all parts of the date has been filled, else false.

**File:** UploadImage.php

**Description:** This php script is called from the upload image form on UploadPage.php. UploadImage.php verifies the file type of the image and that a record was chosen. Upon confirmation a unique image\_id is generated and then the image is resized to thumbnail and ‘regular\_size’. Then all the images are inserted into the pacs\_images table in the database.

**Functions:**

* **generateImageID():**
  + randomly generates an integer that is used as an id for the images being inserted into the pacs\_images table. Id’s are continuously generated until it is unique.
  + **Queries:**
    - "SELECT image\_id FROM pacs\_images WHERE image\_id = " . $id;
  + **Return:** image\_id
* **createThumbnail():**
  + Checks the file type and creates a string that is the file location for the newly generated thumbnail. It then calls the smart\_resize\_image() to resize the image to 100 x 100 thumbnail.
  + **Returns:** Thumbnail image
* **createRegularSize():**
  + Checks the file type and creates a string that is the file location for the newly generated thumbnail. It then calls the smart\_resize\_image() to resize the image to a slightly smaller ‘regular\_size’ image.
  + **Returns:** regular\_size image
* **buildQuery():** 
  + Inserts the 3 images (2 generated ones) and the image\_id into the pacs\_images table in the database.
  + **Query:**
    - 'insert into pacs\_images (record\_id, image\_id, thumbnail, regular\_size, full\_size) ' . 'values(:recordid, :imageid, empty\_blob(), empty\_blob(), ' . 'empty\_blob()) returning thumbnail, regular\_size, ' . 'full\_size into :thumbdata, :blobdata, :fullsize'

**File:** smart\_resize\_image.function.php

**Authors:** Nimrod007, maxim, gedrox, mamamou, Henridv, tchalvak, YuzuruS, XjSv, nunorafaelrocha

**GitHub:** <https://github.com/Nimrod007/PHP_image_resize>

**Description:** single function that resizes an image to the size specified. See GitHub for more indepth description

**License:** None specified.

**Returns:** returns true/false depending on if the image is resized properly

**Search**

This section handles searching the database for lists of relevant radiology records depending on keywords and/or date range. A record is considered valid if it falls withing the date range and it contains at least one of the keywords in either its patient name, diagnosis or description field.

**File:** SearchForm.php

**Description**: Contains the code to create the form required in order to input the keyword and date range for the search. Can also select a sort type.

**File:** Search.php

**Description**: The main entry point for searching. Gathers information from the form and the session then calls the function to generate and return the search.

**File:** SearchModule.php

**Description**: Contains the functions necessary to generate a dynamic search query and return the results.

**Functions:**

* **searchDB($keywords, $start, $end, $order, $userID, $userClass)**: Generates a sql search query based on keywords, date range, and the class of the user.
* **setupSearch($kFlag, $dFlag, $keywords, $start, $end)**: Builds the search query depending on what input exists. Used kFlag to determin if keywords exist, dFlag to determin if a date range exists. The query is built with a base query plus the addition of keyword and/or date range portions.
* **setupOrder($order)**: Returns a string representing the order portion of the query based on the order in the $order variable.
* **Select():** Returns the base select statement. This is the same no matter what type of search we do.
* **keywords($keywords)**: Returns a string representing the keyword search portion of a query
* **dates($start, $end)**: Returns a string representing a date range as specified by the start and end variables.
* **DateAscending():** Returns a string representing order by date ascending date.
* **DateDecending()**: Returns a string representing order by date descending date.
* **Rank()**: Return a string representing order by rank.
* **drawTable($stid)**: Draws the results to the screen in the form of a table.

**Query:**

SELECT r.record\_id, p.full\_name, d.full\_name as doctor\_name, r.full\_name as radiologist\_name, r.test\_type, r.prescribing\_date, r.test\_date, r.diagnosis, r.description

FROM radiology\_record r, persons p, persons d, persons r

WHERE r.patient\_id = p.person\_id AND r.doctor\_id = d.person\_id

AND r.radiologist\_id = r.person\_id

**If we have keywords concatenate:**

AND( (contains(p.first\_name, \'' . $**keywords**. '\', '.'1'. ') > 0)

OR (contains(p.last\_name, \'' . $**keywords**.'\', '.'2'. ') > 0)

OR (contains(r.diagnosis, \'' . $**keywords**. '\', '.'3'. ') > 0)

OR (contains(r.description, \'' . $**keywords**. '\','.'4'. ' ) > 0)

$string .= ')

**If we have date range concatenate:**

AND (r.test\_date BETWEEN

TO\_DATE(\'' . $start . '\', \'MMDDYYYY\')

AND

TO\_DATE(\'' . $end . '\', \'MMDDYYYY\'))

**One of depending on users class**:

1. AND r.doctor\_id = '.$userID
2. AND r.patient\_id = '.$userID
3. AND r.radiologist\_id = '.$userID

**Plus one of:**

1. ORDER BY r.test\_date ASC
2. ORDER BY r.test\_date DESC
3. ORDER BY (RANK() OVER (ORDER BY(6\*(SCORE(1)+SCORE(2)) + 3\*SCORE(3) + SCORE(4)))) DESC

**Data Analysis**

This section handles analyzing number of images based on groupings of any combination of test type, patient name and date range (weekly, monthly, yearly, total).

**File:** DataForm.php

**Description**: Contains the code to create the form required in order to input the required analysis parameters.

**File:** DataAnalysis.php

**Description**: The main entry point for data analysis. Gathers information from the form and the session then calls the function to generate the appropriate analysis.

**File:** DataAnalysisModule.php

**Description**: Contains the functions necessary to generate the proper sql query, execute it and print the returning information to the screen.

**Functions:**

* **printData($patient, $test, $time)**: Generates a sql query based on the input and prints the results to the screen in a table.
* **setupSelect($patient, $test, $time)**: Returns a string with a properly configured select statement portion of the query.
* **setupGroup($patient, $test, $time)**: Returns a string with a properly configured grouping for the query.
* **drawTable($patient, $test, $time,$stid)**: Formats the output table properly and prints the results to the screen.

**Query:**

Query is generated dynamically such that displaying the exact query generated would be tough. An example query for a user looking for the number of images per user, per test\_type, per month would be:

SELECT full\_name, test\_type, to\_char(test\_date, 'MON’), SUM(num\_images)

FROM FACTS

GROUP BY full\_name, test\_type, to\_char(test\_date, 'MON’)