Blaine Lewis
Dayna Lacoursiere
CMPUT 391

Radiology Information System

File Structure:

- web_docs
  - images
  - style
  - misc
  - views
    - lists
    - forms
    - templates
  - models
  - controllers

Images:

Contains a series of images for styling, no dynamic ones. Examples include the delete and edit buttons.

Style:

Contains one CSS file main.css which styles everything.

Views:

Views are a series of php files which display data. They are a mix of PHP and HTML. Some take parameters and these are documented at the top of each file.

Lists:

Contains a view that's responsibility is to list information.

Forms:

A certain type of view for inserting and editing data.

Templates:

A series of views that act as templates for all others. For example, the header displays the top part of all pages.

Models:

Models represent a table in the database and provide means to manipulate them.

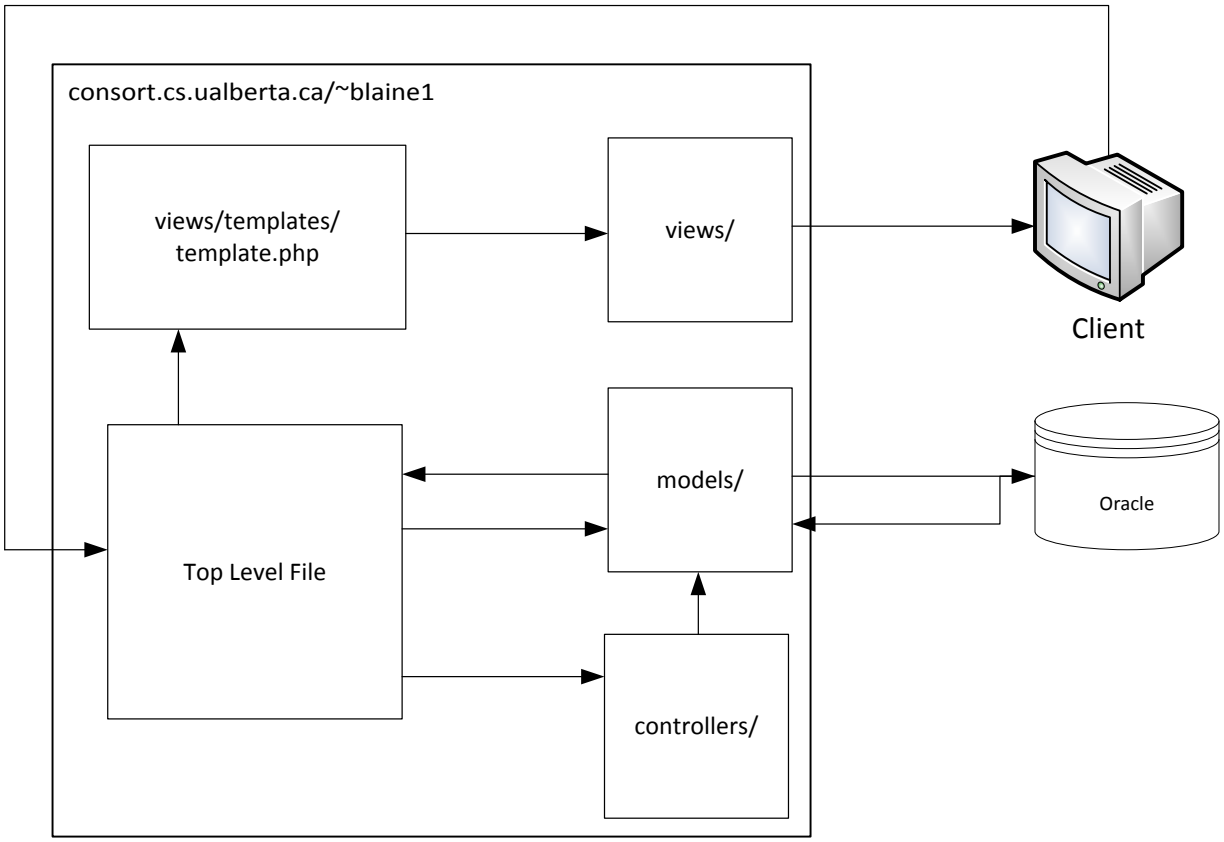
Controllers:

Validate and apply business logic to data then pass it to the models.

Top Level Files:

Files in web_docs not under a folder act as controllers that cannot be generalized anymore. They are directly accessed by the client and are navigable.

Blaine Lewis
Dayna Lacoursiere
CMPUT 391



Seen above is a high level diagram of how a request is processed. The client makes a request to a top level file. If the user submitted data the data is validated in a controller and applied to a model. If the data was valid the top level file calls the model in order to push the data to oracle. If the user requested data then the top level file simply asks the model for the data and the model accesses oracle and returns the requested data. Whether the user requested or submitted data, the top level file passes the results to a view which is then displayed to the user.

An example flow would be user.php. The client sends POST data to user.php. User.php calls controllers/user.php with a User model which validates the POST data and puts the data into the model. The controller returns that the data was valid and tells the User model to submit it. The User model determines the data is new and inserts it into oracle. It returns that the insertion was a failure because an ID was duplicated and the top level file user.php records this message and passes the /views/forms/user.php view to the template view which displays the given view.

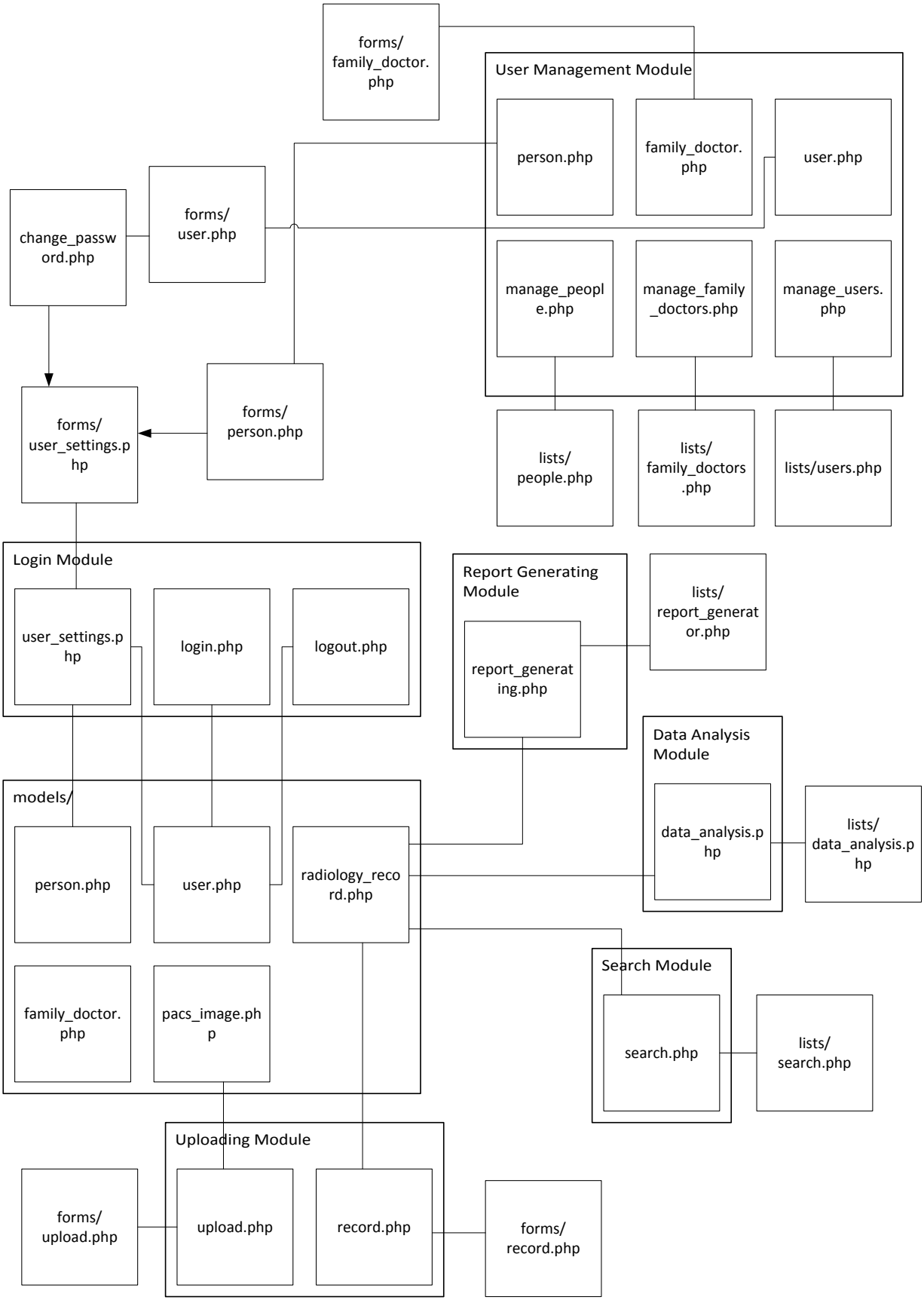
Program Structure:

The overall system is structured using a loose interpretation of MVC. A view is simply a PHP file with HTML that places parameters into a template. Models directly correlate to rows/tables in the database and also allow mass querying from the database. Controllers apply business logic to data supplied by views. They simply retrieve, validate and apply data.

Blaine Lewis
Dayna Lacoursiere
CMPUT 391

Shown on the next page is a diagram showing relationships between files. To reduce complexity the connections between the files in the user management module and their corresponding models are omitted. But person and manage_people go to the Person model and so on. Anything not in a container is a view and can be found in views/. Another omission was the top level file pacs_images.php which is used to display images after retrieval from the database.

Blaine Lewis
Dayna Lacoursiere
CMPUT 391



Blaine Lewis
Dayna Lacoursiere
CMPUT 391

Modules:

Login Module:

The login module involves three major pages. It uses login.php, logout.php and user_settings.php

login.php

Uses the User model to login using the following simple query:

```
SELECT users.user_name
FROM users
WHERE users.user_name = :user_name AND users.password = :password
```

User logins are managed using a session variable and managed with the User model.

logout.php

Removes the session variable denoting user login and redirects to login.php

user_settings.php

uses a simple update query to change personal details. The query is stored in the Person model and is as follows:

```
UPDATE persons
SET
        first_name = :first_name,
        last_name = :last_name,
        address = :address,
        email = :email,
        phone = :phone
WHERE persons.person_id = :person_id
```

It also contains a field to change the password. The query for that is in the User model.

User Management Module:

User management uses a series of pages one for each table it allows editing. The pages are manage_users.php, manage_people.php and manage_family_doctors.php. Each has a corresponding view in views/lists for displaying all the different entries and providing links to edit/delete as required. They all use a simple SELECT query. In the users query there is a join in order to display classes in a nice readable format:

```
SELECT users.user_name, users.person_id, classes.class_name, users.date_registered

FROM users JOIN

        classes ON classes.class_id = users.class
```

To create or edit a record there is a link to user.php, person.php and family_doctor.php respectively. Each of these take an ID as a GET parameter. The name of said ID is dependent on what is being edited. If no ID is sent then it is assumed that a new instance is being created. Each of these files uses either a simple UPDATE or INSERT query. If the row is being edited then a SELECT query is first run to prefill the details.

```
INSERT INTO users
        (person_id, user_name, class, password, date_registered)
VALUES (:person_id, :user_name, :class, :password, SYSDATE)


UPDATE users
SET     user_name = :user_name,
        class = :class
WHERE users.user_name = :old_user_name


SELECT users.user_name, users.person_id, users.class, users.date_registered
FROM users
WHERE users.user_name = :user_name
```

Data validation is applied in the corresponding controller for each model (controllers/user.php, controllers/person.php etc.). Example validations include ensuring the data is of the right length, no duplicate IDs are inserted as well as ensuring that a class for a username is valid.

Report Generating Module:

This module makes use of only a single page found at report_generating.php. It uses a list view to display the data it returns. Its query is built on the fly in order to allow for optional parameters and is stored in the RadiologyRecord model. For example, if a user wants to search for all records after a certain day with any diagnosis they could do so. If all fields are filled out it will look like this:

```
SELECT p.first_name, p.address, p.phone, r.test_date
FROM radiology_record r, persons p
WHERE p.person_id = r.patient_id AND
        r.diagnosis = :diagnosis AND
         r.test_date >= TO_DATE(:start_date) AND
        r.test_date <= TO_DATE(:end_date)
```

Blaine Lewis
Dayna Lacoursiere
CMPUT 391

Uploading Module:

Two pages are part of the uploading module: record.php and upload.php. Both are form style views. Record.php modifies a RadiologyRecord model and inserts it into the database after using controllers/radiology_record.php to validate it. It uses the following query for insertion:

> *INSERT INTO radiology_record*
>
> *(patient_id, doctor_id, radiologist_id, test_type, prescribing_date, test_date, diagnosis, description)*
>
> *VALUES (:patient_id, :doctor_id, :radiologist_id, :test_type, TO_DATE(:prescribing_date, 'YYYY-MM-DD'), TO_DATE(:test_date, 'YYYY-MM-DD'), :diagnosis, :description)*

Upload.php uses a PACSImage model and the controllers/pacs_image.php file for validation. The PACSImage model has the special role of resizing images as well. We use gd to resize all images. In order to place the images in the database we first put empty_blobs into the database and then use the returned descriptors to fill the blobs

> *INSERT INTO pacs_images (pacs_images.record_id, pacs_images.thumbnail, pacs_images.regular_size, pacs_images.full_size) VALUES (:record_id, EMPTY_BLOB(), EMPTY_BLOB(), EMPTY_BLOB())*
>
> *RETURNING pacs_images.thumbnail, pacs_images.regular_size, pacs_images.full_size INTO :thumbnail, :regular_size, :full_size*

Search Module:

The search module uses the search.php view and no controllers. It is accessed through the top level file search.php. It also uses the most complicated query. In order to order the results we calculate the ranking into a column and order by it. The ranking is calculated using the corresponding SCORE from each CONTAINS. LISTAGG was used in order to get all thumbnails into one column. We then process them serverside to create URLs to them. Because the query is populated at runtime it changes often. Below is the query ordered by relevance using a search term with no start or end date.

```
SELECT radiology_record.record_id,
    6*(SCORE(3) + SCORE(4)) + 3*SCORE(1) + SCORE(2) myrank,
    image_agg.images,
    radiologist.first_name radiologist_first_name, radiologist.last_name radiologist_last_name,
radiologist.person_id radiologist_id,
    doctor.first_name doctor_first_name, doctor.last_name doctor_last_name, doctor.person_id
doctor_id,
    patient.first_name patient_first_name, patient.last_name patient_last_name, patient.person_id
patient_id,
    radiology_record.test_type,
    radiology_record.test_date,
    radiology_record.prescribing_date,
    radiology_record.diagnosis,
    radiology_record.description
```

```
FROM radiology_record JOIN
persons doctor ON radiology_record.doctor_id = doctor.person_id JOIN
persons radiologist ON radiology_record.doctor_id = radiologist.person_id JOIN
persons patient ON radiology_record.patient_id = patient.person_id JOIN
        (SELECT LISTAGG(pacs_images.image_id, ',') WITHIN GROUP (ORDER BY pacs_images.image_id)
images,
                        pacs_images.record_id
                FROM pacs_images
                GROUP BY pacs_images.record_id
        ) image_agg ON image_agg.record_id = radiology_record.record_id
WHERE
        (CONTAINS(radiology_record.diagnosis, :diagnosis, 1) > 0 OR
          CONTAINS(radiology_record.description, :description, 2) > 0 OR
          CONTAINS(patient.first_name, :first_name, 3) > 0 OR
          CONTAINS(patient.last_name, :last_name, 4) > 0)
AND :doctor_id IN (SELECT doctor_id FROM family_doctor WHERE family_doctor.patient_id =
radiology_record.patient_id)
ORDER BY myrank DESC
```

Data Analysis Module:

The data analysis module can be accessed via data_analysis.php. It uses a query that is populated at runtime depending on what columns are needed. Below is a populated version of aggregating at the week level across all columns.

```
SELECT test_type, patient_id, to_char(to_date(test_date), 'fmWW, YYYY'), COUNT(*)

FROM radiology_record JOIN

        pacs_images ON radiology_record.record_id = pacs_images.record_id

GROUP BY ROLLUP (test_type, patient_id, to_char(to_date(test_date), 'fmWW, YYYY'))

ORDER BY COUNT(*) DESC
```