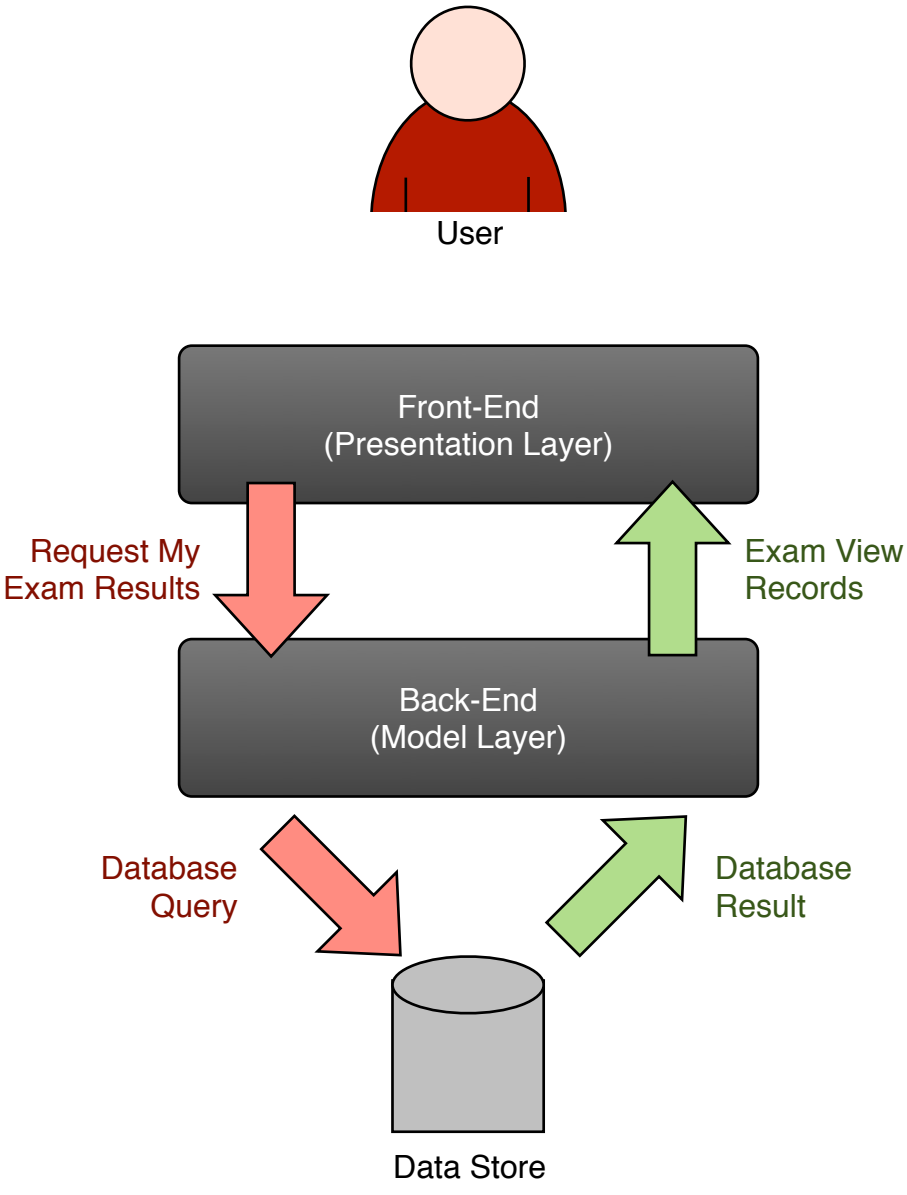
Getting Started with Exam View

Exam View is a multi-tier system for pulling exam results from a student record system. There are different back-end plugins for connecting to MySQL, Oracle (for UNITe) and others. There are different front-ends for displaying results on a simple page, and a block for the Moodle VLE.

System Overview

This is a diagram of the system's structure:



Description of System Overview

When user visits Exam View, the front-end requests exam results from the back-end, passing the student's ID in. The back-end queries the data store (i.e. the student record system and gets back a list of results.

The back-end reads these exam results and translates them to Exam View's own record format. These are then passed back to the front-end for display to the user.

Installing Exam View

The easiest way to configure Exam View is to start from the data layer and work up.

Configuring the Data Layer

You need to make sure your Exam View installation can connect to your data store. This may involve opening up your data store so a remote server can connect to it.

If you're connecting to a database server (e.g. Oracle or MySQL) it's recommended that you set up a new user just for Exam View. Given the sensitive nature of the data stored in your student records system, you should set it up for read only access (i.e. SELECT), and only give it access to the databases and tables you require.

For instructions in setting these things up, see the documentation for your database or student record system.

Configuring the Model Layer

This section assumes you have a data store that is compatible with Exam View. If not, there is a guide to writing one later in this document.

Open the relevant model layer for your data layer. Near the top, you'll see a function called **fetch_data**. At the start of this function you'll find settings you can change to connect to your database server. For example, in the MySQL model layer:

```
$mysql_server       = "localhost";
$mysql_username     = "examview";
$mysql_password     = "mypassword";
$mysql_database     = "examview";
```

Just below this, you'll find a place you can type the database query that will be used to return the results. This database query should, when given a student ID (when for the WHERE clause), return every exam result for this student.

In most database systems, prepared statements mean you can use a placeholder for the student ID. For MySQL, you can use a **?**. For Oracle, Exam View uses **:studentid**. The provided back-ends will automatically insert the student ID in this place. These examples are for MySQL and Oracle:

```
"SELECT * FROM results WHERE student_id = ? ORDER BY
period DESC, unit_title ASC"

"SELECT * FROM results WHERE student_id = :studentid
ORDER BY period DESC, unit_title ASC"
```

Configuring the Presentation Layer

Your front-end needs to be able to determine which student's details to look up. This is done through a function called **get_student_id**. When passed a username or unique key, this should be able to return a student's ID.

For example, at Glasgow Metropolitan College we use Exam View with our Moodle VLE. Students log in to Moodle with the username **met** followed by their matriculation number (e.g. **met12345678**). All our **get_student_id** function has to do is strip the word 'met' from the beginning of the Moodle username to get the student ID.

```
$contains = strpos($username, "met");

if ($contains === 0) {
    return(substr($username, 3));
} else {
    return($username);
}
```

After this, all you need to do is tell your front-end which back-end it should use. To do this, open your chosen **frontend-x.php** file. Near the top of the file, you'll see two lines:

```
include 'classes.php';
include 'backend-mysql.php';
```

All you need to do is change **backend-mysql.php** to the name of the back-end you wish to use.

Writing your Own Model Layer

If you're reading this, chances are you need to write a new model layer to get your student record system working with Exam View. It's actually a very easy task to accomplish, and you can look at the included MySQL and Oracle model layers for inspiration.

To begin writing a new back-end, you'll some way of connecting to your data store. Connecting to a relational database over the network is the best way to do this, because querying and sorting data from relational databases is very efficient. You could also use SOAP, JSON or plain XML to fetch your data.

It is recommended that you filter and sort data at the database or data store level. For example, downloading a list of all student records, then manually searching for matches in Exam View's model layer would place a great load on the Exam View server. This would stop Exam View from scaling well. More than few users trying to access Exam View at the same time could take down the server, since this is a very inefficient process. By contrast, searching, filtering and sorting data from a relational database is very efficient.

The model layer has a few functions in it.

get_student_records takes in a student ID, and passes back the students result in Exam View format. This is the function that is called by the front-end, and it essentially acts as a wrapper, calling the other functions (below) as necessary to return the data.

fetch_data is the function that actually connects to the database. It takes in a student ID and gets the student data from the database. It calls the **process_record** function on each record it gets back, which converts the record into Exam View's internal data format.

process_record takes in a record, in whatever format **fetch_data** returns (for example, an associative array using mysql_fetch_assoc), and converts it to the Exam View data format.

Sometimes, it may not be possible to get the records and convert them one at a time, for example when you get data through XML or JSON. In this case, you should write a function called **process_records** that takes in all the data returned and splits it into individual records.

Frequently Asked Questions

Why does Exam View have its own internal data format? Wouldn't it be simpler to just output the results in the back end's natural format?

Every front-end is written to read data in Exam View's data format. Every back-end is written to translate data into Exam View's data format. This approach means that any back-end can be used interchangeably with any front-end.

What are the system requirements for using Exam View?

Beyond the standard response of 'powerful enough to handle all your incoming requests', Exam View was developed on a CentOS 5 server with Apache 2 and PHP 5.2.

Using relational databases, Exam View scales well during peak periods (i.e. on exam result release day). You should, however, consider the burden Exam View could have on your student record system if you are querying it directly.

In addition, you should **always** close your database connections when you are finished using them. The included MySQL and Oracle backends do this and so should yours!

Common Mistakes

Setting Up Oracle

Depending on how your web server is set up, it should usually be able to connect to MySQL straight out of the box. However, connecting to Oracle requires that your PHP installation supports OCI. To get this up and running, you'll need to:

- Download the Oracle OCI8 Instant Client drivers and install them to a location on your Exam View server.
- Download the PHP source code.

- Compile PHP, adding --with-oci8=shared,instantclient,/path/to/oracle when you run ./configure .

Security Advice

Exam View is only as secure as the system it is running on. Due to the sensitive nature of the data stored in student record systems and exam result databases, there are some security measures you may wish to take when setting up Exam View.

- Always set up a new database user just for Exam View. Give it a good password, and keep those login credentials private.
- Make sure your SSH and FTP access to the server are secure. Finding a way into the server and reading Exam View's PHP files (or your backups) is the only way someone will get your database credentials.
- Exam View trusts that the front-end user is who they say they are. Make sure your front-end is secure. For example, the Moodle plugin takes the user's ID from the session. Combined with server-side sessions, this is pretty secure. Never build a front-end that takes the student ID in the query string ($_GET) because people will quickly figure out they can put any student ID in there.