Introduction to Drupal

(Here, we assume that Drupal has been installed in the directory Drupal-6.8, so all paths are relative to that directory. For installation instructions, please refer to the previous lesson[s])

1. What is Drupal?

- Drupal Getting Started Guide [http://drupal.org/getting-started]
- Drupal Installation Guide [http://drupal.org/getting-started/install]
- Installing Apache, MySQL, and PHP on Linux [http://hostlibrary.com/installing_apache_mysql_php_on_linux]
- Download/Install WAMP for Windows [http://www.wampserver.com/en/]

From Wikipedia: "Drupal is a free and open source modular framework and Content Management System (CMS) written in the programming language PHP."

Drupal may be used to publish content without any programming involved, at the simplest level. To do this, access your Drupal installation through the web browser (the URL should be similar to http://localhost/drupal/drupal-6.8/). Log in, and select **Create Content** in the left navigation menu, then select **Page**. Enter a **Title** and some **Body** text, then click **Save**. The content should appear with a green status message informing you that the page has been created – note that the URL of the newly created page should be something like http://localhost/drupal/drupal-6.8/?q=node/1.

As of now, there is no indication to the casual user that this page even exists. You can display it on the front page by selecting **Administer->Content Management->Content**, selecting the newly created page and updating its status with **Promote to front page**. Click on the Drupal logo at the top left hand corner to return to the home (front) page, and you should see that the content has appeared. Alternatively, you might create a **Story** instead, which is automatically displayed on the front page, and moreover has user comments enabled.

To add a link to pages/stories in the left navigation menu, select **Administer->Site Building->Menus->Navigation**, then click **Add Item**. The **Path** is the tailend of the URL of the page you wish to link to, and the other fields should be self-explanatory. Links within pages themselves may be achieved though the basic <A HREF="…"> HTML tag.

Finally, select **Administer->Site Building->Themes** to modify the look and feel of the Drupal site. You might select one of the available ready-made alternative themes for a start. If a different structure is required, the easiest method to achieve it would probably be to search for as similar a theme as possible, then modify it incrementally to suit your needs. The files that define a theme should all be in a single subdirectory within the **themes** folder in your Drupal installation, so creating a new theme in this way would be just a matter of copying the subdirectory and editing those files. If only minor cosmetic changes (e.g. images, colors) are needed, simply edit the files in the **images** folder of the appropriate theme.

It should be evident that building an entire (largely) non-interactive website is possible (and quite convenient) with this knowledge alone. However, Drupal offers much more. We continue by looking at **modules**.

Prepared by Lim Yong San, Gilbert
U047391X

2. Drupal Modules

- Creating modules – a tutorial [http://drupal.org/node/82920]
- Page Handler Include Files [http://drupal.org/node/146172]

Drupal has a modular architecture, which allow different contributors to add to the Drupal core (which is what we get from a fresh installation) independently, without these additions interfering with each other (unless designed to do so). Importantly, modules allow one to design and implement (sub)systems that interact with the database in custom ways – which is the whole point, after all! It is true that this can be done without Drupal, and indeed without any CMS at all.

Part of the justification for using Drupal is that it provides a uniform framework and forces programmers to conform to certain practices and standards (important for future maintenance!), and moreover saves the trouble of reimplementing certain functions (such as keeping track of users and their permissions). The tradeoff is that it may be rather challenging to implement certain stuff within the framework, but often the desired effect is challenging precisely because it is a Bad Idea™.

What defines a module? At the very least, a module is defined by two files – its **.info** and **.module** files. We continue with a hands-on example. Navigate to the **sites\all\modules** directory in your Drupal installation. Create a new subdirectory in that directory named **testmod**, and in this subdirectory create the files **testmod.info** and **testmod.module** with your favorite text editor (Notepad will do).

The .info file contains basic information about the module. Copy this text into the file, and save it.

```
; The semi-colon indicates a comment in the info file.
name = Test Module
description = "This is a test module"
core = 6.x
package = Test Package
```

The .module file is in PHP, and is headed by <?php (though the usual closing ?> is not needed). Again, copy this text into the file, and save it.

```
<?php

function testmod_menu() {

  $items = array();

// Add link to menu, link to default section callback
  $items['testmod'] = array(
      'title' => t("Test Module Link"),
      'page callback' => 'testmod_output',
      'file' => 'testmod.module',
      'access callback' => 'user_access',
      'type' => MENU_NORMAL_ITEM,
  );
```


Prepared by Lim Yong San, Gilbert
U047391X

```
  return $items;
}

function testmod_output() {
      $output = "Here be test output.";
      return $output;
}
```

Now select **Administer->Site Building->Modules**. The Test Module should be shown on the page, but its checkbox will be unchecked, signifying that it is not yet enabled. Check the box and save. A **Test Module Link** item should appear in the left navigation menu. Click it, and a page with the text "Here be test output." should appear. Congratulations, you have created your first Drupal module! Wasn't that hard, was it?

Another important file is the **.install** file, which like the .module file should contain PHP code, and is run whenever the module is installed (or uninstalled – note that this is different from enabling and disabling a module. Once installed, a module retains the information in its database tables if disabled, just that it does not appear on the site. If uninstalled, however, that information should be lost if its _uninstall hook is properly coded. The link to uninstall is at the top after selecting **Administer->Site Building->Modules**). Generally, the .install file is used to initialize database fields, and other related functions.

The menu system also supports include files (with .inc extensions), so page handler functions can be delegated to their own files, making things less messy. Add the following code to the testmod_menu() function in testmod.module, and save the file:

```
  $items['testmod/testmod2'] = array(
      'title' => t("Test Module Link Two"),
      'page callback' => 'testmod_output2',
      'file' => 'testmod2.inc',
      'access callback' => 'user_access',
      'type' => MENU_NORMAL_ITEM,
  );
```

Exercise #1: **Implement testmod2.inc and have it display some output (Hint: New menu items may only show up after the local cache is cleared at Administer->Performance->Clear cached data)**

We proceed to introduce the Application Programming Interface (API) that Drupal uses, in particular how it is relevant to module development.

Prepared by Lim Yong San, Gilbert
U047391X

3. The Drupal API

- Drupal Hooks [http://api.drupal.org/api/group/hooks/6]
- Drupal Form API [http://api.drupal.org/api/file/developer/topics/forms_api_reference.html]

You may have noted that the Test Module functions above are of the form testmod_[something]. That [something] is known as a **hook**, and is used to extend Drupal functionality. In the Test Module example above, the menu hook was demonstrated, and a bit of fiddling around will reveal that it affects the navigation links to the module's output.

While intimate knowledge of all the hooks is ideal, a more practical approach might be to modify (and understand!) existing module code rather than try and construct a module from scratch from the API definitions. Most modules will likely not use most of the hooks available, and some (such as menu and form) are used more than others.

There will often be more than one method to achieve an effect in Drupal, though most often it should be best to use a Drupal-specific method. For example, to create a web form, one might just code it in HTML. A better way would be to use Drupal's Form API, and leave the rendering and handling of the forms to Drupal, which facilitates matters since Drupal hooks for validation and handling may be used.

We continue with a live example. Create testmod.install and copy and paste the following code, then save the file:

```
<?php

function testmod_install() {
      drupal_install_schema(testmod);
}

function testmod_uninstall(){
      drupal_uninstall_schema(testmod);
}

function testmod_schema() {
      $schema['testmodtable'] = array(
            'description' => 'Names',
            'fields' => array(
                  'id' => array(
                      'type' => 'int',
                      'unsigned' => TRUE,
                      'not null' => TRUE,
                      'size' => 'big',
                      'description' => t('ID'),
                  ),
                  'name' => array(
                      'type' => 'varchar',
                      'length' => 32,
                      'default' => '',
                      'not null' => TRUE,
                      'description' => t('Name'),
                  ),
```


Prepared by Lim Yong San, Gilbert
U047391X

```
            ),
            'primary key' => array('id'),
      );

      return $schema;
}
```

The _install hook contains code that is executed whenever the module is installed within Drupal. Here, the only thing it does is to install a database schema for the module (with a Drupal-specific function). The schema is defined further on in the install file in the function testmod_schema(), which takes the form of an array $schema. An element in this array defines a single table, which here is 'testmodtable'. The internal structure of the element should be quite self-explanatory.

Save testmod.install, and uninstall and reinstall the Test Module. Now visit your local installation of PhpMyAdmin through your web browser (it should exist as it is a prerequisite for running Drupal) at http://localhost/phpmyadmin/ by default (Note: this URL may be case-sensitive). Access the database that Drupal created for itself (specified when Drupal was installed). There should now be a table within that database called testmodtable. Select this table, and two fields (id and name) should be displayed.

Now that your Test Module has somewhere to save its data, let us create a simple interface to do so. Open testmod.module and add the following lines to what is already in the file, then save the file:

```
function testmod_form1() {

      global $user;

        $form['name'] = array(
            '#type' => 'textfield',
            '#title' => t('Full Name'),
            '#default_value' => '',
            '#maxlength' => 20,
            '#size' => 20,
            '#prefix' => '<BR><BR><B>Uid:</B> '.$user->uid.' <BR>',
            '#description' => 'Name',
        );

      $form['submitbutton'] = array(
            '#type' => 'submit',
            '#name' => 'submit1',
            '#value' => t('Save'),
      );

      return $form;
}
```

Now access the Test Module in Drupal through the left navigation menu. A form should appear. Note that the $user->uid variable contains the user id of the currently logged-in Drupal user, and that all the attributes of individual form elements are prefixed by the hex sign (#).

Prepared by Lim Yong San, Gilbert
U047391X

This code does not actually do anything useful yet, as you may have discovered by clicking the Save button. First, let us allow the module to determine when the form has been submitted. Add the following lines to testmod.module and save it:

```
function testmod_form1_submit() {
      drupal_set_message("Form submitted.");
}
```

Access the Test Module again, and a green "Form Submitted" status message should appear if the Save button is clicked. Therefore, the _submit hook is executed whenever its corresponding form is submitted. We can use it to write the appropriate information to the database. Replace the testmod_form1_submit() function with the following, and save the file:

```
function testmod_form1_submit($form, &$form_state) {

      global $user;

      drupal_set_message("Form submitted.");

      $result = db_query("SELECT * FROM {testmodtable} WHERE id=%d",
$user->uid);
      if ($node = db_fetch_object($result)) {
          db_query("UPDATE {testmodtable} SET name='%s' WHERE id=%d",
$form_state['values']['name'], $user->uid);
      } else {
          db_query("INSERT INTO {testmodtable} (id,name) VALUES
(%d,'%s')", $user->uid, $form_state['values']['name']);
      }

}
```

Also add the following code to the testmod_form1() function, right after the `global $user;` statement:

```
      $result = db_query("SELECT * FROM {testmodtable} WHERE id=%d",
$user->uid);
      while ($node = db_fetch_object($result)) {
          $name = $node->name;
      }
```

And modify the `'#default_value' => ''` statement to the following:

```
'#default_value' => $name,
```

Finally, save the file. Now, our very simple Test Module should work as it is expected to. We can enter a name and save it, and the new value will be reflected in the database. *This is probably the gist of what almost all systems you built will do*, though of course most useful systems will be rather more complex.

Exercise #2: **Use the _validate hook and Drupal form_set_error() function to reject names that are three characters or shorter.**

Prepared by Lim Yong San, Gilbert
U047391X

This is a good time to learn about extending currently existing modules. If no changes to the database are required, development may proceed by making whatever code changes are needed in the .module or .inc file.

As your modules grow, it may be wise for functions shared by many individual .inc files within a module to be consolidated within a single .php "library", which can then be required at the head of each individual .inc file. Functions shared by related modules may be placed within a "parent" module which is then made the prerequisite of all "child" modules. Using the PHP define() function to set as constants things that should nearly never be changed (such as database table names) is usually a good idea too.

Exercise #3: Have your testmod.module and testmod2.inc both require the same PHP file.

There will be times when changes to the database schema are required. Let us add an additional field to testmod.module (in the appropriate location…):

```
        $form['age'] = array(
            '#type' => 'select',
            '#title' => t('Age'),
            '#options' => array('1'=>'< 18','2'=>'18-65','3'=>'> 65'),
            '#default_value' => '',
        );
```

The select element should appear when Test Module is accessed in your browser once the file is saved. However, there is currently no field in the testmodtable database table to store the age information. Open testmod.install and add the following code to the end, and save the file:

```
function testmod_update_1000() {
      $ret = array();
      db_add_field($ret, 'testmodtable', 'age', array('type' =>
'int','size' => 'small', 'default' => 1, 'unsigned' => TRUE,
'description' => t('Age')));
      return $ret;
}
```

Access **Administer->Site Building->Modules** and click the link for update.php. Drupal will throw up a page informing you on database updates, and after the Continue button is clicked, a list of all modules with their available updates should appear. Under testmod, the version "1000" should be selected. If not, select it. Click Update, and check the testmodtable in phpMyAdmin. A new field should have been added. This process (as opposed to manually editing the database outside of Drupal) allows updates of a module by other users who already have it installed (such as your fellow developers) to be as automatic as possible.

Exercise #4: Write code to have the Age information saved and retrieved from the database too, as has been demonstrated with the Name.

Exercise #5: Experiment with drupal_change_field() to set the default for Age to 2.

Exercise #6: Experiment with other types of form elements like checkboxes, radio buttons, etc. Implement more fields to make a registration page.

Prepared by Lim Yong San, Gilbert
U047391X

4. Drupal and JavaScript

- JavaScript in Drupal [http://drupal.org/node/121997]
- JavaScript, AJAX, AHAH API [http://drupal.org/node/205296]

JavaScript is useful to implement certain checks and actions on a webpage withot the overhead of reloading the entire page. Short JavaScript snippets *may* possibly be hardcoded into the page source in practice, though Drupal has its own JavaScript API.

In either case, it is especially important to verify that your JavaScript works as intended in the major browsers (Internet Explorer and Firefox), in particular the versions that clients will be using (not everybody may be using the latest browsers!). Additionally, note that JavaScript might possibly be disabled, so this has to be taken into consideration when making certain (critical) functions JavaScript-dependent.

Exercise #7: Implement a JavaScript function to ask the user to confirm that he wants to submit the form when the Submit button for the Test Module is clicked (Hint: JavaScript has a confirm() function). Verify that the JavaScript code is cross-browser compatible.

5. Real-life Examples

If you have successfully completed all the exercises thus far, you should already be able to build a simple system that receives and stores a logged-in user's own information in Drupal. Congratulations!

However, the organizations that you will be helping will probably have slightly more complicated needs. In particular, the users will likely have different roles, and thus permissions – managers may be able to access some functions, but not ordinary users. This may be implemented with Drupal's own permissions system (**Administer->User management**), by setting appropriate user roles. Individual module pages may then be restricted with access arguments.

This is but one of the many possible requirements, though, and it is certainly impossible to cover everything that might be needed in the process of development. In these cases, Google (or your favorite search engine) and the Drupal site (http://drupal.org) are valuable resources.

Best of luck!

Prepared by Lim Yong San, Gilbert
U047391X