# COMP2001 : Information Management & Retrieval

## INTRODUCTION

This document talks you through the implementation details for creating a PHP application that derives its data from a MySQL database application. This follows the Application Fact Sheet planning document provided in Unit 9 and concentrates only on the first user story of

> "As a student I wish to alert the lecturer that I need some help"

The assumption is that you already have the database elements created in the MySQL database. These elements will contain tables, triggers and stored procedures.

There are accompanying podcasts to be found to guide you in developing your prototype application. These can be found on the DLE in the podcast section. Additionally, the code is part of my public GitHub repository ISAD251_2019_PHP.

## APPLICATION

Our application is going to be a prototype proof of concept – this means it is NOT a full application but an exploration of how some aspects might work. We will act as if we were working in sprints, short iterative bursts of activity that will drive us to the end goal.

Each of the tasks are described at a reasonably high level. Use the "How to.. " guides provided on the DLE to help support you with each of the tasks. The full application details are given in the **Application Fact Sheet**.

## IMPLEMENTATION

PHP

Task 1:

Create a new PHP application for the project as you have done in earlier practical sessions. See workbook 1 in Unit 4 for a suggestion on how to create your folder structure. Create a header.php and footer.php page to display the layout as per the storyboard diagram. Use these as includes within the index.php page and ensure you have Bootstrap styling throughout to apply the storyboard plan.

Create a RaiseRequest.php page which holds the web form components.

See ***How do I..Developing Web Applications using PHP*** podcast for more information.

What do I need to know?
How to

- Combine PHP and HTML
- Apply Bootstrap styling classes to achieve the look required
- How to create the folders required to apply MVC pattern
- How to include files for reuse

Task 2:

Using the class diagram from the application fact sheet, create the request, modules and lecturer classes (your repository classes) and create the DbContext class (your unit of work class).

Note: With PHP you do not have helper methods such as get and set. You have to write the code for these yourself. Therefore you could have the example given below for the Modules class. As you can see the properties are declared as private but there is a public method to output them when required.

```php
<?php

class Module
{
    private $code;
    private $title;
    private $id;


    public function __construct($Code, $Title, $Id)
    {
         $this->code = $Code;
        $this->title = $Title;
        $this->id = $Id;
    }

    public function Code()
    {
        return $this->code;
    }
    public function Title()
    {
        return $this->title;
    }
    public function Id()
    {
        return $this->id;
    }
}
```

Refer back to Unit 5 for how to create the classes required for the pattern. See podcast ***How to .. PHP classes to interact with the database*** for more information.

What do I need to know?
How to

- Create a PHP class with properties and methods
- Create a class to represent repository and unit of work pattern
- Use PDO to interact with the database.

Task 3:
Inside the DbContext class place PDO methods so that data can be extracted from the database. Include the repository classes (such as Modules, Request etc). Create a function that provides an array of objects relevant to the class – see the Modules example method below:

```php
public function Modules()
{
    $sql = "SELECT * FROM `sh_modules`";

    $statement = $this->connection->prepare($sql);
    $statement->execute();
    $resultSet = $statement->fetchAll( fetch_style: PDO::FETCH_ASSOC);

    $modules = [];

    if($resultSet)
    {
        foreach($resultSet as $row)
        {
            $module = new Module($row['Code'], $row['Title'], $row['ModuleID']);
            $modules[] = $module;
        }
    }
    return $modules;
}
```

Note here that the back-ticks are needed around the table name. The Modules class is used to output an array of modules when the public function Modules is called.

Inside your public folder create a request.php where the HTML and layout conforms to the template for the application and applies the plan from the storyboard. Link the page to your

DbContext class created above. Code the page so that the Module drop down box is dynamically populated from the relevant table from the database. See the code below and the ***How to.. Drop down list from database PHP*** podcast for more information.

```
<select class="custom-select mr-sm-2 form-control" name="ModuleID">
    <option value="">--Select Module Code--</option>
    <?php
        $optionString = "";

        $db = new DbContext();
        $modules = $db->Modules();

        if($modules)
        {
            foreach($modules as $module)
            {
                $optionString .= "<option value=".$module->Id().">".$module->Code()."</option>";
            }
        }
        echo $optionString;
    ?>
</select>
```

Task 4:
Your next task is to use the stored procedure you created for Enter_Request to take the input from the request form and insert into the database. Refer to podcast ***How to .. Call a Stored Procedure from PHP*** for further information. The step by step details are provided below:

1. Ensure the form tag refers back to itslef by using the $_SERVER['PHP_SELF'] variable inside the action tag.

```
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
```

2. At the top of the RaiseRequest.php check the $_POST variables exist before creating a new request object.
3. Create a method in the DbContext class that calls the Enter_Request stored procedure. Have the method take in a request parameter. See the code below.
4. Create the SQL to call the stored procedure with it's parameters, bind the parameters and execute the SQL. Examine the results.

```php
public function Enter_Request($request)
{
    $sql = "CALL EnterRequest(:Name, :Room, :Row, :Seat, :Problem, :ModuleID)";
    $statement = $this->connection->prepare($sql);
    $statement->bindParam( parameter: ':Name',  &variable: $request->Name(),  data_type: PDO::PARAM_STR);
    $statement->bindParam( parameter: ':Room',  &variable: $request->Room(),  data_type: PDO::PARAM_STR);
    $statement->bindParam( parameter: ':Row',  &variable: $request->Row(),  data_type: PDO::PARAM_INT);
    $statement->bindParam( parameter: ':Seat',  &variable: $request->Seat(),  data_type: PDO::PARAM_INT);
    $statement->bindParam( parameter: ':Problem',  &variable: $request->Problem(),  data_type: PDO::PARAM_STR);
    $statement->bindParam( parameter: ':ModuleID',  &variable: $request->ModuleId(),  data_type: PDO::PARAM_INT);

    $return = $statement->execute();
    return $return;
}
```

5. Ensure you call the new method from the RaiseRequest page.
6. Debug and test.

Task 5:
The next part of the application is to call out to the Hue lights RESTful API to turn the light on. Inside the Request Controller create a new method that does not return anything called Alert. Pass in as a parameter the enter_request object.

Within this method you need to do the following:

1. Set the URI – copy the code below (this one is not an image). This URI is the IP address of the Hue lights in SMB109 and provides the developer key within the URI.

   $URI = "http://192.168.0.50/api/stlaB2I6VZ8O80Qepc-1xfmLrHgyTFvB9IGupaQz/lights";

2. Create an array of integers to hold the Hue values. Colours are as follows
   a. Red = 0 or 65535,
   b. Orange = 5000
   c. Yellow = 12750
   d. Green = 25500
   e. Blue = 46920

```php
//colours are Red 0 or 65535, Orange = 10?, yellow = 12750, Green = 25500, Blue = 46920
$lightsArray = array(0, 5000, 12750, 25500, 46920);
```

3. Use the Row property of the incoming request object to append to the URI for the lights so that you have the light ID and /state afterwards.

```php
//Get the row ID plus the seat ID to map to the lights
$URI .= $request->Row()."/state";
$seatNumber = (int)$request->Seat()-1;
```

4. Create a variable to hold the JSON content to be sent to the URI. Set the “on” property to be true and the “hue” property to hold the appropriate number from the lightsArray based on the Seat Number. Use the json_encode method to turn this into JSON data

```
$content = array("on"=>true, "hue"=>$lightsArray[$seatNumber]);

$jsonData = json_encode($content);
```

5. Use the CURL library as set out below to initialise the library based on the URI, set the HTTP verb to PUT, to pass in the JSON data created in step 4 and execute the library.

```
//Initialise the CURL library
$httpClient = curl_init($URI);
 //Set the HTTP verb
   curl_setopt($httpClient, CURLOPT_CUSTOMREQUEST, "PUT");
curl_setopt($httpClient, CURLOPT_RETURNTRANSFER, true);

//Tell it the content type - or the webservice does not know what to do with it
curl_setopt($httpClient, CURLOPT_HTTPHEADER, array(
   'Content-Type: application/json',
   'Content-Length: ' . strlen($jsonData)));

 //Now give it the data to post in
curl_setopt($httpClient, CURLOPT_POSTFIELDS, $jsonData);

$response = curl_exec($httpClient);

//The HTTP code is important to be able to handle any errors and responses.
$httpCode = curl_getinfo($httpClient, CURLINFO_HTTP_CODE);
curl_close($httpClient);
```

6. Call the Alert function from your RaiseRequest page on successful insert.

Test your application. Check the page in the W3 validator and run the page in two separate browsers.

What do I need to know?
How to

- Use the W3 validator
- Check accessibility

## WRAP UP

The whole of this exercise has shown you how to create a PHP application from start to finish. It shows you how to link your interface with your database layer using the classes in the middle. It finally showed you how to use a RESTful API and the HTTP verb PUT.