

**Management Development Institute of Singapore**

# Assignment 1: Dynamic Internet Technology

**Program Title: *Bachelors of Science (Hons) in Computer& Network technology***

**Module Title: *Dynamic Internet Technology***

**Batch Code: *BNTD2 1608A***

===============================================================

|  |  |
| --- | --- |
| **Name** | Nguyen Duc Liem |
| **MDIS Student FIN** | G1487682R |
| **University Student ID** | 16041325 |
| **Module Convenor** | Mr Kevin Shiflet |
| **Student Email** | Cyberliem.civil@gmail.com |
| **Course Code** | EN0273 |

Contents

[Assignment 1: Dynamic Internet Technology 1](#_Toc468110283)

[**I.** **Introduction and concept** 3](#_Toc468110284)

[**II.** **Website work flow:** 4](#_Toc468110285)

[1. ***General View*** 6](#_Toc468110286)

[2. ***Search*** 8](#_Toc468110287)

[***3.*** ***API*** 9](#_Toc468110288)

[***4.*** ***Database connection: OOP and PHP*** 10](#_Toc468110289)

[***5.*** ***Prevention of SQL injection: PDO*** 11](#_Toc468110290)

[**6.** **Pagination and filtering: Sever-side Memory.** 12](#_Toc468110291)

[***7.*** ***Browse Item*** 14](#_Toc468110292)

[***8.*** ***Login Function*** 14](#_Toc468110293)

[***9.*** ***Edit Item.*** 15](#_Toc468110294)

[10. Credit Page. 18](#_Toc468110295)

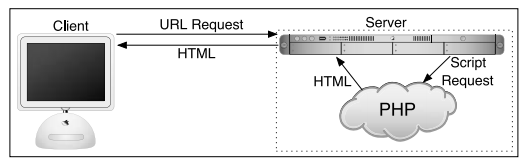
[**II.** **Conclusion** 18](#_Toc468110296)

[Bibliography 19](#_Toc468110297)

## **Introduction and concept**

The word wide web has taken the world of information by storm since its introduction to the general public in 1992 (Connolly, 2000). In its most primitive form, the web as we know it was the HTTP protocols of transferring hypertext from a server to response to any request made by a client. In 1990, the basic components of a working web was introduced by Berners-Lee as follow: the HyperText Transfer Protocol (HTTP), the HyperText Markup Language (HTML), the firs Web browser namely WorldWideWeb, the first HTTP server software namely CERN httpd and the first web server (Lee, 1991).

Moving on toward the modern days of the World Wide Web, there is a need of a dynamic web system, which has a memory to retain the state of client-server, response to different parameters from different request, as well as easier to maintain or upgrade. The original HTML wasn’t be able to handle this, hence the birth of PHP, originally stood for “Personal Home Page”, and now widely known as “Hypertext processor” (Ullman, 2012). The term “Hypertext processor” somewhat describes most of concept behind PHP: Based on user input and the state of client-server, it query the database, processes, renders a HTML response from server to client. Figure 1 shows how PHP works:



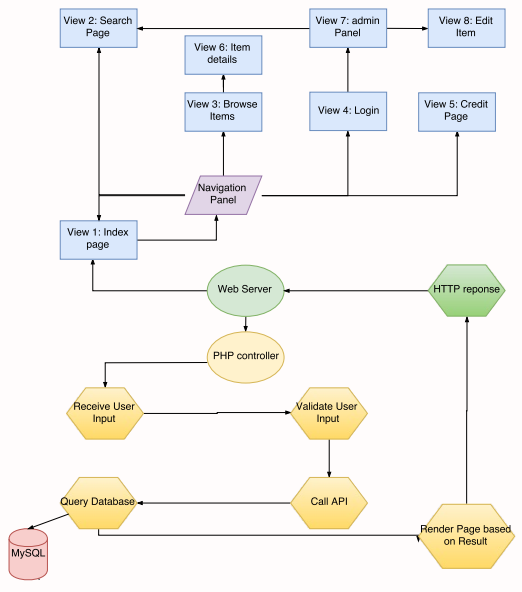
*Figure 1: PHP in a client-server interaction.* (Ullman, 2012)

The final component to this dynamic web project is MySQL. With the growth of data, the need of a relational database management system is inevitable. In adherence to this need, Oracle provide the world with a wonderful and open-source database management, namely MySQL. It can be incorporated with PHP, to securely and systematically organize, query and manage information stored for any website.

Theoretically, all the above languages and framework can run on a most of the server Engine, however, for this particular project, the LAMP (Linux, apache, mysql and php) environment, which is open-source and easy to install as well.

The main purpose of this project is the demonstration of all above concepts, as well as others concept that is relevant to the contemporary internet technology. The object displayed is Tyne-Events, a fictional company which requires its online presence. It mains purpose is to display the events repository, as well as allow user to search, filter for desired event. With authenticated users, it should be possible for them to directly modify the database without having to have advance level of IT expertise. On this request, the website was made by Liem Duc Nguyen, an IT student on his first year of Computer Science Degree with University of Northumbria.

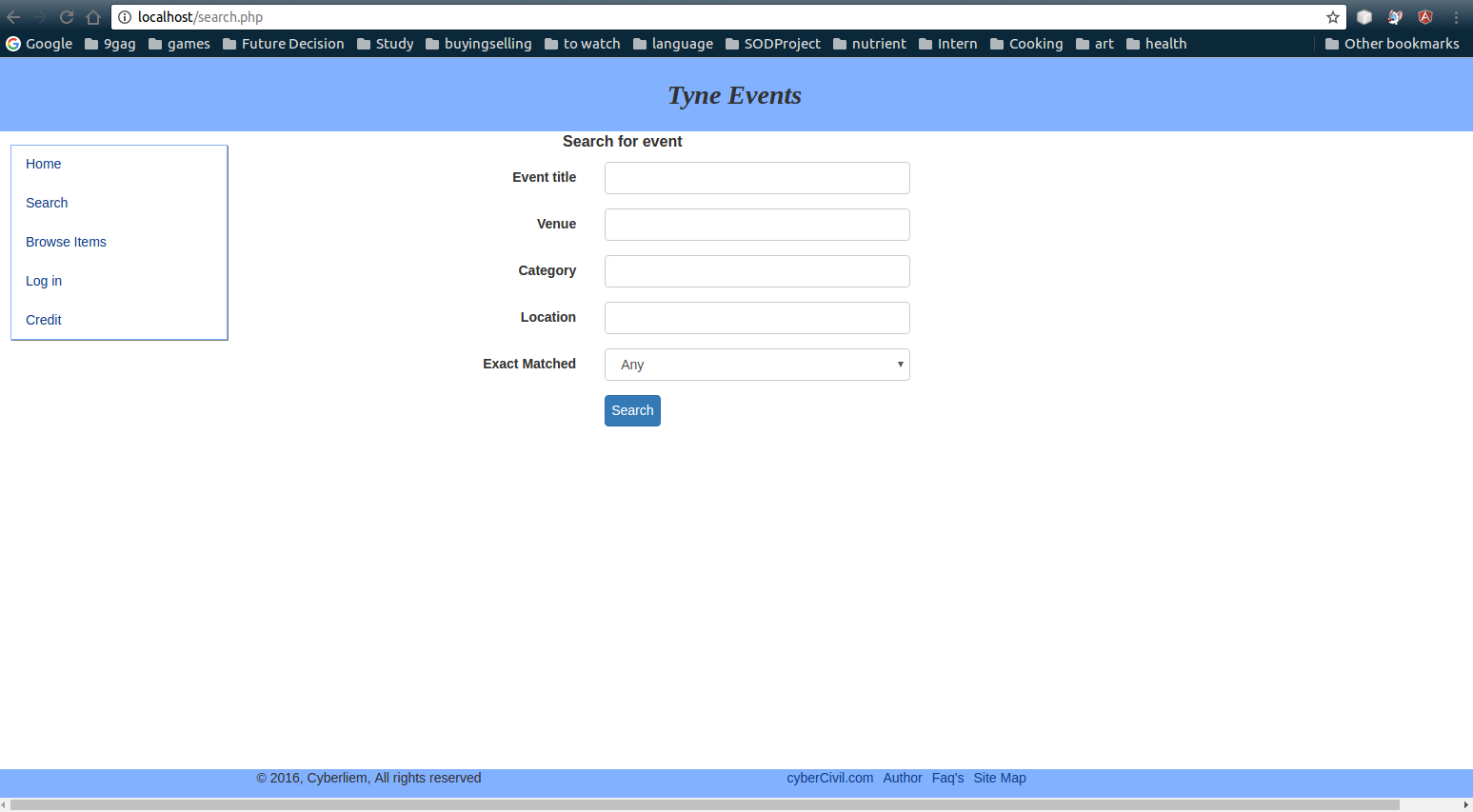
## **Website work flow:**



*Figure 2: Website work flow*

Figure 2 show the work flow chart of the website. Each item will be described in more detail in later section. Generally, the webserver will send all the view on request of user, and based on user’s state. PHP controller in the back of the server will receive the user input, validate it, call API to query Database and render the HTML page based on result. The HTML will then be included in the HTTP response to user.

### ***General View***



*Figure 3: Website Index page*

Figure 3 show the index page. At the most basic concept of a webpage, it has a header, navigation bar, main contain and footer. The header is on the top to display the brand/ logo. The navigation was made to be a sidebar on the left. The design of main content are a bit of a mockup, since no specific requirement was asked by user: it contain some thumbnail to display random event, with nice image and brief introduction for generic user to view the main objects of the pages: Event. There is also a block of welcome message for the ease of use on user.

The HTML elements in this webpage is organized mostly in the folder namely includes. This way of organized make it easier to keeping tracks of all the display (view) as well as reuse the code. Other folder inside project contain specific modules of the project itself. For example, admin contain all the function for administrator users, logonModules contains all the function for login-log out. API provide database interface. Others are universal scripts for website, include js, Images and CSS.

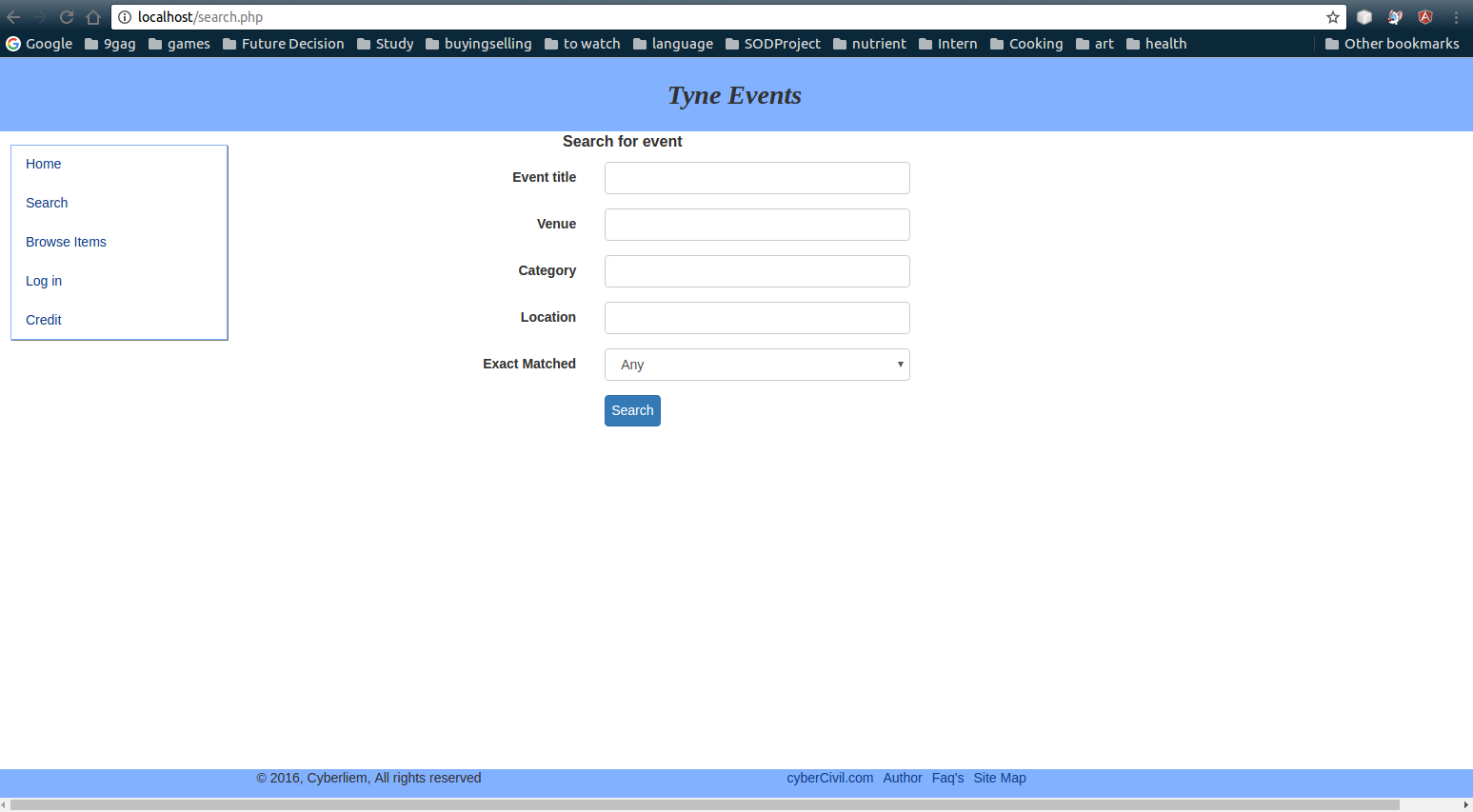
The layout of website is generated by the following procedure:

* include($\_SERVER["DOCUMENT\_ROOT"] .'/includes/header.html');
* Main content
* include($\_SERVER["DOCUMENT\_ROOT"] .'/includes/footer.html');

This is the beautiful of dynamic website: The code can be reuse, and can be dynamically change depend on main content. Header provide the title and the navigation bar. Footer provide user with copyright and other links for contacting author. This create the feeling that the whole web site are “one”. It worth noticed that all the include contain $\_SERVER[“DOCUMENT\_ROOT”], the dynamic part of the server root to generate a relative path, to make any integration on any server machine/ engine safe.

All the view page is decorated and styled based on CSS. In this project, the concept of CSS bootstrap grid (Bootstrap-Team, 2016) is employed: to divide the page into 12 manageable block vertically, and each and every basic building block of the website is styled with a generic position allocation. All these css is stored on /css/buildingBlock.css. Then come the /ccs/components.css, where the other component that made a website (link elements, table etc, pagination etc…) is styled. The /css/Misc.css stored information of styling for miscellaneous elements that found around the web. Finally, /css/style.css provide a quick and experimental file on customizing the view of the page. The guidance for all these element was provided by w3school tutorial (W3School, 1999-2016).

### ***Search***



*Figure 4: Search page*

Figure 4 show the search page. The search was designed with the four main search features, which are event title, Venue, category or location. There are 2 search modes: any match or exact match with the default being any match. Basically, the any match search mode find any presence of the keyword in database with according search entity. The second mode, exact match, allow user for find a more refined result cointain exactly the keyword. If user want to search for multiple features, just enter the key word in multiple input. If left blank, it will assume that the features are unbounded of any parameter (this is equal to WHERE [feature]=\*).

The form was generated dynamically, initialized by the array $searchFeature. Should the requirement arise that there's to be more search feature add (for example, event Description), just add it into $searchFeature array and the php script will generate a new search box. This is one of the advance approach to the static way, which geneate search form by pure hard-coded HTML. The generation can be observed as follow:

<?php

**foreach** ($searchFeatures **as** $k => $v) {

**echo** '<div class="form-group">

            <label class="universal-block block-4 box-label" for="'.$v .'">'. $k .'</label>

            <div class="universal-block block-4">

                <input id="'.$v.'" name="'.$v.'" type="text" class="form-control input-md">

            </div>

        </div>';

}?>

For the nature of search query, there's no need to send it securely, hence the GET method is used to pass the search parameter to the controller, which is searchResult.php. This script will check if the GET array is sent, extract data from it and put into parameter. There are some minor validation, include stripping tags.

### ***API***

After the parameter is passed, the next step is to query the database for result. This website was implemented with one of the most modern concepts in data processing on web application, which is the API. API stands for Application programming interface (Zapier, 2015), it provide a defined method of interact with web server backend, and can be made by simple HTTP request. The API was made on this website are defined as follow:

[server root]/api.php?endpoint=x&parameters.

In which the end point is the action taken on database. It can be either retrieving data (endpoint=view), or update item (endpoint=update). In a simple view call, the PHP server return an encoded json string. This json string can then be decoded by any API call function, in order to return the result array. Figure 5 shows an example of API call on search query.



*Figure 5: API call.*

As figure 5 show, the string result of the query can be read by any application provided HTTP request/ response and parse into their own application. The advance features of API included the cross-platform development. Nowadays, people tends to in-corporate their web system with other system (for example, Facebook allow other web service to integrate with Facebook login, which will be done by calling Facebook API), or with different interface (for example, different front-end engines are used for PC browser and Mobile apps. The API will allow the backend engine to be free of any worry on the restrained of programming language, as well as the stability of the web.

For this project, except for the api.php called directly under the folder, the rest of the files are in /API.

One thing worth notice about calling API is that in order to call it from PHP script, normally the practice is to use cURL library. However this project rely only on the Pure PHP installation without any additional library, Hence, the author has gone at length to write a customize HTTP request. This is basically instead of using a web browser to retrieve information, or to use external library to retrieve it; a HTTP request is made from scratch, from method, argument, header… and send it over to Server’s input stream, then read the output content as a file. Even more than that, the Update API use PUT method, which require the reading of input directly from PHP://input. How fancy is that? It is no longer a simple PHP work, but the whole WWW in making. The code is in /API/callAPI.php.

### ***Database connection: OOP and PHP***

The Gateway to MySQL is provided in /API/DBconnect.php, which provides the class construct, properties and method of the database connection object. The concept of Object Orientated programming is not new, in order to archive modularity of the each components, as well as reuse the blueprint of a method/ components in different part of the web, the *abstraction* of a components are presented in class and later the object is created based on the class declaration (Ullman, PHP Advanced and Object-Oriented Programming, 2013). In this project, the DBconnect contain all the parameters necessary for the database:

<?php

**class** DBConnect{

**protected** $User = 'root';

**protected** $Password = 'l1i2e3m4';

**public** $Host= 'localhost';

**protected** $Name = 'tyneEvents';

**protected** $args;

**protected** $dbc;

}

?>

All the information for DBconnect are here. Note that if you want to use the database of your own, please replace these default property of DBConnect class. In construction of a Database connection object, the contruct will created a PDO object ($dbc) based on default properties and passon any arguments to $args. The DBconnect object is then perform any requested API processing based on its method, and return the result to API object, which then passed to HTTP request as seen in figure 5. Other object orientation implemented in this project are the API class and input validator class. (Yes, anything dealing with database and secure item is made into object to minimize the chance of it being manipulate). The API class is created with the guidance of RESTful API (Maynard, 2013).

### ***Prevention of SQL injection: PDO***

The PDO object created with each instance of Database connection is based on PDO PDO is a Database Access Abstraction Layer (phpdelusions, 2016). Unlike the lower level abstraction of mysql and mysqli, both of which are low level, bare API, not intended to be used directly (but only as a building material for some higher level abstraction layer), PDO provide an abstraction ready to be used directly and provide a much more secure as well as usage and reusability.

PDO comes with parameters binding. This is one crucial part of preventing SQL injection: Instead of executing a static built SQL statement like the following code:

<?php

$sql = "SELECT \* FROM tyne\_events WHERE ID=".$eventID;

$result = $databaseConnection->query($sql);

?>

If the malicious user input $eventID=’1 'x' AND 1=(SELECT \* FROM tyne\_users); --'; then the sql Statement will become "SELECT \* FROM tyne\_events WHERE ID=’1 'x' AND 1=(SELECT \* FROM tyne\_users); --; which will give out all the users details. The password may be hashed and the attacker will take some time to get to it, but if the event is an orgy, the user might not want their name to be known.

In our PDO object implementation, the flow is as follow:

<?php

**public** **function** updateStatement($k, $v, $mode) {

**if** ($mode==1) {

**return** ($k. " = :".$k." AND ");

}

**else** **if** ($mode=2) {

**return** ($k. " LIKE :".$k." AND ");

}

**else** **return**(" AND ");

}

**public** **function** getEvents() {

$sttstr="SELECT \*";

        $sttstr.="FROM te\_events JOIN te\_category ON te\_events.catID= te\_category.catID ";

        $sttstr.="JOIN te\_venue ON te\_events.venueID=te\_venue.venueID ";

//Append args to statement.

**if** (!(**empty**($**this**->args))) {

            $sttstr.=" WHERE ";

**foreach** ($**this**->args **as** $k=>$v) {

$sttstr.= $**this**->updateStatement($k, $v, $mode);

            }

            $sttstr= substr($sttstr, 0, -4);

        }

        $sttstr.="ORDER BY eventTitle ASC";

//Prepare the statement

$stmt= $**this**->dbc->prepare($sttstr);

  //execute according to arguments.

       $stmt->execute($**this**->args);

        $result= $stmt->fetchAll(**PDO**::FETCH\_ASSOC);

        //echo gettype($result);

**return**($result);

}?>

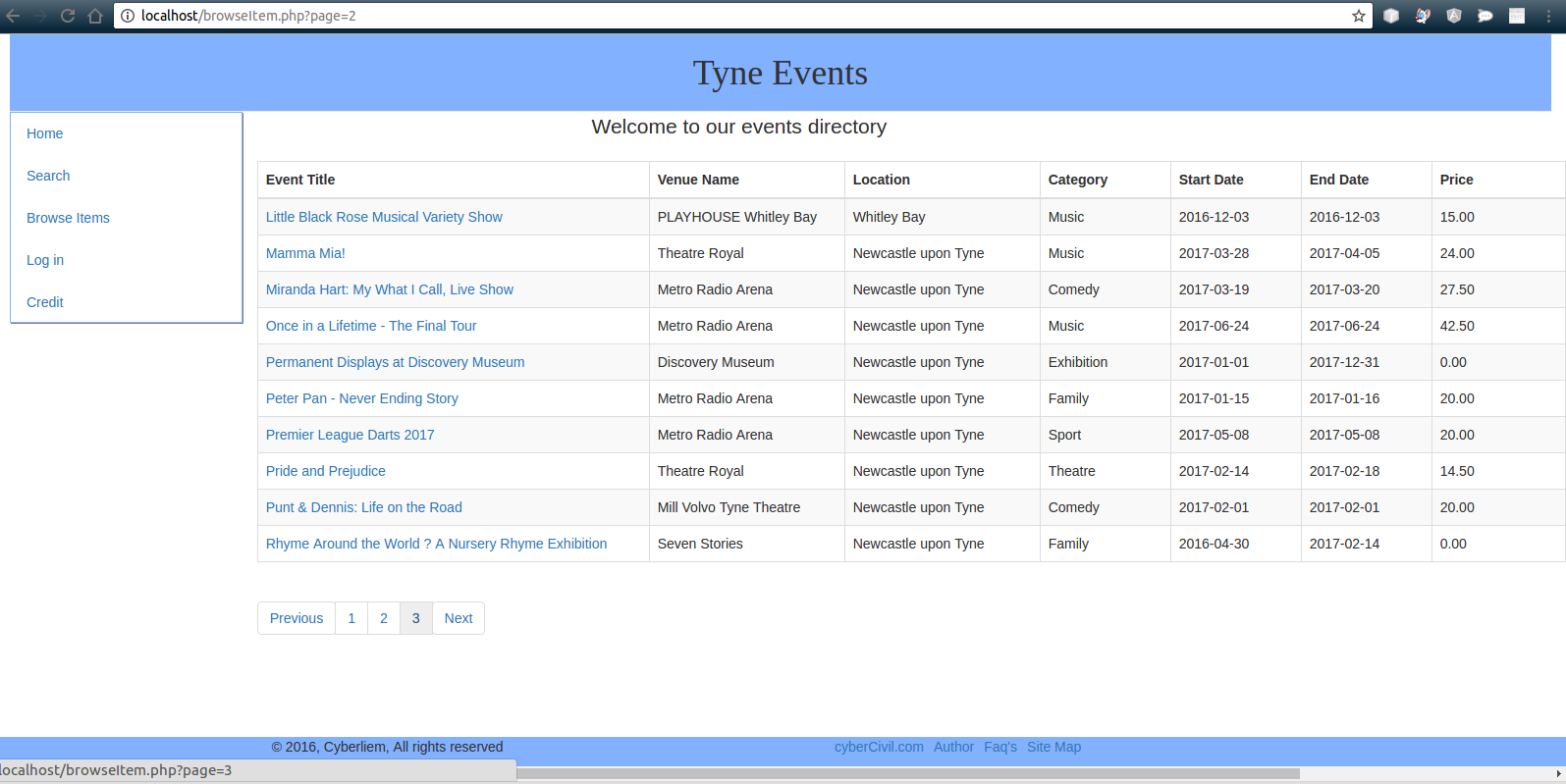
There are two smartations here: The string is built based on the search mode and arguments (the update Statement will append any clause to $sttstr). Once this is done, the $sttstr will have the form of “SQL statement where item=:item”. The $stmt will then be the preparation of statement, i.e, compile the statement into ready to run codes. $stmt->execute([item: value]) will bind the value to item, and these value can not be compiled to SQL functional, but only as parameter. Hence the site is safe from SQL injection. In constract with the injection example above, when all the parameter is included in the SQL string, and compiled as once, which will render user input into SQL functional if the code is available.

PDO object also support multiple update statement, which is advanced and quicker than the normal mysqli. All the example of SQL injection (Friedl, 2007) was tried and prove to be ineffective against this current project.

### **Pagination and filtering: Sever-side Memory.**

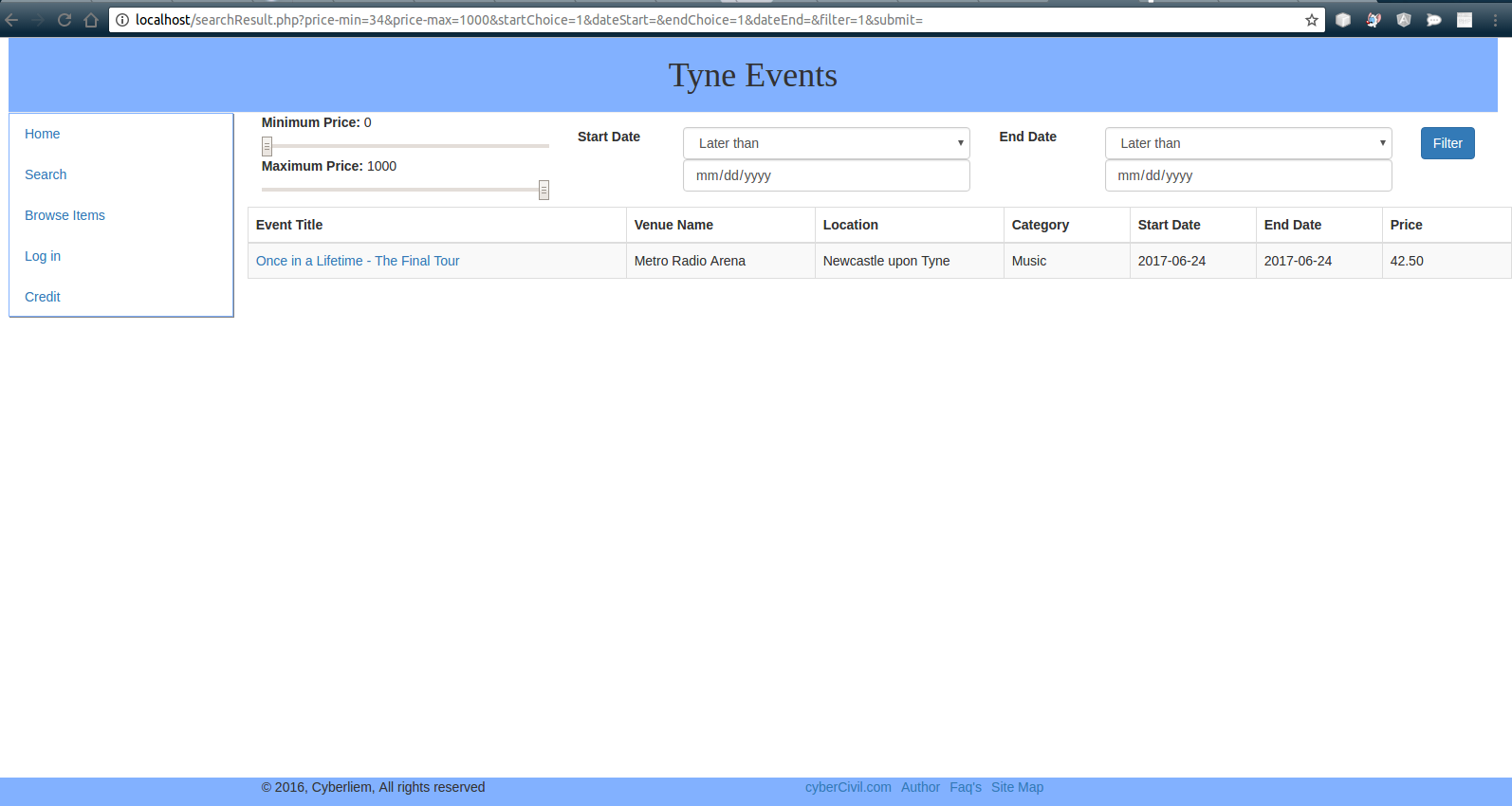
The display result page come with 2 other distince feature: filtering and pagination. To make these function become easier, the concept of PHP SESSION is employed. PHP SESSION is the memory of server, used to retain neccessary information. It make stateless http into stateful HTTP, which means that the http request and response will be based on the “state” of client and server. In this web project, to fasten the load time of a request, the PHP SESSION Stored the result array after any change of request. As long as user still interest in the current result, it will be retained in $\_SESSION[“items”].

Pagination is done by countering how many object is there, and ceil it to get number of pages. Then on each page, the starting item and ending Items number will be passed into the page, and the table only display those item between the starting and end. It worth notice that normally, with other project, pagination is done by either query the page on OFFSET, or to query a sufficient amount of data on each query and use Javascript to show/hide the items. However, due to the purpose of demonstrating the understanding of Server-side memory, the method of storage results in SESSION is employed. It is worth noticed that the SESSION method require server resources (the higher ram you want from hosting company, the more you pay), as well as unsalable in a really big application (For example, Amazon warehouse doesn’t use this method, for obvious reason). The method of query on OFFSET for pagination is subject to increase traffic and might not be good enough given high latency network. Figure 6 shows the pagination function of the webpage.



*Figure 6: pagination.*

Filtering is done on the same concept of pagination: the items are stored in SESSION after the search query, for filtering, the feature include are Price range slider, Start date and End date (all done by HTML input element). After each submission of filter request, the items array in SESSION will go through the filtering, with each item meet the criteria will get their $item[“display”]=TRUE; The displayResult.php will handle display based on the result array. It requires advance programming skill and advance planning. Figure 7 show the example of filtering.

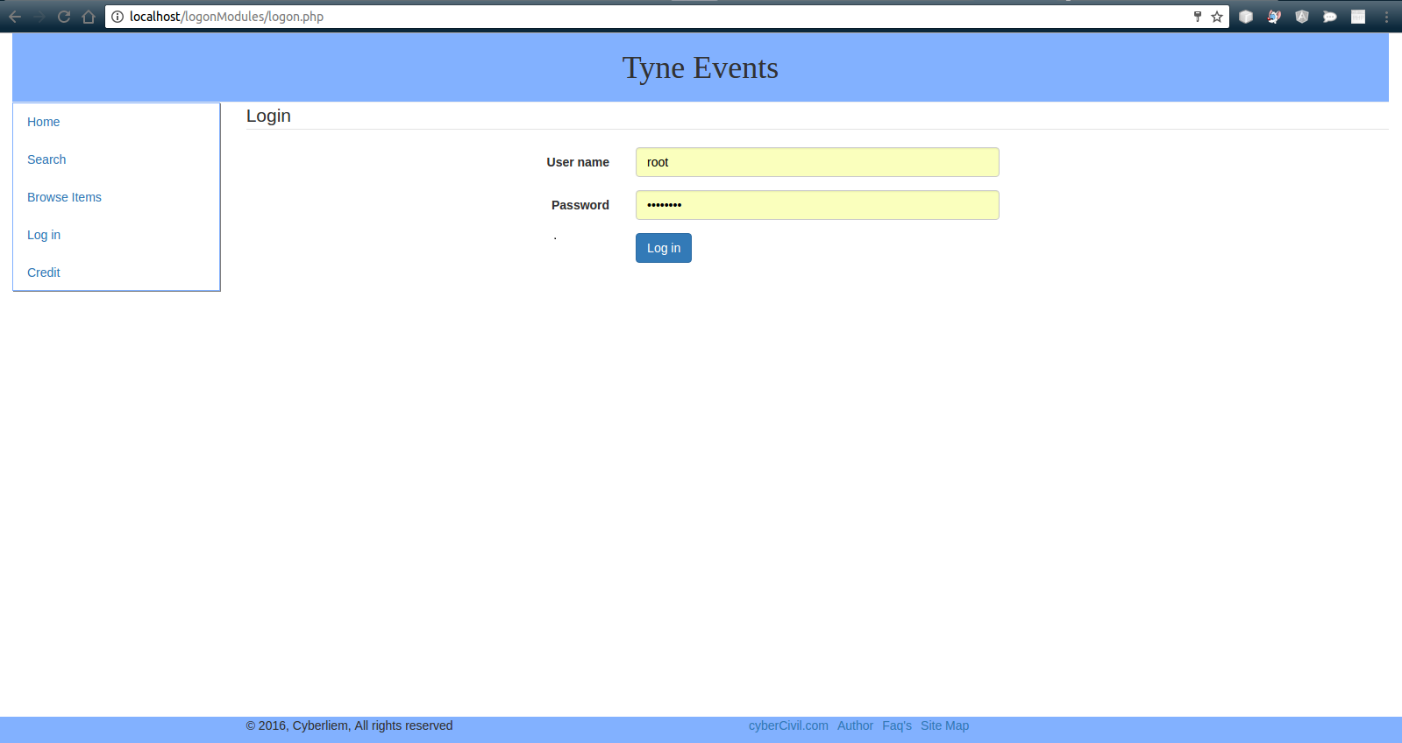


*Figure 7: filtering.*

### ***Browse Item***

Browse Item is no different with display result on search. On browse Item, a simple API call /api.php?endpoin=view will return all the events, and it is paginated into table, as display. The same concept of pagination explained in section 6 is shown.

### ***Login Function***

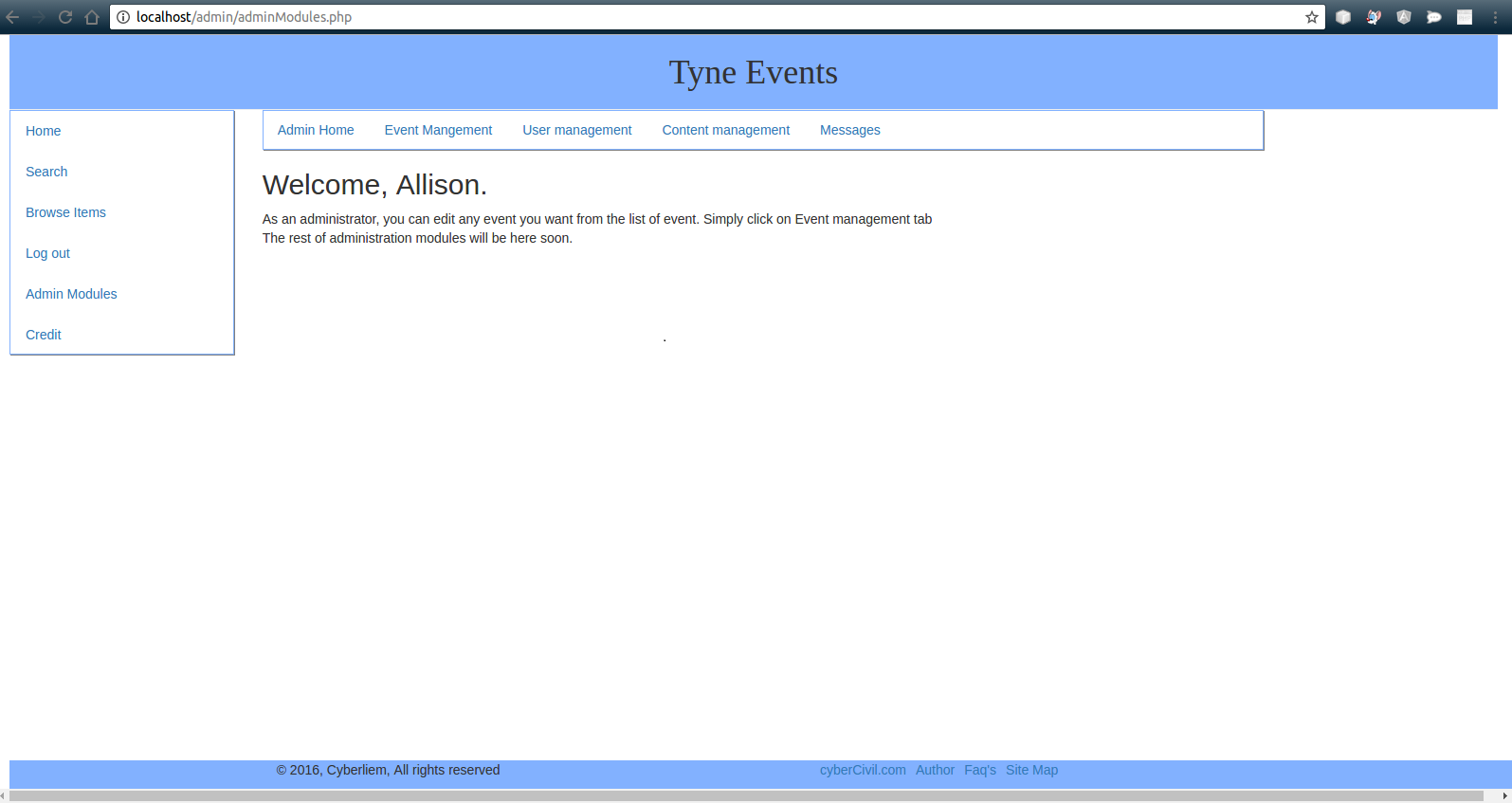


*Figure 8: login form.*

The login form can be seens in figure 8. Whenever user enter ID and password, a PDO object will be created and query against database to verify it. Once it is verify, the $\_SESSION[‘userName’]; $\_SESSION[‘userID’] and $\_SESSION[‘userAgent’]. The userAgent is some extra step for verification, in case someone trying to hijack the session, the userAgent will be compared to make sure that at least the computer accessing Session use the same agent. These will be the state of the client-server.

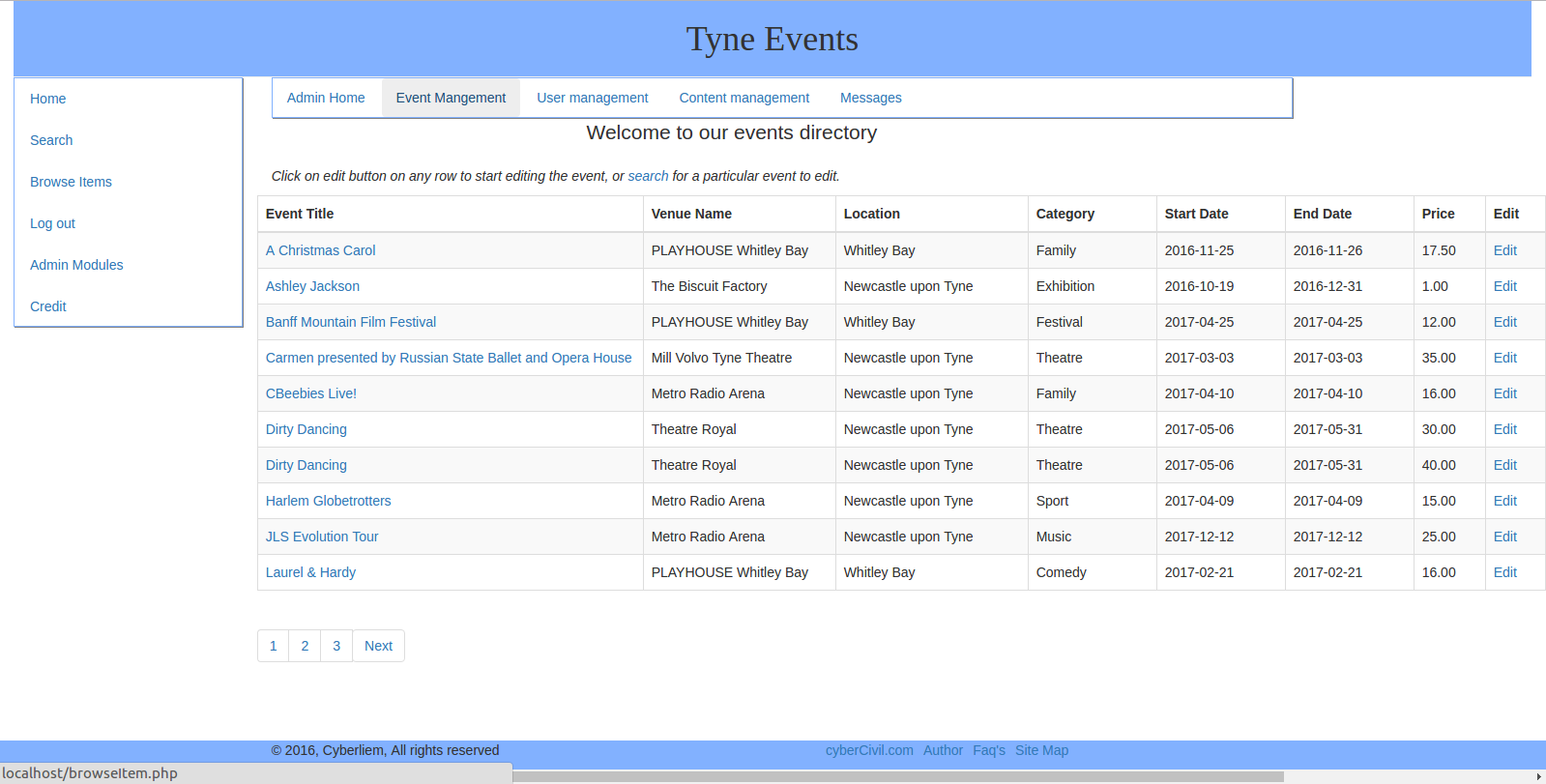
If it is correct, the user will be redirect to loggedIn.php. Once the State is set, the Admin panel will appear as figure 9. Only the Events management has been done, the rest are pseudo code

If it is not, the user will be redirect to login form with error message.



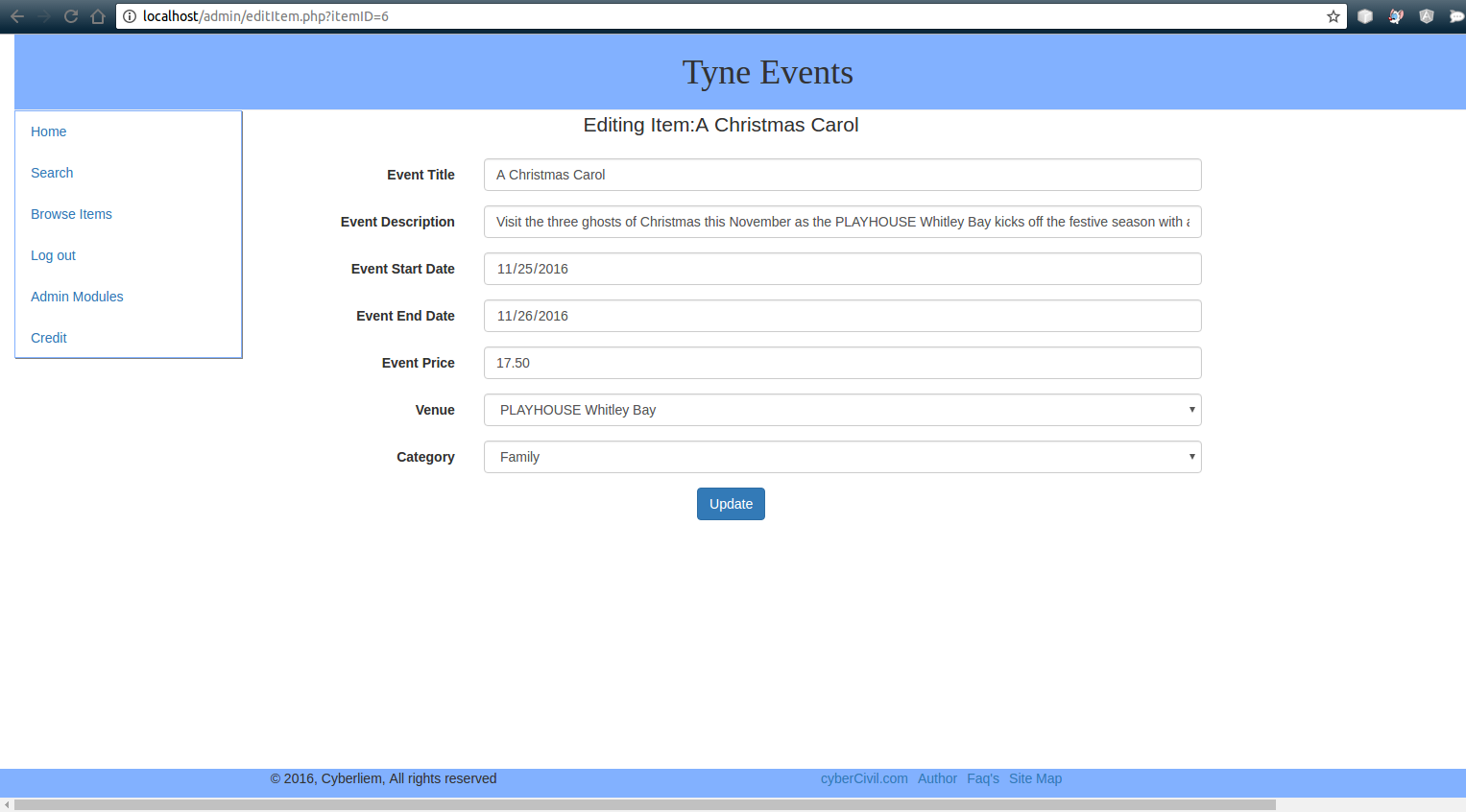
*Figure 9: Login-success: admin panel.*

### ***Edit Item.***



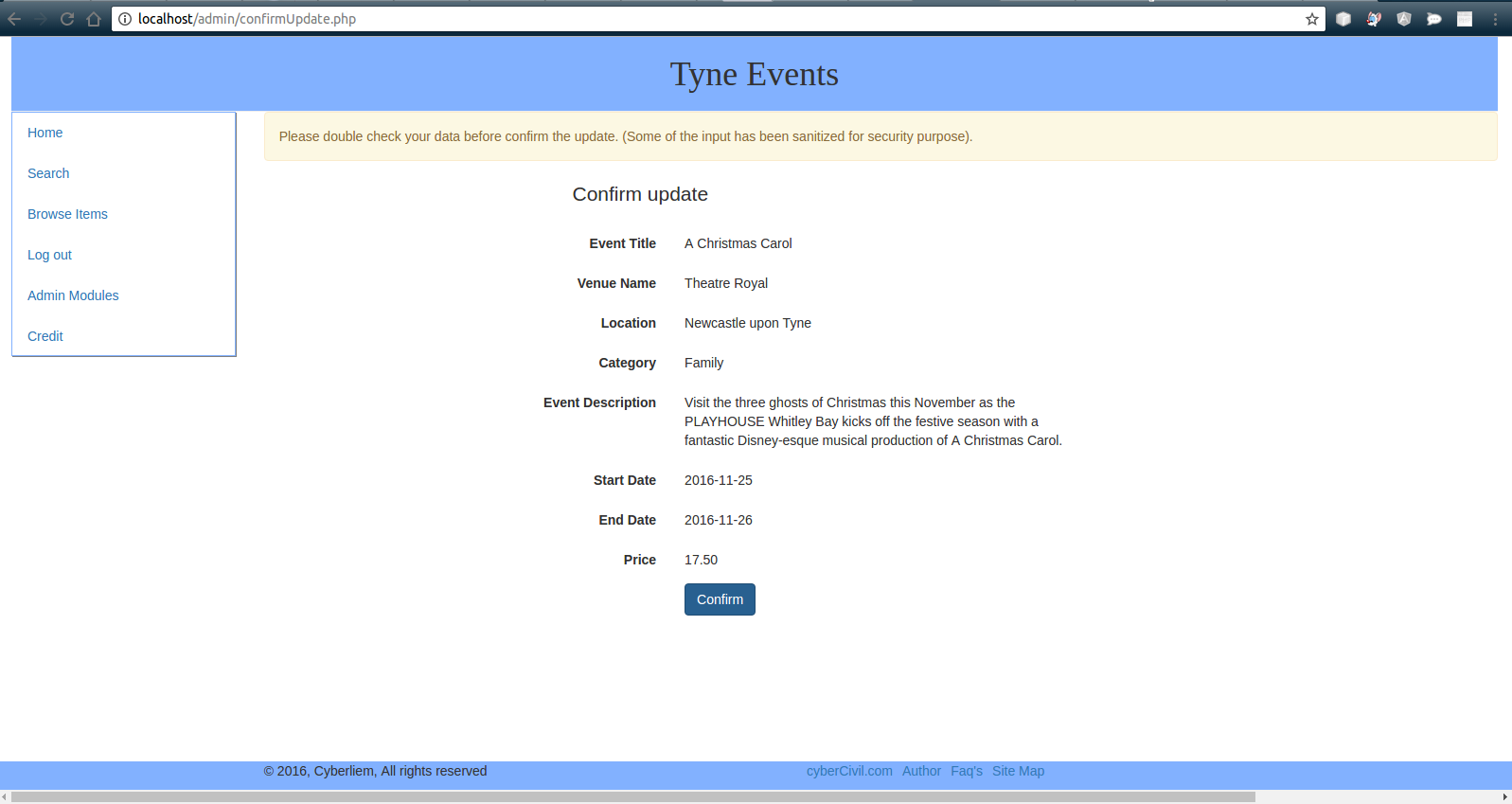
*Figure 10: Events Manager from Admin panel.*

After loggin successfully, on the table displaying event, either from search or from browse item, an extra collum with edit button appear as figure 10 shown. User can click on that and the form will appear as Figure 11. Any detail filled in here will be validate by using validator object, created from ValidateInput.class.php. This will make sure that no tag allow, all the value must be in formate (Date, number etc…). Some of the text input will be stripped of as well. The validator object can be reused.



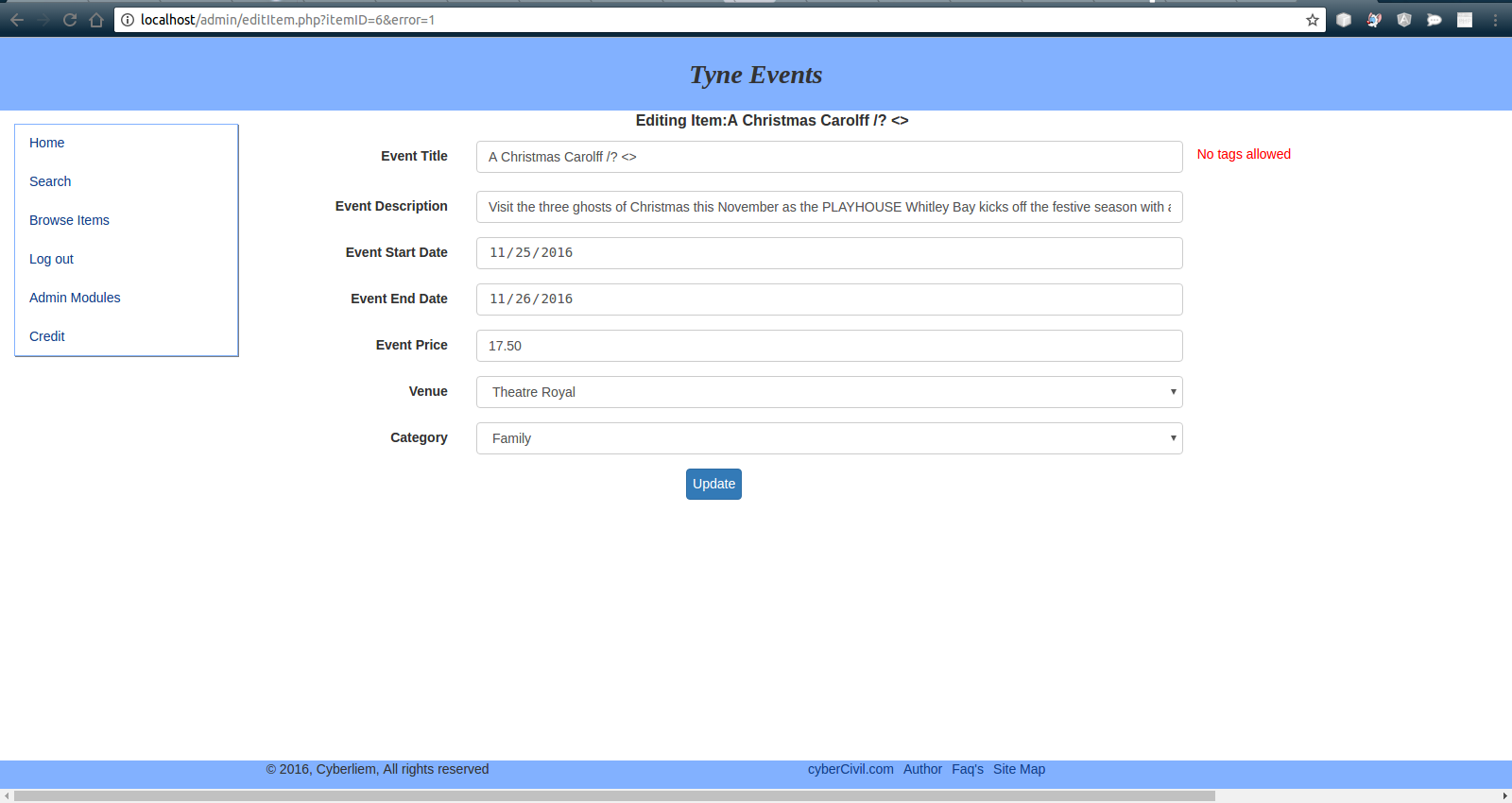
*Figure 11: Edit form*

If there is no error, it will redirect user to the confirmation page, where user can review and submit the final data. Note that in confirmation the data can’t be changed, it’s in the Server-side memory ($\_SESSION) now. So no manipulation can be done, only to submit, which is better security and user friendly. Figure 12 show this.



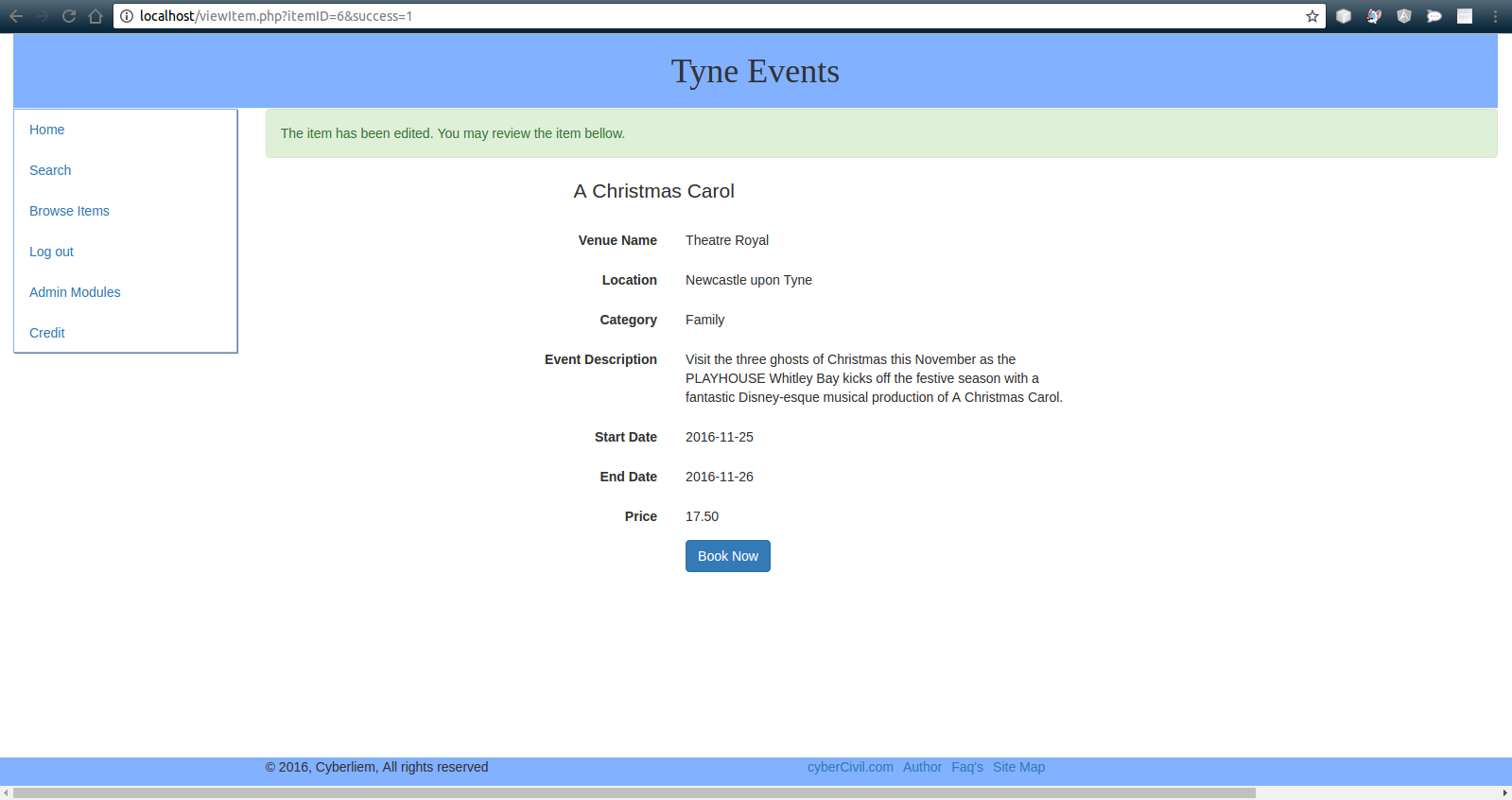
*Figure 12: confirm Edit form.*

If there is error the page will redirected to the edit form, with message display as in figure 13.



*Figure 13: Edit form with error indicator.*

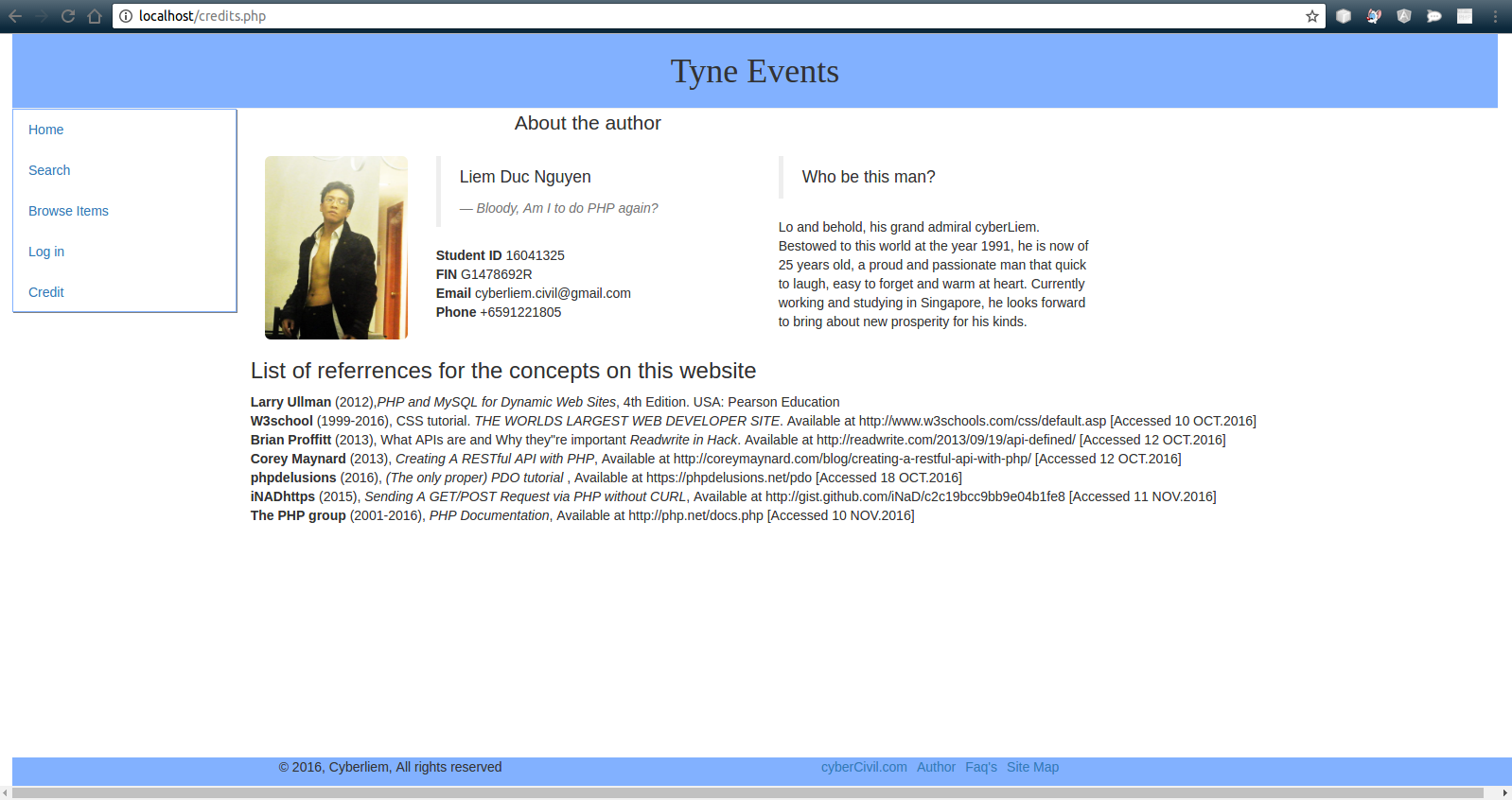
Once the user confirm the update, the update API will be called with PUT method. PUT method is better than POST method for update since it doesn’t depend on the state of the client. And one PUT can be send at a time, to prevent deadlock in writing in Database. To send PUT method, the input must be read in “php://input”, where the abstraction of network-files are demonstrated. More detail can be view in API/callAPI.php. The successful Update will be display with new Item details as figure 14.



*Figure 14: edit successful*

### Credit Page.

Well, this is where you got to know the author, as well as seeing some material the author has used to make this wonderful page. Appreciate the effort and the man is the proper protocol for viewing this page.



*Figure 15: Credit page.*

## **Conclusion**

Again, the use of dynamic web development is essential to any IT worker today. It provide a better, more secure way of presenting information, receive user input and response to them. PHP, MySQL on Linux’s Apache is a popular way of doing this, and it is proved to be effective.

This particular project is much more than what it’s asked for. It’s prepared for bigger and better scalability, as well as provide a very good frame for later development on mobiles and/or integration with other project. It also demonstrate a range of concept in web development, from basic like object orientation, database query to more advance concept, like API, modularity and advance handling of HTTP request. Still, for limited time there are limitation to it. Once other thing worth noticing is that the project is built for academic demonstration purposes, not for actual commercial product, hence the scope as well as configuration might differ. Nevertheless, it is an excellent web app back bone.

# Bibliography

Bootstrap-Team. (2016). *About Bootstrap*. Retrieved from Get Bootstrap: http://getbootstrap.com/about/

Connolly, D. (2000). *History of The Internet& World Wide Web*. Retrieved from w3.org: https://www.w3.org/History.html#refs

Friedl, S. (2007). *SQL Injection Attacks by Example*. Retrieved from Unixwiz.net Tech Tips: http://www.unixwiz.net/techtips/sql-injection.html

Lee, B. (1991). *The original HTTP as defined in 1991*. Retrieved from w3.org: https://www.w3.org/Protocols/HTTP/AsImplemented.html

Maynard, >. (2013). *Creating A RESTful API with PHP*. Retrieved from >Corey Maynard Blog: http://coreymaynard.com/blog/creating-a-restful-api-with-php/

phpdelusions. (2016). *(The only proper) PDO tutorial*. Retrieved from phpDelusons: https://phpdelusions.net/pdo [Accessed 18 OCT.2016]

Ullman, L. (2012). *PHP and MySQL for Dynamic Web Sites, 4th Edition.* USE: Pearson Education.

Ullman, L. (2013). *PHP Advanced and Object-Oriented Programming.* Berkeley, California: Peachpit Press.

W3School. (1999-2016). *CSS tutorial*. Retrieved from The world lagest Web Development Site: http://www.w3schools.com/css/default.asp

Zapier. (2015). *An Introduction to APIs*. Retrieved from The best APPs better together: https://zapier.com/learn/apis/