**Installing**

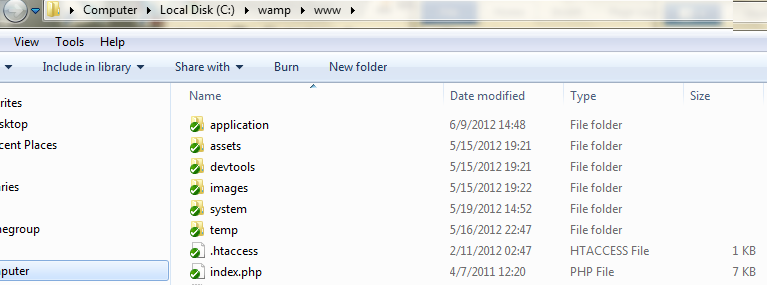
1 ) In the www folder of your web server, extract the files. It should look like this.

Figure 1 Sample layout of files in Windows 7 - with WAMPServer as the web-server package.

2) Open application/config/config.php to key in MySQL DB username and password and set encryption key.

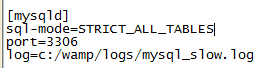
3) If possible, then, in your MySQL configuration file in [mysqld] section add this:

Figure Screenshot of the particular part of "my.ini".

sql-mode=STRICT\_ALL\_TABLES

This is an additional step in securing your system.

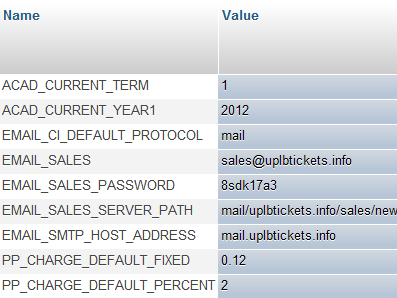
4) Open PHPMyAdmin and import “uplb\_xts.sql”.

Figure DB entry of some settings.

5) System Settings ( table `system\_settings` )

* If you want the functionality of associating classes to events to be operating correctly, edit the entries: ACAD\_CURRENT\_TERM and ACAD\_CURRENT\_YEAR appropriately. For the earlier, the valid values are 1, 2 and 3 (integer) that stands for First Semester, 2nd Semester and Summer respectively. The last constant should be obvious.
* If you want the email functionality operating correctly, edit the entries whose names are starting with ‘EMAIL’. The values for these are typically provided by your web hosting provider.

6) Make sure that the PHP module “SimpleXML” is enabled – needed for some info exchange between URIs/functions.

7) Make sure that “output\_buffering” in PHP is set to “off”. This is the default setting by the way but to be safe, check. It is usually found in “php.ini” file.

8) It would be nice if you the PHP unlink() command can successfully do its task – i.e., delete permissions in /assets/xmltemp is set for the web server. This is to delete temporary XML files.

**Internal conventions**

**Booking Cookies-on-server**

* **VISUALSEAT\_DATA**

PRESENCE: <row><dash><col> | ABSENCE: “0” | SEPARATOR: <underscore>

Example:

H-1\_B-8\_G-28\_0\_0\_F-2

**Program Logic – Form Submission,processing and redirection**

**Air Traffic Control**

Name's inspiration comes from aviation - you know, you need to have clearances and acknowledgments when flying.

This was designed as the new way of sending content to the server and acting appropriately instead of just submitting the form and let the server do all the processing. Here, both server and client is involved – client handles getting clearance from server to proceed to a specified “next page” and the actual deed of proceeding. So this means that JavaScript should be enabled and JQuery is supported (sometimes problematic in IE 8 and earlier).

These are the core files and or functions needed for this functionality

|  |  |  |
| --- | --- | --- |
| C/S ? | Name | Located at |
| server | airtraffic.php | application/libraries |
| server | contact\_tower() | application/controllers/sessctrl |
| server | makexml\_model | application/models |
| client | airtraffic.js | assets/airtraffic.js |

Client side

Now, let’s begin our description of how it works in the client side.

When the form is to be submitted in the client side, the go\_submit() function in “airtraffic.js” should be called, loaded with respective parameters ( see the file for specifics ). The function then initiates an AJAX request to the specified URI – including the data needed by that URI to process our transaction. Once that URI returned headers and output, the browser (client) will call now the “sessctrl/contact\_tower” to request clearance to go to the next page as well as the URI to go. Now both AJAX calls return content in XML that contains whether the transaction succeeded or not. For the second AJAX call, this should be passed to the “atc\_success\_func” function specified when calling go\_submit(). The “atc\_success\_func” function should be defined in the file dedicated for that page/functionality. Any error received is processed accordingly by “airtraffic.js” thru displaying of a modal based on the XML received from server.

Server side

The URI that the first AJAX calls submits to is a function in a controller file. In that function, after those function access eligibility check and all, a call to library airtraffic::initialize() should be done. And then, the command for starting a transaction in the database is issued i.e. ( $this->db->trans\_begin() ). And then all activities/processing proper that the function does is done. Now after that a call to library airtraffic::clearance() is initiated – the return value of which determines whether the transaction should be rolled back or not. While in airtraffic:clearance(), headers and output to the client are sent immediately but the script is still running – this is the response to the first AJAX call in the client. The client then contacts “sessctrl/contact\_tower” for clearance – in which this function/URI modifies the respective XML file to signify our activity to “airtraffic::clearance” . And as such, the still and concurrently running “airtraffic::clearance” waits for the action of “sessctrl/contact\_tower” by reading the respective XML file until it was deemed that the latter has modified that XML file according to what the earlier wants or a specified timeout is reached . The earlier then returns BOOLEAN to the caller function in the controller. The latter on the other hand returns the appropriate headers, cookies and XML response to the client and the client reacts on it (i.e., redirection ).

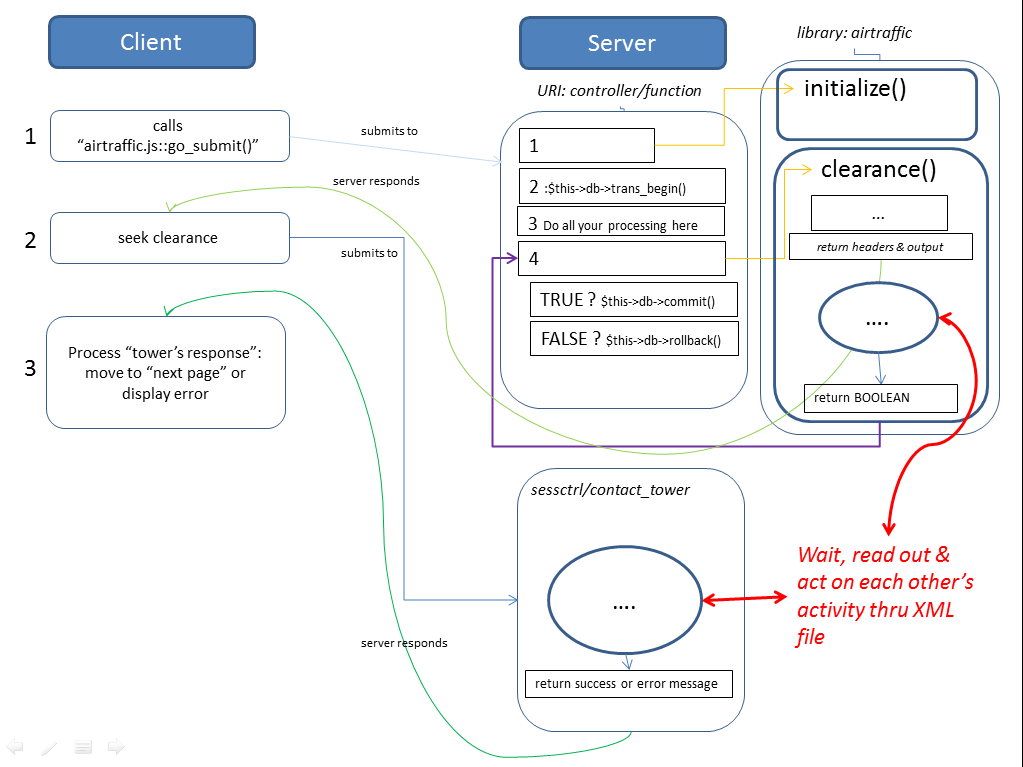


Figure 4: Illustration of air traffic mechanism. ( %userprofile%\documents\cmsc190-2\airtraffic.pptx )

Some noteworthy facts:

* There is somewhat a race condition in sessctrl/contact\_tower. See application/libraries/airtraffic.php:clearance() for the specifics. So far, phenomenon only demonstrated in intentional code modifying.
* With this architecture, we are highly-resistant to connection errors (i.e., network conncetion interrupted, updated headers and cookie were not sent). Since we base on the CI session cookie( activity name and stage number) whether or not a function is accessible, problem would arise if airtraffic::clearance returned TRUE while sessctrl/contact\_tower is about to return success message but failed to send or connection between it and client was interrupted (i.e., network cable disconnected). Since the database changes has been committed but CI session cookie not updated – resubmitting will do the same action and the action will be re-done and this will bring an error like insertion of a duplicate entry in the database. Now, what we did is there is some entry in the database that takes note of the activity stage and number of the last successfully executed function and in function checks, this is checked too – so if ever resubmission to a function is done and it was submitted to already and was successful – user would be told to forward to the next page.
* The PHP command “ignore\_user\_abort(true)” is enabled on the URIs called. This means that even though we suddenly closed the browser (window/tab) we are using the script on the server-side is still continuing the execution. The transaction would be rolled back because the client cannot contact “sessctrl/contact\_tower” for clearance since the browser was exited. However, this could be a resource hog for the server side and a potential for DDoS attack. In the future, a mechanism for limiting page refreshes/resubmissions for some certain time should be set.

**Admin Stuffs**

**Creating Seat Maps**

* The following are the valid values for the `Status` column of both tables `seats\_actual` and `seats\_default`.

|  |  |  |
| --- | --- | --- |
| Code | Title | Descripton |
| 1 | OCCUPPIED | Someone has already chosen this. |
| 0 | VACANT | Functional – seat can be selected for seating. |
| -1 | AISLE | This space is actually aisle space, not a seat. |
| -2 | UNASSIGNED\_1 | Seat is there physically but no ticket class assigned to it. |
| -3 | UNASSIGNED\_2 | For future use/reserved. |
| -4 | RESERVED\_MB | Seat is temporarily reserved due to a change in booking ( Manage Booking feature ). |
| -5 | SEAT404 | The seat specified by the matrix indicator does not exist physically. |

**Specifying Payment Modes**

* For all payment modes, the following WIN5 type data are required to be in the internal\_data field:

|  |  |
| --- | --- |
| processor | What is the online payment processor. Valid values are “paypal”, “2co”, “moneybookers”. However, the implementation at the meantime is for PayPal only. |

* For PayPal payment modes, the following WIN5 type data are allowed in the internal\_data field (take note of the required ones though):

|  |  |  |  |
| --- | --- | --- | --- |
| specifier | requirement | description | valid values |
| merchant\_email | required | The email address of the Paypal recipient. | Email address |
| testmode | optional | Non-presence indicates FALSE – the PayPal transaction is not done in a sandbox (i.e., real money is involved). | “false”, “true” |

**Manage Booking**

**Session Activity Data when accessing the ff features:**

* Change showing time
* Upgrade ticket class
* Change seat

For all of these, when a section is not meant to be accessed, a value of “0” is assigned to it. For example. User just clicked “Change showing time”. The function that handles it should set “ticketclass”, “seat”, and “newcost” to “0”. However, if after that setting part and it was determined that those values be changed, then do it. The corresponding values are allowed.

|  |  |  |
| --- | --- | --- |
| Name & constant | Description | Values / Meaning/Constant in \_constants.inc |
| showingtime | MB\_STAGEPASS1\_SHOWTIME |  | 0 – not meant to pass | MB\_STAGESTAT\_NOTMEANT  1 – can pass this | MB\_STAGESTAT\_CANPASS  2 – should pass this |  MB\_STAGESTAT\_SHOULDPASS  3 – have passed It | MB\_STAGESTAT\_PASSED  4 – have passed and there were changes compared to the current booking | MB\_STAGESTAT\_CHANGED |
| ticketclass | MB\_STAGEPASS2\_TICKETCLASS |  |
| seat | MB\_STAGEPASS3\_SEAT |  |
| newcost | MB\_STAGEPASS4\_NEWCOST |  |