SW Engineering CSC648/848 Section 01 Spring 2017
Website name: TheGatorBay

Team # 02
Ajinkya Chalke (Team lead)(achalke@sfsu.edu)
Ivan Yu (Technical lead)
Bradley Ng
David Rodriguez
Jerry Auyeung
Thao Luu

Vertical Prototype Documentation

March 17, 2017

Contents

I. Vertical Prototype Functionalities

The following lists the functionalities supported by the vertical prototype:

- Displaying a list of categories from the database as links on a page.
- Showing a complete list of records from the database corresponding to items of a specific category.
- For each record, showing an image thumbnail which enlarges the image on the same page when clicked. The image thumbnail is accessed from the database for items corresponding to records with an image LONGBLOB attribute.

Categories Browse Page

Upon visiting the home page, the visitor is shown an option for browsing, searching, logging in and registering for an account. For this vertical prototype, only the browse option is currently supported, and when the visitor clicks on the button for browsing, they are taken to another page (the browse page) having all the valid categories found in the Categories table; the browse page can be found at the following link: http://www.sfsuse.com/~sp17g02/pages/browse. Currently, the Categories table has the following categories: books, clothes and electronics, and therefore, this page will have separate links directing the visitor to the page containing all records belonging to the books category, the clothes category, and the electronics category. Moreover, the "All" category link will always be on the browse page regardless of the categories stored in the Categories table. The "All" category link will take the visitor to a page containing all items stored in the Listing table.

Categories Browse Result Pages

After the visitor clicks on any of the links displayed in the Categories Browse Page, they are taken to the Categories Browse Result Page, which contains all records from the database belonging to the category specified by that link. The following are the pages showing all items of a specific category:

- http://www.sfsuse.com/~sp17g02/categories/view/books
- http://www.sfsuse.com/~sp17g02/categories/view/clothes
- http://www.sfsuse.com/~sp17g02/categories/view/electronics
- http://www.sfsuse.com/~sp17g02/listings

The last link listed refers to the page containing all items contained in the Listings table. All of these pages show the records row by row in an HTML table. Each column of this table corresponds to a column in the Listings table in the database. An image thumbnail is shown in the image column of a row, if the row corresponds to a record having a value for the image LONGBLOB column that isn't null.

Image Thumbnail

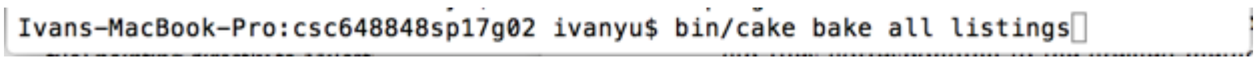
In the Categories Browse Result Pages, the images are clickable thumbnails. Once these images are clicked, an enlarged view of the image is shown in the center of the browser. When the enlarged view is displayed, the mouse cursor is changed to the “zoom-out” cursor, and the visitor must click anywhere on the browser page in order to remove the enlarged view.

II. Set-Up

In setting up the vertical prototype, several classes had to be implemented. Then, when all of the classes and tables have be created, only the classes associated with the Categories table and the Listings table were edited. The other classes were made in the purpose of future development of the project; these classes will be edited as the group develops the project from the vertical prototype. In implementing the Vertical Prototype, the classes and files that were edited were the PagesController class, the ListingsController’s index.ctp file, the CategoriesController’s view.ctp file, and the PagesController’s browse.ctp and homepage.ctp files.

Creation of Classes and Folder Structure

CakePHP provides commands that eases the creation of classes in order to implement the MVC model in the project. The command “bin/cake bake all tablenames”, where “tablenames” is the name of the table and should always be plural, creates a Controller, Table and Entity class, and ctp files corresponding to the created methods for the associated Controller class. All of these classes are in separate php files in separate folders in the project. For instance, the ListingsTable class is in the ListingsTable.php file, found in the src/Model/Table folder, the Listing entity class is found in the Listing.php file in the src/Model/Entity folder, and the ListingsController class would be found in the src/Controller folder’s ListingsController.php. Then, in the src/Template folder, a Listings folder is created, and this folder contains all the ctp files associated with the Listings table. The “bin/cake bake all tablenames” command is executed in the terminal after navigating the the project’s root directory. The following is an example of running the command, which creates the classes and ctp files for the Listings table:



```
Ivans-MacBook-Pro:csc648848sp17g02 ivanyu$ bin/cake bake all listings
```

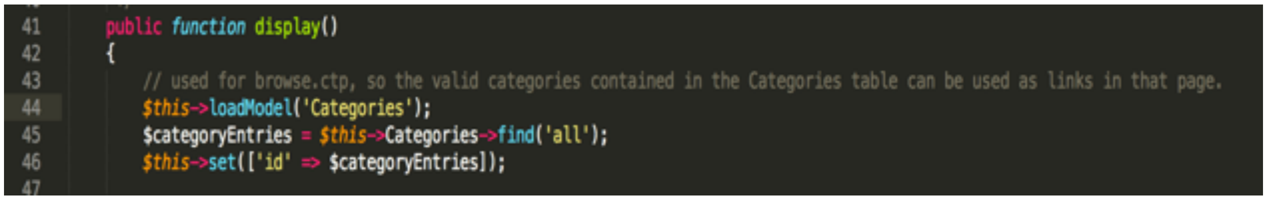
Figure 1, Creation of Classes and CTP Files for Listings Table

The command in **Figure 1**, creates the ListingsController, ListingsTable and Listing classes, and in the src/Template folder of the project, a Listings folder is created, and in this folder, ctp files for each of the methods in the ListingsController is created. CakePHP automates the method creation of these classes also, and after executing the command, the created ListingsController will have its own display(), index(), view(), edit(), add() and delete() methods. The methods in the ListingsTable class creates the rules for storing and manipulating the Listings table, based on its columns, primary keys and foreign keys. For example, these methods will disallow the ListingsController from adding an entry with a foreign key entry that is nonexistent in the table which contains the column that foreign key refers to. The Table classes were not edited in the creation of the Vertical Prototype, and therefore, the methods created by the

CakePHP command were used for these tables. Similarly, the Entity classes were not edited in the implementation of the Vertical Prototype.

PagesController

In the display() method of the PagesController class, the CategoriesTable class is loaded using the loadModel() method; this allows the PagesController to access the Categories table from the database. Then, a query is made, collecting all entries from the Categories table. The query result is stored in a variable and then, the set() method is called, allowing the variable $id to be accessed in the ctp files associated with the PageController class (the browse.ctp and homepage.ctp files). The following displays the lines of codes accomplishing this task:



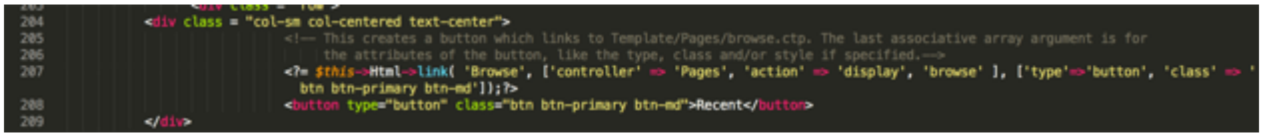
```
public function display()
{
    // used for browse.ctp, so the valid categories contained in the Categories table can be used as links in that page.
    $this->loadModel('Categories');
    $categoryEntries = $this->Categories->find('all');
    $this->set(['id' => $categoryEntries]);
```

Figure 2, PageController Using the Categories Table

The first line shown in **Figure 2** allows the PageController class to access the Categories Table. Then, the second line of codes runs a query statement locating all the entries of the Categories table and storing the result in a variable. Finally, the third line of codes allows the variable $id to be set in the browse.ctp and the homepage.ctp files.

Pages CTP Files

The homepage.ctp file displays the home page of the site. In this page, the "browse" button is implemented so that when pressed, the visitor is taken to the page containing all the categories found in the Categories table. The button is implemented as such:

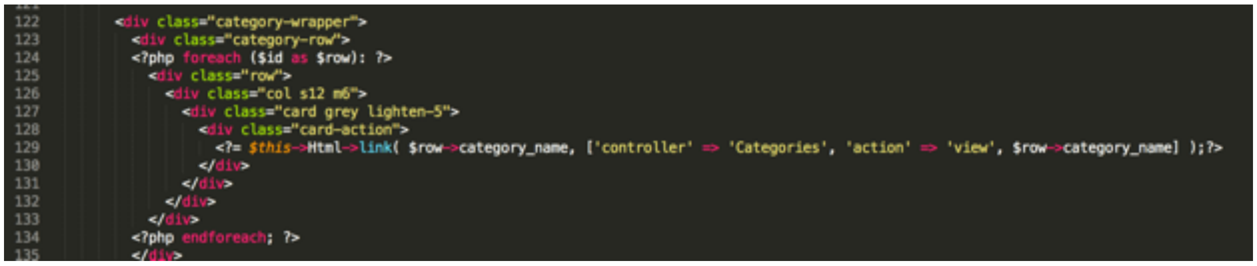


```
<div class = "col-sm col-centered text-center">
        <!-- This creates a button which links to Template/Pages/browse.ctp. The last associative array argument is for
            the attributes of the button, like the type, class and/or style if specified.-->
        <?= $this->Html->link( 'Browse', ['controller' => 'Pages', 'action' => 'display', 'browse' ], ['type'=>'button', 'class' => '
          btn btn-primary btn-md']);?>
        <button type="button" class="btn btn-primary btn-md">Recent</button>
</div>
```

Figure 3, Homepage.ctp "Browse" Button

Here, the button is labels "Browse", and clicking on the button calls the display() method of the PagesController class, so that the browse.ctp file is rendered by this method. The associative array passed as the last argument of the link() method gives the button its attributes. As previously explained, the PagesController's display() method has made it possible for the ctp files in the src/Template/Pages folder to access the Categories table through a variable. Therefore, in the following page (the browse.ctp file), the categories stored in the Categories table can be accessed.

In the corresponding browse.ctp file found in the src/Template/Pages folder, the following shows how the category links are dynamically created:

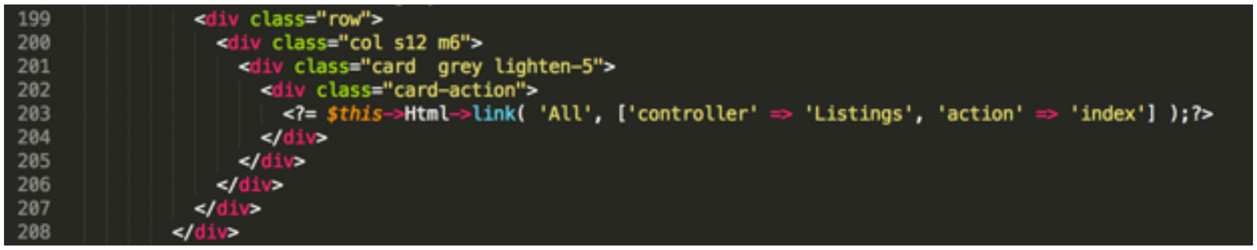


```
<div class="category-wrapper">
  <div class="category-row">
  <?php foreach ($id as $row): ?>
    <div class="row">
      <div class="col s12 m6">
        <div class="card grey lighten-5">
          <div class="card-action">
            <?= $this->Html->link( $row->category_name, ['controller' => 'Categories', 'action' => 'view', $row->category_name] );?>
          </div>
        </div>
      </div>
    </div>
  <?php endforeach; ?>
  </div>
```

Figure 4, Browse.ctp File Category Links Creation

In **Figure 4**, the $id variable is the same variable found in **Figure 2**; as previously explained, the "$this->set(['id' => $categoryEntries]);" line of codes in **Figure 2** allows the $id variable, which contains the query result having all records in the Categories table, to be accessed in the ctp files. The query result is accessed through the $id variable in this browse.ctp file, and as shown, a loop iterating through all of the categories in the Categories table is executed, creating a section for each of the category links for each iteration. Also, when any of these categories are clicked, the CategoriesController's view method is clicked, and the category_name attribute of the corresponding record is passed to this view method.

The browse.ctp file page also has a link going to the page containing all the items in the Listings table. The following shows how such link is created:



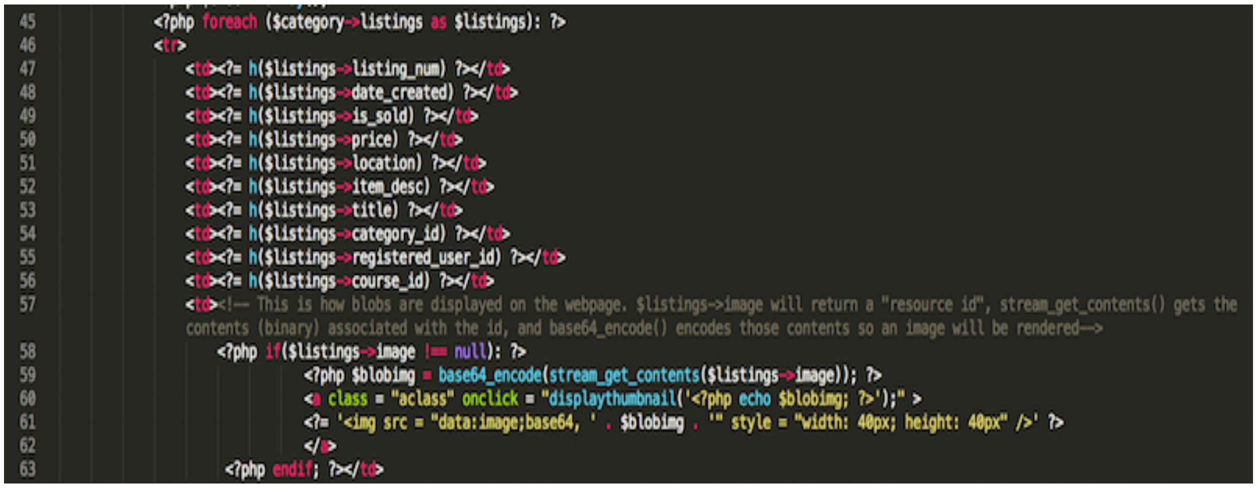
```
        <div class="row">
          <div class="col s12 m6">
            <div class="card  grey lighten-5">
              <div class="card-action">
                <?= $this->Html->link( 'All', ['controller' => 'Listings', 'action' => 'index'] );?>
              </div>
            </div>
          </div>
        </div>
      </div>
```

Figure 5, Browse.ctp File "All" Category Link

Here, the category link is labeled "All", and clicking on the link invokes the ListingsController class's index() method which displays all entries of the Listings table.

Categories View.ctp File, Image Blob and Thumbnails

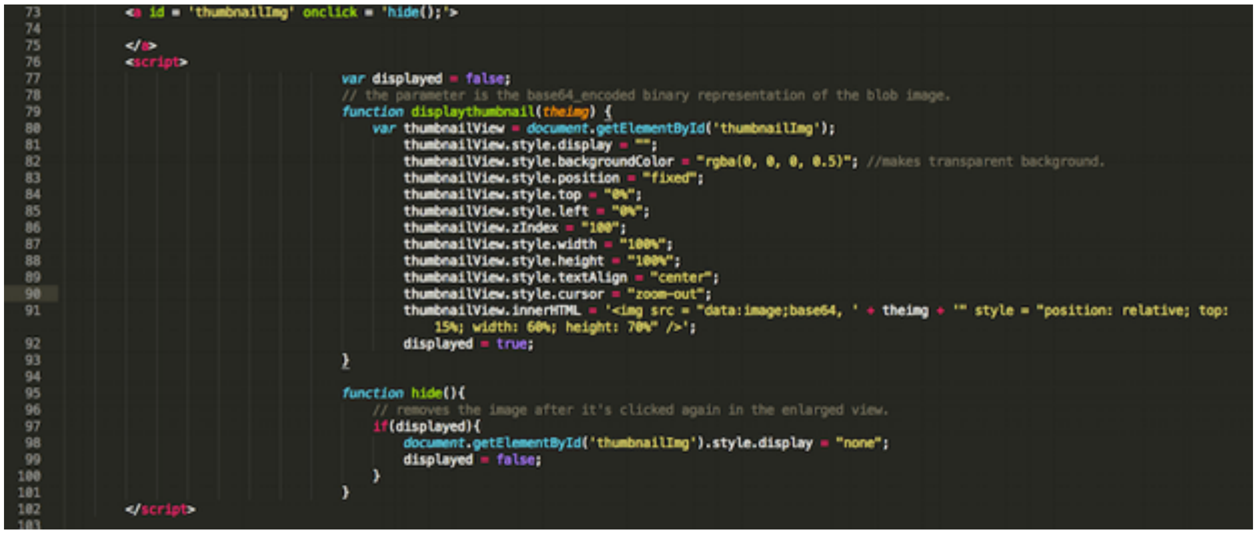
The view.ctp file associated with the Categories table is found in the src/Template/Categories folder. In this view.ctp file, the records are shown on the page using the following set of statements:



```
<?php foreach ($category->listings as $listings): ?>
<tr>
    <td><?= h($listings->listing_num) ?></td>
    <td><?= h($listings->date_created) ?></td>
    <td><?= h($listings->is_sold) ?></td>
    <td><?= h($listings->price) ?></td>
    <td><?= h($listings->location) ?></td>
    <td><?= h($listings->item_desc) ?></td>
    <td><?= h($listings->title) ?></td>
    <td><?= h($listings->category_id) ?></td>
    <td><?= h($listings->registered_user_id) ?></td>
    <td><?= h($listings->course_id) ?></td>
    <td><!-- This is how blobs are displayed on the webpage. $listings->image will return a "resource id", stream_get_contents() gets the
    contents (binary) associated with the id, and base64_encode() encodes those contents so an image will be rendered-->
        <?php if($listings->image !== null): ?>
                <?php $blobimg = base64_encode(stream_get_contents($listings->image)); ?>
                <a class = "aclass" onclick = "displaythumbnail('<?php echo $blobimg; ?>');" >
                <?= '<img src = "data:image;base64, ' . $blobimg . '" style = "width: 40px; height: 40px" />' ?>
                </a>
         <?php endif; ?></td>
```

Figure 6, Categories View.ctp File Display of Records

The h function found here is simply wraps the argument with the htmlspecialchars() php function, and returns it. The arguments shown here ae the the attributes of the given record; these attributes are accessed as the $listings variable in the foreach loop, and the records are accessed from the $category->listings variable, which is made accessible in the CategoriesController class's view() method. The LONGBLOB images are displayed by first using the base64_encode() function, which encodes the LONGBLOB with base64, and then using this encode value to render the image in the following img HTML tag. The images are presented as clickable image thumbnails, as the img tag is wrapped in an anchor elemnt (the HTML <a> tags) which allows the image to call the displaythumbnail() method when clicked. The encoded base64 value is passed to the displaythumbnail() method which enlarges the clicked image. The following shows the javascript and HTML codes that allow the accessed blob images to be represented as thumbnails (in the following page):



```
<a id = 'thumbnailImg' onclick = 'hide();'>

</a>
<script>
                        var displayed = false;
                        // the parameter is the base64_encoded binary representation of the blob image.
                        function displaythumbnail(theimg) {
                            var thumbnailView = document.getElementById('thumbnailImg');
                                thumbnailView.style.display = "";
                                thumbnailView.style.backgroundColor = "rgba(0, 0, 0, 0.5)"; //makes transparent background.
                                thumbnailView.style.position = "fixed";
                                thumbnailView.style.top = "0%";
                                thumbnailView.style.left = "0%";
                                thumbnailView.zIndex = "100";
                                thumbnailView.style.width = "100%";
                                thumbnailView.style.height = "100%";
                                thumbnailView.style.textAlign = "center";
                                thumbnailView.style.cursor = "zoom-out";
                                thumbnailView.innerHTML = '<img src = "data:image;base64, ' + theimg + '" style = "position: relative; top:
                                    15%; width: 60%; height: 70%" />';
                                displayed = true;
                        }

                        function hide(){
                            // removes the image after it's clicked again in the enlarged view.
                            if(displayed){
                                document.getElementById('thumbnailImg').style.display = "none";
                                displayed = false;
                            }
                        }
</script>
```

Figure 7, Blob Thumbnail Images

The anchor element at the top of this image is where the enlarged image will be displayed when the image is clicked. Then, the displaythumbnail() method is called, giving this anchor element its CSS properties which allows the enlarged image to be centered in the page. Moreover, these CSS properties allow the background of the image to be transparently black, and the mouse cursor to resemble the "zoom-out" cursor when the enlarged image is displayed. When setting the innerHTML element of the anchor element, the assigned value is the img tag containing the argument of the displaythumbnail() function, which again, is the base64 encoded BLOB. This img element is also given the position attribute as relative; this allows the image to be centered in the anchor element, which has a fixed position on the screen. Therefore, whenever the enlarged image is shown on the screen, it will always be in the center of the page, even though the visitor may scroll up and down the page. The statement "thumbnailView.style.display = """ allows the anchor element containing the enlarged image to be viewed again after it is hidden using the hide() method.

The hide() method is called when the anchor element is clicked on, and this sets the CSS display property of the anchor element to "none", which removes the anchor element and the enlarged image from the browser page.

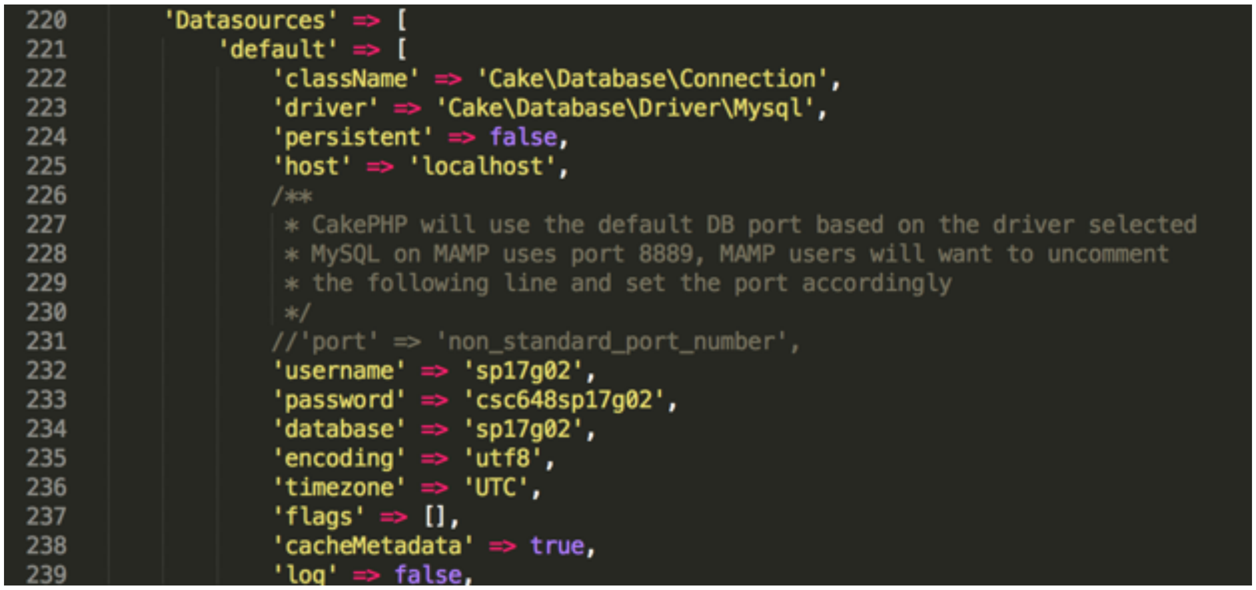
Listings Index.ctp File

Similarly, the index.ctp file associated with the ListingsController class shows the record and thumbnail images in the same manner as the view.ctp file associated with the CategoriesController class. This index.ctp uses the same foreach loop and javascript codes shown respectively in **Figure 6** and **Figure 7**, except the variable names are different.

III. Testing

Database Access Configuration

In order to test the database access in the vertical prototype, the database must be populated and configuration files must be set up properly. In the app.php file found in config folder of the project, the following must be edited:



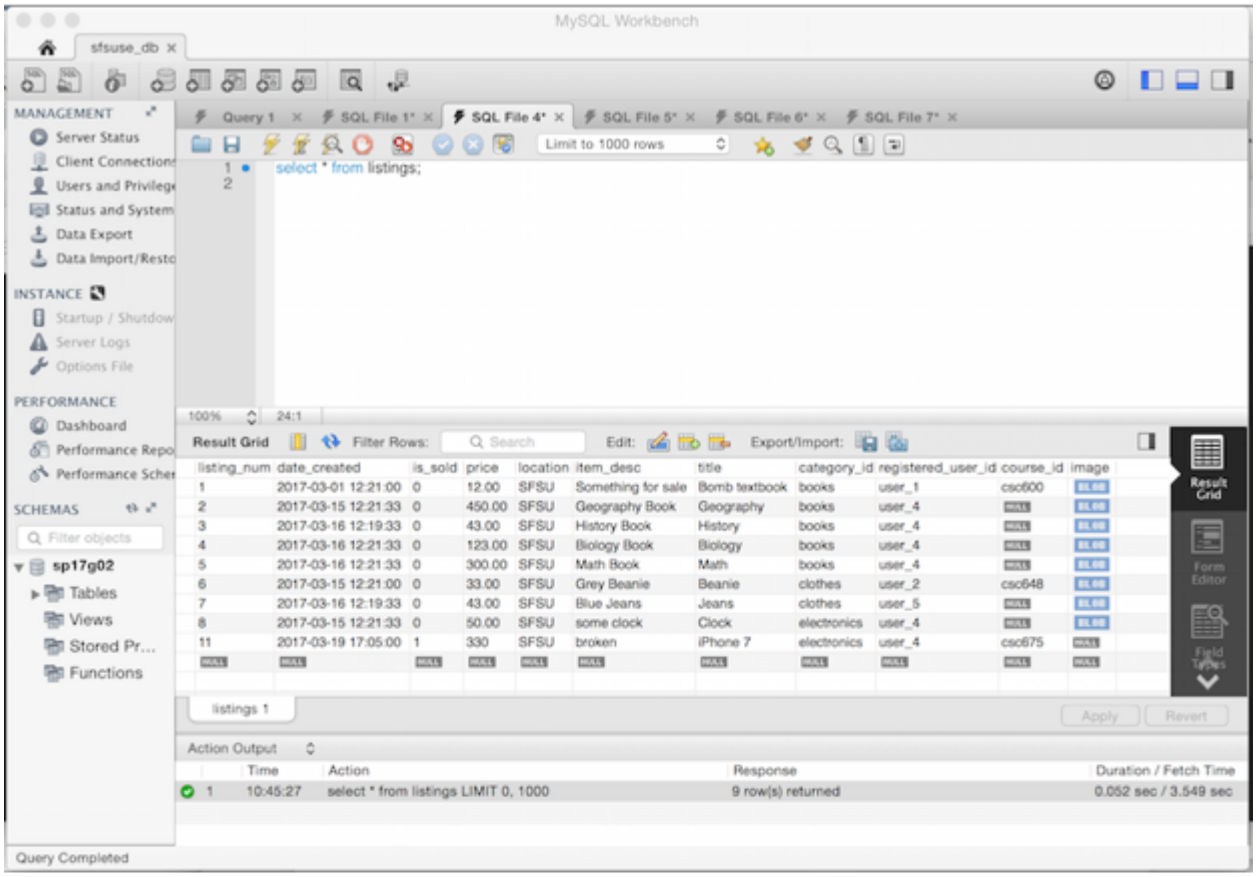
```
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            /**
             * CakePHP will use the default DB port based on the driver selected
             * MySQL on MAMP uses port 8889, MAMP users will want to uncomment
             * the following line and set the port accordingly
             */
            //'port' => 'non_standard_port_number',
            'username' => 'sp17g02',
            'password' => 'csc648sp17g02',
            'database' => 'sp17g02',
            'encoding' => 'utf8',
            'timezone' => 'UTC',
            'flags' => [],
            'cacheMetadata' => true,
            'log' => false,
```

Figure 8, App.php Datasources Configuration

Here, the username, password and database values contained in the associative array is set up so that the database in the group ssh account is accessed. However, when using the individual ssh account, or when testing the application on the local machine, the username, password and database must be changed so it matches the correct values to allow access to the targeted database.

Inserting Blob Images

Since the current vertical prototype does not allow users to upload images to the site, MySQL Workbench was used to insert the BLOB images when testing the images on the ssh account sites, the group sites, or in local machines. In using MySQL Workbench, a query must first be made to select all elements in the desired table. In this example, we used the Listings table:



```
select * from listings;
```

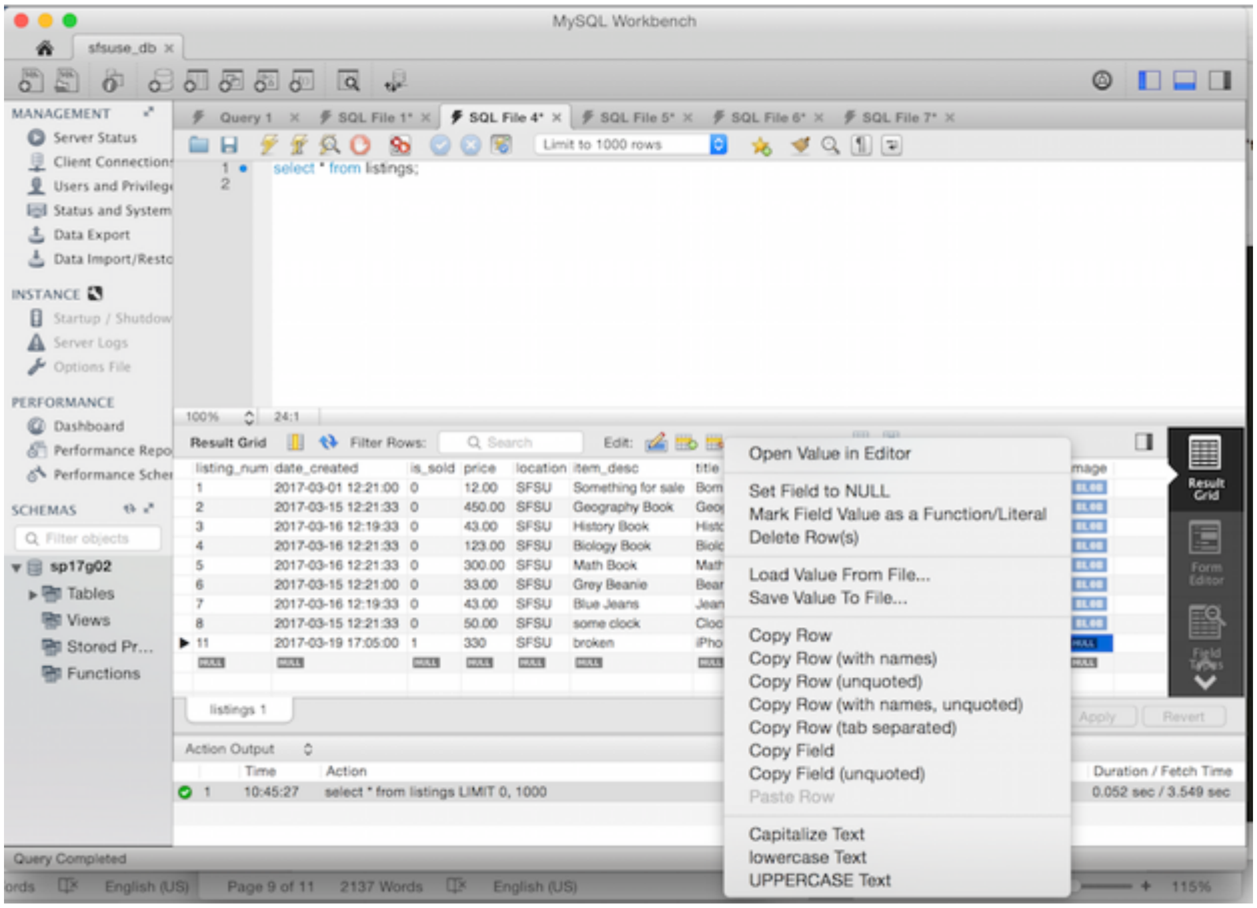
listing_num	date_created	is_sold	price	location	item_desc	title	category_id	registered_user_id	course_id	image
1	2017-03-01 12:21:00	0	12.00	SFSU	Something for sale	Bomb textbook	books	user_1	csc600	BLOB
2	2017-03-15 12:21:33	0	450.00	SFSU	Geography Book	Geography	books	user_4	NULL	BLOB
3	2017-03-16 12:19:33	0	43.00	SFSU	History Book	History	books	user_4	NULL	BLOB
4	2017-03-16 12:21:33	0	123.00	SFSU	Biology Book	Biology	books	user_4	NULL	BLOB
5	2017-03-16 12:21:33	0	300.00	SFSU	Math Book	Math	books	user_4	NULL	BLOB
6	2017-03-15 12:21:00	0	33.00	SFSU	Grey Beanie	Beanie	clothes	user_2	csc648	BLOB
7	2017-03-16 12:19:33	0	43.00	SFSU	Blue Jeans	Jeans	clothes	user_5	NULL	BLOB
8	2017-03-15 12:21:33	0	50.00	SFSU	some clock	Clock	electronics	user_4	NULL	BLOB
11	2017-03-19 17:05:00	1	330	SFSU	broken	iPhone 7	electronics	user_4	csc675	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
   Time      Action                                  Response              Duration / Fetch Time
1  10:45:27  select * from listings LIMIT 0, 1000    9 row(s) returned     0.052 sec / 3.549 sec
```

Query Completed

Figure 9, Query to Select All Records From Listings Table

Here the results are shown below the query statement after it is executed. Then, to insert the BLOB image, simply click on any entries in the "image" column and click on "Load Value From File" as such:



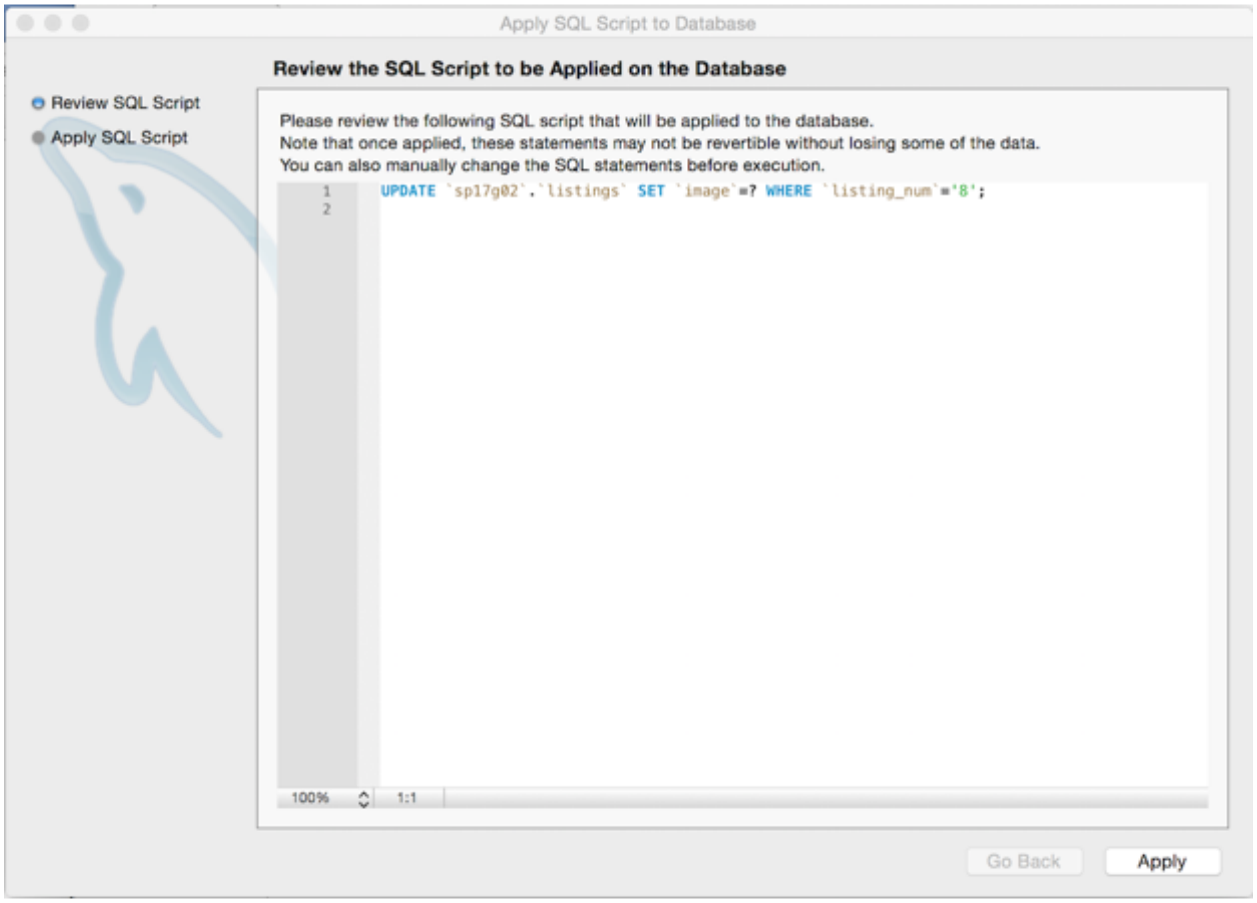
```
select * from listings;
```

listing_num	date_created	is_sold	price	location	item_desc	title	mage
1	2017-03-01 12:21:00	0	12.00	SFSU	Something for sale	Bom	BLOB
2	2017-03-15 12:21:33	0	450.00	SFSU	Geography Book	Geo	BLOB
3	2017-03-16 12:19:33	0	43.00	SFSU	History Book	Histo	BLOB
4	2017-03-16 12:21:33	0	123.00	SFSU	Biology Book	Biolo	BLOB
5	2017-03-16 12:21:33	0	300.00	SFSU	Math Book	Math	BLOB
6	2017-03-15 12:21:00	0	33.00	SFSU	Grey Beanie	Bear	BLOB
7	2017-03-16 12:19:33	0	43.00	SFSU	Blue Jeans	Jean	BLOB
8	2017-03-15 12:21:33	0	50.00	SFSU	some clock	Cloc	BLOB
11	2017-03-19 17:05:00	1	330	SFSU	broken	iPho	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
    Time      Action                                  Duration / Fetch Time
1   10:45:27  select * from listings LIMIT 0, 1000    0.052 sec / 3.549 sec
```

Figure 10, Inserting Blob Images

Once the "Load Value From File…" is selected, simply choose a file, and then click on the "Apply" button directly below the results section in the same MySQL Workbench. A new window will appear as such:


Apply SQL Script to Database
Review the SQL Script to be Applied on the Database
Review SQL Script
Apply SQL Script
100%
1:1
Go Back
Apply

Please review the following SQL script that will be applied to the database.
Note that once applied, these statements may not be revertible without losing some of the data.
You can also manually change the SQL statements before execution.

```
UPDATE `sp17g02`.`listings` SET `image`=? WHERE `listing_num`='8';
```

Figure 11, Insertion of Blob Image Statement

Finally, click on “Apply” and the record will be updated with the chosen image.

Members:

- Ajinkya Chalke (Team Lead)
- Ivan Yu (Technical Lead)
- Bradley Ng (Backend/Database)
- David Rodriguez (Quality Control, Backend/Database)
- Thao Luu (Frontend/Design)
- Jerry Auyeung (Backend/Frontend)