

Books and Tutors SFSU

SW Engineering CSC648/848 Summer2016

Milestone 4

08/5/16

Rev 1.0

Group 4

Lead: Abdullah Arikat (aarikat@mail.sfsu.edu)

Team: Alex Brown, Izaac Garcilazo, Marc Garde, Juris Puchin, Sang
Saephan

1)product summary:

Books and tutors SFSU

Books and Tutors SFSU is a site for SFSU student to sell items such as books, computers, furniture, clothing and more, directly to other SFSU students. In addition to selling items, Books and Tutors SFSU customers can use the site as an easy way to search for and offer tutoring services.

Students of SFSU interested in finding a tutor or advertising their tutoring service will no longer need to rely on a corkboard flier somewhere on campus. Books and Tutors SFSU can be used as a central place to easily search through and find the exact tutor they need, or a place to offer tutoring service themselves.

Our website will initially cater to San Francisco State University students, but as the service grows, we could expand to other schools and offer Books and Tutors UCSF, Books and Tutors UC Berkeley, and so on.

Our service (Books and Tutors SFSU):

We are dedicated to serve the students of SFSU

Nearly 30000 students attend SFSU and are always in search for books as well as other various products at a cheaper price.

We will offer the books as well as other various products listed on our site to the students of SFSU at a very competitive price

We will provide tutoring services to the students of SFSU

If a user is searching for books, we will also have a feature that will recommend tutors at SFSU that match the subject of the books that they are viewing

We believe that our unique feature (tutoring) will have a significant impact on our business plan to help serve the students of SFSU

Priority 1:	
<i>Anonymous User:</i>	
	Use the search bar to look for goods and tutors
	Search for items by category
	Create an account (if also SFSU student)
<i>User (logged in):</i>	

	All of the above plus
	Access his/her shopping cart
	Add tutoring services and items for sale to the cart
	Buy items
	Sell Items
	Order tutoring services
	Register as a tutor
	Edit items for sale
	Edit the shopping cart
	Request to become a tutor
<i>Tutor (logged in):</i>	
	All of the above plus
	Offer tutoring services
	Edit the tutoring services they offer
<i>Administrator:</i>	
	All of the above plus
	Manage the database directly
Priority 2:	
<i>Anonymous User:</i>	
	Sort items by price and date
	Get a list of tutors and items by entering a class
<i>User (logged in):</i>	
	All of the above plus

Our website:

URL: <http://sfsuswe.com/~su16g04/m3/>

2) Usability Test Plan:

Test Objectives:

Discoverability of items and tutoring services offered by Books and Tutors SFSU is an important aspect of our website. That's why the following tests focus on finding specific item or tutor a student may need.

Test Plan:

System Setup:

A desktop computer running a recent version of either Internet Explorer, Edge, Safari, Chrome, or Firefox.

Starting Point: Navigate to <http://sfsuswe.com/~su16g04/m3/>

Tasks:

1. From the webpage, search for a Database tutor
2. Search for a Database book
3. Search for a laptop
4. Search for an iClicker

Intended User: An SFSU student

Completion Criteria: User was able to search for and find an item for each step.

Questionnaire:

Please circle the answer that best describes your feelings to the following statements.

Question 1: *It was easy to find a tutor.*

Strongly Agree Agree Neutral Disagree Strongly Disagree

Comments:

Question 2: *It was easy to sell an item.*

Strongly Agree Agree Neutral Disagree Strongly Disagree

Comments:

Question 3: *It was easy to purchase an item*

Strongly Agree Agree Neutral Disagree Strongly Disagree

Comments:

Question 4: *I would recommend Books and Tutors SFSU to other SFSU students*

Strongly Agree Agree Neutral Disagree Strongly Disagree

Comments:

3) QA Test Plan

Test Objectives:

Ensure the usability and functionality of searching products and searching products with a filter.

HW and SW Setup:

Website supports web browsers (Chrome, FireFox, IE, etc). Internet connection is required to interact with website.

Feature To Be Tested:

Searching products in general and searching products with filter.

Test Cases:

To ensure the stability of the functionality and usability of the search feature, requires numerous tests. The tests started searching products in general. By default, the search bar had no filters and as a result, some results were not expected. For example, if the user searched “book”, the user expects the results to be books. However the results also included “Macbook charger”. To over this problem, the categories list was implemented. This makes the search being entered to be more precise when the user types in a more broad word like “book”. The “Books” category list ensures that only books would show up and this goes for the same for the other categories. The next series of tests includes the search of tutors. When the user types in “tutors” all tutors will be retrieved from the database. The “Tutors” category list gives a more precise search result. When a class name is typed in, tutors who are tutoring that class will show up (ie, CSC340). All in all the tests were generally expected. With the appropriate filters, the user can expect a more precise search result.

Test Cases

Test Number	Description	Test Input	Expected Output	PASS/FAIL
1	Open the webpage: http://sfsuswe.com/~su16g04/m3/ Change the category from "All" to "Books" and search with an empty search bar.	Category: Books Search: " " Sort by: "Best Match"	Products that fall in the "Books" category. 6 Results	PASS
2	Following Test #1, we change "Sort by" from "Best Match" to "Price: Lowest to Highest".	Category: Books Search: " " Sort by: Price: Lowest to Highest	Products that fall in the "Books" category by the lowest price to the highest. 6 Results	PASS
3	Open the webpage: http://sfsuswe.com/~su16g04/m3/ With the "All" category selected, enter "C++" into the search bar.	Category: All Search: "C++" Sort by: "Best Match"	Products that contain the word "C++". In addition to products, tutors that offer services related to "C++" . 3 Results	PASS
4	Following Test #3, select the "c++ introduction book" product.	Category: All Search: "C++" Sort by: "Best Match"	Tutors that offer services related to "C++" will be recommended to the user. 2 Results	PASS
5	Open the webpage: http://sfsuswe.com/~su16g04/m3/ With the "Tutors" category selected, enter "CSC340" to search for tutors related to that class.	Category: Tutors Search: "CSC340" Sort by: "Best Match"	Tutors that offer services for "CSC340". 2 Results	PASS

4) Code Review:

Every member in our group uses Netbeans IDE for this project. We agreed to use One True Brace style of formatting and change the formatting style within the IDE to match this style. In this way we can use the automated format that comes with Netbeans.





We choose One True Brace Style because we like the opening and closing brackets that this style offers. All if, else, while, and for statements have opening and closing braces, even if they are not necessary. Unlike the K&R style which omits them. The purpose is to make it easy to know where to insert new statements and know precisely how these statements will be grouped. It also makes the code easy to read.

Coding style sample:


```
60  |  /**
61  |  * ACTION: signinCustomer
62  |  * This method handles what happens when customer signs in
63  |  */
64  |  public function signinCustomer() {
65  |
66  |      if (isset($_POST["signincustomer"])) {
67  |          $email = $_POST["email"];
68  |          $password = $_POST["password"];
69  |          $salt = "saltedpass4team4";
70  |          $saltedpassword = md5($salt . $password);
71  |          $match = $this->signinmodel->signinCustomer($email, $saltedpassword);
72  |
73  |          // if user fails to login, show error message
74  |          if ($match->email == $email) {
75  |              $_SESSION['CurrentUser'] = $match->id; // create session for user
76  |              $_SESSION['UserName'] = $match->firstname;
77  |              header('location: ' . URL . 'home');
78  |          }
79  |
80  |          if ($match->email != $email) {
81  |              header('location: ' . URL . 'signin?msg=failed');
82  |          }
83  |      }
84  |  }
```


Code Review:

Team member Izaac sends the file sqlcalls.php to be review by team member Juris.

 New |  Reply |  Delete |  Archive | Move to | Categories | ...

Please code review

**izaac garcilazo**
Wed 2:27 PM
jpuchin@mail.sfsu.edu

 **sqlcalls.php**
11 KB

[Download](#) [Save to OneDrive - Personal](#)

Hi Juris please code review this file.

Sqlcalls.php

```
31 public function createInvoice($usr_id, $invoice_data) {
32     $cid = $this->getUserCart($usr_id);
33
34     //add an invoice entry to the invoice table
35     $sql = "INSERT INTO invoice (customer_id, cart_id, order_date, total, shipping_cost, tax,
36     $query = $this->db->prepare($sql);
37     $parameters = array(':uid' => $usr_id,
38         ':cid' => $cid,
39         ':date' => $invoice_data['date'],
40         ':total' => $invoice_data['total'],
41         ':ship' => $invoice_data['shipping'],
42         ':tax' => $invoice_data['tax'],
43         ':g_total' => $invoice_data['g_total']);
44     $query->execute($parameters);
45
46     //make the cart an invoice
47     $sql2 = "UPDATE cart SET name = :invoice WHERE id = :cid";
48     $query2 = $this->db->prepare($sql2);
49     $parameters2 = array(':invoice' => "invoice", ':cid' => $cid);
50     $query2->execute($parameters2);
51 }
52
53 public function addCartItem($uid, $pid, $qty) {
54     $cid = $this->getUserCart($uid);
55     $sql = "INSERT INTO cart_item (cart_id, product_id, item_qty) VALUES (:cid, :pid, :iqty)"
56     $query = $this->db->prepare($sql);
57     $parameters = array(':cid' => $cid, ':pid' => $pid, ':iqty' => $qty);
58     $query->execute($parameters);
59 }
60
```

```

61  /**
62  * Add a customer to database
63  * @param string $table table name
64  * @param array $pars parameters
65  */
66
67  public function addEntry($table, $pars) {
68      $first = True;
69      $values = "(";
70      $dot_values = "(";
71      foreach ($pars as $key => $value) {
72          if ($first) {
73              $first = False;
74          } else {
75              $values = $values . ", ";
76              $dot_values = $dot_values . ", ";
77          }
78          $dot_values = $dot_values . $key;
79          $values = $values . ltrim($key, ':');
80      }
81      $sql = "INSERT INTO " . $table . " " . $values . ") VALUES " . $dot_values . ")";
82      //echo $sql; exit();
83      $query = $this->db->prepare($sql);
84      $query->execute($pars);
85  }
86

```

```

97  public function updateEntry($table, $val, $target) {
98      $sql = "UPDATE " . $table . " SET ";
99      //Set values
100      $first = True;
101      foreach ($val as $key => $value) {
102          if ($first) {
103              $first = False;
104          } else {
105              $sql = $sql . ", ";
106          }
107          $sql = $sql . " " . ltrim($key, ':') . " = " . $key;
108      }
109      //Set target
110      $sql = $sql . " WHERE";
111      $first = True;
112      foreach ($target as $key => $value) {
113          if ($first) {
114              $first = False;
115          } else {
116              $sql = $sql . " AND ";
117          }
118          $sql = $sql . " " . ltrim($key, ':') . " = " . $key;
119      }
120      $query = $this->db->prepare($sql);
121      $pars = array_merge($val, $target);
122      // echo '[ PDO DEBUG ]: ' . Helper::debugPDO($sql, $pars); exit();
123      $query->execute($pars);
124  }
125

```

```

125  public function updateCartItem($uid, $pid, $qty) {
126      $cid = $this->getUserCart($uid);
127      $sql = "UPDATE cart_item SET item_qty = :qty WHERE cart_id = :cid AND product_id = :pid";
128      $query = $this->db->prepare($sql);
129      $parameters = array(':cid' => $cid, ':pid' => $pid, ':qty' => $qty);
130      //echo '[ PDO DEBUG ]: ' . Helper::debugPDO($sql, $parameters); exit();
131      $query->execute($parameters);
132  }
133

```

```

134 //SELECT
135
136 public function getAllEntriesAdv($table, $val, $target) {
137     $sql = "SELECT ";
138     //Set values
139     $first = True;
140     foreach ($val as $key) {
141         if ($first) {
142             $first = False;
143         } else {
144             $sql = $sql . ", ";
145         }
146         $sql = $sql . ltrim($key, ':');
147     }
148
149     $sql = $sql . " FROM " . $table;
150
151     if (count($target) > 0) {
152         //Set target
153         $sql = $sql . " WHERE";
154         $first = True;
155         foreach ($target as $key => $value) {
156             if ($first) {
157                 $first = False;
158             } else {
159                 $sql = $sql . " AND";
160             }
161             $sql = $sql . " " . ltrim($key, ':') . " like " . $key;
162         }
163     }
164     $query = $this->db->prepare($sql);
165     // echo '[ PDO DEBUG ]: ' . Helper::debugPDO($sql, $target); exit();
166     $query->execute($target);
167     return $query->fetchAll();
168 }

```

```

171 public function getAllEntriesAdv2($table, $val, $target, $column, $order) {
172     $sql = "SELECT ";
173     //Set values
174     $first = True;
175     foreach ($val as $key) {
176         if ($first) {
177             $first = False;
178         } else {
179             $sql = $sql . ", ";
180         }
181         $sql = $sql . ltrim($key, ':');
182     }
183
184     $sql = $sql . " FROM " . $table;
185
186     if (count($target) > 0) {
187         //Set target
188         $sql = $sql . " WHERE";
189         $first = True;
190         foreach ($target as $key => $value) {
191             if ($first) {
192                 $first = False;
193             } else {
194                 $sql = $sql . " AND";
195             }
196             $sql = $sql . " " . ltrim($key, ':') . " like " . $key;
197         }
198     }
199
200     $sql = $sql . " ORDER BY " . $column . " " . $order;
201     $query = $this->db->prepare($sql);
202     // echo '[ PDO DEBUG ]: ' . Helper::debugPDO($sql, $target); exit();
203     $query->execute($target);
204     return $query->fetchAll();

```



```

206 public function getInvoice($parameters) {
207
208     $sql = "SELECT customer_id, cart_id, order_date, total, shipping_cost, tax, grand_total
209     $query = $this->db->prepare($sql);
210     //echo '[ PDO DEBUG ]: ' . Helper::debugPDO($sql, $parameters); exit();
211     $query->execute($parameters);
212     return $query->fetch();
213 }

215 public function getUserCart($usr_id) {
216     $sql = "SELECT id FROM cart AS id WHERE name = :current AND customer_id = :usr_id";
217     $query = $this->db->prepare($sql);
218     $parameters = array(':current' => "current", ':usr_id' => $usr_id);
219     $query->execute($parameters);
220     $cid = $query->fetch();
221
222     if ($cid == NULL) {
223         $sql2 = "SELECT COUNT(id) AS num_carts FROM cart";
224         $query2 = $this->db->prepare($sql2);
225         $parameters2 = array(':cid' => $cid);
226         $query2->execute($parameters2);
227         $max_cid = $query2->fetch()->num_carts;
228
229         $cid = $max_cid + 1;
230
231         $sql3 = "INSERT INTO cart (id, customer_id, name) VALUES (:cid, :usr_id, :current)";
232         $query3 = $this->db->prepare($sql3);
233         $parameters3 = array(':cid' => $cid, ':usr_id' => $usr_id, ':current' => "current");
234         $query3->execute($parameters3);
235         return $cid;
236     } else {
237         return $cid->id;
238     }
239 }

241 public function getCartItems($uid) {
242     $cid = $this->getUserCart($uid);
243     $sql = "SELECT cart_id, product_id, item_qty FROM cart_item WHERE cart_id = :cid";
244     $query = $this->db->prepare($sql);
245     $parameters = array(':cid' => $cid);
246     $query->execute($parameters);
247     return $query->fetchAll();
248 }

```

```

251 public function getEntry($table, $val, $target) {
252     $sql = "SELECT ";
253     //Set values
254     $first = True;
255     foreach ($val as $key) {
256         if ($first) {
257             $first = False;
258         } else {
259             $sql = $sql . ", ";
260         }
261         $sql = $sql . ltrim($key, ':');
262     }
263
264     $sql = $sql . " FROM " . $table;
265
266     if (count($target) > 0) {
267         //Set target
268         $sql = $sql . " WHERE";
269         $first = True;
270         foreach ($target as $key => $value) {
271             if ($first) {
272                 $first = False;
273             } else {
274                 $sql = $sql . " AND";
275             }
276             $sql = $sql . " " . ltrim($key, ':') . " = " . $key;
277         }
278     }
279
280     $query = $this->db->prepare($sql);
281     // echo '[ PDO DEBUG ]: ' . Helper::debugPDO($sql, $target); exit();
282     $query->execute($target);
283     return $query->fetch();
284 }

```

```

286 public function getAllEntries($table, $val, $target) {
287     $sql = "SELECT ";
288     //Set values
289     $first = True;
290     foreach ($val as $key) {
291         if ($first) {
292             $first = False;
293         } else {
294             $sql = $sql . ", ";
295         }
296         $sql = $sql . ltrim($key, ':');
297     }
298
299     $sql = $sql . " FROM " . $table;
300
301     if (count($target) > 0) {
302         //Set target
303         $sql = $sql . " WHERE";
304         $first = True;
305         foreach ($target as $key => $value) {
306             if ($first) {
307                 $first = False;
308             } else {
309                 $sql = $sql . " AND";
310             }
311             $sql = $sql . " " . ltrim($key, ':') . " = " . $key;
312         }
313     }
314
315     $query = $this->db->prepare($sql);
316     // echo '[ PDO DEBUG ]: ' . Helper::debugPDO($sql, $target); exit();
317     $query->execute($target);
318     return $query->fetchAll();
319 }

```

```

321 public function deleteEntry($table, $pars) {
322     $sql = "DELETE FROM " . $table . " WHERE";
323     $first = True;
324     foreach ($pars as $key => $value) {
325         if ($first) {
326             $first = False;
327         } else {
328             $sql = $sql . " AND ";
329         }
330         $sql = $sql . " " . ltrim($key, ':') . " = " . $key;
331     }
332     $query = $this->db->prepare($sql);
333     $query->execute($pars);
334 }

```

Juris reads the code and sends an email to Izaac containing comments about the code.

New
Reply
Delete
Archive
Junk

From: Juris Puchin <jpuchin@mail.sfsu.edu>
Sent: Wednesday, August 3, 2016 2:52 PM
To: izaac garcilazo
Subject: Re: Please code review

Looks good!

Here are my comments.

Line 31 createInvoice()
 covert calls to addEntry() and udateEntry()
 move function to cart model

Line 53 addCartItem()
 convert call to addEntry()
 move function to cart model

Line 125 updateCartItem()
 convert call to updateEntry()
 move function to cart model

Line 215 getUserCart()
 convert calls to getEntry() and addEntry()
 move function to cart model

Line 214 getCartItems()
 convert call to getEntry()
 move function to cart model

-Juris

Izaac gets the code review and fixes the problems.
cart.php

```
2  <?php
3
4  class Cart extends Controller {
5
6      public function index() {
7          $categories = $this->homemodel->getAllCategories();
8          require APP . 'view/_templates/header.php';
9          $cart_items = $this->cartmodel->getCartItems($_SESSION['CurrentUser']);
10         $products = array();
11         foreach ($cart_items as $item) {
12             $nextProduct = $this->itemmodel->getProduct($item->product_id);
13             $nextProduct->qty = $item->item_qty;
14             array_push($products, $nextProduct);
15         }
16         require APP . 'view/cart/index.php';
17         require APP . 'view/_templates/footer.php';
18     }
19
20     public function createInvoice() {
21         if (!isset($_SESSION)) {
22             session_start();
23         }
24         if (isset($_POST["submit_create_invoice"])) {
25             $cart_items = $this->cartmodel->getCartItems($_SESSION['CurrentUser']);
26             $invoiceData = Cart::calcInvoice($cart_items, $this);
27
28             if ($invoiceData['total'] > 0) {
29                 //TODO: create notice that cart is empty
30                 $this->cartmodel->createInvoice($_SESSION['CurrentUser'], $invoiceData);
31             }
32         }
33         header('location: ' . URL . 'cart/index');
34     }
35 }
```



```

36 public function itemButton() {
37     if (!isset($_SESSION)) {
38         session_start();
39     }
40     if (isset($_POST["Update"])) {
41         $this->cartmodel->updateCartItem($_SESSION['CurrentUser'], $_POST["pid"], $_POST["qty"]);
42     } else if (isset($_POST["Delete"])) {
43         $this->cartmodel->deleteCartItem($_SESSION['CurrentUser'], $_POST["pid"]);
44     }
45     header('location: ' . URL . 'cart/index');
46 }
47
48 public function addItem() {
49     if (!isset($_SESSION)) {
50         session_start();
51     }
52     if (isset($_POST["submit_add_item"])) {
53         $this->cartmodel->addCartItem($_SESSION['CurrentUser'], $_POST["pid"], $_POST["qty"]);
54         header('location: ' . URL . 'cart/index');
55     }
56     if (isset($_POST["submit_buyitnow"])) {
57         $this->cartmodel->addCartItem($_SESSION['CurrentUser'], $_POST["pid"], $_POST["qty"]);
58         header('location: ' . URL . 'cart/index');
59     }
60 }
61
62 public function removeItem() {
63     if (!isset($_SESSION)) {
64         session_start();
65     }
66     // if we have an id of a song that should be deleted
67     if (isset($_POST["submit_delete_item"])) {
68         $this->cartmodel->deleteCartItem($_SESSION['CurrentUser'], $_POST["pid"]);
69     }
70     header('location: ' . URL . 'cart/index');
71 }
72
73 public static function calcInvoice($cart_items, $thisClass) {
74     $time_stamp = date("Y-m-d H:i:s");
75     $invoice_data = array("date" => $time_stamp, "total" => 0, "shipping" => 0, "tax" => 0, "g_total" => 0);
76     foreach ($cart_items as $item) {
77         //TODO: check if item is out of stock, decrement
78         //TODO: need to update shipping cost
79         //$invoice_data["shipping"] = 1;
80         //Add up item costs
81         $product = $thisClass->itemmodel->getProduct($item->product_id);
82         $invoice_data["total"] += $product->price * $item->item_qty;
83     }
84
85     //TODO: check proper tax value
86     $invoice_data["tax"] = $invoice_data["total"] * 0.0;
87     $invoice_data["g_total"] = $invoice_data["total"] + $invoice_data["shipping"] + $invoice_data["tax"];
88
89     return $invoice_data;
90 }
91
92 }

```

Sqlcalls.php

```
29 //INSERT
30
31 /**
32  * Add a row of values to a table in the database
33  * @param string $table table name
34  * @param array $pars parameters
35  */
36 public function addEntry($table, $pars) {
37     $first = True;
38     $values = "(";
39     $dot_values = "(";
40     foreach ($pars as $key => $value) {
41         if ($first) {
42             $first = False;
43         } else {
44             $values = $values . ", ";
45             $dot_values = $dot_values . ", ";
46         }
47         $dot_values = $dot_values . $key;
48         $values = $values . ltrim($key, ':');
49     }
50     $sql = "INSERT INTO " . $table . " " . $values . ") VALUES " . $dot_values . ")";
51     //echo $sql; exit();
52     $query = $this->db->prepare($sql);
53     $query->execute($pars);
54 }
55
56 //UPDATE
57
58 /**
59  * Update a table in database
60  * @param string $table table name
61  * @param array $val values
62  * @param array $target
63  */
64 public function updateEntry($table, $val, $target) {
65     $sql = "UPDATE " . $table . " SET ";
66     //Set values
67     $first = True;
68     foreach ($val as $key => $value) {
69         if ($first) {
70             $first = False;
71         } else {
72             $sql = $sql . ", ";
73         }
74         $sql = $sql . " " . ltrim($key, ':') . " = " . $key;
75     }
76     //Set target
77     $sql = $sql . " WHERE";
78     $first = True;
79     foreach ($target as $key => $value) {
80         if ($first) {
81             $first = False;
82         } else {
83             $sql = $sql . " AND ";
84         }
85         $sql = $sql . " " . ltrim($key, ':') . " = " . $key;
86     }
87     $query = $this->db->prepare($sql);
88     $pars = array_merge($val, $target);
89     //echo '[ PDO DEBUG ]: ' . Helper::debugPDO($sql, $pars); exit();
90     $query->execute($pars);
91 }
```

```

93 //SELECT
94
95 public function getAllEntriesAdv($table, $val, $target) {
96     $sql = "SELECT ";
97     //Set values
98     $first = True;
99     foreach ($val as $key) {
100         if ($first) {
101             $first = False;
102         } else {
103             $sql = $sql . ", ";
104         }
105         $sql = $sql . ltrim($key, ':');
106     }
107
108     $sql = $sql . " FROM " . $table;
109
110     if (count($target) > 0) {
111         //Set target
112         $sql = $sql . " WHERE";
113         $first = True;
114         foreach ($target as $key => $value) {
115             if ($first) {
116                 $first = False;
117             } else {
118                 $sql = $sql . " AND";
119             }
120             $sql = $sql . " " . ltrim($key, ':') . " like " . $key;
121         }
122     }
123     $query = $this->db->prepare($sql);
124     // echo '[ PDO DEBUG ]: ' . Helper::debugPDO($sql, $target); exit();
125     $query->execute($target);
126     return $query->fetchAll();
127 }
128

```

```

129 public function getAllEntriesAdv2($table, $val, $target, $column, $order) {
130     $sql = "SELECT ";
131     //Set values
132     $first = True;
133     foreach ($val as $key) {
134         if ($first) {
135             $first = False;
136         } else {
137             $sql = $sql . ", ";
138         }
139         $sql = $sql . ltrim($key, ':');
140     }
141
142     $sql = $sql . " FROM " . $table;
143
144     if (count($target) > 0) {
145         //Set target
146         $sql = $sql . " WHERE";
147         $first = True;
148         foreach ($target as $key => $value) {
149             if ($first) {
150                 $first = False;
151             } else {
152                 $sql = $sql . " AND";
153             }
154             $sql = $sql . " " . ltrim($key, ':') . " like " . $key;
155         }
156     }
157
158     $sql = $sql . " ORDER BY " . $column . " " . $order;
159     $query = $this->db->prepare($sql);
160     // echo '[ PDO DEBUG ]: ' . Helper::debugPDO($sql, $target); exit();
161     $query->execute($target);
162     return $query->fetchAll();
163 }

```

```

175 public function getEntry($table, $val, $target) {
176     $sql = "SELECT ";
177     //Set values
178     $first = True;
179     foreach ($val as $key) {
180         if ($first) {
181             $first = False;
182         } else {
183             $sql = $sql . ", ";
184         }
185         $sql = $sql . ltrim($key, ':');
186     }
187
188     $sql = $sql . " FROM " . $table;
189
190     if (count($target) > 0) {
191         //Set target
192         $sql = $sql . " WHERE";
193         $first = True;
194         foreach ($target as $key => $value) {
195             if ($first) {
196                 $first = False;
197             } else {
198                 $sql = $sql . " AND";
199             }
200             $sql = $sql . " " . ltrim($key, ':') . " = " . $key;
201         }
202     }
203
204     $query = $this->db->prepare($sql);
205     // echo '[ PDO DEBUG ]: ' . Helper::debugPDO($sql, $target); exit();
206     $query->execute($target);
207     return $query->fetch();
}

```

```

210 public function getAllEntries($table, $val, $target) {
211     $sql = "SELECT ";
212     //Set values
213     $first = True;
214     foreach ($val as $key) {
215         if ($first) {
216             $first = False;
217         } else {
218             $sql = $sql . ", ";
219         }
220         $sql = $sql . ltrim($key, ':');
221     }
222
223     $sql = $sql . " FROM " . $table;
224
225     if (count($target) > 0) {
226         //Set target
227         $sql = $sql . " WHERE";
228         $first = True;
229         foreach ($target as $key => $value) {
230             if ($first) {
231                 $first = False;
232             } else {
233                 $sql = $sql . " AND";
234             }
235             $sql = $sql . " " . ltrim($key, ':') . " = " . $key;
236         }
237     }
238
239     $query = $this->db->prepare($sql);
240     // echo '[ PDO DEBUG ]: ' . Helper::debugPDO($sql, $target); exit();
241     $query->execute($target);
242     return $query->fetchAll();
}

```



```

245 public function deleteEntry($table, $pars) {
246     $sql = "DELETE FROM " . $table . " WHERE";
247     $first = True;
248     foreach ($pars as $key => $value) {
249         if ($first) {
250             $first = False;
251         } else {
252             $sql = $sql . " AND ";
253         }
254         $sql = $sql . " " . ltrim($key, ':') . " = " . $key;
255     }
256     $query = $this->db->prepare($sql);
257     $query->execute($pars);
258 }

```

Izaac sends the new code back to Juris for a second review. Juris approves the new code.

Re: Please code review



Juris Puchin

Wed 9:53 PM

You

Looks great.

Ship it!

From: izaac garcilazo <izaacgarcilazo@msn.com>

Sent: Wednesday, August 3, 2016 4:31:45 PM

To: Juris Puchin

Subject: Re: Please code review

Hi Juris, I just fixed the code. It was a lot of work, but it looks great. Attached are both files. Cart and sqlcalls. Take care.

5).Adherence to original non-functional spec:

1. Application shall be developed using class provided LAMP stack(**DONE**)
2. Application shall be developed using pre-approved set of SW development and collaborative tools provided in the class. Any other tools or frameworks have to be explicitly approved by Marc Sosnick on a case by case basis.(**DONE**)
3. Application shall be hosted and deployed on Amazon Web Services as specified in the class(**DONE**)
4. Application shall be optimized for standard desktop/laptop browsers, and shall render correctly on the two latest versions of all major browsers: Mozilla, Safari, Chrome and IE. It shall degrade nicely for different sized windows using class approved programming technology and frameworks(**DONE**)
5. Data shall be stored in the database on the class server in the team's account(**DONE**)
6. Application shall be served from the team's account (**DONE**)
7. No more than 50 concurrent users shall be accessing the application at any time
8. Privacy of users shall be protected and all privacy policies will be appropriately communicated to the users. (**DONE**)
9. The language used shall be English.(**DONE**)
10. Application shall be very easy to use and intuitive. No prior training shall be required to use the website.(**DONE**)
11. Google analytics shall be added for major site functions.
12. Messaging between users shall be done only by class approved methods to avoid issues of security with e-mail services.(**DONE**)
13. Site security: basic best practices to be applied (as covered in the class)(done)
14. Modern SE processes and practices must be used as specified in the class, including collaborative and continuous SW development, using the tools approved by instructors(**DONE**)
15. The website shall prominently display the following text on all pages "SFSU/FAU/Fulda Software Engineering Project, Summer 2016. For Demonstration Only". (Important so as to not confuse this with a real application).(**DONE**)