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

Milestone 4
May 20th, 2017

History Table:

Date	May 4th, 2017
1st Draft	April 30th, 2017
Revision 1	May 20th, 2017
Revision 2	

Contents

1. Product Summary

The GatorBay is a one stop destination website for all the buying and selling needs of SFSU students. The website is designed specifically for SFSU students and provides an ecommerce platform with a beautiful, modern and responsive interface. Every aspect of the website is designed with a student centric approach. The website is currently accessible at: - http://sfsuse.com/~sp17g02/.

Students as consumers are very choosy, difficult to please and characterized to be busy. GatorBay provides a sleek UI with an accurate and fast search functionality to display users the best results. The search functionality is complemented with filters such as category, conditions, and price range filter. The website also provides sorting options. The search bar can be accessed from almost every page of the website.

The users are also given an option to view the items in a beautiful quick view so that they never have to leave the search page to see the details of the item. This not only leaves a great impression of the UI but also reduces the number of clicks required to complete the transaction.

Every effort has been made to make GatorBay a good choice to not only buy items but also sell items. Once registered, a user can sell an item with very few clicks, thus opening up a massive community of buyers in no time. The simple and very easy to understand interface requires absolutely no training for posting an item for sale on the website. Being said that, it is important to note that GatorBay requires sellers to mention the location of pickup of the item. This is then in turn shown to the users on Google Maps. But to make this experience seamless, GatorBay also makes use of the Google's Places API to autocomplete the address which also reduces the time required by the seller to enter the address. GatorBay displays an approximate location on map thus making sure that privacy is not breached. It also provides a default location option of SFSU. Many of these features are unique to the GatorBay website.

These features will enable GatorBay to become an excellent hotspot for students to reach out to fulfill all of their buying and selling needs.

List of P1 features: -

All users: -

1. All users will be able to sort the listings based on price and date on which the listings were posted on the website.
2. All users will be able to search for listings using keywords.
3. All users will be able to use the QuickView functionality, which enables the user to view a summary of a chosen item. This summary is displayed in a modal on the same page.
4. All users will be able to see the pickup location of the listing in Google Maps on the details page.
5. Any user will be able to filter the listings based on the condition and price of the items.

Registered Users: -

6. Registered users will be able to message the sellers in order to connect with the them for further transactions.
7. Only registered users will be allowed to list an item for selling on the website.
8. These users will be able to see all the items they are selling in one page.
9. When registered users are trying to list an item for selling they will be able to provide the pickup location using Google's autocomplete address bar.
10. When listing an item for selling, these users will be able to upload multiple images. These images are then displayed on the details page to all the users.
11. A registered user will also be able to edit the price, description, title and condition of the listing which he/she is selling.

2. Usability Test Plan

Test Objective:

The objective of the usability test plan is to test the ease and efficiency of using the site's search function and the effectiveness of searches performed. The ease, efficiency and effectiveness of the search function will be recorded in the Likert scale questionnaire provided in this documentation.

The measure of effectiveness of the performed searches will depend on the correctness of the results generated after the user has attempted to search for items falling in specific categories and/or search keywords. In this test plan, a performed search's effectiveness will depend on the user being able to find the items he/she has intended to search and if the user has been given clarity in the search results. Clarity in search results include the user being notified that the search yielded no results. These measurements will ultimately be recorded by the Likert scale questionnaire.

The measure of efficiency in performing search will depend on how long it took for the user to complete a search since the time the user has thought of items to search for and has access to the search function. Some variables that will affect this measurement in efficiency are: how quickly the user can locate the search function, and how quickly the user can learn how to use the search function.

Test Plan:

The website to be tested can be found at http://sfsuse.com/~sp17g02/. The system setup for this usability test plan involves the following: several items have been posted in the GatorBay site to be searched for, and the GatorBay homepage is loaded in the user's browser, and is served through the group website. Before the user has attempted any search and before the beginning of the usability test plan, the GatorBay site has already been populated by many items available to be searched for. The test plan begins once the users visit the GatorBay site and the system setup has completed. The starting point of the test is as follows: the user is in the homepage of the GatorBay site, and no search has taken place. In this test plan, users will perform a search on the GatorBay site. The users will perform search to find items belonging to categories and related to keywords.

The users targeted for this test plan are college students who have not used the GatorBay site. Making the test available to users who have not used the GatorBay site prior to the test gives a more accurate result in testing the usability of the site's search function. Doing so will enable variables previously described - such as the user's ability to quickly locate and learn to use the search function - that will provide a more accurate representation on the ease, effectiveness and efficiency in using the search function. Since the search function is available for users who are not registered, the users in this test are not required to have registered account with GatorBay. The population of users in this test will include users who either lacks experience, have some experience or who are very experienced in using search functions on other websites.

The successful completion criteria includes the users being able to find items that the user has intended to search for, or in the event that the item did not exist in the site, the users being given clarity on their search result. The benchmark for this usability test is 1 search completed in 15 seconds, and in overcoming this benchmark through the test, the time in which the user is deciding what to search for will not be counted towards the time it took to complete the searches.

The following summarizes the task description of the usability test plan for the search function:

Task	Description
Task	Perform search on the GatorBay site.
Machine state	GatorBay site homepage loaded on the user's browser.
Successful completion criteria	User found items intended for the search if available, or has been given clarity on a search resulting in no items.
Benchmark	1 search in 15 seconds.

Figure 1: Task Description of Search Function Usability Test

Likert Scale Questionnaire:

After performing the usability test plan, the user is provided with the Likert Scale Questionnaire in **Figure 2**. This form records users' opinion on several variables that determine the effectiveness, ease and efficiency in using the site's search function. The user is provided 5 options for each statement, and each option show how much the user agrees or disagrees with the corresponding statement: Strongly Agree, Agree, Undecided, Disagree and Strongly Disagree. The user can The user can check only 1 box per statement and have only 1 answer per statement.

Statement	Strongly Agree	Agree	Undecided	Disagree	Strongly Disagree	Comments
1. The site's feature that allows the search function was easy to find and use.						
2. The search function was easy to use.						
3. The search function was easy to learn.						

Figure 2: Likert Scale Questionnaire for Search Function Usability Test Plan

3. QA Test Plan

Test Objectives

The objective of search function QA test is to check whether the GatorBay site's current search function behave as expected according to the specs planned by the group, and to find any bugs associated with the search function if any exists. Only features and the software of the search function will be tested in this QA test. The search function is expected to perform according to the following specs:

- The search function must return items whose item descriptions and title contain the search field inputs provided by the user.
- The search function must only return items belonging to the category input given by the user.
- The search function must return items corresponding to the item descriptions and title only, if the category selected is "All Categories".
- The search function must treat the search text input field value as separate keywords separated by space, and the resulting items are those that have any of these keywords in its title or item description.
- The search function will return all item corresponding to the selected category if the search input field is blank or if only whitespaces are included in the search input field.
- The search function must return all items contained in the GatorBay site in the event that the user's search inputs yield no results.
- The search function must account for erroneous inputs, notifying the user of the invalid inputs.

In the event the search function does not behave in any way the specs has described, then a bug exists in the search function.

Hardware and Software Setup

The QA test will be accomplished in a Mac laptop, using group's site. The group site will serve the GatorBay web application in which the QA testing of the search function will take place. The site will be visited using a Chrome browser, and then the same test will be run again using the Firefox browser.

The information of each item in the GatorBay site will be served from the group's database provided by the sfsuse.com server. This database will be populated with more entries corresponding to the items to be displayed in the GatorBay site, as shown in **Figure 3**. These items are represented by rows in the "Listings" table of the database, and each item row contains a title, description and category, which are the values checked during a search.

Features To Be Tested

It is important to note that in this QA test, the search results are generated through the user's inputs - search keywords and category options - which are respectively collected in a search input field and a category dropdown menu. While the search function and its entirety will be tested on how it performs according to the specs previously described in the test objectives, the search function's features that will be tested and are responsible for allowing the search function to behave according to the specs are the following:

- The search function's ability to correctly use the search input field's values and the category dropdown option's values to arrive at the correct results.
- The search function's ability to detect erroneous inputs - such as special characters in the search input text field - and trigger the appropriate message to warn the user of such errors.

In the QA test, entering values in the search input text field and choosing from the category drop-down menu, and then executing the search should result in the search function behaving in any of the ways described by the specs. The search function's ability to use the inputs provided by the search text field and the category drop-down menu will depend on how the search function is able to send the input to the built-in controller (the ListingsController) of the application, and how the controller is able to access the database in order to fetch the correct items corresponding to the search inputs. If the search function fails to perform in any of the ways described by the spec, then such test result indicates that a bug exists in the search function.

Actual Test Cases

The QA tests will operate on the following items which have been stored in the database:

Item #1:	Title: Computer Science | Item Description: Algorithms Book | Category: Book
Item #2:	Title: Computer Club Shirt | Item Description: Computer Shirt | Category: Clothes
Item #3:	Title: Mac | Item Description: 2013 Version | Category: Electronics
Item #4:	Title: Windows Computer | Item Description: 2010 Version | Category: Electronics

Figure 3: Title, Item Description And Category of Each Item Stored In Database And Used For QA Test

The following describes 3 different test cases, the inputs used for each test case, as well as the expected outputs, and if the test has passed on the Chrome browser:

Test #	Test Title	Description	Test Input	Expected Output	PASS /FAIL
1	Blank Search Text Test	Search function should read blank text input and return all items of specified category.	Search Text: (Blank) Category Option: All Categories	Items 1, 2, 3 & 4	PASS
2	Category and Search Text Test	Search function returns items of chosen category, and with search text in its title and description.	Search Text: Windows Mac Category Option: Electronics	Items 3 & 4	PASS
3	Error Test	Search function yields error result.	Search Text Input: @!!!#$% Category Option: Book	Message indicating erroneous input. No search takes place	PASS

Figure 4: QA Test Results On Chrome Browser

The following shows the QA test results when the Firefox browser was used instead:

Test #	Test Title	Description	Test Input	Expected Output	PASS /FAIL
1	Blank Search Text Test	Search function should read blank text input and return all items of specified category.	Search Text: (Blank) Category Option: All Categories	Items 1, 2, 3 & 4	PASS
2	Category and Search Text Test	Search function returns items of chosen category, and with search text in its title and description.	Search Text: Windows Mac Category Option: Electronics	Items 3 & 4	PASS
3	Error Test	Search function yields error result.	Search Text Input: @!!!#$% Category Option: Book	Message indicating erroneous input. No search takes place	PASS

Figure 5: QA Test Results On Firefox Browser

4. Code Review

The team is using the snake_case coding style, where the variables contain an underscore between the different words in the variable name. Names of variables that refer to the database tables are named the same as the tables. Codes contained within blocks are indented with 1 tab after the starting line of the enclosing block.

A sample of code review for the search functionality has been included on the next page.

The following **Figure 6** shows the peer code review for the site's search function:

Re: Search code for code review

IY Ivan Wesley Yu
Today, 4:29 PM
David Rodriguez <drodri12@mail.ccsf.edu>

Reply all

Sent Items

Hey, the codes look good overall. We just need to add more comments explaining other parts of the function and change a couple of the variables' names so that the names define what the variable does. We also just need to change the indentation on a couple of if-blocks, and long statements that take up multiple lines, so that these statements are indented the same way as blocks (e.g. function arguments that take up multiple lines should be indented so that its vertically aligned with the argument in the first line). Doing all of this will make it easier to debug the codes later on if we need to.

In terms of code efficiency, the codes are good, and my only suggestion to improve the efficiency is to remove some unnecessary codes. There are only a couple of lines of codes that needs to be taken out in order to do this.

Overall, the codes are great, we just need to implement these minor changes to improve the code quality. In the following, the comments labeled with "CodeReview: " explain suggested changes:

```
 public function index()
  {
      // CodeReview: explain setDefaultData()
      $this->setDefaultData();
      $filtered_listings = NULL;
      $list_was_filtered = false; // True if user filtered by category.
      $no_results_found = false; // True if no results were found.
      if (!empty($this->request->query) && $this->request->query['tags'] !== NULL) {
          $tags = preg_split('/[\s,]+/', trim($this->request->query['tags']));
          // Get all listings with titles/item desc that contain the tags.  The first
          // tag is found out of the loop so that the 'orWhere' method can be applied.
          $j = 0;    // CodeReview: delete because $j is eventually set to 1 regardless.
          $n = count($tags);
//CodeReview: change name $n to $number_of_search_keywords or something else defining what $n is.
          $filtered_listings = $this
                              ->Listings
                              ->find()
                              ->distinct(['listing_num'])
                              ->where(['OR' => [
                                      ['item_desc LIKE' => "%{$tags[$j]}%"],
                                      ['title LIKE' => "%{$tags[$j]}%"]]]);
          //CodeReview: changed indentation in previous statement.

          $j = $j + 1;   // CodeReview: change to $j = 1 instead, since $j = 0 should be deleted.
          while ($j < $n) {
              $filtered_listings = $filtered_listings
                                  ->orWhere(['OR' => [
                                          ['item_desc LIKE' => "%{$tags[$j]}%"],
                                          ['title LIKE' => "%{$tags[$j]}%"]]]);
          // CodeReview: indentation changed in previous statement.
              $j = $j + 1;
          }
      // added 4/16/17. this will allow clicking categories in
browse page to only list items in the same categories.
      // if category ever appears in the query string (not
category_filter, which narrows the search for the search bar instead)
      // then we know that only items of that category should
appear, since a category link from the browse page has been clicked.
      } else if($this->request->query['category']!== NULL) {
          $filtered_listings =$this
                              ->Listings
                              ->find('all')
                              ->where(['category_id' =>$this->request->query['category']]);
          // CodeReview: changed indentation on previous statement
          $list_was_filtered = true;
          $this->set('default_category', $this->request->query['category']);
      }

     // CodeReview: Add comments below as needed. Example comments shown.
      $contain = ['RegisteredUsers', 'Courses', 'Conditions', 'Categories'];
      $conditions = [];
      $item_conditions = [];
      $condition_filters = []; // Stores the conditions that are checked.
      $get_request = $this->request->query;
      if (!empty($get_request['category_filter'])) {
          $conditions['Categories.category_name'] = $get_request['category_filter'];
          // Set the category the user selected so that it can stay selected
          // when a new page is accessed.
```

```
            $this->set('default_category', $get_request['category_filter']);
        }

    // CodeReview: Remove this since we no longer use courses. Might need to change other codes in other files to account for this.
        if (!empty($get_request['course'])) {
            $conditions['Courses.course_name'] = $get_request['course'];
        }

    // CodeReview: Example comment: "this checks if an input for an item condition of new is sent from user's request. If so, the if-block sets the item condition to new and the condition filter will be set so only items that have a new condition will result from search"
        if (!empty($get_request['condition_like_new'])) {
            $item_conditions[] = 'like_new';
            $condition_filters['like_new'] = true;
        }
        if (!empty($get_request['condition_new'])) {
            $item_conditions[] = 'new';
            $condition_filters['new'] = true;
        }
        if (!empty($get_request['condition_good'])) {
            $item_conditions[] = 'good';
            $condition_filters['good'] = true;
        }
        if (!empty($get_request['condition_fair'])) {
            $item_conditions[] = 'fair';
            $condition_filters['fair'] = true;
        }
        if (!empty($get_request['condition_poor'])) {
            $item_conditions[] = 'poor';
            $condition_filters['poor'] = true;
        }
        if (!empty($get_request['price']) && $get_request['price']==6) {
            $conditions['Listings.price > ']=0;
        }
        else if (!empty($get_request['price']) && $get_request['price']!==6) {
            // The 'price' element is between 1 and 5.
            $price_max = 25.0 * ((double) $get_request['price']);
            $conditions['Listings.price >= '] = $price_max - 25.0;
            if($price_max === 100.0){
                $price_max = $price_max - 1;
            }
            if ($price_max < 100.0) {
                $conditions['Listings.price < '] = $price_max;
            }
        }
        if (count($item_conditions) > 0) {
            $conditions['Conditions.condition_name IN'] = $item_conditions;
        }
        $conditions['Listings.is_sold'] = 0; // Only show non-sold listings.

    // CodeReview: Example comment: "this sets the conditions for the paginated query and sets the order of the query such that it is in descending order in terms of when the listing was created"
        $this->paginate = ['contain' => $contain,
                           'conditions' => $conditions,
                           'order' => ['Listings.date_created' => 'desc']];
    // CodeReview: changed indentation on previous statement.
        if ($list_was_filtered!==true && (empty($filtered_listings) || ($filtered_listings->count() == 0))) {
            // CodeReview: Example Comment: "This sets the $listings variable as the queried result based on conditions set in $this->paginate previously"
            $listings = $this->paginate($this->Listings);
            $no_results_found = true;
        }
        else {
            $listings = $this->paginate($filtered_listings);
        }
        $this->set('condition_filters', $condition_filters);
        $this->set('no_results_found', $no_results_found);
        $this->set(compact('listings'));
        $this->set('_serialize', ['listings']);
    }
```

From: David Rodriguez <drodri12@mail.ccsf.edu>
Sent: Thursday, May 4, 2017 2:41:04 PM
To: Ivan Wesley Yu
Cc: Ajinkya Sanjay Chalke
Subject: Search code for code review

Hey, this is the code that performs searches.

```
        $contain = ['RegisteredUsers', 'Courses', 'Conditions', 'Categories'];
        $conditions = [];
        $item_conditions = [];
        $condition_filters = []; // Stores the conditions that are checked.
        $get_request = $this->request->query;
        if (!empty($get_request['category_filter'])) {
            $conditions['Categories.category_name'] =
$get_request['category_filter'];
            // Set the category the user selected so that it can stay selected
            // when a new page is accessed.
            $this->set('default_category', $get_request['category_filter']);
        }
        if (!empty($get_request['course'])) {
            $conditions['Courses.course_name'] = $get_request['course'];
        }
        if (!empty($get_request['condition_like_new'])) {
            $item_conditions[] = 'like_new';
            $condition_filters['like_new'] = true;
        }
        if (!empty($get_request['condition_new'])) {
            $item_conditions[] = 'new';
            $condition_filters['new'] = true;
        }
        if (!empty($get_request['condition_good'])) {
            $item_conditions[] = 'good';
            $condition_filters['good'] = true;
        }
        if (!empty($get_request['condition_fair'])) {
            $item_conditions[] = 'fair';
            $condition_filters['fair'] = true;
        }
        if (!empty($get_request['condition_poor'])) {
            $item_conditions[] = 'poor';
            $condition_filters['poor'] = true;
        }
        if (!empty($get_request['price']) && $get_request['price']==6) {
            $conditions['Listings.price > ']=0;
        }
        else if (!empty($get_request['price']) && $get_request['price']!==6) {
            // The 'price' element is between 1 and 5.
            $price_max = 25.0 * ((double) $get_request['price']);
            $conditions['Listings.price >= '] = $price_max - 25.0;
```

```
            if($price_max === 100.0){
                $price_max = $price_max - 1;
            }
            if ($price_max < 100.0) {
                $conditions['Listings.price < '] = $price_max;
            }
        }
        if (count($item_conditions) > 0) {
            $conditions['Conditions.condition_name IN'] = $item_conditions;
        }
        $conditions['Listings.is_sold'] = 0; // Only show non-sold listings.
        $this->paginate = ['contain' => $contain,
                           'conditions' => $conditions,
                           'order' => ['Listings.date_created' => 'desc']];
        if ($list_was_filtered!==true && (empty($filtered_listings) ||
($filtered_listings->count() == 0))) {
            $listings = $this->paginate($this->Listings);
            $no_results_found = true;
        }
        else {
            $listings = $this->paginate($filtered_listings);
        }
        $this->set('condition_filters', $condition_filters);
        $this->set('no_results_found', $no_results_found);
        $this->set(compact('listings'));
        $this->set('_serialize', ['listings']);
    }
```

Figure 6: Peer Code Review for Search Function

5. Self-check on best practices for Security

The major assets being protected in the sites are:

- User's username, email and password
- User's posted item information

To enforce security in the GatorBay site, passwords given when registering are encrypted prior to their storage in the database. Similarly, logging in will first encrypt the password and then compare with the encrypted password in the database to authenticate the user. When typing in a password, whether for registration or for logging in, the password is not shown on the screen; rather, it is masked by dots.

Features of the site that accepts user inputs, such as the search bar, are also implemented so that the site is protected from SQL injection attacks, attacks where the attacker may use SQL statements as input in order to gain access to or change data the site is intended to protect. To allow this protection, input fields for the search bar is designed such that it will not accept any special characters (such as "=", "$", etc.) and therefore, the attacker may not give SQL statements as inputs. Search input can only contain letters and numbers. If the input contains any other symbols, the input is rejected and a message is displayed notifying about the invalid input. SQL injection prevention techniques are also implemented in other input fields, namely those that are required when creating a listing and when providing an email and username during registration (and login in the case of username).

In addition to only allowing letters and digits as search input, a maximum character length has been enforced. Search input can be no longer than 30 characters, which further aids in the fight against SQL injection. This is true for the inputs for username.

Finally, items can be edited or deleted only by registered users who are the owners of such items.

6. Adherence to Original Non-functional Specs

Application shall be developed using class provided LAMP stack.	On Track
Application shall be developed using the pre-approved set of SW development and collaborative tools provided in the class. Any other tools or frameworks must be explicitly approved by Anthony Souza on a case by case basis.	Done
Application shall be hosted and deployed on Amazon Web Services as specified in the class.	Done
Application shall be optimized for standard desktop/laptop browsers and must render correctly on the two latest versions of all major browsers: Mozilla, Safari, Chrome.	On Track
Application shall have responsive UI code so it is adequately rendered on mobile devices but no mobile native app is to be developed.	On Track
Data shall be stored in the MySQL database on the class server in the team's account.	Done
Application shall be served from the team's account.	Done
No more than 50 concurrent users shall be accessing the application at any time.	Done
Privacy of users shall be protected and all privacy policies will be appropriately communicated to the users.	Done
The language used shall be English.	Done
Application shall be very easy to use and intuitive. No prior training shall be required to use the website.	On Track
Google analytics shall be added.	Done
Messaging between users shall be done only by class approved methods to avoid issues of security with e-mail services.	Done
Pay functionality (how to pay for goods and services) shall not be implemented.	Done
Site security: basic best practices shall be applied (as covered in the class).	Done
Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development.	Done

The website shall prominently display the following text on all pages *"SFSU Software Engineering Project, Spring 2017. For Demonstration Only"*. (Important so as to not confuse this with a real application).	Done