PrimeEstate Project Specifications

SW Engineering CSC648/848 Fall 2014

Milestone #4

Group #3

Jonathan Olson (jolson@sfsuswe.com)

Alex Lyashevych (alyash@sfsuswe.com)

Dong Li (xli@sfsuswe.com)

Rushab Indi (rindi@sfsuswe.com)

Yi-Hsien Chen (ychen@sfsuswe.com)

Date of Revision	Revision	Version
12/12/2014	Initial submission	1.0
12/15/2014	Revision	

Table of Contents

1. Product Summary

PrimeEstate provides a unique user centered approach to the Real Estate buying and selling market. In addition to the core features offered by other prominent Real Estate websites, **PrimeEstate** will provide a number of features and tools that help match the customers with their dream home. The home buyer's guide, customer profile, and clean aesthetics following sound design principles all contribute to the unique **PrimeEstate** experience. The target audience is the home buyer or seller from any walk of life. The website will guide the novice user and provide features that keep the expert user engaged.

Website location: http://sfsuswe.com/~f14g03/

Deliverables:

1. **Users** shall be able to search by city and zip codes.
2. **Users** shall be able to register as a customer.
3. **Customers** shall be able to enter their contact information.
4. **Customers** shall be able to contact agent to express interest in a property.
5. **Customers** shall be given to option to not receive notifications
6. **Customers** shall have personal profile kept private and unaccessible from third parties
7. **Realtor** shall be able to post listings.
8. **Realtor** shall be able to edit their listings.
9. **Realtor** shall be able to delete their listings.
10. **Realtor** shall be able to view list of interested customers.
11. **Administrators** shall be able to remove **realtor** accounts.
12. **Administrators** shall be able to add **realtor** accounts.

13. **Customers** shall be able to review **listings** which match their **profile** and **listings** for which they have expressed interest.

14. **Favorite** feature where **customers** can add **listings** they are interested in, to a special list of properties they can review at a later time through their **profile**.

15. **Realtor** and **Customer** shall be able to contact the **administrator** (have e-mail posted)

16. Map feature - Using open street/google API the **listings**' address will be mapped in the **listing**

17. Images for a **listing** shall be displayed in a gallery

2. Usability Test Plan

2.1 Test Objectives

The goals of usability testing include establishing a baseline of user performance, establishing and validating user performance measures, and identifying potential design concerns to be addressed in order to improve efficiency, productivity, and user satisfaction. The participants will take part in the usability test using a supported browser. The participant's interaction with the website will be monitored by the facilitator seated in the same area. Objective measures include time taken to complete the objective, and number of errors.

2.2 Test Plan

System Setup	User will be logged into a Realtor account
Starting Point	http://sfsuswe.com/~f14g03/views/add_listing.php
Task to be accomplished	Post a listing on the PrimeEstate website
Intended User	Realtor
Completion criteria	New Listing is posted.
URL to be tested	http://sfsuswe.com/~f14g03/views/add_listing.php

2.3 Questionnaire

Please rate your experience with the PrimeEstate website. Circle the response that best represents your feelings.

1. It was easy to add a listing using the PrimeEstate website.

Strongly Agree Agree Neutral Disagree Strongly Disagree

2. Following the steps to post a listing was confusing.

Strongly Agree Agree Neutral Disagree Strongly Disagree

3. Uploading a photo was challenging.

Strongly Agree Agree Neutral Disagree Strongly Disagree

4. I would recommend PrimeEstate to a friend.

Strongly Agree Agree Neutral Disagree Strongly Disagree

4. I like the way my posting looks on the page.

Strongly Agree Agree Neutral Disagree Strongly Disagree

3. Quality Assurance Test Plan

The purpose of testing is to ensure the PrimeEstate website meets all of the technical and functional requirements.

3.1 Test Objectives

- Verify software requirements are complete and accurate
- Perform detailed test planning
- Identify testing standards and procedures that will be employed on the project
- Prepare and document test cases
- Provide test metrics

3.2 Hardware & Software Setup

- Computer will be running

3.3 Feature to be tested

The test cases below will test the add listing function for the PrimeEstate website. It is written in a way that targets the average user.

3.4.1 Test Cases

The following is a list of steps that construct the correct testing environment. Before each test please complete the following:

1. Navigate to the PrimeEstate website located at http://sfsuswe.com/~f14g03.

2. Click ‘sign in’ in the upper right hand corner and sign in as a registered realtor.

3. Once the page loads click ‘Add Listing’ in the header.

Please circle either Pass or Fail based on the outcome of the following input conditions.

Test Case	Input	Output	Results	
1.1	Type 100 in Price field and click ‘Add Listing’ button.	“Please fill out this field.” appears below Address field.	PASS	FAIL
1.2	Type beta in Address and click ‘Add Listing’ button.	“Please fill out this field.” appears below Price field.	PASS	FAIL
1.3	Type beta in City and click ‘Add Listing’ button.	“Please fill out this field.” appears below Price field.	PASS	FAIL
1.4	Type 94112 in Zip and click ‘Add Listing’ button.	“Please fill out this field.” appears below Price field.	PASS	FAIL
1.5	Type beta in Rooms and click ‘Add Listing’ button.	“Please fill out this field.” appears below Price field.	PASS	FAIL
1.6	Type beta in Bathrooms and click ‘Add Listing’ button.	“Please fill out this field.” appears below Price field.	PASS	FAIL
1.7	Type beta in Description and click ‘Add Listing’ button.	“Please fill out this field.” appears below Price field.	PASS	FAIL

3.4.2 End to End Test

Test Case	Input	Output	Results	
2.1	Type values in for **all** fields based on 3.4.1 and click 'Add Listing' button.	Page directs you to upload.php to add images to the new listing. Output on screen must be - "Step 2: Please select the image you with to upload"	PASS	FAIL
2.2	Type values in for **all** fields based on 3.4.1 and click 'Add Listing' button.	Page directs you to upload.php to add images to the new listing. Step updates to 'Upload Images' and 'Enter Details' must be as complete, in the color green.	PASS	FAIL
2.3	Upload an image to the listing (.jpeg or .png)	Page directs to uploadcomplete.php and the image upload is successful. The uploaded image is prominently visible on screen.	PASS	FAIL

Test Case	Input	Output	Results	
2.4	On this page, click on “Upload another image” and add an image to the listing (.jpeg or .png).	Page directs to uploadcomplete.php and the image upload is successful. The uploaded image is prominently visible on screen.	PASS	FAIL
2.5	Upload a pdf to the listing.	Page directs you to upload.php to add images to the new listing. Step updates to ‘Upload Images’ and ‘Enter Details’ must be as complete, in the color green.	PASS	FAIL
2.6	Type beta in Description and click ‘Add Listing’ button.	“Please fill out this field.” appears below Price field.	PASS	FAIL

4. Peer Code Review

The following is a submission for code review between Rushab and Alex.

4.1 Rushab's submission to Alex

Subject: Code Review for Listings Controller
From: "Rushab Indi" <rindi@sfsuswe.com>
Date: Thu, December 11, 2014 10:01 pm
To: alyash@sfsuswe.com
Cc: "Jonathan Olson" <jolson@sfsuswe.com>
Priority: Normal
Options: View Full Header | View Printable Version | Download this as a file

```
Hi Alex,

Could you go over the attached code for the listings controller? Feel
free to optimize the code and give your comments to make the code clearer.

Thanks,
Rushab
```

Attachments:

listings_controller.php	9.6 k	[text/php]	Download | View

Contents of listings_controller.php:

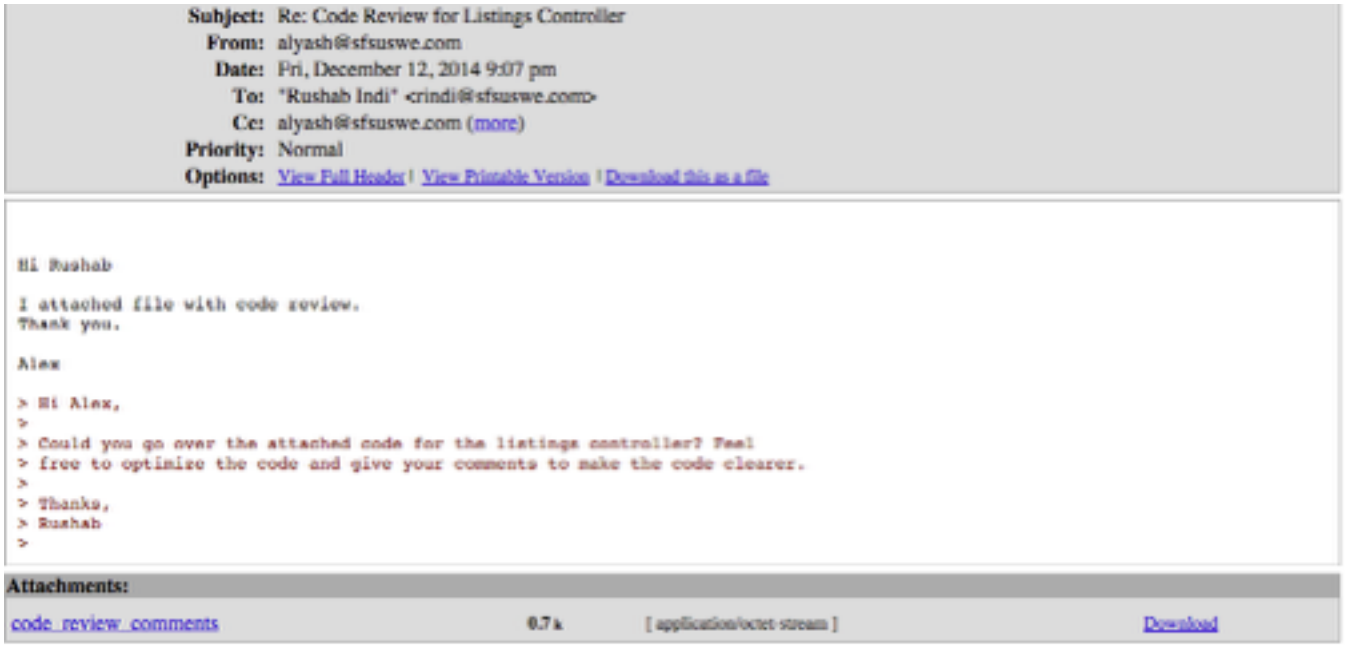
```php
<?php

require_once ("../controllers/controller.php");
require_once ("../models/listing_model.php");
require_once ("../models/profile_model.php");

/**
 * Listings Controller class
 */
class listings_controller extends controller {

    /**
     * Constructor
     */
    public function __construct() {
        parent::__construct();
    }

    /**
     * Search listings
     * @param type $input
     * @return \listing_model
     */
    public function searchListings($input) {
        $check = -1;
        $dataSet = array();
        if (strlen($input) == 0) {

            return NULL;
        }
        if (ctype_alpha(str_replace(' ', '', $input))) {

            $check = 1;
        } else if (ctype_digit(str_replace(' ', '', $input))) {

            $check = 0;
        } else {
            $check = 3;
        }

        if ($check == 1) {
            $sql = "SELECT * FROM listings WHERE city like '%$input%'";
            $res = $this->db_connect->query($sql);

            foreach ($res as $row) {
                $imgstack = $this->getImages($row['id']);
                $newListing = new listing_model($row);
                $newListing->setImages($imgstack);
                $dataSet[] = $newListing;
            }
        }

        if ($check == 0) {
            $sql = "SELECT * FROM listings WHERE zip like '%$input%'";
            $res = $this->db_connect->query($sql);
            foreach ($res as $row) {
                $imgstack = $this->getImages($row['id']);
                $newListing = new listing_model($row);
                $newListing->setImages($imgstack);
                $dataSet[] = $newListing;
            }
        }

        if ($check == 3) {

            $letters = "";
            $digits = "";
            $letters = preg_replace("/[^a-z\s]/i", "", $input);
            $digits = preg_replace("/[^0-9\s]/i", "", $input);
            $letters = str_replace(' ', '', $letters);
            $digits = str_replace(' ', '', $digits);
            if (strlen($letters) == 0 && strlen($digits) == 0) {
                return NULL;
            }

            $sql = "SELECT * FROM listings WHERE zip like '%$digits%' OR city like
'%$letters%'  ";

            $res = $this->db_connect->query($sql);
            foreach ($res as $row) {
                $imgstack = $this->getImages($row['id']);
                $newListing = new listing_model($row);
                $newListing->setImages($imgstack);
                $dataSet[] = $newListing;
            }
        }
        return $dataSet;
    }
```

```
    /**
     *
     */
    public function advanceSearch($input)
    {
        return NULL;
    }
    /**
     * Search Recent listings function
     * @return \listing_model
     */
    public function searchRecent() {
        $past = date('Y/m/d', strtotime('-1 month'));
        $sql = 'SELECT * FROM listings ORDER BY id DESC';

        $res = $this->db_connect->query($sql);
        foreach ($res as $row) {
            $imgstack = $this->getImages($row['id']);
            $newListing = new listing_model($row);
            $newListing->setImages($imgstack);
            $dataSet[] = $newListing;
        }
        return $dataSet;
    }
/**
     * Search Recent listings function
     * @return \listing_model
     */
    public function searchSold() {
        $sql = 'SELECT * FROM listings WHERE sold = 1';

        $res = $this->db_connect->query($sql);
        foreach ($res as $row) {
            $imgstack = $this->getImages($row['id']);
            $newListing = new listing_model($row);
            $newListing->setImages($imgstack);
            $dataSet[] = $newListing;
        }
        return $dataSet;
    }

    /**
     * Get Realtor's Listings from the Database
     * @return \UserData
     */
    public function getRealtorListings($realtorId) {
        $imgstack = array();
        $sql = 'SELECT * from listings WHERE userid = "$realtorId"';
        foreach (parent::$this->db_connect->query($sql) as $row) {
            $imgstack = $this->getImages($row['id']);
            $newListing = new listing_model($row);
            $newListing->setImages($imgstack);
            $dataSet[] = $newListing;
        }
        if (!empty($dataSet))
            return $dataSet;
        else
            return null;
    }

    /**
     * Add a Listing to the Database
     * @param type $input
     */
    public function addListing($input) {
        $sql = 'INSERT INTO listings(address, city, zip, price, rooms, bathrooms,
            description, userid, when_added, when_modified) VALUES (
            :address, :city, :zip, :price, :rooms, :bathrooms,
            :description, :userid, :when_added, :when_modified)';

        $stmt = $this->db_connect->prepare($sql);
        $stmt->bindParam(':address', $input->getAddress(), PDO::PARAM_STR);
        $stmt->bindParam(':city', $input->getCity(), PDO::PARAM_STR);
        $stmt->bindParam(':zip', $input->getZip(), PDO::PARAM_INT);
        $stmt->bindParam(':price', $input->getPrice(), PDO::PARAM_INT);
        $stmt->bindParam(':rooms', $input->getRooms(), PDO::PARAM_INT);
        $stmt->bindParam(':bathrooms', $input->getBathrooms(), PDO::PARAM_INT);
        $stmt->bindParam(':description', $input->getDescription(), PDO::PARAM_STR);
        $stmt->bindParam(':userid', $input->getUserId(), PDO::PARAM_STR);
        $stmt->bindParam(':when_added', date('Y/m/d'), PDO::PARAM_STR);
        $stmt->bindParam(':when_modified', date('Y/m/d'), PDO::PARAM_STR);

        $stmt->execute();
    }
```

```
public function editListing($input) {
    $sql = "UPDATE listings SET address = :address,
        city = :city,
        zip = :zip,
        price = :price,
        rooms = :rooms,
        bathrooms = :bathrooms,
        description = :description,
        when_modified = :when_modified
        WHERE id = :id";

    $stmt = $this->db_connect->prepare($sql);

    $stmt->bindParam(':address', $input[1], PDO::PARAM_STR);
    $stmt->bindParam(':city', $input[2], PDO::PARAM_STR);
    $stmt->bindParam(':zip', $input[3], PDO::PARAM_INT);
    $stmt->bindParam(':price', $input[4], PDO::PARAM_INT);
    $stmt->bindParam(':rooms', $input[5], PDO::PARAM_INT);
    $stmt->bindParam(':bathrooms', $input[6], PDO::PARAM_INT);
    $stmt->bindParam(':description', $input[7], PDO::PARAM_STR);
    $stmt->bindParam(':when_modified', date("Y/m/d"), PDO::PARAM_STR);
    $stmt->bindParam(':id', $input[0], PDO::PARAM_INT);

    $stmt->execute();
}

/**
 * Delete a Listing from the database
 * @param type $id
 */
public function deleteListing($id) {
    $sql = "DELETE FROM listings WHERE id =  :id";
    $stmt = $this->db_connect->prepare($sql);
    //$stmt->bindParam(':id', $id, PDO::PARAM_STR, 12);
    $stmt->bindParam(':id', $id, PDO::PARAM_INT);
    $stmt->execute();
}

/**
 * Set a listings image
 * @param type $id
 * @param string $imgName
 */
public function setImage($id, $imgName) {
    $defaultPath = '/~fl4g03/views/assets/images/';
    $imgName = $defaultPath . $imgName;
    $sql = "INSERT INTO images(houseid, path) VALUES (
        :houseid, :path)";

    $stmt = $this->db_connect->prepare($sql);
    $stmt->bindParam(':houseid', $id, PDO::PARAM_STR);
    $stmt->bindParam(':path', $imgName, PDO::PARAM_STR);

    $stmt->execute();
}
```

```
    /**
     * Get images associated with a listing
     * @param type $listingId
     * @return imgStack
     */
    public function getImages($listingId) {
        $sql = "SELECT * from images WHERE houseid = '$listingId'";
        foreach (parent::$this->db_connect->query($sql) as $row) {
            $imgstack[] = $row['path'];
        }
        if (!empty($imgstack))
            return $imgstack;
        else
            return null;
    }

    /**
     * Remove image based on path
     * @param type $imagePath
     */
    public function removeImage($imagePath) {
        $slash_count = 0;
        $filename = "";
        for ($i = 0; $i < strlen($imagePath); $i++) {
            if ($imagePath[$i] == '/')
                $slash_count++;
            if ($slash_count == 5) {
                $slash_count++;
                continue;
            }
            if ($slash_count == 6) {
                $filename .= $imagePath[$i];
            }
        }

        // delete from dir
        array_map('unlink', glob("/home/f14g03/public_html/views/assets/images/" .
$filename));

        // delete from db
        $affected_rows = $this->db_connect->exec("DELETE FROM images WHERE path =
'$imagePath'");
    }

    /**
     * Get a listing function
     * @param type $id
     * @return \listing_model
     */
    public function getListing($id) {
        $sql = "SELECT * from listings WHERE id = '$id'";
        foreach (parent::$this->db_connect->query($sql) as $row) {
            $listing = new listing_model($row);
            return $listing;
        }
    }

    /**
     * Get a users most recent listing function
     * @param type $userid
     * @return type
     */
    public function getNewListing($userid) {
        $sql = "SELECT * from listings WHERE userid = '$userid' ORDER BY id DESC
LIMIT 1";
        foreach (parent::$this->db_connect->query($sql) as $row) {
            $listing = new listing_model($row);
            return $listing->getId();
        }
    }

}
```

4.2 Alex's Response



Subject: Re: Code Review for Listings Controller
From: alyash@sfsuswe.com
Date: Fri, December 12, 2014 9:07 pm
To: "Rushab Indi" <rindi@sfsuswe.com>
Cc: alyash@sfsuswe.com (more)
Priority: Normal
Options: View Full Header | View Printable Version | Download this as a file

Hi Rushab

I attached file with code review.
Thank you.

Alex

> Hi Alex,
>
> Could you go over the attached code for the listings controller? Feel
> free to optimize the code and give your comments to make the code clearer.
>
> Thanks,
> Rushab
>

Attachments:

code_review_comments 0.7 k [application/octet-stream] Download

contents of code_review_comments.php:

Reviewer: Alex Lyashevych

Code looks well overall. Straight to the point comments & kept to minimum (don't clutter).

Suggestions:

1. between lines 31-44: unnecessary {} can be removed, and comments are not indented.

2. on lines 45,57,69 would also be nice to have little comment regarding what happens inside if statement.

3. on lines 99-102: do we really need that advaceSearch() function? Seems unnecessary.

4. for loops in lines 245-252, 261-271: we can remove unnecessary {} for readability.

5. on line 24: would be nice to have that type of expected input (such as 'string').

5. Risk Analysis

5.1 Skills risks

Risk: Our team doesn't have much experience with website development, design, or deployment. We don't have a specialist with any of the toolsets we will be working with.

Solution: We have worked through a number of tutorials on the toolsets we intend to us and we are working within the team to make sure that we are able to learn and gain the skills required to address our lack of experience.

5.2 Schedule Risks

Risk: Our group has 5 members with very different schedules for this semester. We understand that there is a limited amount of time we need to develop and deploy the site on time and to specification even though the semester is already half over.

Solution: We plan to stick to the suggested schedule of meeting twice a week and follow the required steps in completing the milestones while trying to not over-commit.

5.3 Technical Risks

Risk: We have been considering mapping our listings on our site but we haven't figured out what would be the best tool to accomplish this task. We also are not sure the best way to accomplish formatting of the website across platforms (ie making sure that the website appears clearly on a desktop and a mobile device).

Solution: We have assigned research to our team members in order to research these areas to determine the best plan. By assigning research tasks we will be able to hope to be able to better analyze the cost associated with learning and implementing these features.

5.4 Teamwork Risks

Risk: We have a variety of personality types in the group, which is to be expected in such a diverse group. We also have a mix of undergraduate students and graduate students which have the potential to cause disagreements in our perceived target objective.

Solution: We have agreed to work together and align our objectives for the purpose of deploying a successful product. We try make decisions as a group and ensure that everyone has a voice in the decisions.

5.5 Legal Risks

Risk: Whenever working with software and using other libraries there is always a risk that the library we choose to use has a security risk. If we are collecting user information and someone were to sign up for an account — thinking that it were an actual real estate website — and someone was able to access the user's information, we could be held legally responsible.

Solution: To avoid this issue and other issues relating to this, we will not collect credit card information from users in the beginning phase of our project. We will also be sure to prominently display "SFSU/FAU/Fulda Software Engineering Project, Fall 2014. For Demonstration Only".