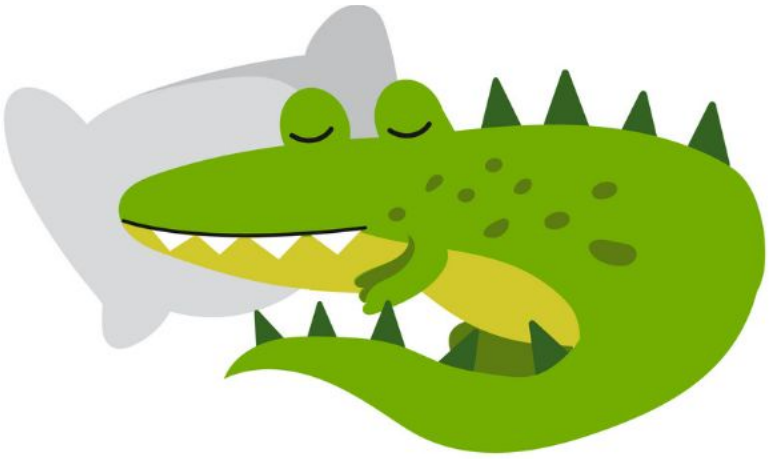
SW Engineering CSC 648/848 Spring 2019



Team 13
Amari Bolmer - Team Lead - amarirules@gmail.com
Brian Ho, Simon Tan, Adeel Bhatti,
Kim Wang, Kurtis Hoang, Sushil Kumar Plassar

https://github.com/CSC-648-SFSU/csc648-sp19-team13/tree/master

Milestone 4

May 8, 2019

History Table:
Submission: May 8th, 2019

Product summary

LiveGator

LiveGator is a rental website targeted towards helping San Francisco State University students find somewhere to live. SFSU students will easily be able to find a place specifically to their needs, whether its is based on price, size or distance from SFSU.

Committed Functions:

- Users shall be able to sign up, log into, and log out of the website.
- Users shall be able to use the search bar to find all available listings.
- Users shall be able to filter listings by price and size.
- Users shall be able to sort listings by price and size.
- Sellers shall be able to post and delete listings.

LiveGator is uniquely focused on students having the ability to search for rooms, housing, or apartments with the option to sort by distance to campus of San Francisco State University. The function to sort by distance shall give students peace of mind to make the decision of where to buy or rent easy as a click of a button.

Website url: http://18.223.29.166/

Usability test plan

Test Objective:

We will be testing the post a listing function. Listings are integral to the success of our product. If landlords find it difficult to post their property, the purpose of our product is defeated. We want the landlords to be able to easily create a listing because more properties listed on the website will bring more potential renters to our site. We also think that the landlords will have less experience with software because they tend to be older. With this in consideration, we want to make the posting process as easy as possible.

Test Description:

The user will attempt to post a listing for a property they would like to lease. The user will attempt to use a firefox web browser to begin the process. The main purpose of this test is to measure the satisfaction of our landlords when they make a post. Users will start as a registered user on the homepage of our site. The job of the user is to post a property they want to rent.

We will collect data with a questionnaire after they have completed the task. The user will be tested on Chrome browser on a laptop. One of our team members will be observing the user while they post the listing. Feedback will be collected in a questionnaire and we will be evaluating the user satisfaction after they have completed the task. The user will be a volunteer. The final report will contain the level that the user agrees with the below statements.

Usability Task Description:

- Locate posting page
- Access posting page
- Recognize required fields
- Create listing and post it to the server

Questionnaire:

	Strong Disagree	Disagree	Neutral	Agree	Strong Agress
The posting page was be easily found.					
The required inputs were easily indicated.					
The inputs were relevant to my objective.					
Creating a listing was straightforward.					

QA test plan

Feature to be Tested:

Post a listing feature (Only accessed by landlord)

Test Objective:

We are testing the "post a listing" feature for the landlords using our app. We are making sure this feature allows a user to create a database instance of a listing properly.

We are going to be testing three different types of inputs. One will be a normal address input, and the other two will be invalid input and an empty input. It would be optimal that when there is an invalid or empty input the user listing is notified immediately so that they can modify it and adhere to a greater overall user experience where addresses are accurate.

Hardware/SW:

We are using a Macbook Pro with two different browsers (Chrome and Firefox) to test the feature. It is currently being tested hosted from local machine, with database being pulled from AWS server. The feature will be tested from the "add a new listing" page which can be found on our local machine here:

The URL of the page is: http://127.0.0.1:5000/renter_dashboard/add_a_new_listing

Browser used for testing: Firefox 66.0.5 (Official Build) (64-bit)

ID	Test Description	Test Input	Expected Output	Actual Results	Pass/Fail
T_01	User enters a valid input while submitting a post request on listing page	1600 Holloway Ave. San Francisco, CA	Listing gets posted successfully	Listing gets posted successfully	Pass
T_02	Enter an "Invalid/alphanumeric Character/Input" in the input fields on listing page	"Jkjd9$!" in Street Number and ZipCode	A prompt with "Invalid Input" should be displayed to user	Listing gets posted successfully	Fail
T_03	To test if the User is forced to populate the minimum input fields before submitting the post requests	All Fields Left Empty	Fields Required* Prompt should be displayed to the user	Listing gets posted successfully	Fail

Browser used for testing: Google Chrome 73.0.3683.103 (Official Build) (64-bit)

ID	Test Description	Test Input	Expected Output	Actual Results	Pass/Fail
T_01	User enters a valid input while submitting a post request on listing page	1600 Holloway Ave. San Francisco, CA	Listing gets posted successfully	Listing gets posted successfully	Pass
T_02	Enter an "Invalid/alphanumeric Character/Input" in the input fields on listing page	"Jkjd9$!" in Street Number and ZipCode	A prompt with "Invalid Input" should be displayed to user	Listing gets posted successfully	Fail
T_03	To test if the User is forced to populate the minimum input fields before submitting the post requests	All Fields Left Empty	Fields Required* Prompt should be displayed to the user	Listing gets posted successfully	Fail

Code Review:

a) The coding style we have chosen is PEP 8

b) Example of the code under review (from post a listing feature):

Original code submitted:

```python
@listing_endpoints.route('/renter_dashboard/add_a_new_listing', methods=['GET', 'POST'])
@login_required
def add_a_new_listing():
    if request.method == 'GET':
        return render_template("renter_add_a_new_listing.html")
    try:
        user_id = current_user.user_id
        house_name = request.form.get("house_name", "N/A")
        type = request.form.get("type", "N/A")
        description = request.form.get("description", "N/A")
        price = request.form.get("price", "N/A")
        size = request.form.get("size", "N/A")
        distance = request.form.get("distance", "N/A")
        number = request.form.get("number", "N/A")
        street = request.form.get("street", "N/A")
        city = request.form.get("city", "N/A")
        state = request.form.get("state", "N/A")
        zipcode = request.form.get("zipcode", "N/A")
        user_photos = request.files.getlist("photos")
        filenames = []
        for photo in user_photos:
            filenames.append(uploaded_photos.save(photo))
        image_url = ""
        for filename in filenames:
            image_url = image_url + filename + " "
        bedroom_count = request.form.get("bedroom_count", "N/A")
        bathroom_count = request.form.get("bathroom_count", "N/A")
        parking_count = request.form.get("parking_count", "N/A")
        listings.add_a_new_listing(user_id, house_name, type, description, price, size, distance, number, street, city, state, zipcode, image_url, bedroom_count, bathroom_count, parkin
        return redirect('/renter_dashboard/view_listings')
    except:
        return abort(400)
```

```python
def add_a_new_listing(user_id, house_name, type, description, price, size, distance, number, street, city, state, zipcode, image_url, bedroom_count, bathroom_count, parking_count):
    DButils.create_new_listing(user_id, house_name, type, description, price, size, distance, number, street, city, state, zipcode, image_url, bedroom_count, bathroom_count, parking_coun
```

```python
def create_new_listing(user_id, house_name, type, description, price, size, distance, number, street, city, state, zipcode, image_url, bedroom_count, bathroom_count, parking_count):
    conn.connect()
    cur = conn.cursor()
    sql_str = "INSERT INTO LISTINGS (landlord_id, house_name, type, description, price, size, distance, number, street, city, state, zipcode, image_url, bedroom_count, bathroom_count, "
    cur.execute(sql_str, (user_id, house_name, type, description, price, size, distance, number, street, city, state, zipcode, image_url, bedroom_count, bathroom_count, parking_count))
    conn.commit()
    cur.close()
    conn.close()
```

Email communication:

Hi Kim, could you review my 'add_a_new_listing' method? Inbox ×

Adeel S <adeel182@gmail.com>
to me

Let me know what to change or if I need to add anything

Yangyifan Wang
to Adeel

Hi Adeel,

Your code looks good to me. Just made some minor changes:

- Lines longer than 120 characters are separated into multiple lines
- A comment header is added to the file
- I also added a line comment explaining what the method is doing

Hope this helps!

Regards,
Kim

•••

Updated code after code review:

```
@listing_endpoints.route('/renter_dashboard/add_a_new_listing', methods=['GET', 'POST'])
@login_required
def add_a_new_listing():
    '''
        This function retrieves input parameters from request and submit to DB
    '''
    if request.method == 'GET':
        return render_template("renter_add_a_new_listing.html")
    try:
        user_id = current_user.user_id
        house_name = request.form.get("house_name", "N/A")
        type = request.form.get("type", "N/A")
        description = request.form.get("description", "N/A")
        price = request.form.get("price", "N/A")
        size = request.form.get("size", "N/A")
        distance = request.form.get("distance", "N/A")
        number = request.form.get("number", "N/A")
        street = request.form.get("street", "N/A")
        city = request.form.get("city", "N/A")
        state = request.form.get("state", "N/A")
        zipcode = request.form.get("zipcode", "N/A")
        user_photos = request.files.getlist("photos")
        filenames = []
        for photo in user_photos:
            filenames.append(uploaded_photos.save(photo))
        image_url = ""
        for filename in filenames:
            image_url = image_url + filename + " "
        bedroom_count = request.form.get("bedroom_count", "N/A")
        bathroom_count = request.form.get("bathroom_count", "N/A")
        parking_count = request.form.get("parking_count", "N/A")
        listings.add_a_new_listing(user_id, house_name, type, description, price, size, distance, number, street,
                                   city, state, zipcode, image_url, bedroom_count, bathroom_count, parking_count)
        return redirect('/renter_dashboard/view_listings')
    except:
        return abort(400)
```

```
'''
Created on 08 Mar 2019
Author: Kim Wang, Adeel Bhatti
Description: This file implements listing related APIs
'''
```

```
def add_a_new_listing(user_id, house_name, type, description, price, size, distance, number, street, city, state,
                      zipcode, image_url, bedroom_count, bathroom_count, parking_count):
    '''
        This method takes input parameters and submit to DB util file
    '''
    DButils.create_new_listing(user_id, house_name, type, description, price, size, distance, number, street, city,
                               state, zipcode, image_url, bedroom_count, bathroom_count, parking_count)
```

```
def create_new_listing(user_id, house_name, type, description, price, size, distance, number, street, city, state,
                       zipcode, image_url, bedroom_count, bathroom_count, parking_count):
    '''
        This method execute mysql query
    '''
    conn.connect()
    cur = conn.cursor()
    sql_str = "INSERT INTO LISTINGS (landlord_id, house_name, type, description, price, size, distance, number, " \
              "street, city, state, zipcode, image_url, bedroom_count, bathroom_count, parking_count, create_date) " \
              "VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, now())"
    cur.execute(sql_str, (user_id, house_name, type, description, price, size, distance, number, street, city,
                          state, zipcode, image_url, bedroom_count, bathroom_count, parking_count))
    conn.commit()
    cur.close()
    conn.close()
```

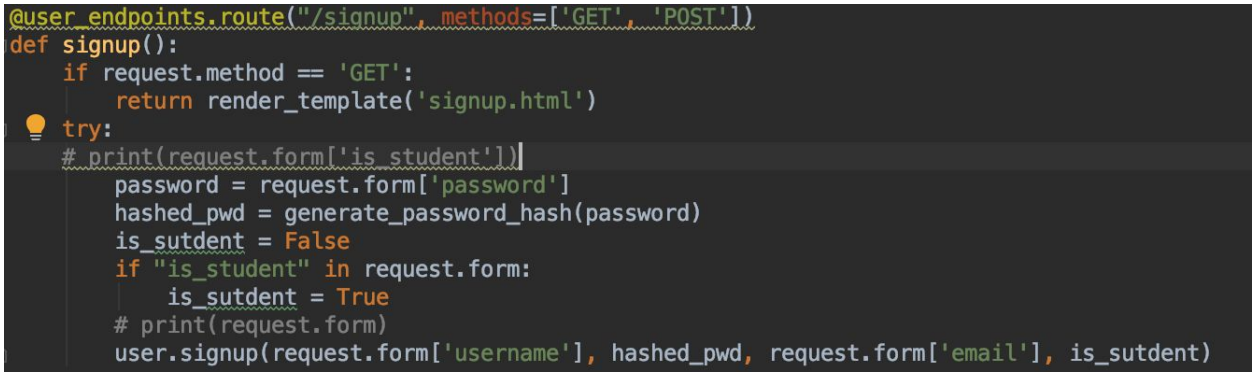
Code review communication email attached:
message:%3CCAC1ipTG5mssopz4NprCNAGmx72kt6SeVyRdrzOZey8OrgVykkg@mail.gmail.com%3E

Self-Check on best practices for security

Major assets we are protecting:

1. User information such as username, password, email
2. Listing information such as description, location, and images

To encrypt the password, we are hashing the password, when the user signs up. This protects the user's personal information, if there was a breach in our database.



```
@user_endpoints.route("/signup", methods=['GET', 'POST'])
def signup():
    if request.method == 'GET':
        return render_template('signup.html')
    try:
    # print(request.form['is_student'])
        password = request.form['password']
        hashed_pwd = generate_password_hash(password)
        is_sutdent = False
        if "is_student" in request.form:
            is_sutdent = True
        # print(request.form)
        user.signup(request.form['username'], hashed_pwd, request.form['email'], is_sutdent)
```

To confirm input data validation, we are limiting the user input to a maximum of 40 characters. We are also encrypting the user's information and validating the login. We do validation for the search bar based on price, size, and distance.

Self-Check: Adherence to original Non-functional specs

1. Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in M0 (some may be provided in the class, some may be chosen by the student team but all tools and servers have to be approved by class CTO).
    - On Track

2. Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers
    - On Track

3. Selected application functions must render well on mobile devices
    - On Track

4. Data shall be stored in the team's chosen database technology on the team's deployment server.
    - Done

5. No more than 50 concurrent users shall be accessing the application at any time
    - Done

6. Privacy of users shall be protected and all privacy policies will be appropriately communicated to the users.
    - On Track

7. The language used shall be English.
    - Done

8. Application shall be very easy to use and intuitive.
    - On Track

9. Google analytics shall be added
    - On Track

10. No email clients shall be allowed
    - Done

11. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated.

- Done

12. Site security: basic best practices shall be applied (as covered in the class)

- Done

13. Before posted live, all content (e.g. apartment listings and images) must be approved by site administrator

- On Track

14. Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development

- Done / Continuous development

15. The website shall prominently display the following exact text on all pages "SFSU Software Engineering Project CSC 648-848, Spring 2019.
For Demonstration Only" at the top of the WWW page. (Important so as to not confuse this with a real application).

- On Track