SOFTWARE TESTING REPORT

Virtuwoof

—

Overview

This project endeavors to create a collective online space for Pet-owners and all Animal lovers in general. It will have a range of features from helping lost pets reach their owners

using location based APIs to providing for Street Animals. It will be a great addition to the lives of pet owners as the App will keep track of vet appointments and other ordeals for them. The App will have informative features such as Blog Posts, and NGO operatives nearby.

TESTING

Unit Testing

MODELS

In django models we should test the labels for all the fields, because even though we haven't explicitly specified most of them, we have a design that says what these values should be. If we don't test the values, then we don't know that the field labels have their intended values.

```
# Get a blog object to test
blog = BlogPosts.objects.get(id=1)

# Get the metadata for the required field and use it to query the required field data
field_label = blog._meta.get_field('blogContent').verbose_name

# Compare the value to the expected result
self.assertEqual(field_label, 'blogContent')
```

This unit test case snippet checks that the values of the field labels (verbose_name) and that the content and size of the Blogs' text is as expected.

FORMS

We performed form testing to check all additional validation that is expected to be performed on the fields and any messages that code will generate for errors.

The below unit test case snippet is for search form . We ensure that the user entered all the required fields. Then a check is performed whether the entered data is valid or not , if it's invalid appropriate errors are thrown, else the search will be performed. This way there will not be any unnecessary api calls and the service time on our website will be faster as only genuine requests will be completed .

```python
class SearchForm(forms.Form):

    FILTER_CHOICES = (

        ('pet.veterinary', 'Veterinary'),
        ('pet.shop', 'Pet Shops'),
        ('pet.service', 'Pet Services'),
        ('pet.dog_park', 'Pet Parks'),

    )
    address = forms.CharField(widget=forms.TextInput(attrs={"placeholder":"Enter your address"}))
    city = forms.CharField(widget=forms.TextInput(attrs={"placeholder":"city"}))
    state= forms.CharField(widget=forms.TextInput(attrs={"placeholder":"state"}))
    filter_by = forms.ChoiceField(choices = FILTER_CHOICES)

    def __init__(self,*args,**kwargs):
        super(SearchForm,self).__init__(*args,**kwargs)
        for field in self.fields:
            self.fields[field].widget.attrs['class'] = 'inputBox'
            self.fields[field].label = ""
```

VIEWS

View unit testing allows us to confirm that our template is getting all the data it needs. In other words we can check that we're using the intended template and what data the template is getting, which goes a long way to verifying that any rendering issues are solely due to the template.

```python
def test_lists_all_blogs(self):

        response = self.client.get(reverse('blogs')+'?page=2')
        self.assertEqual(response.status_code, 200)
        self.assertTrue('is_paginated' in response.context)
        self.assertTrue(response.context['is_paginated'] == True)
        self.assertEqual(len(response.context['blog_list']), 10)
```

Below is the snippet attached for those views in our website that are only restricted to logged in users like New post view, direct messaging view, donation view etc. So by testing we have made sure that only logged in users can access that page and redirect all the anonymous users to the login page.

```
def test_redirect_if_not_logged_in(self):
        response = self.client.get(reverse('newPost'))
        self.assertRedirects(response, '/accounts/login/')

    def test_logged_in_uses_correct_template(self):
        login = self.client.login(username='testuser1', password='1X<ISRUkw+tuK')
        response = self.client.get(reverse('newPost'))

        # Check our user is logged in
        self.assertEqual(str(response.context['user']), 'testuser1')
        # Check that we got a response "success"
        self.assertEqual(response.status_code, 200)

        # Check we used correct template
        self.assertTemplateUsed(response, 'newPost/')
```

PAYMENT

We used stripe to accept the payment from the user .

To confirm that our integration works correctly, we simulated transactions without moving any money, using special values in test mode provided by stripe.

We simulated following scenarios:

- Successful payments by card, brand or country
- Card errors due to declines, fraud, or invalid data
- Disputes and refunds

Following is the snippet to test in case of card errors like expired or banned card or invalid data

```
def test_change_card_error(self, retry_mock, send_mock, update_mock):
    update_mock.side_effect = stripe.CardError("Bad card", "Param", "CODE")
    self.client.login(username=self.user.username, password=self.password)
    response = self.client.post(
        reverse("payments_ajax_change_card"),
        {"stripe_token": "XXXXX"},
        HTTP_X_REQUESTED_WITH="XMLHttpRequest"
    )
    self.assertEqual(update_mock.call_count, 1)
    self.assertEqual(send_mock.call_count, 0)
    self.assertEqual(retry_mock.call_count, 0)
    self.assertEqual(response.status_code, 200)
```

We used manual and unit testing to check all the possible test cases . We wrote unit testing code for card errors , fraud etc, and manual checking for successful payment by cards.

Manual Testing

Two test accounts were created to test the flow of entire web application

Personal User account

- Three dummy test posts were created for all the categories
- Reacted on other posts
- Interacted with other people who had information about his lost pet via inbuilt direct messaging service on website
- Searched for pet services near the locality
- Fetched information about various NGOs helpline numbers to rescue injured animals
- Interacted with the bot to get first aid assistance for the injured pet
- Donated to the people and organizations

Admin Account

- Verified the registration details of all the Non profit organizations
- Only after account verification by admin NGO could raise donation request
- Checked chat logs with bot in case user wants to interact with service executive regarding pet injury
- Verified the payment links of NGO
- Tracked any suspicious recurrent transactions from the same account

Taking the account through the complete process from signing up to testing every feature ensured that none of the features were breaking in production and that they were all working properly.

We also ran the tests in various devices and resolutions to make sure the UI did not vary incorrectly. This process was repeated with varying the inputs and corner case inputs to check for integrity of the application. We employed various techniques to test input validation, including bug checks.

Integration Testing

We ran all unit test modules. We found that all the modules are loosely coupled and don't require separate integration testing.

To perform manual integration testing,we have individually tested all the modules and made sure that they are working properly in combination with each other.

We prepared a top-down flowchart of all the modules to check the flow of our web application.

System Testing

We performed manual System testing on a complete integrated system to evaluate the compliance of the system with all the requirements.

We tested the design and behavior of the system and also the expectations of the customer.

1) **USER REGISTRATION**

TEST CASES	**RESULT**
User registration with new email id	Successful user registration
User registration with already registered email id	User already existing error
User registration with wrong email id	Enter valid email id

2) LOGIN

TEST CASES	RESULT
Login with correct email and password	Logged in Successfully
Login with wrong email and password	Wrong credentials error
Login with correct email and wrong password	Wrong credentials error

3) CREATE POST

TEST CASES	RESULT
Access create post without login	Redirects to login page
Access create post after login	Create post page opens
Create post with all valid information as per the required constraints	Post created successfully
Invalid or missed data while creating post	Fill all required fields or enter valid data error

4) FEED POST

TEST CASES	RESULT
Access feed page	All posts are displayed with pagination
Action button on posts	All buttons are working fine
Post notification	Any action performed on Post, notifications are sent to the owner

On clicking **donate** action button	Redirected to donation page of the post owner
On clicking **adoption** or **lost** action button	Post owner is notified and then contact to the corresponding user

5) DIRECT MESSAGING

TEST CASES	RESULT
Direct chat access without login	Redirects to login page
Direct chat access after login	Opens chat window
Logged in user sends message	Message sent successfully and notified the receiver
network disconnected while sending message	Message not sent

6) SEARCH BASED ON LOCATION

TEST CASES	RESULT
Enters all the required field in search box	Displays search result based on location
Leaves house address field in search box	Displays search result based on city and state
Enters wrong information	Enter valid data error
Search for location with no vet shops	No data available error

After running the above test cases we fixed all detected irregularities between the unit tests that are integrated together. We made sure that none of the features are breaking and the system as a whole is working smoothly in all circumstances .