UNIVERSITY OF PERADENIYA

COMPUTER ENGINEERING

Software Engineering Project

CO 328

AN ONLINE PERSONALIZED ADVERTISEMENT PLACING PLATFORM FOR LOCAL SHOPS

STUDENT ID :
E/14/237 Pankayaraj. P
E/14/302 Hiruna Samarakoon
E/14/390 Mabeesha Wijekoon
E/14/222 Rahal Medawatte

Contents

1 ABSTRACT

The concept behind this project is to design and implement a web page to connect the local customers with the local shop owners by building a platform for advertisements. This project is build on the basis of providing an interactive interface for both users and shop owners with the ability to convey the information about them as much as possible while focusing also on the development of a capable algorithm to capture the preference of the customer dynamically.

2 INTRODUCTION

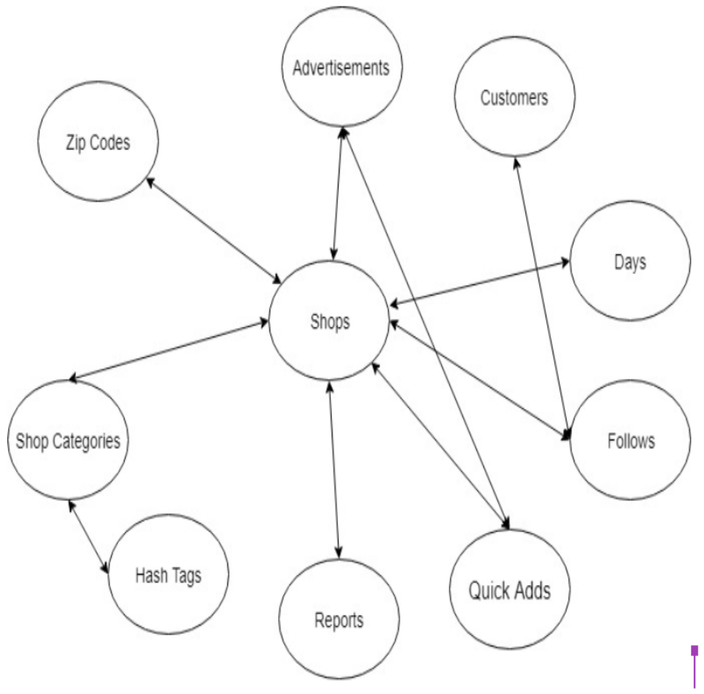
When it comes to suggesting local shops to local customers there is a general lesser degree of personalizing when it comes to the existing online platforms due to their lack of flexibility in allowing the shops to maintain an informative profile and the absense of proper algorithms to capture the customer's preferences. So as for the first part of the project is concerned with the creation of an interactive interface in order to make the end product accessible and easily understandable for the customer while letting the user have a flexible customization. Even with a better design and accessibility with the lack of personalizing the app is only as good as an advertising page in a newspaper. Thus the proper working of the algorithm with it's ability to maintain an computational efficiency for all of it's online updates becomes a vital bottleneck for the product. Thus the second part of the project is focused on the development of the algorithm and the challenges faced during the process.

3 MOTIVATION

Even at a time where everything is one click away, information of products, deals and shops are scattered everywhere. Some are in social media pages, some are on their websites and some information are physically inside the shops. Due to these reasons, most of the valuable information go unnoticed. So why not make every information available at one place?

4 INTERFACE DESIGN

4.1 Models(schema) and their relationships to each other



4.2 Design

The entire system was designed using the following diagrams.

The use-case diagram: Mainly there are three actors in the system

1. Users (Customers)
2. Shop owners
3. Site Maintainers

We identify each of these actors/roles have different set of rules in the system. The following use-case diagram shows the interactions are done with the system.

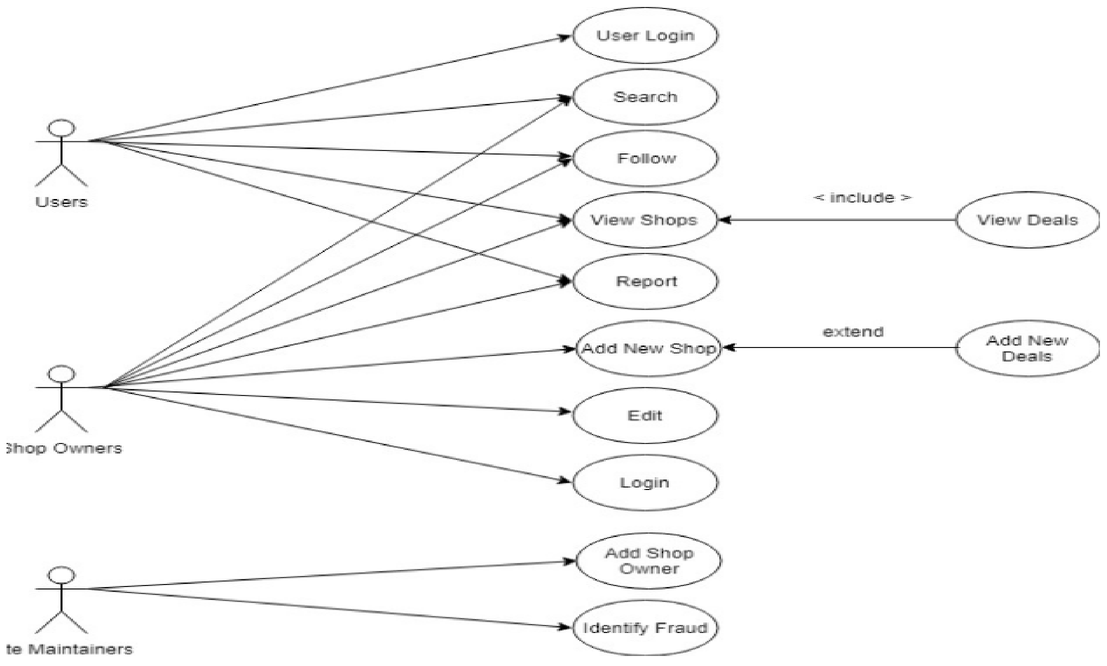
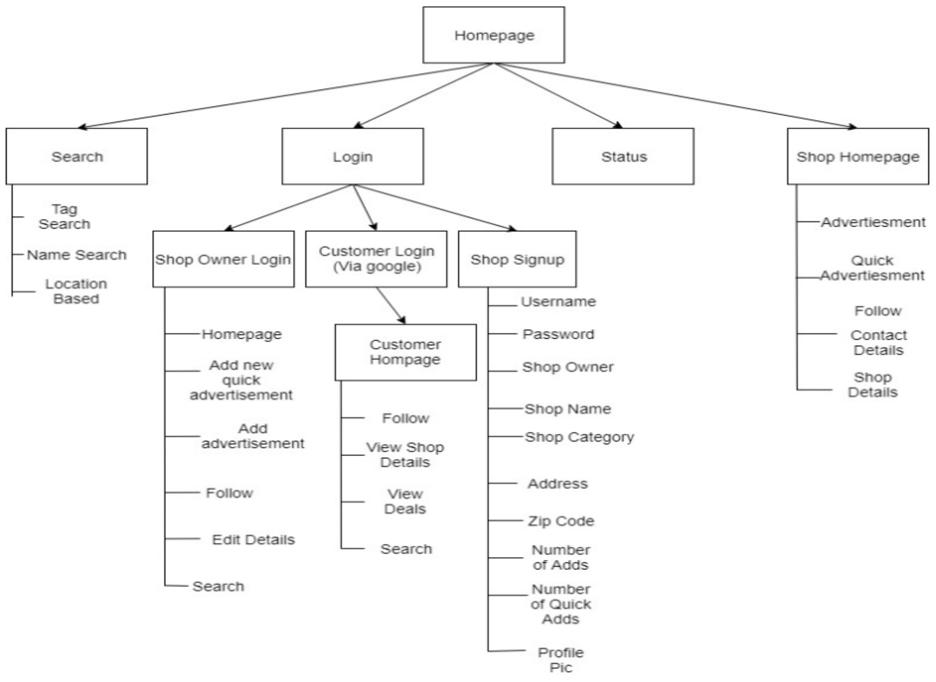
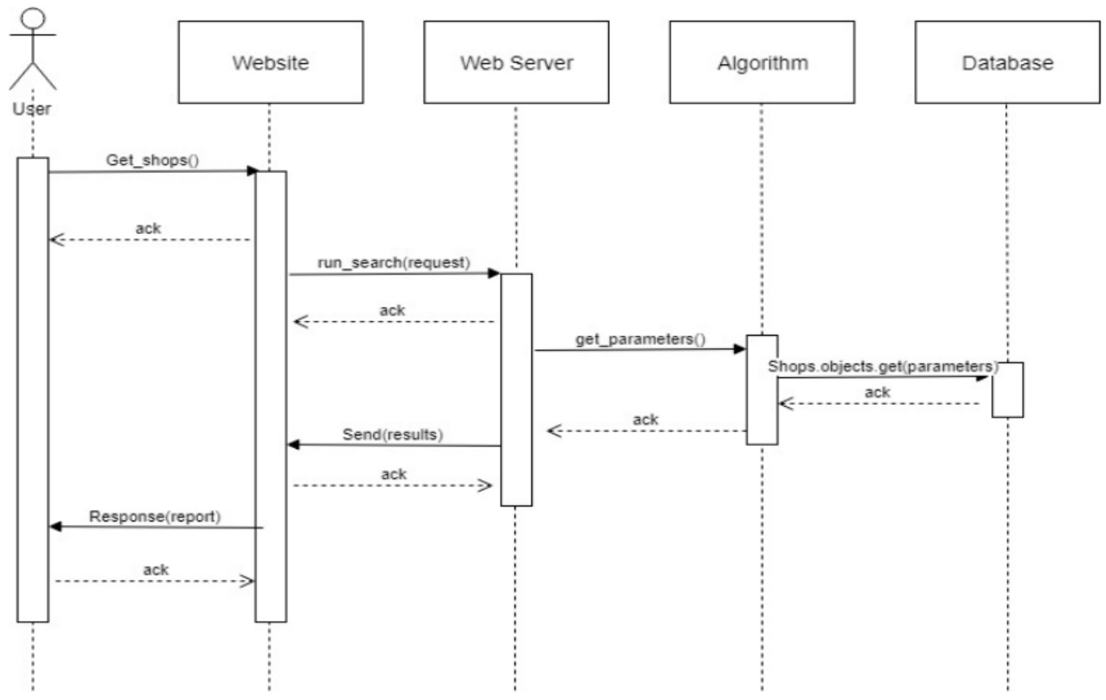


Figure: Use-case diagram of the system

The site map gives the overall view of the website



The sequence diagram



Some of the design patterns used in the development

1. Singleton Pattern.
2. Factory Pattern.
3. Abstract Factory Pattern etc.

The concepts in the above said design patterns were used in developing our system. For an example, the model ZipCode' was implemented as a single instance class, thus the idea singleton pattern was used.

Since the versatility of the factory pattern concept is higher, it was the major design pattern behind most of the structure.

Even though the design patterns were not followed in strict order, it made the development easier and debugging efficient.

4.3 Development

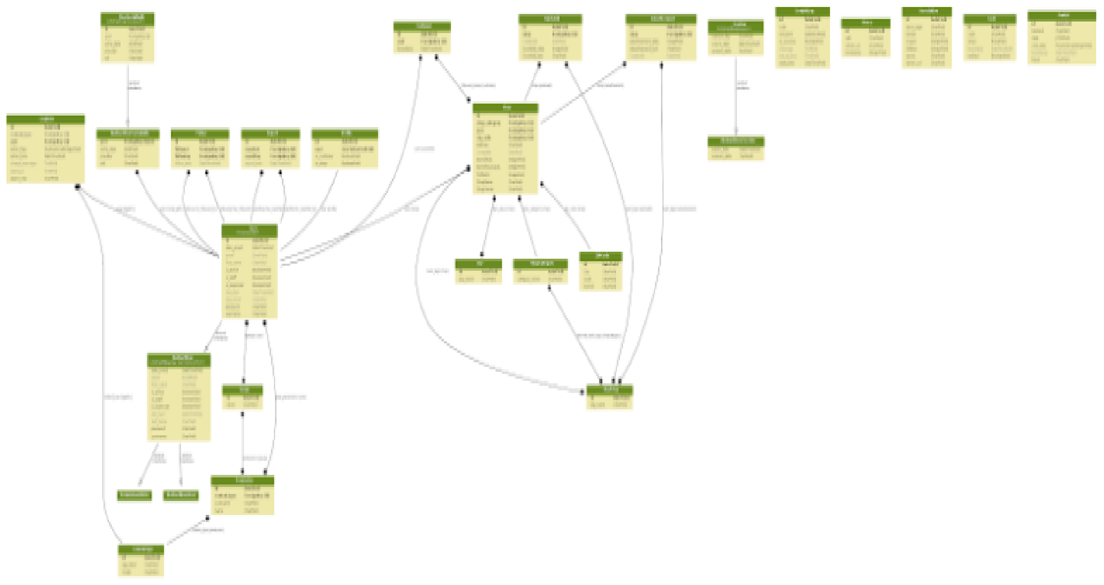


Figure : Class Diagram of the Project

A better diagram at: `https://drive.google.com/file/d/1vT6Cpih0oYaRTugktVQt78fjJQs_6Uy-/view?usp=sharing`

4.4 Django Testing

Since Web Server is written in django it is mostly django testing There are three important aspects in django structure

1. Views (functions)
2. Models (database)
3. Forms (html forms)

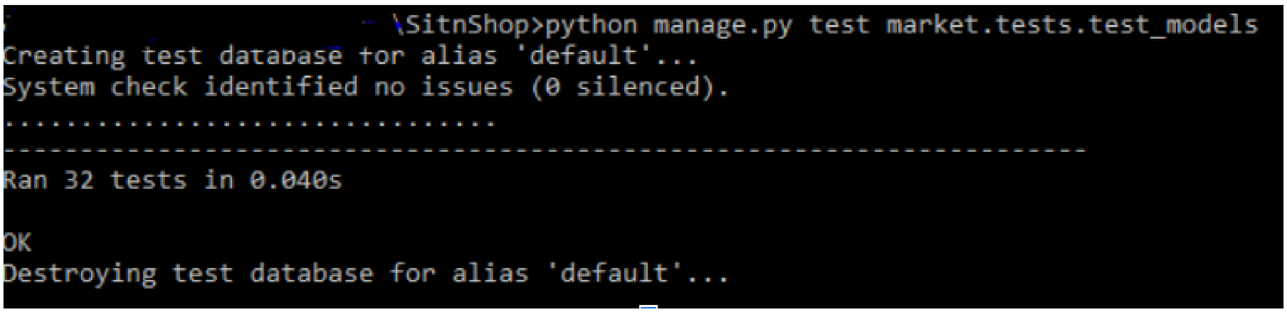
Testing was carried under above three categories separately, interactively covering the three subparts of testing

1. Unit test Regression test
2. Integration test

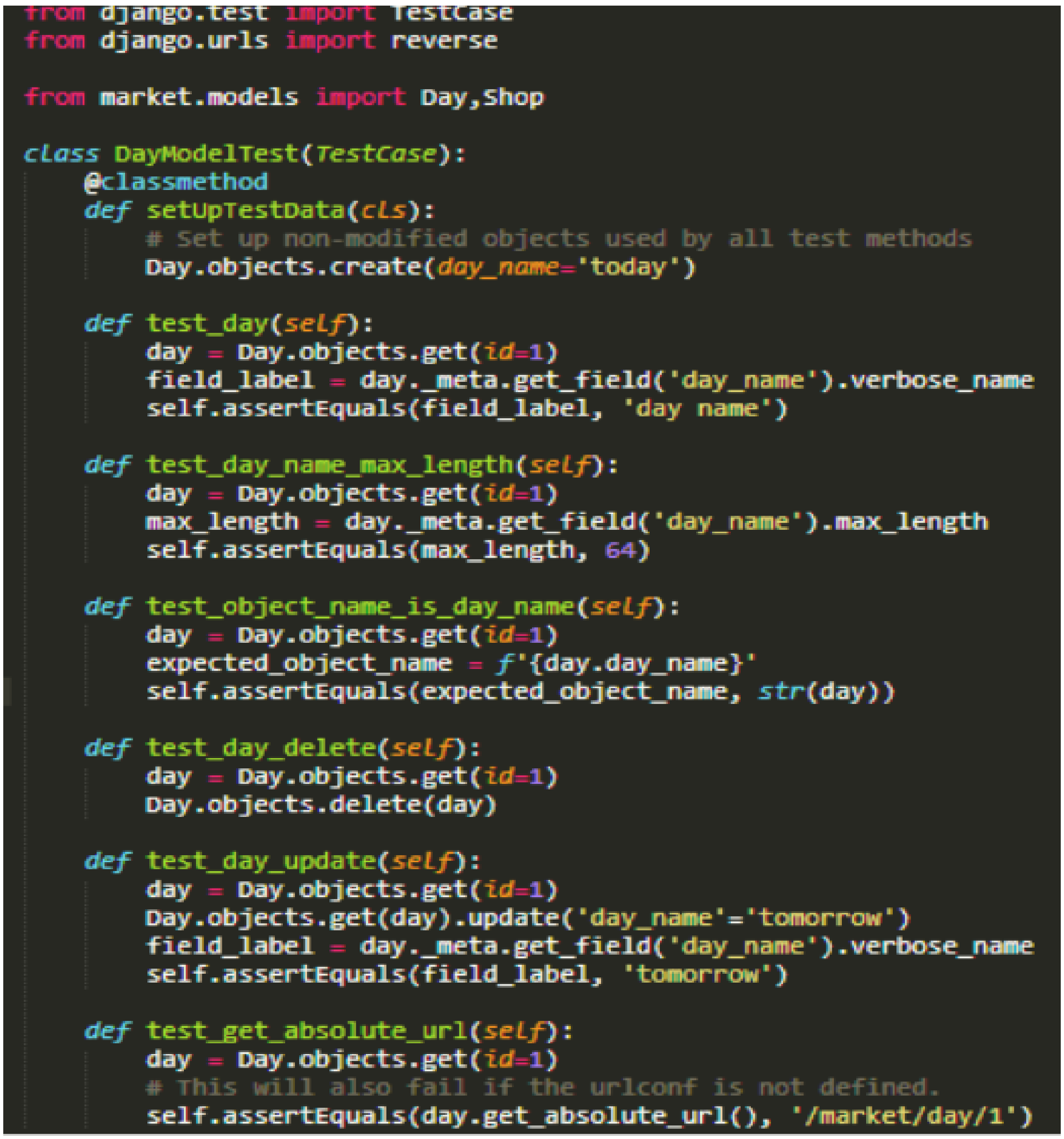
For all tests the expectation is that the result is either expected, unexpected, or an error. Here we have tested the code of our design specifications, not the behaviour of the underlying framework and other third party libraries. For an example it is not needed to verify field type as Django validates the field type correctly.

4.4.1 Models testing

In testing environment all the models were created with dummy data and tested for create object, fields' length, one to one, many to many.. relationships, object name, absolute URL mapping, update object and delete object

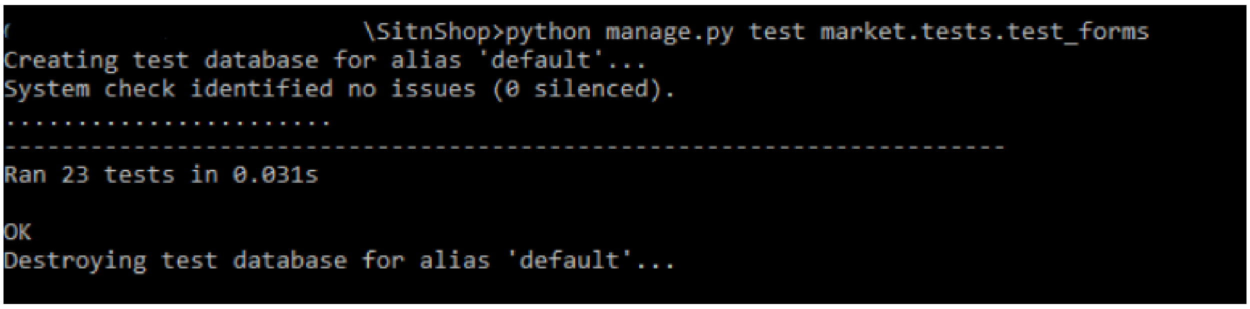


```
                     \SitnShop>python manage.py test market.tests.test_models
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
................................
----------------------------------------------------------------------
Ran 32 tests in 0.040s

OK
Destroying test database for alias 'default'...
```

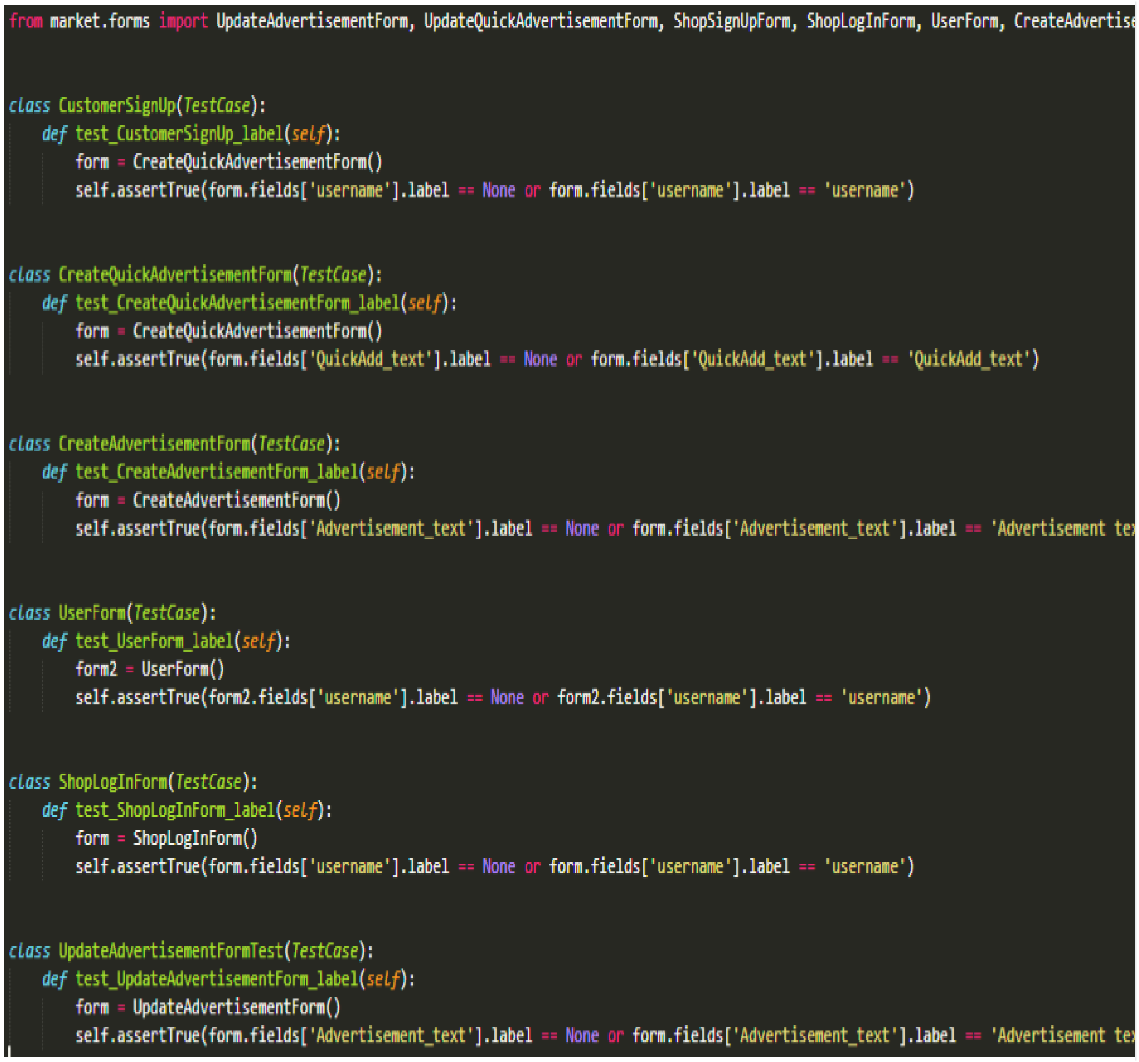


```
from django.test import TestCase
from django.urls import reverse

from market.models import Day,Shop

class DayModelTest(TestCase):
    @classmethod
    def setUpTestData(cls):
        # Set up non-modified objects used by all test methods
        Day.objects.create(day_name='today')

    def test_day(self):
        day = Day.objects.get(id=1)
        field_label = day._meta.get_field('day_name').verbose_name
        self.assertEquals(field_label, 'day name')

    def test_day_name_max_length(self):
        day = Day.objects.get(id=1)
        max_length = day._meta.get_field('day_name').max_length
        self.assertEquals(max_length, 64)

    def test_object_name_is_day_name(self):
        day = Day.objects.get(id=1)
        expected_object_name = f'{day.day_name}'
        self.assertEquals(expected_object_name, str(day))

    def test_day_delete(self):
        day = Day.objects.get(id=1)
        Day.objects.delete(day)

    def test_day_update(self):
        day = Day.objects.get(id=1)
        Day.objects.get(day).update('day_name'='tomorrow')
        field_label = day._meta.get_field('day_name').verbose_name
        self.assertEquals(field_label, 'tomorrow')

    def test_get_absolute_url(self):
        day = Day.objects.get(id=1)
        # This will also fail if the urlconf is not defined.
        self.assertEquals(day.get_absolute_url(), '/market/day/1')
```

4.4.2 Forms Testing



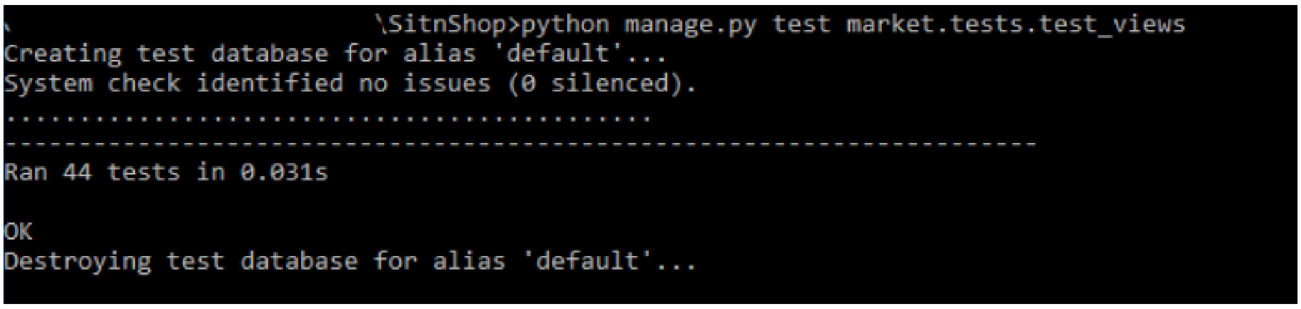
```
( [redacted] \SitnShop>python manage.py test market.tests.test_forms
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.......................
----------------------------------------------------------------------
Ran 23 tests in 0.031s

OK
Destroying test database for alias 'default'...
```



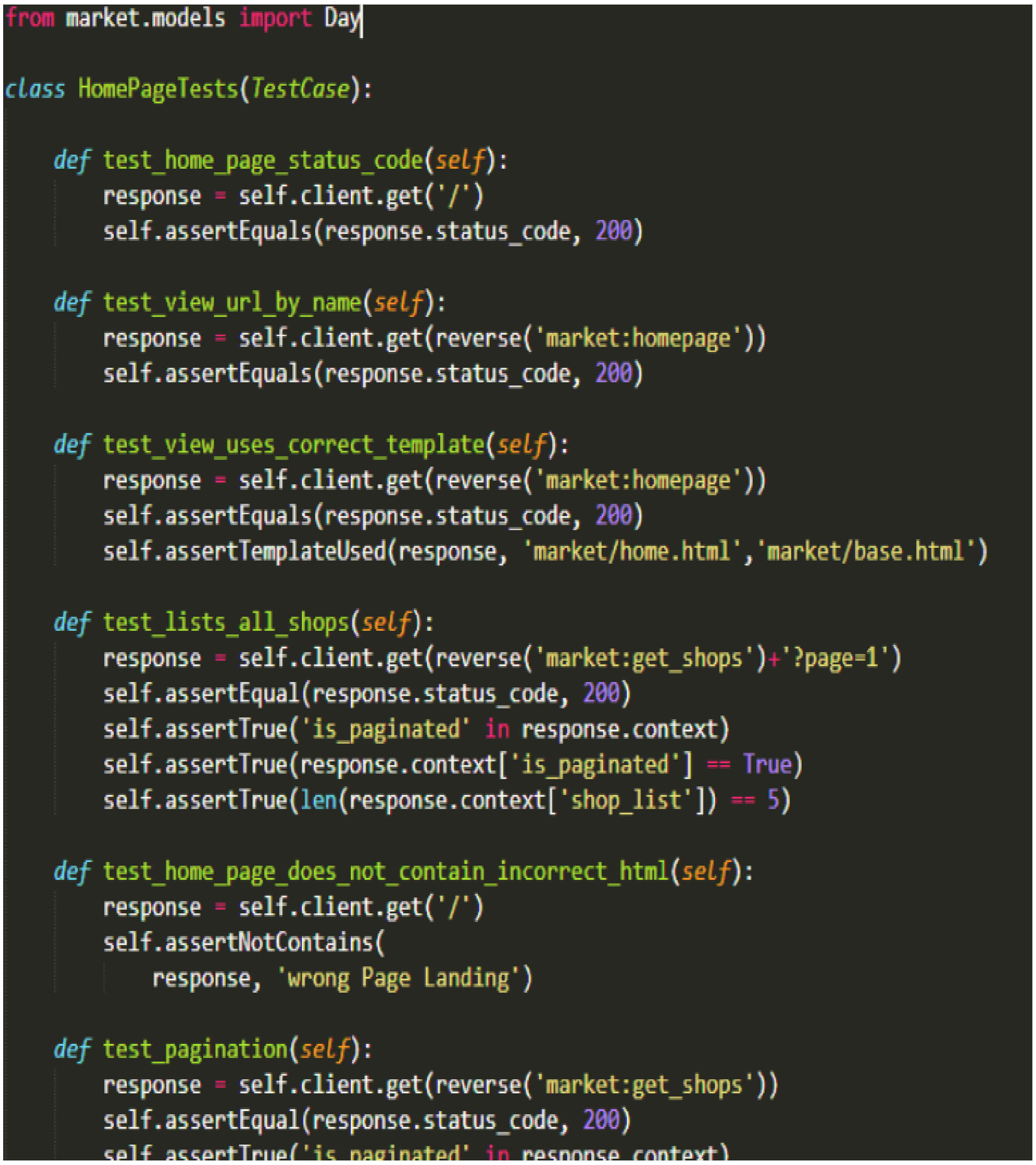
```python
from market.forms import UpdateAdvertisementForm, UpdateQuickAdvertisementForm, ShopSignUpForm, ShopLogInForm, UserForm, CreateAdvertise


class CustomerSignUp(TestCase):
    def test_CustomerSignUp_label(self):
        form = CreateQuickAdvertisementForm()
        self.assertTrue(form.fields['username'].label == None or form.fields['username'].label == 'username')


class CreateQuickAdvertisementForm(TestCase):
    def test_CreateQuickAdvertisementForm_label(self):
        form = CreateQuickAdvertisementForm()
        self.assertTrue(form.fields['QuickAdd_text'].label == None or form.fields['QuickAdd_text'].label == 'QuickAdd_text')


class CreateAdvertisementForm(TestCase):
    def test_CreateAdvertisementForm_label(self):
        form = CreateAdvertisementForm()
        self.assertTrue(form.fields['Advertisement_text'].label == None or form.fields['Advertisement_text'].label == 'Advertisement te


class UserForm(TestCase):
    def test_UserForm_label(self):
        form2 = UserForm()
        self.assertTrue(form2.fields['username'].label == None or form2.fields['username'].label == 'username')


class ShopLogInForm(TestCase):
    def test_ShopLogInForm_label(self):
        form = ShopLogInForm()
        self.assertTrue(form.fields['username'].label == None or form.fields['username'].label == 'username')


class UpdateAdvertisementFormTest(TestCase):
    def test_UpdateAdvertisementForm_label(self):
        form = UpdateAdvertisementForm()
        self.assertTrue(form.fields['Advertisement_text'].label == None or form.fields['Advertisement_text'].label == 'Advertisement te
```

4.4.3 Views Testing

To validate view behaviour we use the Django test Client. This class acts like a dummy web browser that we can use to simulate GET and POST requests on a URL and observe the response. Almost everything about the response, from low-level HTTP (result headers and status codes) through to the templates using to render the HTML and the context data passing to it. Also can see the chain of redirects (if any) and check the URL and status code at each step. This verifies that each view is doing what is expected.

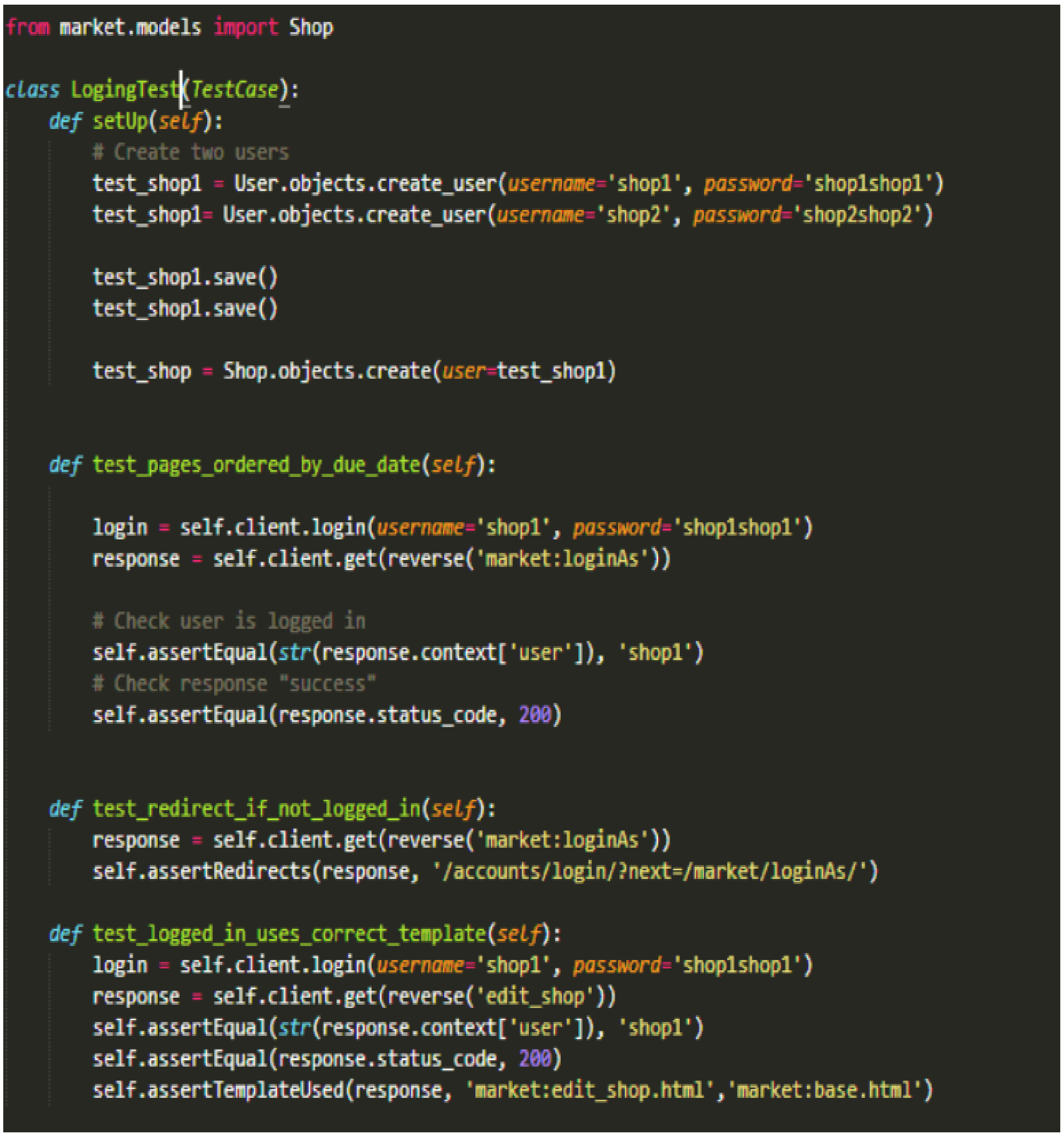


```
\SitnShop>python manage.py test market.tests.test_views
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
............................................
----------------------------------------------------------------------
Ran 44 tests in 0.031s

OK
Destroying test database for alias 'default'...
```



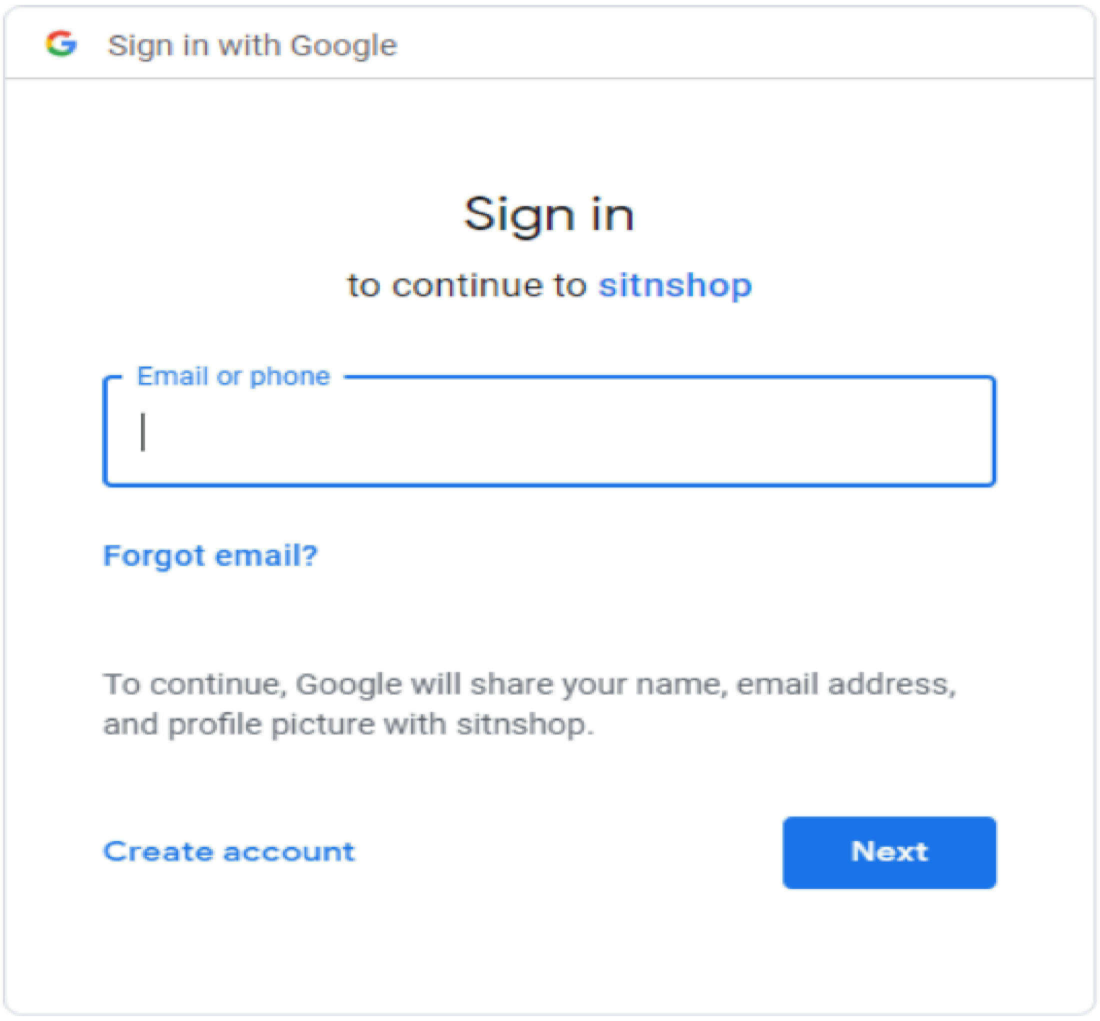
```
from market.models import Day

class HomePageTests(TestCase):

    def test_home_page_status_code(self):
        response = self.client.get('/')
        self.assertEquals(response.status_code, 200)

    def test_view_url_by_name(self):
        response = self.client.get(reverse('market:homepage'))
        self.assertEquals(response.status_code, 200)

    def test_view_uses_correct_template(self):
        response = self.client.get(reverse('market:homepage'))
        self.assertEquals(response.status_code, 200)
        self.assertTemplateUsed(response, 'market/home.html','market/base.html')

    def test_lists_all_shops(self):
        response = self.client.get(reverse('market:get_shops')+'?page=1')
        self.assertEqual(response.status_code, 200)
        self.assertTrue('is_paginated' in response.context)
        self.assertTrue(response.context['is_paginated'] == True)
        self.assertTrue(len(response.context['shop_list']) == 5)

    def test_home_page_does_not_contain_incorrect_html(self):
        response = self.client.get('/')
        self.assertNotContains(
            response, 'wrong Page Landing')

    def test_pagination(self):
        response = self.client.get(reverse('market:get_shops'))
        self.assertEqual(response.status_code, 200)
        self.assertTrue('is_paginated' in response.context)
```

Views restricted to logged in user testing (auth login testing)



```
from market.models import Shop

class LogingTest(TestCase):
    def setUp(self):
        # Create two users
        test_shop1 = User.objects.create_user(username='shop1', password='shop1shop1')
        test_shop1= User.objects.create_user(username='shop2', password='shop2shop2')

        test_shop1.save()
        test_shop1.save()

        test_shop = Shop.objects.create(user=test_shop1)


    def test_pages_ordered_by_due_date(self):

        login = self.client.login(username='shop1', password='shop1shop1')
        response = self.client.get(reverse('market:loginAs'))

        # Check user is logged in
        self.assertEqual(str(response.context['user']), 'shop1')
        # Check response "success"
        self.assertEqual(response.status_code, 200)


    def test_redirect_if_not_logged_in(self):
        response = self.client.get(reverse('market:loginAs'))
        self.assertRedirects(response, '/accounts/login/?next=/market/loginAs/')

    def test_logged_in_uses_correct_template(self):
        login = self.client.login(username='shop1', password='shop1shop1')
        response = self.client.get(reverse('edit_shop'))
        self.assertEqual(str(response.context['user']), 'shop1')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'market:edit_shop.html','market:base.html')
```

Other than these testing, manual testing were done at lots of times. Sometimes it was easier to test without written code, for an example when the customer login via social media authentication, manual testing was used.


Sign in with Google

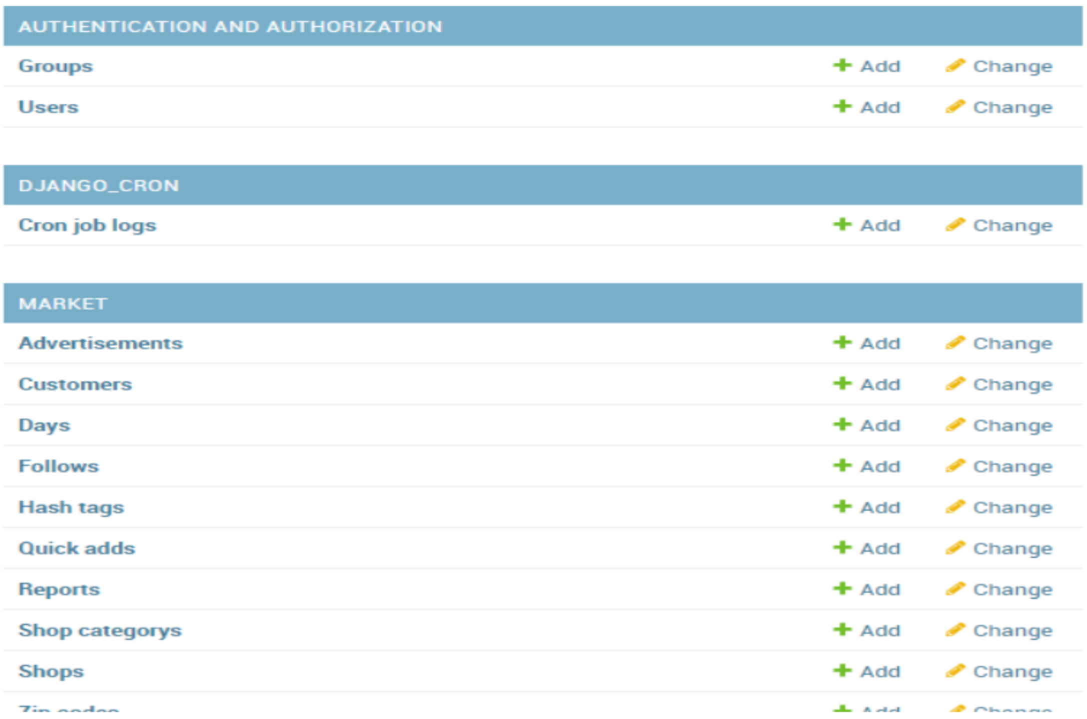
Sign in

to continue to sitnshop

To continue, Google will share your name, email address, and profile picture with sitnshop.

4.4.4 Admin Dashboard

Django admin panel is a handy tool to test and debug. For an example adding zip codes (which is a one time objective) was tested for the correct format of the zipcodes using admin panel. It also gives a lot of information about the structure of the site.



AUTHENTICATION AND AUTHORIZATION		
Groups	✚ Add	Change
Users	✚ Add	Change

DJANGO_CRON		
Cron job logs	✚ Add	Change

MARKET		
Advertisements	✚ Add	Change
Customers	✚ Add	Change
Days	✚ Add	Change
Follows	✚ Add	Change
Hash tags	✚ Add	Change
Quick adds	✚ Add	Change
Reports	✚ Add	Change
Shop categorys	✚ Add	Change
Shops	✚ Add	Change

4.5 Security

Since the time was limited for our project, we did not go detail into security. But there are minor implementations done in the security arena with the help of Django implementation.

4.5.1 Cross site scripting (XSS) protection

XSS attacks allow a user to inject client side scripts into the browsers of other users. This is usually achieved by storing the malicious scripts in the database where it will be retrieved and displayed to other users, or by getting users to click a link which will cause the attacker's JavaScript to be executed by the user's browser. So, instead of explicitly using custom made template, we used Django template thus it was avoided.

4.5.2 Cross site request forgery (CSRF) protection

CSRF attacks allow a malicious user to execute actions using the credentials of another user without that user's knowledge or consent.

Django has built-in protection against most types of CSRF attacks, providing you have enabled and used it where appropriate

4.5.3 SQL injection protection

SQL injection is a type of attack where a malicious user is able to execute arbitrary SQL code on a database. This can result in records being deleted or data leakage.

Django's querysets are protected from SQL injection since their queries are constructed using query parameterization

4.6 System testing

The full system was tested after the above tests and it involved automated and manual testing. System testing was reviewed under following categories,

1. User Interaction
2. Database Interaction
3. Network performance and Web server

each of the above categories had an acceptable result.

4.7 GUI testing(For Different Resolutions)

Gui Testing was done manually for the most common resolutions of devices that are currently in use and the adjustments were made.

Tested Resolutions - 360x640, 411x731, 411x823, 320x568, 375x667, 414x736, 375x812, 768x1024, 1024x1366

5 ALGORITHM DESIGN

5.1 Suggested algorithm

The task of the algorithm is to find the next shop to suggest for the user on his time line based on his past preferences. Users preferences will be captured by the ratings he give for a shop when it is shown in his time line and by the fact weather he marks some shop as visited. Most part of the final suggested algorithm follows the 2003 paper named Bayes Meets Bellman: The Gaussian Process Approach to Temporal Difference Learning [1]. The idea of Markov Decision Processes was used to design the algorithm. The idea is to represent the shops that are supposed to be suggested next as the possible actions leading to the next states and consider the currently displayed shop as the current state. The rating and the fact user marks some shop as visited is modified as a rewards. This now the problem can be modified into a Markov Decision Process for each user.

5.1.1 State Representation

The state representation of a shop was made to primarily capture it's geographical location and the property of the shop. Thus a trade off is made between these two vectors with more priority given towards the geographical location. With $i > j$

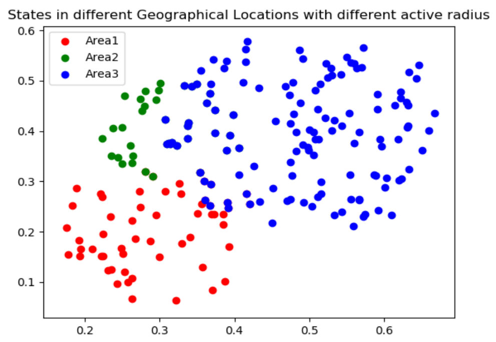
$$StateVector = i * LocationVector + j * CharactersiticVector$$

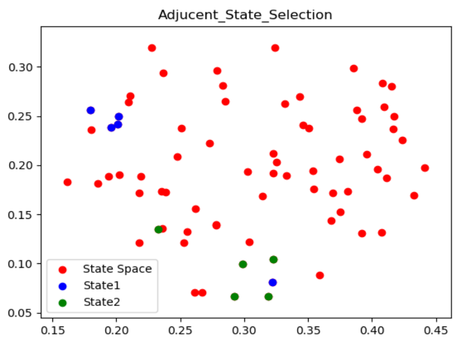
Even though the idea of using all the shops that are available in the database as the adjacent states sounds profound it attracts a load of computational resources and optimization which can be expensive given this is a product aimed for local users. Thus only having the adjacent shops within a radius as the adjacent state space for a user reduces the complexity of the problem while not compromising the accuracy. Thus the total state space is defined as the shops around a certain radius around the user.

Now in a given state space we select the states with the lowest euclidean distance as the adjacent state space. And to avoid the states getting stuck at a particular position a random state is selected from the state space as an adjacent state.

$$Distance = (CurrentState - State)^2$$

$$AdjucentState = argsort(Distance)[: n] + RandomState$$



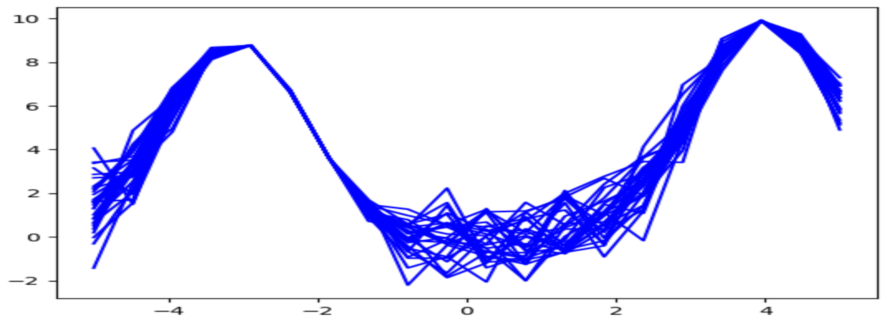


5.1.2 Algorithm Description

A Markov Reward Process is made of a tuple S, R, γ which is State, Reward, and decay parameter. A value function V for a state s is defined as

$$V(s) = E(\sum_{t=0}^{\infty} \gamma^t * R(s_t, s_{t+l}) \mid s_0 = s)$$

where the expectation is taken over the policy π based state transition $p^{\pi}(s_{t+1} \mid s_t)$. The action that leads to a state with the most V is selected in a solved Markov Decision Process. There are several ways to obtain the solution, optimal policy π_*. One such method is to use a value function estimator that is trained such that when given a particular state representation it spits out the corresponding value function. Gaussian Process is the value function estimator used in our case because of the confidence interval it gives apart from the value estimate.



The training data is provided to the estimator by TD(0) update as per the paper[1] which is

$$v(s_t) = v_t + \gamma * \delta_t$$
$$\delta_t = r_t + \gamma * v(s_{t+1}) - v(s_t)$$
$$r_t = R(s_{t+1}, s_t)$$

The TD(0) rule can be written as a matrix format as follows

$$H = \begin{bmatrix} 1 & -\gamma & 0 & \dots & 0 \\ 0 & 1 & -\gamma & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & -\gamma \end{bmatrix}$$

$$R_{t-1} = H_t V_t + N_{t-1}$$
$$N_{t-1} = noise$$

Then a Gaussian process can be fit as mentioned in the paper [1] for the equation as follows

$$\begin{bmatrix} [R_{t-1} \\ V(s) \end{bmatrix} \sim N \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} H_t K_t H_t^T + \Sigma_{t-1} & H_t k_t(s) \\ (H_t k_t(s))^T & k(s,s) \end{bmatrix}$$

where $k(s,s), k_t(s), K_t$ are kernel dependent functions as defined in the paper [1].Due to the complexity of the Gaussian Process to be used as it is the paper [1] suggests an online algorithm which performs at $O(n^2)$ compared to the $O(n^3)$ complexity the original performs at. To reduce the no of states that are involved in the estimation process it uses a sparsification test where it takes the least square solution for the Hilbert space transformation of the new state from the available states (also as a Hilbert space transformation) and if the error in the estimation is lesser than than a threshold then it let them represent the new state. Or else it adds the new state to the existing states. The least square solution and error is as follows

$$argmin_a(a^T K_{t-1} a - 2a^T k_{t-1}(s_t) + k_{tt})$$
$$error = k_{tt} - k_{t-1}(s_t)^T a_t$$

Then as per the respective parameters mentioned in the paper for the case of error being less than the threshold

$$H_t = \begin{bmatrix} H_{t-1} \\ a_{t-1} - \gamma * a_t^T \end{bmatrix}$$

$$\alpha_t = \alpha_{t-1} + c_t/s_t(\Delta k_{t-1}^T \alpha_{t-1} - r_{t-1})$$

$$C_t = C_{t-1} + 1/s(c_t c_t^T)$$

For the case of error being less than the threshold

$$H_t = \begin{bmatrix} H_{t-1} & 0 \\ a_{t-1}^T & -\gamma \end{bmatrix}$$

$$\alpha_t = \begin{bmatrix} \alpha_{t-1} + c_t/s_t(\Delta k_{t-1}^T \alpha_{t-1} - r_{t-1}) \\ \gamma/s_t(\Delta k_{t-1}^T \alpha_{t-1} - r_{t-1}) \end{bmatrix}$$

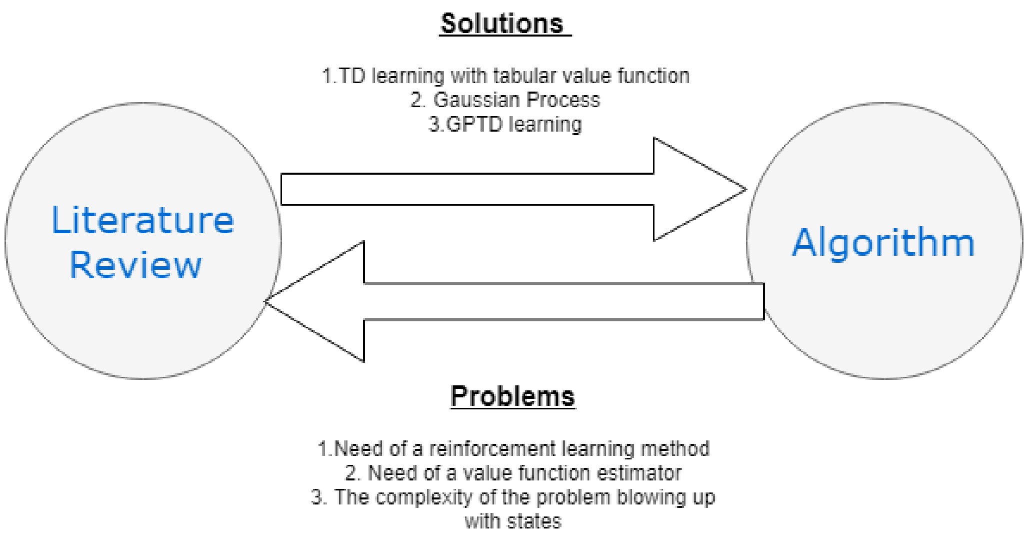
$$C_t = \begin{bmatrix} C_{t-1} + 1/s(c_t c_t^T) & \gamma/s_t(c_t) \\ \gamma/s_t(c_t)^T & \gamma^2/s_t \end{bmatrix}$$

Then the value function and the confidence interval can be found in $O(n^2)$ as follows

$$valuefuncton = k_t(s)^T \alpha_t$$

$$confidenceInterval = k_{ss} - k_t(s)^T C_t k_t(s)$$

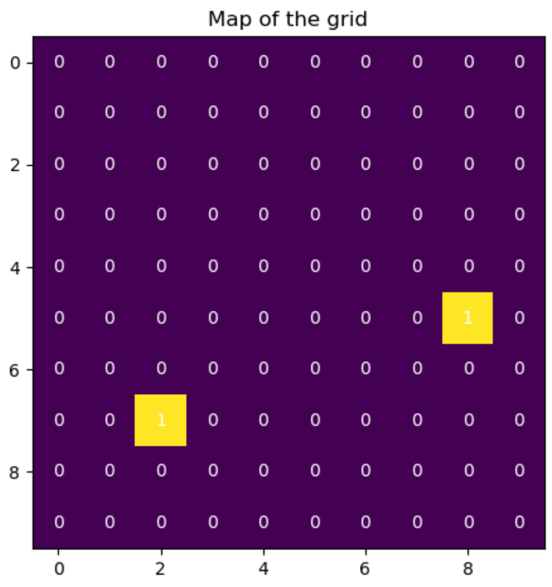
5.1.3 Agile Development during the algorithm phase


Solutions
1.TD learning with tabular value function
2. Gaussian Process
3.GPTD learning
Literature Review
Algorithm
Problems
1.Need of a reinforcement learning method
2. Need of a value function estimator
3. The complexity of the problem blowing up with states

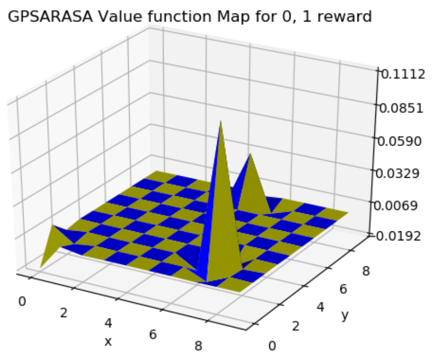
When we started the problem we in order to make it an interactive real time solution we had to go towards reinforcement learning. Having to decide with only the current shop user is viewing did not contradict with the reality. Thus we choose t model it as Markov decision process. Initially we choose $TD(\lambda)$ method with tabulating the value functions for all states and updating by the running average. Even though this method guaranteed a convergence in proof we needed a tabulation of every state. So the memory grew along with the no of states. Thus we had to find a value function approximation even though we were aware of them no guaranteeing convergence to an ultimate optima. We choose to approximate the function with Gaussian process along with $TD(\lambda)$ updates. The availability of a confidence interval estimate from the Gaussian Processes along with a value function estimate prompted us to use it. But the we faced the issue with we having to computationally having to do the inverse of a large state sized matrix every time step. So we after a literature review we found the 2003 paper which proposed an algorithm which not only bring the computation down but also reduces the complexity of the memory space as much as we want by letting us pick a threshold which would let us approximate a given state by using the already met states. Since is gave us a way to do online update and also to sparsify we choose this as the method as our long term solution for the given problem.

5.1.4 Test

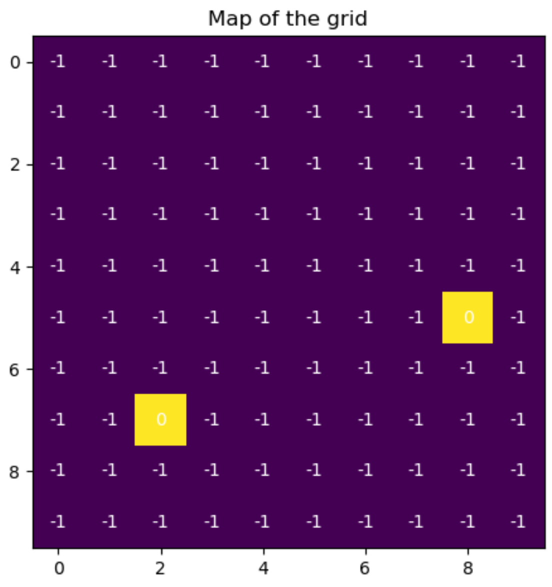
In order to test the algorithm in the absence of real time data we have decided to test it on a simulation. We choose total adjacent states as 4. Then they were mapped into a 2D grid and then the tests were run on that grid. Now an example of zero indexed 10*10 grid is considered.As an extreme case it was assumed that the user kept on avoiding the ratings until he visits the shops at position (5, 8) and (7,2). Thus the reward system is initially modified as give a reward of 1 at those 2 states and 0 elsewhere.



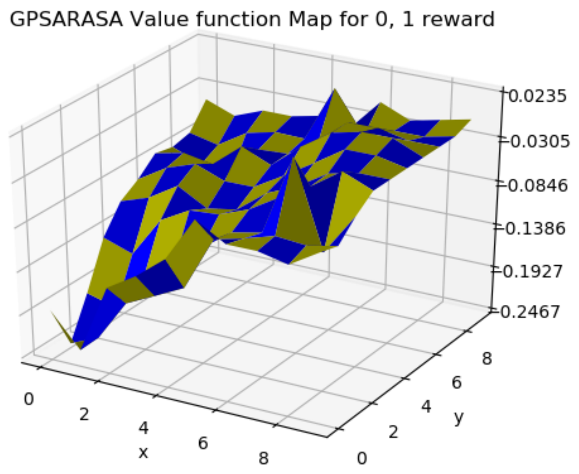
And the value function result is obtained as follows.



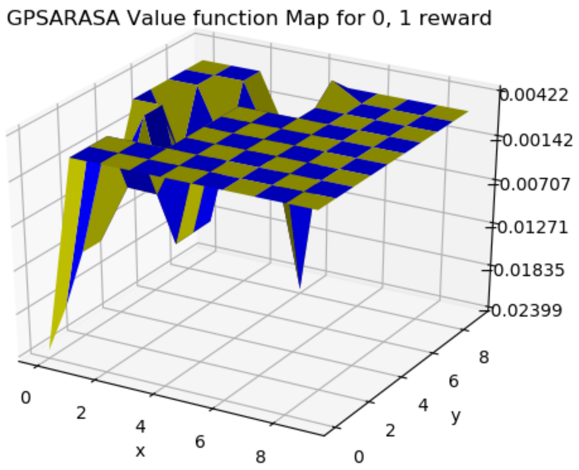
The same system was now made to penalize itself with -1 every time it didn't give the user what he wanted and was given a 0 reward when it got to the user's wanted stage.



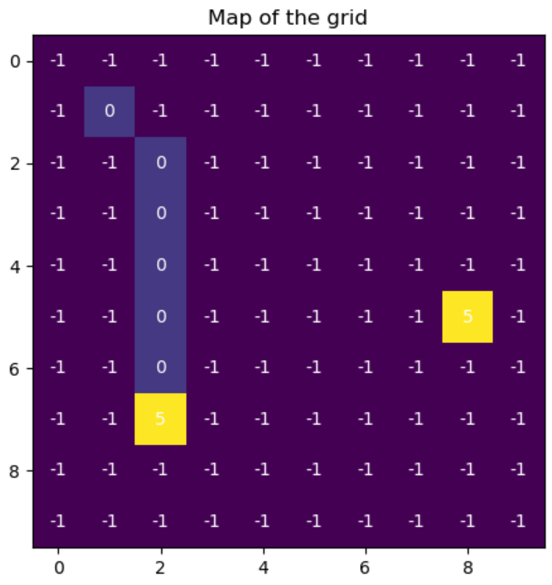
And the results are as follows.



Important fact to observe in are the peaks of the graph and the dynamics of the graph leading to those peaks. In both graphs the algorithm peaked at the customer's favoured places and near (0, 0) when customer doesn't have any favourite it fell down and gradually increased towards the favoured places. According to the results we can interpret the systems work as follows. That is when the user is in a random shop the algorithm will maximize the value function and then climb towards the visited shop. Thus the user can view advertisements from the similar shops and also the new offer advertisements from the already visited shop. The value function at the intermediate state of the crawling shows how it penalizes the undesired states that are visited. Thus on the next stage it would explore more.

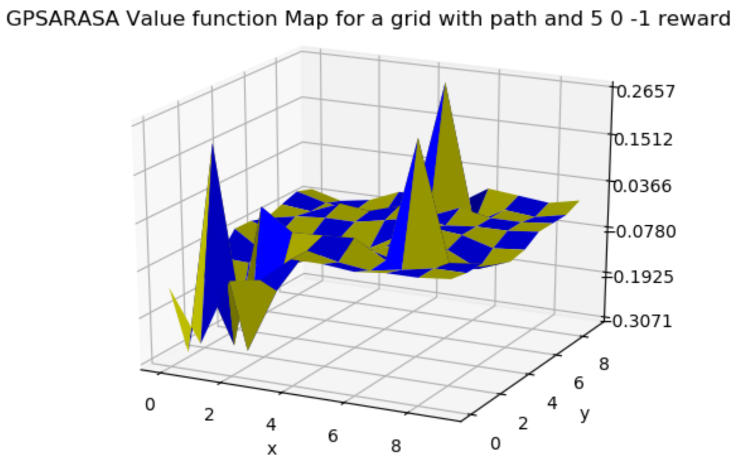


Furthermore in all the diagrams the scale of value functions doesn't change by a greater margin due to our usage of rewards $< \|1\|$. Thus the estimator can adapt the change with ease when the user's preference changes thus enabling to add dynamics.Next if we assume that the user is giving ratings along a path and visiting the above mentioned states. To simulate that scenario we let our grid provide a reward of 5 for both visited states and then a reward of 0 for a certain path and -1 for the rest of the states and let the user start from (0, 0) state.



-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	0	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	0	-1	-1	-1	-1	-1	-1	-1
-1	-1	0	-1	-1	-1	-1	-1	-1	-1
-1	-1	0	-1	-1	-1	-1	-1	-1	-1
-1	-1	0	-1	-1	-1	-1	-1	5	-1
-1	-1	0	-1	-1	-1	-1	-1	-1	-1
-1	-1	5	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

The value function we got was



It has made smaller spikes for the path we defined and larger spikes for the visited shops. Thus capturing more information with ratings. In all these cases a higher reward was proposed for visited shops in the hindsight that a customer would want to know more of the new offers from the shops he has visited. These dynamics can be varied given what is the priority and the reward can also be treated as a hyper parameter of the algorithm. Furthermore in all the above processes the algorithm represented the states by using 30-40 states instead of the whole 100 states thus solving part of the third problem mentioned under the previous section

6 REFERENCES

1. Yaakov Engel, Shie Mannor, Ron Meir Bayes Meets Bellman: The Gaussian Process Approach to Temporal Difference Learning (2003)

2. `https://wsvincent.com/django-testing-tutorial/`

3. `https://docs.djangoproject.com/en/2.1/topics/auth/default/`