STW302CEM Group Work

Technical Report
for CAKE WEBSTORE: A cake ordering web app delivered with Scrum

Authored by

Team Phoenix:
Sushan Tha Shrestha [9638073]
Netra Dahal [9636138]
Biju Ale [9638350]
Romisha Thapa [9637412]

Assignment report submitted for the partial fulfilment of BSc (Hons) Computing degree.

Module Title	Agile Development
Module Code	STW302CEM
Course Title	BSc (Hons) Computing
Course Code	EECU033
Submission Date	04 Nov, 2019
Submitted To	Sudeep Bajimaya (Module Leader)

Faculty of Engineering, Environment and Computing
School of Computing, Electronics and Mathematics
Coventry University

Contents

List of Figures

1 Project background

Cake Web-store is a web application developed in Django web framework, made exclusively for the management and dealings of cake shop related inquiries and orders. It will facilitate customers to order cakes online, track the progress of the order and when finished cake can be collected from the shop. Major features include the following:

- Cake Inventory: The customers can view different categories of cakes and their details suited to occasions from weddings to birthdays. The admin of the store is able to perform creation,update, deletion, and retrieval of cake details.
- Customer account: To place an order, the customer first needs to register on the web app. After successful registration, log in details will be provided to customer enable order placement.
- Order management: After order confirmation, a summary report of customer orders along with billing information will be provided to the customer. The admin is able to view customer wise order placements.
- Order history: The customer is able to view the details of their past transactions.
- Expiry enabled discounts: The admin is able to set discounts on cakes along with the expiry date for the valid period of the discount.

The complete source code along with the workflow is available at `https://github.com/BijuAle/STW302CEM`

2 Version Control

There can be an assortment of 'levels of engagement' with a version control system (Narębski 2016: 28). Among them, we have employed Git, the well established, version control system for our project for three purposes. (1) For collaborative development. (2) For change monitoring. And (3) For archaeology.

Figure 1: Logos of a prominent distributed VCS - Git and its repo - GitHub.

2.1 Collaborative Development

Git enables efficient concurrent work among team members on a common piece of software. In our case, 4 members subscribed to the same repository in GitHub as 'remote repo'. Their local code-base would be shared among each other and the versions of it shared on the remote. In addition, in a feature-branch workflow employed by our team, each subscriber creates a new branch from the master branch to work on a feature (Laster 2017: 54), which is merged after completion back to the master branch. While working with multiple collaborators, version control systems like Git ensures that the changes made between the shared common files do not conflict with each other.

3 Analysis of Workflows

Collaborative development assumes one of many available workflows. In any given project, there may be few, or many developers. There may be a project maintainer, or if the project is large, there can even be many subsystem maintainers. In other cases, one might work in a tight but efficient team. And in many cases, one might prefer convenience for an external collaborator to propose changes such as bug-fixes, or refactors (Narębski 2016: 27). These are various scenarios that call for the selection of a git workflow:

1. Centralized Workflow
2. Feature-branch Workflow
3. P2P(peer-to-peer) or Forking Workflow
4. Maintainer or Integration Manager Workflow
5. Hierarchical or Dictator-Lieutenants Workflow

3.1 Centralized Workflow

Among the two types of repositories: (1) Bare and (2) Non-bare. The former is used for synchronizing development outputs among multiple collaborators while the later is used for solo development for persisting newer histories (Loeliger and McCullough 2012: 31). Moreover, bare repositories contain no working directory but the innards of git whereas the regular non-bare repositories contain both the git files and the working repositories from where the changes are loaded to the staging area. In git, the command to create bare repository is: 'git init –bare project.git'

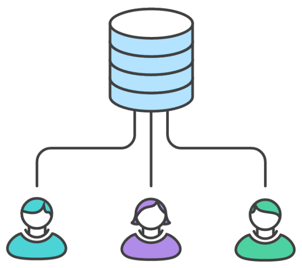


Figure 2: Centralized workflow (Atlassian Corporation Plc 2019).

The centralized workflow contains one bare repository. Here, each technical member possesses his/her clone of the central repository, used to develop revisions of the software. Once the revision is ready, the member pushes the changes to the central repo, hence integration is distributed. Centralized workflow has the following advantages and disadvantages.

- The foremost advantage of this workflow is simplicity. It is simple to set up and familiar to developers with prior experience in the centralized version control systems. It also provides centralized backup and security in the form of access control. This might be a good choice for a private project with a few members.
- The major disadvantage here is the single domain of failure shared by all collaborators. If there is a problem with the main trunk then there is no way to resolve it by synchronization (Narębski 2016). And each developer making changes available for others might require updating one's own repo first and then merging other's changes. Another issue with this workflow is the assumption that all members are trustworthy with access to the central repo.

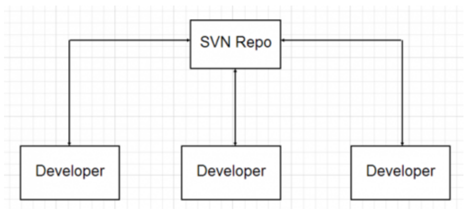


Figure 3

The Apache Subversion or SVN is a popular centralized based VCS (Apache 2019). Git, on the other hand, is a prominent distributed-version control system and allows varying degrees of distribution models for collaboration (Santacroce 2017).

3.2 Feature-branch workflow

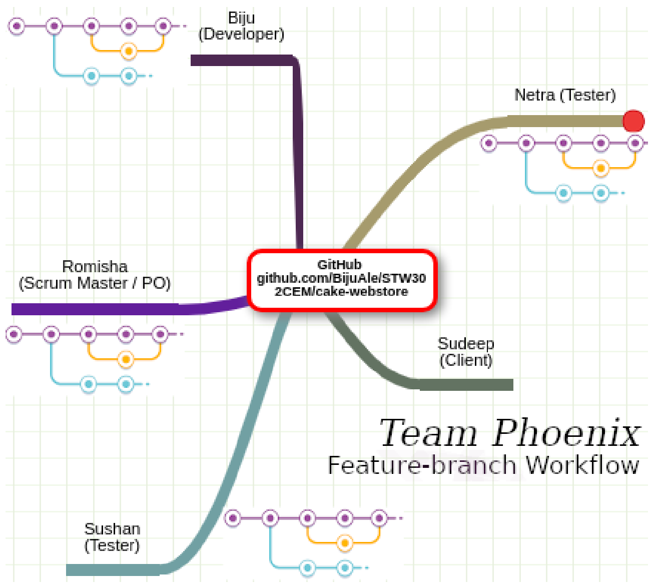


Figure 4: Our team's preferred Feature-branch Workflow

Our project uses git over svn for the reason that git enables distributed workflow and seamlessly integrates the feature-branch workflow seamlessly. The main idea with Feature-branch workflow is a separation of feature development onto individual self-sustained branches (Atlassian Corporation Plc 2019), as shown below in figure 5.Following are advantages and disadvantages.

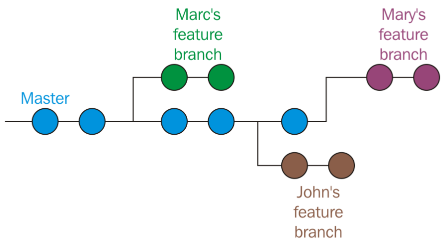


Figure 5: Example of Feature-branch workflow (O'Reilly Media, Inc. 2019).

- This encapsulation keeps the main codebase or the master branch unperturbed by unintended changes. This ensures the master branch is always free of broken-code. This is the main advantage of the feature branch workflow (Geisshirt et al. 2018: 74).
- Another advantage is the stimulation of feedback among developers be leveraging the pull requests (Atlassian Corporation Plc 2019; O'Reilly Media, Inc. 2019). They give the chance to refactor and finalize a feature before it gets integrated into the main software. If one wants help with a

specific feature, then a pull request can be generated which notifies other members to respond.

- Feature-branch also allows targetted testing (O'Reilly Media, Inc. 2019) which means each feature can be tested without cascading the effects onto other features which are marked as working correctly.

The cons of this workflow is not so significant as the benefits outweigh them. However there are minor disadvantages such as:

- Merging binary files, and images can be difficult to achieve with some version control systems, which can be time-consuming (Loeliger and McCullough 2012: 121).
- If a developer wants to add to the published feature branch, he or she may want to first merge the feature to their current branch, or wait until it is merged to the main branch. So, there may be a possibility that the other branch can go out-of-date.

3.3 P2P(peer-to-peer) or Forking Workflow

The inverse of a centralized workflow is the peer-to-peer or what is commonly known as the forking workflow (Narębski 2016: 129). As shown in figure 6 below, each member has a public bare repository in addition to the private working repository that is non-bare and contains the working directory.

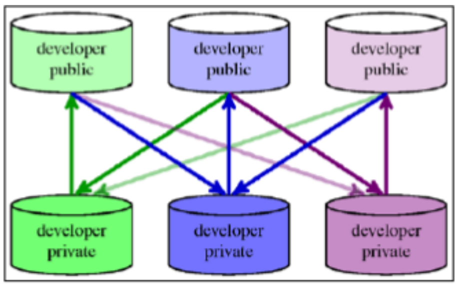


Figure 6: Forking Workflow Model.

The changes that developers make are pushed to their own public repositories. To merge the changes from others, one requires to to fetch from other's public repositories. The advantages and disadvantages of this workflow are given below.

- There is no need for central repository to incorporate the changes; it is a complete distributed workflow.Likewise, developers are not complied to integrate at a specific time; one can merge at one's leisure time. Narebskii notes that this workflow is suited best for organic team requiring little setup (Narębski 2016: 129).
- The disadvantage is the absence of 'canonical' or official versions, a centralized management, and the requirement for developers to track multiple repositories (Narębski 2016: 129). The setup needs to be such that each

workstation can talk to each other which is challenging in terms of networking, relative to the easy setup of centralized workflow. Also, as the size of the team grows, the collaboration and network is made more complicated. This is also visible in figure 6 shown above.

3.4 Maintainer or Integration Manager Workflow

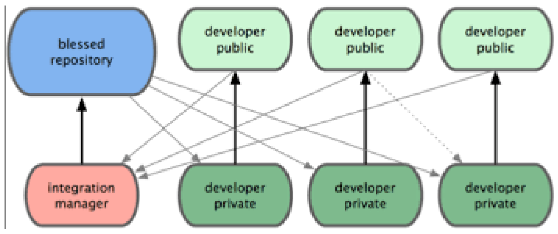


Figure 7: Maintainer Workflow Model (Git SCM 2019).

As discussed above, the P2P workflow lacked canonical version of a project, that could be used by-non-developers (Narębski 2016: 129). Another con was that each developers were responsible for their own integration. The maintainer workflow is designed to mitigate these issues with the P2P workflow.

If we extend the P2P workflow by promoting one of the public repositories as ‘canonical’ or official, and assign one of the developers in the team as a integration manager, then we achieve the maintainer workflow. For this, reason, maintainer workflows are also known as integration manager workflow (Laster 2017; Santacroce 2017).

In this workflow, after the developer pushes his changes to his public repository, he notifies the maintainer regarding his changes via a pull request. The maintainer, then pulls the changes into his working directory and finally pushes the merged changes to the canonical repository for everyone to realize the changes.

- The advantage of this workflow is the availability of an official version of the codebase. And thus, the developer need not wait for integration as the maintainer is responsible for pulling their changes. This workflow is best suited for open-source projects which comprises of large team. Since social consensus decides the maintainer, there is an easy switch temporarily or permanently i.e. forking of a project.

- The major disadvantage of this workflow is that the maintainer can be the bottleneck to the project since he is solely responsible for integration management (Narębski 2016: 131). Therefore, for a very large project with huge organic team around the globe, such as Linux kernel development this workflow is not suitable. Rather, hierarchical workflow is preferred.

3.5 Hierarchical or Dictator-Lieutenants Workflow

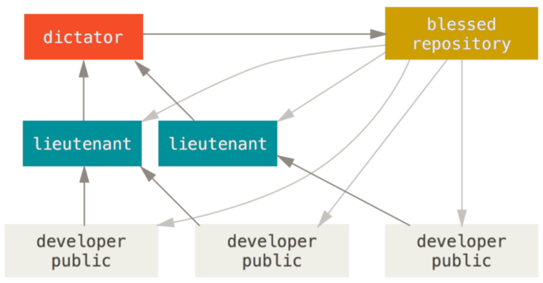


Figure 8: Hierarchical Workflow Model (Git SCM 2019).

The hierarchical workflow is the extension of the maintainer workflow, generally employed by huge projects comprising of hundreds of distributed collaborators. Unlike the maintainer workflow, here there are multiple integration managers. The main maintainer is aided by other assistant maintainers who are responsible for managing subsystems. These maintainers are known as the 'lieutenants'. And the main maintainer is called the 'benevolent dictator'. As shown in figure 8, the benevolent dictator's public repository called the 'blessed repository' is pulled by both the developers and the subsystem maintainers, resulting in a hierarchy or network of repositories. Respective lieutenants are sent changes by their developers who first pull the updates from the blessed repository. Lieutenants merge the changes in their respective responsibility domains. The dictator who is the master maintainer pulls from the lieutenants and sometimes directly from developers too. The dictator then pushes the merged changes to the blessed repository. The pros and cons of this workflow are the following.

- This workflow is best suited for huge collaborative project such as the Linux kernel development. The advantage is that the dictator or the master maintainer or leader can delegate integration workload, which is useful for hierarchical projects with many subsystems or modules.
- The disadvantage of this workflow is complexity. If it is applied to the simple projects then, it can be unnecessarily complicated to manage.

4 Automated Testing

4.1 Performing Unit-Test

Django REST framework was used to develop the back-end of the project. Testing Django REST views and testing the regular Django views are different. Following test classes are provided by the Django REST framework

- APISimpleTestCase
- APITransactionTestCase
- APITestCase

- APILiveServerTestCase

These test case classes implement the common interface available in Django's TestCase class. However, instead of Django's client they, use the HTTP client to specifically test the API views. As a result, JSON format is specified during implementation when testing API client requests, for example, as in self.client.post(url, data, format = 'json'

All these classes down the line, implement the standard python library 'unittest' a JUnit inspired testing framework which has a familiar flavour from major unit testing frameworks in other languages. Therefore the comparative testing of the assertion of expected value was the basic style in writing the unit-tests (Django Software Foundation 2019).

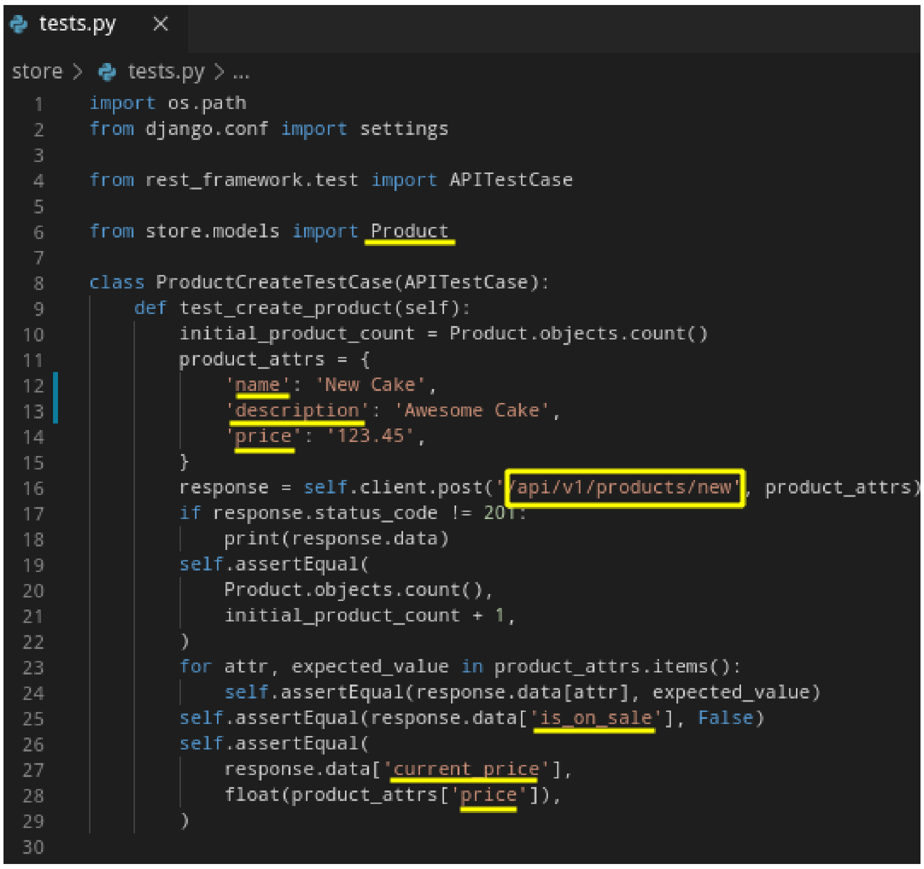
4.2 Performing Functional Tests

As we build our methods driven by unit-tests, we would want to test higher sets of features or functions that are comprised of these smaller methods. These smaller units are tested by the APITestCase described above. For testing the higher units of functions such as user registration and CRUD operations with renderings of form elements and events, we have used the Selenium framework.

Selenium is a browser automation tool used largely for functional testing web applications. It supports a variety of languages such as Python across various web-browsers and operating systems (Zhimin 2018: 17). In order to create automated functional suites and tests for our project, we have used the Selenium WebDriver, which is a collection of programming language-specific bindings to drive a web browser.

4.3 Test-Driven Development

The functional and unit test implementation discussed above were done in a TDD approach which is the basis for Agile testing strategies. The unit tests were written first. The test was executed in the absence of the respective code base. As the test fails, the code was supplied to pass the test. Similarly, with the front end, Selenium was used to drive the HTML templating in Django. Front end tests were written first.



```python
import os.path
from django.conf import settings

from rest_framework.test import APITestCase

from store.models import Product

class ProductCreateTestCase(APITestCase):
    def test_create_product(self):
        initial_product_count = Product.objects.count()
        product_attrs = {
            'name': 'New Cake',
            'description': 'Awesome Cake',
            'price': '123.45',
        }
        response = self.client.post('/api/v1/products/new', product_attrs)
        if response.status_code != 201:
            print(response.data)
        self.assertEqual(
            Product.objects.count(),
            initial_product_count + 1,
        )
        for attr, expected_value in product_attrs.items():
            self.assertEqual(response.data[attr], expected_value)
        self.assertEqual(response.data['is_on_sale'], False)
        self.assertEqual(
            response.data['current_price'],
            float(product_attrs['price']),
        )
```

Figure 9: TDD example: Unit Test without the Code.

As highlighted in figure 9 abvove, the test requires model of Product to be created along with the attributes. Without supplying the model, this test would not pass. Therefore to pass the test, Product model was created after the failed test. In this manner, all unit were written to drive the source code.

4.3.1 Advantages of TDD

In TDD, the testing is the actual coding. And the source code is a product of the testing being conducted (Beck 2003). By making testing the aim of the project, we are performing test-driven development. TDD has following advantages (Percival 2017).

- As the application takes forms and evolves, the testing is simultaneously performed while covering all the features of the software, without the need for assigning explicit timeblock for testing.
- Gives the opportunity to introduce developers engage in pair-programming, and ping-pong programming.
- Ensures all parts of the codebase are covered without leaving behind any holes or missed functionalities.

- It is the basis for behavior driven TDD which ensures that the software behaves as intended by the user.
- Fail early and fail fast principle in TDD allows developers to pick up bugs in their code as they are being generated rather than having to discover accumulated errors during the integration phase.
- TDD resolves the problem of gold-plating - the silent build-up of unspecified features and functionalities.

4.4 Acceptance Testing

Acceptance testing is a testing method that is done to determine if the software system has met the requirements or not. The main need for the test is to examine the system's compliance with the given requirements and confirm if it has met the requirements for deployment to the users. As it is one of the final stage of testing software, it occurs before accepting the software. User tests as what will occur in the real scenarios and confirm if the product meets all the requirements.

Acceptance testing helps to encourage collaboration between the customers and developers as it entails that the requirements should be pointed out. It allows a clear and transparent contract between developers and client and the product which passes the test is considered enough for it. As they confirm that the requirements are met, the satisfaction of the customer increases. Confidence of the customer grows as all things are tested and already know how the software will behave in real scenarios and if any critical problem will arise when least expected. The quality standard of the software is already pre-defined at the early stage of project development. Stakeholders can better understand the target users by using the information collected during the acceptance test.

4.5 Automated Tests and Integration

Python Paver and Jenkins were the tools chosen be used to automate tests and build. Paver can be used to automate the test scripts. And Jenkins can execute them at every check-ins. We can also gain instant feedback regarding our test execution status. Jenkins can not only automate build process but also generate statistics and graphs for each build connecting it with PyLint Report.

Although these tools were chosen at first, we were unable to implement them in our project as much of our time were spent tp resolve technical issues and team issues. We had to learn most of the tools used in our project as they were new to us. Thus continuous integration was done manually by regularly pulling the master branch and building the codebase.

-

References

Apache (2019). *Apache Subversion.* URL: `https://subversion.apache.org/` [2 October 2019].

Atlassian Corporation Plc (2019). *Git Feature Branch Workflow | Atlassian Git Tutorial.* URL: `https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow` [3 October 2019].

Beck, Kent (2003). *Test-driven development: by example.* Boston: Addison-Wesley.

Django Software Foundation (2019). *unittest — Unit testing framework — Python 3.8.0 documentation.* URL: `https://docs.python.org/3/library/unittest.html` [5 September 2019].

Geisshirt, Kenneth et al. (2018). *Git Version Control Cookbook.* Birmingham, UK: Packt Publishing.

Git SCM (2019). *Git - Distributed Workflows.* URL: `https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows` [2 October 2019].

Laster, Brent (2017). *Professional Git.* Indianapolis, Indiana: John Wiley & Sons.

Loeliger, Jon and Matthew McCullough (2012). *Version Control with Git.* 2nd edition. Beijing: O'Reilly.

Narębski, Jakub (2016). *Mastering Git.* Birmingham, UK: Packt Publishing Ltd.

O'Reilly Media, Inc. (2019). *Feature Branch Workflow.* URL: `https://www.oreilly.com/library/view/git-essentials-/9781787120723/b9b6d860-ad8c-4b41-affb-94662149f3fa.xhtml` [4 August 2019].

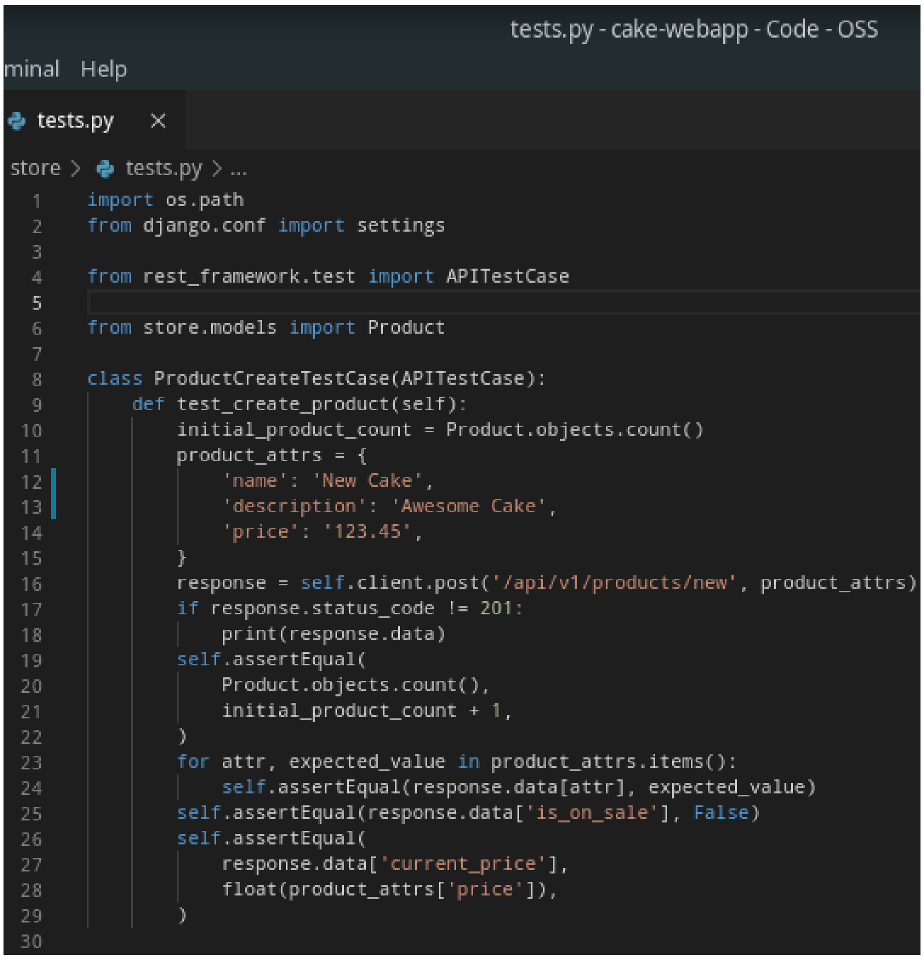
Percival, Harry (2017). *Test-driven development with Python.* Second edition. Sebastopol, CA: O'Reilly Media.

Santacroce, Ferdinando (2017). *Git Essentials.* Birmingham, UK: PACKT Publishing Ltd.

Zhimin, Zhan (2018). *Selenium WebDriver Recipes in Python.* British Columbia, Canada: Leanpub.

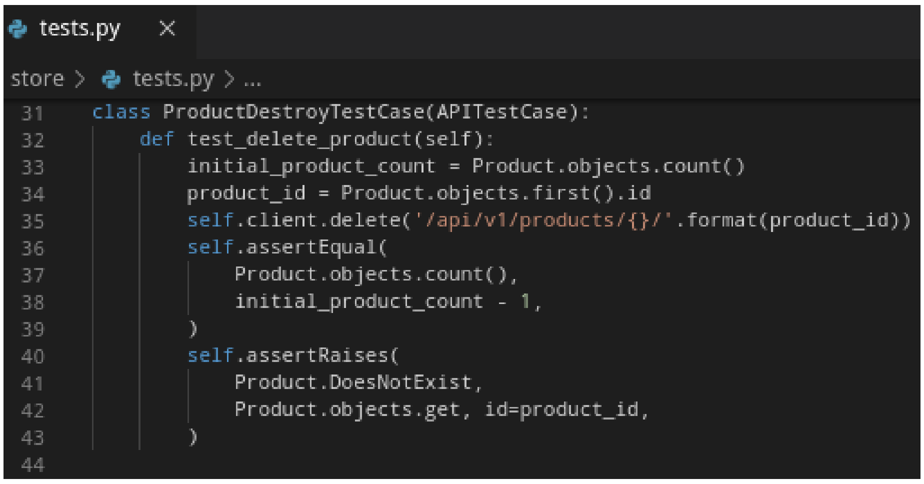
5 Appendix

5.1 Test Scripts: Unit Testing (Backend/API)



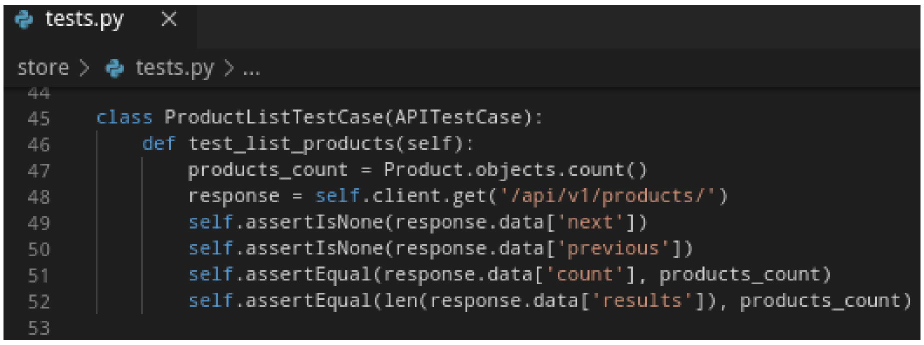
```python
import os.path
from django.conf import settings

from rest_framework.test import APITestCase

from store.models import Product

class ProductCreateTestCase(APITestCase):
    def test_create_product(self):
        initial_product_count = Product.objects.count()
        product_attrs = {
            'name': 'New Cake',
            'description': 'Awesome Cake',
            'price': '123.45',
        }
        response = self.client.post('/api/v1/products/new', product_attrs)
        if response.status_code != 201:
            print(response.data)
        self.assertEqual(
            Product.objects.count(),
            initial_product_count + 1,
        )
        for attr, expected_value in product_attrs.items():
            self.assertEqual(response.data[attr], expected_value)
        self.assertEqual(response.data['is_on_sale'], False)
        self.assertEqual(
            response.data['current_price'],
            float(product_attrs['price']),
        )
```

Figure 10: Unit Test - Product Creation API



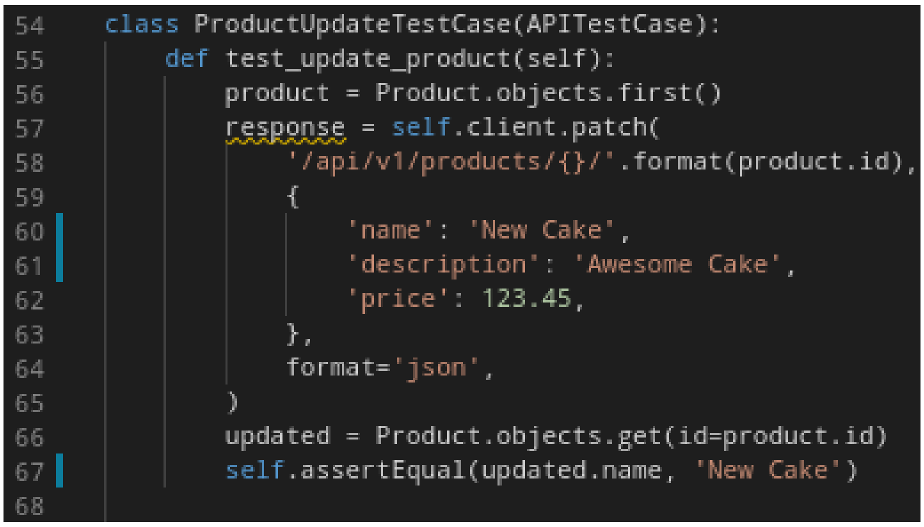
```python
class ProductDestroyTestCase(APITestCase):
    def test_delete_product(self):
        initial_product_count = Product.objects.count()
        product_id = Product.objects.first().id
        self.client.delete('/api/v1/products/{}/'.format(product_id))
        self.assertEqual(
            Product.objects.count(),
            initial_product_count - 1,
        )
        self.assertRaises(
            Product.DoesNotExist,
            Product.objects.get, id=product_id,
        )

```

Figure 11: Unit Test - Product Deletion API



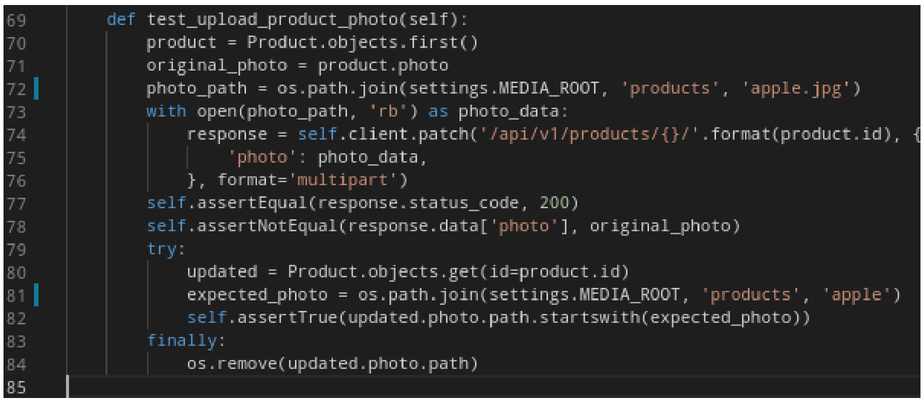
```python

class ProductListTestCase(APITestCase):
    def test_list_products(self):
        products_count = Product.objects.count()
        response = self.client.get('/api/v1/products/')
        self.assertIsNone(response.data['next'])
        self.assertIsNone(response.data['previous'])
        self.assertEqual(response.data['count'], products_count)
        self.assertEqual(len(response.data['results']), products_count)

```

Figure 12: Unit Test - Product Retrieval API



```python
class ProductUpdateTestCase(APITestCase):
    def test_update_product(self):
        product = Product.objects.first()
        response = self.client.patch(
            '/api/v1/products/{}/'.format(product.id),
            {
                'name': 'New Cake',
                'description': 'Awesome Cake',
                'price': 123.45,
            },
            format='json',
        )
        updated = Product.objects.get(id=product.id)
        self.assertEqual(updated.name, 'New Cake')

```

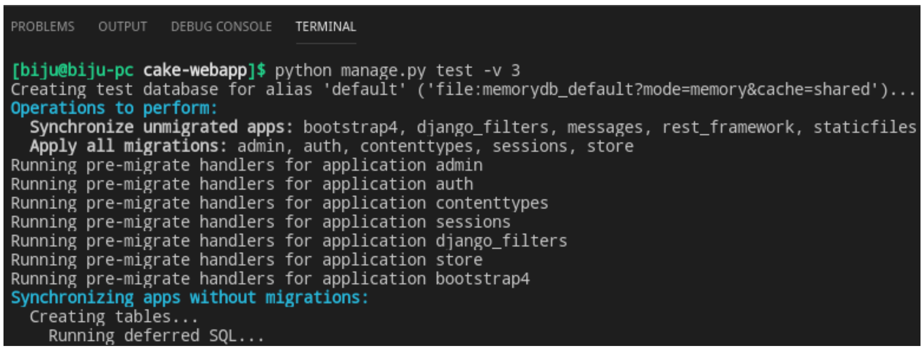
Figure 13: Unit Test - Product Updation API



```python
    def test_upload_product_photo(self):
        product = Product.objects.first()
        original_photo = product.photo
        photo_path = os.path.join(settings.MEDIA_ROOT, 'products', 'apple.jpg')
        with open(photo_path, 'rb') as photo_data:
            response = self.client.patch('/api/v1/products/{}/'.format(product.id), {
                'photo': photo_data,
            }, format='multipart')
        self.assertEqual(response.status_code, 200)
        self.assertNotEqual(response.data['photo'], original_photo)
        try:
            updated = Product.objects.get(id=product.id)
            expected_photo = os.path.join(settings.MEDIA_ROOT, 'products', 'apple')
            self.assertTrue(updated.photo.path.startswith(expected_photo))
        finally:
            os.remove(updated.photo.path)

```

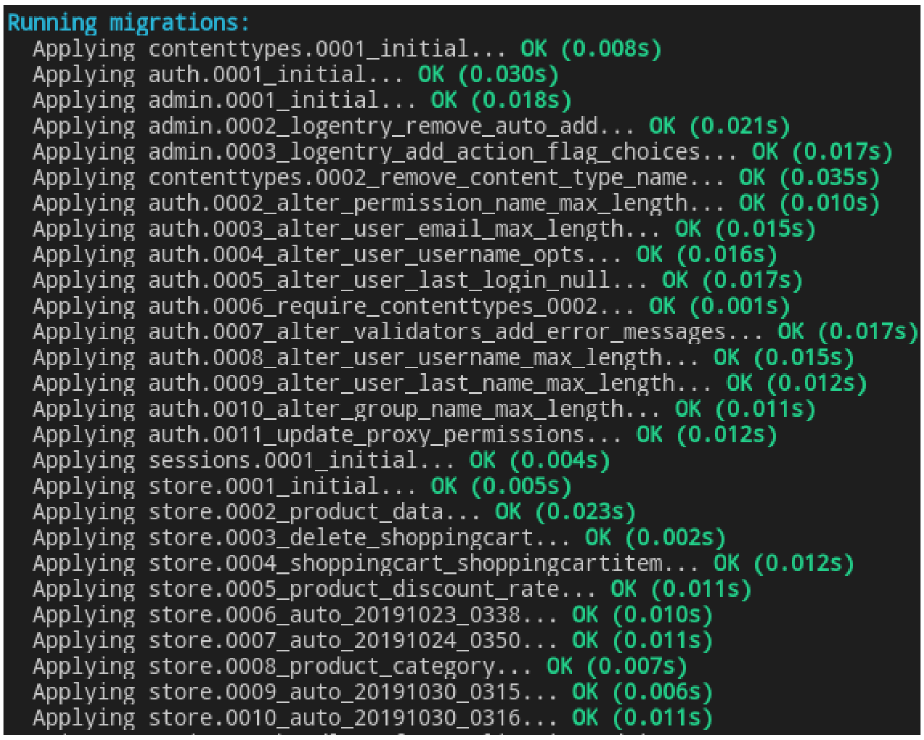
Figure 14: Unit Test - Product Image Upload API

5.2 Test Result: Unit Testing (Backend/API)


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

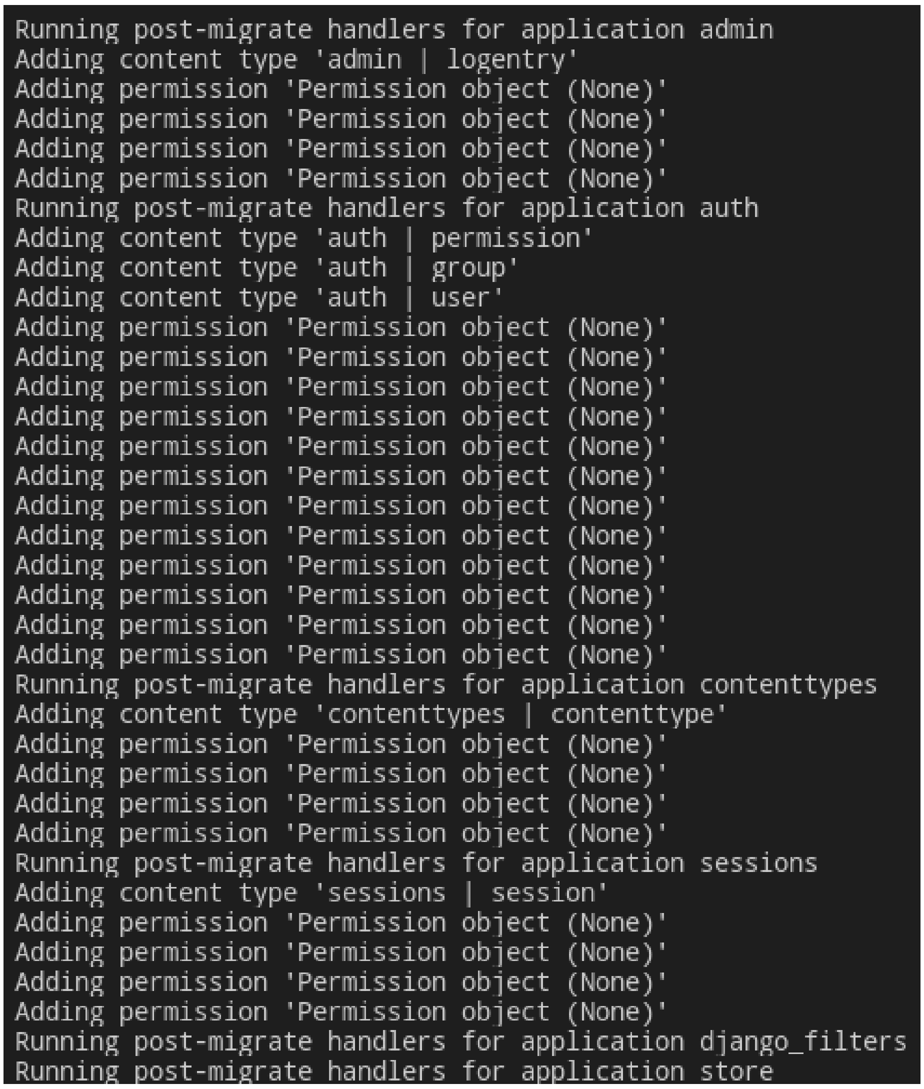
```
[biju@biju-pc cake-webapp]$ python manage.py test -v 3
Creating test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...
Operations to perform:
  Synchronize unmigrated apps: bootstrap4, django_filters, messages, rest_framework, staticfiles
  Apply all migrations: admin, auth, contenttypes, sessions, store
Running pre-migrate handlers for application admin
Running pre-migrate handlers for application auth
Running pre-migrate handlers for application contenttypes
Running pre-migrate handlers for application sessions
Running pre-migrate handlers for application django_filters
Running pre-migrate handlers for application store
Running pre-migrate handlers for application bootstrap4
Synchronizing apps without migrations:
  Creating tables...
    Running deferred SQL...
```

Figure 15: Unit Test Result 1 of 4



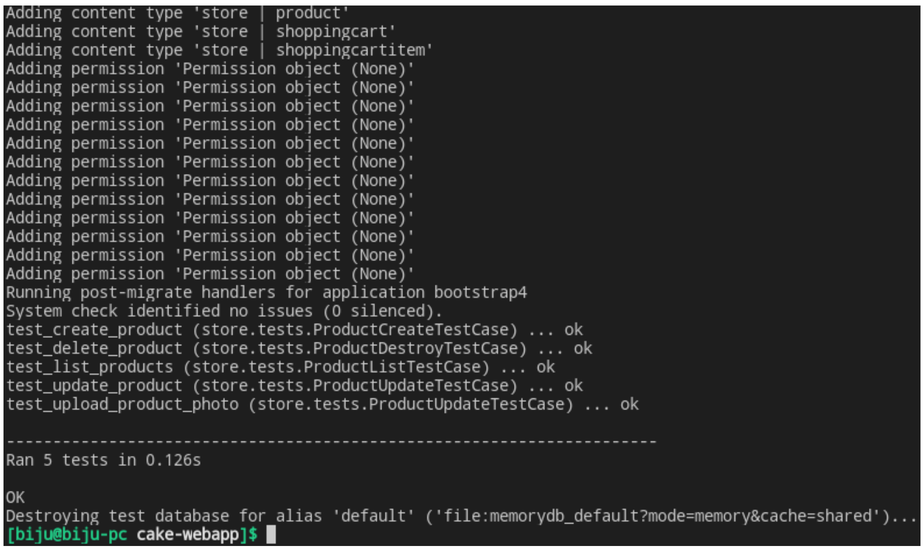
```
Running migrations:
  Applying contenttypes.0001_initial... OK (0.008s)
  Applying auth.0001_initial... OK (0.030s)
  Applying admin.0001_initial... OK (0.018s)
  Applying admin.0002_logentry_remove_auto_add... OK (0.021s)
  Applying admin.0003_logentry_add_action_flag_choices... OK (0.017s)
  Applying contenttypes.0002_remove_content_type_name... OK (0.035s)
  Applying auth.0002_alter_permission_name_max_length... OK (0.010s)
  Applying auth.0003_alter_user_email_max_length... OK (0.015s)
  Applying auth.0004_alter_user_username_opts... OK (0.016s)
  Applying auth.0005_alter_user_last_login_null... OK (0.017s)
  Applying auth.0006_require_contenttypes_0002... OK (0.001s)
  Applying auth.0007_alter_validators_add_error_messages... OK (0.017s)
  Applying auth.0008_alter_user_username_max_length... OK (0.015s)
  Applying auth.0009_alter_user_last_name_max_length... OK (0.012s)
  Applying auth.0010_alter_group_name_max_length... OK (0.011s)
  Applying auth.0011_update_proxy_permissions... OK (0.012s)
  Applying sessions.0001_initial... OK (0.004s)
  Applying store.0001_initial... OK (0.005s)
  Applying store.0002_product_data... OK (0.023s)
  Applying store.0003_delete_shoppingcart... OK (0.002s)
  Applying store.0004_shoppingcart_shoppingcartitem... OK (0.012s)
  Applying store.0005_product_discount_rate... OK (0.011s)
  Applying store.0006_auto_20191023_0338... OK (0.010s)
  Applying store.0007_auto_20191024_0350... OK (0.011s)
  Applying store.0008_product_category... OK (0.007s)
  Applying store.0009_auto_20191030_0315... OK (0.006s)
  Applying store.0010_auto_20191030_0316... OK (0.011s)
```

Figure 16: Unit Test Result 2 of 4



```
Running post-migrate handlers for application admin
Adding content type 'admin | logentry'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Running post-migrate handlers for application auth
Adding content type 'auth | permission'
Adding content type 'auth | group'
Adding content type 'auth | user'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Running post-migrate handlers for application contenttypes
Adding content type 'contenttypes | contenttype'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Running post-migrate handlers for application sessions
Adding content type 'sessions | session'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Running post-migrate handlers for application django_filters
Running post-migrate handlers for application store
```

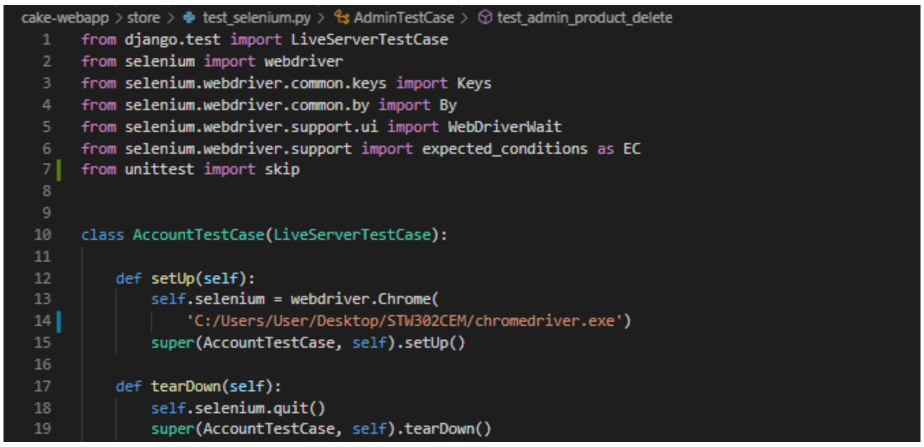
Figure 17: Unit Test Result 3 of 4



```
Adding content type 'store | product'
Adding content type 'store | shoppingcart'
Adding content type 'store | shoppingcartitem'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Adding permission 'Permission object (None)'
Running post-migrate handlers for application bootstrap4
System check identified no issues (0 silenced).
test_create_product (store.tests.ProductCreateTestCase) ... ok
test_delete_product (store.tests.ProductDestroyTestCase) ... ok
test_list_products (store.tests.ProductListTestCase) ... ok
test_update_product (store.tests.ProductUpdateTestCase) ... ok
test_upload_product_photo (store.tests.ProductUpdateTestCase) ... ok

----------------------------------------------------------------------
Ran 5 tests in 0.126s

OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...
[biju@biju-pc cake-webapp]$ 
```

Figure 18: Unit Test Result 4 of 4

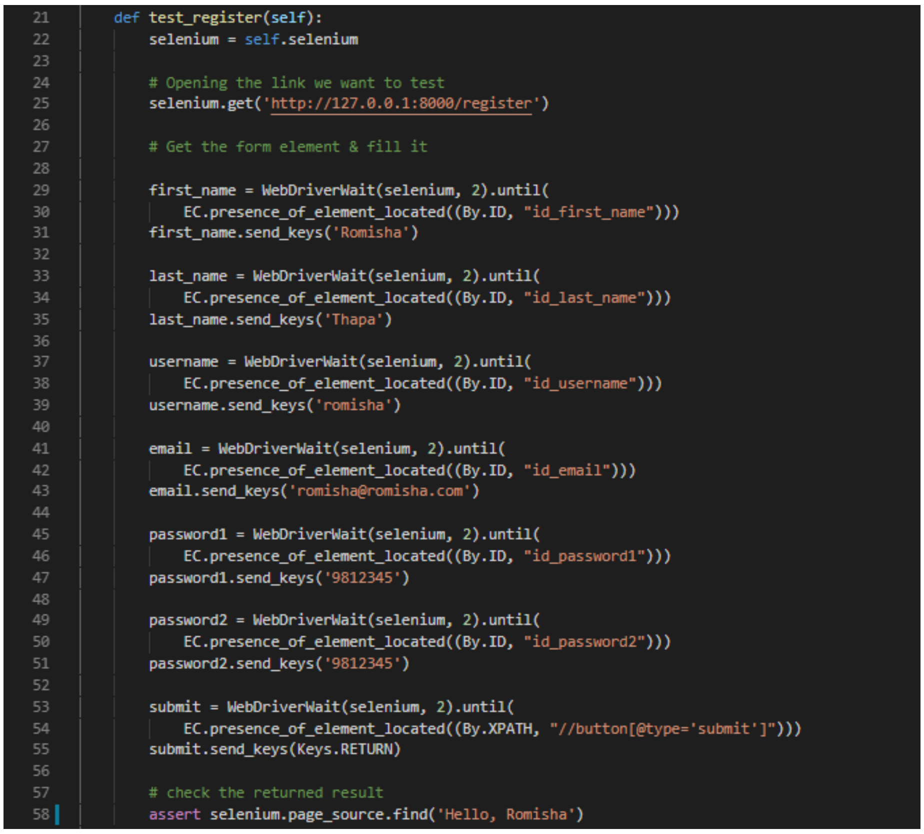
5.3 Test Scripts: Functional Testing (Frontend)



cake-webapp > store > test_selenium.py > AdminTestCase > test_admin_product_delete

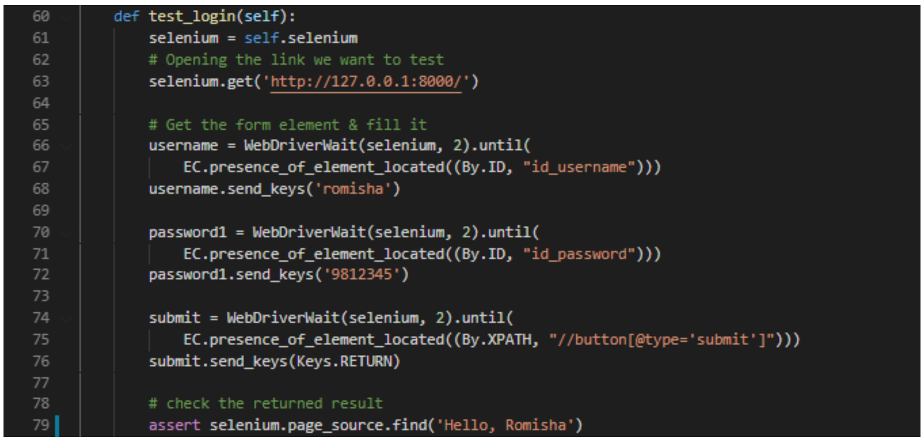
```python
from django.test import LiveServerTestCase
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from unittest import skip


class AccountTestCase(LiveServerTestCase):

    def setUp(self):
        self.selenium = webdriver.Chrome(
            'C:/Users/User/Desktop/STW302CEM/chromedriver.exe')
        super(AccountTestCase, self).setUp()

    def tearDown(self):
        self.selenium.quit()
        super(AccountTestCase, self).tearDown()
```

Figure 19: Functional Test - User Registration 1 of 2



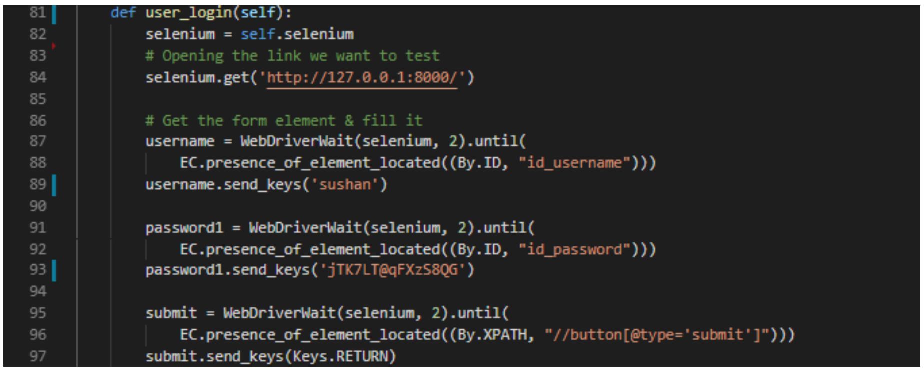
```python
    def test_register(self):
        selenium = self.selenium

        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/register')

        # Get the form element & fill it

        first_name = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_first_name")))
        first_name.send_keys('Romisha')

        last_name = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_last_name")))
        last_name.send_keys('Thapa')

        username = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_username")))
        username.send_keys('romisha')

        email = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_email")))
        email.send_keys('romisha@romisha.com')

        password1 = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_password1")))
        password1.send_keys('9812345')

        password2 = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_password2")))
        password2.send_keys('9812345')

        submit = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.XPATH, "//button[@type='submit']")))
        submit.send_keys(Keys.RETURN)

        # check the returned result
        assert selenium.page_source.find('Hello, Romisha')
```

Figure 20: Functional Test - User Registration 2 of 2



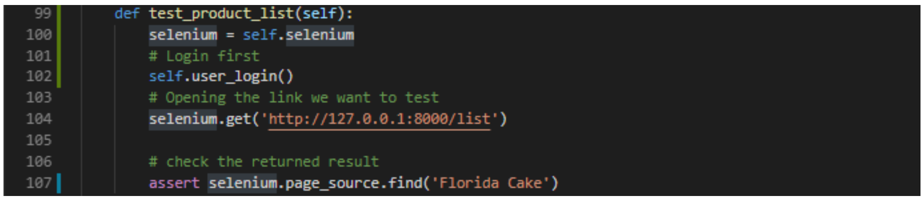
```python
    def test_login(self):
        selenium = self.selenium
        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/')

        # Get the form element & fill it
        username = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_username")))
        username.send_keys('romisha')

        password1 = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_password")))
        password1.send_keys('9812345')

        submit = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.XPATH, "//button[@type='submit']")))
        submit.send_keys(Keys.RETURN)

        # check the returned result
        assert selenium.page_source.find('Hello, Romisha')
```

Figure 21: Functional Test - User Login



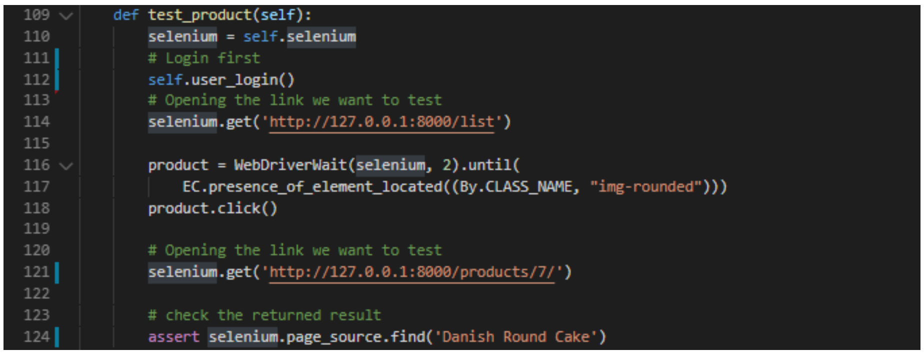
```python
    def user_login(self):
        selenium = self.selenium
        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/')

        # Get the form element & fill it
        username = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_username")))
        username.send_keys('sushan')

        password1 = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_password")))
        password1.send_keys('jTK7LT@qFXzS8QG')

        submit = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.XPATH, "//button[@type='submit']")))
        submit.send_keys(Keys.RETURN)
```

Figure 22: Functional Test - User Login (private method)



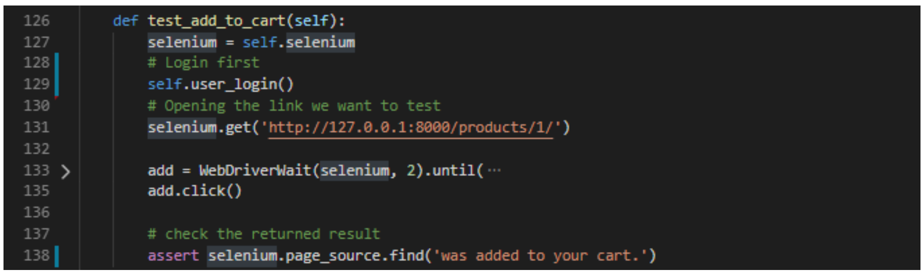
```python
    def test_product_list(self):
        selenium = self.selenium
        # Login first
        self.user_login()
        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/list')

        # check the returned result
        assert selenium.page_source.find('Florida Cake')
```

Figure 23: Functional Test - List Products



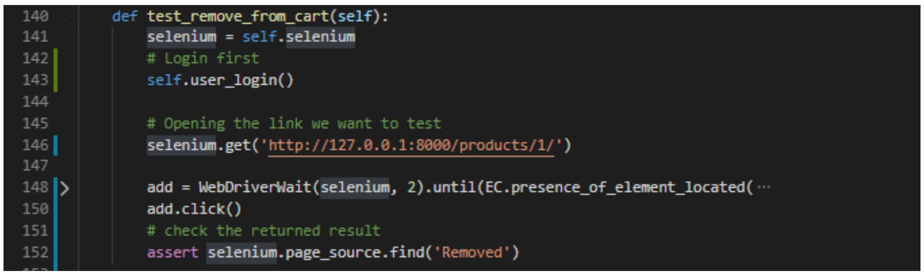
```python
    def test_product(self):
        selenium = self.selenium
        # Login first
        self.user_login()
        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/list')

        product = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.CLASS_NAME, "img-rounded")))
        product.click()

        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/products/7/')

        # check the returned result
        assert selenium.page_source.find('Danish Round Cake')
```

Figure 24: Functional Test - View Single Product



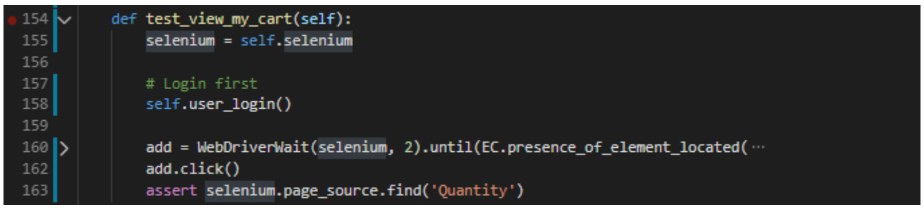
```python
    def test_add_to_cart(self):
        selenium = self.selenium
        # Login first
        self.user_login()
        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/products/1/')

        add = WebDriverWait(selenium, 2).until(···
        add.click()

        # check the returned result
        assert selenium.page_source.find('was added to your cart.')
```

Figure 25: Functional Test - Add Product to Cart



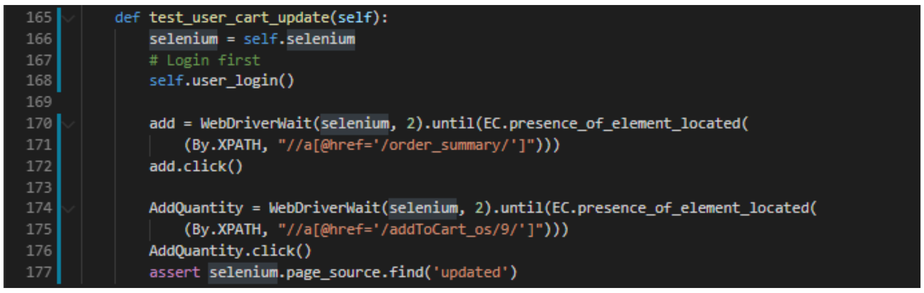
```
    def test_remove_from_cart(self):
        selenium = self.selenium
        # Login first
        self.user_login()

        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/products/1/')

        add = WebDriverWait(selenium, 2).until(EC.presence_of_element_located(···
        add.click()
        # check the returned result
        assert selenium.page_source.find('Removed')
```

Figure 26: Functional Test - Remove Product from Cart



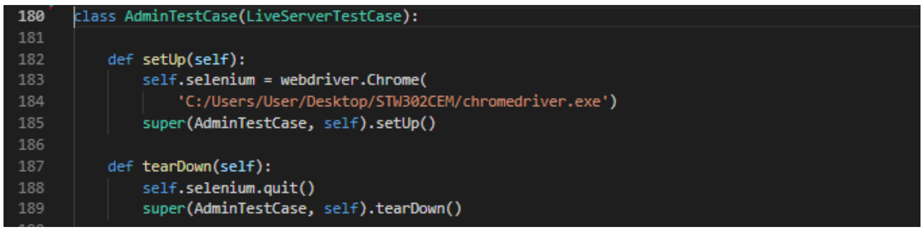
```
    def test_view_my_cart(self):
        selenium = self.selenium

        # Login first
        self.user_login()

        add = WebDriverWait(selenium, 2).until(EC.presence_of_element_located(···
        add.click()
        assert selenium.page_source.find('Quantity')
```

Figure 27: Functional Test - View Order Summary



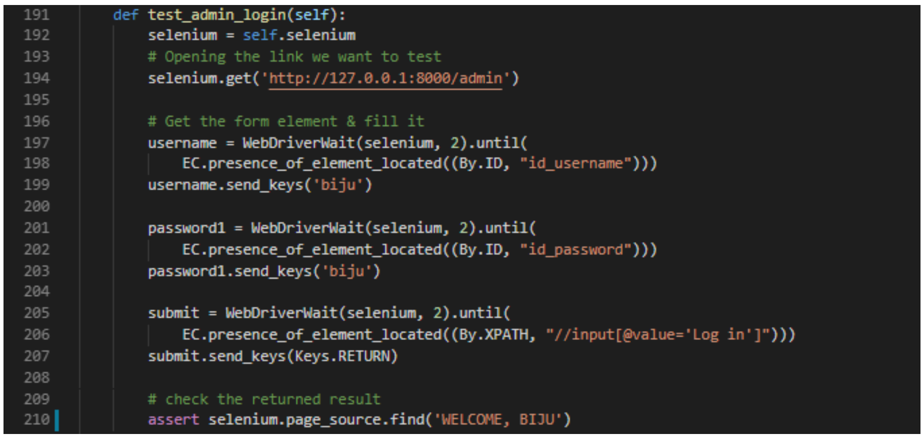
```
    def test_user_cart_update(self):
        selenium = self.selenium
        # Login first
        self.user_login()

        add = WebDriverWait(selenium, 2).until(EC.presence_of_element_located(
            (By.XPATH, "//a[@href='/order_summary/']")))
        add.click()

        AddQuantity = WebDriverWait(selenium, 2).until(EC.presence_of_element_located(
            (By.XPATH, "//a[@href='/addToCart_os/9/']")))
        AddQuantity.click()
        assert selenium.page_source.find('updated')
```

Figure 28: Functional Test - Update User Cart



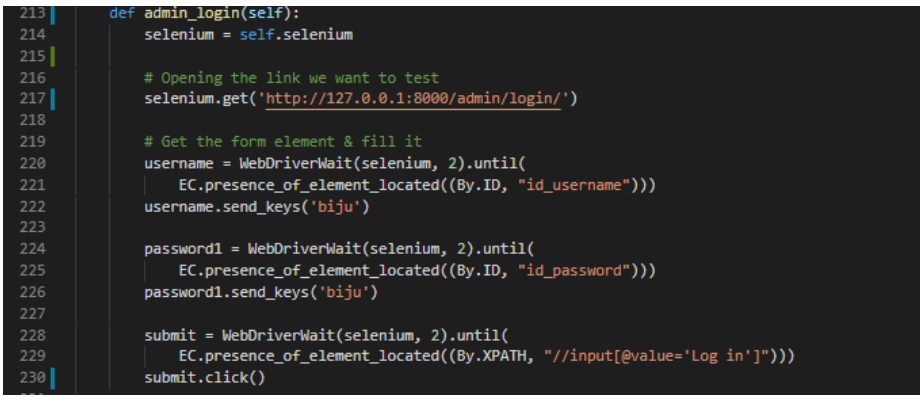
```
class AdminTestCase(LiveServerTestCase):

    def setUp(self):
        self.selenium = webdriver.Chrome(
            'C:/Users/User/Desktop/STW302CEM/chromedriver.exe')
        super(AdminTestCase, self).setUp()

    def tearDown(self):
        self.selenium.quit()
        super(AdminTestCase, self).tearDown()
```

Figure 29: Functional Test - Admin Test Case Setup



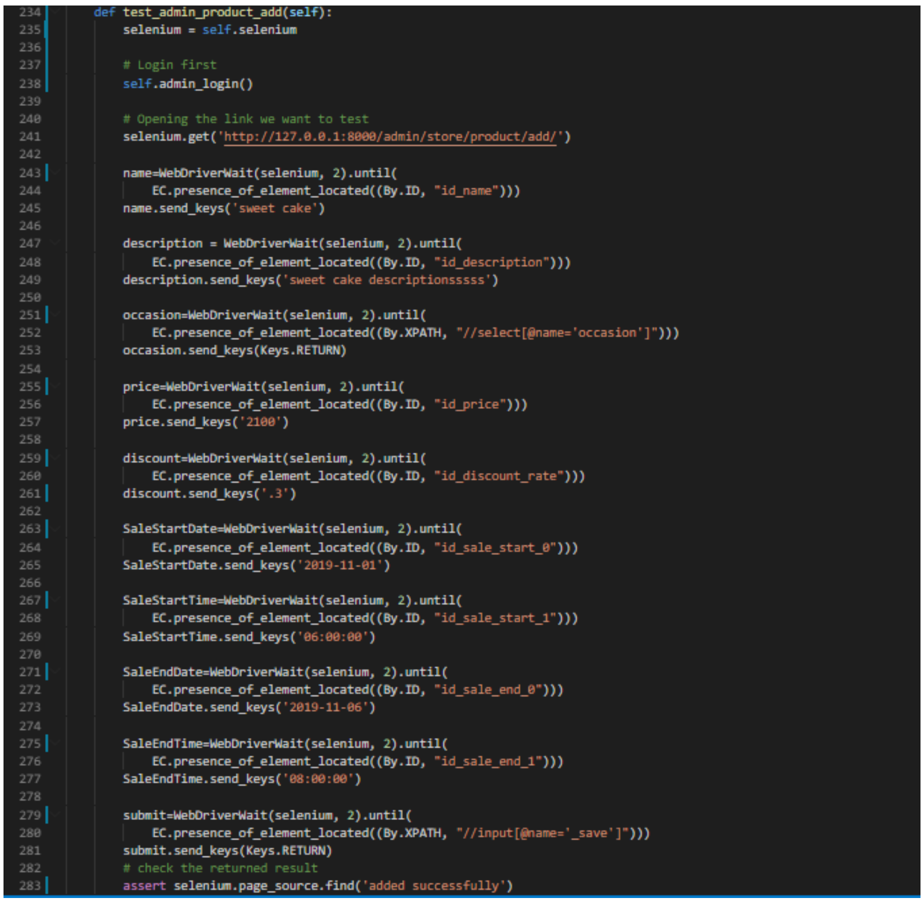
```python
    def test_admin_login(self):
        selenium = self.selenium
        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/admin')

        # Get the form element & fill it
        username = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_username")))
        username.send_keys('biju')

        password1 = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_password")))
        password1.send_keys('biju')

        submit = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.XPATH, "//input[@value='Log in']")))
        submit.send_keys(Keys.RETURN)

        # check the returned result
        assert selenium.page_source.find('WELCOME, BIJU')
```

Figure 30: Functional Test - Admin Login



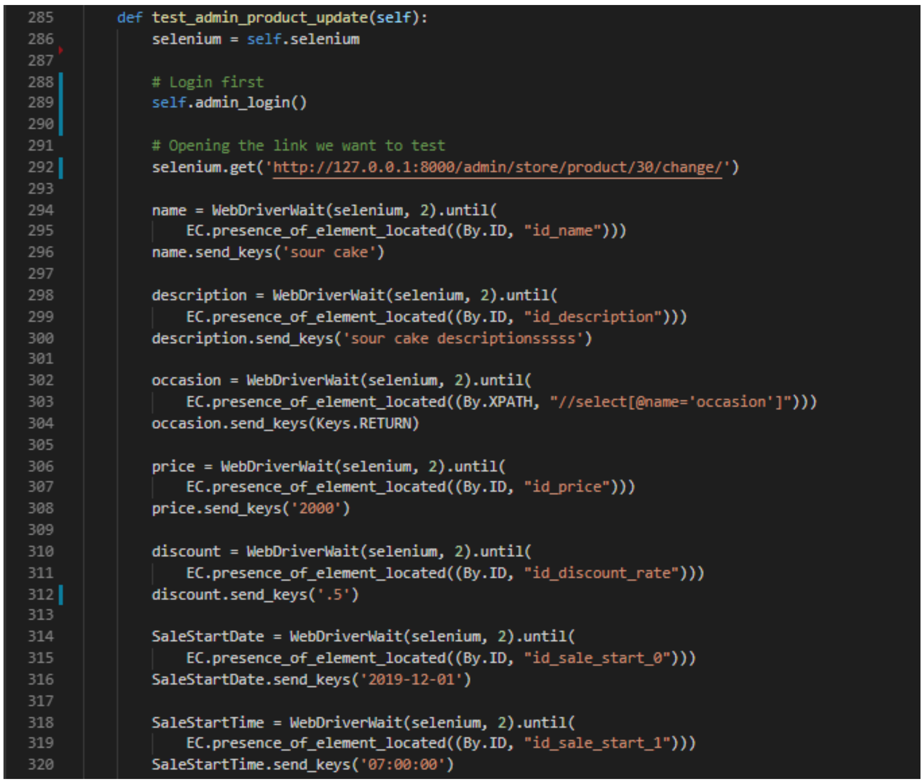
```python
    def admin_login(self):
        selenium = self.selenium

        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/admin/login/')

        # Get the form element & fill it
        username = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_username")))
        username.send_keys('biju')

        password1 = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_password")))
        password1.send_keys('biju')

        submit = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.XPATH, "//input[@value='Log in']")))
        submit.click()
```

Figure 31: Functional Test - Admin Login(Private Method)



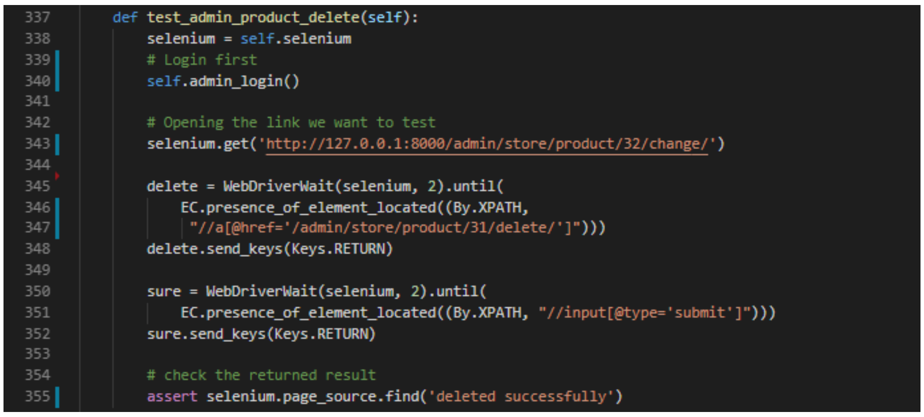
```python
    def test_admin_product_add(self):
        selenium = self.selenium

        # Login first
        self.admin_login()

        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/admin/store/product/add/')

        name=WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_name")))
        name.send_keys('sweet cake')

        description = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_description")))
        description.send_keys('sweet cake descriptionsssss')

        occasion=WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.XPATH, "//select[@name='occasion']")))
        occasion.send_keys(Keys.RETURN)

        price=WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_price")))
        price.send_keys('2100')

        discount=WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_discount_rate")))
        discount.send_keys('.3')

        SaleStartDate=WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_sale_start_0")))
        SaleStartDate.send_keys('2019-11-01')

        SaleStartTime=WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_sale_start_1")))
        SaleStartTime.send_keys('06:00:00')

        SaleEndDate=WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_sale_end_0")))
        SaleEndDate.send_keys('2019-11-06')

        SaleEndTime=WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_sale_end_1")))
        SaleEndTime.send_keys('08:00:00')

        submit=WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.XPATH, "//input[@name='_save']")))
        submit.send_keys(Keys.RETURN)
        # check the returned result
        assert selenium.page_source.find('added successfully')
```

Figure 32: Functional Test - (Admin) Add Product



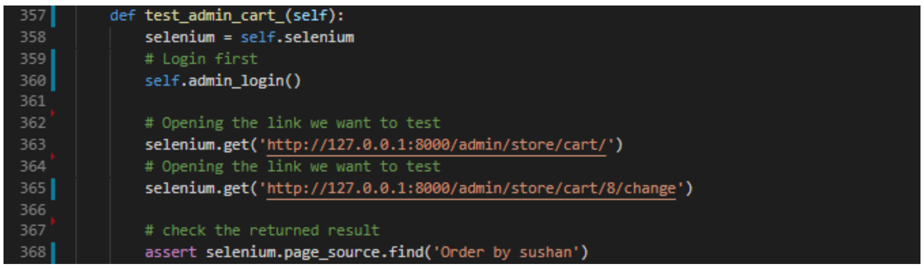
```python
    def test_admin_product_update(self):
        selenium = self.selenium

        # Login first
        self.admin_login()

        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/admin/store/product/30/change/')

        name = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_name")))
        name.send_keys('sour cake')

        description = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_description")))
        description.send_keys('sour cake descriptionsssss')

        occasion = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.XPATH, "//select[@name='occasion']")))
        occasion.send_keys(Keys.RETURN)

        price = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_price")))
        price.send_keys('2000')

        discount = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_discount_rate")))
        discount.send_keys('.5')

        SaleStartDate = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_sale_start_0")))
        SaleStartDate.send_keys('2019-12-01')

        SaleStartTime = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_sale_start_1")))
        SaleStartTime.send_keys('07:00:00')
```

Figure 33: Functional Test - (Admin) Update Product



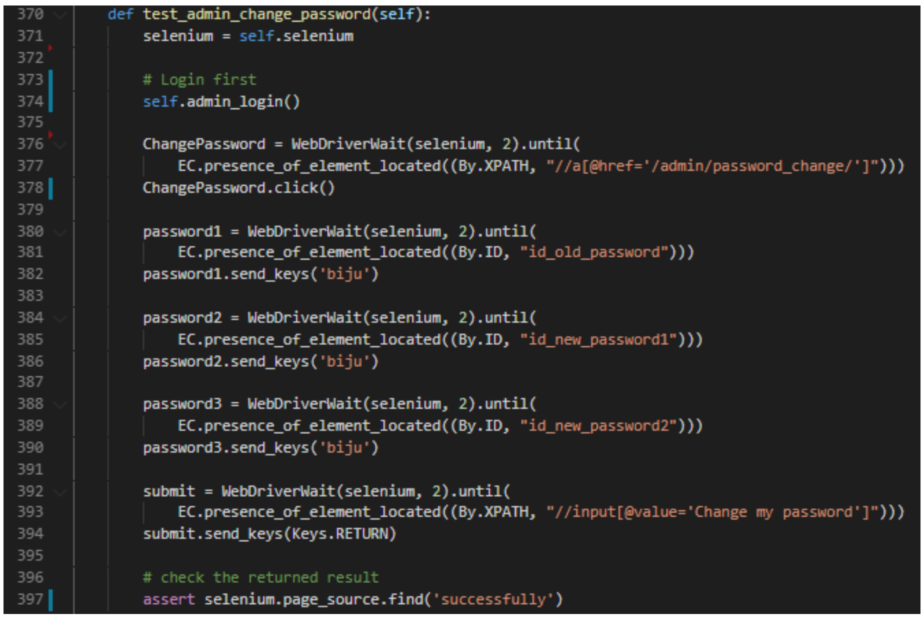
```python
    def test_admin_product_delete(self):
        selenium = self.selenium
        # Login first
        self.admin_login()

        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/admin/store/product/32/change/')

        delete = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.XPATH,
             "//a[@href='/admin/store/product/31/delete/']")))
        delete.send_keys(Keys.RETURN)

        sure = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.XPATH, "//input[@type='submit']")))
        sure.send_keys(Keys.RETURN)

        # check the returned result
        assert selenium.page_source.find('deleted successfully')
```

Figure 34: Functional Test - (Admin) Delete Product



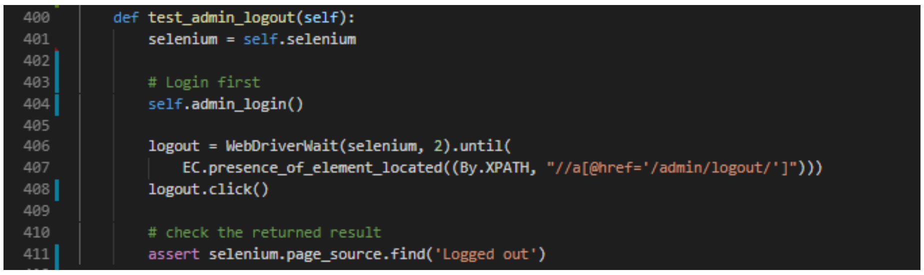
```python
    def test_admin_cart_(self):
        selenium = self.selenium
        # Login first
        self.admin_login()

        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/admin/store/cart/')
        # Opening the link we want to test
        selenium.get('http://127.0.0.1:8000/admin/store/cart/8/change')

        # check the returned result
        assert selenium.page_source.find('Order by sushan')
```

Figure 35: Functional Test - (Admin) View Orders



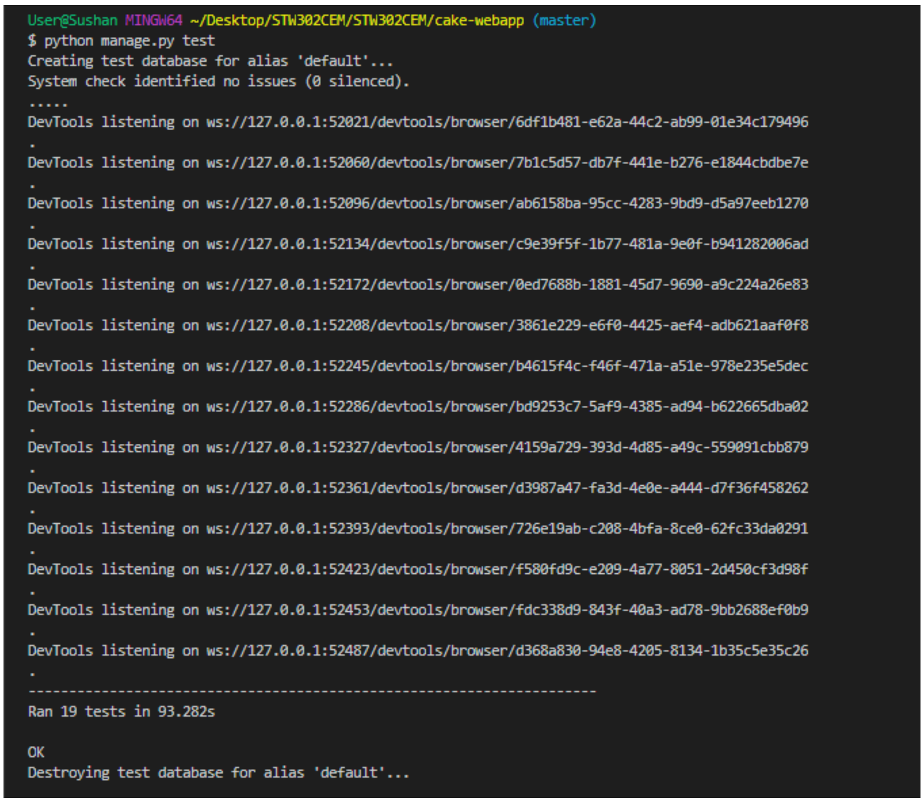
```python
    def test_admin_change_password(self):
        selenium = self.selenium

        # Login first
        self.admin_login()

        ChangePassword = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.XPATH, "//a[@href='/admin/password_change/']")))
        ChangePassword.click()

        password1 = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_old_password")))
        password1.send_keys('biju')

        password2 = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_new_password1")))
        password2.send_keys('biju')

        password3 = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.ID, "id_new_password2")))
        password3.send_keys('biju')

        submit = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.XPATH, "//input[@value='Change my password']")))
        submit.send_keys(Keys.RETURN)

        # check the returned result
        assert selenium.page_source.find('successfully')
```

Figure 36: Functional Test - (Admin) Change Password



```python
    def test_admin_logout(self):
        selenium = self.selenium

        # Login first
        self.admin_login()

        logout = WebDriverWait(selenium, 2).until(
            EC.presence_of_element_located((By.XPATH, "//a[@href='/admin/logout/']")))
        logout.click()

        # check the returned result
        assert selenium.page_source.find('Logged out')
```

Figure 37: Functional Test - (Admin) Logout



```
User@Sushan MINGW64 ~/Desktop/STW302CEM/STW302CEM/cake-webapp (master)
$ python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
DevTools listening on ws://127.0.0.1:52021/devtools/browser/6df1b481-e62a-44c2-ab99-01e34c179496
.
DevTools listening on ws://127.0.0.1:52060/devtools/browser/7b1c5d57-db7f-441e-b276-e1844cbdbe7e
.
DevTools listening on ws://127.0.0.1:52096/devtools/browser/ab6158ba-95cc-4283-9bd9-d5a97eeb1270
.
DevTools listening on ws://127.0.0.1:52134/devtools/browser/c9e39f5f-1b77-481a-9e0f-b941282006ad
.
DevTools listening on ws://127.0.0.1:52172/devtools/browser/0ed7688b-1881-45d7-9690-a9c224a26e83
.
DevTools listening on ws://127.0.0.1:52208/devtools/browser/3861e229-e6f0-4425-aef4-adb621aaf0f8
.
DevTools listening on ws://127.0.0.1:52245/devtools/browser/b4615f4c-f46f-471a-a51e-978e235e5dec
.
DevTools listening on ws://127.0.0.1:52286/devtools/browser/bd9253c7-5af9-4385-ad94-b622665dba02
.
DevTools listening on ws://127.0.0.1:52327/devtools/browser/4159a729-393d-4d85-a49c-559091cbb879
.
DevTools listening on ws://127.0.0.1:52361/devtools/browser/d3987a47-fa3d-4e0e-a444-d7f36f458262
.
DevTools listening on ws://127.0.0.1:52393/devtools/browser/726e19ab-c208-4bfa-8ce0-62fc33da0291
.
DevTools listening on ws://127.0.0.1:52423/devtools/browser/f580fd9c-e209-4a77-8051-2d450cf3d98f
.
DevTools listening on ws://127.0.0.1:52453/devtools/browser/fdc338d9-843f-40a3-ad78-9bb2688ef0b9
.
DevTools listening on ws://127.0.0.1:52487/devtools/browser/d368a830-94e8-4205-8134-1b35c5e35c26
.
----------------------------------------------------------------------
Ran 19 tests in 93.282s

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

Figure 38: Functional Test Results (Selenium)