6410685165 Thanadon Boontawee
6410685215 Patcharapon Tappakwan

course/test.py

```python
from django.test import TestCase
from django.test import TestCase, Client
from django.urls import reverse
from django.contrib.auth.models import User
from .models import Course, Enroll
from users.models import Student
from django.contrib.auth.hashers import make_password
from django.contrib.auth import authenticate
# Create your tests here.
class CourseTestCase(TestCase):
	def setUp(self):
		# create users
		user1 = User.objects.create_user(username="user1",password="password1")
		user2 = User.objects.create_user(username="user2",password="password2",
			is_staff=True)

		student1 = Student.objects.create(ID=user1, fname="first1",
			lname="last1", email="user@mail.com")
		course1 = Course.objects.create(ID="CN331", name="Software Engineering",
			quota=True, enrolled=0)
		course2 = Course.objects.create(ID="CN360", name="Microcontroller",
			quota=True, enrolled=0)
		course_neg = Course.objects.create(ID="CN321", name="Network", quota= -1,
			enrolled=0)

		course2.quota -= 1
		course2.enrolled += 1
		course_enroll = Enroll.objects.create(student_id=user1,
			course_id=course2)

		course_enroll.save()
		course_neg.save()
		course1.save()
		course2.save()
		user1.save()
		user2.save()
		student1.save()

	''' setUp(self) function is used for initial test data in the database.
	It creates users, students, and courses for testing the application. '''
```

6410685165 Thanadon Boontawee
6410685215 Patcharapon Tappakwan

```
    def test_model_course_str(self):
        course = Course.objects.get(ID="CN331")
        self.assertEqual(str(course), "CN331 Software Engineering - 1")
    ''' test_model_course_str(self) function is used for test the ___str___ method
    in course model. It creates users, students, and courses for testing the
    application. '''

    def test_index(self):
        c = Client()
        c.login(username='user1', password="password1")
        response = c.get(reverse('course'))
        self.assertEqual(response.status_code, 200)
    ''' test_index(self) function is used for test accessing the main index page.
    It verifies if the main index page can be accessed. '''

    def test_page_course(self):
        c = Client()
        c.login(username="user1", password="password1")
        response = c.get(reverse('page_course'))
        self.assertEqual(response.status_code, 200)
    ''' test_page_course(self) function is used to test the course page.
    It verifies the course page can be accessed. '''

    def test_page_user(self):
        c = Client()
        c.login(username="user1", password="password1")
        response = c.get(reverse('page_user'))
        self.assertEqual(response.status_code, 200)
    ''' test_page_user(self) function is used to test the user page.
    It verifies the user page can be accessed. '''

    def test_page_user_staff(self):
        c = Client()
        c.login(username="user2", password="password2")
        response = c.get(reverse('page_user'))
        self.assertEqual(response.status_code, 200)
    ''' test_page_user_staff(self) function is used to test the user page by a
    staff user. It verifies if a staff user can access the user page. '''

    def test_page_board(self):
        c = Client()
        c.login(username="user1", password="password1")
```

6410685165 Thanadon Boontawee
6410685215 Patcharapon Tappakwan

```
        response = c.get(reverse('page_board'))
        self.assertEqual(response.status_code, 200)
    ''' test_page_board(self) function is used to test accessing the board page.
    It verifies if the board page can be accessed. '''

    def test_course_enroll_admin(self):
        c = Client()
        c.login(username="user2", password="password2")
        response = c.get(reverse('course_enroll'))
        self.assertEqual(response.status_code, 200)
    ''' test_course_enroll_admin(self) function is used for test accessing the
    course enrollment page by admin user. It verifies if an admin user can access
    The course enrollment page. '''

    def test_course_enroll_quota_less_zero(self):
        c = Client()
        c.login(username="user1", password="password1")
        response = c.post(reverse('course_enroll'), {"course_id" : "CN321"})
        self.assertEqual(response.status_code, 200)
    ''' test_course_enroll_quota_less_zero(self) function is used for test course
    With a quota less than zero. It checks if enrolling in a course with a negative
    quota is handled correctly. '''

    def test_course_enroll_success(self):
        c = Client()
        c.login(username="user1", password="password1")
        response = c.post(reverse('course_enroll'), {"course_id" : "CN331"})
        self.assertEqual(response.status_code, 200)
    ''' test_course_enroll_success(self) function is used for test successful
    course enrollment. It verifies if a user can successfully enroll in a course.'''

    def test_course_drop_admin(self):
        c = Client()
        c.login(username="user2", password="password2")
        user1 = User.objects.get(username="user1")
        cn360 = Course.objects.get(ID="CN360")

        self.assertEqual(cn360.enrolled, 1)
        response = c.post(reverse('course_drop'), {"user_id" : "user1",
        "course_id" : "CN360"})
        self.assertEqual(response.status_code, 200)
    ''' test_course_drop_admin(self) function is used for test dropping a course
```

6410685165 Thanadon Boontawee
6410685215 Patcharapon Tappakwan

```
        by an admin user. It checks if an admin user can drop a user from a course.'''

        def test_course_drop_user(self):
                c = Client()
                c.login(username="user1", password="password1")
                user1 = User.objects.get(username="user1")
                cn360 = Course.objects.get(ID="CN360")
                self.assertEqual(cn360.enrolled, 1)
                response = c.post(reverse('course_drop'), {"user_id" : "user1",
                "course_id" : "CN360"})
                self.assertEqual(response.status_code, 200)
        ''' test_course_enroll_admid(self) function is used for test dropping a course
        by a regular user. It verifies if an admin user can drop themselves from
        a course. '''

        def test_manager_get(self):
                c = Client()
                c.login(username="user2", password="password2")
                response = c.get(reverse('manager'))
                self.assertEqual(response.status_code, 200)
        ''' test_manager_get(self) function is used for test accessing the manager page
        by a staff user. It verifies if a staff user can access the manager page. '''


class CourseTestCase_Zero(TestCase):
        def setUp(self):
                # create users
                user1 = User.objects.create_user(username="user1",password="password1")
                user2 = User.objects.create_user(username="user2",password="password2",
                        is_staff=True)

                student1 = Student.objects.create(ID=user1, fname="first1",
                        lname="last1", email="user@mail.com")
                user1.save()
                user2.save()
                student1.save()
        ''' setUp(self) function is used for initial test data with zero quota.
        It checks how the application handles courses with a zero quota on the manager
        page. '''

        def test_manager_dropdown_zero(self):
                c = Client()
```

6410685165 Thanadon Boontawee
6410685215 Patcharapon Tappakwan

```
            c.login(username="user2", password="password2")
            response = c.post(reverse('manager'))
            self.assertEqual(response.status_code, 200)
      ''' test_manager_dropdown_zero(self) function is used for test handling zero
      quota on the manager page. It checks how the application handles courses with
      a zero quota on the manager page. '''
```

users/test.py

```
from django.test import TestCase, Client
from django.urls import reverse
from django.contrib.auth.models import User
from .models import Student

# Create your tests here.
class UserTestCase(TestCase):
      def setUp(self):
            # create users
            user1 = User.objects.create_user(username="user1",password="password1")
            user2 = User.objects.create_user(username="user2",password="password2",
                  is_staff=True)
            student1 = Student.objects.create(ID=user1, fname="first1",
                  lname="last1", email="user@mail.com")
            self.staff_user = User.objects.create_user(
                  username='admin',
                  password='admin1234',
                  is_staff=True
            )
            user1.save()
            user2.save()
            student1.save()
      ''' setUp(self) function is used for setting up initial for the test. It
      creates users and a student object in the test database, which can be used by
      other test functions. It tests the setup process and ensures that the required
      users and student objects are created successfully. '''

      def test_models_student_tostr(self):
            student1 = Student.objects.get(fname="first1")
            self.assertEqual(str(student1),"user1 | user@mail.com")
      ''' test_models_student_tostr(self) function is used for test the ___str___
      method of the Student model. It checks if the ___str___ method of the Student
      model returns the expected string representation. '''
```

6410685165 Thanadon Boontawee
6410685215 Patcharapon Tappakwan

```
    def test_annoymous(self):
        c = Client()
        response = c.get(reverse('login'))
        self.assertEqual(response.status_code, 200)
    ''' test_annoymous(self) function is used for test if an anonymous user can
    access the login page. It verifies that the login page is accessible to users
    Who are not logged in. '''

    def test_login_already_user(self):
        c = Client()
        c.login(username="user1", password="password1")
        response = c.get(reverse('login'))
        self.assertEqual(response.status_code, 200)
    ''' test_login_already_user(self) function is used for test if a user who is
    already logged in can access the login page. It ensures that users who are
    already logged in are still able to access the login page. '''

    def test_login_already_staff(self):
        c = Client()
        c.login(username="user2", password="password2")
        response = c.get(reverse('login'))
        self.assertEqual(response.status_code, 200)
    ''' test_login_already_staff(self) function is used for test if a staff user
    who is  already logged in can access the login page. It verifies that staff
    users, even when logged in, can still access the login page. '''

    def test_login_get(self):
        c = Client()
        response = c.get("")
        self.assertEqual(response.status_code, 200)
    ''' test_login_get(self) function is used for test the HTTP GET request to the
    login page. It verifies that the login page can be accessed via a GET request. '''

    def test_login_as_user(self):
        c = Client()
        form = {'username': "user1", 'password' : "password1"}
        response = c.post(reverse('login'), form)
        self.assertEqual(response.status_code, 200)
    ''' test_login_as_user(self) function is used for test the user login process
    with correct credentials. It checks if a user can log in successfully with the
    correct username and password. '''
```

6410685165 Thanadon Boontawee
6410685215 Patcharapon Tappakwan

```
    def test_login_as_staff(self):
        c = Client()
        form = {'username': "admin", 'password' : "admin1234"}
        response = c.post(reverse('login'), form)
        self.assertTrue(response.context['user'].is_authenticated)
        self.assertTrue(response.context['user'].is_staff)
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'course/page_user.html')
        self.assertTrue(response.context['admin'])
    ''' test_login_as_staff(self) function is used for test the staff user login
    process with correct credentials. It verifies if a staff user can log in successfully
    And is redirected to the correct page. '''

    def test_is_authenticated_wrong_pass(self):
        c = Client()
        form = {'username': "user1", 'password': "wrongpassword"}
        response = c.post(reverse('login'), form)
        self.assertEqual(response.status_code, 200)
    ''' test_is_authentidated_wrong_pass(self) function is used for test the login
    process with incorrect password. It checks if a user with an incorrect password is
    not able to login. '''

    def test_is_authenticated_form_invalid(self):
        c = Client()
        form = {'username': "user1"}
        response = c.post(reverse('login'), form)
        self.assertEqual(response.status_code, 200)
    ''' test_is_authentidated_from_invalid(self) function is used for test the login
    process with missing data. It ensures that the login form validation works
    Correctly when some form data is missing. '''

    def test_logout(self):
        c = Client()
        c.login(username="user1", password="password1")
        response = c.post(reverse('logout'))
        self.assertEqual(response.status_code, 302)
    ''' test_logout(self) function is used to test the user logout process. It checks
    if a logged-in user can successfully log out and is redirected to the expected
    page. '''
```