## 测试用例

### 用户注册测试

这个测试用例验证用户注册功能是否正常工作。它发送一个 POST 请求到注册 URL，并检查响应状态码是否为 302（重定向），以及用户是否成功创建。

python

import json

from django.test import TestCase

from django.urls import reverse

from .models import User

class UserRegistrationTest(TestCase):

def test\_register\_user(self):

发送 POST 请求到注册 URL

response = self.client.post(reverse('register'), {

'email': 'test@example.com',

'password': 'password123'

})

检查响应状态码是否为 302（重定向）

self.assertEqual(response.status\_code, 302)

检查用户是否成功创建

self.assertTrue(User.objects.filter(email='test@example.com').exists())

在这个测试用例中，我们首先导入了必要的模块和类。然后，我们定义了一个 UserRegistrationTest 类，该类继承自 TestCase。在这个类中，我们定义了一个 test\_register\_user 方法，该方法用于测试用户注册功能。

在 test\_register\_user 方法中，我们使用 self.client.post 方法发送一个 POST 请求到注册 URL。请求的 URL 是通过 reverse 函数生成的，参数是 URL 的名称 'register'。请求的内容是一个包含用户邮箱和密码的字典。

发送请求后，我们使用 self.assertEqual 方法检查响应的状态码是否为 302（重定向）。接着，我们使用 self.assertTrue 方法检查用户是否成功创建。具体来说，我们检查 User 模型中是否存在邮箱为 'test@example.com' 的用户。

### 用户登录测试

这个测试用例验证用户登录功能是否正常工作。它首先创建一个用户，然后发送一个 POST 请求到登录 URL，并检查响应状态码是否为 302（重定向），以及用户是否成功登录。

python

class UserLoginTest(TestCase):

def setUp(self):

创建一个用户

self.user = User.objects.create\_user(email='test@example.com', password='password123')

def test\_login\_user(self):

发送 POST 请求到登录 URL

response = self.client.post(reverse('login'), {

'email': 'test@example.com',

'password': 'password123'

})

检查响应状态码是否为 302（重定向）

self.assertEqual(response.status\_code, 302)

检查用户是否成功登录

self.assertTrue(self.client.session['is\_logged\_in'])

在这个测试用例中，我们定义了一个 UserLoginTest 类，该类继承自 TestCase。在这个类中，我们定义了两个方法：setUp 和 test\_login\_user。

在 setUp 方法中，我们使用 User.objects.create\_user 方法创建一个用户。这个方法接受两个参数：用户的邮箱和密码。

在 test\_login\_user 方法中，我们使用 self.client.post 方法发送一个 POST 请求到登录 URL。请求的 URL 是通过 reverse 函数生成的，参数是 URL 的名称 'login'。请求的内容是一个包含用户邮箱和密码的字典。

发送请求后，我们使用 self.assertEqual 方法检查响应的状态码是否为 302（重定向）。接着，我们使用 self.assertTrue 方法检查用户是否成功登录。具体来说，我们检查 self.client.session 中是否存在键 'is\_logged\_in'。

### 药品添加测试

这个测试用例验证药品添加功能是否正常工作。它首先创建并登录一个用户，然后发送一个 POST 请求到添加药品的 URL，并检查响应状态码是否为 302（重定向），以及药品是否成功添加。

python

from .models import Drug

class AddDrugTest(TestCase):

def setUp(self):

创建并登录一个用户

self.user = User.objects.create\_user(email='test@example.com', password='password123')

self.client.login(email='test@example.com', password='password123')

def test\_add\_drug(self):

发送 POST 请求到添加药品的 URL

response = self.client.post(reverse('add\_drug'), {

'name': 'Aspirin',

'company': 'Pharma Inc.',

'quantity': 100,

'code': 'ASP100'

})

检查响应状态码是否为 302（重定向）

self.assertEqual(response.status\_code, 302)

检查药品是否成功添加

self.assertTrue(Drug.objects.filter(drug\_name='Aspirin').exists())

在这个测试用例中，我们定义了一个 AddDrugTest 类，该类继承自 TestCase。在这个类中，我们定义了两个方法：setUp 和 test\_add\_drug。

在 setUp 方法中，我们使用 User.objects.create\_user 方法创建一个用户，并使用 self.client.login 方法登录该用户。create\_user 方法接受两个参数：用户的邮箱和密码。login 方法也接受两个参数：用户的邮箱和密码。

在 test\_add\_drug 方法中，我们使用 self.client.post 方法发送一个 POST 请求到添加药品的 URL。请求的 URL 是通过 reverse 函数生成的，参数是 URL 的名称 'add\_drug'。请求的内容是一个包含药品名称、公司、数量和代码的字典。

发送请求后，我们使用 self.assertEqual 方法检查响应的状态码是否为 302（重定向）。接着，我们使用 self.assertTrue 方法检查药品是否成功添加。具体来说，我们检查 Drug 模型中是否存在药品名称为 'Aspirin' 的药品。

### 客户添加测试

这个测试用例验证客户添加功能是否正常工作。它发送一个 POST 请求到分发器 URL，并检查响应状态码是否为 200，以及客户是否成功添加。

python

from django.urls import reverse

from .models import Customer

class AddCustomerTest(TestCase):

def test\_add\_customer(self):

发送 POST 请求到分发器 URL

response = self.client.post(reverse('dispatcher'), json.dumps({

'action': 'add\_customer',

'data': {

'name': 'John Doe',

'phonenumber': '1234567890',

'address': '123 Main St'

}

}), content\_type='application/json')

检查响应状态码是否为 200

self.assertEqual(response.status\_code, 200)

检查客户是否成功添加

self.assertTrue(Customer.objects.filter(name='John Doe').exists())

在这个测试用例中，我们定义了一个 AddCustomerTest 类，该类继承自 TestCase。在这个类中，我们定义了一个 test\_add\_customer 方法，该方法用于测试客户添加功能。

在 test\_add\_customer 方法中，我们使用 self.client.post 方法发送一个 POST 请求到分发器 URL。请求的 URL 是通过 reverse 函数生成的，参数是 URL 的名称 'dispatcher'。请求的内容是一个包含操作类型和客户数据的 JSON 字符串。我们使用 json.dumps 方法将字典转换为 JSON 字符串，并将 content\_type 设置为 'application/json'。

发送请求后，我们使用 self.assertEqual 方法检查响应的状态码是否为 200。接着，我们使用 self.assertTrue 方法检查客户是否成功添加。具体来说，我们检查 Customer 模型中是否存在名称为 'John Doe' 的客户。

### 用户注册测试

在用户注册测试中，我们首先导入了必要的模块和类。然后，我们定义了一个 UserRegistrationTest 类，该类继承自 TestCase。在这个类中，我们定义了一个 test\_register\_user 方法，该方法用于测试用户注册功能。

在 test\_register\_user 方法中，我们使用 self.client.post 方法发送一个 POST 请求到注册 URL。请求的 URL 是通过 reverse 函数生成的，参数是 URL 的名称 'register'。请求的内容是一个包含用户邮箱和密码的字典。

发送请求后，我们使用 self.assertEqual 方法检查响应的状态码是否为 302（重定向）。接着，我们使用 self.assertTrue 方法检查用户是否成功创建。具体来说，我们检查 User 模型中是否存在邮箱为 'test@example.com' 的用户。

用户登录测试

在用户登录测试中，我们定义了一个 UserLoginTest 类，该类继承自 TestCase。在这个类中，我们定义了两个方法：setUp 和 test\_login\_user。

在 setUp 方法中，我们使用 User.objects.create\_user 方法创建一个用户。这个方法接受两个参数：用户的邮箱和密码。

在 test\_login\_user 方法中，我们使用 self.client.post 方法发送一个 POST 请求到登录 URL。请求的 URL 是通过 reverse 函数生成的，参数是 URL 的名称 'login'。请求的内容是一个包含用户邮箱和密码的字典。

发送请求后，我们使用 self.assertEqual 方法检查响应的状态码是否为 302（重定向）。接着，我们使用 self.assertTrue 方法检查用户是否成功登录。具体来说，我们检查 self.client.session 中是否存在键 'is\_logged\_in'。

### 药品添加测试

在药品添加测试中，我们定义了一个 AddDrugTest 类，该类继承自 TestCase。在这个类中，我们定义了两个方法：setUp 和 test\_add\_drug。

在 setUp 方法中，我们使用 User.objects.create\_user 方法创建一个用户，并使用 self.client.login 方法登录该用户。create\_user 方法接受两个参数：用户的邮箱和密码。login 方法也接受两个参数：用户的邮箱和密码。

在 test\_add\_drug 方法中，我们使用 self.client.post 方法发送一个 POST 请求到添加药品的 URL。请求的 URL 是通过 reverse 函数生成的，参数是 URL 的名称 'add\_drug'。请求的内容是一个包含药品名称、公司、数量和代码的字典。

发送请求后，我们使用 self.assertEqual 方法检查响应的状态码是否为 302（重定向）。接着，我们使用 self.assertTrue 方法检查药品是否成功添加。具体来说，我们检查 Drug 模型中是否存在药品名称为 'Aspirin' 的药品。

### 客户添加测试

在客户添加测试中，我们定义了一个 AddCustomerTest 类，该类继承自 TestCase。在这个类中，我们定义了一个 test\_add\_customer 方法，该方法用于测试客户添加功能。

在 test\_add\_customer 方法中，我们使用 self.client.post 方法发送一个 POST 请求到分发器 URL。请求的 URL 是通过 reverse 函数生成的，参数是 URL 的名称 'dispatcher'。请求的内容是一个包含操作类型和客户数据的 JSON 字符串。我们使用 json.dumps 方法将字典转换为 JSON 字符串，并将 content\_type 设置为 'application/json'。

发送请求后，我们使用 self.assertEqual 方法检查响应的状态码是否为 200。接着，我们使用 self.assertTrue 方法检查客户是否成功添加。具体来说，我们检查 Customer 模型中是否存在名称为 'John Doe' 的客户。

通过这些详细的测试用例，我们可以确保用户注册、用户登录、药品添加和客户添加功能的正常工作。这些测试用例验证了功能的正确性，还提供了详细的步骤和检查点。