To integrate the provided Selenium task with Django and run it asynchronously using Django-Q, you'll need to follow these steps:

### 1. ****Install Django-Q****

First, make sure you have Django-Q installed. You can install it using pip:

bash

Copy code

pip install django-q

### 2. ****Update Django Settings****

Add django\_q to your INSTALLED\_APPS and configure it in your settings.py:

python

Copy code

INSTALLED\_APPS = [

# other apps

'django\_q',

]

Q\_CLUSTER = {

'name': 'DjangoQ',

'workers': 4,

'timeout': 90,

'retry': 500,

'queue\_limit': 50,

'bulk': 10,

'orm': 'default',

}

### 3. ****Create a Django-Q Task****

Create a new file for tasks, for example, tasks.py inside one of your Django apps. Move your Selenium logic into a function in this file.

python

Copy code

# tasks.py

from selenium import webdriver

from selenium.webdriver.chrome.service import Service

from webdriver\_manager.chrome import ChromeDriverManager

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.common.exceptions import TimeoutException

import os

import requests

from PIL import Image

import pytesseract

import cv2

# Define the task function

def selenium\_task():

input\_email = "Waqasali885875867@gmail.com"

input\_password = "Azhar2233"

\_image = 'res/c2.jpg'

tesseract\_path = 'res/Tesseract-OCR/tesseract.exe'

def basic\_captcha\_solver(img\_path):

img = cv2.imread(img\_path)

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

\_, thresh = cv2.threshold(gray, 50, 255, cv2.THRESH\_BINARY\_INV)

white\_img = cv2.merge([255 - thresh, 255 - thresh, 255 - thresh])

cv2.imwrite('res/output\_image.jpg', white\_img)

image = Image.open('res/output\_image.jpg')

captcha\_text = pytesseract.image\_to\_string(image)

return captcha\_text.strip()

try:

chrome\_options = webdriver.ChromeOptions()

chrome\_options.add\_argument("--start-maximized")

driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=chrome\_options)

driver.get("https://blsitalypakistan.com/account/login")

email\_input = WebDriverWait(driver, 10).until(

EC.presence\_of\_element\_located((By.XPATH, "//input[@type='text' and @placeholder='Enter Email']"))

)

email\_input.send\_keys(input\_email)

password\_input = driver.find\_element(By.NAME, "login\_password")

password\_input.send\_keys(input\_password)

captcha\_image\_element = driver.find\_element(By.XPATH, "//img[@id='Imageid']")

captcha\_image\_url = captcha\_image\_element.get\_attribute("src")

captcha\_image\_response = requests.get(captcha\_image\_url)

captcha\_image\_path = os.path.join(os.getcwd(), "res/captcha\_image.jpg")

with open(captcha\_image\_path, 'wb') as file:

file.write(captcha\_image\_response.content)

extracted\_captcha\_text = basic\_captcha\_solver(captcha\_image\_path)

captcha\_input = driver.find\_element(By.NAME, "captcha\_code")

captcha\_input.send\_keys(extracted\_captcha\_text)

login\_button = driver.find\_element(By.XPATH, "//button[@name='submitLogin']")

login\_button.click()

print("Login successful")

page\_text = driver.find\_element(By.TAG\_NAME, "body").text

print(page\_text)

except TimeoutException:

print("Timeout while trying to login")

finally:

driver.quit()

### 4. ****Create a Django-Q Job****

To run this task asynchronously, you need to add it to the Django-Q job queue. You can do this in your Django view, management command, or wherever appropriate.

For example, in a Django view:

python

Copy code

# views.py

from django.shortcuts import render

from django\_q.tasks import async\_task

def start\_selenium\_task(request):

async\_task('your\_app\_name.tasks.selenium\_task')

return render(request, 'your\_template.html', {'message': 'Task has been started!'})

### 5. ****Start Django-Q Cluster****

To start the Django-Q cluster, use the following command:

bash

Copy code

python manage.py qcluster

This setup will ensure that your Selenium script runs asynchronously in the background using Django-Q, freeing up your web server to handle other requests.

The {{ bot.get\_status\_display }} syntax in a Django template is used to display the human-readable value of a field that has choices defined.

**How It Works:**

1. **Choices in Model Field:**
   * When you define a model field with choices, Django automatically creates a method called get\_FOO\_display(), where FOO is the name of the field.
   * This method returns the human-readable name for the field's current value.

For example, in your Bot model:

python

Copy code

class Bot(models.Model):

bot\_name = models.CharField(max\_length=255, unique=True)

class Status(models.TextChoices):

ACTIVE = "AE", "Active"

IDLE = "IE", "Idle"

status = models.CharField(

max\_length=2,

choices=Status.choices,

default=Status.IDLE

)

def \_\_str\_\_(self):

return self.bot\_name

* + The status field uses Status.choices, which defines two possible values: "AE" for "Active" and "IE" for "Idle".

1. **Using get\_status\_display():**
   * When you use bot.get\_status\_display() in your Django template, it will return the display name associated with the current value of the status field.
   * If bot.status is "AE", then bot.get\_status\_display() will return "Active".
   * If bot.status is "IE", then bot.get\_status\_display() will return "Idle".

**Example in Template:**

In your template:

html

Copy code

<div class="status-tab">

<h3>Bot Status</h3>

{% if bot %}

<h2 class="status-text">{{ bot.get\_status\_display }}</h2>

{% else %}

<h2 class="status-text">No Bot Found</h2>

{% endif %}

</div>

* **{{ bot.get\_status\_display }}**: This template tag calls the get\_status\_display() method on the bot object and outputs the human-readable value of the status field.

**Summary:**

* **bot.status** contains the internal value ("AE" or "IE").
* **bot.get\_status\_display()** returns the human-readable equivalent ("Active" or "Idle").

Using {{ bot.get\_status\_display }} in your template is a straightforward way to present a more user-friendly version of your status field directly in your UI.

The {{ bot.get\_status\_display }} syntax in Django templates is used to access the human-readable name for a model field that uses the choices option.

**How get\_status\_display() Works**

When you define a field in a Django model with the choices parameter, Django automatically provides a method named get\_FOO\_display() for that field. Here’s how it works:

* **FOO** is the name of the field. In your case, the field is named status.
* **get\_status\_display()** is a method generated by Django that returns the human-readable name associated with the current value of the status field.

**Example Breakdown**

Let's revisit your Bot model:

python

Copy code

from django.db import models

class Bot(models.Model):

bot\_name = models.CharField(max\_length=255, unique=True)

class Status(models.TextChoices):

ACTIVE = "AE", "Active"

IDLE = "IE", "Idle"

status = models.CharField(

max\_length=2,

choices=Status.choices,

default=Status.IDLE

)

def \_\_str\_\_(self):

return self.bot\_name

* **status Field:** This field can have one of two values: "AE" for "Active" or "IE" for "Idle". These values are stored in the database.
* **choices Parameter:** The Status inner class defines the possible choices for the status field. Each choice is a tuple where the first element is the actual value stored in the database (e.g., "AE"), and the second element is the human-readable name (e.g., "Active").

**Using get\_status\_display in a Template**

When you use {{ bot.get\_status\_display }} in your template:

* Django calls the get\_status\_display() method on the bot object.
* This method looks up the current value of the status field (e.g., "AE") and returns the corresponding human-readable name (e.g., "Active").

**Example in a Template**

If a Bot object has a status value of "AE", using {{ bot.get\_status\_display }} in your template will output "Active".

html

Copy code

<div class="status-tab">

<h3>Bot Status</h3>

<h2 class="status-text">{{ bot.get\_status\_display }}</h2> <!-- This will display "Active" if bot.status is "AE" -->

</div>

**Summary**

* **get\_status\_display()** is a Django-provided method that converts a stored value (e.g., "AE") into its human-readable equivalent (e.g., "Active").
* **{{ bot.get\_status\_display }}** in a template displays this human-readable value, making your app more user-friendly by showing descriptive labels rather than raw data values.

In Django, when you define a model field with choices, Django automatically provides a method to get the human-readable name for the field's value. For a field like status in your Bot model, which uses choices, you can use the get\_FOO\_display() method to get the human-readable version of the field’s value.

Here's a detailed explanation of {{ bot.get\_status\_display }}:

**1. Field Definition with Choices:**

In your Bot model, the status field uses the choices option:

python

Copy code

class Bot(models.Model):

class Status(models.TextChoices):

ACTIVE = "AE", "Active"

IDLE = "IE", "Idle"

status = models.CharField(

max\_length=2,

choices=Status.choices,

default=Status.IDLE

)

**2. What get\_FOO\_display() Does:**

Django provides a method called get\_FOO\_display() for any field defined with choices. In your case, it would be get\_status\_display().

* **get\_status\_display() Method:** This method returns the human-readable name of the status field.

For example, if status is set to "AE" (which corresponds to "Active"), then bot.get\_status\_display() will return "Active". If status is set to "IE" (which corresponds to "Idle"), then bot.get\_status\_display() will return "Idle".

**3. Using get\_status\_display in a Template:**

In your Django template, you can use {{ bot.get\_status\_display }} to display the human-readable name of the status:

html

Copy code

<div class="status-tab">

<h3>Bot Status</h3>

{% if bot %}

<h2 class="status-text">{{ bot.get\_status\_display }}</h2>

{% else %}

<h2 class="status-text">No Bot Found</h2>

{% endif %}

</div>

**How It Works:**

* **Field Value:** The status field stores a value like "AE" or "IE".
* **get\_status\_display():** This method looks up the value in the choices definition and returns the corresponding human-readable name.

**Summary:**

* **Purpose:** {{ bot.get\_status\_display }} provides a way to render the user-friendly label for the status field in your template, instead of displaying the raw value ("AE" or "IE").
* **Usage:** Use {{ bot.get\_status\_display }} in templates to show descriptive text instead of raw choice values.

This makes your templates more readable and user-friendly by presenting the actual names of the choices rather than the internal codes.

Python code using OpenCV to remove all colors apart from black and replace them with white:

Python

import cv2

import numpy as np

# Load the image

img = cv2.imread('input\_image.jpg')

# Convert the image to grayscale

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Threshold the grayscale image to get only black pixels

\_, thresh = cv2.threshold(gray, 50, 255, cv2.THRESH\_BINARY\_INV)

# Create a 3-channel image with white pixels where threshold is 255

white\_img = cv2.merge([255-thresh, 255-thresh, 255-thresh])

# Output the resulting image

cv2.imwrite('output\_image.jpg', white\_img)

# Display the output image

cv2.imshow('Output Image', white\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

This code works by:

1. Loading the input image
2. Converting the image to grayscale
3. Thresholding the grayscale image to get a binary image with black pixels (value 0) and non-black pixels (value 255)
4. Creating a 3-channel image with white pixels (value 255) where the threshold is 255 (non-black pixels)
5. Saving and displaying the output image

Note:

* You can adjust the threshold value (50) to include different shades of black.
* This code assumes that "black" means pixel values close to (0, 0, 0). If your image has dark gray or dark brown pixels that you want to consider "black", adjust the threshold value accordingly.

Replace 'input\_image.jpg' with the path to your input image.

To change the status of the bot from "Idle" to "Active" using the switch in your Django model, you need to set up a few things:

1. **JavaScript to handle the switch toggle:**
   * When the switch is toggled, send an AJAX request to update the bot's status.
2. **Django view to handle the AJAX request:**
   * A view that receives the request and updates the bot's status in the database.
3. **URL mapping for the view:**
   * A URL that points to the view that handles the status change.

Here’s how you can implement this:

**1. Update the Switch Form to Include JavaScript**

html

Copy code

<form id="status-form" method="post">

{% csrf\_token %}

<!-- Rounded switch -->

<label class="switch">

<input id="status-switch" name="switch" type="checkbox" {% if bot.status == 'AE' %} checked {% endif %}>

<span class="slider round"></span>

</label>

</form>

<script>

document.getElementById('status-switch').addEventListener('change', function() {

const isChecked = this.checked;

const botId = "{{ bot.id }}";

const csrfToken = "{{ csrf\_token }}";

fetch("{% url 'change\_status' bot.id %}", {

method: "POST",

headers: {

"X-CSRFToken": csrfToken,

"Content-Type": "application/json"

},

body: JSON.stringify({ status: isChecked ? 'AE' : 'IE' })

})

.then(response => response.json())

.then(data => {

console.log(data.message);

})

.catch(error => {

console.error('Error:', error);

});

});

</script>

**2. Create the Django View**

python

Copy code

from django.http import JsonResponse

from django.views.decorators.http import require\_POST

from django.views.decorators.csrf import csrf\_exempt

from .models import Bot

@csrf\_exempt

@require\_POST

def change\_status(request, bot\_id):

try:

bot = Bot.objects.get(id=bot\_id)

data = json.loads(request.body)

new\_status = data.get('status')

if new\_status in dict(Bot.Status.choices):

bot.status = new\_status

bot.save()

return JsonResponse({"message": "Status updated successfully."}, status=200)

else:

return JsonResponse({"message": "Invalid status value."}, status=400)

except Bot.DoesNotExist:

return JsonResponse({"message": "Bot not found."}, status=404)

**3. Add the URL Mapping**

In your urls.py:

python

Copy code

from django.urls import path

from .views import change\_status

urlpatterns = [

# Other paths...

path('change-status/<int:bot\_id>/', change\_status, name='change\_status'),

]

**4. CSRF Token**

Ensure that the {% csrf\_token %} is included in your form and passed correctly to the JavaScript fetch request, which is already done in the provided code.

**How This Works**

* **Form and Switch:** The form includes a checkbox switch that reflects the bot's current status. The switch is checked if the status is "Active" (AE).
* **JavaScript:** When the switch is toggled, a JavaScript event listener sends an AJAX request to the Django backend with the new status.
* **Django View:** The view receives the request, updates the bot's status in the database, and returns a JSON response.

With this setup, toggling the switch in the UI will change the status of the bot between "Idle" and "Active" in the database.