نکته مهمی که در کلاس UserRegister وجود دارد این است که هنگامیکه کاربر از طریق متد post اطلاعات خود را ارسال کند به دلیل این که در template مربوطه novalidate وجود دارد خطاها باید از طرف جنگو ارسال شود برای همین اگر فرم valid نبود باید return زیر وجود داشته باشد :

template\_name = 'accounts/register.html'

return render(request, self.template\_name, {'form':form})

و باید حتما از render استفاده شود و نباید در این مورد redirect استفاده شود چرا که ارور های جنگو در redirect هندل نمیشود

================================================================================

کلاس LoginRequiredMixin برای این مورد کاربرد دارد که کلاس ها وتوابعی که لازم است حتما کاربر لاگین شده باشد تا کاری را انجام دهد بررسی میکند

================================================================================

هنگامیکه بخواهید از طریق فرمها عکس آپلود کنید حتما باید در فرمی که در templateها ساخته اید کد زیر را قرار دهید

<form action="" method="post" enctype="multipart/form-data">

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

برای دسترسی به فایلی که آپلود کرده اید میتوانید از طریق request.FILES به آن دسترسی داشته باشید و هنگامیکه میخواهید فرم ProfileImageForm را ایجاد کنید باید به روش زیر انجام دهید

form = self.form\_class(request.POST, request.FILES, instance=request.user.profile)

================================================================================

دو روش برای ذخیره file ها درون bucket آروان کلود وجود دارد روش اتوماتیک و روش دستی :

برای روش اتوماتیک از پکیج django-storages استفاده میکنیم

https://django-storages.readthedocs.io/en/latest/

همانطور که در داکیومنت بالا میبینید انواع مختلف storage ها را پشتیبانی میکند

در این قستمت از Amazon S3 استفاده میکنیم

https://django-storages.readthedocs.io/en/latest/backends/amazon-S3.html

django-storages از پکیج boto3 استفاده میکند که boto3 یک sdk میباشد به معنی softwate development kit .  
boto3 از طرف شرکت آمازون به برنامه نویسان پایتون داده شده است که بتوانند از طریق آن به سرویس های آمازون متصل شوند که ما میتوانیم به صورت دستی از boto3 استفاده کنیم

برای آپلود کردن و ذخیره کردن media file ها میتوانیم از مقدار زیر استفاده کنیم

DEFAULT\_FILE\_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'

و برای آپلود کردن و ذخیره کردن static file ها میتوانیم از مقدار زیر استفاده کنیم

STATICFILES\_STORAGE = 'storages.backends.s3boto3.S3StaticStorage'

بعد از نصب پکیج django-storages با دستور pip install django-storages باید storages را به install app های django اضافه کنیم

مقادیر بعدی که باید بعد از DEFAULT\_FILE\_STORAGE باید تغریف شوند  
1. AWS\_ACCESS\_KEY\_ID میباشد که باید به عنوان str ذخیره کنید و مقدار Access key آروان کلود را باید قرار دهید 2. مقدار بعدی AWS\_SECRET\_ACCESS\_KEY میباشد که باید مقدار Secret key آروان کلود را به عنوان str به آن بدهید   
3. مقدار بعدی AWS\_STORAGE\_BUCKET\_NAME میباشد که باید اسم bucket خود را قرار دهید (در این پروژه sajjadtest میباشد)  
4. مقدار بعدی AWS\_SERVICE\_NAME میباشد که در حالت اتوماتیک django-storage به آن نیازی ندارد اما برای حالت دستی به آن نیاز پیدا میکنیم و مقدار آن را برابر s3 قرار میدهیم   
5. مقدار بعدی AWS\_S3\_ENDPOINT\_URL میباشد که به این معنی میباشد که نهایتا خوده storage کجاست   
url storage که همان Endpoint URL آروان کلود میباشد را باید قرار دهید   
6. مقدار بعدی یک مقدار پیشنهادی میباشد و لازم نیست که حتما مقدار دهی شود چرا که به صورت پیش فرض مقدار   
AWS\_S3\_FILE\_OVERWRITE برابر true میباشد و باید آن را با مقدار false عوض کنیم این مقدار را اگر false قرار دهیم به این معنی میباشد که اگر فایلی upload شد و هم نام با فایل دیگری بود فایلی که قبلا وجود داشت را overwrite نکن و اسم جدید به فایلی که قرار است upload شود بده   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

نکته ی مهمی که در کلاس UserDashboard وجود دارد این است که usernameی که از طریق url به این کلاس فرستاده میشود فقط متد get به آن نیاز دارد و متد post به آن نیازی ندارد حال اگر این مورد هندل نشود به ارور میخوریم چرا که ورودی username برای post تعریف نشده است بهترین کار اضافه کردن args و kwargs به پارامتر های متد post میباشد

================================================================================

در این قسمت به توضیخ اینکه چگونه به bucketهای آروان کلود به صورت دستی متصل شویم و عمل crud را انجام دهیم میپردازیم به دلیل اینکه این موارد زمانبر هستند باید به صورت asynchronous انجام شوند به خاطر همین باید از celery و یک بروکر استفاده شود   
برای نوشتن configهای celery باید فایلی در کنار settings.py ساخته شود و این config ها در آن فایل نوشته شود ( اسم فایل مورد نظر در این پروژه celery\_conf.py میباشد )

اولین موردی که باید ایجاد شود environment variable میباشد که celery متوجه میشود تنظیمات شما کجاست :

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'A.settings')

دومین مورد ایجاد یک instance جدید از کلاس Celery میباشد ( دقت شود که A اسم پروژه میباشد )

celery\_app = Celery(‘A’)

سومین مورد این است که task هایی که مربوط به celery در پروژه وجود دارد را به صورت auto کشف کن (نحوه پیدا کردن به این صورت میباشد که در appهاتون هر فایلی که با نام task.py باشد را کشف میکند )

celery\_app.autodiscover\_tasks()

چهارمین مورد broker میباشد که در این پروژه از rabbitmq استفاده شده است ( باید آدرس rabbitmq همراه با username و password قرار بگیرد به صورت پیشفرض quest برای username و پسورد میباشد )

celery\_app.conf.broker\_url = ‘amqp://quest:[quest@localhost](mailto:quest@localhost)’

پنجمین مورد result\_backend میباشد به دلیل اینکه از rabbitmq استفاده میکنیم باید rpc قرار دهیم

celey\_app.conf.result\_backend = ‘rpc://’

ششمین مورد task\_serializer میباشد . تعیین میکنیم که تسک هایی که قرار است فرستاده شود به صورت json باشد

celery\_app.conf.task\_serializer = ‘json’

هفتمین مورد result\_serializer میباشد . تعیین میکنیم که نتیجه ی تسکها به صورت pickle باشد (pickle اون objectهایی که شما دارید رو داخل یک محیط ایزوله نگهداری میکنیم و آنها را به pickle در می آورد و آنها را میتوانید داخل دیتابیس دخیره کنید یا داخل شبکه جابجا کنید در حالت عادی امکان ذخیره کردن object وجود ندارد)

celery\_app.conf.result\_serializer = ‘pickle’

هشتمین مورد accept\_content میباشد . مقادیری که قرار است قبول کنی از نوع json و pickle میباشد

celery\_app.conf.accept\_contetnt = [‘json', ‘pickle’]

نهمین مورد result\_expires میباشد . تاریخ انقضای تسک هارا مشخص میکند . تاریخ انقضا برابر ۱ روز میباشد

celery\_app.conf.result\_expires = timedelta(days=1)

دهمین مورد task\_always\_eager میباشد . اگر برابر true باشد به این معنی میباشد که این تسکی که در حال انجام است میاد و کل برنامه را بلاک میکند تا زمانی که نتیجه اش برگردد ( به صورت پیش فرض هم false میباشد )

celery\_app.conf.task\_always\_eager = False

یازدهمین مورد worker\_prefetch\_multiplier میباشد . به این معنی که worker های من به صورت همزمان چند تسک را قبول کنند (پیشفرض این مورد 4 میباشد) هر چه مدت زمان انجام تسک طولانی تر باشد عدد را کمتر کنید (0 به این معنی میباشد که worker ها هرچندتا تسک که دلشون میخواست بگیرن)

celery\_app.conf.worker\_prefetch\_multiplier = 1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

بعد از set کردن موارد بالا باید در فایل py.ــinitــ باید خط زیر را قرار دهید ( این مورد برای این است که به محض اجرای پروژه celery نیاز اجرا شود )

from A.celery\_conf import celery\_app

================================================================================

برای وصل شدن به صورت دستی به آروان کلود از پکیجی به نام boto3 استفاده میکنیم

https://boto3.amazonaws.com/v1/documentation/api/latest/index.html

s3 ===> simple storage service

https://www.arvancloud.com/help/fa/article/360012836040-%D8%A2%D8%BA%D8%A7%D8%B2-%DA%A9%D8%A7%D8%B1-%D8%A8%D8%A7-%D9%85%D8%AD%D8%B5%D9%88%D9%84-%D9%81%D8%B6%D8%A7%DB%8C-%D8%A7%D8%A8%D8%B1%DB%8C-%D8%A2%D8%B1%D9%88%D8%A7%D9%86

url بالا راهنمای آروان کلود میباشد

برای ایجاد کردن اتصال ابتدا از طریق boto3 یک session را ایجاد میکنید بعد از طریق این اتصال اطلاعات لازم را قرار میدهید

در آموزشی که آروان کلود قرار داده است میتوانید از طریق آن لیست تمام باکت ها , دریافت اطلاعات یک باکت , آپلود کردن , ایجاد یک url خاص برای فایل وجود دارد و پاک کردن فایل وجود دارد (درباره ی دانلود کردن آموزشی وجود ندارد)

به طور کلی بهتر است مواردی که تمام تسک های celery به آن نیاز دارند مثلا conection را در یک فایل جداگانه قرار دهیم این فایل bucket.py میباشد

اولین متدی که قرار است در کلاس Bucket ساخته شود get\_object میباشد که از طریق این متد میخواهیم اطلاعات هر bucket را نمایش دهیم . طبق مستندات آروان کلود باید از list\_object\_v2 استفاده کنیم   
به دلیل اینکه خواندن bucket از آروان کلود عملی زمان بر میباشد باید آن را از طریق celery بخوانیم

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

به دلیل اینکه باید از کلاس Bucket در فایل task.py استفاده کنیم و برای هر فانکشن باید یک instance ساخته شود بهتر است این instance در خود فایل bucket.py ساخته شود

bucket = Bucket()

اکنون میتوانیم از bucket در کل توابع فایل task.py استفاده کنیم

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

حال در کلاس BucketHome میتوانیم از فانکشن all\_bucket\_objects\_task استفاده کنیم . وقتی از فانکشن delay استفاده کنیم فانکشن all\_bucket\_objects\_task به صورت async رفتار میکند

================================================================================

به صورت کلی سعی شود که داخل متدها مقداری را return کنید چرا که به صورت پیشفرض اگر در پایتون و جنگو چیزی را return نکنید None برگشت داده میشود

================================================================================

به دلیل اینکه در داکیومنت های آروان کلود چیزی در مورد دانلود کردن فایل وجود ندارد میتوانید از داکیومنت زیر که مربوط به boto3 میباشد استفاده کنید

https://boto3.amazonaws.com/v1/documentation/api/latest/guide/s3-example-download-file.html

منظور از file\_name همان چیزی هست که میخواهید ذخیره کنید ( اسمی که میخواهید با آن ذخیره کنید )

================================================================================