Channel App Template

Yayım 0.1.0

Eyüp Tuğrul, Mustafa Yetiş, Sabri Özgür

01 Eyl 2022

İçindekiler

Channel App Template projesi, Akinon Commerce Cloud (ACC) üzerinde farklı pazaralanlarına yapılacak entegrasyonlar için taslak oluşturmak amacıyla hazırlanmıştır.

Channel App Template projesi ile birlikte yeterli teknik uzmanlığa sahip şirketler, istedikleri pazaralanlarına entegrasyon yapabilirler.

Bunun için projeyi klonlayıp, gerekli bağlantı kodlarını yazarak, ürün, stok, fiyat gibi akışlarda hedef pazaralanına özgü özelleştirmeleri de yapmaları yeterlidir. Bu adımlardan sonra, yeni bir uygulama olarak ACC üzerinde kurulabilir.

Yerel ortamda testler tamamlanıp gerekli joblar hazır olduğu düşünüldüğünde, geliştirmeleri yayına alma veya sunucularda test etmek gerekecektir.

developers.akinon.com

dokümanındaki adımları izleyerek ACC üzerinde *project* ve *application* tanımlamanız gerekmektedir. Bu adımları aynı uygulama için daha önce takip ettiyseniz kodun ve etiketlerin gönderildiği noktaya geçebilirsiniz.

BÖLÜM 1

Teknolojiler ve Kütüphaneler

1. Python 3.8
2. Celery 5
3. Flower [Opsiyonel]
4. Sentry [Opsiyonel]

BÖLÜM 2

İçerik

2.1 Kurulum ve Kullanım

Yazılacak uygulama Omnitron'a bir Akinon Commerce Cloud (ACC) uygulaması olarak bağlanacak ve o uygulamaya tanımlı kullanıcı bilgilerini kullanacak.

Yerel ortam için test etedilecek senaryoda Omnitron kullanıcı bilgileri ile ACC uygulamasını hazırlamadan da test edilebilir. Yapılacak geliştirmelerin hazır olduğu düşünülen noktada uygulamayı sunucu ortamında(ACC) yayına alınabilir ve test edilebilir.

Acc tarafında application oluşturulduktan sonra kullanılacak olan projede add new app diyerek geliştrilen uygulama seçilir ve install denilir. Markanın omnitronunda channel ve katalog otomatik olarak oluşur.

Dokümantasyon Ubuntu tabanlı işletim sistemleri için hazırlanmıştır.

2.1.1 Python Kurulumu

- python sürümü 3.8 ve üzerinde ise Channel App Template projesi için yeterlidir.

```
python --version
python3 --version

$: Python 3.8.10
```

- Eğer Python yüklü değilse ya da sürümünüz eskiyse Python kurulumunu apt komutlarıyla tamamlayabilirsiniz. Öncelikle *apt install* komutunu denemeniz tavsiye edilir. Versiyonu bulamaması durumunda add-apt-repository ile ilerleyip sonrasında tekrar *apt install* ile kurulumu yapabilirsiniz. Bu şekilde sisteminizdeki *apt* aradığınız pakete ulaşabiliyorsa yeni bir *ppa* eklemenize gerek kalmaz.

```
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update -y
sudo apt install python3.10
```

- *apt* komutlarıyla gerekli paketleri sisteme yüklüyoruz. Bunlar Python paket yönetici pip, versiyon kontrol aracı Git, task sunucusu Celery ve Celery'nin broker ihtiyacı için Redis'ten oluşuyor.

```
sudo apt install python3-pip git redis-server python-celery-common
pip3 install --upgrade pip
```

- Farklı projelerin bağımlıkları arasında çakışma olmaması adına virtual environment kullanmanız önerilir. Virtual environment kurulumu için aşağıdaki linki takip edebilirsiniz.

  Virtual environment kurulumu

2.1.2 Projenin Kopyalanması ve Zorunlulukların Kurulması

- Projeyi lokal sistemimize kopyalıyoruz. git klonlama işlemini terminalde aktif olan klasöre yapacağı için, komutu çalıştırmadan önce tercih ettiğiniz proje klasörüne geçmeniz önerilir.

```
git clone git@bitbucket.org:akinonteam/channel_app_template.git
```

- Aktif virtual environment ile proje için gerekli paketleri kurabilirsiniz.

```
pip install -r requirements.txt
```

2.1.3 Celery ve Flower Kurulumu

Celery, dağıtık mesaj geçişine dayanan açık kaynaklı, bir asenkron görev kuyruğu veya iş kuyruğudur. Genellikle gerçek zamanlı görev işleme üzerine yoğunlaşsa ve kullanılsa dahi zamanlanmış görevleri de yönetebilir. Detaylı bilgi için (Celery)

- Celery işçi processleri de çalıştıralım. Kullanıcı bilgileri ve bazı ortam değişkenleri sizin ortamınız için farklı değerlerde olacaktır. MAIN_APP_URL: Protokol bilgisi hariç Omnitron url'i OMNITRON_CHANNEL_ID: Uygulamanın bağlanacağı satış kanalı id değeri. OMNITRON_CATALOG_ID: Bağlı satış kanalının katalog id değeri.

```
# Topluca export (öncesinde .env dosyasını oluşturmak gerekiyor)
export $(grep -v '^#' .env | xargs)

# Tek tek export
export MAIN_APP_URL=localhost:8000
export OMNITRON_USERNAME=admin
export OMNITRON_PASSWORD=password
export OMNITRON_CHANNEL_ID=1
export OMNITRON_CATALOG_ID=1
export BROKER_HOST=127.0.0.1
export BROKER_PORT=6379
export BROKER_DATABASE_INDEX=4
export CACHE_HOST=127.0.0.1
export CACHE_PORT=6379
export CACHE_DATABASE_INDEX=3

celery -A channel_app.celery_app worker -l info
```

- Redis sunucusu varsayılan olarak kurulum sonrası özellikle kapatılmadıkça ayakta oluyor. Ping komutuyla test edip *redis-server* ile kaldırabilirsiniz.

```
redis-cli ping
redis-server
```

Flower celery ile ilgili yapılan işlemleri görüntülemek ve yönetmek için kullanılan web tabanlı bir araçtır. Eğer yönetilecek düzenli çalışan joblarınız yoksa veya alternatif bir araç kullanıyorsanız kurulumu zorunlu değildir.

- Flower kurmak için bazı ortam değişkenlerini *export* etmemiz gerekiyor. Her seferinde tek tek export yapmak yerine, bu değerleri *.env* dosyasına KEY=VALUE şeklinde kaydedip topluca export edecek komutu da çağırabilirsiniz.

```
# Topluca export (öncesinde .env dosyasını oluşturmak gerekiyor)
export $(grep -v '^#' .env | xargs)

# Tek tek export
export BROKER_HOST=127.0.0.1
export BROKER_DATABASE_INDEX=4
export BROKER_PORT=6379

celery -A channel_app_template.celery_app flower --address=127.0.0.1 --port=8008
```

Flower Üzerinden Task Tetiklemek

Flower ayni zamanda ön tanımlı olan taskları tetiklemeyi sağlar. Taskları tetiklemek için aşağıdaki örnek **curl** isteği aşağıdaki gibidir.

```
curl --request POST \
  --url https://<omnitron-url>.lb.akinoncloud.com/api/task/apply/channel_app_template.
↪app.tasks.<task_name> \
  --header 'authorization: Basic <auth token>' \
  --header 'content-type: application/json' \
  --data '{\n"args":[]\n}'
```

İstek başarılı olur ise 200 status kodu ile aşağıdaki cevabı döner.

```
HTTP/1.1 200 OK
Content-Length: 71
Content-Type: application/json; charset=UTF-8
{
  "state": "SUCCESS",
  "task-id": "c60be250-fe52-48df-befb-ac66174076e6",
  "result": 3
}
```

Dönen cevap içerisindeki **task-id** parametresi ile flower paneli üzerinden tetiklenen taskın durumu sorgulanabilir.

Yukarıdaki işlemleri tamamladıktan sonra sistem için gerekli her şey hazır.

2.2 Geliştirme Adımları

2.2.1 Klasör Yapısı

Kopyalacağınız proje ile ilgili klasör ve dosya yapısı aşağıdaki gibidir. ChannelTemplateApp projesini kullanan firmaların yeni satış kanalı geliştirmek için *channel* klasörü altındaki ilgili sınıfları geliştirmeleri beklenmektedir.

```
channel_app_template
├── akinon
│   └── integration.py
├── channel
│   └── integration.py
│   └── commands.py
│       └── setup.py
│       └── products.py
│       └── product_prices.py
│       └── product_stocks.py
│       └── product_images.py
│       └── orders
│           └── orders.py
├── app
│   └── tasks.py
└── celery_app
    └── celery.py
    └── celery_schedule_conf.py
    └── celeryconfig.py
```

2.2.2 Başlangıç Adımı

Dikkat: Geliştirmeye başlamadan önce aşağıdaki ayarları yapmak gerekiyor.

1. **setup.py** dosyasına yer alan proje adı, versiyon bilgisi, uygulamanın depolandığı (bitbucket, github vb…) reponun url'i, uygulamanın tanımı, uygulamanın yazar veya yazarları, python versiyon zorunluluğu ve kurulu olması gereken paketlerin listesi gibi bilgilerin girilmesi gerekiyor.
2. **channel_app_template/celery_app/celery_schedule_conf.py** içerisinde yer alan **CELERYBEAT_SCHEDULE** objesi içerisinde çalışmasını istediğiniz süreçler bulunmaktadır. Akinon ile yapılacak satış kanalı entegrasyonunda kullanılacak süreçler ön tanımlı olarak yazılmış ve yorum satırına alınmıştır. Düzenli aralıklar ile çalışmasını gereken süreçleri yorum satırından kaldırıp, ne sıklık ile çalışması gerektiğini yazabilirsiniz.

`class` channel_app_template.channel.integration.**ChannelIntegration**
Actions özelliğinde yer alan komutlar aracılığı ile Satış Kanalının servisleri haberleşir. Her komut üzerinde *get_data*, *validated_data*, *transform_data*, *send_request* ve *normalize_response* fonksiyonlarını barındırmak zorundadır

channel
Satış kanalı objesidir. Channel Detaylı Bilgi Örnek channel objesi.

Schema alanında FAILED_INTEGRATION, ATTRIBUTE_SET_STRATEGY eklenmsi zorunlu alanlardır.

1. ATTRIBUTE_SET_STRATEGY: "CategoryNode" veya "None" değerleri alabilir. Category nodeların attribute setlerin category bağlı olup olmadığı bilgisi için gereklidir.

2. FAILED_INTEGRATION: Hata alınan işlemlerin tekrar edilmesi için gereken sürelerin girildiği alandır.

```
{
        "pk": 1,
        "name": "test1:1_Channel",
        "channel_type": "sales_channel",
        "catalog": 1,
        "modified_date": "2022-04-01T11:35:56.485644Z",
        "created_date": "2022-02-18T14:17:35.169367Z",
        "category_tree": 24,
        "is_active": True,
        "conf":
        {
            "FAILED_INTEGRATION":
            {
                "DEFAULT":
                {
                    "EXPIRATION_DATE": 213,
                    "RETRY_INTERVAL": 0,
                    "MAX_RETRY_COUNT": 3
                }
            },
            "base_url": "https://vendor-api-staging.saleschannel.com/",
            "client_id": "69b5c9c1-20e1-4c62-89f9-7186bde1b13f",
            "ATTRIBUTE_SET_STRATEGY": "CategoryNode"
        },
        "schema": {
            "FAILED_INTEGRATION": {
                "label": "FAILED_INTEGRATION",
                "key": "FAILED_INTEGRATION",
                "data_type": "json",
                "required": True},
            "ATTRIBUTE_SET_STRATEGY": {
                   "key": "ATTRIBUTE_SET_STRATEGY",
                   "label": "ATTRIBUTE_SET_STRATEGY",
                   "required": True,
                   "data_type": "text"}}
}

>>> self.channel.pk
1
>>> self.name
test1:1_Channel
>>> self.conf.get("base_url")
https://vendor-api-staging.saleschannel.com/
```

catalog

Katalog objesidir. Katalog Detaylı Bilgi. Örnek katalog objesi

```
{
    "pk": 1,
    "name": "test1:1_Catalog",
    "stock_list": None,
    "price_list": None,
    "category_tree": None,
    "modified_date": "2022-02-18T14:17:35.159703Z",
    "created_date": "2022-02-18T14:17:35.159683Z",
    "priority_list": None,
    "extra_stock_lists": [],
    "extra_price_lists": []
}

>>> self.catalog.pk
1
```

create_session()
Session nesnesi, belirli parametreleri istekler arasında kalıcı hale getirmenize olanak tanır. Ayrıca, Session ile yapılan tüm isteklerde tanımlama bilgilerini(Çerezleri) taşır. Bu nedenle, aynı ana bilgisayara birden fazla istekte bulunuyorsanız, temeldeki TCP bağlantısı yeniden kullanılacak ve bu da önemli bir performans artışına neden olacaktır. Daha detaylı bilgi için Session Objects

session
Satış kanalının servislerine istek atılırken kullanılacak objedir. Session objesi komutlar içerisinde ki *send_request* fonksiyonu içerisinde kullanılabilir.

```
>>> session.get("google.com")
```

do_action(*key: str*, ***kwargs*) → Any
Servisler aracılığı ile tetiklenir. Çağırabilmek için öncesinde bir entegrasyon nesnesi yaratılmış olmalıdır. Parametre olarak verilen key çalışacak olan komutu temsil eder. Bu komut ilgili entegrasyonun *actions* özelliği içerisinde olmalıdır. *kwargs* olarak verilen parametreler doğrudan Komut nesnesi oluşturmak için kullanılır. Son olarak ilgili komutun *run* fonksiyonunu çağırarak komutun çalışmasını sağlar.

```
with OmnitronIntegration(content_type=ContentType.category_tree.value) as␣
↪omnitron_integration:
   channel_integration = ChannelIntegration()
   category_tree, report, _ = channel_integration.do_action(
       key='get_category_tree_and_nodes',
       batch_request=omnitron_integration.batch_request)
```

2.2.3 Satış Kanalının Kodlanması

Bu klasör içerisinde uygulamanın yazılma amacı olan satış kanalının kodlamasının yapılacağı yerdir.

Entegre olunacak satış kanalı için kodlanlamasını istediğimiz commandların listesi aşağıdaki gibidir.

Module	Açıklama
Setup	Kurulum aşamasına ait komutlar
Products	Ürün ile ilgili komutlar
Product Price	Fiyat ile ilgili komutlar
Product Stock	Stok ile ilgili komutlar
Product Image	Resim ile ilgili komutlar
Orders	Sipariş ile ilgili komutlar

Setup

Kurulum adımlarında Sistem genelinde çalışan düzenli job'lara ait kodlar *channel_app_template.app.tasks* dosyasında yer almaktadır. Setup adımları ile ilgili 3 adet düzenli çalışabilecek durumda olan job vardır.

1. **update_channel_conf_schema**

   Düzenlenmesi gereken class GetChannelConfSchema

   İş sürecinde oluşturulacak olan satış kanalı için ihtiyaç duyulan ayarları oluşturur. İşlem sırası olarak öncelikle satış kanalı entegrasyonunda yapılan gelişltirme aracılığıyla ayarları okur, sonrasında Akinon'a iletir.

   Setup Servisi içerisinde yer alan update_channel_conf_schema fonksiyonu ile süreç işletilir.

   Satış kanalının içerisinde erişmek istediğiniz ayarlarınızı key, value ikilisi koyabilir ve buna göre geliştirmenizi yapabilirsiniz.

2. **create_or_update_category_tree_and_nodes**

   Düzenlenmesi gereken class GetCategoryTreeAndNodes

   Satış kanalına ait kategori ağacını akinon tarafında oluşturur.

3. **create_or_update_category_attributes**

   Düzenlenmesi gereken class GetCategoryAttributes

   Satış kanalının kategorilerine göre attribute, attribute value değerlerini omnitronda oluştururan servistir.

4. **create_or_update_attributes**

   Düzenlenmesi gereken class GetAttributes

   Satış kanalında bulunan attribute, attribute value değerlerini omnitronda oluştururan servistir.

   Bu servis kategoriye bağlı attributeların olmadığı senaryolarda kullanılabilir.

Products

Ürün entegrasyonu ile ilgili channel_app_template.app.tasks altında çalışan tasklar

1. insert_products

   Akinon'da satış tarafına ilk defa insert edilmesi gereken ürünleri alır ve bu ürünleri `channel.commands.products.SendInsertedProducts`'da yer alan komut'a gönderir. Bu komut ile satış kanalına ürün bilgileri oluşturulur.

   İstenildiği durumda aşağıda listesi verilen parametre değerleri ile Ürün Verisi zenginleştirilebilir.

   ProductService içerisinde yer alan insert_products servisine ait parametreler ve açıklamaları aşağıdaki gibidir.

Dikkat: *Ürün Servisi* içerisinde yer alan insert_products servisine ait parametreler

add_mapped : Mapping bilgileri ürün verisine eklenir. Üründe Mapping Verisi

add_stock : Ürün stok bilgileri ürün verisine eklenir. Üründe Stok Verisi

add_price : Ürün fiyat bilgileri ürün verisine eklenir. Üründe Fiyat Verisi
add_categories : Ürün kategori bilgileri ürün verisine eklenir. Üründe Kategori Verisi
is_sync : Ürün satış kanalına yollandığında durumu hemen mi ediniliyor Senkron veya Asenkron Satış Kanalı Süreç yoksa asenkron bir şekilde mi ediniliyor olduğudur.

class **SendInsertedProducts**(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)

get_data()

Bu fonksiyon productları omnitron'dan satış kanalına iletmek için atılacak istekte gönderilecek veri hazırlanır. Response olarak liste içerinde productlar döndürülmesi gerekir.

validated_data(*data*)

Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalına gönderilecek productlar üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

```
valid_data = defaultdict(list)
for product_id, products in data.items():
    images = self.get_images(products)
    if len(images) < 3:
        for product in products:
            product.failed_reason_type = FailedReasonType.remote.
↪value
            self.failed_object_list.append(
                (product, ContentType.product.value,
                 "Product images counts are less than three"))
    else:
        valid_data[product_id].extend(products)
return valid_data
```

transform_data(*data*)

Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

send_request(*transformed_data*)

Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

Dikkat: Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

normalize_response(*data*, *validated_data*, *transformed_data*, *response*)

Bu fonksiyon insert_product adımında productlarımızı satış kanalına iletmek için kullanmış olduğumuz verileri toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceğ cevap doğrudan insert_products fonksiyonunda kullanılacaktır.

Bu methoda süreç asenkron ise satış kanalından dönen remote_batch_id batch_request'e işlenmelidir.

```
>>> remote_batch_id = response.get("remote_batch_request_id")
>>> self.batch_request.remote_batch_id = remote_batch_id
>>> return "", report, data
```

Dikkat: Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.

response_data: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi ProductBatchRequestResponseDto olmak zorundadır.

report: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.

data: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.

```
# örnek return
return response_data, report, data
```

2. update_products

Akinon'da satış tarafına daha önce insert edilmiş fakat güncellenmesi gereken ürünleri alır ve bu ürünleri *channel.commands.products.SendUpdatedProducts*'da yer alan komut'a gönderir. Bu komut ile satış kanalına mevcut olan ürün bilgileri güncellenir.

İstenildiği durumda aşağıda listesi verilen parametre değerleri ile Ürün Verisi zenginleştirilebilir.

ProductService içerisinde yer alan update_products servisine ait parametreler ve açıklamaları aşağıdaki gibidir.

Dikkat: *Ürün Servisi* içerisinde yer alan update_products servisine ait parametreler

add_mapped : Mapping bilgileri ürün verisine eklenir. Üründe Mapping Verisi

add_stock : Ürün stok bilgileri ürün verisine eklenir. Üründe Stok Verisi

add_price : Ürün fiyat bilgileri ürün verisine eklenir. Üründe Fiyat Verisi

add_categories : Ürün kategori bilgileri ürün verisine eklenir. Üründe Kategori Verisi

is_sync : Ürün satış kanalına yollandığında durumu hemen mi ediniliyor Senkron veya Asenkron Satış Kanalı Süreç yoksa asenkron bir şekilde mi ediniliyor olduğudur.

class SendUpdatedProducts(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)

`get_data()`
Bu fonksiyon satış kanalına iletilmiş productlara ait omnitron'da yapılan güncellemeleri satış kanalına iletmek için atılacak istekte gönderilecek veri hazırlar. Response olarak liste içerinde productlar döndürülmesi gerekir.

`validated_data(data)`
Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalında güncellenecek productlar üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

`transform_data(data)`
Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

`send_request(transformed_data)`
Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

> **Dikkat:** Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

`normalize_response(data, validated_data, transformed_data, response)`
Bu fonksiyon update_product adımında productlarımızı satış kanalına güncellemek için kullanmış olduğumuz verileri toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceğ cevap doğrudan update_products fonksiyonunda kullanılacaktır.

Bu methoda süreç asenkron ise satış kanalından dönen remote_batch_id batch_request'e işlenmelidir.

```
>>> remote_batch_id = response.get("remote_batch_request_id")
>>> self.batch_request.remote_batch_id = remote_batch_id
>>> return "", report, data
```

> **Dikkat:** Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.
>
> **response_data**: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi ProductBatchRequestResponseDto olmak zorundadır.
>
> **report**: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.
>
> **data**: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.
>
> ```
> # örnek return
> return response_data, report, data
> ```

3. delete_products

Akinon'da satış tarafına daha önce insert edilmiş fakat silinmesi istenen ürünleri alır ve bu ürünleri *channel.commands.products.SendDeletedProducts*'da yer alan komut'a gönderir. Bu komut ile

satış kanalına mevcut olan ürünler silinir.

İstenildiği durumda aşağıda verilen parametre değeri ile Komutun çalışması zenginleştirilebilir.

is_sync : Ürünün silinme bilgisi satış kanalına yollandığında durumu hemen mi ediniliyor Senkron veya Asenkron Satış Kanalı Süreç yoksa asenkron bir şekilde mi ediniliyor olduğudur.

`class SendDeletedProducts(integration, objects=None, batch_request=None, **kwargs)`

`get_data()`

Bu fonksiyon satış kanalına iletilmiş productlara ait omnitron'da yapılan silinecek ürünleri satış kanalına iletmek için atılacak istekte gönderilecek veri hazırlar. Response olarak liste içerinde productlar döndürülmesi gerekir.

`validated_data(data)`

Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalında silinecek productlar üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

`transform_data(data)`

Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

`send_request(transformed_data)`

Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

> **Dikkat:** Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

`normalize_response(data, validated_data, transformed_data, response)`

Bu fonksiyon delete_product adımında productların silindiği bilgisinin satış kanalına iletmek için kullanmış olduğumuz verileri toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceğ cevap doğrudan delete_products fonksiyonunda kullanılacaktır.

Bu methoda süreç asenkron ise satış kanalından dönen remote_batch_id batch_request'e işlenmelidir.

```
>>> remote_batch_id = response.get("remote_batch_request_id")
>>> self.batch_request.remote_batch_id = remote_batch_id
>>> return "", report, data
```

> **Dikkat:** Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.
>
> **response_data**: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi ProductBatchRequestResponseDto olmak zorundadır.
>
> **report**: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.

data: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.

```
# örnek return
return response_data, report, data
```

4. check_delete_products

Akinon'da satış tarafına daha önce silinme isteği gönderilmiş fakat silinme işlemi asenkron olduğu için işlemin sonucu bilinmeyen ürünleri `channel.commands.products.CheckDeletedProducts`'da yer alan komut'a gönderir. Bu komut ile satış kanalına iletilmiş silme isteklerinin durumu öğrenilir.

class CheckDeletedProducts(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)

get_data()

Bu fonksiyon satış kanalına iletilmiş silinme isteklerinin durumunun öğrenilmesi için gerekli olan verileri hazırlar. Response olarak BatchRequest döndürülmesi gerekir.

validated_data(*data*)

Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalında silinmiş durumu sorgulanacak productlar üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

transform_data(*data*)

Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

send_request(*transformed_data*)

Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

Dikkat: Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

normalize_response(*data*, *validated_data*, *transformed_data*, *response*)

Bu fonksiyon check_delete_products adımında productların silinip silinmediği bilgisinin satış kanalından okumak için kullanmış olduğumuz verileri toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceğ cevap doğrudan delete_products fonksiyonunda kullanılacaktır.

Dikkat: Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.

response_data: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi ProductBatchRequestResponseDto olmak zorundadır.

report: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.

data: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.

```
# örnek return
return response_data, report, data
```

5. check_products

   Akinon'da satış tarafına daha önce oluşturma veya güncelleme isteği gönderilmiş fakat bu işlem asenkron olduğu için işlemin sonucu bilinmeyen ürünleri *channel.commands.products.CheckProducts*'da yer alan komut'a gönderir. Bu komut ile satış kanalına iletilmiş oluşturma veya güncelleme isteklerinin durumu öğrenilir.

   class CheckProducts(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)

   get_data()

   Bu fonksiyon satış kanalına iletilmiş oluşturma veya güncelleme isteklerinin durumunun öğrenilmesi için gerekli olan verileri hazırlar. Response olarak BatchRequest döndürülmesi gerekir.

   validated_data(*data*)

   Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalında oluşturma veya güncelleme durumu sorgulanacak BatchRequest üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

   transform_data(*data*)

   Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

   send_request(*transformed_data*)

   Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

   Dikkat: Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

   normalize_response(*data*, *validated_data*, *transformed_data*, *response*)

   Bu fonksiyon check_products adımında productların yaratılma veya güncellenme bilgisinin satış kanalından okumak için kullanmış olduğumuz verileri toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceğ cevap doğrudan get_product_batch_requests fonksiyonunda kullanılacaktır.

   Dikkat: Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.

   response_data: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi ProductBatchRequestResponseDto olmak zorundadır.

   report: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.

   data: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.

```
# örnek return
return response_data, report, data
```

Product Price

ProductPrice Data


```
{
    "pk": 2,
    "product": 913,
    "price": "62.44",
    "price_list": 1,
    "currency_type": "try",
    "tax_rate": "8.00",
    "retail_price": "249.75",
    "extra_field": {},
    "discount_percentage": "75.00",
    "modified_date": "2017-01-23T18:29:23.716095Z"
}
```


productprice yer alan verinin içerisinde;

product kısmında ürünün omnitrondaki pk vardır.

price satış fiyatı vardır

price_list kısmında akinondaki fiyat listesinin ID bilgisi vardır

currency_type kısmında fiyat bilgisinin birimi vardır.

tax_rate kısımında vergi oranı vardır.

retail_price kısmında ürünün mağaza fiyatı vardır.

discount_percentage kısmında ürünün üzerindeki indirim bilgisi vardır.

modified_date son güncelleme tarihi vardır.

Ürün entegrasyonu ile ilgili channel_app_template.app.tasks altında çalışan tasklar

1. insert_prices

   Akinon'da satış tarafına ilk defa insert edilmesi gereken ProductPriceları alır ve bu verileri

   channel.commands.product_prices.SendInsertedPrices'da yer alan komut'a gönderir. Bu komut ile satış kanalına ürünün fiyat bilgisi oluşturulur.

   İstenildiği durumda aşağıda listesi verilen parametre değerleri ile Fiyat Verisi zenginleştirilebilir.

   PriceService içerisinde yer alan insert_product_prices servisine ait parametreler ve açıklamaları aşağıdaki gibidir.

Dikkat: *Fiyat Servisi* içerisinde yer alan insert_product_prices servisine ait parametreler

add_stock : Ürün Fiyatına ürün stok verisi eklenir. *Ürün Fiyat Datasına Stock Datası Eklemek*

add_product_objects : Ürün Fiyatına ürün verisi eklenir. *Ürün Fiyat Datasına Ürün Datası Eklemek*

is_sync : Ürün Fiyatı satış kanalına yollandığında durumu hemen mi ediniliyor Senkron veya Asenkron Satış Kanalı Süreç yoksa asenkron bir şekilde mi ediniliyor olduğudur.

`class SendInsertedPrices(integration, objects=None, batch_request=None, **kwargs)`

`get_data()`

Bu fonksiyon ürünlerin fiyat bilgisini omnitron'dan satış kanalına iletmek için atılacak istekte gönderilecek veri hazırlanır. Response olarak liste içerinde ProductPrice döndürülmesi gerekir.

`validated_data(data)`

Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalına gönderilecek ürün fiyatları üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

`transform_data(data)`

Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

`send_request(transformed_data)`

Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

Dikkat: Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

`normalize_response(data, validated_data, transformed_data, response)`

Bu fonksiyon insert_prices adımında ürünlerimizin fiyatını satış kanalına iletmek için kullanmış olduğumuz verileri toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceği cevap doğrudan insert_product_prices fonksiyonunda kullanılacaktır.

Bu methoda süreç asenkron ise satış kanalından dönen remote_batch_id batch_request'e işlenmelidir.

```
>>> remote_batch_id = response.get("remote_batch_request_id")
>>> self.batch_request.remote_batch_id = remote_batch_id
>>> return "", report, data
```

Dikkat: Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.

response_data: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen

response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi BatchRequestResponseDto olmak zorundadır.

report: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.

data: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.

```
# örnek return
return response_data, report, data
```

2. update_prices

   Akinon'da satış tarafına güncellenmesi gereken ProductPriceları alır ve bu verileri

   channel.commands.product_prices.SendUpdatedPrices'da yer alan komut'a gönderir. Bu komut ile satış kanalında bulunan ürünün fiyat bilgisi güncellenir.

   İstenildiği durumda aşağıda listesi verilen parametre değerleri ile Fiyat Verisi zenginleştirilebilir.

   PriceService içerisinde yer alan update_product_prices servisine ait parametreler ve açıklamaları aşağıdaki gibidir.

Dikkat: *Fiyat Servisi* içerisinde yer alan insert_product_prices servisine ait parametreler

add_stock : Ürün Fiyatına ürün stok verisi eklenir. *Ürün Fiyat Datasına Stock Datası Eklemek*

add_product_objects : Ürün Fiyatına ürün verisi eklenir. *Ürün Fiyat Datasına Ürün Datası Eklemek*

is_sync : Ürün satış kanalına yollandığında durumu hemen mi ediniliyor Senkron veya Asenkron Satış Kanalı Süreç yoksa asenkron bir şekilde mi ediniliyor olduğudur.

`class SendUpdatedPrices(integration, objects=None, batch_request=None, **kwargs)`

`get_data()`

Bu fonksiyonda ürünlerin güncellenmiş fiyat bilgisini omnitron'dan satış kanalına iletmek için atılacak istekte gönderilecek veri hazırlanır. Response olarak liste içerinde ProductPrice döndürülmesi gerekir.

`validated_data(data)`

Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalına gönderilecek ürün fiyatları üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

`transform_data(data)`

Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

`send_request(transformed_data)`

Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

> **Dikkat:** Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

normalize_response(*data*, *validated_data*, *transformed_data*, *response*)

Bu fonksiyon update_prices adımında ürünlerimizin fiyatını satış kanalına iletmek için kullanmış olduğumuz verileri toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceği cevap doğrudan insert_product_prices fonksiyonunda kullanılacaktır.

Bu methoda süreç asenkron ise satış kanalından dönen remote_batch_id batch_request'e işlenmelidir.

```
>>> remote_batch_id = response.get("remote_batch_request_id")
>>> self.batch_request.remote_batch_id = remote_batch_id
>>> return "", report, data
```

> **Dikkat:** Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.
>
> **response_data**: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi BatchRequestResponseDto olmak zorundadır.
>
> **report**: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.
>
> **data**: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.
>
> ```
> # örnek return
> return response_data, report, data
> ```

3. check_prices

Akinon'da satış tarafına asenkron olarak güncellenmiş veya yaratılmış fakat durumu bilinmeyen BatchRequesti alır ve bu verileri

channel.commands.product_prices.CheckPrices'da yer alan komut'a gönderir. Bu komut ile satış kanalında bulunan ürünün fiyat bilgisinin yaratılma veya güncellenme durumunun öğrenilmesini sağlar.

İstenildiği durumda aşağıda listesi verilen parametre değerleri ile Fiyat Verisi zenginleştirilebilir.

PriceService içerisinde yer alan get_price_batch_requests fonksiyonu kullanılır.

class CheckPrices(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)

get_data()

Bu fonksiyonda ürünlerin satış kanalına iletilmiş fiyat bilgisinin durumunu öğrenmek için atılacak istekte gönderilecek veri hazırlanır. Response olarak liste içerinde BatchRequest döndürülmesi gerekir.

validated_data(*data*)

Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalına gönderilmiş ürün fiyatları verisi üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

transform_data(*data*)

Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri

göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

`send_request`(*transformed_data*)
Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

> **Dikkat:** Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

`normalize_response`(*data*, *validated_data*, *transformed_data*, *response*)
Bu fonksiyon get_price_batch_requests adımında ürünlerimizin fiyatının işlenme durumunu kontrol etmek için satış kanalına sorgu atarken kullanmış olduğumuz verileri ve dönen cevabı toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceği cevap doğrudan get_price_batch_requests fonksiyonunda kullanılacaktır.

> **Dikkat:** Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.
>
> **response_data**: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi BatchRequestResponseDto olmak zorundadır.
> **report**: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.
> **data**: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.
>
> ```
> # örnek return
> return response_data, report, data
> ```

Ürün Fiyat Datasına Stock Datası Eklemek

```
{
    "pk": 2,
    "product": 913,
    "price": "62.44",
    "price_list": 1,
    "currency_type": "try",
    "tax_rate": "8.00",
    "retail_price": "249.75",
    "extra_field": {},
    "discount_percentage": "75.00",
    "modified_date": "2017-01-23T18:29:23.716095Z",
    "productstock":{
            "pk": 1,
            "product": 2250,
            "stock": 46,
            "stock_list": 1,
```

(sonraki sayfaya devam)

(önceki sayfadan devam)

```
            "unit_type": "qty",
            "extra_field":{},
            "sold_quantity_unreported": 0,
            "modified_date": "2017-01-23T13:37:31.947171Z"
        }
}
```

Ürün Fiyat Datasına Ürün Datası Eklemek

```
{
    "pk": 2,
    "product": 913,
    "price": "62.44",
    "price_list": 1,
    "currency_type": "try",
    "tax_rate": "8.00",
    "retail_price": "249.75",
    "extra_field": {},
    "discount_percentage": "75.00",
    "modified_date": "2017-01-23T18:29:23.716095Z",
    "product":    {
        "pk": 12227,
        "name": "Kırmızı Tişört",
        "base_code": "1KBATC0231",
        "sku": "1KBATC0231001",
        "product_type": "0",
        "is_active": true,
        "parent": null,
        "attributes": {
            "boyut": "34X34",
            "renk": "001",
            "uretim_yeri": "Türkiye",
            "materyal": "%100 POLYESTER",
        },
        "productimage_set": [
            {
                "pk": 20044,
                "status": "active",
                "image": "http://localhost:8001/media/products/2021/10/17/12227/1bfe74b4-
↪175e-4c1a-80f2-b355feae498c.jpg"
            }
        ],
        "attribute_set": 2,
        "productization_date": "2017-01-23T16:40:58.578504Z"
    }
}
```

Product Stock

ProductStock Data

```
{
    "pk": 1,
    "product": 2250,
    "stock": 46,
    "stock_list": 1,
    "unit_type": "qty",
    "extra_field":{},
    "sold_quantity_unreported": 0,
    "modified_date": "2017-01-23T13:37:31.947171Z"
}
```

productstock yer alan verinin içerisinde;

product kısmında ürünün omnitrondaki pk vardır.

stock kısmında satılabilir stok miktarı vardır.

stock_list kısmında akinondaki stok listesinin ID bilgisi vardır

unit_type kısmında miktar bilgisinin birimi vardır.

sold_quantity_unreported kısımında akinondaki rezerve stok miktarı vardır.

modified_date son güncelleme tarihi vardır.

Ürün entegrasyonu ile ilgili channel_app_template.app.tasks altında çalışan tasklar

1. insert_stocks

   Akinon'da satış tarafına ilk defa insert edilmesi gereken ProductStockları alır ve bu verileri

   `channel.commands.product_stocks.SendInsertedStocks`'da yer alan komut'a gönderir. Bu komut ile satış kanalına ürünün stok bilgisi oluşturulur.

   İstenildiği durumda aşağıda listesi verilen parametre değerleri ile Stok Verisi zenginleştirilebilir.

   StockService içerisinde yer alan insert_product_stocks servisine ait parametreler ve açıklamaları aşağıdaki gibidir.

   Dikkat: *Stok Servisi* içerisinde yer alan insert_product_stocks servisine ait parametreler

   add_price : Ürün Stocklarına ürün fiyat verisi eklenir. *Ürün Stock Datasına Fiyat Datası Eklemek*

add_product_objects : Ürün Stocklarına ürün verisi eklenir. *Ürün Stock Datasına Ürün Datası Eklemek*

is_sync : Stok satış kanalına yollandığında durumu hemen mi ediniliyor Senkron veya Asenkron Satış Kanalı Süreç yoksa asenkron bir şekilde mi ediniliyor olduğudur.

class **SendInsertedStocks**(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)

get_data()

Bu fonksiyon ürünlerin stok bilgisini omnitron'dan satış kanalına iletmek için atılacak istekte gönderilecek veri hazırlanır. Response olarak liste içerinde ProductStock döndürülmesi gerekir.

validated_data(*data*)

Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalına gönderilecek ürün stokları üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

transform_data(*data*)

Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

send_request(*transformed_data*)

Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

> **Dikkat:** Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

normalize_response(*data*, *validated_data*, *transformed_data*, *response*)

Bu fonksiyon insert_stocks taskında ürünlerimizin stoklarını satış kanalına iletmek için kullanmış olduğumuz verileri toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceği cevap doğrudan insert_product_stocks fonksiyonunda kullanılacaktır.

Bu methoda süreç asenkron ise satış kanalından dönen remote_batch_id batch_request'e işlenmelidir.

```
>>> remote_batch_id = response.get("remote_batch_request_id")
>>> self.batch_request.remote_batch_id = remote_batch_id
>>> return "", report, data
```

> **Dikkat:** Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.
>
> **response_data**: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi BatchRequestResponseDto olmak zorundadır.
> **report**: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.
> **data**: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.

```
# örnek return
return response_data, report, data
```

2. update_stocks

   Akinon'da satış tarafına güncellenmesi gereken ProductStockları alır ve bu verileri

   channel.commands.product_stocks.SendUpdatedStocks'da yer alan komut'a gönderir. Bu komut ile satış kanalında bulunan ürünün stok bilgisi güncellenir.

   İstenildiği durumda aşağıda listesi verilen parametre değerleri ile Stok Verisi zenginleştirilebilir.

   StockService içerisinde yer alan update_product_stocks servisine ait parametreler ve açıklamaları aşağıdaki gibidir.

> **Dikkat:** *Stok Servisi* içerisinde yer alan insert_product_stocks servisine ait parametreler
>
> **add_price** : Ürün Stocklarına ürün fiyat verisi eklenir. *Ürün Stock Datasına Fiyat Datası Eklemek*
>
> **add_product_objects** : Ürün Stocklarına ürün verisi eklenir. *Ürün Stock Datasına Ürün Datası Eklemek*
>
> **is_sync** : Stok satış kanalına yollandığında durumu hemen mi ediniliyor Senkron veya Asenkron Satış Kanalı Süreç yoksa asenkron bir şekilde mi ediniliyor olduğudur.

class SendUpdatedStocks(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)

get_data()

Bu fonksiyonda ürünlerin güncellenmiş stok bilgisini omnitron'dan satış kanalına iletmek için atılacak istekte gönderilecek veri hazırlanır. Response olarak liste içerinde ProductStock döndürülmesi gerekir.

validated_data(*data*)

Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalına gönderilecek ürün stokları üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

transform_data(*data*)

Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

send_request(*transformed_data*)

Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

> **Dikkat:** Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

normalize_response(*data*, *validated_data*, *transformed_data*, *response*)

Bu fonksiyon update_prices adımında ürünlerimizin stoklarını satış kanalına iletmek için kullanmış olduğumuz verileri toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceği cevap doğrudan update_product_stocks fonksiyonunda kullanılacaktır.

Bu methoda süreç asenkron ise satış kanalından dönen remote_batch_id batch_request'e işlenmelidir.

```
>>> remote_batch_id = response.get("remote_batch_request_id")
>>> self.batch_request.remote_batch_id = remote_batch_id
>>> return "", report, data
```

Dikkat: Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.

response_data: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi BatchRequestResponseDto olmak zorundadır.

report: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.

data: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.

```
# örnek return
return response_data, report, data
```

3. check_stocks

Akinon'da satış tarafına asenkron olarak güncellenmiş veya yaratılmış fakat durumu bilinmeyen BatchRequesti alır ve bu verileri

channel.commands.product_stocks.CheckStocks'da yer alan komut'a gönderir. Bu komut ile satış kanalında bulunan ürünün stok bilgisinin yaratılma veya güncellenme durumunun öğrenilmesini sağlar.

StockService içerisinde yer alan get_stock_batch_requests fonksiyonu kullanılır.

class CheckStocks(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)

get_data()

Bu fonksiyonda ürünlerin satış kanalına iletilmiş stok bilgisinin durumunu öğrenmek için atılacak istekte gönderilecek veri hazırlanır. Response olarak liste içerinde BatchRequest döndürülmesi gerekir.

validated_data(*data*)

Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalına gönderilmiş ürün stokları verisi üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

transform_data(*data*)

Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

send_request(*transformed_data*)

Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

> **Dikkat:** Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

normalize_response(*data*, *validated_data*, *transformed_data*, *response*)

Bu fonksiyon check_stocks taskında ürünlerimizin stoklarının işlenme durumunu kontrol etmek için satış kanalına sorgu atarken kullanmış olduğumuz verileri ve dönen cevabı toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceği cevap doğrudan get_stock_batch_requests fonksiyonunda kullanılacaktır.

> **Dikkat:** Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.
>
> **response_data**: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi BatchRequestResponseDto olmak zorundadır.
>
> **report**: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.
>
> **data**: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.

```
# örnek return
return response_data, report, data
```

Ürün Stock Datasına Fiyat Datası Eklemek

```
{
    "pk": 1,
    "product": 2250,
    "stock": 46,
    "stock_list": 1,
    "unit_type": "qty",
    "extra_field":{},
    "sold_quantity_unreported": 0,
    "modified_date": "2017-01-23T13:37:31.947171Z",
    "productprice":    {
        "pk": 2,
        "product": 913,
        "price": "62.44",
        "price_list": 1,
        "currency_type": "try",
        "tax_rate": "8.00",
        "retail_price": "249.75",
        "extra_field": {},
        "discount_percentage": "75.00",
        "modified_date": "2017-01-23T18:29:23.716095Z"
    }
}
```

Ürün Stock Datasına Ürün Datası Eklemek


```
{
    "pk": 1,
    "product": 2250,
    "stock": 46,
    "stock_list": 1,
    "unit_type": "qty",
    "extra_field":{},
    "sold_quantity_unreported": 0,
    "modified_date": "2017-01-23T13:37:31.947171Z",
    "product": {
        "pk": 12227,
        "name": "Kırmızı Tişört",
        "base_code": "1KBATC0231",
        "sku": "1KBATC0231001",
        "product_type": "0",
        "is_active": true,
        "parent": null,
        "attributes": {
            "boyut": "34X34",
            "renk": "001",
            "uretim_yeri": "Türkiye",
            "materyal": "%100 POLYESTER",
        },
        "productimage_set": [
            {
                "pk": 20044,
                "status": "active",
                "image": "http://localhost:8001/media/products/2021/10/17/12227/1bfe74b4-
↪175e-4c1a-80f2-b355feae498c.jpg"
            }
        ],
        "attribute_set": 2,
        "productization_date": "2017-01-23T16:40:58.578504Z"
    }
}
```


Product Image

Ürün entegrasyonu ile ilgili channel_app_template.app.tasks altında çalışan tasklar

1. insert_images

   Akinon'da satış tarafına ilk defa insert edilmesi gereken ProductImage'ları alır ve bu verileri

   channel.commands.product_images.SendInsertedImages'da yer alan komut'a gönderir. Bu komut ile satış kanalına ürünün resim bilgisi oluşturulur.

   İstenildiği durumda aşağıda listesi verilen parametre değerleri ile Resim Verisi zenginleştirilebilir.

   ImageService içerisinde yer alan insert_product_images servisine ait parametreler ve açıklamaları aşağıdaki gibidir.

Dikkat: *Resim Servisi* içerisinde yer alan insert_product_images servisine ait parametreler

is_sync : Ürün satış kanalına yollandığında durumu hemen mi ediniliyor Senkron veya Asenkron Satış Kanalı Süreç yoksa asenkron bir şekilde mi ediniliyor olduğudur.

class SendInsertedImages(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)

get_data()

Bu fonksiyon ürünlerin resim bilgisini omnitron'dan satış kanalına iletmek için atılacak istekte gönderilecek veri hazırlanır. Response olarak liste içerinde ProductImage döndürülmesi gerekir.

validated_data(*data*)

Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalına gönderilecek ürün resimleri üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

transform_data(*data*)

Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

send_request(*transformed_data*)

Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

Dikkat: Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

normalize_response(*data*, *validated_data*, *transformed_data*, *response*)

Bu fonksiyon insert_images adımında ürünlerimizin resmini satış kanalına iletmek için kullanmış olduğumuz verileri toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceği cevap doğrudan insert_product_images fonksiyonunda kullanılacaktır.

Bu methoda süreç asenkron ise satış kanalından dönen remote_batch_id batch_request'e işlenmelidir.

```
>>> remote_batch_id = response.get("remote_batch_request_id")
>>> self.batch_request.remote_batch_id = remote_batch_id
>>> return "", report, data
```

Dikkat: Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.

response_data: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi BatchRequestResponseDto olmak zorundadır.

report: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.

data: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.

```
# örnek return
return response_data, report, data
```

2. update_images

   Akinon'da satış tarafına güncellenmesi gereken ProductImage'ları alır ve bu verileri

   channel.commands.product_images.SendUpdatedImages'da yer alan komut'a gönderir. Bu komut ile satış kanalında bulunan ürünün resim bilgisi güncellenir.

   İstenildiği durumda aşağıda listesi verilen parametre değerleri ile Resim Verisi zenginleştirilebilir.

   ImageService içerisinde yer alan update_product_images servisine ait parametreler ve açıklamaları aşağıdaki gibidir.

   Dikkat: *Resim Servisi* içerisinde yer alan insert_product_images servisine ait parametreler

   is_sync : Ürün satış kanalına yollandığında durumu hemen mi ediniliyor Senkron veya Asenkron Satış Kanalı Süreç yoksa asenkron bir şekilde mi ediniliyor olduğudur.

   class SendUpdatedImages(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)

   get_data()

   Bu fonksiyonda ürünlerin güncellenmiş resim bilgisini omnitron'dan satış kanalına iletmek için atılacak istekte gönderilecek veri hazırlanır. Response olarak liste içerinde ProductImage döndürülmesi gerekir.

   validated_data(*data*)

   Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalına gönderilecek ürün resimleri üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

   transform_data(*data*)

   Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

   send_request(*transformed_data*)

   Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

   Dikkat: Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

   normalize_response(*data*, *validated_data*, *transformed_data*, *response*)

   Bu fonksiyon update_images adımında ürünlerimizin resmini satış kanalına iletmek için kullanmış olduğumuz verileri toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceği cevap doğrudan insert_product_images fonksiyonunda kullanılacaktır.

   Bu methoda süreç asenkron ise satış kanalından dönen remote_batch_id batch_request'e işlenmelidir.

```
>>> remote_batch_id = response.get("remote_batch_request_id")
>>> self.batch_request.remote_batch_id = remote_batch_id
>>> return "", report, data
```

Dikkat: Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.

response_data: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi BatchRequestResponseDto olmak zorundadır.

report: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.

data: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.

```
# örnek return
return response_data, report, data
```

3. check_images

   Akinon'da satış tarafına asenkron olarak güncellenmiş veya yaratılmış fakat durumu bilinmeyen BatchRequesti alır ve bu verileri

   channel.commands.product_images.CheckImages'da yer alan komut'a gönderir. Bu komut ile satış kanalında bulunan ürünün resim bilgisinin yaratılma veya güncellenme durumunun öğrenilmesini sağlar.

   İstenildiği durumda aşağıda listesi verilen parametre değerleri ile Resim Verisi zenginleştirilebilir.

   ImageService içerisinde yer alan get_image_batch_requests fonksiyonu kullanılır.

   class CheckImages(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)

   get_data()

   Bu fonksiyonda ürünlerin satış kanalına iletilmiş resim bilgisinin durumunu öğrenmek için atılacak istekte gönderilecek veri hazırlanır. Response olarak liste içerinde BatchRequest döndürülmesi gerekir.

   validated_data(*data*)

   Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalına gönderilmiş ürün resimleri verisi üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

   transform_data(*data*)

   Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

   send_request(*transformed_data*)

   Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

   Dikkat: Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

normalize_response(*data*, *validated_data*, *transformed_data*, *response*)

Bu fonksiyon get_image_batch_requests adımında ürünlerimizin resminin işlenme durumunu kontrol etmek için satış kanalına sorgu atarken kullanmış olduğumuz verileri ve dönen cevabı toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceği cevap doğrudan get_image_batch_requests fonksiyonunda kullanılacaktır.

Dikkat: Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.

response_data: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi BatchRequestResponseDto olmak zorundadır.

report: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.

data: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.

```
# örnek return
return response_data, report, data
```

Orders

Sipariş entegrasyonu ile ilgili channel_app_template.app.tasks altında çalışan tasklar

1. fetch_orders

   Satış kanalı tarafından yeni oluşmuş siparişleri almak ve akinonda işlemek için *channel.commands.orders.orders.GetOrders* yer alan komut çalıştırılır ve Akinon'a istenilen formatta veri sağlar. Bu komut ile satış kanalına oluşmuş siparişler okunup Akinon uzerinde oluşturulması sağlanır.

   OrderService içerisinde yer alan fetch_and_create_order fonksiyonu ile süreç işletilir.

   Siparişleri satış kanalından almak ve istenlen formata çevirmek için aşağıda listesi verilen açıklamalara göre bu command hazırlanmalıdır.

   class GetOrders(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)

   get_data()

   Bu fonksiyonda satış kanalı üzerinde oluşmuş siparişlere ulaşmak için verilerin hazırladığı yerdir. Herhangi bir parametre almaz.

   validated_data(*data*)

   Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalından siparişleri okumak için hazırlanan veri üzerinde bir doğrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

   transform_data(*data*)

   Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalından sipariş okumadan önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

   send_request(*transformed_data*)

   Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

> **Dikkat:** Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

normalize_response(*data*, *validated_data*, *transformed_data*, *response*)

Bu fonksiyon fetch_orders taskında satış kanalında oluşmuş siparişlerimizi okumak için hazırladığımız verileri ve satış kanalından gelen cevabı toplayıp Akınında siparişleri yaratmak için son haline getirdiğimiz yerdir. Bu fonksiyonun döneceği cevap doğrudan fetch_and_create_order fonksiyonundaki süreç ile işlenir.

> **Dikkat:** Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır. BU METHOD GENERATOR tipinde donmelidir.
>
> **response_data**: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi ChannelCreateOrderDto olmak zorundadır.
>
> **report**: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.
>
> **data**: None

```
# örnek generator
yield response_data, report, None
```

2. update_orders

Akinon'dan satış kanalı tarafına güncellenmesi gereken **siparişleri** alır ve bu verileri *channel.commands.orders.orders.SendUpdatedOrders*'da yer alan komut'a gönderir. Bu komut ile satış kanalına güncellenmiş sipariş bilgilerini iletir.

OrderService içerisinde yer alan update_orders fonksiyonuna ait süreçte kullanılacak olan bu command ile Akinon'da durumu güncellenmiş sipariş bilgileri bu command ile satış kanalına gönderilebillir.

> **Dikkat:** Sipariş Servisi içerisinde yer alan update_orders fonksiyonuna ait parametreler
>
> **is_sync** : Sipariş üzerindeki güncelleme satış kanalına yollandığında durumu hemen mi ediniliyor Senkron veya Asenkron Satış Kanalı Süreç yoksa asenkron bir şekilde mi ediniliyor olduğudur.

class SendUpdatedOrders(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)

get_data()

Bu fonksiyon siparişlerin güncellenme bilgisini omnitron'dan satış kanalına iletmek için atılacak istekte gönderilecek veri hazırlanır. Response olarak liste içerinde Order döndürülmesi gerekir. objects datası içinde sipariş listesi mevcuttur.

validated_data(*data*)

Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalına gönderilecek güncellenmiş siparişler verisi üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

transform_data(*data*)

Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri

göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

send_request(*transformed_data*)
Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

> **Dikkat:** Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

normalize_response(*data*, *validated_data*, *transformed_data*, *response*)
Bu fonksiyon update_orders taskında güncellenmiş siparişlerimizi satış kanalına iletmek için kullanmış olduğumuz verileri ve satış kanalından aldığımız cevabı toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceği cevap doğrudan update_orders fonksiyonunda kullanılacaktır.

Bu methoda süreç asenkron ise satış kanalından dönen remote_batch_id batch_request'e işlenmelidir.

```
>>> remote_batch_id = response.get("remote_batch_request_id")
>>> self.batch_request.remote_batch_id = remote_batch_id
>>> return "", report, data
```

> **Dikkat:** Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.
>
> **response_data**: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi OrderBatchRequestResponseDto olmak zorundadır.
> **report**: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.
> **data**: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.
>
> ```
> # örnek return
> return response_data, report, data
> ```

3. check_orders

Akinon'da satış kanalı tarafına asenkron olarak iletilen Order güncellemelerinin BatchRequestlerini alır ve bu verileri *channel.commands.orders.orders.CheckOrders*'da yer alan komut'a gönderir. Bu komut ile satış kanalından güncelleme isteği atılmuş order'in güncel durum bilgisi satış kanalından okunur.

class CheckOrders(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)
get_data()
Bu fonksiyon güncelleme isteği satış kanalına iletilmiş orderların bilgisini Akinon'dan satış kanalı üzerinden durumunu sorgulamak için atılacak istekte gönderilecek veri hazırlanır. Response olarak liste içerinde BatchRequest döndürülmesi gerekir.

validated_data(*data*)
Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalından sorgulana-

cak order güncellemesi isteği verisi üzerinde bir değrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

`transform_data`(*data*)
Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalına veri göndermeden önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

`send_request`(*transformed_data*)
Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

> **Dikkat:** Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

`normalize_response`(*data*, *validated_data*, *transformed_data*, *response*)
Bu fonksiyon check_orders taskında daha önce güncellenmiş orderlarımızın durum sorgularını satış kanalına iletmek için kullanmış olduğumuz verileri ve satış kanalından dönen cevabı toplayıp son haline getirdiğimiz yerdir. Bu fonksiyonun döneceği cevap doğrudan get_order_batch_requests fonksiyonunda kullanılacaktır.

> **Dikkat:** Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır.
>
> **response_data**: Satış kanalından dönen verinin işlenmiş halidir. Tipi string veya liste olabilir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen veri liste tipinde ve içerisindeki elemanların tipi OrderBatchRequestResponseDto olmak zorundadır.
> **report**: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.
> **data**: Fonksiyonumuzun aldığı ilk parametre, get_data fonksiyonundan aldığımız cevap.
>
> ```
> # örnek return
> return response_data, report, data
> ```

4. fetch_and_create_cancel

Satış kanalı tarafında yeni oluşmuş iptal sipariş kayıtları *channel.commands.orders.orders.GetCancelledOrders* komutu aracılığı ile alınır ve akinon'da gönderilir. OrderService içerisinde yer alan fetch_and_create_cancel fonksiyonu ile kullanılır.

Aşağıda listesi verilen parametre değerleri ile Sipariş İptal Verisi çekilip istenilen formata getirilir.

`class GetCancelledOrders`(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)
`get_data`()
Bu fonksiyonda satış kanalı üzerinde oluşmuş siparişleri Akinona yazmak için satış kanalına atılacak istekte gönderilecek veri hazırlanır. Parametre almaz.

`validated_data`(*data*)
Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalından iptal siparişleri okumak için hazırlanan veri üzerinde bir doğrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

transform_data(*data*)
Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalından iptal sipariş kayıtları okumadan önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

send_request(*transformed_data*)
Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

> **Dikkat:** Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

normalize_response(*data*, *validated_data*, *transformed_data*, *response*)
Bu fonksiyon fetch_and_create_cancel taskında satış kanalında oluşmuş iptal siparişlerimizi okumak için hazırladığımız verileri ve satış kanalından gelen cevabı toplayıp Akınında siparişleri iptal etmek için son haline getirdiğimiz yerdir. Bu fonksiyonun döneceği cevap doğrudan fetch_and_create_cancel fonksiyonunda kullanılacaktır.

> **Dikkat:** Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır. BU METHOD GENERATOR tipinde donmelidir.
>
> **response_data**: Satış kanalından dönen verinin işlenmiş halidir. Dönen cevapda kullanılacak bir veri yok ise boş string dönülmesi yeterlidir. Dönen response kullanılacak ise dönen verinin tipi CancelOrderDto olmak zorundadır.
> **report**: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.
> **data**: None
>
> ```
> # örnek generator dönüş tipi
> yield response_data, report, None
> ```

5. fetch_and_update_order_items

   Satış kanalında güncellenmiş siparişleri almak ve Omnitron'a OrderItem bazında aktarmak için *channel.commands.orders.orders.GetUpdatedOrderItems* yer alan komut çalıştırılır ve Akinon'a istenilen formatta veri sağlar. (ChannelUpdateOrderItemDto). Bu komut ile satış kanalına güncellenmiş siparişlerin okunup Omnitron'a aktarılması sağlanır.

   OrderService içerisinde yer alan fetch_and_update_order_items fonksiyonu ile süreç işletilir.

   Güncellenmiş siparişleri satış kanalından almak ve istenilen formata çevirmek için aşağıda listesi verilen açıklamalara göre bu command hazırlanmalıdır.

   class GetUpdatedOrderItems(*integration*, *objects=None*, *batch_request=None*, ***kwargs*)
   get_data()
   Bu fonksiyonda satış kanalı üzerinde güncellenmiş siparişlere ulaşmak için verilerin hazırladığı yerdir. Herhangi bir parametre almaz.

   validated_data(*data*)
   Parametre olarak get_data fonksiyonunun döndüğü cevabı alır. Eğer satış kanalından siparişleri okumak için hazırlanan veri üzerinde bir doğrulama yapılması gerekiyor ise kullanılır. Doğrulama yapılmayacak ise parametre olarak verilen data'nın döndürülmesi gerekir.

`transform_data`(*data*)
Parametre olarak validated_data fonksiyonunun döndürdüğü cevabı alır. Eğer satış kanalından sipariş okumadan önce veri üzerinde değişiklik yapılması gerekiyor ise kullanılır. Cevap olarak iletilmek istenen verinin son halini döndürür.

`send_request`(*transformed_data*)
Parametre olarak transform_data fonksiyonunun döndürdüğü cevabı alır. Bu fonksiyon aldığı veriyi satış kanalının ilgili uç noktasına isteğin atılacağı yerdir. Cevap olarak response veya response ile gelen veriyi dönmesi gerekir.

> **Dikkat:** Bu kısımda dönülecek cevap normalize_response fonksiyonuna iletileceği için veri döndürürken veri tipleri konusunda dikkat etmek gerekmektedir.

`normalize_response`(*data*, *validated_data*, *transformed_data*, *response*)
Bu fonksiyon fetch_orders taskında satış kanalında oluşmuş siparişlerimizi okumak için hazırladığımız verileri ve satış kanalından gelen cevabı toplayıp Akınında siparişleri yaratmak için son haline getirdiğimiz yerdir. Bu fonksiyonun döneceği cevap doğrudan fetch_and_update_order_items fonksiyonundaki süreç ile işlenir.

> **Dikkat:** Bu kısımda dönülecek cevap 3 parçadan oluşmalıdır. BU METHOD GENERATOR tipinde donmelidir.
>
> **response_data**: Satış kanalından dönen verinin işlenmiş halidir. Dönen veri liste tipinde ve içerisindeki elemanların tipi ChannelUpdateOrderItemDto olmak zorundadır.
> **report**: Satış kanalından dönen cevabı işlerken oluşturduğumuz hata raporlarıdır.
> **data**: None
>
> ```
> # örnek generator
> yield response_data, report, None
> ```

2.2.4 Akinon'a Yeni Komut Eklemek

Uygulama içerisindeki akinon klasör içerisinde Akinon'nun omnitron ürününe ait ihtiyaç duyduğu servisler ile haberleşen süreçler kodlanmıştır. Geliştirmenin yapılacağı satış kanalı için omnitron servislerinde veya süreçlerinde bir eksiklik duyulması halinde buradaki geliştirmeye hazır yapıdan faydalanılır.

Mevcut süreçlerin ve servislerin ihtiyacı karşılamaması durumunda `channel_app_template.akinon.integration.OmnitronIntegration` kısmına yeni yazmış olduğunuz command'larınızı aşağıdaki adımları takip ederek ekleyebilirsiniz.

1. Komut oluşturmak

1. Komut oluşturmak için yaratacağınız *class OmnitronCommandInterface* i miras almalıdır.
2. Komutun istek atacağı uç nokta için *OmnitronApiEndpoint* i miras alan bir *class* oluşturmak ve yaratılan komutun *endpoint* özelliğine atamak gerekmektedir.
3. Opsiyonel olarak yarattığınız *OmnitronApiEndpoint* üzerinden birden farklı istek atmak isterseniz yaratılan komuta *path* özelliğini girip gerekli özelleştirme yapılabilir.

4. Opsiyonel olarak komutun bir seferde işleyeceği veri sayısını belirtmek için *BATCH_SIZE* isminde bir özellik tanımlayıp gerekli özelleştirme yapılabilir.
5. Komutun işleyeceği veri tipini tutmak ve olası bir hata durumunda daha detaylı log oluşturmak için *content_type* özelliği kullanılır. Atanabilecek content_type lara bakmak için *channel_app.omnitron.constants.ContentType* kontrol edebilirsiniz.
2. Oluşturduğumuz komutu *OmnitronIntegration* içerisinde ki *new_actions* özelliğine atayacağımız bir key(isim) ile birlikte eklenmeli.
3. Komutu çağırmak için `channel_app.channel.integration.ChannelIntegration.do_action` örneğinden faydalanılabilir.

2.3 Mimari

Proje, Python 3.8'de herhangi bir web servis kütüphanesi kullanılmadan Celery üzerinde belirli aralıklarla çalışan tasklar üzerinden asenkron olarak çalışacak şekilde tasarlanmıştır.

Celery broker olarak Redis'i kullanıyor, veritabanı bulunmuyor ve bunun dışında minimal durum bilgisi barındırdığı için (stateless) yatay ölçeklenme konusunda esnek bir altyapı sunuyor. Şu ana kadar olan geliştirmelerde Redis üzerinde de broker'ın kendi kullanımı dışında cache amaçlı olarak herhangi bir veri tutulmadı. Durum bilgisi olarak sayılacabilecek ortam değişkenleri, ACC üzerinde uygulama kurulduğu zaman besleniyor ve bazı nadir durumlarda güncelleme alıyor. Bu senaryoda uygulama yeniden başlatılarak, uygulamanın güncellemeleri alması gerekiyor.

Mimari, genel olarak 3 temel bloktan oluşuyor: Omnitron Entegrasyonu, Satış Kanalı Entegrasyonu ve de tasklar.

2.3.1 Entegrasyon

Entegrasyon kısımları ortak bir ebeveyn sınıf üzerinden Komut (Command) tasarım örüntüsünü kullanarak tanımlandı. Gerekli olan akışlar için OmnitronIntegration ve ChannelIntegration sınıflarında varsayılan komutlar geliştirildi.

Komutlar task metotlarının tanımlandığı noktada birbiri ardına çağrılmaktadır. Birinin çıktısı, sonrakinin girdisi olduğu için uygun formatta çıktı üretip girdi almaktadırlar. Bu sebeple tanımlanan arayüzlere sadık kalınması şiddetle tavsiye edilmektedir.

class BaseIntegration

To integrate with any system you must create a class which inherits from BaseIntegration. This class was designed to work with *command design pattern* which basically defines a task procedure interface. All defined commands override some of the default base methods according to their requirements.

do_action(*key: str*, ***kwargs*) → Any

Runs the command given with the key and supplies the additional parameters to the command.

Parametreler

- **key** – Command key
- **kwargs** – Any additional parameters can be specified, for example *objects* must be supplied if you want to provide input to the action.

Dönüşler Result of the command

do_action_async_run(*key: str*, ***kwargs*) → Any

Runs the command given with the key asynchronously and supplies the additional parameters to the command.

Parametreler

- **key** – Command key

- **kwargs** – Any additional parameters can be specified, for example *objects* must be supplied if you want to provide input to the action.

Dönüşler Result of the command

property catalog: omnisdk.omnitron.models.Catalog
Retrieves the catalog object using the *catalog_id* stored in the *self*.

Side effect: It stores the result in the *self.catalog_object*, if catalog is updated on the currently running task you must delete self.catalog_object and re-call this method

property channel: omnisdk.omnitron.models.Channel
Retrieves the channel object using the *channel_id* stored in the *self*.

Side effect: It stores the result in the *self.channel_object*, if channel is updated on the currently running task you must delete self.channel_object and re-call this method

Omnitron Entegrasyonu

Omnitron entegrasyonu, Channel App Template'ın, Omnitron servislerini çağırıp, CRUD işlemlerini, farklı servislerden veri toplamayı ve verilerin uygun formata dönüşümünü yaptığı sınıftır.

Burada farklı amaçlar için tanımlanmış komutlar bulunmaktadır. Örneğin; ürün oluşturma, ürün silme ve stok güncelleme bunlardan birkaçı. Komutların tamamının listesi için referans dokümanını inceleyebilirsiniz.

OmnitronIntegration sınıfı altındaki tüm komutlar, standart bir arayüz sunması ve de temel olarak kullanıldığı her projede tekrar tekrar yazılmasının önüne geçmek adına, girdi ve çıktı olarak DTO(Data Transfer Object) sınıflarını kullanmaktadırlar. Böylece farklı pazaralanları için geliştirilen projelerde aynı veri formatına dönüştürüldüğü sürece komutlar çalışmaya devam edecektir.

İdeal bir senaryoda OmnitronIntegration sınıfını türetmeye gerek olmayacak ve sınıf doğrudan kullanılabilecektir. Omnitron ve Channel App Template arasındaki iletişimde de değişen noktalar olabileceği gibi, modeller ve yapı çoğunlukla sabit kalacağından, A ve B pazaralanları için farklı geliştirme yapılması ihtimali düşüktür.

class OmnitronIntegration(*create_batch=True*, *content_type=None*)
Communicates with the Omnitron Api services through the commands defined. It manages OmnitronApiClient object on enter and exit methods.

__init__(*create_batch=True*, *content_type=None*)
Some environment parameters are stored in the integration object for convenience.

Parametreler **create_batch** – Flag to decide whether a batch request to be created

Satış Kanalı Entegrasyonu

Satış kanalı entegrasyonu, Channel App Template'ın, satış yapmak istediği pazaralanının servislerini çağırıp, CRUD işlemlerini, farklı servislerden veri toplamayı ve verilerin uygun formata dönüşümünü yaptığı sınıftır.

Satış kanalı servisleri ile bağlantı kurmak için istemci sınıfı yazılacaksa ya da requests kütüphanesi üzerinden herhangi bir sarmalayıcı bir yapı kullanmadan istekler atılacaksa, gerekli nesnelerin ve ayar değişkenlerinin bu sınıfın *__init__* metodunda tanımlanması önerilir.

ChannelIntegration sınıfındaki komutlar *__mocked_request* adında mock veri ile çalışan varsayılan bir metot barındırıyor. Bunlar taslak olarak kullanılan metotların baştan sona çalışması için hazırlandı. Her pazaralanı için farklı bağlantı ve servisler bulunacağından dolayı ortak bir çözüm uygulanması teknik olarak mümkün değil. Channel App geliştiricileri buradaki komutları türetmeli ve *send* metodunu ezerek komutu tamamlamalı. Yeni *send* metodunda pazaralanı servislerine istek atıp verileri de Omnitron komutlarının beklediği DTO nesnelerine dönüştürmeli.

class ChannelIntegration

Communicates with the Channel Api services through the commands defined.

If an Api Client class is developed, initialization and deletion should be handled in ChannelIntegration class so that commands have easier access to the api object.

Komut Arayüzü

Komut arayüzü, yapılacak işlemler için standart metotlar belirleyen bir tasarım örüntüsüdür. Çalıştır, gönder, getir gibi varsayılan metotların farklı komutlarda ihtiyaca göre değişen kısımlarının ezilmesiyle minimal değişikliklerle farklı komutlar geliştirilir. Böylece hata yönetimi, genel akış, ekstra modüllerin statü yönetimleri gibi diğer gereksinimler her bir komut için tekrar tanımlanmaz.

class CommandInterface

get_data() → object

This method fetches the input data for the command.

transform_data(*data*) → object

This method can be used to format the input data before it is executed on the run method.

validated_data(*data*) → dict

If the input data needs to satisfy some conditions or contain required a parameter, the validation is done here.

send(*validated_data*) → object

If the command sends a request using input data to achieve the main object of the command, it is recommended to place those operations in this method.

Parametreler **validated_data** –

Tasklar

Tasklar, farklı komutların birbiri ardına çalıştırılmasıyla bir iş akışını tamamlayan düzenli aralıklarla çalışan metotları ifade eder. Uygulamanın giriş noktalarıdır. Komutlar kendi başlarına çağrılmaz. Bir task içerisinde sadece o komutun çalışacağı şekilde tanımlanabilir. Tasklar, Celery üzerindeki tanımlı programa göre düzenli olarak çalıştırılır ya da manuel olarak Flower üzerinden tetiklenebilir. Tasklar akışları oluşturan birimlerdir. Bu konu hakkında daha detaylı bilgi için Akışlar bölümünü inceleyebilirsiniz.

2.3.2 Servisler(Flowlar)

Servisler, tasklar aracılığıyla Akinon ile Satış Kanalı Entegrasyonu arasındaki iletişimi sağlayan kısımdır. Akinondan verinin okunması/yazılması ve Satış Kanalı ile iletişime geçilmesi olmak üzere 2 temel adımdan oluşur.

Module	Açıklama
Kurulum Servisi	Kurulum aşamasına ait komutlar
Ürün Servisi	Ürün ile ilgili komutlar
Fiyat Servisi	Fiyat ile ilgili komutlar
Stok Servisi	Stok ile ilgili komutlar
Resim Servisi	Resim ile ilgili komutlar
Sipariş Servisi	Sipariş ile ilgili komutlar

Kurulum Servisi

Ürün Servisi

Ürün servisi üzerinde ürünlerin oluşturma, güncelleme ve silme aksiyonlarını üzerinde barındırır. Bu aksiyonları gerçekleştirmek için OmnitronIntegration ve ChannelIntegration entegrasyolanrından yardım alır.

class ProductService(*object*)

insert_products(*self*, *add_mapped=True*, *add_stock=True*, *add_price=True*, *add_categories=True*, *is_sync=True*, *is_success_log=True*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına eklenebilecek ürünleri çeker. Girilen parametrelere göre, *add_mapped* için ürün mapping bilgilerini, *add_stock* için ürünlerin stoklarını, *add_price* için ürün fiyat bilgilerini, *add_categories* için ürün kategori bilgilerini satış kanalına iletmek için Akinon üzerinden çeker. Sonrasında Satış Kanalına *send_inserted_products* komutu aracılığıyla iletilir. *is_sync* parametresinin aldığı değere göre satış kanalı ile kurulacak iletişimin senkron mu asenkron mu olacağına karar verir. Asenkron olacak ise **batch_service** üzerinden gerkli kayıtlar oluşturulur. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

update_products(*self*, *add_mapped=True*, *add_stock=True*, *add_price=True*, *add_categories=True*, *is_sync=True*, *is_success_log=True*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına güncellemesi iletilebilecek ürünleri çeker. Girilen parametrelere göre, *add_mapped* için ürün mapping bilgilerini, *add_stock* için ürünlerin stoklarını, *add_price* için ürün fiyat bilgilerini, *add_categories* için ürün kategori bilgilerini satış kanalına iletmek için Akinon üzerinden çeker. Sonrasında Satış Kanalına *send_updated_products* komutu aracılığıyla iletilir. *is_sync* parametresinin aldığı değere göre satış kanalı ile kurulacak iletişimin senkron mu asenkron mu olacağına karar verir. Asenkron olacak ise **batch_service** üzerinden gerkli kayıtlar oluşturulur. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

delete_products(*self*, *is_sync=True*, *is_content_object=True*, *is_success_log=True*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalından çıkartılabilecek ürünleri çeker. Sonrasında Satış Kanalına *send_deleted_products* komutu aracılığıyla iletilir. *is_sync* parametresinin aldığı değere göre satış kanalı ile kurulacak iletişimin senkron mu asenkron mu olacağına karar verir. Asenkron olacak ise **batch_service** üzerinden gerkli kayıtlar oluşturulur. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

get_delete_product_batch_requests(*self*, *is_success_log=True*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına ürün silmek için iletilmiş ve işlemi devam eden **batch_request** 'leri çeker. Sonrasında Satış Kanalından *check_deleted_products* komutu aracılığıyla sorgular. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*. Son olarak satış kanalından gelen cevabı Akinon Omnitrona ileterek akışı tamamlar.

get_product_batch_requests(*self*, *is_success_log=True*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına ürün yaratmak/güncellemek için iletilmiş ve işlemi devam eden **batch_request** 'leri çeker. Sonrasında Satış Kanalından *check_products* komutu aracılığıyla sorgular. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*. Son olarak satış kanalından gelen cevabı Akinon Omnitrona ileterek akışı tamamlar.

Fiyat Servisi

Fiyat servisi üzerinde ürünlerin fiyatlarını oluşturma, güncelleme ve silme aksiyonlarını üzerinde barındırır. Bu aksiyonları gerçekleştirmek için OmnitronIntegration ve ChannelIntegration entegrasyolanrından yardım alır.

class PriceService(*object*)

update_product_prices(*self*, *is_sync=True*, *is_success_log=True*, *add_product_objects=False*, *add_stock=False*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına güncellemesi iletilebilecek ürünlerin fiyat bilgisini çeker. Sonrasında Satış Kanalına *send_updated_prices* komutu aracılığıyla iletilir. *is_sync* parametresinin aldığı değere göre satış kanalı ile kurulacak iletişimin senkron mu asenkron mu olacağına karar verir. Asenkron olacak ise **batch_service** üzerinden gerkli kayıtlar oluşturulur. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

insert_product_prices(*self*, *is_sync=True*, *is_success_log=True*, *add_product_objects=False*, *add_stock=False*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına eklenebilecek ürünlerin fiyat bilgisini çeker. Sonrasında Satış Kanalına *send_inserted_prices* komutu aracılığıyla iletilir. *is_sync* parametresinin aldığı değere göre satış kanalı ile kurulacak iletişimin senkron mu asenkron mu olacağına karar verir. Asenkron olacak ise **batch_service** üzerinden gerkli kayıtlar oluşturulur. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

insert_product_prices_from_extra_price_list(*self*, *is_sync=True*, *is_success_log=True*, *add_product_objects=False*, *add_stock=False*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına eklenebilecek ürünlerin ekstra fiyat bilgisini çeker. Sonrasında Satış Kanalına *send_inserted_prices* komutu aracılığıyla iletilir. *is_sync* parametresinin aldığı değere göre satış kanalı ile kurulacak iletişimin senkron mu asenkron mu olacağına karar verir. Asenkron olacak ise **batch_service** üzerinden gerkli kayıtlar oluşturulur. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

update_product_prices_from_extra_price_list(*self*, *is_sync=True*, *is_success_log=True*, *add_product_objects=False*, *add_stock=False*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına güncellemesi iletilebilecek ürünlerin ekstra fiyat bilgisini çeker. Sonrasında Satış Kanalına *send_updated_prices* komutu aracılığıyla iletilir. *is_sync* parametresinin aldığı değere göre satış kanalı ile kurulacak iletişimin senkron mu asenkron mu olacağına karar verir. Asenkron olacak ise **batch_service** üzerinden gerkli kayıtlar oluşturulur. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

get_currency_mappings(*self*)

Bu fonksiyon yardımcı görevi görür. OmnitronIntegration üzerinde bulunan konfigrasyon üzerinden *CURRENCY_MAPPINGS* okur ve formatlayıp döndürür.

get_price_batch_requests(*self*, *is_success_log=True*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına ürün yaratmak/güncellemek için iletilmiş ve işlemi devam eden **batch_request** 'leri çeker. Sonrasında Satış Kanalından *check_products* komutu aracılığıyla sorgular. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*. Son olarak satış kanalından gelen cevabı Akinon Omnitrona ileterek akışı tamamlar.

Stok Servisi

Stok servisi üzerinde ürünlerin stoklarını oluşturma, güncelleme ve silme aksiyonlarını üzerinde barındırır. Bu aksiyonları gerçekleştirmek için OmnitronIntegration ve ChannelIntegration entegrasyolanrından yardım alır.

class StokService(*object*)

update_product_stocks(*self*, *is_sync=True*, *is_success_log=True*, *add_product_objects=False*, *add_price=False*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına güncellemesi iletilebilecek ürünlerin stok bilgisini çeker. Sonrasında Satış Kanalına *send_updated_stocks* komutu aracılığıyla iletilir. *is_sync* parametresinin aldığı değere göre satış kanalı ile kurulacak iletişimin senkron mu asenkron mu olacağına karar verir. Asenkron olacak ise **batch_service** üzerinden gerkli kayıtlar oluşturulur. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

insert_product_stocks(*self*, *is_sync=True*, *is_success_log=True*, *add_product_objects=False*, *add_price=False*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına eklenebilecek ürünlerin stok bilgisini çeker. Sonrasında Satış Kanalına *send_inserted_stocks* komutu aracılığıyla iletilir. *is_sync* parametresinin aldığı değere göre satış kanalı ile kurulacak iletişimin senkron mu asenkron mu olacağına karar verir. Asenkron olacak ise **batch_service** üzerinden gerkli kayıtlar oluşturulur. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

insert_product_stocks_from_extra_stock_list(*self*, *is_sync=True*, *is_success_log=True*, *add_product_objects=False*, *add_price=False*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına eklenebilecek ürünlerin ekstra stok bilgisini çeker. Sonrasında Satış Kanalına *send_insert_stocks* komutu aracılığıyla iletilir. *is_sync* parametresinin aldığı değere göre satış kanalı ile kurulacak iletişimin senkron mu asenkron mu olacağına karar verir. Asenkron olacak ise **batch_service** üzerinden gerkli kayıtlar oluşturulur. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

update_product_stocks_from_extra_stock_list(*self*, *is_sync=True*, *is_success_log=True*, *add_product_objects=False*, *add_price=False*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına güncellemesi iletilebilecek ürünlerin ekstra stok bilgisini çeker. Sonrasında Satış Kanalına *send_updated_stocks* komutu aracılığıyla iletilir. *is_sync* parametresinin aldığı değere göre satış kanalı ile kurulacak iletişimin senkron mu asenkron mu olacağına karar verir. Asenkron olacak ise **batch_service** üzerinden gerkli kayıtlar oluşturulur. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

get_warehouse_mappings(*self*)

Bu fonksiyon yardımcı görevi görür. OmnitronIntegration üzerinde bulunan konfigrasyon üzerinden *WAREHOUSE_CODES* okur ve formatlayıp döndürür. Ürün stock listelerini maplemek için kullanılır.

get_stock_batch_requests(*self*, *is_success_log=True*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına ürün yaratmak/güncellemek için iletilmiş ve işlemi devam eden **batch_request** 'leri çeker. Sonrasında Satış Kanalından *check_products* komutu aracılığıyla sorgular. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*. Son olarak satış kanalından gelen cevabı Akinon Omnitrona ileterek akışı tamamlar.

Resim Servisi

Resim servisi üzerinde ürünlerin resimlerini oluşturma, güncelleme ve silme aksiyonlarını üzerinde barındırır. Bu aksiyonları gerçekleştirmek için OmnitronIntegration ve ChannelIntegration entegrasyolanrından yardım alır.

class ImageService(*object*)

update_product_images(*self*, *is_sync=True*, *is_success_log=True*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına güncellemesi iletilebilecek ürünlerin resim bilgisini çeker. Sonrasında Satış Kanalına *send_updated_images* komutu aracılığıyla iletilir. *is_sync* parametresinin aldığı değere göre satış kanalı ile kurulacak iletişimin senkron mu asenkron mu olacağına karar verir. Asenkron olacak ise **batch_service** üzerinden gerkli kayıtlar oluşturulur. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

insert_product_images(*self*, *is_sync=True*, *is_success_log=True*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına eklenebilecek ürünlerin resim bilgisini çeker. Sonrasında Satış Kanalına *send_inserted_images* komutu aracılığıyla iletilir. *is_sync* parametresinin aldığı değere göre satış kanalı ile kurulacak iletişimin senkron mu asenkron mu olacağına karar verir. Asenkron olacak ise **batch_service** üzerinden gerkli kayıtlar oluşturulur. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

get_image_batch_requests(*self*, *is_success_log=True*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına ürün yaratmak/güncellemek için iletilmiş ve işlemi devam eden **batch_request** 'leri çeker. Sonrasında Satış Kanalından *check_products* komutu aracılığıyla sorgular. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*. Son olarak satış kanalından gelen cevabı Akinon Omnitrona ileterek akışı tamamlar.

Sipariş Servisi

Sipariş servisi üzerinde siparişleri oluşturma, güncelleme ve iptal aksiyonlarını üzerinde barındırır. Bu aksiyonları gerçekleştirmek için OmnitronIntegration ve ChannelIntegration entegrasyolanrından yardım alır.

class OrderService(*object*)

fetch_and_create_order(*self*, *is_success_log=True*)

Bu fonksiyon öncelikle Satış Kanalına bağlanır ve oluşmuş siparişleri çeker. Sonrasında Akinon'a üzerinde bulunan **create_order** fonksiyonu aracılığıyla iletilir. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

create_order(*self*, *omnitron_integration:* OmnitronIntegration, *channel_order: ChannelCreateOrderDto*) → Union[Order, None]

Bu fonksiyon satış kanalından gelen siparişleri Akinon'a aktarmayı sağlar. Satış kanalından gelen veriler üzerinden sırasıyla müşteri ve adres bilgilerini Akinon'a yazar ve dönen cevabı tutar. Sipariş üzerindeki kargo firmasının da Akinondaki karşılığını okuyarak sipariş yaratmak için gerekli olan tüm bilgileri toplamış olur. Son olarak Akinon'a sipariş bilgileri iletilir.

update_orders(*self*, *is_sync=True*, *is_success_log=True*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına iletilebilecek güncellenmiş siparişleri çeker. Sonrasında Satış Kanalına *send_updated_orders* komutu aracılığıyla iletilir. *is_sync* parametresinin aldığı değere göre satış kanalı ile kurulacak iletişimin senkron mu asenkron mu olacağına karar verir. Asenkron olacak ise **batch_service** üzerinden gerkli kayıtlar oluşturulur. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

get_order_batch_requests(*self*, *is_success_log=False*)

Bu fonksiyon öncelikle Akinon Omnitron'a bağlanır ve satış kanalına sipariş yaratmak/güncellemek için iletilmiş ve işlemi devam eden **batch_request** 'leri çeker. Sonrasında Satış Kanalından *check_orders* komutu aracılığıyla sorgular. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*. Son olarak satış kanalından gelen cevabı Akinon Omnitrona ileterek akışı tamamlar.

`fetch_and_create_cancel`(*self*, *is_success_log=True*)
Bu fonksiyon öncelikle Satış Kanalına bağlanır ve oluşmuş sipariş iptallerini çeker. Sonrasında Akinon'a üzerinde bulunan **create_cancel** fonksiyonu aracılığıyla iletilir. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

`create_cancel`(*self*, *omnitron_integration:* OmnitronIntegration, *cancel_order_dto: CancelOrderDto*)
Bu fonksiyon satış kanalından gelen sipariş iptallerini Akinon'a aktarmayı sağlar. Akinon'a *create_order_cancel* komutu aracılığıyla parametre olarak verilen sipariş iptallerini iletir.

`fetch_and_update_order_items`(*self*, *is_success_log=True*)
Bu fonksiyon öncelikle Satış Kanalına bağlanır ve güncellenmiş siparişleri çeker. Sonrasında Omnitron'a *update_order_items* komutu aracılığıyla güncellemeleri OrderItem bazında iletilir. Bir hata ile karşılaşılır ise *error_report* oluşturulur *Sales Channel Logları*.

2.4 Terminoloji

Proje kapsamında kullanılan terimler aşağıda açıklanmıştır.

Pazaralanı

Çevrimiçi satış platformlarında satış yapmak isteyen şirketlerin, çevrimiçi mağazalar açarak ürünlerini sunduğu ve kolayca büyük bir müşteri havuzuna ulaşabildiği sistemlerdir.

Akinon Commerce Cloud

Akinon'un bulut çözümlerini içeren ve de müşterilerin ihtiyaçlarına uygun çözümleri kendierinin de geliştirebildiği modüler bulut ortamıdır.

Omnitron

Akinon'un ürün envanterinde temel e-ticaret fonksiyonlarını oluşturan ve tüm satış kanalı operasyonlarını yönetebildiği bir panel sunan üründür.

Channel App

Yeni bir pazaralanı entegrasyonu için gereken temel yapıyı bulunduran ve farklı pazaralanları için farklı ihtiyaçlar olabileceğinden dolayı esnek bir altyapı sunan projedir.

Channel App Template

Channel App'i bir bağımlılık olarak barındıran ve yeni pazaralanı entegrasyonu için doğrudan klonlanarak ilerlenebilecek taslak projedir.

Celery

Asenkron olarak, belirli zamanlarda ve belirli aralıklarda task çalıştırma imkanı sunan , dağıtık kuyruk yapısı ile taskları sıralı olarak işleyen ve yoğun kaynak gerektiren kuyruklar için de işçi sayısının ayarlanabildiği teknolojidir.

Flower

Flower, Celery için önyüz sağlayan bir yönetim aracıdır.

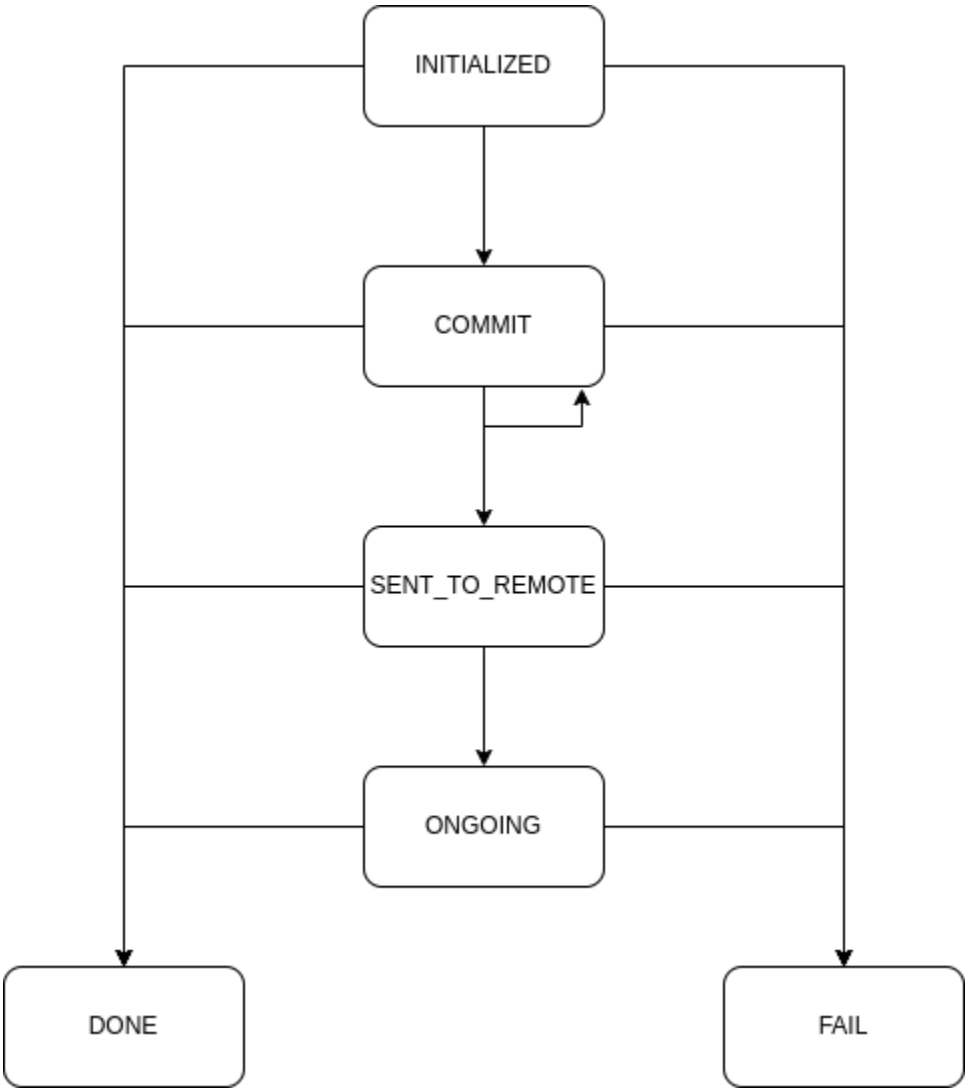
2.5 Akışlar

Akışlar bir iş fonksiyonunun gerçekleşmesi için yapılacak adımları, çalıştırılacak komutları listeler. Omnitron ile Satış Kanalı entegrasyonuna ait komutlardan yapılacak işle alakalı olanları akışın gerektirdiği sıraya göre çalıştırır.

Akışlarda komutların yanısıra, akışın herhangi bir noktasında hata alıp almadığımızı görebilmek ya da parçalara bölünebilir işleri birden fazla parçada yapıp birbiri ile ilişkisini kurabilmek için BatchRequest adında bir model tanımlandı.

2.5.1 BatchRequest Modeli

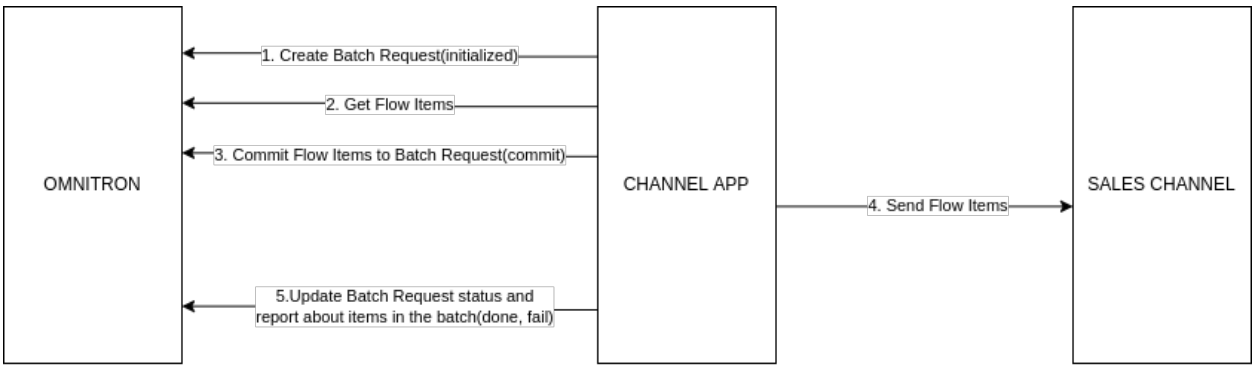
BatchRequest modeli, işlenen nesneleri yöneten, takip eden, anlamlı hatalar üreten ve de bir sürecin birden fazla parçada çalışmasına olanak sağlayan bir durum makinesi olarak çalışıyor. Örneğin bir ürün gönderme akışında 10 tane ürün gönderiliyor ve bunların bir tanesinde uygulama tarafındaki kodlama hatasından dolayı hata fırlatıldı. Bu durumda hata loglarına ulaşmak mümkünse dahi beraberinde gönderilemeyen ürünleri takip edemeyebiliriz. Ya da ürünleri gönderdik ama satış kanalı tarafındaki istek asenkron çalışıyor. Bu isteğin sonucunu beklemek Channel App tarafındaki taskın da beklemesini gerektireceği için sağlıklı bir karar olmaz. Asenkron çalışan istekleri gönderip anında dönen referans id değerlerini saklayabilecek ve işlemekte olduğumuz ürünler ile iliştirip sonradan kontroller yapabilecek bir yapı sunuyor.



Statüler

1. **initialized** Başlangıç durumunu ifade eder. BatchRequest oluşturulduğunda hayatına bu durum üzerinden başlar.
2. **commit** Commit statüsüne geçildiğinde BatchRequest bir Omnitron modeline kilitlenir. Örneğin ürün gönderme akışı için bir BatchRequest oluşturup onu commit statüsüne aldığımızda, model tipi (content_type) "product" olarak belirlenir. Paralel olarak "product" tipinde başka bir BatchRequest'in commit statüsüne gelmesi engellenir. Bu şekilde aynı anda yalnızca bir ürün gönderme akışı çalışır. Bu statüye geçişte BatchRequest'te işlenecek nesneler bildirilir. Ayrıca tek seferde tüm nesnelerin bildirilmesine gerek duyulmaz. Bunun için commit statüsünde olan bir BatchRequest'e tekrar commit statüsüne geçirilerek nesne eklemesi yapılabilir.
3. **sent_to_remote** Satış kanalı entegrasyonunu kullanarak nesneleri ilettiğimizde sent_to_remote statüsüne çekilir. Eğer satış kanalının servisi asenkron çalışıyorsa ve toplu işlem için bir id değeri iletiyorsa, bu durum geçişinde id bilgisini de remote_batch_id alanıyla birlikte beslemek gerekir. Bu alan taskın sonuçlanıp sonuçlanmadığı bilgisini kontrol edip BatchRequest statüsünü ilerletmek amaçlı kullanılacaktır. Eğer bir id değeri dönmüyorsa, statüyü ilerletmek için yapılması gereken kontrolü başka metotlarla yapmak gerekir. Örneğin ürün gönderme akışında, ürünleri listeleyen servisi çağırıp onun çıktısını filtreleyerek kontrol edilebilir ya da bazı kendini bilmez API tasarımcılarının listeleme için bir servis sunmaması durumunda oluşturma isteği atıp hata almak gibi yeraltı metotlarına başvurmak da gerekebilir.
4. **ongoing** Bu statü, sent_to_remote statüsü ile aynı durumu ifade ediyor arada ufak bir nüans farkı var. O da sent_to_remote statüsünde olan BatchRequest nesneleri için işlemin sonuçlanıp sonuçlanmadığını kontrol ettiğimizde hala devam ettiği durumda ongoing statüsüne ilerletiyoruz.
5. **done** BatchRequest içerisindeki tüm nesnelerin işlendiğini ifade ediyor. Bu nesnelerin hepsi hata almış da olabilir. Buradaki *done* BatchRequest'in başarılı sonlanması anlamına geliyor. Ürünlerin kendi başlarına hata alması *done* statüsüne geçişe engel bir teşkil etmiyor. Statü geçişi sırasında, mevcut BatchRequest nesnesi içerisinde işlenen ürünler için geri bildirim verilmesi gerekiyor. Örneğin ürünler satış kanalında oluştuysa, satış kanalında bir id'ye sahip oldukları anlamına geliyor. Bu id'nin Omnitron tarafına beslenmesi gerekiyor ki sonradan yapılan güncellemelerde satış kanalına hangi id ile istek atılması gerektiği bilinsin.
6. **fail** Mevcut BatchRequest'te işlenen nesnelerin tamamı işlenmeden, işlemin durdurulmasına yol açacak bir hata olduysa ya da daha satış kanalına gönderme isteğine sıra gelmeden nesnelerin tamamında eksik bir nokta varsa bu statü ile işlem sonlanabilir.

2.5.2 Senkron Akışlar



Satış Kanalı servisleri senkron olarak çalışıyorsa, atılan isteğe referans id'si dönmek yerine doğrudan asıl beklediğimiz sonuçla alakalı bir çıktı üretiyorsa, ortaya daha basit bir süreç çıkıyor.

Senkron Akış Adımları

1. **BatchRequest kaydını oluştur.** *initialized* statüsünde olarak oluşuyor. Bu aşamada BatchRequest, herhangi bir model tipine atanmaz.
2. **Akış içerisinde işlenecek nesneler çekilir.** Bunlar ürün akışında ürünleri, stok akışında stokları temsil eder.
3. **Akıştaki işlediğimiz nesneleri belirtmek için BatchRequest'i *commit* statüsüne geçiriyoruz.** Burada model tipi (content_type) olarak ana işlenen akış neyse onu koymalıyız. Örneğin ürünleri gönderiyorsak ve de ürün akışında stok, fiyat ve fotoğraf gibi modelleri de beslememiz gerekiyorsa, BatchRequest'in model tipi "product" olmalı. BatchRequest içerisinde gönderilen *objects* bloğunda ise her bir model kendi tipiyle (Evet, hem BatchRequest'in model tiplerini, hem de içerisinde *objects* bloğunda gönderilen nesnelerin model tiplerini besliyoruz.) gönderiliyor.
4. Satış kanalına isteği gönderiyoruz ve sonucunu alıyoruz.
5. **Satış kanalından gelen sonuca göre BatchRequest'i *done* ya da *fail* statülerine ilerletiyoruz.** *fail* statüsü için *objects* bilgisini beslemeye gerek yok ancak nesnelerle ilgili tekil olarak hataları saklamak istiyorsak *objects* parametresini koymak gereklidir.

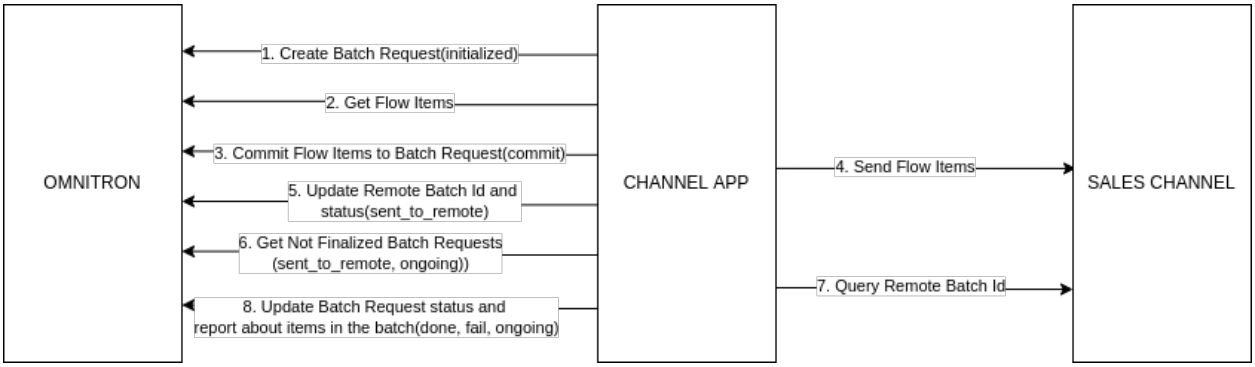
2.5.3 Asenkron Akışlar

Satış Kanalı servisleri asenkron olarak çalışıyorsa, atılan isteğin gerçek sonucunu değil, sonucu kontrol etmek için bir referans id'si dönmesi beklenir. Bunun sebebi yapılacak işlemin uzun sürebilmesinden kaynaklıdır. Örneğin ürün gönderme akışında 1000 ürün işlediğini düşünürsek, senkron çalışacak şekilde tasarlandığında sağlıklı bir süreç oluşturması pek olası değil. Aynı zamanda onlarca yüzlerce farklı müşteri de ürün girişi yapıyor olabilir. Bunların sıralı olarak işlenmesi gerekiyor. İşlenen veriler de birbirini etkileyebileceği için paralel olarak işlem yapmak mümkün de olmayabilir.

İlk üç adım senkron yapı ile aynı olduğu için 4. adımdan başlanarak anlatılacaktır.

Asenkron Akış Adımları

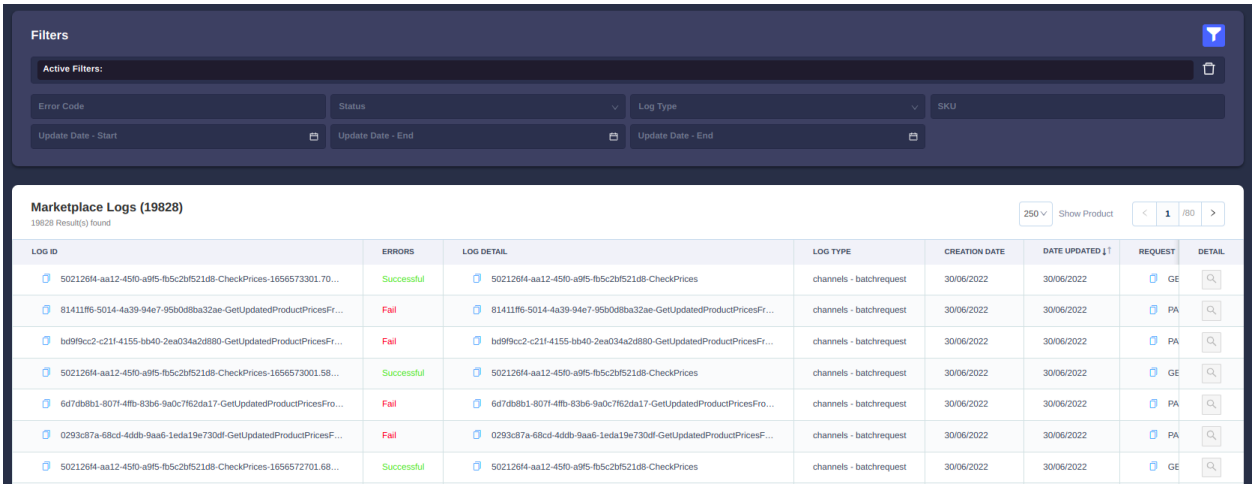
4. **Satış kanalına attığımız istekte, isteğin asıl sonucunu değil, asıl sonucu kontrol etmek için** gönderilen id değerini alıyoruz. Herhangi bir değer gönderilmezse satış kanalına kontrol için attığımız sorguyu (7. adım) farklı bir şekilde yapmamız gerekiyor.
5. **BatchRequest'in statüsünü *sent_to_remote* statüsüne güncelliyoruz ve 4. adımda bir referans id'si** geri döndüyse, güncelleme sırasında *remote_batch_id* alanına da o değeri besliyoruz.
6. **Bu adımdan başlayarak akış yeni bir task üzerinden devam ediyor.** Önceki adımda süreç asenkron olduğu için statüyü ilerletip task'ı sonlandırmıştık. Belirli aralıklarla deneyerek sürecin tamamlandığını kontrol etmemiz gerekiyor. Başarılı ya da başarısız bir sonuç aldığımızda da gerekli güncellemeleri yapıp süreci sonlandırıyoruz. Statüsü *sent_to_remote* ya da *ongoing* olan BatchRequest nesnelerini sorguluyoruz.
7. **BatchRequest içerisindeki *remote_batch_id* değerini kullanarak akış içerisindeki nesnelerin** durumlarını sorguluyoruz. Eğer referans için bir id yoksa satış kanalının farklı servislerini kullanarak işlemin durumunu kontrol etmek gerekecektir. Bu aşamada BatchRequest nesnelerini çektikten sonra Omnitron tarafında BatchRequest'e bağlı olan nesneleri de sorgulayıp onların *remote_id* değerlerini kullanarak istek atmak gerekir.
8. **Son aşama olarak da satış kanalındaki güncel bilgilerle Omnitron tarafındaki statüyü de** güncellememiz gerekiyor. Eğer işlem tamamlanmadıysa *ongoing* olarak güncelliyoruz ve bir sonraki denemede 6. adımdan başlayarak süreç yenileniyor. Eğer işlem tamamlandıysa ve satış kanalı, işlenen BatchRequest için genel bir hata döndüyse, statüyü *fail* olarak güncelliyoruz.

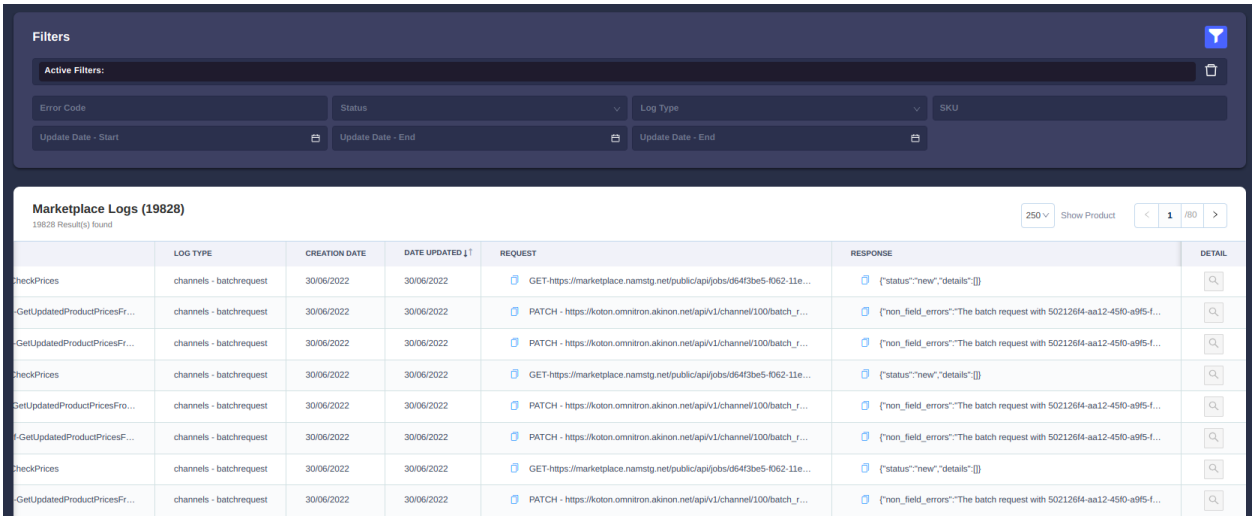


2.6 Sales Channel Logları

Sales Channel'ın olduğu kanala geçilir.

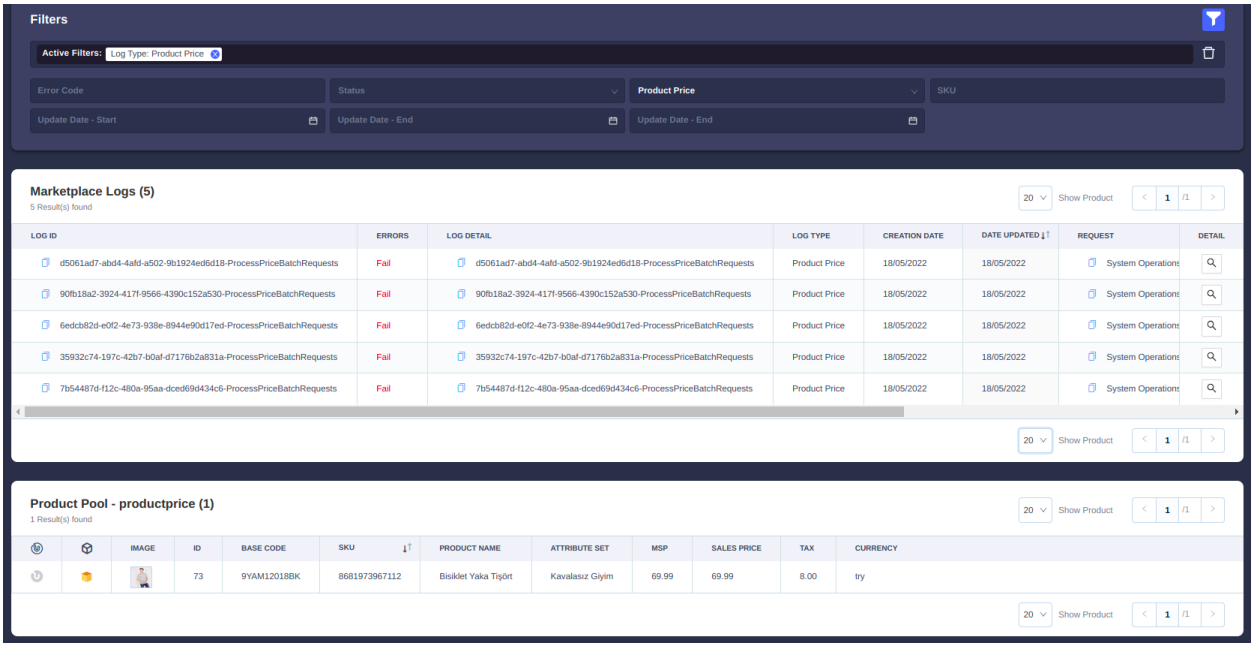
Menu -> Sales Channels -> Sales Channel Settings -> Marketplace Logs





Ürün Detayları

Ürün ile ilgili detay bilgi bulunması halinde bu kısım açılabilir. Tabloda bulunan detay kolonu tıklanabilir olur.



2.6.1 Tablo Başlıkları

Log Id Hangi işlemin yapıldığı gözükür. Formatı {batchrequest local batch id}-{İşlemin adı}-{uniqlik sağlayacak ekstra bilgi}

Log Detail Log Id ile aynıdır

Errors Status ü gösterir Succesful veya Fail

Log type İşlemin türünün bilgisidir.

request İşlemde istek atılıyorsa bu istek ile ilgili bilgiler bulunur.

response İşlemde bir cevap alınıyor ise bu bilgiler tutulur.

Detail Log type product, productprice, productstock ise detaylarını sayfanın altında yeni bir pencerede gösterilir.

2.6.2 Filtreler

Error Code Log Id alanında arama yapılır

Status Errors alanında Successful veya Fail seçilebilir.

Log Type Log type alanında arama yapar.

Update Date Start Oluşturulma Tarih aralığı başlangıç.

Update Date End Oluşturulma Tarih aralığı bitti.

Dizin

Semboller

B

C

D

F

G

I

N

O

P

S

T

U

V