Lab: Adobe Audience Manager REST API for Automated Segmentation & Activation - L4284

In this hand-on lab you will modify a sample e-commerce application that demonstrates calling Adobe Audience Manager's REST APIs and creates the data required to segment your site visitors automatically.

The sample app is built using Python language and Django framework. It has no other external dependencies. These are already installed for you.

Adobe Audience Manager (AAM) background

AAM is a Data Management Platform (DMP). It allows you to combine audience information on your visitors from different sources such as from your site, your CRM systems, strategic partners and third party relationships. You can make these audiences actionable by sending this information to wherever you choose, whether it's your own systems for site customization or to ad networks for ad targeting.

Getting started

- Find **L4284** user on the screen to login
- Your computer password is **summit2017**
- You will be provided with a unique username and password for Adobe Audience Manager Sandbox environment that corresponds to a unique company
- You will all share the same client id and client secret
- Audience Manager credentials will be deactivated after summit

Installing and starting the sample app

1. Launch a terminal window. You can do this by hitting **command + space**, then typing **terminal** in spotlight search, and hitting enter.
2. Change the directory to Desktop:

```
cd Desktop
```

3. Run the following git command to get the code in your Desktop:
(this step is not required on lab computer)

```
git clone https://github.com/Adobe-Marketing-Cloud/audiencemanager-api-lab
```

4. Change the directory to the lab project:

```
cd audiencemanager-api-lab
```

5. To install the dependencies, type the following command:
(this step is not required on lab computer)

```
bash install.sh
```

6. To start the sample application, type the following command:

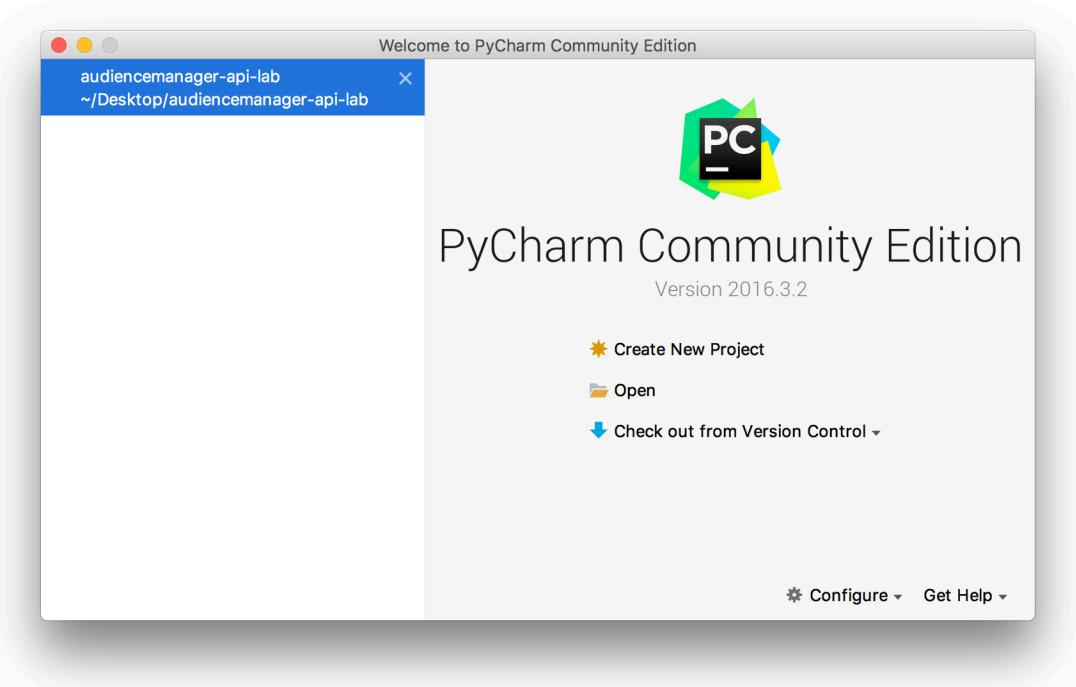
```
bash run.sh
```

The application will now start and will automatically build and restart as you make changes to the code.

7. In a browser, visit http://localhost:8000 to access the app
8. To access product administration, visit http://localhost:8000/admin
Administrator username: **admin,** password: **admin**

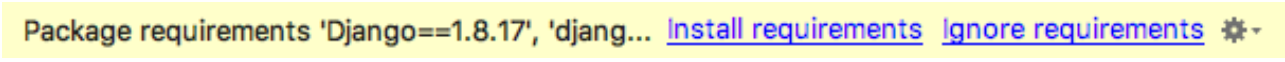
Configuring an IDE or Code Editor

PyCharm Community Edition has been installed on your lab computer as an IDE, and the lab project is already imported in PyCharm. If the project is not imported, you can do it by clicking open and then selecting the lab project folder under Desktop.

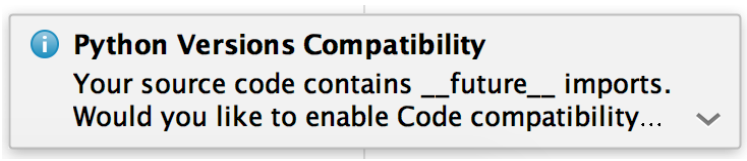


Alternatively, you can choose to use Brackets (already installed on lab computer) or download any Text Editor of your choice and use that for this lab.

You can ignore messages in PyCharm that looks like this:



Or this:



Exercises

The exercises will build off of each other. If you get stuck on any exercise, you can **`git checkout exercise-<exercise-number>`** which contains the solution up to that exercise number.

You should refer to the Audience Manager REST API Documentation to help you complete these exercises. Link: https://adobe.ly/1QR8eY8

You will be using Audience Manager Sandbox environment for this lab.

- AAM Sandbox UI location: https://bank-sandbox.demdex.com
- AAM Sandbox API location: https://api-sandbox.demdex.com

You will need an Audience Manager client id and client secret for generating OAuth2 tokens. You will all share the same client id and client secret for this lab:

- OAuth2 Client ID: **summit2017**
- OAuth2 Client Secret: **summit2017**

You are being provided with a unique username and password for Adobe Audience Manager Sandbox environment that corresponds to a unique company:

- AAM lab username: **summit-lab-user-<your computer id>**
- AAM lab password: **Summit##2017**

You can access AAM UI and API using the same credentials in this lab. Before proceeding with the exercises, you should check your UI access by visiting the AAM Sandbox UI location from a browser and by entering your unique AAM credentials.

There is a folder called **solutions** where you can find the required codes for each of the exercises.

Exercise-01: Get an access token and refresh token

1. Use curl to send a request to https://api-sandbox.demdex.com/oauth/token to get your access token and refresh token. For this exercise, you'll be using the OAuth2 password flow where with a single request you exchange client id, client secret, username, and user's password for tokens.

   For example:

```
curl -X POST https://api-sandbox.demdex.com/oauth/token
--user summit2017:summit2017 –data 'grant_type=password
&username=summit-lab-user-00&password=Pa$$W0rd'

Response:
{
    "access_token": "abcd1234-b8b2-abcd-61109b86dcba",
    "token_type": "bearer",
    "refresh_token": "fedcba12-43dd-811e-c94905f4abcd",
    "expires_in": 86399,
    "scope": "read write"
}
```

2. Now we will check whether our access_token is working for AAM API access:

```
curl https://api-sandbox.demdex.com/v1/segments
-H 'Authorization: Bearer abcd1234-b8b2-abcd-61109b86dcba'

Response:
[]
```

3. After that, you need to place the access_token in a config file located inside the lab project. The location of the file is: **configs/aam_configs.py**

   Replace the placeholder inside the config file with your access token in this line:

```
# API Credentials
AAM_API_ACCESS_TOKEN = 'PLACE-YOUR-ACCESS-TOKEN-HERE'
```

4. Basic AAM API paths are already added for you in **aam_configs.py** file.

Exercise-02: Create aam.py and implement an API connector

5. Create a new file under app directory: **`app/aam.py`**
6. Now write the following code inside the file to implement a basic API exchange:

```
import requests, time, json
from django.conf import settings as configs


GET    = 'GET'
POST   = 'POST'
PUT    = 'PUT'
DELETE = 'DELETE'


# generic method that will exchange API requests between client and AAM
def api_exchange(method, url, payload=None):

    print ('AAM API Request: %s %s Payload:%s' % (method, url, str(payload)))

    headers = {
        'Authorization': 'Bearer ' + configs.AAM_API_ACCESS_TOKEN,
        'Content-Type': 'application/json'
    }

    if method == GET:
        response = requests.get(url, headers=headers)
    elif method == POST:
        response = requests.post(url, headers=headers, data=json.dumps(payload))
    elif method == PUT:
        response = requests.put(url, headers=headers, data=json.dumps(payload))
    elif method == DELETE:
        response = requests.delete(url, headers=headers)
    else:
        raise ValueError('Unsupported HTTP method: %s' % method)

    print ('AAM API Response: %s' % response.status_code)
    if response: print json.dumps(response.json(), indent=4)
    return response
```

7. Create a method for checking http response codes:

```
# checks whether the response from the request was successful
def successful(response):
    return response is not None and response.status_code in [200, 201, 204]
```

8. Create a method for getting self-user information:

```
def get_self():
    response = api_exchange(GET, configs.AAM_SELF_USER_API_PATH)
```

9. Open **app/models.py** and add these lines under rest of the import statements:

```
from . import aam

aam.get_self()
```

10. Now switch to your terminal. If you check the logs, you will notice the application was restarted automatically and an API call was made like this:

```
AAM API Request: GET https://api-sandbox.demdex.com/v1/users/self
Payload:None

AAM API Response: 200
{
    "status": "ACTIVE",
    "username": "summit-lab-user-00",
    "updateTime": 1487212015000,
    "uid": 51070,
    "firstName": "Summit 2017",
    "loginFailures": 0,
    "admin": true,
    "lastName": "Lab User 00",
    "companyName": "Summit 2017 Lab Company 00",
    "pid": 11761,
    "upUID": 51070,
    "crUID": 824,
    "createTime": 1487211928000,
    "oktaEnabled": false,
    "groups": [],
    "email": "hossain@adobe.com"
}
```

Exercise-03: Create Shop Data Source automatically

11. Data Sources signify where a visitor data came from. Traits and Segments must be associated with a Data Source. We will first check whether a Shop Data Source already exists in AAM by querying the API using an Integration Code “shop”. If the Data Source does not exist, we will create one automatically.

    Create a new method to do this in **app/aam.py**

```
def get_or_create_shop_datasource():
    response = api_exchange(GET, configs.AAM_DATASOURCE_API_PATH + '?integrationCode=shop')

    found = False
    if successful(response) and len(response.json()) > 0:
        found = True
        datasource = response.json()[0]

    if not found:
        datasource = {
            'name': 'Shop Data Source',
            'integrationCode': 'shop',
            'uniqueTraitIntegrationCodes': True,
            'uniqueSegmentIntegrationCodes': True
        }
        response = api_exchange(POST, configs.AAM_DATASOURCE_API_PATH, datasource)
        datasource = response.json()
    print 'AAM datasource:', datasource
    return datasource
```

12. Test the method from **app/models.py**

```
from . import aam

aam.get_self()
aam.get_or_create_shop_datasource()
```

13. Check your terminal logs again to see the output from Data Source API calls.

Exercise-04: Create Trait folders for each Category automatically

14. Traits are stored in folder taxonomies. We will create a Trait folder for each product category automatically.

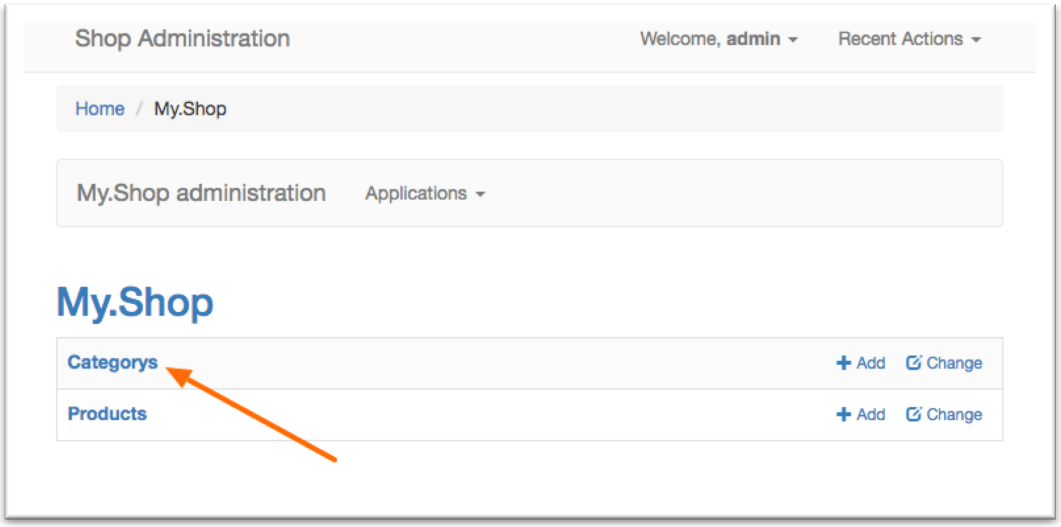
    First, create these methods in **app/aam.py** for managing Trait Folders:

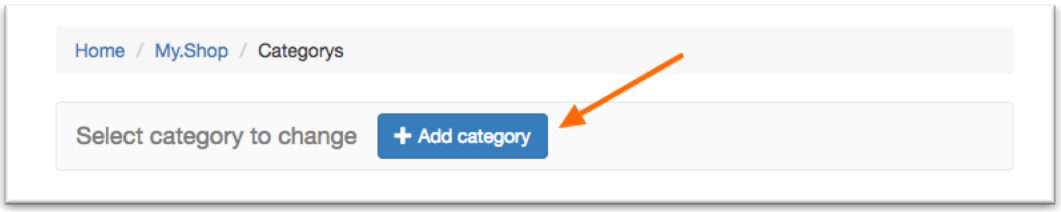
```
def get_category_trait_folder(category):
    folder_name = category.name
    trait_folder = None
    response = api_exchange(GET, configs.AAM_TRAIT_FOLDER_API_PATH)
    if successful(response):
        root_folder = response.json()[0]
        for sub_folder in root_folder['subFolders']:
            if sub_folder['name'] == folder_name:
                trait_folder = sub_folder
                break
    return trait_folder


def create_category_trait_folder(category):
    folder = {
        'name': category.name,
        'parentFolderId': 0
    }
    response = api_exchange(POST, configs.AAM_TRAIT_FOLDER_API_PATH, folder)
    print 'AAM trait folder create:', response
    return folder


def update_category_trait_folder(old_category, new_category):
    folder = get_category_trait_folder(old_category)
    if folder is not None:
        folder['name'] = new_category.name
        folder_id = str(folder['folderId'])
        response = api_exchange(PUT, configs.AAM_TRAIT_FOLDER_API_PATH + '/' + folder_id, folder)
        print 'AAM trait folder update:', response
        return folder
    else:
        return create_category_trait_folder(new_category)


def get_or_create_category_trait_folder(category):
    folder = get_category_trait_folder(category)
    if folder is not None:
        return folder
    else:
        return create_category_trait_folder(category)
```

15. Add these two lines in **`app/models.py`** for triggering the methods we just created automatically when we create or update a Category in our sample app:

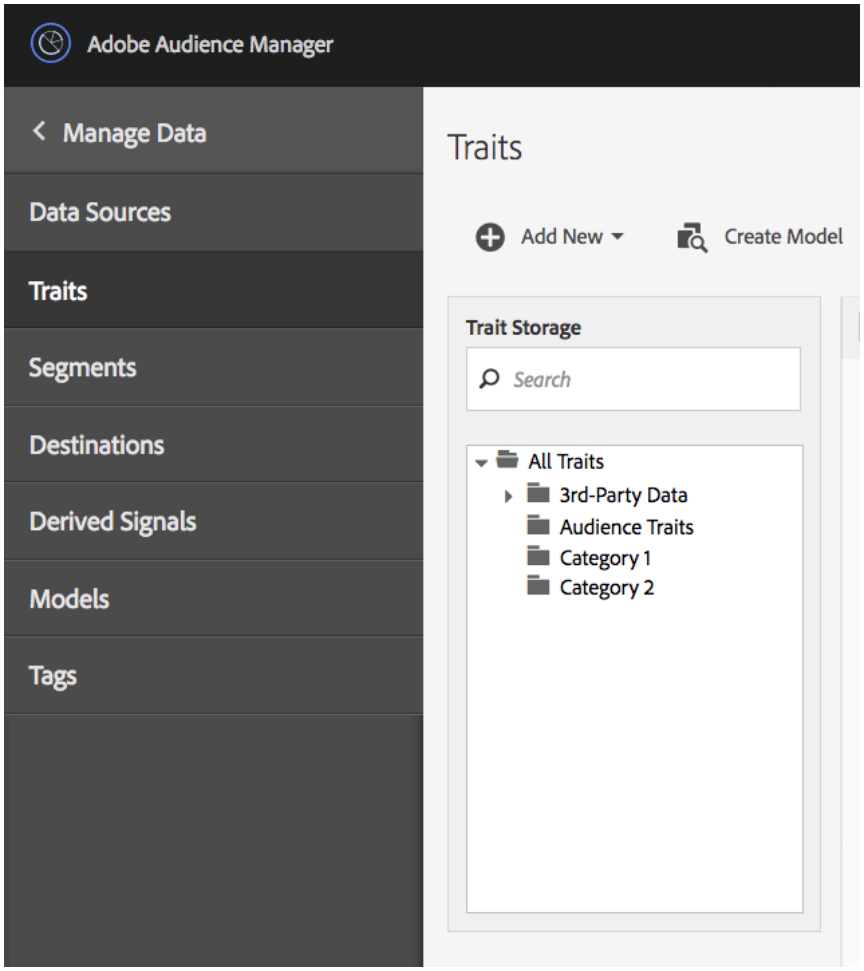
```
def save(self, *args, **kwargs):
    if not self.id:
        # Category is being created
        super(Category, self).save(*args, **kwargs)
        aam.create_category_trait_folder(self)
    else:
        # Category is being updated
        old_category = Category.objects.get(id=self.id)
        super(Category, self).save(*args, **kwargs)
        aam.update_category_trait_folder(old_category, self)
```

16. Now go to Shop Administration in your browser: http://localhost:8000/admin
17. And try adding a new Category:





18. Open Adobe Audience Manager (Sandbox) in a separate browser tab by visiting: https://bank-sandbox.demdex.com and using your AAM lab credentials.
19. If you navigate to the Traits tab, you will notice that a new folder was automatically created with the same name as your Product Category.
20. Now go back to Shop Administration and update one of the existing Categories. If you refresh the Traits tab in AAM, you will notice that another folder was automatically created for the updated category. This is because inside **`update_category_trait_folder`** method, we create the trait folder if the folder was not found.
21. You can also try renaming an existing Category and see the name change being reflected in AAM.



Exercise-05: Create Traits for each Product automatically

22. We will auto create a trait to represent a visitor who is interested in a product. To do this, we will implement few methods to GET, POST and PUT traits on AAM side.

    We will use a unique integration code for each trait. We will use **`product-<product-id>`** as an integration code so that we can find the trait in future based on the product id.

    As a trait rule, we will use **`product == <product-id>.`** This will eventually be the data collection signal that will come from the visitor's browser who is interested in that product.

```
def get_product_trait(product):
    trait_ic = 'product-' + str(product.id)
    response = api_exchange(GET, configs.AAM_TRAIT_API_PATH + '/ic:' + trait_ic)
    if successful(response):
        trait = response.json()
        return trait
    else:
        return None


def create_product_trait(product):
    trait_ic = 'product-' + str(product.id)
    trait = {
        'name': 'Interested in ' + product.name,
        'traitRule': 'product==' + str(product.id),
        'folderId': get_or_create_category_trait_folder(product.category)['folderId'],
        'dataSourceId': get_or_create_shop_datasource()['dataSourceId'],
        'integrationCode': trait_ic,
        'traitType': 'RULE_BASED_TRAIT'
    }
    response = api_exchange(POST, configs.AAM_TRAIT_API_PATH, trait)
    trait = response.json()
    print 'AAM trait create:', trait
    return trait
```

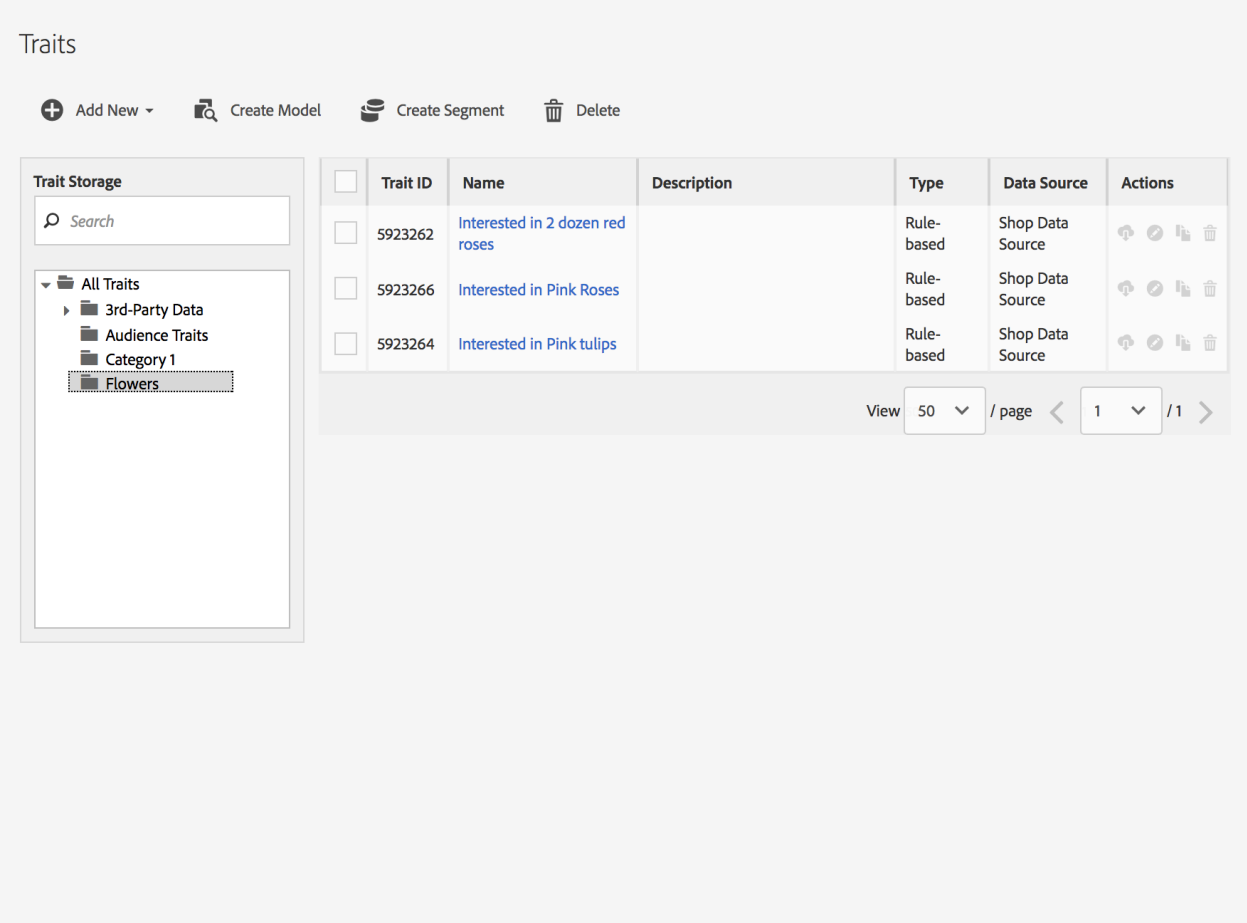
```
def update_product_trait(old_product, new_product):
    trait = get_product_trait(old_product)
    if trait is not None:
        trait['name'] = 'Interested in ' + new_product.name
        trait_id = str(trait['sid'])
        response = api_exchange(PUT, configs.AAM_TRAIT_API_PATH + '/' + trait_id, trait)
        trait = response.json()
        print 'AAM trait update:', trait
        return trait
    else:
        return create_product_trait(new_product)


def get_or_create_product_trait(product):
    trait = get_product_trait(product)
    if trait is not None:
        return trait
    else:
        return create_product_trait(product)
```

23. We will add these methods in **app/models.py** for triggering them automatically when a product is added or modified:

```
    def save(self, *args, **kwargs):
        if not self.id:
            # Product is being created
            super(Product, self).save(*args, **kwargs)
            aam.create_product_trait(self)
        else:
            # Product is being updated
            old_product = Product.objects.get(id=self.id)
            super(Product, self).save(*args, **kwargs)
            aam.update_product_trait(old_product, self)
```

24. After making these code changes, go to product administration and add or edit a Product. Then refresh Traits page in AAM and you will notice the corresponding traits were automatically created for those products.

Example of auto created Traits in Audience Manager:



Exercise-06: Create Segments for each Category automatically

25. We will auto create a segment to represent a visitor who is interested in any product within a category. To do this, we will implement few methods to GET, POST and PUT segments on AAM side.

    Similar to Trait, we will use a unique integration code for each Segment. We will use **`category-<category-id>`** as an integration code so that we can find the segment based on the category id.

    As a segment rule, we will use all the Traits that corresponds to all the Products under this specific category. For example, if Category-1 has two Products Product-A and Product-B, and we have two traits: Trait ID 123 and Trait ID 456 for them respectively, then segment rule will be 123T OR 456T. That means any visitor that qualifies for either of those product traits will get qualified for the category segment.

```
def get_category_segment(category):
    segment_ic = 'category-' + str(category.id)
    response = api_exchange(GET, configs.AAM_SEGMENT_API_PATH + '/ic:' + segment_ic)
    if successful(response):
        segment = response.json()
        return segment
    else:
        return None


def generate_segment_rule_for_category(category):
    trait_sids = []
    for product in category.get_products():
        trait_sids.append(str(get_or_create_product_trait(product)['sid']) + 'T')
    print trait_sids
    return ' OR '.join(trait_sids)
```

```
def create_category_segment(category):
    if len(category.get_products()) == 0:
        # no need to create segment because category has no products
        return

    segment_ic = 'category-' + str(category.id)
    segment_rule = generate_segment_rule_for_category(category)

    segment = {
        'name': 'Interested in ' + category.name,
        'segmentRule': segment_rule,
        'folderId': 0,
        'dataSourceId': get_or_create_shop_datasource()['dataSourceId'],
        'integrationCode': segment_ic
    }
    response = api_exchange(POST, configs.AAM_SEGMENT_API_PATH, segment)
    segment = response.json()
    print 'AAM segment create:', segment
    return segment


def update_category_segment(old_category, new_category):
    segment = get_category_segment(old_category)
    if segment is not None:
        segment['name'] = 'Interested in ' + new_category.name
        segment['segmentRule'] = generate_segment_rule_for_category(new_category)
        segment_id = str(segment['sid'])
        response = api_exchange(PUT, configs.AAM_SEGMENT_API_PATH + '/' + segment_id, segment)
        segment = response.json()
        print 'AAM segment update:', segment
    else:
        segment = create_category_segment(new_category)
    return segment
```

26. We will add these methods in our **`app/models.py`** similar to the previous exercises. In this case, we will call the **`update_category_segment`** function whenever a new Product is created, or an existing Category is updated.

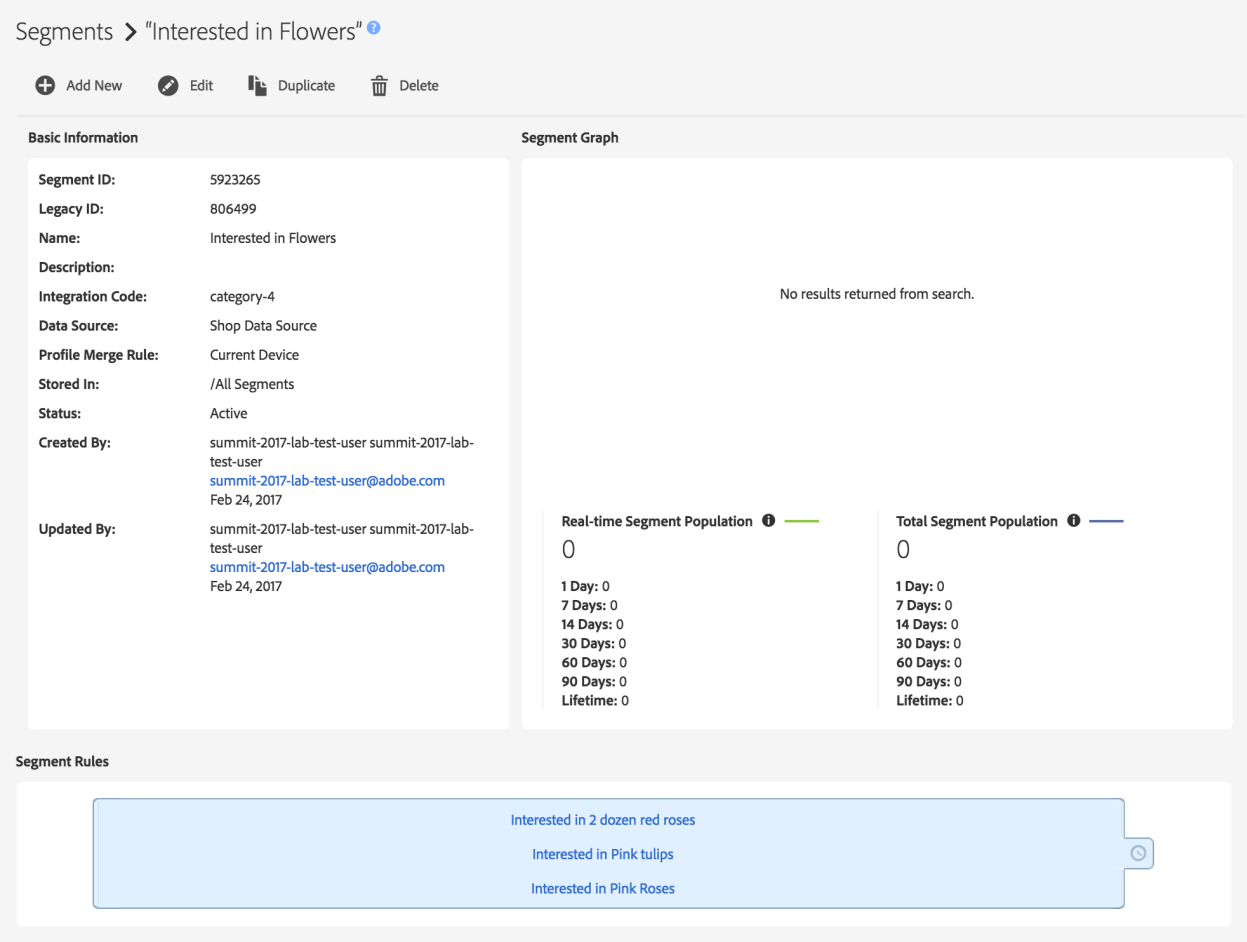
Saving an existing Category:

```
def save(self, *args, **kwargs):
    if not self.id:
        # Category is being created
        super(Category, self).save(*args, **kwargs)
        aam.create_category_trait_folder(self)
    else:
        # Category is being updated
        old_category = Category.objects.get(id=self.id)
        super(Category, self).save(*args, **kwargs)
        aam.update_category_trait_folder(old_category, self)
        aam.update_category_segment(old_category, self)
```

Creating a new Product:

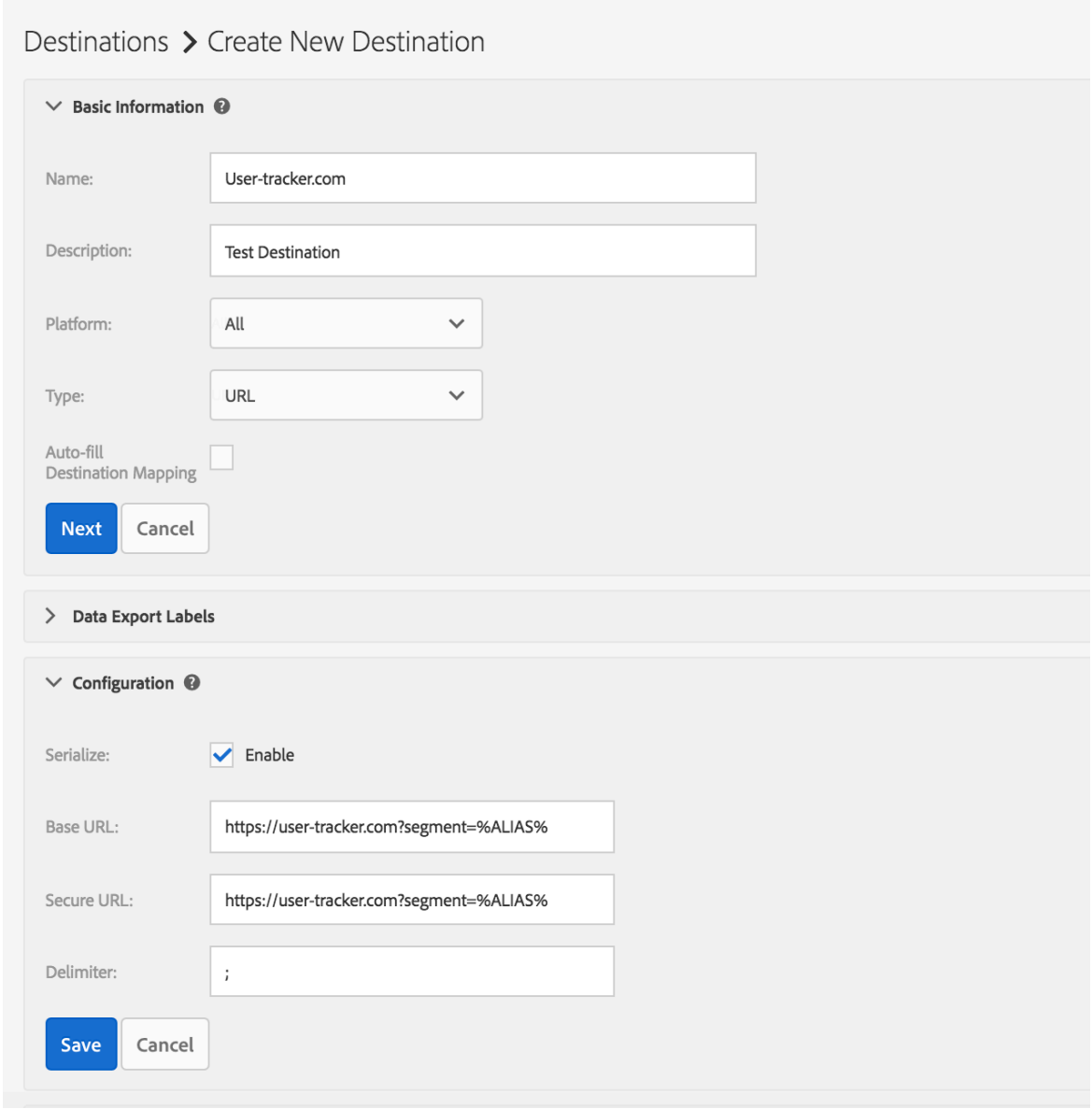
```
def save(self, *args, **kwargs):
    if not self.id:
        # Product is being created
        super(Product, self).save(*args, **kwargs)
        aam.create_product_trait(self)
        aam.update_category_segment(self.category, self.category)
    else:
        # Product is being updated
        old_product = Product.objects.get(id=self.id)
        super(Product, self).save(*args, **kwargs)
        aam.update_product_trait(old_product, self)
```

Example of auto created Segment in Audience Manager:

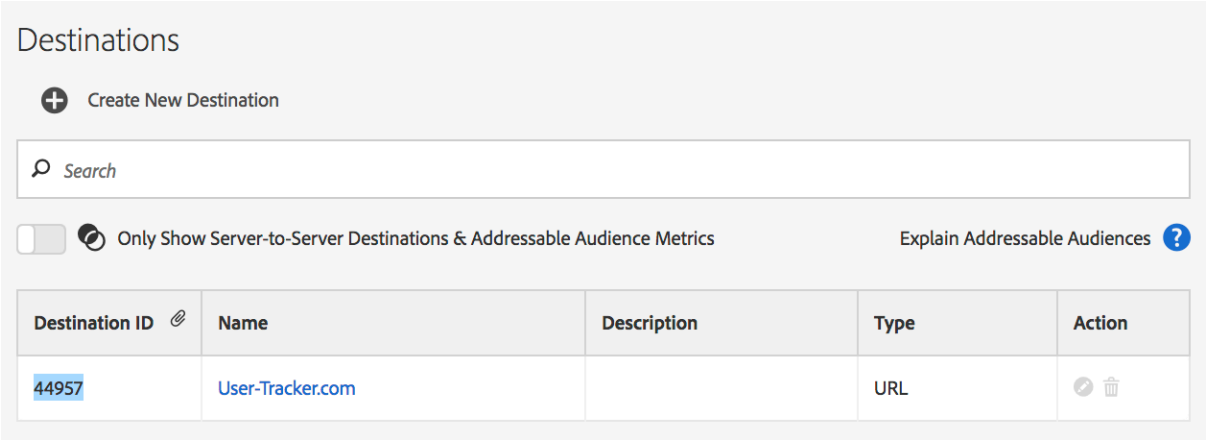


Exercise-07: Create a URL Destination in AAM UI and then map the Segments automatically from our sample app for real-time activation

28. We want to let a company we partner with know whenever a visitor qualifies for a category segment. We will give them our category id for the visitor. We will first create a URL type destination in AAM UI to receive segment activations. Each time a visitor qualifies for a segment, we'll send the visitor and segment information to this destination.



29. Visit the destination list page in AAM and copy the id of the created destination:



Add this destination id in **`config/aam_config.py`**

```
# Data Elements
AAM_DESTINATION_ID = '44957'
```

30. Add a method in **`app/aam.py`** to auto map a segment to this destination:

```
def map_segment_to_destination(segment_id):
    destination_id = configs.AAM_DESTINATION_ID

    response = api_exchange(GET, configs.AAM_DESTINATION_API_PATH+ '/' + destination_id + '/mappings')
    if successful(response):
        mappings = response.json()
        for mapping in mappings:
            if mapping['sid'] == segment_id:
                return mapping

    mapping = {
        "traitType": "SEGMENT",
        "sid": segment_id,
        "startDate": "2017-02-14",
        "traitAlias": segment_id
    }
    response = api_exchange(POST, configs.AAM_DESTINATION_API_PATH+ '/' + destination_id + '/mappings', mapping)
    mapping = response.json()

    print 'AAM segment mapping create:', mapping
    return mapping
```

31. Now modify previously created **`create_category_segment`** and **`update_category_segment`** methods in **`app/aam.py`** and add this line just above the return statements in both methods.

```
map_segment_to_destination(segment['sid'])
return segment
```

32. Go back to Shop Administration and modify one of the existing Categories and hit save. You will notice in the terminal logs that an API call was made to auto map the Category segment to the URL destination. If you visit AAM destinations tab and open the URL destination, you will see the auto created mapping information.

Segments Mapped to User-Tracker.com

Segment ID	Name	Description	Segment Status	Start Date	End Date	Mapping
5923258	Interested in Category 1		Active	02/14/2017		5923258

Self-guided next steps

- Create separate traits for visitors who have clicked, searched, added to cart or bought a product.
- Use Visitor API or Data Integration Library to send visitor activity to Audience Manager.
- Use Declared ID for authenticated activity.
- Join with traits from other sources like Audience Marketplace and 3rd Party data sharing to create richer segments.
- Display metrics on a category or product trait or segment.
- Handle access token expiration.
- Store Audience Manager credentials encrypted.
- Use a dedicated API user that is separate from UI user.