Create Django Proxy Services Of Cloud Functions

Estimated time needed: 150 minutes

In previous labs, you created car model and car make Django models residing in a local SQLite repository.
You also created dealer and review models in a remote Cloudant repository (served by IBM Cloud Function actions).

Now, you need to integrate those models and services to manage all entities such as dealers, reviews, and cars, and present the results in Django templates.

To integrate external dealer and review data, you need to call the cloud function APIs from the Django app and process the API results in Django views. Such Django views can be seen as proxy services to the end user because they fetch data from external resources per users' requests.

In this lab, you will create such Django views as proxy services.

Environment setup

If your Theia workspace has been reset and you want to continue from what you have done previously, please `git clone` or pull from your created GitHub repository.

- Set up the Python runtime if Theia workspace has been reset.

```
1
```

```
python3 -m pip install -U -r requirements.txt
```

Copied!

Note: You may need to perform models migrations for a new Theia environment.

Create a `get_dealerships` Django view to get dealer list

In the previous lab, you would have created a cloud function service called `dealer-get` to return a list of dealerships. Next, let's see how to call that `dealer-get` service from the Django app.

Before you learn how to make REST calls in Django, let's create a dealer model in `models.py` to represent and store a dealer entity.

- Open `/models.py`, add a `CarDealer` class. Note that this is a plain Python class instead of a subclass of Django model.

An instance of `CarDealer` is used as a plain data object for storing a dealer object returned from `dealer-get` service:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

```
class CarDealer:

    def __init__(self, address, city, full_name, id, lat, long, short_name, st, zip):
        # Dealer address
        self.address = address
        # Dealer city
        self.city = city
```

```
        # Dealer Full Name
        self.full_name = full_name
        # Dealer id
        self.id = id
        # Location lat
        self.lat = lat
        # Location long
        self.long = long
        # Dealer short name
        self.short_name = short_name
        # Dealer state
        self.st = st
        # Dealer zip
        self.zip = zip

    def __str__(self):
        return "Dealer name: " + self.full_name
```

Copied!

The actual instance attributes may be different from the object returned by the service you created. Update them in above `CarDealer` class accordingly.

Now we can start calling `review-get` cloud function service and load the JSON results into a list of `CarDealer` objects.

There are many ways to make HTTP requests in Django.
Here we use a very popular and easy-to-use Python library called `requests`.

- Create a new Python file called `restapis.py` under `djangoapp/` folder and add a sample `get_request` method:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

```
import requests
import json
from .models import CarDealer
from requests.auth import HTTPBasicAuth

def get_request(url, **kwargs):
    print(kwargs)
    print("GET from {} ".format(url))
    try:
        # Call get method of requests library with URL and parameters
        response = requests.get(url, headers={'Content-Type': 'application/json'},
                                params=kwargs)
    except:
        # If any error occurs
        print("Network exception occurred")
    status_code = response.status_code
    print("With status {} ".format(status_code))
    json_data = json.loads(response.text)
    return json_data
```

Copied!

The `get_request` method has two arguments, the URL to be requested, and a Python keyword arguments representing all URL parameters to be associated with the get call.

The `requests.get(url, headers={'Content-Type': 'application/json'}, params=kwargs)` calls a `GET` method in `requests` library with a URL and any URL parameters such as `dealerId` or `state`.

The content of the response will be a JSON object to be consumed by other methods such as a Django view.

Next, let's parse the dealership JSON result returned by the `get_request` call.

- Create a `get_dealers_from_cf` method to call `get_request` and parse its JSON result.
  One example method may look like the following:

```
1
2
3
4
5
```

```
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

```
def get_dealers_from_cf(url, **kwargs):
    results = []
    # Call get_request with a URL parameter
    json_result = get_request(url)
    if json_result:
        # Get the row list in JSON as dealers
        dealers = json_result["rows"]
        # For each dealer object
        for dealer in dealers:
            # Get its content in `doc` object
            dealer_doc = dealer["doc"]
            # Create a CarDealer object with values in `doc` object
            dealer_obj = CarDealer(address=dealer_doc["address"], city=dealer_doc["city"], full_name=dealer_doc["full_name"],
                                   id=dealer_doc["id"], lat=dealer_doc["lat"], long=dealer_doc["long"],
                                   short_name=dealer_doc["short_name"],
                                   st=dealer_doc["st"], zip=dealer_doc["zip"])
            results.append(dealer_obj)

    return results
```

Copied!

You can see parsing JSON in Python is very similar to accessing data with Python dictionary (key-value pair).
You just need to get values from keys. A value could be an object or a collection of objects such as list or dictionary.

Next, let's create a Django view to call `get_dealers_from_cf` and return the result as a HTTPResponse to browser.

- Find the `get_dealerships` view method in `djangoapp/views.py`, update the method with following:

```
1
2
3
4
5
6
7
8
9
```

```
def get_dealerships(request):
    if request.method == "GET":
        url = "your-cloud-function-domain/dealerships/dealer-get"
        # Get dealers from the URL
        dealerships = get_dealers_from_cf(url)
        # Concat all dealer's short name
        dealer_names = ' '.join([dealer.short_name for dealer in dealerships])
        # Return a list of dealer short name
        return HttpResponse(dealer_names)
```

Copied!

- Configure the route for `get_dealerships` view method in `url.py`:

```
1
2
3
4
5
6
7
8
9
10
11
12
```

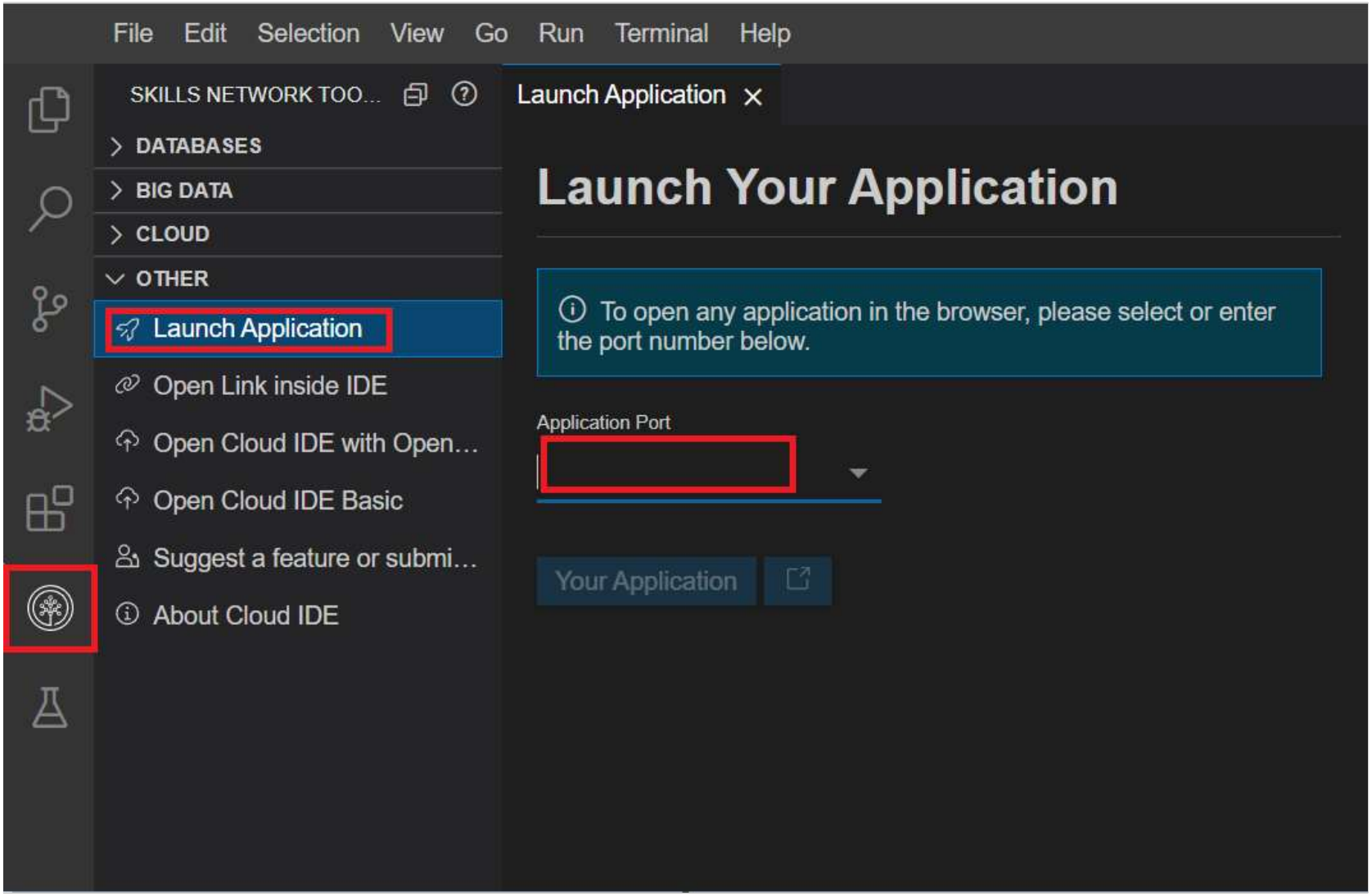
```
from django.urls import path
from django.conf.urls.static import static
from django.conf import settings
from . import views

app_name = 'djangoapp'
urlpatterns = [
    # route is a string contains a URL pattern
    # view refers to the view function
    # name the URL
    path(route='', view=views.get_dealerships, name='index')
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Copied!

- Test the `get_dealerships` view in Theia

To do this, click on the Skills Network button on the right, it will open the "Skills Network Toolbox". Then click `OTHER` then `Launch Application`. From there you should be able to enter the port as `8000` and launch the development server.



- Go to `https://userid-8000.theiadocker-1.proxy.cognitiveclass.ai/djangoapp`.
  You should see a list of dealership names.

More details about how to create Django view and configure URL can be found in this lab:

Views and templates

You will find more detailed references about `requests` package at the end of this lab.

Coding practice: create a `get_dealer_by_id` or `get_dealers_by_state` method in `restapis.py`.
HINT, the only difference from the `get_dealers_from_cf` method is adding a dealer id or state URL parameter argument when calling the `def get_request(url, **kwargs):` method such as `get_request(url, dealerId=dealerId)`.

Create a Django `get_dealer_details` view to get reviews of a dealer

By now, you should understand how to call a cloud function using `requests` library in Django and convert the JSON results into Python objects.

Next, let's create another `get` call to the `revew-get` cloud function to get reviews for a dealer.

- Define a `DealerReview` class in `models.py`, it may contain the following attributes:
  - dealership
  - name
  - purchase
  - review
  - purchase_date
  - car_make
  - car_model
  - car_year
  - sentiment
  - id

The value of `sentiment` attribute will be determined by Watson NLU service. It could be `positive`, `neutral`, or `negative`.
You will make a Watson NLU call in the next step.

- Create a `get_dealer_reviews_from_cf` method in `restapis.py`. It makes a `get_request(url, dealerId=dealer_id)` method call to get all reviews by dealer's id. Then it converts the JSON result into a list of `DealerReview` objects.

- Define a `def get_dealer_details(request, dealer_id):` view method in `views.py` to call `get_dealer_reviews_from_cf` method in `restapis.py`,
  and append a list of reviews to context and return a `HttpResponse`, similar to the dealer names in previous step.

Here we need to define `dealer_id` argument to specify which dealer you want to get reviews from.

- Configure the route for `get_dealer_details` view in `url.py`.
  For example, `path('dealer/<int:dealer_id>/', views.get_dealer_details, name='dealer_details'),`.

- Test the `get_dealer_details` view in Theia by launching the application with the development server on port `8000` as done earlier.
  You should see a list of reviews for a specific dealer.

Update the `get_dealer_reviews_from_cf` view to call Watson NLU for analyzing the sentiment/tone for each review

Now that you get a list of reviews for a dealer, you could use Watson NLU to analyze their sentiment/tone.
Since Watson NLU is not public, you will need to add authentication argument to `requests.get()` method.

- Open `restapis.py`, update `get_request(url, **kwargs)` by providing an `auth` argument
  with an API key you created in IBM Watson NLU.

```
1
2
```

```
requests.get(url, params=params, headers={'Content-Type': 'application/json'},
                                    auth=HTTPBasicAuth('apikey', api_key))
```

Copied!

You may use a `if` statement to check if `api_key` was provided and call `requests.get()` differently.

```
1
2
3
4
5
6
```

```
 if api_key:
   # Basic authentication GET
   request.get(url, params=params, auth=, ...)
 else:
   # no authentication GET
   request.get(url, params=params)
```

Copied!

- In `restapis.py` file, create a new `analyze_review_sentiments(dealerreview)`.

In the method, make a call to the updated `get_request(url, **kwargs)` method with following parameters:

```
1
2
3
4
5
6
7
8
```

```
9. 9
```

```
1. ...
2.   params = dict()
3.   params["text"] = kwargs["text"]
4.   params["version"] = kwargs["version"]
5.   params["features"] = kwargs["features"]
6.   params["return_analyzed_text"] = kwargs["return_analyzed_text"]
7.   response = requests.get(url, params=params, headers={'Content-Type': 'application/json'},
8.                                   auth=HTTPBasicAuth('apikey', api_key))
9. ...
```

Copied!

You can find more detailed references about Watson NLU text analysis at the end of this lab.

- In `restapis.py` file, update `get_dealer_reviews_from_cf` method by assigning the Watson NLU result to the `sentiment` attribute of a `DealerReview` object:

```
1. 1
2. 2
```

```
1. ...
2. review_obj.sentiment = analyze_review_sentiments(review_obj.review)
```

Copied!

- Update `get_dealer_details` view to also print sentiment for each review:

- Test the updated `get_dealer_details` view in Theia by launching the application with the development server on port `8000` as done earlier.

Create a `add_review` Django view to post a dealer review

By now you have learned how to make various GET calls.

Next, you will be creating a POST call to the `review-post` cloud function to add a review to a specific dealer.

- Open `restapis.py`, create a new `post_request(url, json_payload, **kwargs):` method.
  The key statement in this method is calling `post` method in `requests` package.

For example, `requests.post(url, params=kwargs, json=json_payload)`.

The key difference from the `get()` method is you need to add a `json` argument with a Python dictionary-like object as request body.

- Open `views.py`, create a new `def add_review(request, dealer_id):` method to handle review post request.

- In the `add_review` view method:

  - First check if user is authenticated because only authenticated users can post reviews for a dealer.
  - Create a dictionary object called `review` to append keys like (`time`, `name`, `dealership`, `review`, `purchase`) and any attributes you defined in your `review-post` cloud function.

  For example:

  ```
  1. 1
  2. 2
  3. 3
  1. review["time"] = datetime.utcnow().isoformat()
  2. review["dealership"] = 11
  3. review["review"] = "This is a great car dealer"
  ```

  Copied!

  - Create another dictionary object called `json_payload` with one key called `review`. Like `json_payload["review"] = review`. The `json_payload` will be used as the request body.
  - Call the `post_request` method with the payload

  `post_request(url, json_payload, dealerId=dealer_id)`.

  - Return the result of `post_request` to `add_review` view method. You may print the post response in console or append it to HTTPResponse and render it on browser.

- Configure the route for `add_review` view in `url.py`.

For example, `path('dealer/<int:dealer_id>/', views.get_dealer_details, name='dealer_details')`.

- Test the `add_review` view in Theia. So when you make an add review post request, the `add_view` method will create a JSON payload contains a review object and post it to your `review-post` cloud function action.

Commit your updated project to GitHub

Commit all updates to the GitHub repository you created so that you can save your work.

If you need to refresh your memory on how to commit and push to GitHub in Theia lab environment, please refer to this lab Working with git in the Theia lab environment

External References

- Requests Developer Interface
- Watson NLU API Reference

Summary

In this lab, you have learned how to create proxy services to call the cloud functions in Django,
convert their JSON results into Python objects such as `CarDealer` or `DealerReview`, and return the objects as a HTTPResonse.

In the next lab, you will create Django templates to present those objects.

Author(s)

Yan Luo

Other Contributor(s)

Upkar Lidder

Priya

Changelog

Date	Version	Changed by	Change Description
2021-02-22	1.0	Yan Luo	Created new instructions for Capstone project
2022-09-01	1.1	K Sundararajan	Updated `Launch Application` instructions as per the new Theia IDE
2022-09-20	1.2	K Sundararajan	Updated pip (packages installation) command