Demo Game!!

To demonstrate the functionality of the service, we’ll setup a hypothetical scenario of developer and some test users who would like to receive notifications. Here are our players:

Gil Designer : Gil is a LinkedIn developer focusing on user components. He would like to send a push notification to a user when their profile is viewed by a recruiter.

David Developer : David is a developer at a large tech company who is happy at his job, but he’s always on the lookout for new and exciting opportunities.

Shensi Founder : Shensi is a founder at a new start-up looking browsing LinkedIn for engineers to help build her company.

Follow the tutorial to help Gil setup a module he will build to send a push notification when people view each other’s profile for the first time.

Step 1: Setup

Before starting with Gil, we need to setup our system first. Run the DB, upload the schema, and run the API server following the instructions. When the app is up and running, lets add our two users to our mock LinkedIn platform:

Run this API call for the two payloads:

POST http://0.0.0.0:5000/person

```
{“username”: “daviddev”, “first_name”: ‘David”, “last_name”: “Developer”}
{“username”: “shensifound”, “first_name”: ‘Shensi”, “last_name”: “Founder”}
```

You should get a response with 2 person_id ’s back. Save those for later

Step 2: Gil Creates Service Type

Now that our users are in the system we can help Gil start sending notifications. The notification service needs to know what kind of notification to send, so Gil needs to create a new **NotificationType** in the admin portal of LinkedIn. To make this, lets first look at the required parameters:

source_entity_name : The root entity name that will be the source entity of the notification

notification_description: Description of the type of notification this is

entity_redirect_url: redirect root application url that Notification service will redirect on a notification click

After reading the docs, Gil sees that the key to configuring a notification is thinking about his source, and where he wants a user to be redirected when they click the notification. Since he is pushing a notification for a person viewing another's profile, he wants the notification to point at the viewer. Becasue he works on this entity, he knows its the **Person** entity, and that the root uri endpoint for each person's view is just **/person.** That means his notification type is going to look like:

```
{
    "source_entity_name": "Person",
    "notification_description": “Someone views another’s profile",
    "entity_redirect_uri": "/person"
}
```

```
Send this payload via POST http://0.0.0.0:5000/admin/notificationtype
```

Save the notification_type_id that you get back and move to the next step.

Step 3: Notifications Start Flying!

Now that Gil has created his notification type, he's read the rest of the docs and realized its as easy as a single API call including a message, who should get it, and what notification type it is.

He builds a module in his own Person views tracking system that gets alerted when someone views a profile for the first time.

Meanwhile, after this feature has been built, Shensi is looking around LinkedIn for qualified candidates to work at her new startup. As she's searching, she comes across a candidate she has never seen before, David Developer.

To mimic Gil's profile viewer automation, run the following API call to create a notification

POST http://0.0.0.0:5000/notification

```
{
    "notification_type_id": <from_above>,
    "source_id": <person_id (Shensi)>,
    "message": "David, Shensi viewed your profile for the first 
time!",
    "recipients":[
        {
            "username": "daviddev",
            "user_type": "Person",
            "user_id": <person_id (David)>
        }
    ]
}
```

To understand the payload that Gil sent, the notification_type_id is from the type we created in step 2, The Source ID is Shensi's person_id becasue she is the source entity. For this type of notification theres only one recipient, who we've made David

**Note when you send this call you should see a broker message from the dummy Queuing service built into the app

Step 3: View a Notification

David is sitting around working when he gets a ping from LinkedIn. Logging into his page he can see that he has a new notification. CLicking to his notification page he can see an unread notification.

To mimic the Notification page UI, run the following API call:

GET http://0.0.0.0:5000/user/notifications/<user_id (David)>

This directs to an API endpoint that combines some data into single viewable models that can be queued by user. This can be expanded obviously to deal with load and many other observations, but this is just the simplest version for now. You should see the notification Gil built to automatically push show up:

```
[
    {
        "recipient_id": “<person_id>(David)”,
        "record_id": "3",
        "message": "David, Shensi viewed your profile for the
first time!",
        "username": “daviddev”
    }
]
```

Step 4: Click a Notification

Finally, we'll show how the notification service provides source back routing configured by Gil in the NotificationType. When David looks at this notifications list he sees that Shensi has viewed his profile! He's not sure if he knows who this is, but he also knows his memory isn't great so he would like to see Shensi's profile if it will jog his memory.

To simulate David clicking the notification, run the following API call with the **record_id** parameter we can see in step 3

GET http://0.0.0.0:5000/click/notification/3

You should see this payload back:

```
{
    "username": "shensifound",
    "person_id": <person_id(Shensi),
    "last_name": "Founder",
    "first_name": "Shensi"
}
```

If you'll notice, this is nothing like a notification! Thats becasue the NotificationType that Gil configured in Step 2 included an HTTP redirect uri that is triggered from the /click endpoint along with the **source_id** configured in the corresponding **Notification** originally sent by Gil's automation