

---

# **vtc-video-link-post**

***Release 1.0***

**MBAMBA FABRICE DAMIEN**

**Sep 14, 2020**



## TABLE OF CONTENT:

<b>1</b>	<b>Report of task project</b>	<b>3</b>
<b>2</b>	<b>Get Started</b>	<b>5</b>
2.1	Python Installation . . . . .	5
2.2	PostgresSql Installation . . . . .	5
2.3	Create User and Database . . . . .	6
2.4	Run vtcvlp app . . . . .	9
<b>3</b>	<b>Launching the app</b>	<b>11</b>
3.1	First time or not . . . . .	11
<b>4</b>	<b>Documentation</b>	<b>17</b>
4.1	The CustomUser class . . . . .	17
4.2	The VideoLink class . . . . .	18
4.3	The CustomUserViewSet class . . . . .	19
4.4	The VideoLinkViewSet class . . . . .	20
4.5	The CustomUserSerializer class . . . . .	20
4.6	The VideoLinkSerializer class . . . . .	21
4.7	Views modules: . . . . .	21
	<b>Python Module Index</b>	<b>23</b>
	<b>Index</b>	<b>25</b>



This document describes how to launch the VTC-VLP application step by step on a local web server.



## REPORT OF TASK PROJECT

Name: VTC - vlp

version: v1.0

Description: It a simple app that help user to create an account, logged in and store videos links.

**Register informations:**

- Name
- User name
- Email
- Password

**Log in information:**

- Username
- Password

Time taken: 2 Hours and 48 minutes

Technologies used: Python 3, Javascript, CSS and HTML

Team: I'm alone (MBAMBA Fabrice Damien)





## GET STARTED

### 2.1 Python Installation

Before use app locally you must install python 3 according to your system requirement.

**Example:**

- If you have windows installed and it 32 bit you must install python 3 32 bits version (Same method if you have 64 bits version)
- If you have Mac you probably have python installed, if not choose mac os python 3 version
- Same if you have Linux

Now to get download python build visit: <http://www.python.org/download/> and check with your system os correspondance

When terminate download install and take care to check add parameter of installation directory to your system PATH

If all done, Open your console and set:

```
> python
```

You will get something like this:

```
Python 3.7.8 (tags/v3.7.8:4b47a5b6ba, Jun 28 2020, 08:53:46) [MSC v.1916 64 bit_
↪ (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

It's all python is installed now correctly, but if you get an error message make a post in stackoverflow forum and you get very fast answer.

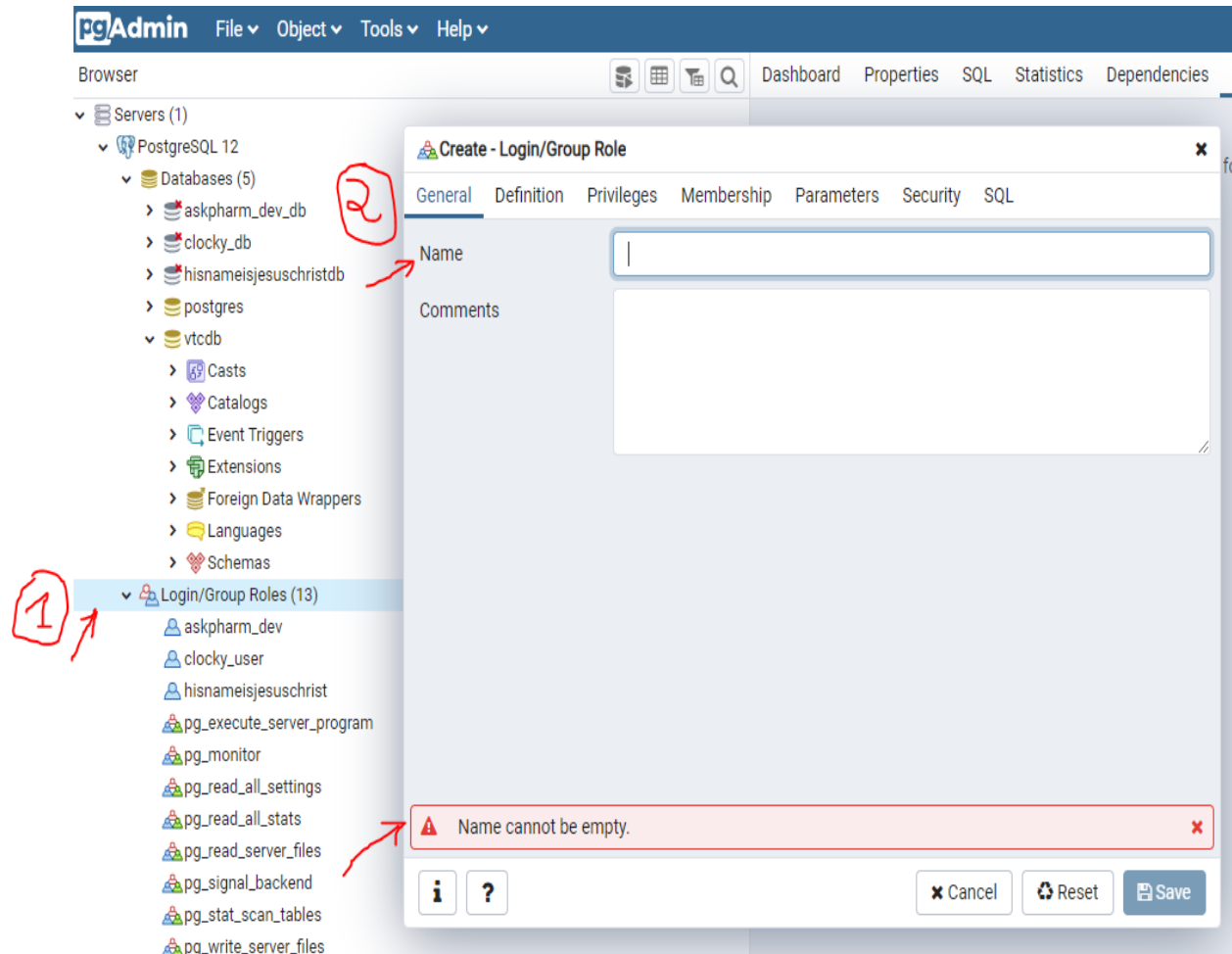
### 2.2 PostgresSql Installation

Now python is install, you must install postgresSql similary to python according to your system requirement. To do it browse this and select your according package: <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>

## 2.3 Create User and Database

Now we have python and postgres installed we must create Database manually and user of this database(It called role)  
The most efficiently tool have postgres is pgadmin-browser with it we are going fast. To open pgadmin into your Windows start menu(If you use windows) and select postgres after pgadmin

When open into your browser first select option: Login/Group Roles Like image bellow:



Now you must fill field like this:

name: vtcuser

Select definition tab into pgadmin

**Create - Login/Group Role**

General | **Definition** | Privileges | Membership | Parameters | Security | SQL

Password:

Account expires: YYYY-MM-DD HH:mm:ss Z

Connection limit: -1

**Name cannot be empty.**

and fill field password like so:

password: vtcpass

Last tab select privileges like image:

**Create - Login/Group Role**

General Definition **Privileges** Membership Parameters Security SQL

Can login?	<input type="checkbox"/> No	←
Superuser?	<input type="checkbox"/> No	←
Create roles?	<input type="checkbox"/> No	
Create databases?	<input type="checkbox"/> No	
Update catalog?	<input type="checkbox"/> No	
Inherit rights from the parent roles?	<input checked="" type="checkbox"/> Yes	
Can initiate streaming replication and backups?	<input type="checkbox"/> No	

⚠ Name cannot be empty. ✕

**i ?** **✕ Cancel** **🔄 Reset** **💾 Save**

Check case Can login ? and Superuser

Now the final piece into postgres is to create our database to make it really simple

Select first database option push your right click and select create

New windows appear, first thing to do is to filled field like so:

database: vtddb

Owner Select vtcuser

We have python install and database next the last peace.

Open vtcvlp directory inside your console like this:

If our first time to run app, we need to add requirements library

Open vtcvlp directory inside your console like this:

Secondly we need to populate our database for do it

```
C:\Users\personal\vtcvlp\ python manage.py migrate
```

Finally

Run command `python manage.py runserver` like this:

```
C:\Users\personal\vtcvlp\ python manage.py runserver
```

Now to use app we browse it with every browser inside it, in url place we set:

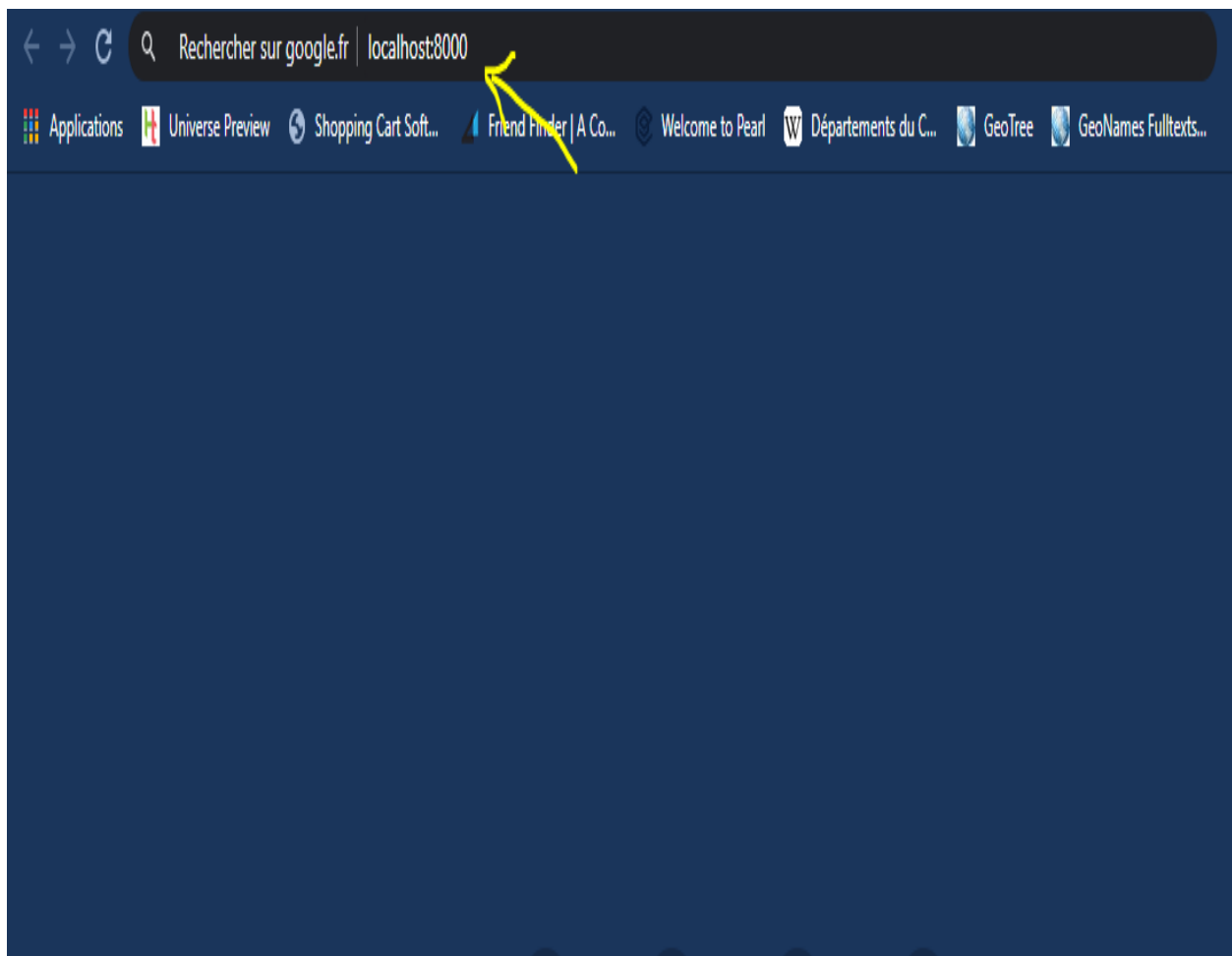
`localhost:8000`

Now we can use it.

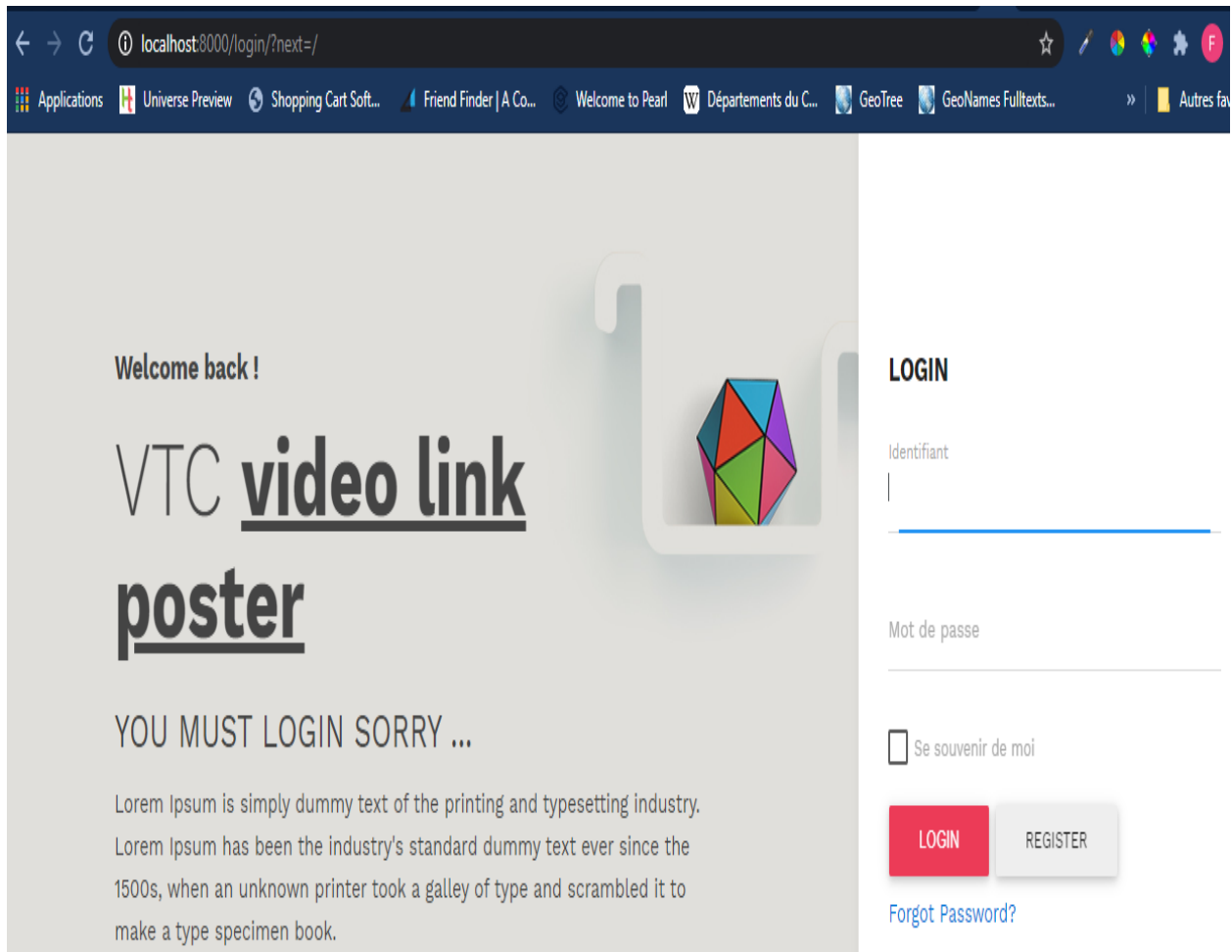
## LAUNCHING THE APP

### 3.1 First time or not

If this is your first time to launch the application, then know that you should first launch your browser example for me it's chrome and once launched go to the urls area and type: *localhost:8000*



Once entered press the Enter key. That's it.



You should register first before you can post a link to click on the register button.



Welcome

# VTC video link poster

YOU MUST SIGN UP AND GET YOUR OWN ACCOUNT TO START

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

[NEED HELP READ THE DOC](#) [API](#)

## REGISTER

E-mail

Pseudonyme

Nom

Mot de passe

Mot de passe (à nouveau)

THAT'S ALL

I ALREADY HAVE ACCOUNT

Once the registration form is correctly filled in go to your console and select the generated registration confirmation link because remember you are on a server running locally.

```
Content-Type: text/plain; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: 8bit
Subject: [example.com] Veuillez confirmer votre adresse e-mail
From: webmaster@localhost
To: poster@vtc.cm
Date: Sun, 13 Sep 2020 22:44:46 -0000
Message-ID: <160003708627.3132.4344356810501022025@DESKTOP-ETMA2JU>
```

Salut !

Vous recevez cet e-mail parce que l'utilisateur poster a donné votre adresse e-mail pour se connecter à son compte.

Pour confirmer l'exactitude de cette information, cliquez sur <http://7fc55807b738.ngrok.io/confirmation/Nw:1kHajm:OdvgK5uWbNu39sLBzP1kDFLggU/>

Cordialement !

Paste this link on the URL field and press Enter. Then click on the button to confirm your registration Now login with your username or e-mail address and then your password and click on the login button.

Welcome back !

# VTC video link poster

YOU MUST LOGIN SORRY ...

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

[NEED HELP READ THE DOC](#) [API](#)

## LOGIN

Identifiant

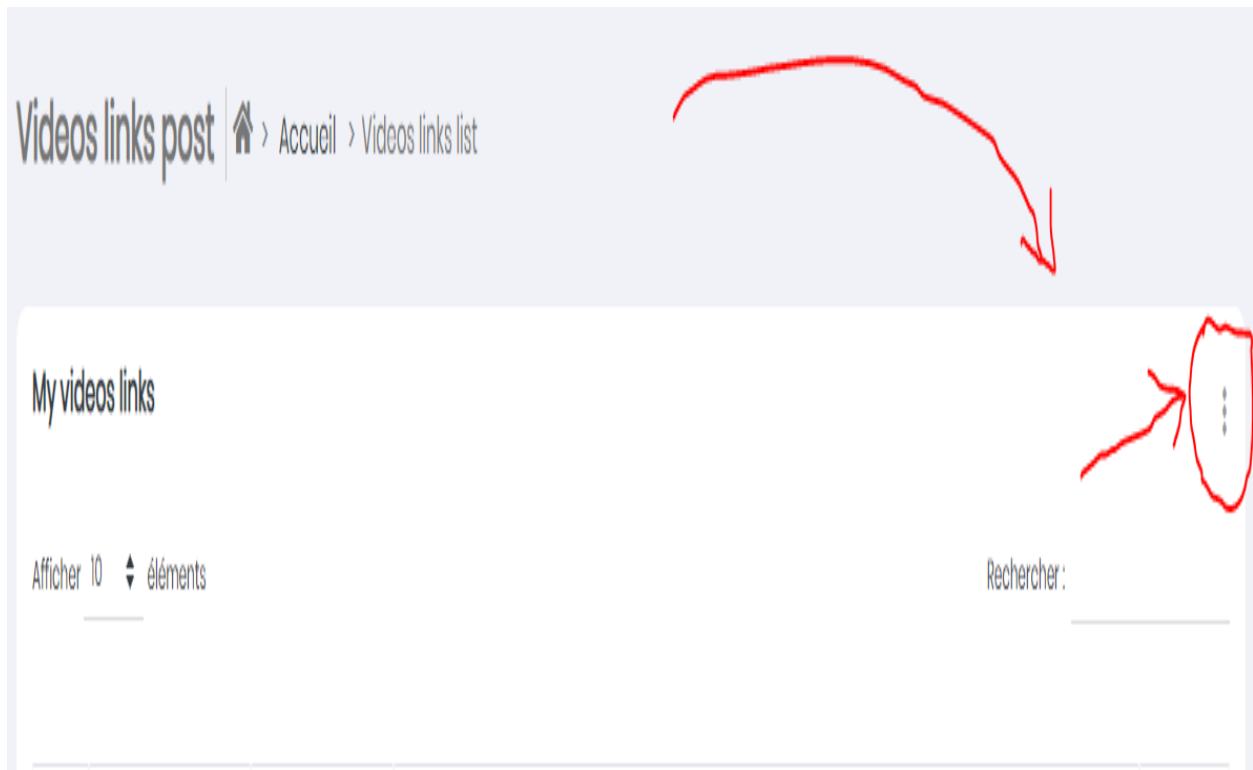
Mot de passe

☐ Se souvenir de moi

[Forgot Password?](#)

[LOGIN](#) [REGISTER](#)

Now that you are connected, click on the three vertical buttons on your right you will have a modal window then finally indicate the name of your link and its address, finally click on the save button. Now you have just recorded your first video link. That's all you have to do



## 4.1 The CustomUser class

**class** vtcuser.models.CustomUser (\*args, \*\*kwargs)

Bases: django.db.models.base.Model

Custom User class :param models.Model: models.Model class; :type user: User;

### Methods

---

### Attributes

---

**exception DoesNotExist**

Bases: django.core.exceptions.ObjectDoesNotExist

**exception MultipleObjectsReturned**

Bases: django.core.exceptions.MultipleObjectsReturned

**create\_user\_profile** (instance, created, \*\*kwargs)

Callable function for create user profile when create account :param kwargs: Any other param :type kwargs: dict :param sender: User object :type sender: User Object :param instance: CustomUser :type instance: CustomUser object :param created: Return True if creation else return False :type created: bool: True or False :return None :rtype None object

**id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**objects** = <django.db.models.manager.Manager object>

**user**

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant (Model):  
    place = OneToOneField(Place, related_name='restaurant')
```

Restaurant.place is a ForwardOneToOneDescriptor instance.

**user\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**videolink\_set**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

## 4.2 The VideoLink class

```
class vtcuser.models.VideoLink(*args, **kwargs)
```

Bases: `django.db.models.base.Model`

VideoLink object :param models.Model: models.Model class :type: name: str, required :type link: Url object, required :type date\_add: DateTime object, default;

### Methods

---

### Attributes

---

**exception DoesNotExist**

Bases: `django.core.exceptions.ObjectDoesNotExist`

**exception MultipleObjectsReturned**

Bases: `django.core.exceptions.MultipleObjectsReturned`

**date\_add**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**get\_api\_url** (*request=None*)

**get\_next\_by\_date\_add** (\*, field=<django.db.models.fields.DateTimeField: date\_add>, is\_next=True, \*\*kwargs)

**get\_previous\_by\_date\_add** (\*, field=<django.db.models.fields.DateTimeField: date\_add>, is\_next=False, \*\*kwargs)

**id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**link**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is

executed.

**name**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**objects** = <django.db.models.manager.Manager object>

**property owner**

**poster**

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

**poster\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

## 4.3 The CustomUserViewSet class

**class** vtcuser.api.CustomUserViewSet (\*\*kwargs)

CustomUserViewSet class :param viewsets.ModelViewSet A viewset that provides default *create()*, *retrieve()*, *update()*, *partial\_update()*, *destroy()* and *list()* actions. :type serializer\_class: class: *CustomUserSerializer* :type queryset: class: *Queryset*

### Methods

—

### Attributes

—

**queryset** = <QuerySet [<CustomUser: fly@fly.fr>, <CustomUser: messi>, <CustomUser: v

**serializer\_class**

alias of *vtcuser.serializers.CustomUserSerializer*

## 4.4 The VideoLinkViewSet class

```
class vtcuser.api.VideoLinkViewSet (**kwargs)
    Bases: rest_framework.viewsets.ModelViewSet
```

VideoLinkViewSet class :param viewsets.ModelViewSet A viewset that provides default *create()*, *retrieve()*, *update()*, *partial\_update()*, *destroy()* and *list()* actions. :type serializer\_class: class: *VideoLinkSerializer* :type queryset: class: *Queryset*

### Methods

---

### Attributes

---

```
queryset = <QuerySet [<VideoLink: Vtc link>, <VideoLink: It work ?>, <VideoLink: No
serializer_class
    alias of vtcuser.serializers.VideoLinkSerializer
```

## 4.5 The CustomUserSerializer class

```
class vtcuser.serializers.CustomUserSerializer (*args, **kwargs)
    Bases: rest_framework.serializers.ModelSerializer
```

CustomUserSerializer class A *ModelSerializer* is just a regular *Serializer*, except that: \* A set of default fields are automatically populated. \* A set of default validators are automatically populated. \* Default *.create()* and *.update()* implementations are provided.

### Methods

---

### Attributes

---

```
class Meta
    Bases: object
    fields = '__all__'
    model
        alias of vtcuser.models.CustomUser
```



## 4.6 The VideoLinkSerializer class

```
class vtcuser.serializers.VideoLinkSerializer(*args, **kwargs)
```

Bases: `rest_framework.serializers.ModelSerializer`

CustomUserSerializer class A *ModelSerializer* is just a regular *Serializer*, except that: \* A set of default fields are automatically populated. \* A set of default validators are automatically populated. \* Default *.create()* and *.update()* implementations are provided.

### Methods

—

### Attributes

—

```
class Meta
```

Bases: `object`

```
fields = ['uri', 'id', 'name', 'link', 'poster']
```

```
model
```

alias of `vtcuser.models.VideoLink`

```
read_only_fields = ['id', 'poster']
```

```
get_uri(obj)
```

```
validate_name(value)
```

Validate a name of link given are uniq :param value: name of link to check :rtype str

## 4.7 Views modules:

VTC user video link posted app views.

```
class vtcuser.views.VideoLinkAPIView(**kwargs)
```

Bases: `rest_framework.mixins.CreateModelMixin`, `rest_framework.generics.ListAPIView`

VideoLinkPostRudView class :param class: `generics.RetrieveUpdateDestroyAPIView` :type class: `generics.RetrieveUpdateDestroyAPIView`

```
authentication_classes = [<class 'rest_framework_jwt.authentication.JSONWebTokenAuthen
```

```
get_queryset()
```

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

```
lookup_field = 'pk'
perform_create(serializer)
post(request, *args, **kwargs)

serializer_class
    alias of vtcuser.serializers.VideoLinkSerializer

class vtcuser.views.VideoLinkAPIView(**kwargs)
    Bases: rest_framework.views.APIView
    VideoLinkAPIView class :param class: APIView :type class: APIView

    get(request)
        This get function :param request: Current request :return class: Response :rtype class: Response

        Return type Response

    post(request)
        This get function :param request: Current request :return class: Response :rtype class: Response

        Return type Response

class vtcuser.views.VideoLinkPostRudView(**kwargs)
    Bases: rest_framework.generics.RetrieveUpdateDestroyAPIView
    VideoLinkPostRudView class :param class: generics.RetrieveUpdateDestroyAPIView :type class: gener-
ics.RetrieveUpdateDestroyAPIView

    authentication_classes = [<class 'rest_framework_jwt.authentication.JSONWebTokenAuthen

    get_queryset()
        Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using
        self.queryset.

        This method should always be used rather than accessing self.queryset directly, as self.queryset gets eval-
        uated only once, and those results are cached for all subsequent requests.

        You may want to override this if you need to provide different querysets depending on the incoming
        request.

        (Eg. return a list of items that is specific to the user)

    lookup_field = 'pk'

    permission_classes = [<class 'vtcuser.permissions.IsOwnerOrReadOnly'>]

    serializer_class
        alias of vtcuser.serializers.VideoLinkSerializer

vtcuser.views.saved_videos(request)
    This view show all posted videos

vtcuser.views.video_crud(request, video_link_pk=None)
    This view add and edit video link

vtcuser.views.video_delete(request, video_link_pk)
    This view Delete video link
```

## PYTHON MODULE INDEX

### V

`vtcuser.views`, [21](#)



## A

authentication\_classes (vt-cuser.views.VideoLinkAPIView attribute), 21

authentication\_classes (vt-cuser.views.VideoLinkPostRudView attribute), 22

## C

create\_user\_profile() (vt-cuser.models.CustomUser method), 17

CustomUser (class in vtcuser.models), 17

CustomUser.DoesNotExist, 17

CustomUser.MultipleObjectsReturned, 17

CustomUserSerializer (class in vtcuser.serializers), 20

CustomUserSerializer.Meta (class in vtcuser.serializers), 20

CustomUserViewSet (class in vtcuser.api), 19

## D

date\_add (vtcuser.models.VideoLink attribute), 18

## F

fields (vtcuser.serializers.CustomUserSerializer.Meta attribute), 20

fields (vtcuser.serializers.VideoLinkSerializer.Meta attribute), 21

## G

get() (vtcuser.views.VideoLinkAPIView method), 22

get\_api\_url() (vtcuser.models.VideoLink method), 18

get\_next\_by\_date\_add() (vt-cuser.models.VideoLink method), 18

get\_previous\_by\_date\_add() (vt-cuser.models.VideoLink method), 18

get\_queryset() (vtcuser.views.VideoLinkAPIView method), 21

get\_queryset() (vt-cuser.views.VideoLinkPostRudView method), 22

get\_uri() (vtcuser.serializers.VideoLinkSerializer method), 21

## I

id (vtcuser.models.CustomUser attribute), 17

id (vtcuser.models.VideoLink attribute), 18

## L

link (vtcuser.models.VideoLink attribute), 18

lookup\_field (vtcuser.views.VideoLinkAPIView attribute), 21

lookup\_field (vtcuser.views.VideoLinkPostRudView attribute), 22

## M

model (vtcuser.serializers.CustomUserSerializer.Meta attribute), 20

model (vtcuser.serializers.VideoLinkSerializer.Meta attribute), 21

module

vtcuser.views, 21

## N

name (vtcuser.models.VideoLink attribute), 19

## O

objects (vtcuser.models.CustomUser attribute), 17

objects (vtcuser.models.VideoLink attribute), 19

owner() (vtcuser.models.VideoLink property), 19

## P

perform\_create() (vt-cuser.views.VideoLinkAPIView method), 22

permission\_classes (vt-cuser.views.VideoLinkPostRudView attribute), 22

post() (vtcuser.views.VideoLinkAPIView method), 22

post() (vtcuser.views.VideoLinkAPIView method), 22

poster (vtcuser.models.VideoLink attribute), 19

poster\_id (vtcuser.models.VideoLink attribute), 19

## Q

`queryset (vtcuser.api.CustomUserViewSet attribute)`, [19](#)  
`queryset (vtcuser.api.VideoLinkViewSet attribute)`, [20](#)

## R

`read_only_fields (vtcuser.serializers.VideoLinkSerializer.Meta attribute)`, [21](#)

## S

`saved_videos()` (in module `vtcuser.views`), [22](#)  
`serializer_class (vtcuser.api.CustomUserViewSet attribute)`, [19](#)  
`serializer_class (vtcuser.api.VideoLinkViewSet attribute)`, [20](#)  
`serializer_class (vtcuser.views.VideoLinkAPIView attribute)`, [22](#)  
`serializer_class (vtcuser.views.VideoLinkPostRudView attribute)`, [22](#)

## U

`user (vtcuser.models.CustomUser attribute)`, [17](#)  
`user_id (vtcuser.models.CustomUser attribute)`, [17](#)

## V

`validate_name()` (`vtcuser.serializers.VideoLinkSerializer` method), [21](#)  
`video_crud()` (in module `vtcuser.views`), [22](#)  
`video_delete()` (in module `vtcuser.views`), [22](#)  
`VideoLink` (class in `vtcuser.models`), [18](#)  
`VideoLink.DoesNotExist`, [18](#)  
`VideoLink.MultipleObjectsReturned`, [18](#)  
`videolink_set (vtcuser.models.CustomUser attribute)`, [18](#)  
`VideoLinkAPIView` (class in `vtcuser.views`), [21](#)  
`VideoLinkApiView` (class in `vtcuser.views`), [22](#)  
`VideoLinkPostRudView` (class in `vtcuser.views`), [22](#)  
`VideoLinkSerializer` (class in `vtcuser.serializers`), [21](#)  
`VideoLinkSerializer.Meta` (class in `vtcuser.serializers`), [21](#)  
`VideoLinkViewSet` (class in `vtcuser.api`), [20](#)  
`vtcuser.views` module, [21](#)