Python | Extending and customizing django-allauth

Last Updated : 22 May, 2024

Prerequisite: Django-allauth setup and Configuration

Let's deal with customizing django-allauth signup forms, and intervening in registration flow to add custom processes and validations.

Extending the Signup Form or adding custom fields in Django-allauth:

One of the most common queries about allauth is about adding additional fields or custom fields to the signup form. You can extend the SignupForm class from allauth.account.forms. All you need to do is create a custom class pass the SignupForm to the custom class and define the custom fields and save it. It's *vital* to return the user object as it will be passed to other modules for validation. You also need to include a variable in settings.py.

Let's see this using an example ***forms.py.***

Python3

```
from allauth.account.forms import SignupForm
from django import forms


class CustomSignupForm(SignupForm):
    first_name = forms.CharField(max_length=30, label='First Name')
    last_name = forms.CharField(max_length=30, label='Last Name')

    def save(self, request):
         user = super(CustomSignupForm, self).save(request)
        user.first_name = self.cleaned_data['first_name']
        user.last_name = self.cleaned_data['last_name']
```

```
        user.save()
        return user
```

In the above snippet, CustomSignupForm is extended the class which inherits all the features of SignupForm class and adds the necessary features. Here custom fields by the name first_nameand last_name are created and saved using the signup module in the same class.

The ACCOUNT_FORMS in settings.py for the above code is

Python3

```
ACCOUNT_FORMS = {
'signup': 'YourProject.forms.CustomSignupForm',
}
```

Any other custom forms created can be extended in ACCOUNT_FORMS. The list is illustrated in the documentation.
Similarly, *LoginForm UserForm AddEmailForm* and others can be extended. However, remember that when you extend these forms and link them in settings.py. Don't forget to pass the original form (For example *SignupForm*) as a parameter to your class and sometimes you might have to handle custom validations and return users or some other value. Please refer to the source code to follow the correct flow.

User Intervention and Custom validations:

Let's discuss adding custom validations and intervening in the user registration flow. *DefaultAccountAdapter* is very useful and indeed can be used to solve most of the customization problems that a user might encounter while using *django–allauth*.

Example #1: Restricted List of email's

After figuring out a way to store and fetch the restricted list, you can use the adapters and raise validation errors in the registration form when a restricted email tries to register. Extend a *DefaultAccountAdapter* and override the *clean_email* method. Create an adapter.py in your project directory and extend the default adapter class.

Python3

```
from allauth.account.adapter import DefaultAccountAdapter
from django.forms import ValidationError

class RestrictEmailAdapter(DefaultAccountAdapter):
    def clean_email(self, email):
        RestrictedList = ['Your restricted list goes here.']
        if email in RestrictedList
            raise ValidationError('You are restricted from registering.\
                                                    Please contact admin.')
        return email
```

Finally, point the account adapter in settings.py to your extended class.

```
ACCOUNT_ADAPTER='YourProject.adapter.RestrictEmailAdapter'
```

Example #2: Add a Maximum length to a username

As ACCOUNT_USERNAME_MAX_LENGTH doesn't exist in the allauth library, *DefaultAccountAdaptercan* is used to achieve this feature without much pain. Extend the *DefaultAccountAdapterclass* and overriding the *clean_username* method. You need to also reference the *clean_username* once again after our custom validation to complete other inbuilt validations.

The last sentence in the above paragraph is the key to work with *DefaultAccountAdapter*. You should never forget to reference the original module name for the module to complete other validations.

Python3

```
from allauth.account.adapter import DefaultAccountAdapter
from django.forms import ValidationError

class UsernameMaxAdapter(DefaultAccountAdapter):
    def clean_username(self, username):
        if len(username) &gt; 'Your Max Size':
```

```
            raise ValidationError('Please enter a username value\
                                   less than the current one')

        # For other default validations.
        return DefaultAccountAdapter.clean_username(self, username)
```

Finally, point to the subclass in your settings.py

```
ACCOUNT_ADAPTER = 'YourProject.adapter.UsernameMaxAdapter'
```

You can refer to the *adapters.py* file in the source code and extend other modules and add a process or change their flow. The modules such as *populate_username*, *clean_password* (which can be customized to restrict commonly used passwords) can have the custom process and flow without rewriting them.
DefaultAccountAdapter can be a potent tool if used in the right circumstances to intervene in allauth's default process. allauth comes with a vast variety of inbuilt settings, and they are here.

You can also download the entire source code from the link in the references below.

References: stack overflow thread on Example 1

 gjbht 5

Previous Article

Python | Django Admin Interface

Next Article

Django - Dealing with warnings

Similar Reads

Creating custom user model API extending AbstractUser in Django

Every new Django project should use a custom user model. The official Django documentation says it is “highly recommended” but I’ll go a step further and say...

4 min read

Customizing Object Level Permissions - Django REST Framework

In this article, we will discuss how to customize Object Level Permissions in Django REST Framework. To customize permission classes in Django REST...

5 min read

Customizing Filters in Django REST Framework

Prerequisite: Adding Filtering in APIs – Django REST Framework [link needed article on published yet] Django filters facilitate filtering the queryset to retrieve...

4 min read

Customizing Phone Number Authentication in Django

As we know, Introducing phone number-based authentication in the Django Admin Panel is a strategic move to modernize and enhance your web...

3 min read

Extending a list in Python (5 different ways)

In this article, we are going to learn different methods of extending a list in Python. The list is the widely used Python data structure and it can be extended...

3 min read

Customizing Styles in Matplotlib

Here, we'll delve into the fundamentals of Matplotlib, exploring its various classes and functionalities to help you unleash the full potential of your data...

12 min read

Customizing Swagger UI

FastAPI is a state-of-the-art, high-performance web framework that uses normal Python type hints to develop APIs with Python 3.7+. FastAPI's ability to...

6 min read

Adding Tags Using Django-Taggit in Django Project

Django-Taggit is a Django application which is used to add tags to blogs, articles etc. It makes very easy for us to make adding the tags functionality to our djang...

2 min read

How to customize Django forms using Django Widget Tweaks ?

Django forms are a great feature to create usable forms with just few lines of code. But Django doesn't easily let us edit the form for good designs. Here, we...

3 min read

Styling Django Forms with django-crispy-forms

Django by default doesn't provide any Django form styling method due to which it takes a lot of effort and precious time to beautifully style a form. django-crispy...

1 min read

Article Tags : Articles Python Technical Scripter

Practice Tags : python

Tower K, Gulshan Vivante Apartment,
Sector 137, Noida, Gautam Buddh
Nagar, Uttar Pradesh, 201305