TgPostman Documentation

Release v1.0.0

Дымников Михаил (dym-dino)

Apr 06, 2025

CONTENTS

CHAPTER

ONE

DJANGO TGPOSTMAN

1.1 scheduled_posts

1.1.1 tests

test_scheduled_posts

Post tests

This file contains unit tests for post creation with delay using Django REST Framework and Celery.

class scheduled_posts.tests.test_scheduled_posts.PostTests(methodName='runTest')

Bases: APITestCase

Test case for creating posts with delayed execution via Celery.

setUp() → None

Set up a test user and a Telegram chat for use in test methods.

test_create_post_with_delay(mock_async: patch) → None

Test creating a post with a delay. Verifies that Celery async task is triggered. :param mock_async: Mock for Celery apply_async method

1.1.2 admin

Admin config

This file contains admin panel configuration for the ScheduledPost model.

class scheduled_posts.admin.PostAdmin(model, admin_site)

Bases: ModelAdmin

Admin configuration for displaying and managing ScheduledPost entries.

list_display = ('user', 'schedule_time', 'status', 'created_at')

search_fields = ('user__username', 'content')

property media

1.1.3 apps

App config

This file contains application configuration for the scheduled_posts app.

class scheduled_posts.apps.ScheduledPostsConfig(app_name, app_module)

Bases: AppConfig

Configuration class for the scheduled_posts Django app.

default_auto_field = 'django.db.models.BigAutoField'

name = 'scheduled_posts'

1.1.4 forms

Create post form

This file defines the form for creating scheduled posts in the Django admin panel.

class scheduled_posts.forms.CreatePostForm(*args, **kwargs)

Bases: Form

Form for creating a scheduled post, including content, HTML option, file upload, target chats, and delay in seconds.

base_fields = {'content': <django.forms.fields.CharField object>, 'delay_seconds': <django.forms.fields.IntegerField object>, 'file': <django.forms.fields.FileField object>, 'html': <django.forms.fields.BooleanField object>, 'schedule_option': <django.forms.fields.ChoiceField object>, 'schedule_time': <django.forms.fields.DateTimeField object>, 'targets': <django.forms.models.ModelMultipleChoiceField object>}

declared_fields = {'content': <django.forms.fields.CharField object>, 'delay_seconds': <django.forms.fields.IntegerField object>, 'file': <django.forms.fields.FileField object>, 'html': <django.forms.fields.BooleanField object>, 'schedule_option': <django.forms.fields.ChoiceField object>, 'schedule_time': <django.forms.fields.DateTimeField object>, 'targets': <django.forms.models.ModelMultipleChoiceField object>}

property media

Return all media required to render the widgets on this form.

1.1.5 models

Scheduled post model

This file defines the ScheduledPost model, which represents a post scheduled for future delivery.

class scheduled_posts.models.ScheduledPost(*args, **kwargs)

Bases: Model

Model representing a post scheduled to be sent to Telegram chats.

STATUS_CHOICES = [('pending', 'Pending'), ('sent', 'Sent'), ('failed', 'Failed')]

user

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.

content

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

html

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

file

The descriptor for the file attribute on the model instance. Return a FieldFile when accessed so you can write code like:

```
>>> from myapp.models import MyModel
>>> instance = MyModel.objects.get(pk=1)
>>> instance.file.size
```

Assign a file object on assignment so you can do:

```
>>> with open('/path/to/hello.world') as f:
...     instance.file = File(f)
```

targets

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

Pizza.toppings and Topping.pizzas are ManyToManyDescriptor instances.

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

schedule_time

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

created_at

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

status

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

error_message

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

celery_task_id

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

save(*args, **kwargs) → None

Override the default save method to set a default schedule time if none is provided.

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

get_next_by_created_at(*, field=<django.db.models.fields.DateTimeField: created_at>, is_next=True, **kwargs)

get_next_by_schedule_time(*, field=<django.db.models.fields.DateTimeField: schedule_time>, is_next=True, **kwargs)

get_previous_by_created_at(*, field=<django.db.models.fields.DateTimeField: created_at>, is_next=False, **kwargs)

get_previous_by_schedule_time(*, field=<django.db.models.fields.DateTimeField: schedule_time>, is_next=False, **kwargs)

get_status_display(*, field=<django.db.models.fields.CharField: status>)

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_id

1.1.6 post_sender

Post sender

This file contains logic for sending scheduled posts to Telegram chats using the TeleBot library.

scheduled_posts.post_sender.send_post(post: ScheduledPost) → None

Send the given ScheduledPost to all its target Telegram chats and delete the file after sending.

Parameters
post – ScheduledPost instance containing content and target chats

Raises
Exception – If sending fails for any of the target chats

1.1.7 serializers

Scheduled post serializer

This file defines a serializer for the ScheduledPost model, supporting optional delays and Celery task scheduling.

class scheduled_posts.serializers.ScheduledPostSerializer(*args, **kwargs)

Bases: ModelSerializer

Serializer for creating and displaying ScheduledPost instances. Supports delay in seconds and triggers Celery task scheduling.

class Meta

Bases: object

model

alias of ScheduledPost

fields = ('id', 'content', 'html', 'file', 'targets', 'schedule_time', 'delay_seconds', 'status', 'created_at', 'error_message')

read_only_fields = ('status', 'created_at', 'error_message', 'schedule_time')

create(validated_data: dict) → ScheduledPost

Create a ScheduledPost instance, apply delay if provided, and schedule a Celery task.

Parameters
validated_data – Validated input data

Returns
ScheduledPost instance

1.1.8 tasks

Scheduled task

This file defines a Celery task to send a scheduled post and update its status accordingly.

1.1.9 urls

URL patterns

This file contains the URL patterns for the scheduled posts functionality.

1.1.10 views

Views for scheduled posts

This file contains views for listing, creating, and displaying scheduled posts.

class scheduled_posts.views.ScheduledPostListCreateView(**kwargs)

Bases: ListCreateAPIView

View for listing and creating scheduled posts. Requires the user to be authenticated.

serializer_class

alias of ScheduledPostSerializer

permission_classes = [<class 'rest_framework.permissions.IsAuthenticated'>]

get_queryset()

Return the queryset of posts for the current user.

scheduled_posts.views.create_post_view(request)

View for creating a new scheduled post.

scheduled_posts.views.my_posts_view(request)

View for displaying all posts created by the current user.

scheduled_posts.views.send_post_now(request, post_id)

View to manually send a scheduled post that is still pending.

scheduled_posts.views.cancel_post(request, post_id)

View to cancel a scheduled post.

1.2 telegram_accounts

1.2.1 tests

test_telegram_accounts

Telegram chat tests

This file contains unit tests for the Telegram chat integration, including the add chat functionality.

class telegram_accounts.tests.test_telegram_accounts.TelegramChatTests(methodName='runTest')

Bases: APITestCase

Test case for testing the functionality related to Telegram chats.

setUp() → None

Set up a test user and client credentials for the test cases.

test_add_chat(mock_get_chat_info) → None

Test the addition of a new Telegram chat using mocked chat info. :param mock_get_chat_info: Mock for the get_chat_info function.

1.2.2 admin

Telegram chat admin config

This file contains the admin configuration for managing Telegram chats in the Django admin panel.

class telegram_accounts.admin.TelegramChatAdmin(model, admin_site)

Bases: ModelAdmin

Admin configuration for the TelegramChat model, defining how Telegram chats are displayed and searched.

list_display = ('title', 'chat_id', 'user', 'can_post', 'added_at')

search_fields = ('title', 'chat_id')

property media

1.2.3 apps

class telegram_accounts.apps.TelegramAccountsConfig(app_name, app_module)

Bases: AppConfig

default_auto_field = 'django.db.models.BigAutoField'

name = 'telegram_accounts'

1.2.4 forms

class telegram_accounts.forms.AddChatForm(data=None, files=None, auto_id='id_%s', prefix=None, initial=None, error_class=<class 'django.forms.utils.ErrorList'>, label_suffix=None, empty_permitted=False, field_order=None, use_required_attribute=None, renderer=None)

Bases: Form

clean_chat_id()

base_fields = {'chat_id': <django.forms.fields.CharField object>}

declared_fields = {'chat_id': <django.forms.fields.CharField object>}

property media

Return all media required to render the widgets on this form.

class telegram_accounts.forms.TelegramChatForm(*args, **kwargs)

Bases: ModelForm

class Meta

Bases: object

model

alias of TelegramChat

fields = ['chat_id']

widgets = {'chat_id': <django.forms.widgets.TextInput object>}

clean_chat_id()

save(commit=True)

Save this form's self.instance object if commit=True. Otherwise, add a save_m2m() method to the form which can be called after the instance is saved manually at a later time. Return the model instance.

base_fields = {'chat_id': <django.forms.fields.IntegerField object>}

declared_fields = {}

property media

Return all media required to render the widgets on this form.

1.2.5 models

class telegram_accounts.models.TelegramChat(id, user, chat_id, chat_type, title, can_post, added_at)

Bases: Model

user

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.

chat_id

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

chat_type

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

title

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

can_post

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

added_at

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

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

get_next_by_added_at(*, field=<django.db.models.fields.DateTimeField: added_at>, is_next=True, **kwargs)

get_previous_by_added_at(*, field=<django.db.models.fields.DateTimeField: added_at>, 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.

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

scheduledpost_set

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

Pizza.toppings and Topping.pizzas are ManyToManyDescriptor instances.

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

user_id

1.2.6 serializers

Telegram chat serializer

This file defines the serializer for the TelegramChat model, including field definitions and object creation logic.

class telegram_accounts.serializers.TelegramChatSerializer(*args, **kwargs)

Bases: ModelSerializer

Serializer for creating and displaying Telegram chat objects.

class Meta

Bases: object

model

alias of TelegramChat

fields = '__all__'

read_only_fields = ('user', 'title', 'can_post', 'chat_type')

create(validated_data: dict) → TelegramChat

Create a new TelegramChat instance.

Parameters
validated_data – Validated data for creating a Telegram chat

Returns
The created TelegramChat instance

1.2.7 telegram_api

Telegram chat info

This file contains logic for interacting with the Telegram bot API to retrieve chat information.

telegram_accounts.telegram_api.get_chat_info(chat_id: int) → dict

Retrieve information about a Telegram chat, such as title, posting permissions, and chat type.

Parameters
chat_id – The ID of the chat to retrieve information about

Returns
A dictionary containing the chat's title, post permission, and type

Raises
ValueError – If the chat info cannot be fetched

1.2.8 urls

Telegram chat URLs

This file defines the URL patterns for managing Telegram chats via both web and API interfaces.

1.2.9 views

Telegram chat views

This file contains views for managing Telegram chats, including listing, adding, deleting, and API-based interactions.

class telegram_accounts.views.TelegramChatListCreateView(**kwargs)

Bases: View

View for listing and creating Telegram chats.

get(request)
Handle GET request to display the form and user's Telegram chats.

post(request)
Handle POST request to create a new Telegram chat.

dispatch(request, *args, **kwargs)

class telegram_accounts.views.TelegramChatDeleteView(**kwargs)
Bases: View

View for deleting a Telegram chat.

post(request, pk)
Handle POST request to delete a chat by its primary key (pk).

dispatch(request, *args, **kwargs)

telegram_accounts.views.add_chat_view(request)
View for adding a new Telegram chat.

class telegram_accounts.views.TelegramChatViewSet(**kwargs)
Bases: ModelViewSet

ViewSet for managing Telegram chats via API.

queryset

serializer_class
alias of TelegramChatSerializer

permission_classes = [<class 'rest_framework.permissions.IsAuthenticated'>]

get_queryset()
Return the queryset of Telegram chats for the current user.

perform_create(serializer)
Override the default create method to add chat info during creation.

list_my_chats(request)
List chats that belong to the authenticated user.

basename = None

description = None

detail = None

name = None

suffix = None

1.3 tgpostman

1.3.1 celery

Celery configuration

This file sets up the Celery application for the project.

1.3.2 settings

Django settings for tgpostman project.

Generated by ‘django-admin startproject’ using Django 5.1.8.

For more information on this file, see https://docs.djangoproject.com/en/5.1/topics/settings/

For the full list of settings and their values, see https://docs.djangoproject.com/en/5.1/ref/settings/

1.3.3 urls

URL configuration for tgpostman project.

This file contains the URL patterns for the web interface, API modules, and documentation.

1.4 users

1.4.1 tests

test_users

User API tests

This file contains tests for the user registration and API key generation, as well as verifying the API key functionality.

class users.tests.test_users.UserTests(methodName='runTest')

Bases: APITestCase

Test case for user registration and API key generation.

test_register_user_and_get_api_key() → None

Test the user registration and API key generation process. Verify that the user is created, and an API key is generated.

1.4.2 admin

User admin configuration

This file contains the admin configuration for managing User objects in the Django admin panel.

class users.admin.UserAdmin(model, admin_site)

Bases: ModelAdmin

Admin configuration for the User model.

list_display = ('username', 'api_key', 'is_staff', 'is_active')

property media

1.4.3 apps

Users app configuration

This file contains the configuration for the ‘users’ Django app.

class users.apps.UsersConfig(app_name, app_module)

Bases: AppConfig

Configuration for the ‘users’ app.

default_auto_field = 'django.db.models.BigAutoField'

name = 'users'

1.4.4 authentication

API key authentication

This file contains the custom authentication class for API key authentication in the Django REST Framework.

class users.authentication.ApiKeyAuthentication

Bases: BaseAuthentication

Custom authentication class that authenticates users based on an API key.

authenticate(request)

Authenticate the user based on the provided API key.

Parameters

request – The HTTP request object containing the API key in the headers.

Returns

A tuple of user and None if the API key is valid, or raises AuthenticationFailed.

1.4.5 forms

User registration form

This file contains the form for user registration using a custom User model.

class users.forms.RegisterForm(*args, **kwargs)

Bases: UserCreationForm

Custom form for user registration. Extends the default UserCreationForm to use the custom User model.

class Meta

Bases: object

model

alias of User

fields = ('username',)

base_fields = {'password1': <django.forms.fields.CharField object>, 'password2': <django.forms.fields.CharField object>, 'username': <django.forms.fields.CharField object>}

declared_fields = {'password1': <django.forms.fields.CharField object>, 'password2': <django.forms.fields.CharField object>}

property media

Return all media required to render the widgets on this form.

1.4.6 models

Custom user model

This file defines the custom User model, extending the default AbstractUser model with an API key.

class users.models.User(*args, **kwargs)

Bases: AbstractUser

Custom user model that extends the default Django AbstractUser with an API key.

api_key

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

save(*args, **kwargs) → None

Override the save method to generate a new API key if it does not exist.

Parameters

- args – Positional arguments passed to the parent save method
- kwargs – Keyword arguments passed to the parent save method

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

date_joined

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

email

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

first_name

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

get_next_by_date_joined(*, field=<django.db.models.fields.DateTimeField: date_joined>, is_next=True, **kwargs)

get_previous_by_date_joined(*, field=<django.db.models.fields.DateTimeField: date_joined>, is_next=False, **kwargs)

groups

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

Pizza.toppings and Topping.pizzas are ManyToManyDescriptor instances.

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

id

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

is_active

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

is_staff

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

is_superuser

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

last_login

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

last_name

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

logentry_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.

password

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

scheduledpost_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.

telegramchat_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.

user_permissions

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

Pizza.toppings and Topping.pizzas are ManyToManyDescriptor instances.

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

username

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

1.4.7 serializers

User serializers

This file contains serializers for user registration and API key generation.

class users.serializers.UserRegisterSerializer(*args, **kwargs)

Bases: ModelSerializer

Serializer for user registration, including password validation.

class Meta

Bases: object

model

alias of User

fields = ('username', 'password')

create(validated_data: dict) → User

Create a new user with the validated data.

Parameters

validated_data – Data that has passed validation

Returns

The newly created user

class users.serializers.ApiKeySerializer(*args, **kwargs)

Bases: ModelSerializer

Serializer for returning the user's username and API key.

class Meta

Bases: object

model

alias of User

fields = ('username', 'api_key')

1.4.8 urls

User API URLs

This file defines the URL patterns for user-related API endpoints, including registration, login, and API key retrieval.

1.4.9 views

User views

This file contains views for user registration, login, and API key management.

class users.views.RegisterView(**kwargs)

Bases: CreateAPIView

View for user registration. Allows any user to create an account.

queryset

serializer_class

alias of UserRegisterSerializer

permission_classes = [<class 'rest_framework.permissions.AllowAny'>]

class users.views.LoginAPIView(**kwargs)

Bases: APIView

View for logging in a user using username and password. Returns an API key if credentials are valid.

permission_classes = [<class 'rest_framework.permissions.AllowAny'>]

post(request)

Handle user login and return the API key if credentials are valid.

Parameters

request – The request object containing 'username' and 'password'

Returns

API key for authenticated user or error message

class users.views.ApiKeyView(**kwargs)

Bases: RetrieveAPIView

View to retrieve the API key of the authenticated user.

serializer_class

alias of ApiKeySerializer

get_object()

Return the current user object.

Returns

The user associated with the current request

users.views.register_view(request)

View for user registration via a web form.

users.views.dashboard_view(request)

View for rendering the user dashboard page. Accessible only by authenticated users.