Django Task API Documentation

Release 1.0.0

Nikolas Stevenson-Molnar

Aug 18, 2018

Contents

Django Task API lets you quickly write background tasks for your Django project and easily call then using the provided REST API, or the included JavaScript library.

CHAPTER 1

What does it look like?

Tasks are defined as classes with typed input and output parameters, and a *run* function with the task implementation, to be called by a worker processes.

```
from task_api.tasks import Task
from task_api.params import ListParameter, NumberParameter

class SumNumbers(Task):
    name = 'sum'

    inputs = {
        'numbers': ListParameter(NumberParameter())
    }

    outputs = {
        'sum': NumberParameter()
    }

    def run(self, numbers):
        return sum(numbers)
```

Tasks are easily called and monitored in front-end code using the included JavaScript API. The API supports both promises (will Polyfill for older browsers) and traditional callbacks.

```
<script src="{% static 'django-task-api.js' %}"></script>

<script type="text/javascript">
    function sumTask(numbers) {
        TaskAPI
            .run('sum', {'numbers': numbers})
            .then(function(json) {
                console.log('Sum: ' + json.outputs.sum)
            })
    }
</script>
```

CHAPTER 2

Next Steps

- *Getting Started*
- GitHub

2.1 Getting Started

2.1.1 Install Django Task API

Install the Python library with pip:

```
$ pip install django-task-api
```

Django Task API is compatible and tested with Python versions 2.7, 3.5, 3.6, and 3.7, and with Django versions 1.11, 2.0, and 2.1.

2.1.2 Set up Celery

By default, Django Task API uses Celery to manage background tasks. If you're not already using Celery, follow the First steps with Django document to configure Celery for your project.

2.1.3 Create a task

Create a module in your Django app called *background.py* and add a task class to it:

Listing 1: myapp/background.py

```
from time import sleep

from task_api.params import IntParameter
```

(continues on next page)

(continued from previous page)

```
from task_api.tasks import Task

class WaitTask(Task):
    name = 'wait'

    inputs = {
        'seconds': IntParameter(required=False)
    }

    def run(seconds=10):
        self.set_target(10)
        self.set_progress(0)

        for _ in range(seconds):
            sleep(1)
            self.inc_progress(1)
```

The `WaitTask` task accepts an integer value and counts toward that number, one second at a time. Since it updates its progress, we'll be able to monitor it from the front-end.

2.1.4 Configure settings & URLS

Edit `settings.py`, add the `task_api` app, and add tasks to `TASK_API_BACKGROUND_TASKS`.

Listing 2: settings.py

```
INSTALLED_APPS += [
    'task_api'
]

TASK_API_BACKGROUND_TASKS = ['myapp.background.WaitTask']
```

With the app added to settings, run Django's migrate command:

```
$ python manage.py migrate
```

We'll also need a URL route to the task API:

Listing 3: urls.py

```
from django.conf.urls import url
from django.urls import include

urlpatterns = [
    url('^', include('task_api.urls'))
]
```

2.1.5 Add front-end Java Script

Django Task API includes a JS API for starting and monitoring background tasks. If you're using Django to manage your static files, then you can include the library using the `{% static %}` template tag. You can also install the JavaScript library from npm. For purposes of this walk through, let's create a template with some simple HTML and JavaScript to start and monitor a task:

Listing 4: myapp/templates/task.html

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Task API Example</title>

    <script src="{% static "django-task-api.js" %}"></script>
    <script type="text/javascript">
        function startTask() {
            TaskAPI.run('wait', {'seconds': 10}, function(json) {
                if (json.target === null || json.progress === null) {
                    return
                }
                document.getElementById('status').innerHTML = 'Progress: ' + json.
↪progress + ' / ' + json.target
            }).then(function() {
                document.getElementById('button').disabled = false
            })

            document.getElementById('button').disabled = true
        }
    </script>
    <style type="text/css">
        .content {
            position: fixed;
            top: 25%;
            left: 50%;
            transform: translate(-50%, -50%);
        }

        .content button {
            margin-top: 5px;
            width: 100px;
        }

        .center {
            text-align: center;
        }
    </style>
</head>
<body>
    <div class="content">
        <div id="status">Click "Run" to start the task.</div>
        <div class="center"><button onclick="startTask()" id="button">Run</button></
↪div>
    </div>
</body>
</html>
```

This gives the user a “Run” button, which when clicked, starts the task defined earlier. As the task counts towards its target, the UI updates to show the current progress.

To finish everything out, we need to add a URL route for this template:

Listing 5: myapp/urls.py

```
from django.conf.urls import url
from django.urls import include
from django.views.generic import TemplateView

url_patterns = [
    # ... other URL patterns
    url('^$', TemplateView.as_view(template_name='example.html'))
]
```

If you haven't already, add your app to your project `urls.py` file:

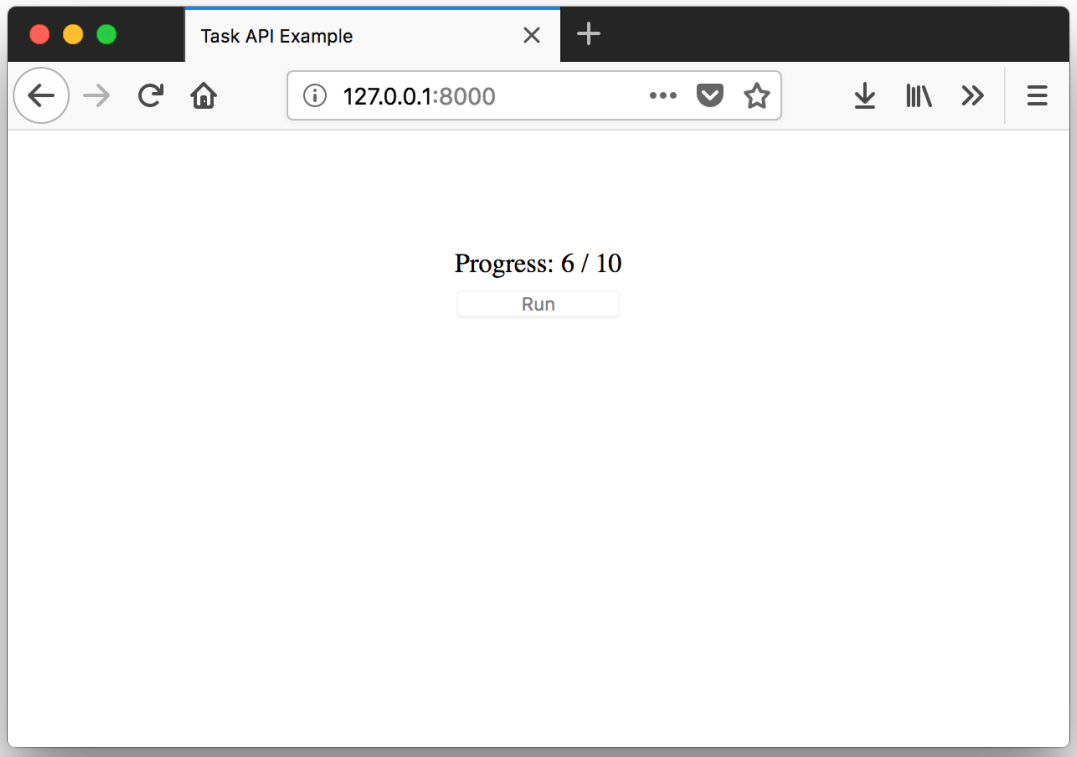
Listing 6: urls.py

```
from django.conf.urls import url
from django.urls import include

urlpatterns = [
    url('^', include('task_api.urls')),
    url('^myapp', include('myapp.urls'))
]
```

2.1.6 Try it out

Your new task should be ready to go. Make sure that the Django debug server and Celery worker process are both running, then open the page in your browser. Click "Run" and watch your task progress.



2.1.7 Problems?

If you run into problems, take a look at the fully-functional example project here.

2.2 References

2.2.1 Settings

TASK_API_AUTHENTICATION_CLASSES

Authentication classes to be used by the Task API view. These should be Django Rest Framework (DRF) authentication classes. See DRF's Authentication guide for more info.

TASK_API_BACKEND

The task backend class. Defaults to `task_api.backends.celery.CeleryBackend`, which is the only built-in backend.

TASK_API_BACKGROUND_CLASSES

A list of `Task` classes to be provided by the API. For example:

```
TASK_API_BACKGROUND_CLASSES = [
    'myapp.background.MyTask'
]
```

You can also use class objects directly. For example:

```
from myapp.background import MyTask

TASK_API_BACKGROUND_CLASSES = [MyTask]
```

Note that the above example uses the class definition, *not* a class instance.

TASK_API_PERMISSIONS_CLASSES

Permissions classes to be used by the Task API view. These should be Django Rest Framework (DRF) permissions classes. See DRF's Permissions guide for more info.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search