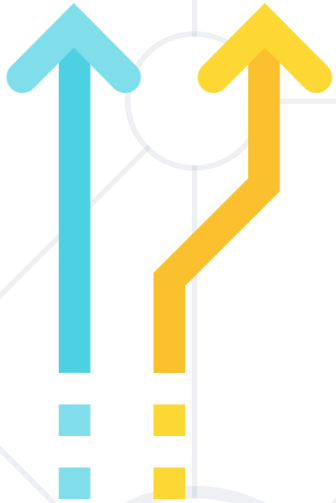
Asynchronous Tasks with Django

Celery and Redis



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.org

Table of Contents

1. Asynchronous Tasks
2. Celery
3. Redis
4. Simple App with Celery and Redis

Have a Question?

Software University

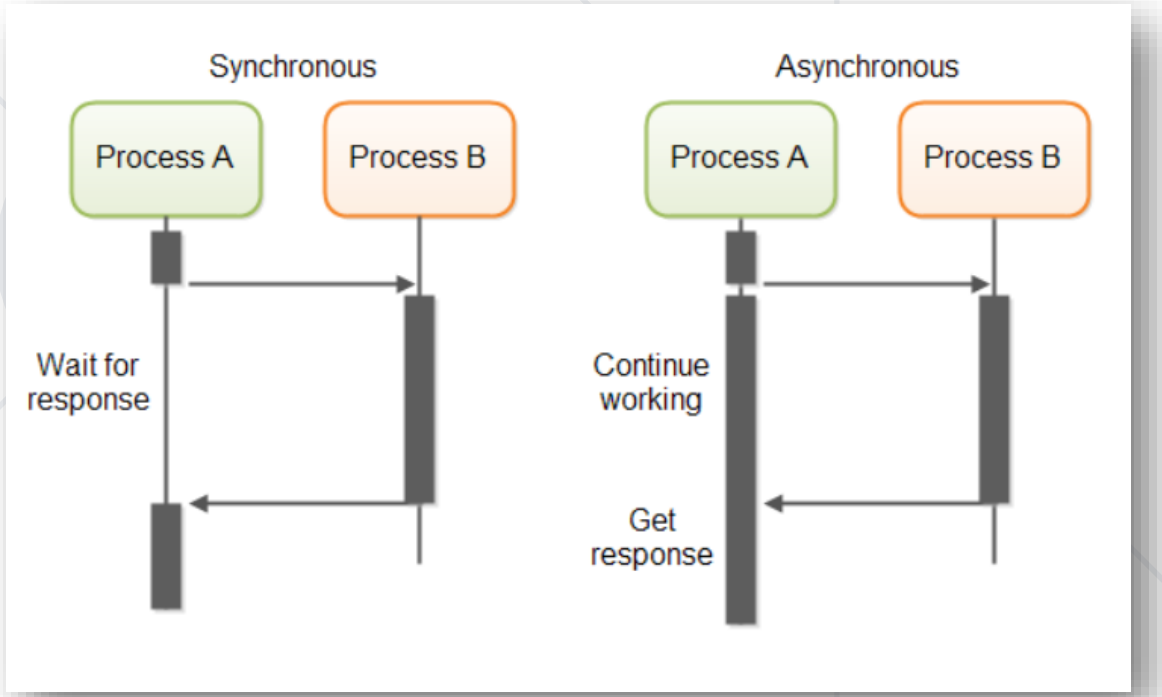
sli.do

#python-web

Asynchronous Tasks

What is Asynchronous task?

- Computation that runs on a **background thread** and whose result is published on the UI thread
- When you execute something **synchronously**, you **wait** for it to finish **before moving on** to another task
- When you execute something **asynchronously**, you can **move on** to another task **before it finishes**



Advantages

- With **asynchronous** execution, you begin a **routine**, and let it run in the **background** while you start your next, then at some point, say "wait for this to finish"

A -> B -> C -> D

- You can **execute B**, **C**, and or **D** while **A** is **still running**, so you can take better advantage of your resources and have **fewer** "hangs" or "waits"

What is Celery?

What is Celery?

- Celery is an **asynchronous** task queue/job queue based on **distributed message passing**
- It is focused on **real-time operation** but supports scheduling as well

Advantages

- Think of all the times you have had to run a certain task in the future
- Perhaps you needed to **access an API every hour**
- Or maybe you needed to **send a batch of emails** at the end of the day
- Large or small, Celery makes scheduling such periodic tasks easy
- You never want end users to have to **wait** unnecessarily for **pages to load** or **actions to complete**

What is Redis?

What is Redis?

- Redis is an **in-memory** data structure project implementing a distributed, **in-memory key-value database** with **optional durability**
- The most common Redis use cases are **session cache**, **full-page cache**, **queues**, **leaderboards** and counting, publish-subscribe, and much more

Live Demo

Creating Simple Image Thumbnails App

Installing Needed Packages

- Installing Redis
  - Download the Redis zip file and unzip it in some directory
  - Find the file named **`redis-server.exe`** and double click to launch the server in a command window
  - Similarly, find another file named **`redis-cli.exe`** and double-click it to open the program in a separate command window
- Create a **project folder** and install the following **dependencies**

```
pip install Django Celery redis Pillow django-widget-tweaks
pip freeze > requirements.txt
```

Setting Up the Django Project

Software University

- Create a new Django **project** and in it create a **single app**

```
django-admin startproject image_parroter
cd image_parroter
python manage.py startapp thumbnailer
```

- To integrate Celery, add a new **module** in the app **`celery.py`**

```
# image_parroter/image_parroter/celery.py
import os
from celery import Celery
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'image_parroter.settings')
celery_app = Celery('image_parroter')
celery_app.config_from_object('django.conf:settings', namespace='CELERY')
celery_app.autodiscover_tasks()
```

Defining Celery Settings

- Go to the project's settings and define a section for **celery settings**

```
# … skipping to the bottom

# celery
CELERY_BROKER_URL = 'redis://localhost:6379'
CELERY_RESULT_BACKEND = 'redis://localhost:6379'
CELERY_ACCEPT_CONTENT = ['application/json']
CELERY_RESULT_SERIALIZER = 'json'
CELERY_TASK_SERIALIZER = 'json'
```

Injecting Celery in Django

- To ensure that the celery application gets **injected** into the Django application, **import** the it within the Django project's main **`__init__.py`** script

```
# image_parroter/image_parroter/__init__.py

from .celery import celery_app

__all__ = ('celery_app',)
```

Adding Tasks Module

- Create `tasks.py` within the "thumbnailer" application
- Inside the `tasks.py` module import the `shared_tasks` function decorator and use it to define a celery task function called `adding_task`

```
# image_parroter/thumbnailer/tasks.py
from celery import shared_task

@shared_task
def adding_task(x, y):
    return x + y
```

Django INSTALLED_APPS

- Lastly, we need to add the **thumbnailer app** to the list of **INSTALLED_APPS**
- While we're in there we should also add the **"widget_tweaks"** application

```
# image_parroter/image_parroter/settings.py
INSTALLED_APPS = [
    …
    'thumbnailer.apps.ThumbnailerConfig',
    'widget_tweaks',
]
```

Creating Image Thumbnails within a Celery Task

Software University

- Back in the `tasks.py` module import the Image class from the PIL package
- Add a new task called **make_thumbnails**
- It should accept an image file path and a list of **2-tuple** width and height dimensions to create thumbnails of

Creating Image Thumbnails within a Celery Task

Software University

```
# image_parroter/thumbnailer/tasks.py
import os
from zipfile import ZipFile
from celery import shared_task
from PIL import Image
from django.conf import settings

@shared_task
def make_thumbnails(file_path, thumbnails=[]):
    os.chdir(settings.IMAGES_DIR)
    path, file = os.path.split(file_path)
    file_name, ext = os.path.splitext(file)
    zip_file = f"{file_name}.zip"
    results = {'archive_path': f"{settings.MEDIA_URL}images/{zip_file}"}
    # Continues on the next slide
```

Creating Image Thumbnails within a Celery Task

Software University

```
# Continues from the previous slide
    try:
        img = Image.open(file_path)
        zipper = ZipFile(zip_file, 'w')
        zipper.write(file)
        os.remove(file_path)
        for w, h in thumbnails:
            img_copy = img.copy()
            img_copy.thumbnail((w, h))
            thumbnail_file = f'{file_name}_{w}x{h}.{ext}'
            img_copy.save(thumbnail_file)
            zipper.write(thumbnail_file)
            os.remove(thumbnail_file)
        img.close()
        zipper.close()
    # TODO: catch IOError and return the result
```

Setup Media Root and URL

- Give a **MEDIA_ROOT** location where image files and zip archives can reside, specify the **MEDIA_URL** where the content can be served from

```
# image_parroter/settings.py
STATIC_URL = '/static/'
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.abspath(os.path.join(BASE_DIR, 'media'))
IMAGES_DIR = os.path.join(MEDIA_ROOT, 'images')
if not os.path.exists(MEDIA_ROOT) or not
os.path.exists(IMAGES_DIR):
    os.makedirs(IMAGES_DIR)
```

Creating the View

Software University

```
# thumbnailer/views.py

import os

from celery import current_app

from django import forms
from django.conf import settings
from django.http import JsonResponse
from django.shortcuts import render
from django.views import View

from .tasks import make_thumbnails

class FileUploadForm(forms.Form):
    image_file = forms.ImageField(required=True)
# Continues on the next slide
```

Creating the View

Software University

```
class HomeView(View):
    def get(self, request):
        form = FileUploadForm()
        return render(request, 'thumbnailer/home.html', { 'form': form })
    def post(self, request):
        form = FileUploadForm(request.POST, request.FILES)
        context = {}
        if form.is_valid():
            file_path = os.path.join(settings.IMAGES_DIR, request.FILES['image_file'].name)
            with open(file_path, 'wb+') as fp:
                for chunk in request.FILES['image_file']:
                    fp.write(chunk)
            task = make_thumbnails.delay(file_path, thumbnails=[(128, 128)])
            context['task_id'] = task.id
            context['task_status'] = task.status
            return render(request, 'thumbnailer/home.html', context)
        context['form'] = form
        return render(request, 'thumbnailer/home.html', context)
```

Creating the View

```
class TaskView(View):
    def get(self, request, task_id):
        task = current_app.AsyncResult(task_id)
        response_data = {
            'task_status': task.status,
            'task_id': task.id
        }

        if task.status == 'SUCCESS':
            response_data['results'] = task.get()

        return JsonResponse(response_data)
```

Adding the URL's

- Add a **urls.py** module inside the **thumbnailer** application and define the following URLs

```
# thumbnailer/urls.py
from django.urls import path
from . import views

urlpatterns = [
  path('', views.HomeView.as_view(), name='home'),
  path('task/<str:task_id>/', views.TaskView.as_view(), name='task'),
]
```

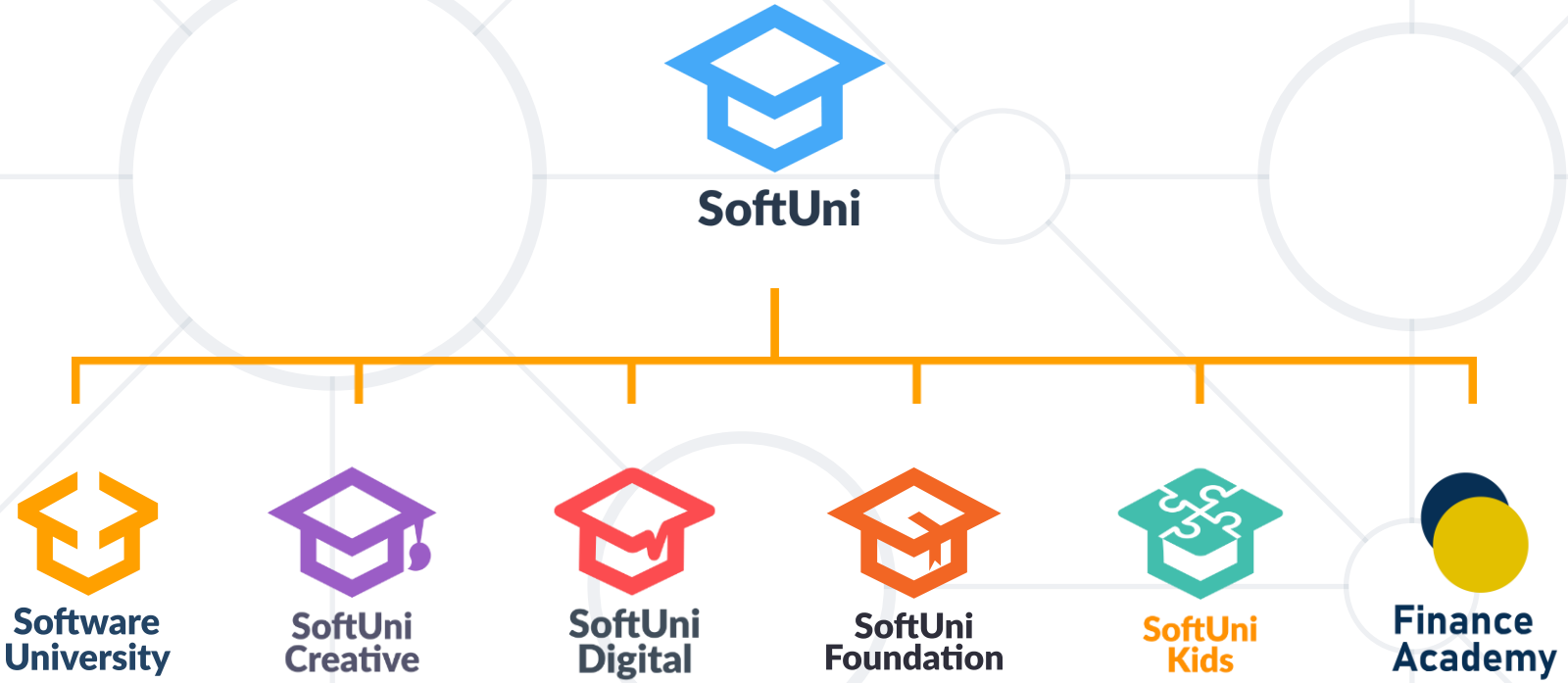
Creating the Template

- Next, begin building out a simple **template view** for a user to submit an image file
- The user should be able to:
  - Check the **status** of the submitted make_thumbnails tasks
  - Initiate a **download** of the resulting thumbnails
- To start off, create a **directory** to house this single template within the thumbnailer directory
- Use the HTML file from here
- Test your application on *http://localhost:8000*

Summary

- When you execute something **asynchronously**, you can move on to another task before it finishes
- **Celery** is an asynchronous task **queue/job** queue based on distributed message passing
- **Redis** is an **in-memory** data structure project

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

Postbank
Решения за твоето утре

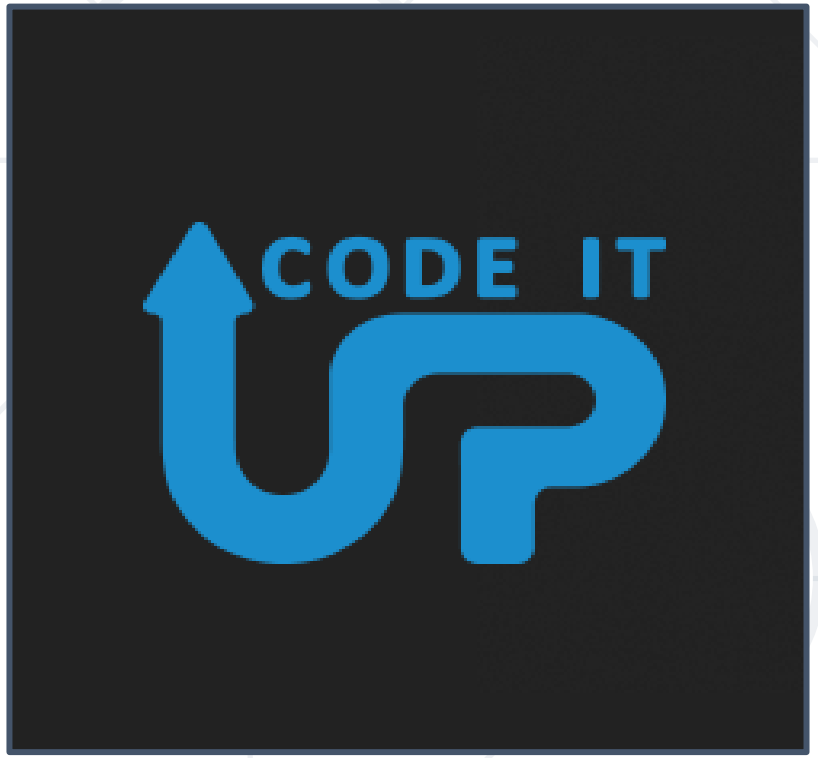
BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Software University



Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg