

Tutorial 03-B to do in class – Remember to upload the repo link to Teams

DIP

Antes de iniciar:

- Terminar los talleres no calificables anteriores
- Este taller muestra el ejemplo de una inversión de dependencias y de una inyección de dependencias.
- ¿Puede identificar donde está la inversión, y donde la inyección?
- Al final vemos un ejemplo de cómo sería sin inversión de dependencias.

A. An interface

Interface

- Go to *pages* create a file *interfaces.py* with the next content:

Add Entire Code

```
from abc import ABC, abstractmethod
from django.http import HttpRequest

class ImageStorage(ABC):

    @abstractmethod # any class that inherits from this one must implement this method
    def store(self, request: HttpRequest):
        pass
```

B. A util library

Image local storage

- Go to *pages* create a file *utils.py* with the next content:

Add Entire Code

```
from django.core.files.storage import default_storage
from django.http import HttpRequest
from .interfaces import ImageStorage
```

```
class ImageLocalStorage(ImageStorage):
    def store(self, request: HttpRequest):
        profile_image = request.FILES.get('profile_image', None)
        if profile_image:
            # Store the image
            file_name = default_storage.save('uploaded_images/' + profile_image.name, profile_image)
            return default_storage.url(file_name)
```

- Go to *helloworld_project/setting.py* and add the following content:

Add bold Code

```
#top of the file
from pathlib import Path
import os
...
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

- Go to *helloworld_project* and create a folder named *media*.
- Go to *helloworld_project/urls.py* and add the following content:

• Add bold Code

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path("", include('pages.urls')),
    path('accounts/', include('accounts.urls')),
]

if settings.DEBUG:
```

```
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

C. A service provider

Provider

- Go to *helloworld_project/settings.py* and add the following content at the bottom of the file:

Add Entire Code

```
IMAGE_STORAGE_CLASS = 'pages.utils.ImageLocalStorage'}
```

Registering the new provider

- In *pages/apps.py*, make the following changes in **bold**.

Modify Bold Code

```
from django.apps import AppConfig
from django.utils.module_loading import import_string
from django.conf import settings

class PagesConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'pages'

    def ready(self):
        ImageStorageClass = import_string(settings.IMAGE_STORAGE_CLASS)
```

D. Rest of the code

View/Controller

- Go to *pages/views.py* and add the next content:

Add bold Code

```
...
from .models import Product
...
```

```
def ImageViewFactory(image_storage):
    class ImageView(View):
        template_name = 'images/index.html'

    def get(self, request):
        image_url = request.session.get('image_url', '')
        return render(request, self.template_name, {'image_url': image_url})

    def post(self, request):
        image_url = image_storage.store(request)
        request.session['image_url'] = image_url
        return redirect('image_index')

    return ImageView
```

Routes

- Go to *pages/urls.py* and add these new routes by adding the next lines at the end of the *urlpatterns* (check **bold**).

Modify Bold Code

```
...
path('image/', ImageViewFactory(ImageLocalStorage()).as_view(), name='image_index'),
path('image/save', ImageViewFactory(ImageLocalStorage()).as_view(), name='image_save'),
```

Template/view

- Go to *pages/templates* create a folder *images*.
- Go to *resources/views/image/* create a file *index.html* with the next content:

Add Entire Code

```
{% extends 'pages/base.html' %}
{% block title %} Image Storage - DI {% endblock %}

{% block content %}
```

```
<div class="container">
  <div class="row justify-content-center">
    <div class="col-md-8">
      <div class="card">
        <div class="card-header">Upload image</div>
        <div class="card-body">
          <form action="{% url 'image_save' %}" method="post" enctype="multipart/form-data">
            {% csrf_token %}
            <div class="form-group">
              <label>Image:</label>
              <input type="file" name="profile_image" />
            </div>
            <button type="submit" class="btn btn-primary">Submit</button>
          </form>

          
        </div>
      </div>
    </div>
  </div>
</div>
{% endblock %}
```

Artisan

- Run the next command in Terminal

Execute in Terminal

```
python manage.py runserver
```

Execution

- Go to <http://127.0.0.1:8000/image/>

A Django EAFIT App

Upload image

Image: Ninguno archivo selec.

Copyright - Daniel Correa

Upload image

- Upload a new image

A Django EAFIT App

Upload image

Image: Ninguno archivo selec.



SAME APPLICATION WITHOUT DEPENDENCY INVERSION

Controller

- Go to *pages/views.py* create a View *ImageNotDIVView* with the next content:

Add Entire Code

```
class ImageViewNoDI(View):
    template_name = 'images/index.html'

    def get(self, request):
        image_url = request.session.get('image_url', '')

        return render(request, self.template_name, {'image_url': image_url})

    def post(self, request):
        image_storage = ImageLocalStorage()
        image_url = image_storage.store(request)
        request.session['image_url'] = image_url

        return redirect('image_index')
```

Routes

- Go to *pages/urls.py* and add these new routes by adding the next lines at the end of the file (check **bold**).

Modify Bold Code

...

```
path('imagenotdi/', ImageViewNoDI.as_view(), name='imagenodi_index'),

path('image/save', ImageViewNoDI.as_view().as_view(), name='imagenodi_save'),
```

View

- Go to *pages/templates/* create a folder *imagesnotdi*.
- Go to *pages/templates /imagesnotdi/* create a file *index.html* with the next content:

Add Entire Code

```
{% extends 'pages/base.html' %}
```

{% block title %} Image Storage - No DI {% endblock %}

{% block content %}

```
<div class="container">
  <div class="row justify-content-center">
    <div class="col-md-8">
      <div class="card">
        <div class="card-header">Upload image</div>
        <div class="card-body">
          <form action="{% url 'imagenotdi_save' %}" method="post" enctype="multipart/form-data">
            {% csrf_token %}
            <div class="form-group">
              <label>Image:</label>
              <input type="file" name="profile_image" />
            </div>
            <button type="submit" class="btn btn-primary">Submit</button>
          </form>

          
        </div>
      </div>
    </div>
  </div>
</div>
{% endblock %}
```


Execution

- Go to <http://127.0.0.1:8000/image-not-di>

A Django EAFIT App

Upload image

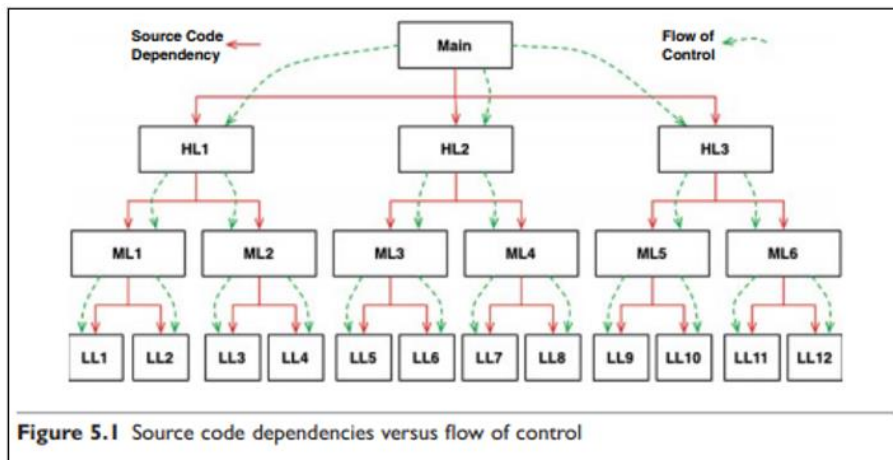
Image: Ninguno archivo selec.



Preguntas

- ¿Puedes entender las diferencias entre las dos propuestas?
- ¿Ventajas/desventajas de cada una?
- Qué tal si tratas de comparar estas propuestas con lo siguiente (ver siguiente página):

Programación estructurada

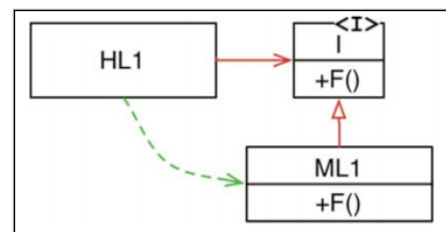


Podemos resumir el paradigma de programación estructurada de la siguiente manera:

La programación estructurada impone disciplina de transferencia directa de control.

Programación orientada a objetos

- En la figura de la derecha, el módulo "HL1" llama a la función F() en el módulo "ML1". El hecho de que llame a esta función a través de una interfaz es una posibilidad que brinda la POO. En tiempo de ejecución, la interfaz no existe. Y finalmente, "HL1" simplemente llama a F() dentro de "ML1".
- Por otro lado, veamos que la dependencia del código fuente (la relación de herencia) entre "ML1" y la interfaz "I" apunta en la dirección opuesta en comparación con el flujo de control. Esto se llama **"inversión de dependencias"**, y sus implicaciones para el arquitecto de software son profundas.



Podemos resumir el paradigma de programación orientada a objetos de la siguiente manera:

La programación orientada a objetos impone disciplina de transferencia indirecta de control.

¿Será que con el ejemplo ya te queda clara esa carreta? ¿Sí? ¿No? La próxima clase conversaremos sobre esto