Indice de contenido

Documentación repositorio digital FCA realizado con django

Descripción General

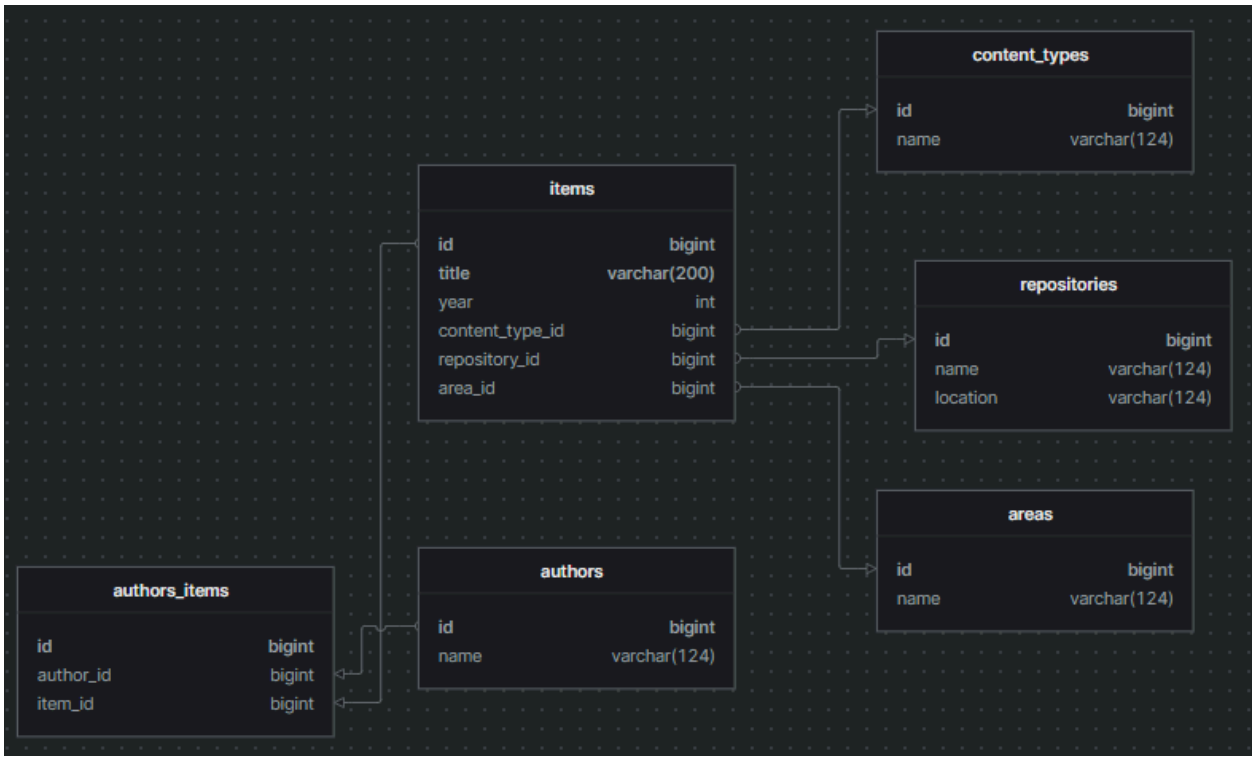
Esta documentación ofrece una visión integral del sistema del Repositorio Digital de la FCA. Este sistema está diseñado para almacenar y facilitar el acceso a una amplia gama de documentos digitales, incluyendo tesis, artículos académicos, reportes de investigación, y otros recursos educativos. A continuación, se describen las principales características y funcionalidades del sistema:

- **Gestión de Contenidos:** Permite la creación de obras dentro del repositorio. Las obras pueden estar asociadas con múltiples autores, áreas temáticas, tipos de contenido y repositorios específicos.
- **Carga de Documentos:** Facilita la subida de documentos en formato PDF. Además, se genera automáticamente una miniatura de la primera página del PDF para una visualización rápida.
- **Búsqueda y Filtros:** Incluye un potente motor de búsqueda que permite a los usuarios filtrar y buscar documentos por título de obra o nombre del autor.
- **Paginación:** Ofrece una navegación paginada para mejorar la experiencia del usuario al explorar grandes cantidades de documentos.
- **Seguridad de Acceso:** Incorpora un sistema de validación mediante clave para asegurar que solo usuarios autorizados puedan acceder a ciertas funcionalidades administrativas, como la creación de nuevas obras.
- **Mensajes de Éxito y Error:** Proporciona retroalimentación al usuario mediante mensajes de éxito o error al realizar operaciones, como la carga de documentos o la validación de acceso.
- **Interfaz Amigable:** Presenta una interfaz intuitiva y fácil de usar, diseñada para mejorar la accesibilidad y eficiencia en la gestión de contenidos.
- **Diseño Responsivo:** El sistema está diseñado con un enfoque responsivo, lo que asegura que la interfaz y todas sus funcionalidades se adapten de manera óptima a cualquier dispositivo, ya sea una computadora de escritorio, una tableta o un teléfono móvil.

Migraciones y Seeders*

Todo lo realizado dentro del contenido de esta seccion sera realizado dentro de la carpeta del programa “library-uabc”

Esquema de la base de datos



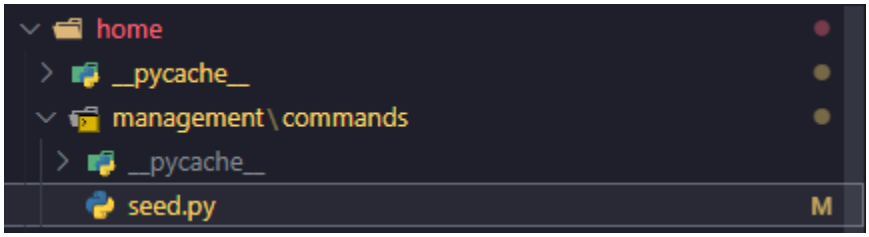
Realizar Migraciones*

Aplicar las migraciones mediante el siguiente comando en la consola para la creación de las tablas en la base de datos:

```
python manage.py migrate
```

Seeders *

Para poblar la base de datos con datos iniciales, puedes modificar el script **seed.py** que se encuentra en el siguiente directorio.



El siguiente código es una parte del archivo **seed.py** en donde deberá agregar los datos necesarios para las tablas:

```
# home/management/commands/seed.py
class Command(BaseCommand):
    def _create_repositories(self):
        repository_data = [
            "Revistas UABC",
            "Blibioteca",
            # agregar los repositorios aquí separadas por coma
        ]
        for repository_name in repository_data:
            Repository.objects.create(name=repository_name)
```

Para ejecutar el script de seeders, ejecuta el siguiente comando en la consola:

```
python manage.py seed
```

Borrar y Reiniciar la Base de Datos

Para borrar todos los datos de la base de datos y reiniciar el esquema:

1. Borrar todos los datos:

```
python manage.py flush
```

Este comando eliminará todos los datos de todas las tablas.

2. Realizar migraciones nuevamente:

```
python manage.py migrate
```

Esto recreará el esquema de la base de datos según las migraciones aplicadas.

3. Aplicar seed nuevamente para poblar la base de datos nuevamente:

```
python manage.py seed
```

Acceso Seguro a la Vista `agregar`

Middleware

Para restringir el acceso a la vista `agregar`, se añadió un middleware para asegurar que solo las sesiones autenticadas puedan acceder.

```
# middleware.py
from django.shortcuts import redirect

class ClearSessionOnNavigationMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        response = self.get_response(request)
        # Clear session if the user navigates away from add_item
        if request.session.get('has_access') and request.path !=
'/contenido/agregar' and response.status_code == 200:
            request.session['has_access'] = False
        return response
```

Vista para Autenticación*

Para la autenticación del formulario de las obras se agregó un vista el cual el profesor deberá ingresar una clave para poder acceder a ello.
Asegúrese de proporcionar una clave segura

```
# views.py
def validacion(request):
    if request.method == 'POST':
        key = request.POST.get('key')
        if key == '1234':  <-- Reemplazar aquí por clave segura
            request.session['has_access'] = True
```

```
            return redirect('agregar')
        else:
            return HttpResponse('Clave invalida', status=403)

    return render(request, 'validacion.html')
```

Manejo de Formularios y Gestión de Errores

Formularios

El formulario utilizado para manejar el formulario de las obras.

```
#forms.py
class ItemForm(forms.ModelForm):
    class Meta:
        model = Item
        fields = ['title', 'year', 'area', 'repository', 'content_type',
'pdf']

#recolectar los datos del formulario antes de ser procesado para la
verificación de errores.
    def clean(self):
        cleaned_data = super().clean()
        pdf = cleaned_data.get('pdf')

        if not pdf:
            raise forms.ValidationError("Porfavor suba un archivo pdf.")

        if pdf and not pdf.name.endswith('.pdf'):
            raise forms.ValidationError("Solo se admiten archivos pdf.")

        return cleaned_data
```

Vista para Manejo de Formularios

La vista para manejar la presentación de formularios y la gestión de errores.

```
#views.py
def agregar(request):
    if not request.session.get('has_access'):
        return redirect('validacion')

    if request.method == 'POST':
        form = ItemForm(request.POST, request.FILES)
        formset = AuthorItemFormSet(request.POST)
```

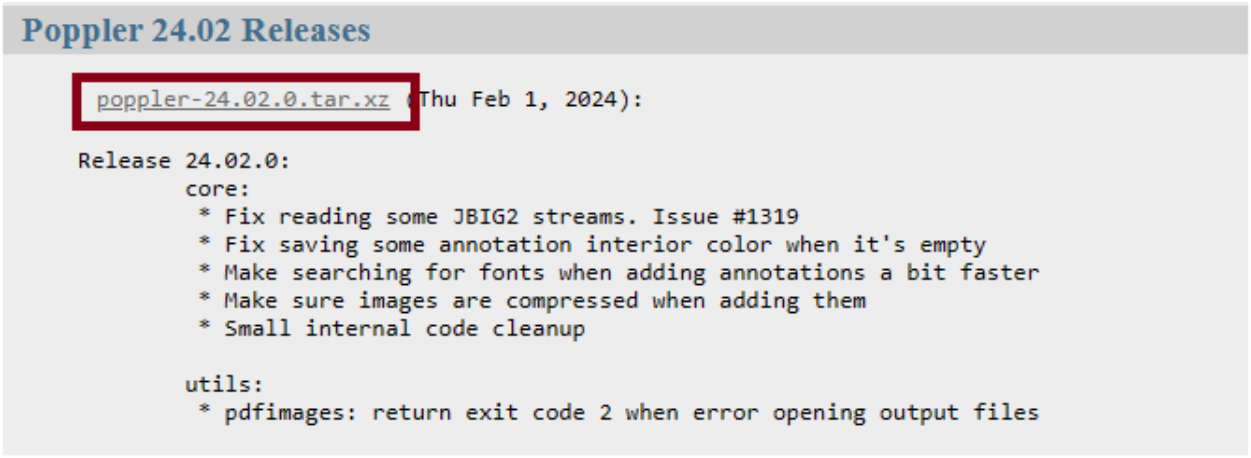
```
      #validación del formulario
        if form.is_valid() and formset.is_valid():
            try:
                item = form.save()
                pdf = request.FILES['pdf']
                pdf_path = os.path.join(settings.MEDIA_ROOT, item.pdf.name)

                #Ciclo para agregar los autores que se requieran
                for author_form in formset:
                    if author_form.cleaned_data:
                        first_name = author_form.cleaned_data.get('first_name')
                        last_name = author_form.cleaned_data.get('last_name')
                        if first_name and last_name:
                            author, created = Author.objects.get_or_create(
                                first_name=first_name,
                                last_name=last_name
                            )
                            AuthorItem.objects.create(author=author, item=item)


            #Convertir la portada del pdf a una imagen para almacenarla en la
obra con id unico
                thumbnail_path = os.path.join(settings.MEDIA_ROOT,
f'thumbnails/{item.id}.jpg')
                generate_pdf_thumbnail(pdf_path, thumbnail_path)

                messages.success(request, 'La obra fue subida exitosamente!')
                form = ItemForm()
                formset = AuthorItemFormSet()
      #Mostrar mensaje de error si es que se presenta un error no identificado
            except Exception as e:
                messages.error(request, f'Error: {str(e)}')
    else:
        form = ItemForm()
        formset = AuthorItemFormSet()

    return render(request, 'agregar.html', {'form': form, 'formset': formset})
```

Procesamiento de PDFs*

Para poder procesar los archivos PDF y almacenar las portadas como archivos JPEG, es necesario descargar e instalar Poppler en https://poppler.freedesktop.org/releases.html. A continuación se detallan los pasos para realizar esta instalación:

Instalación de Poppler en Windows

1. Descarga y extracción del Archivo:
Una vez descargado el archivo va a extraer el archivo dentro del disco local “C:” en la carpeta de archivos de programa “C:\Program Files”. Para esto debe tener instalado un programa que permita descomprimir archivos (como WinRAR o 7zip), ya que el archivo binario se descarga en formato comprimido.



Poppler 24.02 Releases

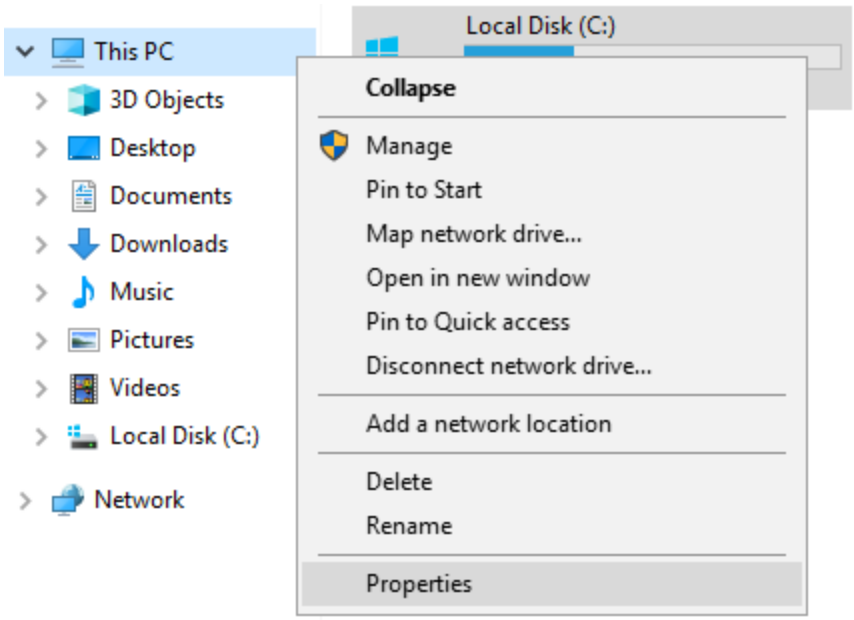
poppler-24.02.0.tar.xz (Thu Feb 1, 2024):

```
Release 24.02.0:
        core:
         * Fix reading some JBIG2 streams. Issue #1319
         * Fix saving some annotation interior color when it's empty
         * Make searching for fonts when adding annotations a bit faster
         * Make sure images are compressed when adding them
         * Small internal code cleanup

        utils:
         * pdfimages: return exit code 2 when error opening output files
```

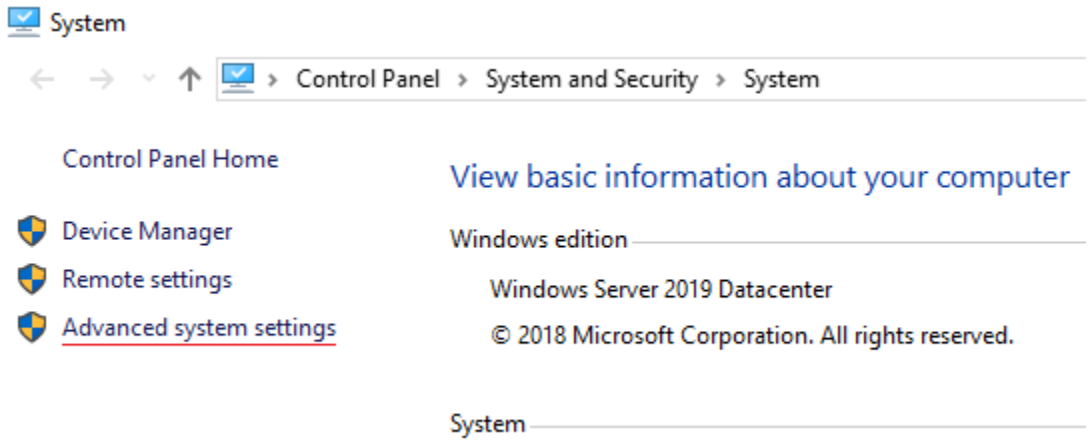
2. Agregar ruta

Agregar la ruta de la carpeta bin de poppler “C:\Program Files\poppler-x\bin” al path del sistema haciendo lo siguiente:

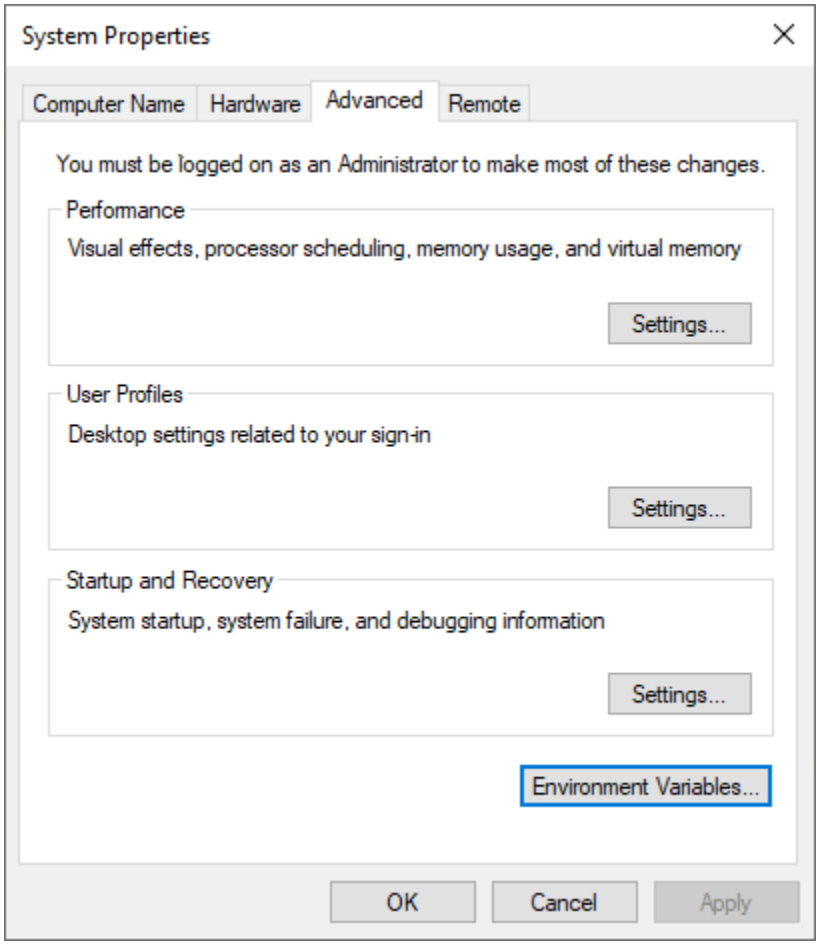
- En el explorador de archivos hacer clic derecho sobre “This PC” (“Este equipo”) (* se presentarán los nombres en español dentro de paréntesis como referencia para los usuarios que tengan el sistema operativo en ese idioma) y luego clic en “Properties” (“Propiedades”).



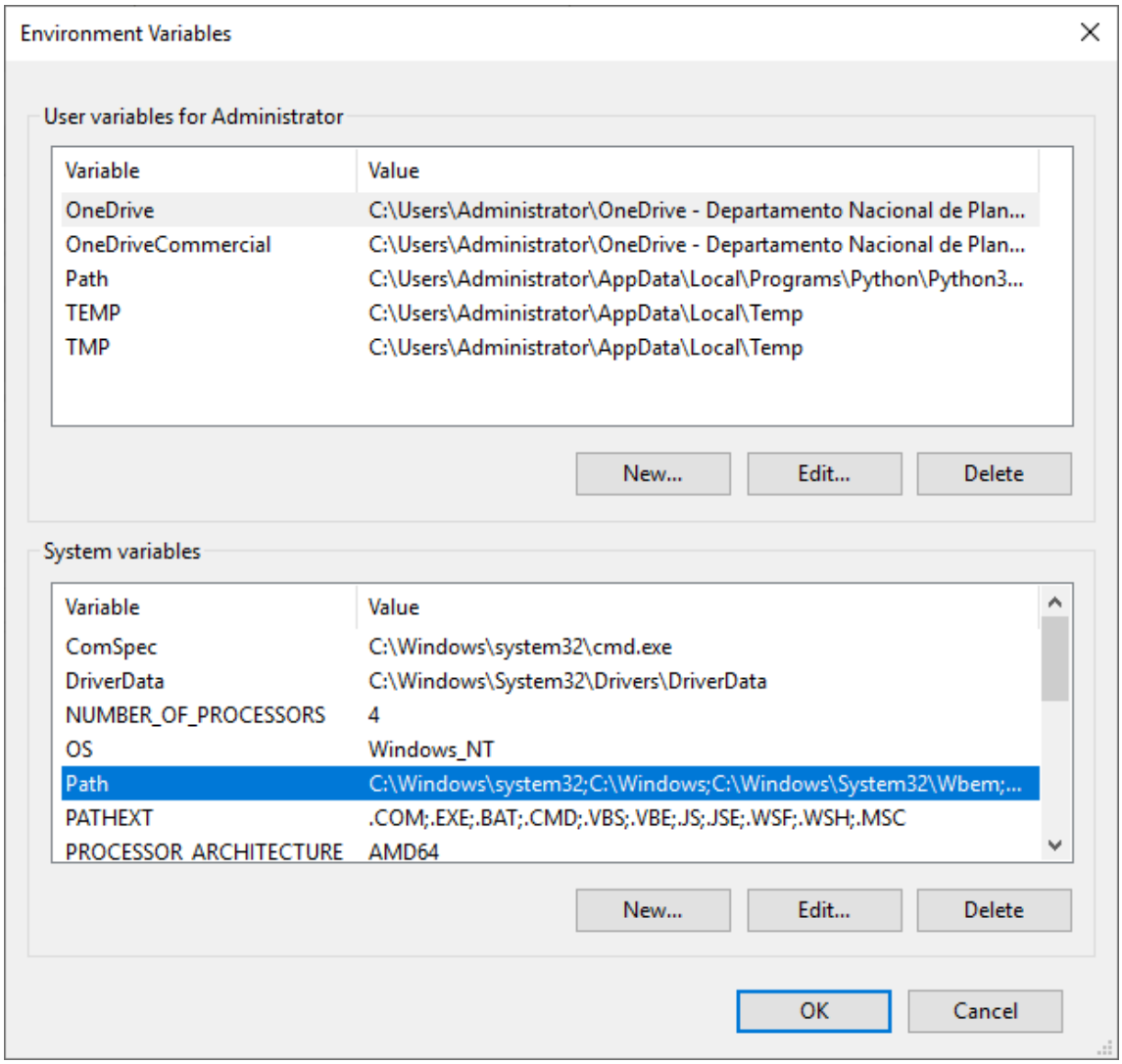
- Hacer clic en la opción “Advanced system settings” (“Configuración avanzada del sistema”) ubicada en la parte superior izquierda.



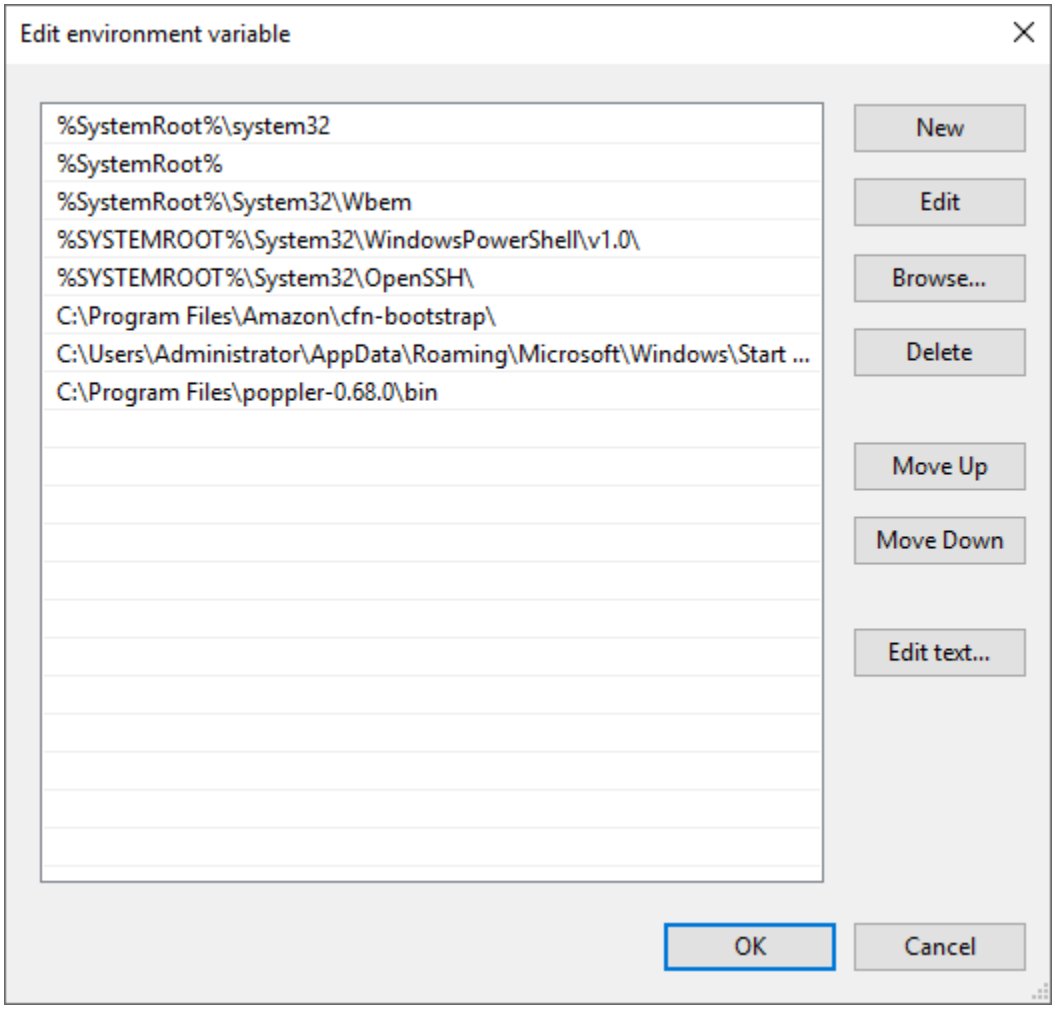
- Se abrirá una nueva ventana de “System Properties” (“Propiedades del sistema”), aquí se debe hacer clic en la opción de “Environment Variables…” (“Variables de entorno…”).



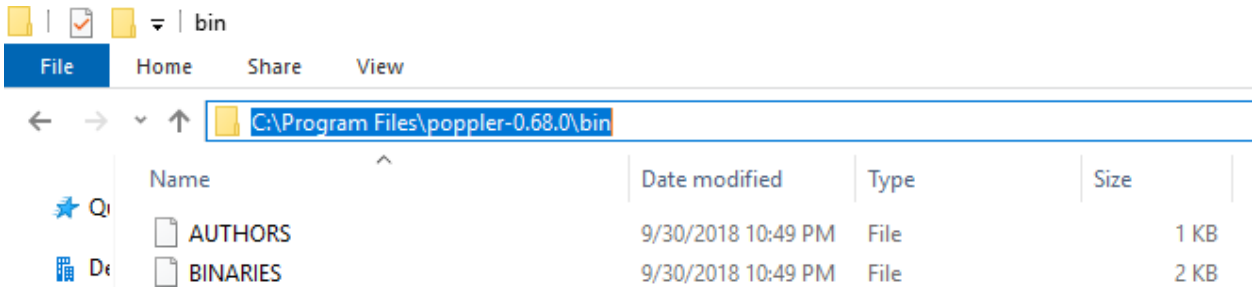
- En la sección de “System variables” (“Variables del sistema”) se debe hacer doble clic en la variable “Path”.



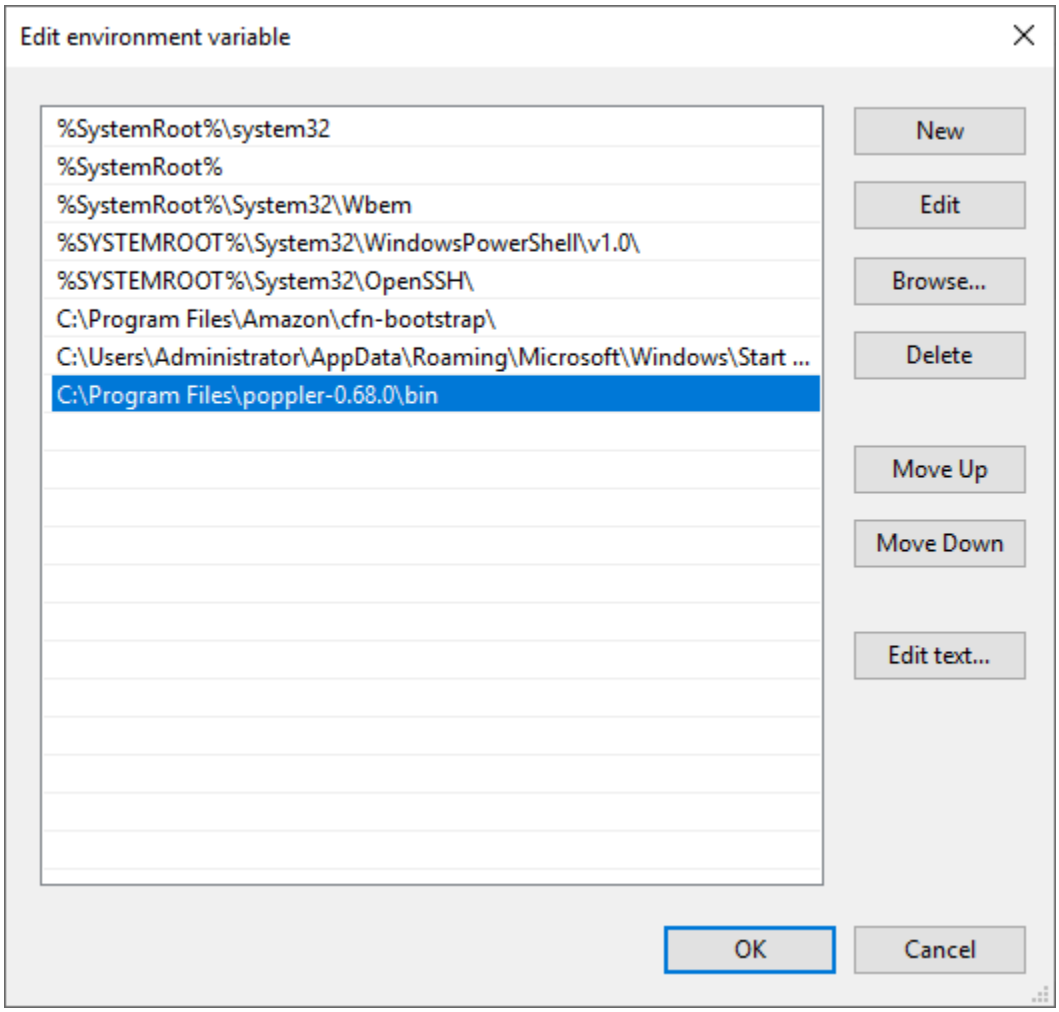
- Se abrirá una nueva ventana de edición de las variables de entorno, al hacer clic en el botón “New” (“Nuevo”) se podrá ingresar la ruta de la carpeta bin que está dentro del archivo que se extrajo previamente.



- Se debe copiar la ruta donde se encuentra la carpeta bin de poppler, la ruta debería ser “C:\Program Files\poppler-x\bin” si el archivo binario de poppler se extrajo en archivos de programa.



- Finalmente, se agrega la ruta “C:\Program Files\poppler-x\bin” en la ventana de edición de variables de entorno y se hace clic en “OK” (“Aceptar”) para guardar los cambios y cerrar las ventanas.



- Igualmente en el archivo **views.py** vamos a agregar la misma ruta en la variable poppler_path como se puede ver a continuación:

```python
def generate_pdf_thumbnail(pdf_path, thumbnail_path):
    pages = convert_from_path(pdf_path, 60, poppler_path=r'E:\poppler-24.02.0\Library\bin')
    page = pages[0]
    page.save(thumbnail_path, 'JPEG')
```

-

Si estos pasos no se realizan correctamente no se podrá visualizar las portadas de los pdf.*

Instalación de Poppler en macOS

1. Instalación usando Homebrew:

- Abre la terminal.

- Si no tienes Homebrew instalado, instálalo ejecutando el siguiente comando:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

- Una vez que Homebrew esté instalado, instala Poppler ejecutando el siguiente comando:

```
brew install poppler
```

Instalación de Poppler en Linux

1**. Uso del Gestor de Paquetes:**

- Abre la terminal.

- En distribuciones basadas en Debian (como Ubuntu), puedes instalar Poppler ejecutando:

```
sudo apt-get install poppler-utils
```

- En distribuciones basadas en Red Hat (como Fedora), puedes instalar Poppler ejecutando:

```
sudo dnf install poppler
```

Verificación de la Instalación

Para verificar que Poppler se haya instalado correctamente, abre una terminal o línea de comandos y ejecuta el siguiente comando:

```
pdfinfo -v
```

Funcionamiento e instalación *

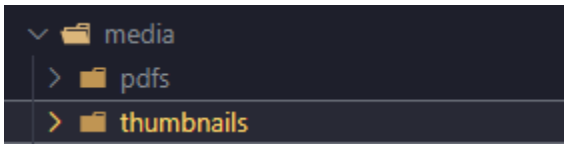
Ya instalado poppler en el sistema asegúrese de ejecutar el siguiente comando en la consola dentro del directorio en donde se encuentra el programa “library-uabc”:

```
pip install pdf2image
```

Una función de utilidad para generar miniaturas a partir de PDFs.

```python
#views.py
def generate_pdf_thumbnail(pdf_path, thumbnail_path):
    pages = convert_from_path(pdf_path, 60)
    page = pages[0]
    page.save(thumbnail_path, 'JPEG')
```

Los pdf y portada de la misma se encuentran localizados en la siguientes carpetas:



Paginación y Filtrado

Vista para Paginación y Filtrado

Manejo de paginación y filtrado en la vista.

```
#views.py
def contenido (request):
    #obtener parámetros del url para poder utilizarlos para hacer
peticiones a la base de datos con ciertos filtros.
    search = request.GET.get('search', '')
    year = request.GET.get('year', '')
    content_type = request.GET.get('content_type', '')
    area = request.GET.get('area', '')
    repository = request.GET.get('repository', '')
    page_number = request.GET.get('page', 1)
    author_id = request.GET.get('author_id', '')

    items = Item.objects.all().prefetch_related('authors')

    #Barra de busqueda.
    #Permite hacer la búsqueda por título de obra o nombre del profesor.
    if search:
        search_terms = search.split()
        if len(search_terms) == 2:
            first_name, last_name = search_terms
            items = items.filter(
                Q(title__icontains=search) |
```

```
                Q(authors__first_name__icontains=first_name) &
                Q(authors__last_name__icontains=last_name)
            ).distinct()
        else:
            items = items.filter(
                Q(title__icontains=search) |
                Q(authors__first_name__icontains=search) |
                Q(authors__last_name__icontains=search)
            ).distinct()

    #Filtros
    if year:
        items = items.filter(year=year)

    if content_type:
        items = items.filter(content_type=content_type)

    if area:
        items = items.filter(area=area)

    if repository:
        items = items.filter(repository=repository)

    if author_id:
        items = items.filter(authors__id=author_id).distinct()

    #Paginación con un límite de 12 obras por página
    paginator = Paginator(items, 12)
    page_obj = paginator.get_page(page_number)

    #Recuperar los datos de las tablas de repositorio, áreas, etc para ser
desplegados en los filtros.
    content_types = Item.objects.values('content_type_id',
'content_type__name').annotate(num_items=Count('id'))
    areas = Item.objects.values('area_id',
'area__name').annotate(num_items=Count('id'))
    repositories = Item.objects.values('repository_id',
'repository__name').annotate(num_items=Count('id'))
    year_production =
Item.objects.values('year').annotate(recursos=Count('year'))
```

```
    last_added = Item.objects.all().order_by('-id').values()[:50]
    authors = Author.objects.all()

    context = {
        'items': page_obj,
        'content_types': content_types,
        'areas': areas,
        'repositories': repositories,
        'year_production': year_production,
        'last_added': last_added,
        'author_id': author_id,
        'search': search,
        'year': year,
        'content_type': content_type,
        'area': area,
        'repository': repository,
        'paginator': paginator,
        'page_obj': page_obj,
        'authors': authors
    }

    return render(request, 'contenido.html', context)
```