

# P1 - PSI

Iniciar postgresql (laboratorios EPS)

```
sudo systemctl restart postgresql
```

Configurar entorno virtual Python

```
python3 -m venv venv  
source venv/bin/activate  
pip install -r requirements.txt
```

Reiniciar base de datos

```
dropdb -U alumnodb -h localhost psi  
createdb -U alumnodb -h localhost psi  
python3 populate_catalog.py
```

Crear el proyecto

```
django-admin startproject <project-name>
```

Crear aplicación

```
python3 manage.py startapp <name>
```

Registrar aplicación

Añadir a settings.py en la lista INSTALLED\_APPS: 'name.apps.NameConfig'

Definir urls (ejemplo catalog)

```
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('catalog/', include('catalog.urls')),  
    path('', RedirectView.as_view(url='catalog/')),  
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Actualizar base de datos

```
python3 manage.py makemigrations  
python3 manage.py migrate
```

Arrancar la web

```
python3 manage.py runserver 8001
```

Crear superusuario

```
python3 manage.py createsuperuser
```

### Añadir CSS estático

```
<!-- Add additional CSS in static file -->
{% load static %}
<link rel="stylesheet" href="{% static 'css/styles.css' %}" />
```

### O una imagen

```
{% load static %}

```

### Usar una clase para view (Con muchas opciones extra)

```
class BookListView(generic.ListView):
    model = Book
    context_object_name = 'book_list' # your own name for the list as a template
    variable
    queryset = Book.objects.filter(title__icontains='war')[:5] # Get 5 books
    containing the title war
    template_name = 'books/my_arbitrary_template_name_list.html' # Specify your
    own template name/location

    def get_context_data(self, **kwargs):
        # Call the base implementation first to get the context
        context = super(BookListView, self).get_context_data(**kwargs)
        # Create any data and add it to the context
        context['some_data'] = 'This is just some data'
        return context
```

### Ejemplo sin usar clase

```
def book_detail_view(request, primary_key):
    try:
        book = Book.objects.get(pk=primary_key)
    except Book.DoesNotExist:
        raise Http404('Book does not exist')

    return render(request, 'catalog/book_detail.html', context={'book': book})
```

### Alternativa

```
from django.shortcuts import get_object_or_404

def book_detail_view(request, primary_key):
    book = get_object_or_404(Book, pk=primary_key)
    return render(request, 'catalog/book_detail.html', context={'book': book})
```

### Sesiones

```

# Get a session value by its key (e.g. 'my_car'), raising a KeyError if the key
is not present
my_car = request.session['my_car']

# Get a session value, setting a default if it is not present ('mini')
my_car = request.session.get('my_car', 'mini')

# Set a session value
request.session['my_car'] = 'mini'

# Delete a session value
del request.session['my_car']

# Set session as modified to force data updates/cookie to be saved.
request.session.modified = True

```

#### Restringir acceso a una vista

```

from django.contrib.auth.decorators import login_required

@login_required
def my_view(request):
    # ...

```

#### Comprobar permisos en template

```

{% if perms.catalog.can_mark_returned %}
    <!-- We can mark a BookInstance as returned. -->
    <!-- Perhaps add code to link to a "book return" view here. -->
{% endif %}

```

#### Comprobar permisos en view

```

from django.contrib.auth.decorators import permission_required

@permission_required('catalog.can_mark_returned')
@permission_required('catalog.can_edit')
def my_view(request):
    # ...

```

#### Comprobar permisos en view (alternativa con Mixin)

```

from django.contrib.auth.mixins import PermissionRequiredMixin

class MyView(PermissionRequiredMixin, View):
    permission_required = 'catalog.can_mark_returned'
    # Or multiple permissions
    permission_required = ('catalog.can_mark_returned', 'catalog.change_book')
    # Note that 'catalog.change_book' is permission
    # Is created automatically for the book model, along with add_book, and
    delete_book
    # ...

```

## ModelForm

```
class RenewBookModelForm(ModelForm):  
    class Meta:  
        model = BookInstance  
        fields = ['due_back']  
        labels = {'due_back': _('New renewal date')}  
        help_texts = {'due_back': _('Enter a date between now and 4 weeks (default 3).')}]
```

## Ejemplo de .env

```
POSTGRES_URL="postgres://alumnodb:alumnodb@localhost:5432/psi"  
DEBUG=true  
TESTING=true  
ALLOWED_HOSTS='*'  
SECRET_KEY='4fa236d892eeeb1cbce626e8eecf097e'
```

CreateView, UpdateView, DeleteView