# Tutorial Completo: Desarrollo de un Gestor de Cursos con Django

## Índice



## Introducción

Este tutorial te guiará en la creación de una aplicación web completa para gestionar cursos online y documentación técnica usando Django. La aplicación permite a los usuarios:

- Organizar cursos por rutas de aprendizaje
- Marcar cursos como completados
- Recibir recomendaciones personalizadas
- Gestionar documentación técnica
- Personalizar la interfaz

### ¿Por qué Django?

Django es un framework web de alto nivel que fomenta el desarrollo rápido y el diseño limpio y pragmático. Elegimos Django por:

- Robusta arquitectura MVT (Model-View-Template)
- ORM potente para gestión de base de datos
- Sistema de administración automático
- Seguridad incorporada
- Excelente documentación

# Configuración del Entorno

## 1. Preparación del Entorno Virtual

```
# Crear un entorno virtual con Anaconda
conda create -n py311 python=3.11
conda activate py311

# Instalar dependencias
pip install django==4.2
pip install pillow  # Para procesamiento de imágenes
pip install python-dotenv  # Para variables de entorno
pip install requests  # Para peticiones HTTP
```

## 2. Crear el Proyecto Django

```
django-admin startproject course_manager
cd course_manager
python manage.py startapp courses
```

## 3. Configuración Inicial (settings.py)

```
INSTALLED_APPS = [
    # ...
    'courses',
]

# Configuración de archivos estáticos y media
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

## Estructura del Proyecto

La estructura del proyecto sigue el patrón MVT de Django:

```
course_manager/
├── courses/
│   ├── models.py      # Definición de datos
│   ├── views.py       # Lógica de negocio
│   ├── forms.py       # Formularios
│   ├── urls.py        # Rutas URL
│   └── apps.py        # Configuración de la app
├── templates/
│   └── courses/
│       ├── base.html          # Template base
│       ├── home.html          # Página principal
│       ├── add_course.html    # Formulario de cursos
│       └── documentation.html # Gestión de docs
└── static/
    └── css/
        └── styles.css
```

## Modelos de Datos

### Route (models.py)

```
class Route(models.Model):
    name = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name
```

Este modelo representa una ruta de aprendizaje:

- `name` : Nombre de la ruta (ej: "Desarrollo Web", "Machine Learning")
- `created_at` : Fecha de creación (se establece automáticamente)
- `__str__` : Método que define cómo se muestra el objeto

## Course (models.py)

```
class Course(models.Model):
    PLATFORM_CHOICES = [
        ('youtube', 'YouTube'),
        ('udemy', 'Udemy'),
        ('platzi', 'Platzi'),
        # ...
    ]

    title = models.CharField(max_length=200)
    url = models.URLField()
    platform = models.CharField(max_length=20,
choices=PLATFORM_CHOICES)
    language = models.CharField(max_length=20,
choices=LANGUAGE_CHOICES)
    route = models.ForeignKey(Route, on_delete=models.CASCADE)
    thumbnail = models.ImageField(upload_to='thumbnails/')
    completed = models.BooleanField(default=False)
    completed_at = models.DateTimeField(null=True, blank=True)
```

Explicación del modelo Course:

- `PLATFORM_CHOICES` : Lista de plataformas soportadas
- `title` : Título del curso

- `url` : Enlace al curso
- `platform` : Plataforma del curso (usando choices para validación)
- `language` : Lenguaje de programación
- `route` : Relación ForeignKey con Route
- `thumbnail` : Imagen miniatura del curso
- `completed` : Estado de finalización
- `completed_at` : Fecha de finalización

### Documentation (models.py)

```
class Documentation(models.Model):
    CATEGORY_CHOICES = [
        ('framework', 'Framework'),
        ('language', 'Lenguaje de Programación'),
        ('library', 'Librería'),
        ('tool', 'Herramienta'),
    ]

    title = models.CharField(max_length=200)
    url = models.URLField()
    description = models.TextField(blank=True)
    category = models.CharField(max_length=20,
choices=CATEGORY_CHOICES)
    icon_url = models.URLField(blank=True)
    favorite = models.BooleanField(default=False)
```

## Vistas y Lógica de Negocio

### Vista Principal (views.py)

```python
def home(request):
    routes = Route.objects.all()
    courses = Course.objects.all().order_by('-created_at')
    return render(request, 'courses/home.html', {
        'routes': routes,
        'courses': courses
    })
```

Esta vista:

1. Obtiene todas las rutas y cursos
2. Ordena los cursos por fecha de creación
3. Renderiza el template con los datos

## Sistema de Recomendaciones (views.py)

```python
def recommendations(request):
    # Obtener rutas más frecuentes
    user_routes = Course.objects.values('route').annotate(
        total_courses=Count('id')
    ).order_by('-total_courses')[:3]

    # Obtener lenguajes más usados
    user_languages = Course.objects.values_list('language', flat=True)
    common_languages = Counter(user_languages).most_common(3)

    # Generar recomendaciones
    recommendations = {
        'by_route': {},
        'by_language': {},
        'similar_courses': []
    }
```

Este sistema:

1. Analiza patrones de uso
2. Identifica preferencias del usuario
3. Genera recomendaciones personalizadas

# Formularios

## CourseForm (forms.py)

```
class CourseForm(forms.ModelForm):
    class Meta:
        model = Course
        fields = ['title', 'url', 'platform', 'language', 'route']
        widgets = {
            'title': forms.TextInput(attrs={'class': 'form-control'}),
            'url': forms.URLInput(attrs={'class': 'form-control'}),
            # ...
        }
```

Este formulario:

- Está basado en el modelo Course
- Define campos específicos a mostrar
- Personaliza widgets con clases Bootstrap

# Templates y Frontend

## Base Template (base.html)

```
<!DOCTYPE html>
<html>
<head>
    <title>Gestor de Cursos</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.
min.css" rel="stylesheet">
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.4/css/all.min.css" rel="stylesheet">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <!-- Navegación -->
    </nav>

    <div class="container mt-4">
        {% block content %}
        {% endblock %}
    </div>

    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bu
ndle.min.js"></script>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js">
</script>
</body>
</html>
```

## Documentación Template (documentation_list.html)

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

{% block content %}
<div class="container">
    <!-- Cards de documentación -->
    <div class="row">
        {% for doc in docs %}
        <div class="col-md-4 mb-4">
            <div class="card doc-card">
                <!-- Contenido de la tarjeta -->
            </div>
        </div>
        {% endfor %}
    </div>
</div>
{% endblock %}
```

## Mejores Prácticas y Optimizaciones

### 1. Organización del Código

- Separar lógica en módulos
- Usar nombres descriptivos
- Documentar funciones importantes

### 2. Seguridad

- Usar CSRF tokens
- Validar entradas de usuario
- Proteger rutas sensibles

### 3. Rendimiento

- Optimizar consultas a la base de datos
- Usar caché cuando sea posible
- Minimizar peticiones AJAX

### 4. UX/UI

- Diseño responsive
- Feedback visual para acciones
- Mensajes de error claros

## Conclusión

Este proyecto demuestra:

1. Arquitectura MVT de Django
2. Gestión de datos relacionales
3. Frontend interactivo
4. Buenas prácticas de desarrollo

Para expandir el proyecto, considera:

- Implementar autenticación de usuarios
- Agregar tests automatizados
- Mejorar el sistema de recomendaciones
- Implementar búsqueda avanzada

## Recursos Adicionales

- Documentación oficial de Django (https://docs.djangoproject.com/)
- Bootstrap Documentation (https://getbootstrap.com/docs/)
- MDN Web Docs (https://developer.mozilla.org/)