PROGRAMACIÓN WEB 2

Profesor(a):
Carlo Jose Luis Corrales Delgado

Estudiantes:
Mamani Anahua, Victor Narciso
Cuno Cahuari, Armando Steven

Repositorio GitHub:
https://github.com/VictorMA18/Lab06-Django

Video:

28 de mayo, 2024

Clases en Models.py

Este código define tres modelos en Django: Alumno, Curso, y Nota. El modelo Alumno tiene un campo nombre de tipo CharField con un máximo de 255 caracteres. El modelo Curso contiene un campo curso que es un CharField de hasta 100 caracteres y debe ser único. El modelo Nota establece relaciones con los modelos Alumno y Curso mediante claves foráneas (ForeignKey), y tiene tres campos nota, nota2, y nota3, todos ellos de tipo DecimalField con un máximo de cuatro dígitos y dos decimales, inicializados en 0. Además, cada modelo tiene un método str que retorna una representación legible del objeto.

```python
from django.db import models

class Alumno(models.Model):
    nombre = models.CharField(max_length=255)

    def __str__(self):
        return self.nombre

class Curso(models.Model):
    curso = models.CharField(max_length=100, unique=True)
    def __str__(self):
        return self.curso

class Nota(models.Model):
    alumno = models.ForeignKey(Alumno, on_delete=models.CASCADE)
    curso = models.ForeignKey(Curso, on_delete=models.CASCADE)
    nota = models.DecimalField(max_digits=4, decimal_places=2, default=0)
    nota2 = models.DecimalField(max_digits=4, decimal_places=2, default=0)
    nota3 = models.DecimalField(max_digits=4, decimal_places=2, default=0)
```

Figura 1: Codigo de Models

Funciones en Views.py

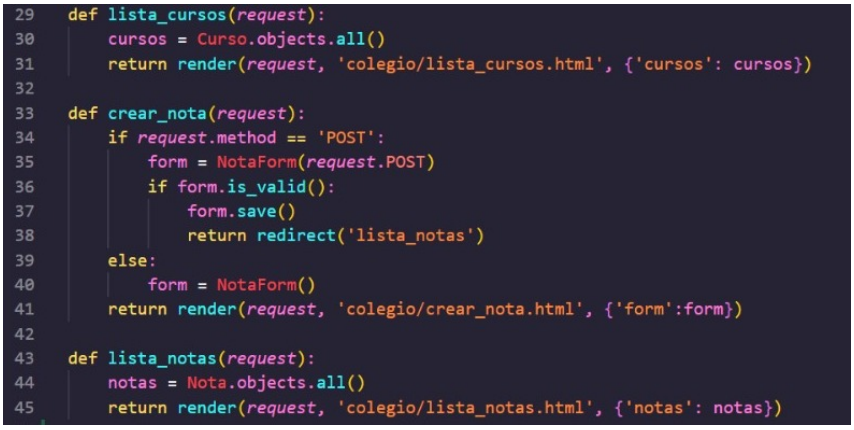
El código proporciona seis-funciones de vista en Django para gestionar un sistema escolar. La función crear-alumno(request) maneja la creación de nuevos registros de alumnos, mientras que lista-alumnos(request) recupera y muestra todos los registros de alumnos. Por otro lado, crear-curso(request) gestiona la creación de nuevos registros de cursos, y lista-cursos(request) muestra todos los cursos existentes. Además, crear-nota(request) permite la creación de nuevas notas, y lista-notas(request) muestra todas las notas registradas. Cada función valida y procesa los formularios correspondientes, redirigiendo a las páginas de lista respectivas después de guardar los datos en la base de datos. En conjunto, estas funciones proporcionan una interfaz completa para gestionar los datos de alumnos, cursos y notas en el sistema de notas.

```python
from django.shortcuts import render, redirect
from .forms import AlumnoForm, CursoForm, NotaForm
from .models import Alumno, Curso, Nota
# Create your views here.
def crear_alumno(request):
    if request.method == 'POST':
        form = AlumnoForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('lista_alumnos')
    else:
        form = AlumnoForm()
    return render(request, 'colegio/crear_alumno.html', {'form': form})

def lista_alumnos(request):
    alumnos = Alumno.objects.all()
    return render(request, 'colegio/lista_alumnos.html', {'alumnos': alumnos})

def crear_curso(request):
    if request.method == 'POST':
        form = CursoForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('lista_cursos')
    else:
        form = CursoForm()
    return render(request, 'colegio/crear_curso.html', {'form':form})
```

Figura 2: Codigo de Views



```python
def lista_cursos(request):
    cursos = Curso.objects.all()
    return render(request, 'colegio/lista_cursos.html', {'cursos': cursos})

def crear_nota(request):
    if request.method == 'POST':
        form = NotaForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('lista_notas')
    else:
        form = NotaForm()
    return render(request, 'colegio/crear_nota.html', {'form':form})

def lista_notas(request):
    notas = Nota.objects.all()
    return render(request, 'colegio/lista_notas.html', {'notas': notas})
```

Figura 3: Codigo de Views

Funciones en Forms.py

El código proporciona tres formularios en Django para los modelos de datos relacionados con un sistema escolar. El formulario AlumnoForm está asociado al modelo Alumno, permitiendo la creación de nuevos registros de alumnos con un campo para el nombre del alumno. CursoForm está vinculado al modelo Curso, facilitando la creación de nuevos registros de cursos. Por último, NotaForm se asocia al modelo Nota, ofreciendo campos para ingresar información sobre la nota de un alumno en un curso específico, incluyendo múltiples notas si es necesario. Cada formulario está diseñado para interactuar directamente con su respectivo modelo, simplificando el proceso de captura y almacenamiento de datos relacionados con el sistema escolar.

```python
from django import forms
from .models import Alumno, Curso, Nota

class AlumnoForm(forms.ModelForm):
    nombre = forms.CharField(max_length=255, label='Nombre del Alumno')

    class Meta:
        model = Alumno
        fields = ['nombre']

    def save(self, commit=True):
        alumno = super().save(commit=False)
        if commit:
            alumno.save()
        return alumno

class CursoForm(forms.ModelForm):
    class Meta:
        model = Curso
        fields = ['curso']

class NotaForm(forms.ModelForm):
    class Meta:
        model = Nota
        fields = ['alumno', 'curso', 'nota', 'nota2', 'nota3']
```

Figura 4: Codigo de Forms

Funciones en Urls.py en la aplicación

El bloque de código define un conjunto de rutas en Django que asignan solicitudes HTTP a funciones de vista específicas dentro de la aplicación. Cada ruta, configurada mediante el método path(), está asociada a una función de vista correspondiente en el módulo de vistas. Por ejemplo, /crearAlumno/ dirige las solicitudes a la función crear-alumno, encargada de la creación de nuevos registros de alumnos, mientras que /listaAlumno/ muestra la lista de todos los alumnos registrados utilizando la función lista-alumnos. Además, /crearNota/ dirige las solicitudes a la función crear-nota para crear nuevas notas, y /listaNota/ muestra todas las notas registradas utilizando la función lista-notas. Este enfoque permite a los usuarios interactuar con diversas funcionalidades del sistema de notas, como la gestión de alumnos, cursos y notas, a través de URLs específicas y funciones de vista asociadas.

```python
from django.urls import path
from . import views

urlpatterns = [
    path('crearAlumno/', views.crear_alumno, name='crear_alumno'),
    path('listaAlumno/', views.lista_alumnos, name='lista_alumnos'),
    path('crearCurso/', views.crear_curso, name='crear_curso'),
    path('listaCurso/', views.lista_cursos, name='lista_cursos') ,
    path('crearNota/', views.crear_nota, name='crear_nota'),
    path('listaNota/', views.lista_notas, name='lista_notas'),
]
```

Figura 5: Codigo de Urls.py en la aplicación

Funciones en Urls.py en el proyecto

El bloque de código configura las rutas principales de la aplicación Django. La ruta /admin/ dirige a la interfaz de administración predeterminada de Django, mientras que la ruta /notas/ es manejada por la aplicación colegio.urls, que a su vez incluye un conjunto de rutas definidas en el archivo urls.py

de la aplicación colegio. Esto permite una estructura de URL modular, donde las rutas relacionadas con las notas (definidas en colegio.urls) se pueden gestionar de manera independiente de las rutas de administración u otras partes de la aplicación.

```python
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('notas/', include('colegio.urls'))
]
```

Figura 6: Codigo de Urls.py en el proyecto

Los archivos HTML

crear_alumno.html

```html
{% load static %}
<!DOCTYPE html>
<html>
<head>
    <title>Crear Alumnos</title>
    <link rel="stylesheet" type="text/css" href="{% static 'css/style_formulario.css'%}"></link>
</head>
<body>
    <div class="content">
        <h1>Crear Alumno</h1>
        <form method="post">
            {% csrf_token %}
            {{ form.as_p }}
            <button type="submit">Guardar</button>
        </form>
        <a href="{% url 'lista_alumnos' %}">Ver todas los alumnos</a>
    </div>
</body>
</html>
```

Figura 7: Codigo de crearAlumno.html

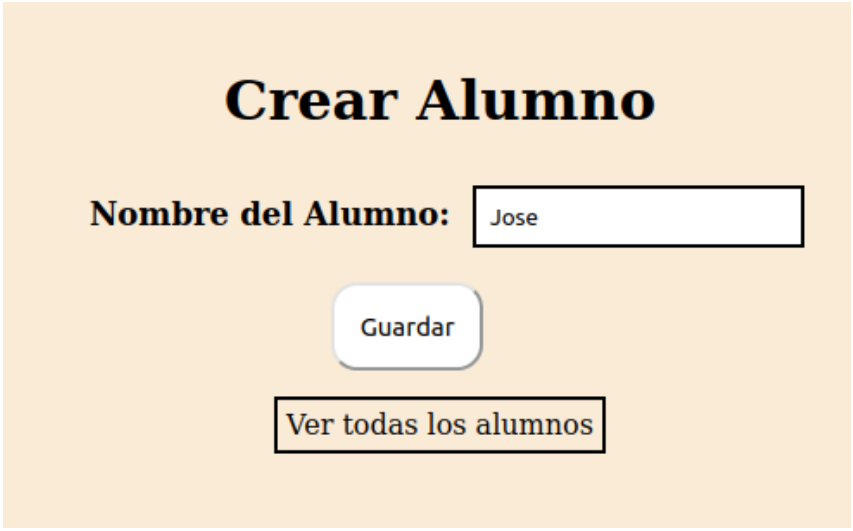


Figura 8: Pagina

lista_alumnos.html

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
    <title>Lista de Alumnos</title>
    <link rel="stylesheet" type="text/css" href="{% static 'css/style_lista.css'%}"></link>
</head>
<body>
    <div class="content">
        <h1>Lista de Alumnos</h1>
        <ul>
            {% for alumno in alumnos %}
                <li>{{ alumno.nombre }}</li>
            {% endfor %}
        </ul>
        <a href="{% url 'crear_alumno' %}">Crear nuevo alumno</a>
    </div>
</body>
</html>
```

Figura 9: Codigo de listaAlumnos.html

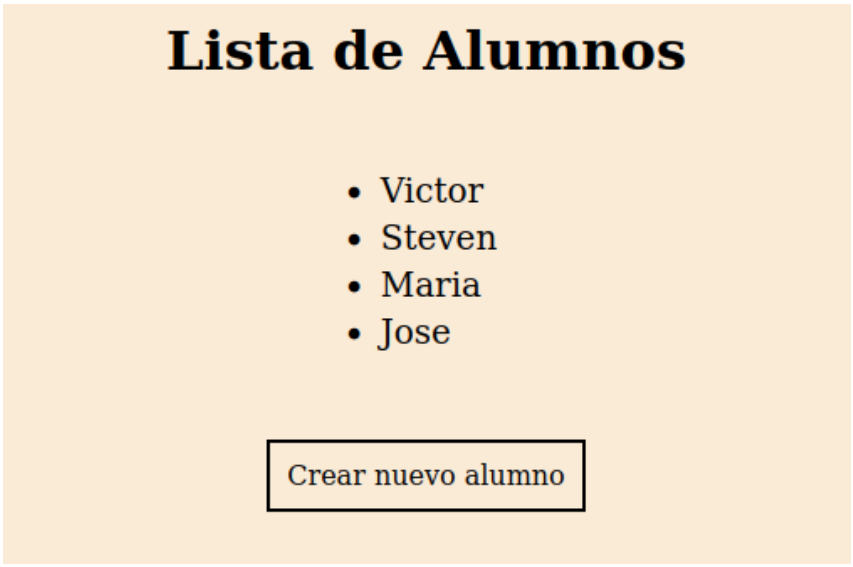


Figura 10: Pagina

crear_curso.html

```html
{% load static %}
<!DOCTYPE html>
<html>
<head>
    <title>Crear Cursos</title>
    <link rel="stylesheet" type="text/css" href="{% static 'css/style_formulario.css'%}"></link>
</head>
<body>
    <div class="content">
        <h1>Crear Curso</h1>
        <form method="post">
            {% csrf_token %}
            {{ form.as_p }}
            <button type="submit">Guardar</button>
        </form>
        <a href="{% url 'lista_cursos' %}">Ver todas los cursos</a>
    </div>
</body>
</html>
```

Figura 11: Codigo de crearCurso.html

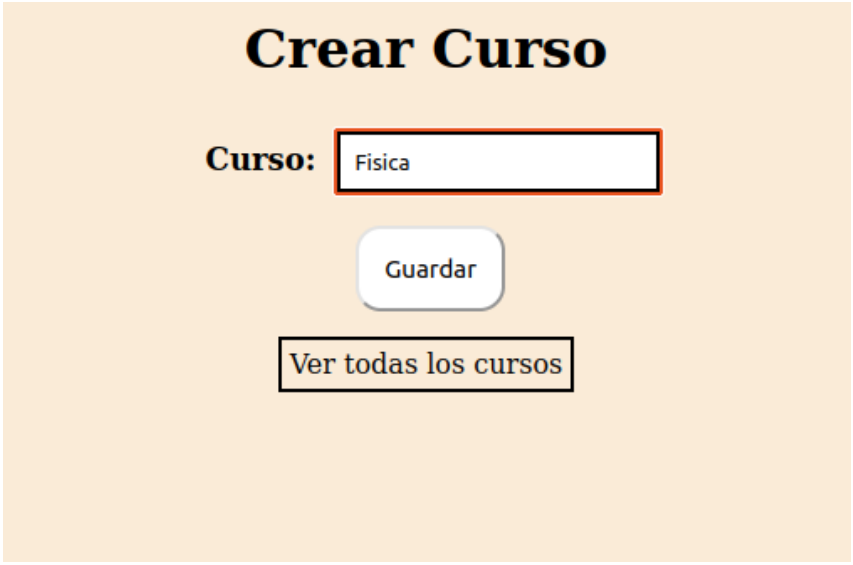


Figura 12: Pagina

lista_cursos.html

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
    <title>Lista de Cursos</title>
    <link rel="stylesheet" type="text/css" href="{% static 'css/style_lista.css'%}"></link>
</head>
<body>
    <div class="content">
        <h1>Lista de Cursos</h1>
        <ul>
            {% for curse in cursos %}
                <li>{{ curse.curso }}</li>
            {% endfor %}
        </ul>
        <a href="{% url 'crear_curso' %}">Crear nuevo curso</a>
    </div>
</body>
</html>
```

Figura 13: Codigo de listaCurso.html

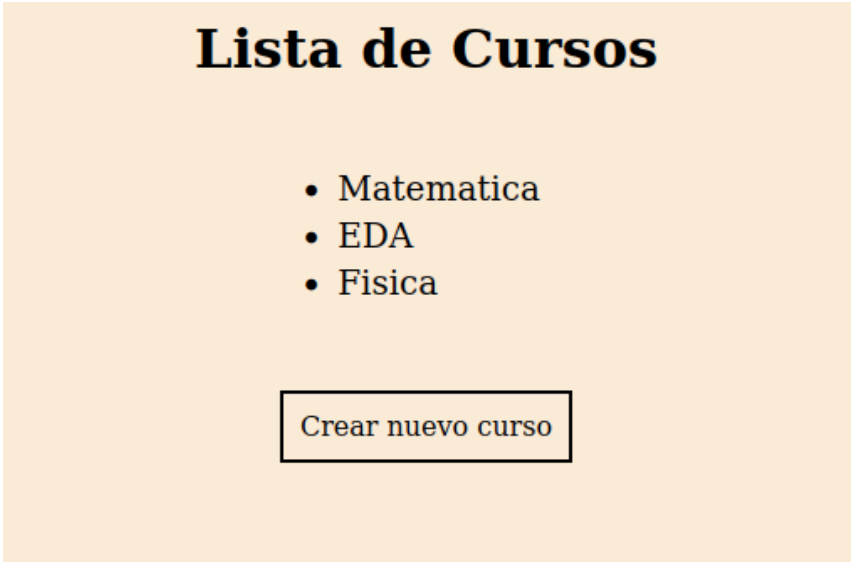


Figura 14: Pagina

crear_nota.html

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
    <title>Crear Notas</title>
    <link rel="stylesheet" type="text/css" href="{% static 'css/style_crearNota.css'%}"></link>
</head>
<body>
    <div class="content">
        <h1>Crear Nota</h1>
        <form method="post">
            {% csrf_token %}
            {{ form.as_p }}
            <button type="submit">Guardar</button>
        </form>
        <a href="{% url 'lista_notas' %}">Ver todas las notas</a>
    </div>
</body>
</html>
```

Figura 15: Codigo de crearNota.html

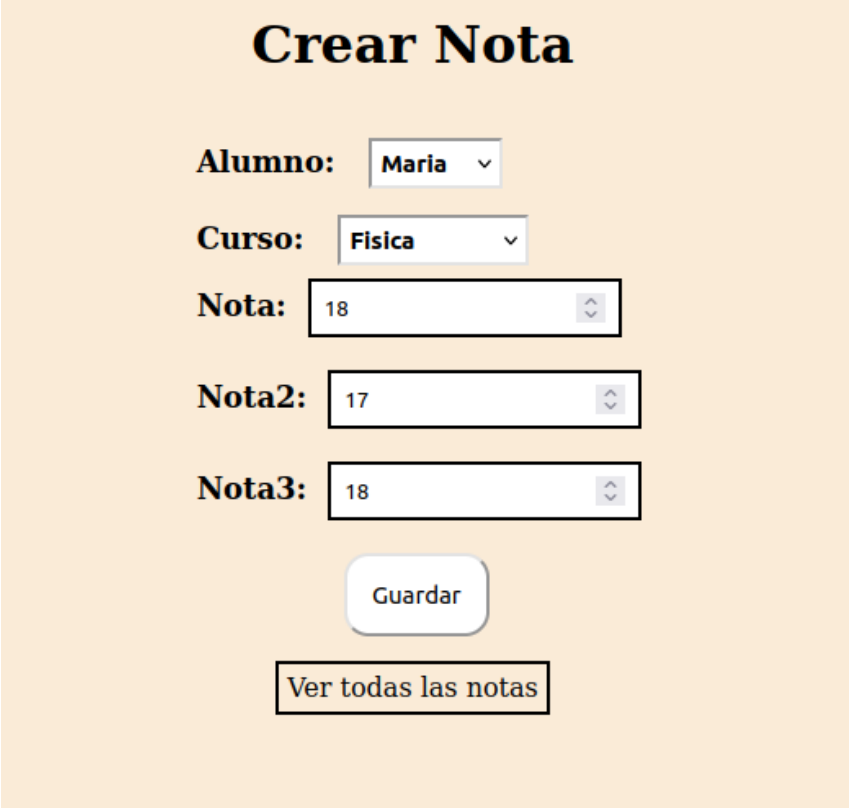


Figura 16: Pagina

lista_notas.html

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
    <title>Lista de Notas</title>
    <link rel="stylesheet" type="text/css" href="{% static 'css/style_table.css'%}"></link>
</head>
<body>
    <div class="content">
        <h1>Lista de Notas</h1>
        <table border="1">
            <thead>
                <tr>
                    <th> Alumno </th>
                    <th> Curso </th>
                    <th> Nota 1 </th>
                    <th> Nota 2 </th>
                    <th> Nota 3 </th>
                </tr>
            </thead>
            <tbody>
                {% for nota in notas %}
                <tr>
                    <td>{{ nota.alumno.nombre }}</td>
                    <td>{{ nota.curso.curso }}</td>
                    <td>{{ nota.nota }}</td>
                    <td>{{ nota.nota2 }}</td>
                    <td>{{ nota.nota3 }}</td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
        <a href="{% url 'crear_nota' %}">Crear nueva nota</a>
    </div>
</body>
</html>
```

Figura 17: Codigo de listaNota.html

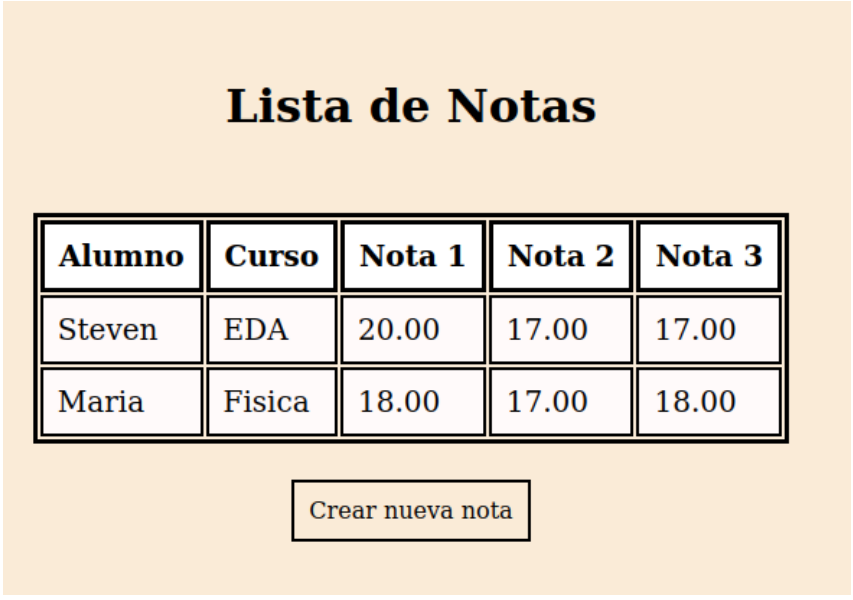


Figura 18: Pagina

Ejecutar el servidor

El comando python manage.py makemigrations colegio se utiliza en Django para crear nuevas migraciones basadas en los cambios realizados en los modelos del aplicativo colegio. Las migraciones son archivos que describen cómo modificar la estructura de la base de datos para mantenerla sincronizada con los modelos de Django. Este comando analiza los modelos en la aplicación colegio y genera los archivos de migración necesarios para aplicar esos cambios a la base de datos.

```
python manage.py makemigrations colegio
python manage.py migrate
```

El comando python manage.py runserver se utiliza en Django para iniciar el servidor de desarrollo local. Una vez ejecutado, el servidor se activa y permite acceder a la aplicación web en desarrollo a través de un navegador en la dirección http://localhost:8000/ por defecto. Es una herramienta fundamental durante el proceso de desarrollo de aplicaciones web con Django, ya que proporciona un entorno de prueba para probar y depurar el código antes de desplegar la aplicación en un entorno de producción.

```
python manage.py runserver
```

Acceder a la aplicacion en el navegador

Crear un alumno en http://127.0.0.1:8000/notas/crearAlumno/

Crear un curso en http://127.0.0.1:8000/notas/crearCurso/

Crear una nota en http://127.0.0.1:8000/notas/crearNota/

URL de video de explicación:

URL de repositorio de GitHub: `https://github.com/VictorMA18/Lab06-Django`