

Fase 1 proyecto Final

Josue Eduardo Nimatuj Piedrasanta, 201632173¹,*

¹Facultad de Ingeniería, Universidad de San Carlos, Ciudad Universitaria, Zona 12, Guatemala.

Creacion de pagina web de una academia de Cursos, con el nombre academia USAC, utilizando el framework de django. Se realiza la creacion de sus diferentes aplicaciones como pagina principal, catalogo de cursos, informacion sobre cursos, registro de usuarios, inicio de sesion y el despliegue de nuestra pagina web en la nube.

I. CODIGO

A. Descarga de programas y librerías

1. Git

Para la realizacion del sitio web Academia USAC se descargo git para el control de versiones de mi proyecto y para el uso de creacion de ambientes virtuales de desarrollo aislado o independiente del sistema operativo. Para descargarlo nos dirigimos al siguiente link: <https://git-scm.com/download/win>

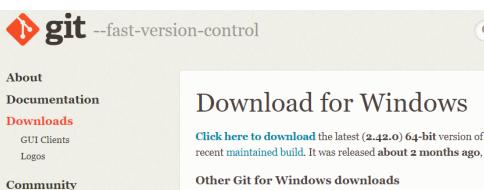


Figura 1: Enter Caption

2. Django

Un framwork con soluciones que una comunidad o alguna organización ha desarrollado en la que nos ofrecen formas de trabajo estandarizado para la creacion de paginas web. Para la instalacion unicamente ejecutamos el siguiente comando:



Figura 2: Enter Caption

B. Creacion de virtual environmet

Se realiza la instalcion de git. Seguido dentro de la carpeta de nuestro proyecto iniciamos el git bash, en que

despliega una interfaz de linea de comandos de git. Lo que queremos con el virtual environment es que todos los paquetes y librerías de mi aplicación web no afecten los paquetes globales de Python de mi sistema operativo. Creamos el virtual environment e ingresamos al contexto del virtual environment.

A terminal window with a black background and white text. It shows the user navigating to the directory '/c/Academia_Usac', creating a new virtual environment with 'python -m venv env', and activating it with 'source env/Scripts/activate'. The prompt changes to '(env)'.

Figura 3

C. Creacion de carpeta de proyecto

Se crea un proyecto de tipo django y junto con el nombre que el proyecto va a tener, adicional con un punto para indicar que se cree a nivel de la carpeta raiz.

A terminal window with a black background and white text. The command 'pip install django' is typed in and is being processed by the system.

Figura 4

A terminal window with a black background and white text. It shows the user creating a new django project named 'academia' with 'django-admin startproject academia .' and then running the development server with 'python manage.py runserver'.

Figura 5

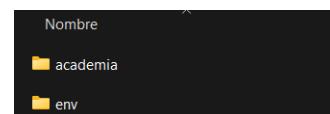


Figura 6

* josuenim9@gmail.com

D. Creacion de pagina principal

Para redirigir la pagina de inicio de mi proyecto web, se debe registrar dentro del archivo urls.py en la variable 'urlpatterns'. Ademas se crea un archivo views.py.

```
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name="home"),
]
```

Figura 7

Definimos la funcion "home" para que lea la data desde un "template.html" que se va a crear. Para que pueda procesar el request se crea una funcion de tipo render en nuestro archivo views.py.

```
academia > 📄 views.py > ...
1 from django.shortcuts import render
2
3 def home(request):
4     return render(request, 'home.html')
```

Figura 8

A nivel de la raiz del proyecto creamos un nuevo folder llamado templates y dentro un nuevo archivo llamado 'home.html' que sera mi pagina principal de mi proyecto. En el archivo settings.py buscamos el atributo TEMPLATES en el que cambiamos la propiedad de directorios llamada DIRS, agregamos el directorio 'templates'.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends',
        'DIRS': ['templates'],
        'APP_DIRS': True,
```

Figura 9

1. Creacion de pagina principal utilizando el framework bootstrap

Para ello debemos importamos un documento html con una barra de navegacion, un body y un footer. Seguido debemos importar los archivos de tipo imagen, estilos CSS y javascript de bootstrap que necesita el proyecto dentro del folder static en la carpeta proyecto llamada

'academia'.

Para que estos archivo media puedan ser leidos, necesitamos crear un estatic root dentro del archivo settings.py de nuestro proyecto.

```
STATIC_URL = 'static/'
STATIC_ROOT = BASE_DIR / 'static'
STATICFILES_DIRS = [
    'academia/static'
```

Figura 10

Guardamos los cambios. Algo adicional a hacer es ejecutar un comando dentro del git bash que me permita generar estos archivos estaticos a nivel de la raiz de la carpeta. De esta forma van a ser publicos. Y van a poder ser leidos por cualquier componente del proyecto.

```
Usuario@LAPTOP-SSG4V27S MINGW64 /c/Academia_Usac
$ python manage.py collectstatic
218 static files copied to 'C:\Academia_Usac\static'.
(env)
```

Figura 11

Dentro de nuestro archivo template/home.html en la parte superior del codigo se debe cargar las variables de tipo server, que en este caso representan la ruta estatica de los archivos CSS y javascript de bootstrap.

```
1  {% load static %}
```

Figura 12

Modificamos el archivo home.html de mi pagina principal e incluimos los archivos de media como icono, banner e imagenes. Y cambiamos el codigo dentro de mi documento home.html la forma en que va a leer los archivos media django. De la siguiente forma:

```

    <span class="mr-md-auto"><b>{{curso_count}} Cursos</b></span>
</div>
```

Figura 24

Realizamos la consulta a la base de datos en la aplicación cursos y luego lo presentamos en nuestra template para el catalogo. Utilizando código de servidor de Django creamos un bucle for { % for curso in cursos %} con esto indicamos que la variable el objeto curso, que representa una iteración dentro de la colección de cursos que viene de la base de datos, se toma y se imprime dentro del documento html.

```
{% for curso in cursos %}


<figure class="card card-product-grid">
        <div class="img-wrap">
            
        </div>
        <!-- img-wrap.-->
        <figcaption class="info-wrap">...
            </figcaption>
    </figure>
</div>
<!-- col.-->
{% endfor %}


```

Figura 25

1. Filtrar cursos por categoria

Aplicamos filtros dependiendo de la categoria, ya sea medicina, informatica,etc. Para ello se utiliza el 'category_slug' definido dentro de models.py de mi aplicación categorias. El parámetro del slug sera enviado dentro de la url, por ello se debe crear un nuevo path dentro de urls.py.

```
urlpatterns = [
    path('', views.Cursos, name = "Cursos"),
    #"/cursos/{category_slug}/"
    path('<slug:category_slug>',views.Cursos,name="cursos_by_category"),
```

Figura 26

Dentro de mi archivo views.py de la aplicación cursos importamos nuestro modelo de categorías. Seguido definimos una nueva variable llamada 'category_slug', se utiliza la función get_object_or_404 para recuperar un objeto de la base de datos de la clase Categorías, slug_category (izquierda) representa un campo de la tabla categoria que esta haciendo la comparación contra el parámetro category_slug que se define como variable y que viene desde la url.

```

from django.shortcuts import render, get_object_or_404
from .models import Curso
from categorias.models import Categorias

def Cursos(request, category_slug=None):
    categories = None
    cursos = None
    curso_count = 0 # Inicializamos curso_count aqui

    if category_slug is not None:
        categories = get_object_or_404(Categorias, category_slug=category_slug)
        cursos = Curso.objects.filter(category=categories, is_available=True)
        curso_count = cursos.count()

    else:
        cursos = Curso.objects.all().filter(is_available=True)
        curso_count = cursos.count()

    context = {
        'cursos': cursos,
        'curso_count': curso_count,
    }

    return render(request, 'cursos/CursosEstudiante.html', context)

```

Figura 27

En el caso de la variable cursos filtra los objetos segun 'Category=categories' Los cursos deben pertenecer a la categoría específica que se obtuvo en la linea de codigo anterior (categories). Si no se encuentra un objeto que coincida con el filtro (category_slug), se generará una excepción Http404, lo que significa que la página devolverá un error 404 si no se encuentra la categoría. El diccionario context se pasa a la plantilla cuando se llama a la función render en Django, y los valores de las claves se utilizan en la plantilla para mostrar dinámicamente la información. En el contexto de desarrollo web, la renderización involucra la combinación de datos dinámicos (como contenido de base de datos o variables) con una plantilla o estructura HTML para producir una página web completa y lista para ser visualizado por el usuario.

Para enviar la informacion a mi template cursos, primero creamos un nuevo archivo dentro de la aplicacion categorias llamado 'context_processors.py'. Importamos el modelo categorias desde el modulo actual. Definimos un funcion llamada 'menu_links' con la funcion de proporcionar enlaces a las categorias en la interfaz de la aplicacion.

```

1   from .models import Categorias
2
3   def menu_links(request):
4       links=Categorias.objects.all()
5       return dict(links=links)

```

Figura 28

Categorias.objects.all(): En esta línea, se realiza una consulta a la base de datos para recuperar todos los objetos de tipo Categorias, devuelve todas las categorías almacenadas en la base de datos.

Después de obtener el conjunto de objetos de categorías, se crea un diccionario llamado links, donde la clave es 'links' y el valor es el conjunto de objetos recuperados

en el paso anterior. Este diccionario se utiliza para proporcionar información sobre las categorías a la plantilla HTML.

Accedemos a los objetos de la base de datos 'Categorias', utilizando el diccionario 'links' de la función 'menu_links', realizamos la consulta de la informacion de la siguiente forma:

```

{%for category in links %}
<li>
    <a href="{{category.get_url}}>{{category.category_name }}</a>
</li>
{%endfor%}

```

Figura 29

En este código, { % for category in links %} itera sobre los objetos del conjunto de categorías almacenados en links, que se pasó desde la función menu_links. Para cada categoría, se puede acceder a sus atributos, como category_name, a través de la variable category. El punto en category.category_name se utiliza para acceder a un atributo o campo de un objeto en Python y, en este caso, se usa para acceder al atributo category_name del objeto category que se está iterando en un bucle

La expresión {{ category.get_url }} llama al método get_url definido en el modelo Categorias para generar la URL correspondiente a esa categoría.

Se define un nuevo metodo llamada 'get_url' dentro de la clase 'Categorias' de la aplicacion categorias. La función get_url del modelo Categorias se utiliza para obtener la URL de una categoría específica.

```

def get_url(self):
    return reverse('cursos_by_category',args=[self.category_slug])
#genera la url localhost:8000/cursos+slug ejecutando el evento
#para hacer la consulta en la base de datos devolviendo la lista de productos

```

Figura 30

'get_url' es un método personalizado que genera la URL de una categoría específica en función de su category_slug

En el metodo 'get_url', self se utiliza para acceder a los atributos y métodos de esa instancia (objeto que se crea a partir de una clase) específica. Esto es necesario porque un modelo de Django puede representar múltiples registros de la base de datos, y self permite diferenciar entre estos registros cuando se accede a sus datos y comportamientos.

'Reverse' es una función de Django que se utiliza para generar URLs inversas basadas en nombres de patrones de URL, en este caso 'cursos_by_category' es el nombre del patrón de URL, que se utilizará para acceder a la

vista que mostrará los cursos de una categoría específica, 'args=[self.category_slug]' proporciona los argumentos necesarios para generar la URL.

2. Pagina detalle del producto

Al hacer click sobre el nombre del producto nos debe llevar a la información del producto. Dentro de la aplicación cursos dentro de urls.py definimos un nuevo path. En la que se enviará por medio de la url el tipo de producto. El view debe redirigirnos a la página 'curso_detail'.

```
path('<slug:category_slug>/<slug:curso_slug>/',  
     views.curso_detail, name="curso_detail"),
```

Figura 31

Dentro de views.py de mi aplicación cursos creamos la función llamada 'curso_detail', colocamos un try para el control de errores, primero comparamos por el category_slug de mis categorías dentro de la función get(), se define el slug de la categoría a la que pertenece el producto se valide igualandolo con category_slug de la función, 'category__category_slug', se llama a 'category' de mi clase 'Cursos' y utilizando dos rayas para abajo para indicar u obtener el valor del campo de una estructura (category). Segundo comparamos por el slug del producto sea igual al product_slug de la función.

```
def curso_detail(request, category_slug, curso_slug):  
    try: # get() args: el slug de mi categoría a la que pertenece el producto se valide igualando  
        single_curso = Curso.objects.get(category__category_slug=category_slug,slug=curso_slug)  
    except Exception as e:  
        raise e  
    context={  
        'single_curso':single_curso,  
    }  
    return render(request, 'cursos/product_detail.html',context)
```

Figura 32

Agregamos un nuevo template llamado 'product_detail' dentro de la carpeta templates. Debemos incluir el '{% extends 'base.html' %}' para importar los links para los estilos de la página. Mostramos la información del curso que está en la base de la aplicación cursos. Ahora pasamos los datos dinámicos de mi función product_detail a mi plantilla html. Utilizando la llave 'single_curso' del diccionario context, llamamos la información de mi función 'curso_detail' dentro de views.py.

```
<article class="content-body">  
    <h2 class="title">{{single_curso.nombre}}</h2>  
  
    <div class="mb-3">  
        <var class="price h4">$ {{single_curso.costo}}.00</var>  
    </div>  
  
    <p>  
        {{single_curso.descripcion}}  
    </p>  
  
    {% if single_curso.cupo <= 0%}  
        <h4>No hay Cupos disponibles</h4>  
    {%else%}  
  
        <button type="submit" class="btn btn-primary">  
            <span class="text">Agregar al carrito</span>  
            <i class="fas fa-shopping-cart"></i>  
        </button>  
    {% endif %}
```

Figura 33

En mi carpeta models.py importamos la función reverse y creamos una función llamada get_url(self) que retorne una url inversa, utilizando la función reverse en patrones de url. La función reverse reemplaza '<slug:category_slug>' de mi path en urls.py por self.category.category_slug y '/<slug:curso_slug>/' por self.slug.

```
def get_url(self):  
    #le pasamos a curso_detail los siguientes argumentos  
    return reverse('curso_detail',args=[self.category.category_slug,self.slug])
```

Figura 34

Implementamos la función get_url() para saber la información de nuestro producto que nos redirigirá al template product_detail.html dentro de nuestro archivo CursosEstudiantes.html específicamente donde se necesita conocer los detalles del producto.

```
<div class="fix-height">  
    <a  
        href="{{curso.get_url}}"  
        class="title"  
        style="text-align: center"  
        >{{curso.nombre}}  
    </a>  
    <div class="price-wrap mt-2">  
        <span class="price">$ {{curso.costo}}</span>  
    </div>  
    <!-- price-wrap.-->  
</div>
```

Figura 35

A continuación se muestra la lógica para indicar que el cupo para los cursos es cero. Dentro de mi template 'product_detail.html' creamos una lógica condicional al botón 'agregar curso' para que solo se muestre si hay cupo utilizando el código de servidor de Django.

```

{% if single_curso.cupo <= 0%}
| <h4>No hay Cupos disponibles</h4>x
{%else%}

<button type="submit" class="btn btn-primary">
| <span class="text">Agregar al carrito</span>
| <i class="fas fa-shopping-cart"></i>
</button>
{% endif %}

```

Figura 36

3. Pagina de carrito de compras

Creacion de la estructuras y las tablas de mi carrito de compras

Creamos una nueva aplicacion llamadas 'carts', seguido la agregamos dentro de las aplicaciones del settings.py de proyecto 'academia_usac' y agregamos la url de mi aplicacion a la urlpatterns principal del proyecto.

Se crea un nuevo archivo llamado urls.py dentro la aplicacion carts de la siguiente manera:

```

carts > 📁 urls.py > ...
1  from django.urls import path
2  from . import views
3
4  urlpatterns = [
5      path('',views.cart,name='cart'),

```

Figura 37

A continuacion se crea un nuevo template dentro de los templates llamado 'asignacion_cursos.html', importamos el { % extends 'base.html' %},

```

{% extends "base2.html" %} {% load static %} {% block content %}
<section class="section-content padding-y bg">...
</section>
{% endblock content %}

```

Figura 38

Para llamar un icono o una imagen de mi archivo static, se realiza de la siguiente forma:

```

<p class="text-center mb-3">
| 
</p>

```

Figura 39

Creamos la base de datos dentro del models.py creamos la clase llamada cart que representa al carrito de compras y sera la tabla padre de CartItem. Ademas creamos un clase para elemento del carrito o el producto que se ha seleccionado o que se ve a comprar a futuro, esta clase la llamamos CartItem.

```

rts > 📁 models.py > 📁 CartItem
1  from django.db import models
2  from cursos.models import Curso
3  # Create your models here.
4
5  #Estructura del carrito de compra. Padre de Cart_item
6  class Cart(models.Model):
7      cart_id= models.CharField(max_length=250, blank=True)
8      date_added= models.DateField(auto_now_add=True)
9
10     def __str__(self):
11         return self.cart_id
12
13 #Cart_Id o producto selecciona que sera seleccionado y comprado
14 class CartItem(models.Model):
15     curso=models.ForeignKey(Curso,on_delete=models.CASCADE)
16     cart=models.ForeignKey(Cart,on_delete=models.CASCADE)
17     quantity=models.IntegerField()
18     is_active=models.BooleanField(default=True)
19

```

Figura 40

Dentro de la clase CartItem se crea un campo llamado curso que se relaciona con la clase Curso, y se indica que cuando elimine el producto relacionado tambien se elimine el cartItem o record, utilizando 'on_delete=models.CASCADE'.

Registraremos estas entidades dentro del archivo admin.py de mi aplicacion.

```

from django.contrib import admin
from .models import Cart,CartItem
# Register your models here.
class CartItemAdmin(admin.ModelAdmin):
    list_display=("curso","is_active")

admin.site.register(Cart)
admin.site.register(CartItem,CartItemAdmin)

```

Figura 41

Agregar elemento al carrito de compras

Dentro de la pagina 'product_detail' aparece el boton 'Agregar al carrito' el cual debe registrar un record en la base de datos que representa la creacion del carrito de compra y tambien la creacion del record que represente al item seleccionado.

Dentro del views.py creamos la funcion que represente la creacion del carrito de compras. Primero importamos la informacion de la clase Curso.

```

from django.shortcuts import render,redirect,get_object_or_404
from cursos.models import Curso
from .models import Cart,CartItem
from django.core.exceptions import ObjectDoesNotExist

def _get_card_id(request):
    cart= request.session.session_key
    if not cart:
        cart=request.session.create()
    return cart

```

Figura 42

La funcion _get_card_id solicita un objeto request porque necesita acceder a la sesion de Django para obtener o crear un identificador unico para el carrito

de compras asociado a la sesión del usuario. La función `_get_card_id` es una función privada ya que comienza con un guion bajo. La función `_get_card_id` tiene como objetivo proporcionar un identificador único para el carrito de compras asociado a la sesión de un usuario. `request.session.session_key` intenta obtener la clave de sesión actual asociada con la solicitud. Si el usuario ya tiene una sesión activa, esta línea obtendrá la clave de esa sesión.

Se crea una nueva sesión llamando a `request.session.create()`. Esto generará una nueva clave de sesión única para el usuario. Finalmente, la función devuelve la clave de sesión del carrito. Este valor se utiliza para identificar de manera única el carrito asociado con la sesión del usuario.

A continuación se implementa la funcionalidad del carrito de compras definiendo la función `add_cart`.

```
def add_cart(request, curso_id):
    #condicion que busque por el id
    curso = Curso.objects.get(id=curso_id)
    try:
        cart = Cart.objects.get(cart_id=_get_card_id(request))
    except Cart.DoesNotExist:
        cart = Cart.objects.create(cart_id=_get_card_id(request))
    cart.save()

    try:
        cart_item=CartItem.objects.get(curso=curso,cart=cart)
        cart_item.save()
    except CartItem.DoesNotExist:
        cart_item=CartItem.objects.create(
            curso = curso,
            quantity =1,
            cart=cart,
        )
        #Guardamos el record en la base de datos
        cart_item.save()
    return redirect('cart')
```

Figura 43

El `curso_id` se pasa como argumento para obtener el objeto `Curso` correspondiente de la base de datos.

* Para el primer Try. La función `_get_card_id` permite realizar la búsqueda para que el `cart_id` sea la sesión del usuario actual dentro del browser.

Se intenta obtener un objeto `Cart` relacionado con la sesión actual (`_get_card_id` proporciona el identificador de sesión). Si no existe, se crea uno nuevo y se guarda en la base de datos.

* Segundo Try. Se intenta obtener un objeto `CartItem` asociado al curso y al carrito. Si existe, se actualiza. Si no existe, se crea uno nuevo con una cantidad inicial de 1 y se guarda en la base de datos.

Finalmente, después de realizar las operaciones necesarias, el usuario se redirige a la vista del carrito.

Logica para consultar la data de la base de datos con respecto al carrito de compras y sus items

Primer try, Para saber si existe o no el elemento que se

desea buscar en la base de datos. Indicamos que busque el objeto `cart` dentro de la estructura (clase)

Primer try va a suceder en el caso que encuentre los valores para `cart` y `cart_items`.

Para `cart` instanciamos al objeto `Cart` para ello se utiliza la condición de búsqueda `cart_id=_get_card_id` y `cart_items` busca los `CartItems` que pertenecen al `cart`. El ciclo `for` para `cart_items` se usa para saber el precio total de mis productos y también para saber la cantidad total que están en el carrito de compras.

Para la excepción no realiza nada. Se realiza un diccionario `context` para indicar que valores son lo que se debe enviar al template que corresponde al carrito de compras.

```
def cart(request,total=0,quantity=0,cart_items=None):
    try:
        cart=Cart.objects.get(cart_id=_get_card_id(request))
        cart_items=CartItem.objects.filter(cart=cart,is_active=True)
        valor=1
        #Saber el precio total de mis cursos
        for cart_item in cart_items:
            # llamamos al campo 'curso' de mi clase CartItem
            total += cart_item.curso.costo
            #cantidad total de cursos
            quantity += cart_item.quantity
        # si no encontramos los valores
    except ObjectDoesNotExist:
        valor=0
        print("error")
        pass #ignora la excepcion
    #dic indicar que valores son los que se tiene que enviar al template cart
    context = {
        'total':total,
        'quantity':quantity,
        'cart_items':cart_items,
        'prueba': valor,
    }
    return render(request, 'cursos/asignacion_curso.html', context)
```

Figura 44

Se envía la información al template 'asignacion_cursos.html'.

```
{%for cart_item in cart_items %}
|  |
| --- |
| <figure class="itemside align-items-center"> <div class="aside">  </div> <figcaption class="info">  | {{cart\_item.curso.nombre}} |</a> |> |

```

Figura 45

Se solicita el costo y sub total del cursos.

```
<div class="price-wrap">
<var class="price">${{cart_item.sub_total}}</var>
<small class="text-muted">
| ${{cart_item.curso.costo}}
</small>
</div>
```

Figura 46

Indicamos que el nuestra sección `<tr>` o `table row`, se repite n-veces dependiendo de los elementos que se encuentren en la variable `cart_items`. Cambiando la im-

presion del los atributos como: la imagenes y el nombre. Enviamos el valor del total de los cursos a mi template de la siguiente forma:

```
<dl class="dlist-align">
  <dt>Total:</dt>
  <dd class="text-right text-dark b">
    <strong>${{total}}</strong>
  </dd>
</dl>
```

Figura 47

Implementar la funcionalidad para el boton eliminar

Primero instanciamos el objeto Cart en funcion del cart_id(request), ahora se repite para el objeto 'curso' en funcion del id de curso de la clase 'Curso'. Seguido buscamos dentro del CartItem con la condicion que busque al carrito de compras en funcion del curso y el cart. Por ultimo se elimina el cart utilizando cart_item.delete() y que redireccione a la pagina 'asignacion_curso.html'.

```
def remove_cart_item(request,curso_id):
    cart = Cart.objects.get(cart_id=_get_cart_id(request))
    curso = get_object_or_404(Curso,id=curso_id)
    cart_item=CartItem.objects.get(curso=curso,cart=cart)
    cart_item.delete()
    return redirect('cart')
```

Figura 48

Creamos un nuevo path para la funcion remove_cart.

```
path('remove_cart_item/<int:curso_id>',views.remove_cart_item,
      name='remove_cart_item'),
```

Figura 49

Dentro de el template 'asignacion_curso.html' cambiamos el href para el boton eliminar un producto, se llama el nombre del path para remover el producto y enviandolo la instancia el id del curso del cart_item actual.

```
<td class="text-right">
  <a href="{% url 'remove_cart_item' cart_item.curso.id %}" class="btn btn-danger">
    Eliminar</a>
  </td>
```

Figura 50

Revisar si existe un curso en el carrito

Definimos dentro de nuestro template 'asignacion_curso.html' la tabla de cursos aparesca seguna la

siguiente la condicion:

```
{% if not cart_items %}
  <h2 class="text-center">El carrito de cursos esta vacio</h2>
  <br>
  <div class="text-center">
    <a href="{% url 'Cursos' %}" class="btn btn-primary">Escoge un curso </a></div>
{% else %}
<div class="row">
  <aside class="col-lg-9">
    <div class="card">
      <table class="table table-borderless table-shopping-cart">
        <thead class="text-muted">
          <tr class="small text-uppercase">
```

Figura 51

Agregamos un boton que nos redireccione al catalogo de cursos.

Dentro de mi aplicacion 'cursos', dentro de views.py en mi funcion produc_detail, importamos el modelo del cartItem y la funcion get_cart_id desde la aplicacion carts, agregamos una nueva variable in_cart que realize la busqueda del producto dentro del carrito de compras, para poder obtener la propiedad del objeto cart_id del la clase Cart se instancia utilizando dos guiones bajos antes del cart de CartItem, se compara si pertenece al cart actual get_cart_id(). En resumen se realiza una consulta a la base de datos de CartItem para saber si existe un producto dentro del carrito de compras y que sea representado por medio de un flag verdadero o falso, por ello se utiliza la funcion .exists(). Y agregamos la varibel in_cart dentro del diccionario context.

```
def curso_detail(request,category_slug,curso_slug):
    try: # get() args: el slug de mi categoria a la pertenece el producto se valide igualandolo con
        single_curso = Curso.objects.get(category__category_slug=category_slug,slug=curso_slug)
        in_cart=CartItem.objects.filter(cart__cart_id = _get_cart_id(), curso = single_curso).exists()
    except Exception as e:
        raise e
    context={
        'single_curso':single_curso,
        'in_cart': in_cart
    }
    return render(request, 'cursos/product_detail.html',context)
```

Figura 52

Dentro del template 'curso_detail' seguido se va a la seccion de la evaluacion del stock.

```
{% if single_curso.cupo <= 0%}
  <h4>No hay Cupos disponibles</h4>
{%else%}
  {%if in_cart %}
    <a href="#" class="btn btn-success"><span class="text">Curso Agregado</span></a>
  <a href="{% url 'cart' %}" class="btn btn-outline-primary"><span class="text">Carrito</span></a>
  {%else%}
    <a href="{%url 'add_cart' single_curso.id%}" class="btn btn-primary"><span class="text">Añadir al Carrito</span></a>
  {%endif %}
  {%endif %}
```

Figura 53

Icono carrito de compras

Este procesador de contexto se utiliza para calcular el contador de la cantidad de productos en el carrito de compras. Se crea un archivo de tipo context_processors para crear un funcion global. Utiliza el metodo filter para

indicar que devuelva un conjunto vacío si no hay resultados. Obtiene todos los elementos del carrito asociados a ese carrito aunque solo se tenga un carrito establecemos que solo devuelva el primer elemento del arreglo `cart=cart[:1]`. Luego se itera sobre los elementos del carrito y suma las cantidades

```
from .models import Cart, CartItem
from .views import _get_card_id

def counter(request):
    cart_count=0

    try:
        cart=Cart.objects.filter(cart_id=_get_card_id(request))
        cart_items = CartItem.objects.all().filter(cart=cart[:1])
        for cart_item in cart_items:
            cart_count += cart_item.quantity
    except Cart.DoesNotExist:
        cart_count = 0
    # Variable global que es cart_count para ser usada en mi
    return dict(cart_count=cart_count)
```

Figura 54

`return 'cart_count': cart_count:` Devuelve un diccionario que contiene el contador del carrito (`cart_count`) para ser utilizado como parte del contexto global en las plantillas de Django. Agregamos `context_processors` dentro del `settings.py` de mi proyecto en la lista `TEMPLATES`.

Dentro del template `navbar.html` modificamos para que muestre el conteo de ítems en el carrito de compra utilizando la función global `cart_count`.

```
<a href="{% url 'cart' %}" class="widget-header pl-3 ml-3">
    <i class="fa fa-shopping-cart"></i>
    <span class="badge badge-pill badge-danger notify">{{cart_count}}</span>
</a>
```

Figura 55

Código de archivo `urls.py` para la aplicación `cursos`

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

urlpatterns = [
    path('', views.Cursos, name = "Cursos"),
    #"/cursos/{category_slug}/"
    path('<slug:category_slug>',views.Cursos,name="cursos_by_category"),
    #para acceder a cursos/cualquiercategoria/cualquierproducto/
    path('<slug:category_slug>/<slug:curso_slug>', views.curso_detail, name="curso_detail"),
]
```

Figura 56: Archivo `urls.py` App Cursos

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

urlpatterns = [
    path('',views.cart,name='cart'),
    path('add_cart/<int:curso_id>',views.add_cart, name='add_cart'),
    path('remove_cart_item/<int:curso_id>',views.remove_cart_item, name='remove_cart_item'),
]
```

Figura 57: Archivo `urls.py` App cart

G. Administración de usuarios

Se trabaja el módulo de seguridad y administración de usuarios en el proyecto. Utilizando la estructura de seguridad que trae por defecto el framework de django. Creamos una aplicación llamada `'accounts'` desde la cual vamos a gestionar tanto el registro como el login. Creamos una clase llamada `account` que se extienda desde el `AbstractBaseUser`, donde se crean los campos personalizados que se desean crear. También indicamos los atributos o propiedades por defecto de django. Seguidamente se cambia la configuración de usuario por defecto de inicio de sesión de django por el email

```
class Account(AbstractBaseUser):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    username = models.CharField(max_length=50,unique=True)
    email = models.CharField(max_length=100,unique=True)
    phone_number= models.CharField(max_length=50)

    # Campos Atributos de django

    date_joined = models.DateTimeField(auto_now_add=True)
    last_login=models.DateTimeField(auto_now_add=True)
    is_admin=models.BooleanField(default=False)
    is_staff = models.BooleanField(default=False)
    is_active=models.BooleanField(default=False)
    is_superuser=models.BooleanField(default=False)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS =['username','first_name','last_name']
```

Figura 58

Se crea un función para el registro del label por cada record, una función para indicar si tiene permisos de administrador solo si el atributo `admin` devuelve como True y una función para el administrador que tenga acceso a los módulos.

```
def __str__(self):
    return self.email
def has_perm(self, perm, obj=None):
    return self.is_admin
def has_module_perms(self, add_label):
    return True
```

Figura 59

Se crea una clase que tenga las operaciones que permita crear un nuevo usuario y un super admin usuario. Se incluyen los métodos o funciones dentro de la clase `Account`

```

class MyAccountManager(BaseUserManager):
    def create_user(self, first_name, last_name,email,username,password=None):
        if not email:
            raise ValueError('El usuario debe tener un email')
        if not username:
            raise ValueError('El usuario debe tener un username')
        user=self.model(
            email=self.normalize_email(email),
            username = username,
            first_name= first_name,
            last_name = last_name,
        )

        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_superuser(self, first_name, last_name, email, username,password):
        user=self.create_user(
            email = self.normalize_email(email),
            username =username,
            password = password,
            first_name=first_name,
            last_name=last_name,
        )
        user.is_admin=True
        user.is_active=True
        user.is_staff=True
        user.is_superuser=True
        user.save(using=self._db)
        return user

```

Figura 60

Se configura en el archivo setting.py del proyecto que reconozca como clase principal de almacenamiento de usuario la clase Account. Registraremos la clase account dentro del admin.py como parte del model de django Framework.//

```
AUTH_USER_MODEL = 'accounts.Account'
```

Se crea el superusuario de mi proyecto//

```
(env) PS C:\django\AcademiasUSAC> python manage.py createsuperuser
Email: josuenim9@gmail.com
Username: Josue
```

Figura 61: Creacion de super usuario

H. Creacion usuario

1. Archivo urls

Creamos las rutas URL y las vistas correspondientes para una aplicación Accounts.

```

from django.urls import path
from . import views

urlpatterns = [
    path('register/', views.register, name='register'),
    path('login/', views.login, name='login'),
    path('logout/',views.logout, name='logout'),
]

```

Figura 62: urls.py accounts

Figura 63

2. Uso de django forms

Importamos la clase Account del archivo models.py a nuestro archivo forms.py donde se creó la estructura de un nuevo usuario y superusuario dentro del proyecto. El modelo se va a basar en la clase Account. Definimos los datos que se necesita para registrar o crear un nuevo usuario, el cual será la estructura modelo que el utilizará mi formulario en el template HTML de registro.

```

class Meta:
    Model = Account
    fields = ['first_name','last_name','dpi',
              'fecha_de_nacimiento','phone_number','email','password']

```

Figura 64

Creamos las cajas de texto desde forms.py como puede ser el password, podemos darle estilos CSS desde el documento forms.py.

```

class RegistrationForm(forms.ModelForm):
    password = forms.CharField(widget=forms.PasswordInput(attrs={
        'placeholder': 'Ingrese Password',
        'class': 'form-control',
    }))

    confirm_password = forms.CharField(widget=forms.PasswordInput(attrs={
        'placeholder': 'Confirmar Password',
        'class': 'form-control',
    }))

```

Figura 65: Creacion de Password en forms.py

También podemos agregarle estilos a los demás campos de nuestro formulario usando un función de tipo def __init__:

```

def __init__(self, *args, **kwargs):
    super(RegistrationForm, self).__init__(*args, **kwargs)
    #Cada field representa cada caja de texto componente que se tiene registrado
    self.fields['first_name'].widget.attrs['placeholder']='Ingrese nombre'
    self.fields['last_name'].widget.attrs['placeholder']='Ingrese Apellido'
    self.fields['phone_number'].widget.attrs['placeholder']='Ingrese Telefono'
    self.fields['email'].widget.attrs['placeholder']='Ingrese email'
    for field in self.fields:
        self.fields[field].widget.attrs['class']='form-control'

```

Figura 66: Asignacion de estilos a nuestros campos de formulario

Validación de la contraseña:

```

def clean(self):
    cleaned_data=super(RegistrationForm,self).clean()
    password = cleaned_data.get('password')
    confirm_password=cleaned_data.get('confirm_password')

    if password != confirm_password:
        raise forms.ValidationError("El password no coincide!")

    if password:
        if len(password) < 8:
            raise forms.ValidationError("La contraseña debe tener al menos 8 caracteres.")
        if not re.search(r'\d', password):
            raise forms.ValidationError("La contraseña debe contener al menos 1 dígito.")
        if not re.search(r'[A-Z]', password):
            raise forms.ValidationError("La contraseña debe contener al menos 1 letra mayúscula.")
        if not re.search(r'[@#$%^&]', password):
            raise forms.ValidationError("La contraseña debe contener al menos 1 símbolo (@#$%^&).")

```

Figura 67: Enter Caption

3. Archivo views

En la funcion register creamos un objeto form basado en el RegistrationForm(), deseamos que se diriga hacia el template html

```
def register(request):
    form = RegistrationForm()
    context = {
        'form':form
    }
    return render(request, 'accounts/register.html',context)

def login(request):
    return render(request, 'accounts/login.html')
```

Figura 68

Dentro de views.py importamos nuestro models.py, ya que en la clase Account es el modelo sobre la cual insertaremos nuestros records dentro de la base de datos. Primero debemos capturar la data que envia el cliente el cual esta viajando dentro de un metodo POST, dentro del objeto 'request' para ello realizamos un condicional if. Se instancia un objeto de tipo form pasandole RegistrationForm, se valida si el formulario es valido, capturamos los datos que esta enviando el cliente y realizamos la instancia de un nuevo user al que le vamos a enviar los parametros para crear un nuevo usuario asociando los valores que me pide el metodo create-user de mi classe MyAccountManager del archivo models.py con los valores que obtenemos del cliente. Y por ultimo guardamos con le metodo safe()

```
def register(request):
    form = RegistrationForm()
    if request.method == 'POST':
        #Valor =0
        form = RegistrationForm(request.POST)
        if form.is_valid():
            first_name=form.cleaned_data['first_name']
            last_name = form.cleaned_data['last_name']
            username = form.cleaned_data['username']
            email = form.cleaned_data['email']
            phone_number = form.cleaned_data['phone_number']
            dpi = form.cleaned_data['dpi']
            fecha_de_nacimiento = form.cleaned_data['fecha_de_nacimiento']
            password = form.cleaned_data['password']
            user=Account.objects.create_user(first_name=first_name,last_name=last_name,username=username,email=email,password=password)
            user.phone_number = phone_number
            user.dpi=dpi
            user.fecha_de_nacimiento = fecha_de_nacimiento
            user.save()
        #Valor =1
    context = {
        'form':form,
    }
    return render(request, 'accounts/register.html',context)
```

Figura 69

Seguido creamos nuestro template para login y register enlazamos los links para login y register a nuestro navbar para que aparesca en todos los demas templates.

```
<form action="{% url 'register' %}" method="POST">
    {% csrf_token %}
    <div class="form-row">
        <div class="col form-group">
            <label>Nombre</label>
            {{form.first_name}}
        </div>
        <!-- form-group end.-->
        <div class="col form-group">
            <label>Apellido</label>
            {{form.last_name}}
        </div>
        <!-- form-group end.-->
    </div>
    <!-- form-row end.-->
    <div class="form-row">
        <div class="form-group col-md-6">
            <label>Telefono</label>
            {{form.phone_number}}
        </div>
```

H

Figura 70

4. Creacion de mi template para Registro

Se crea un nuevo documento html llamado register en la que importamos el codigo de navbar y footer. Importamos por medio de codigo de tipo server el contenido de los archivos views.py y forms.py.

Mensajes de alerta en django.

Dentro de django existe un paquete que me va a permitir manejar los disparos de estos eventos, lo configuramos dentro del settings.py de mi proyecto. Importamos los estilos que por defecto tiene el Bootstrat ingresando "messages.Error='danger'"

```
from django.contrib.messages import constants as messages
MESSAGE_TAGS={
    messages.ERROR: 'danger',#Incorpora estilo de alerta
}
```

Figura 71

Creamos un template para lanzar las alertas:

```
s > includes > dj_alerts.html
<!-- Solo si existen alertas-->
{%if messages%}
    <div id="message" class="container">
        {%for message in messages%}
            <div message.tags> para saber si de error o confirmacion de evento/<-->
                <div (%if message.tags%) class="alert alert-{%message.tags%}"%endif% role="alert">
                    {%if message.level==DEFAULT_MESSAGE_LEVELS.ERROR%} Importante: {%endif%}
                    {{message}}
                </div>
            </div>
        {%endfor %}
    {%endif%}
```

Figura 72

Agregamos el cambio en views.py://

```
messages.success(request,'Se registro el usuario exitosamente')
return redirect('register')
```

Figura 73

Incluimos el archivo alerts.html al template de registro:

```
{% include 'includes/alerts.html'%}

<header class="mb-4">
| <h4 class="card-title">Registro de Usuario</h4>
</header>

<form action="{% url 'register' %}" method="POST">
```

Figura 74

5.

I. Archivo urls.py del proyecto

urlpatterns es una variable utilizada en el archivo urls.py de un proyecto Django. Definen las rutas URL y las vistas correspondientes para una aplicación web Django. Cada elemento en la lista urlpatterns asocia una URL con una vista, permitiendo que Django enlace las peticiones del usuario a las vistas apropiadas.

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

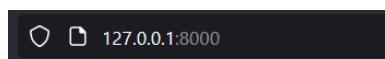
from django.conf.urls.static import static
from django.conf import settings

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('academiaApp.urls')),
    path('', include('asignaciones.urls')),
    path('curso/', include('cursos.urls')),
    path('cart/', include('carts.urls')),
    path('accounts/', include('accounts.urls')),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Figura 75

II. RESULTADOS

A. Verificacion de servidor activo



django

Figura 76

B. Base de datos Academia USAC

	id [PK]	nombre	costo	horario	cupo
1	12	Mercadotecnia	400	3:00-6:00 PM	4
2	13	Marketing	400	9:00-12:00 AM	4
3	14	Banca y finanzas	450	2:00-5:00 PM	4
4	15	Programador Fronted	450	3:00-6:00 PM	5
5	16	Dermatología	600	8:00-12:00 AM	2
6	17	Cirugía Pediátrica	500	8:00-12:00 AM	4
7	18	Medico Cirujano	500	9:00-12:00 AM	0

Figura 77: Base de datos en Postgresql

C. Pagina principal



Figura 78

D. Contenido de la pagina principal



Medicina

Ofrecemos una puerta de entrada al emocionante mundo de la ciencia y el cuidado de la salud. Nuestros programas de formación están diseñados para inspirar y capacitar a la próxima generación de profesionales de la medicina.



Disciplinas
Informática

En nuestra academia de informática, te ofrecemos las llaves para el éxito en la era digital. Aprende las habilidades tecnológicas más demandadas, desde programación y desarrollo web hasta ciberseguridad y análisis de datos.

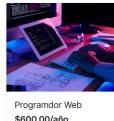


Negocios

Ofrecemos una experiencia educativa de élite diseñada para impulsar tus habilidades empresariales y abrir las puertas a un futuro de éxito. Con instructores expertos y programas personalizados.



Medico Cirujano
\$600.00/año



Programador Web
\$600.00/año



Administración de Empresas
\$400.00/año

[Ver Todo](#)

Figura 79

E. Catalogo de Cursos

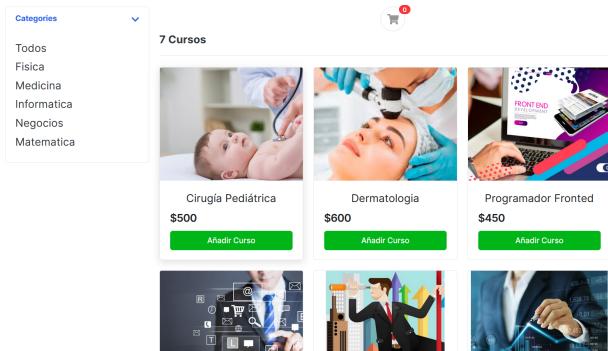


Figura 80

H. Registro de usuario y Validacion de contraseñas

Crear password Repetir password

Su contraseña debe contener:
Al menos 8 caracteres
Al menos 1 dígito
Al menos 1 símbolo
Al menos 1 mayúscula

Registrar

- El password no coincide!

Figura 83: Validar si la contraseña coinciden

F. Creacion de base de datos

Diagrama entidad relacion de mi base de datos:

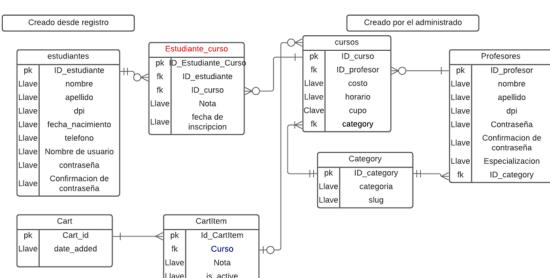


Figura 81

G. panel de administracion de django

Figura 82

Crear password Repetir password

Su contraseña debe contener:
Al menos 8 caracteres
Al menos 1 dígito
Al menos 1 símbolo
Al menos 1 mayúscula

Registrar

- La contraseña debe tener al menos 8 caracteres.

Figura 84: Validad contraseña 8 caracteres

I. Creacion de Superusuario e incriptacion de contraseña

Change account
josuenim9@gmail.com

Password: pbkdf2_sha256\$600000\$6VB691g0xJwyZ1kc

First name: Josue

Last name: Nimatuj

Username: jnim

Email: josuenim9@gmail.com

Phone number: 53878987

Is admin
 Is staff
 Is active
 Is superadmin

Figura 85: Panel de administrador

J. Pagina Register.html

Se registro el usuario exitosamente

Registro de Usuario

Nombre Apellido
Ingrese nombre Ingrese Apellido

Teléfono DPI
Ingrese Telefono

Email
Ingrese email

Fecha de nacimiento

Nombre de usuario

Crear password Repetir password
Ingres Password Confirmar Password

Su contraseña debe contener:
Al menos 8 caracteres
Al menos 1 dígito
Al menos 1 símbolo
Al menos 1 mayúscula

Registrar

Figura 86: Enter Caption

L. Pagina login.html**M. Formulario de registro y edición de profesor**

Add Profesor

Nombre: Hector

Apellido: Sevilla

Dpi: 313609
Enter a positive integer.

Especialidad: Gestión Recursos Humanos

Category: Negocios

SAVE Save and add another Save and continue editing

Figura 88

N. Formulario de Cursos

Cirugía Pediátrica

Nombre: Cirugía Pediátrica

Slug: cirugia-pediatrica

Costo: 500

Horario: 8:00 - 12:00 AM

Cupo: 4

Figura 89

K. Registro de usuario exitosamente**Ñ. Pagina con información de cada curso**

Se registro el usuario exitosamente

Registro de Usuario

Nombre Apellido
Ingrese nombre Ingrese Apellido

Figura 87



Figura 90



Figura 91: Curso Agregado

R. Ocultar Curso con cupo lleno

Figura 94

O. Informacion del curso si este ya esta agregado**P. Cerrar Sesión**

Has salido de sesión

Login

Email

Password

Figura 92

S. Inicio de sesión catedratico

Figura 95: Pagina para Catedraticos

T. Inicio de sesión Estudiante

Figura 96: Dashboard Estudiante

Figura 93: Log out

U. Cambio de diseño del administrador de Django

Figura 97: Uso de AdminSoftDashboard

V. Activacion de cuenta por correo

III. ANEXOS



Figura 98: Correo Activacion

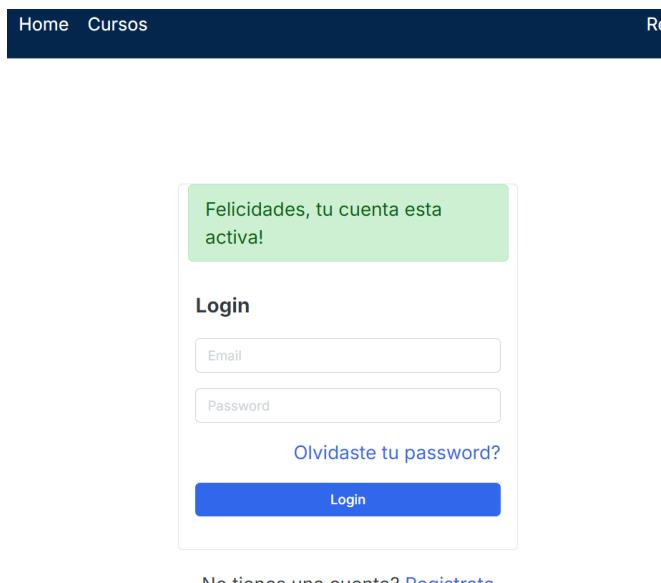


Figura 99: Mensaje de cuenta activada

<https://github.com/josuenim/academiaUsac.git>