

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 para una academia de Cursos, llamada academia USAC, utilizano el framwork 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 Cerrar sesion.

I. CODIGO

A. Descarga de programas y librerias

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 organizacion 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.

```
MINGW64:/c/Academia_Usac
Usuario@LAPTOP-SSG4V27S MINGW64 /c/Academia_Usac
$ python -m venv env
Usuario@LAPTOP-SSG4V27S MINGW64 /c/Academia_Usac
$ source env/Scripts/activate
(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.

```
Usuario@LAPTOP-SSG4V27S MINGW64 /c/Academia_Usac
$ pip install django
```

Figura 4

```
Usuario@LAPTOP-SSG4V27S MINGW64 /c/Academia_Usac
$ django-admin startproject academia .
(env)
Usuario@LAPTOP-SSG4V27S MINGW64 /c/Academia_Usac
$ python manage.py runserver
```

Figura 5

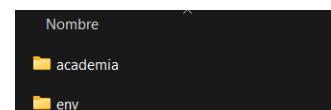


Figura 6

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

* josuenim9@gmail.com

'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 direcotorio '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 ell codigo dentro de mi documento home.html la forma en que va a leer los archivos media django. De la siguiente forma:

```

```

Figura 13

Para poder guardar los articulos del cart trabajamos con sesiones y para ello el usuario debe autenticarse.

E. Creacion Categorias y profesores

Para crear una aplicacion en django vamos en el shell de django escribimos el siguiente comando para crear un aplicacion:

```
python manage.py creatapp categorias
```

Figura 14: Enter Caption

La aplicacion categorias clasificara la especialidades que provee la academia usac. Realizamos la base de datos dentro del archivo models.py de la aplicacion categorias:

```
from django.db import models
from django.urls import reverse
# Create your models here.

class Categorias(models.Model):
    category_name=models.CharField(max_length=50)
    description= models.TextField(blank=True)
    category_slug =models.CharField(max_length=100,blank=True)
    cat_image=models.ImageField(upload_to= 'photos/categorias')
    class Meta:
        verbose_name='Categoria'
        verbose_name_plural = 'Categorias'
```

Figura 15

Indicamos de que forma queremos que se vea nuestro contenido en el panel de administracion de django:

```
from django.contrib import admin
from .models import Categorias

# Register your models here.
class CategoriasAdmin(admin.ModelAdmin):
    list_display=(‘category_name’, ‘category_slug’)
    prepopulated_fields={‘category_slug’: (‘category_name’,)}

admin.site.register(Categorias,CategoriasAdmin)
```

Figura 16

Creamos una aplicacion llamada 'profesores' en donde definiremos la informacion de contacto y especialida de cada profesor. Dentro del archivo models.py creamos nuestra base de datos.

```
from django.db import models
from categorias.models import Categorias

class Profesor(models.Model):
    nombre = models.CharField(max_length=50)
    apellido=models.CharField(max_length=50)
    dpi=models.IntegerField(help_text="Enter a positive integer.",unique=True)
    especialidad = models.CharField(max_length=50, blank=True)
    category = models.ForeignKey(Categorias,on_delete=models.SET_NULL,null=True)

    class Meta:
        verbose_name='Profesor'
        verbose_name_plural = 'Profesores'
    def __str__(self):
        return "{} {}".format(self.nombre, self.apellido, self.especialidad)
```

Figura 17

E indicamos en el archivo admin que campos queremos que se muestre en el panel de administracion:

```
from django.contrib import admin
from .models import Profesor

# Register your models here.
class ProfesoresAdmin(admin.ModelAdmin):
    list_display=(“nombre”, “apellido”, “especialidad”)
    list_filter=(“category”,)

admin.site.register(Profesor,ProfesoresAdmin)
```

Figura 18

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

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

Figura 21: Enter Caption

F. Creacion de Catalogo

Creamos una nueva aplicacion llamada 'cursos':

```
python manage.py creatapp cursos
```

Figura 19: Enter Caption

Creamps nuestra base de datos en el archivo models.py

```
from django.db import models
from profesores.models import Profesor
from categorias.models import Categorias
from django.urls import reverse
# Create your models here.
class Curso(models.Model):
    nombre = models.CharField(max_length=50)
    slug =models.CharField(max_length=100,blank=True)

    costo=models.PositiveIntegerField()
    horario=models.CharField(max_length=20)
    cupo=models.PositiveIntegerField()
    cat_image=models.ImageField(upload_to= 'photos/cursos')
    descripcion = models.TextField(blank=True)
    is_available = models.BooleanField(default=True)
    create_date=models.DateTimeField(auto_now_add=True)
    #Foreignkey#
    profesor = models.ForeignKey(Profesor, on_delete=models.SET_NULL, null=True)
    category = models.ForeignKey(Categorias, on_delete=models.SET_NULL,null=True)
```

Figura 20: Enter Caption

Codigo de archivo urls.py

G. Administracion de usuarios

Se trabaja el modulo de seguridad y administracion de usuarios en el proyecto. Utilizando la estructura de seguriada que trae por defecto el framework de django. Creamos una aplicacion 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. Tambien indicamos los atributos o propiedades por defecto de django. Seguido se cambia la configuracion de usuario por defecto de inicio de sesion 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 22

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

```

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 23

Se crea una clase que tenga las operaciones que permita crear un nuevo usuario y un super admin usuario. Se incluyen los metodos 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 24

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\AcademiaUSAC> python manage.py createsuperuser
Email: josuenim9@gmail.com
Username: Josue
```

Figura 25: 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 26: urls.py accounts

Figura 27

2. Uso de django forms

Importamos la clase Account del archivo models.py donde se creo 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 sera la estructura modelo que el utilizara 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 28

Creamos las cajas de texto desde forms.py como puede ser el password, podemos darle estilos CSS desdel 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 29: Creacion de Password en forms.py

Tambien podemos agregarle estilos a los demas campos de nuestro formulario usando un funcion 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 30: Asignacion de estilos a nuestros campos de formulario

Validacion 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("[a-zA-Z]", password):
            raise forms.ValidationError("La contraseña debe contener al menos 1 digito.")
        if not re.search("[a-zA-Z]", password):
            raise forms.ValidationError("La contraseña debe contener al menos 1 letra mayúscula.")
        if not re.search("[!@#$%^&]", password):
            raise forms.ValidationError("La contraseña debe contener al menos 1 simbolo (!@#$%^&).")
```

Figura 31: Enter Caption

3. Archivo views

En la funcion register creamos un objeto form basado en basado en el RegistrationForm(), deseemos 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 32

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=
            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 33

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

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.

```

<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>
  </div>

```

H

Figura 34

```

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

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

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

```

Figura 38

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.messages import constants as messages
MESSAGE_TAGS={
  |  messages.ERROR: 'danger',#Incorpora estilo de alerta
}

```

Figura 35

Creamos un template para lanzar las alertas:

```

s> includes > dj_alerts.html
<!--Solo si existen alertas-->
{%if messages%}
  &lt;div id="message" class="container"&gt;
    <!-- 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>
  {%endif%}

```

Figura 36

```

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 39

II. RESULTADOS

A. Verificación de servidor activo

```

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

```

Figura 37

Incluimos el archivo alerts.html al template de registro:

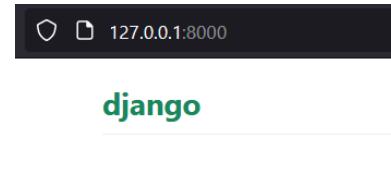


Figura 40

B. Pagina principal

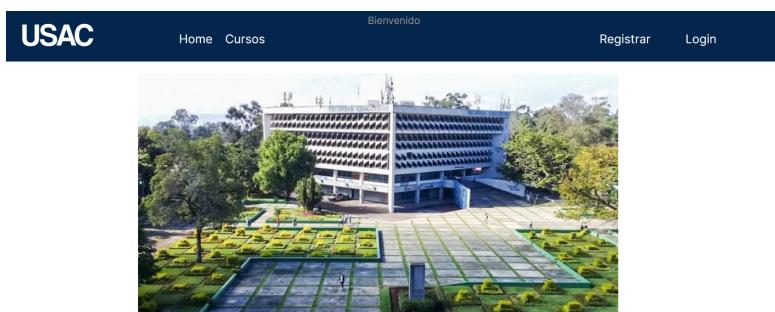


Figura 41

C. Contenido de la pagina principal

Disciplinas

- Medicina**: Ofrecemos un espacio de estudio 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.
- 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.

Cursos

Médico Cirujano \$600.00/año	Programador Web \$600.00/año	Administración de Empresas \$400.00/año
Ver Todo		

Figura 42

D. Catálogo de Cursos

Categorías

- Todos
- Física
- Medicina
- Informática
- Negocios
- Matemática

7 Cursos

Cirugía Pediátrica \$500	Dermatología \$600	Programador Fronted \$450
Añadir Curso	Añadir Curso	Añadir Curso

Figura 43

E. Creacion de base de datos

Diagrama entidad relacion de mi base de datos:



Figura 44

F. panel de administracion de django

Django administration

Home > Accounts > Accounts

Start typing to filter...

ACCOUNTS	Add
Accounts	+ Add

AUTHENTICATION AND AUTHORIZATION

GROUPS	Add
Groups	+ Add

CARTS

CART ITEMS	Add
Carts	+ Add

CATEGORIAS

CATEGORIAS	Add
Categorías	+ Add

CURSOS

CURSOS	Add
Cursos	+ Add

PROFESORES

PROFESORES	Add
Profesores	+ Add

Select account to change

Action: ----- Go 0 of 5 selected

EMAIL	FIRST NAME	USERNAME
jose_lopez@gmail.com	José	jose_lopez
juan_perez@gmail.com	Juan	juan_perez
lorenzo_lamas@gmail.com	Lorenzo	lorenzo_lamas
john_ortiz@gmail.com	John	john_ortiz
jose_ultimo@gmail.com	José	jim

5 accounts

Figura 45

G. Registro de usuario y Validacion de contraseñas

Crear password

Repetir password

• El password no coincide!

Figura 46: Validar si la contraseña coinciden

Crear password

Repetir password

• La contraseña debe tener al menos 8 caracteres.

Figura 47: Validad contraseña 8 caracteres

H. Creacion de Superusuario e incriptacion de contraseña

Change account
josuenim9@gmail.com
Password: pbkdf2_sha256\$60000\$6VB69f1g0xJwyZ1kcr
First name: Josue
Last name: Nimatuj
Username: jnim
Email: josuenim9@gmail.com
Phone number: 53878987
 Is admin
 Is staff
 Is active
 Is superadmin

Figura 48: Panel de administrador

J. Registro de usuario exitosamente

Se registro el usuario exitosamente

Registro de Usuario

Nombre Ingrese nombre
Apellido Ingrese Apellido

Figura 50

K. Pagina login.html

L. Formulario de registro y edicion de profesor

Add Profesor
Nombre: Hector
Apellido: Sevilla
Dpi: 313609 ⓘ
Enter a positive integer.
Especialidad: Gestion Recursos Humanos
Category: Negocios ⓘ

Figura 51

Se registro el usuario exitosamente

Registro de Usuario
Nombre Ingrese nombre
Apellido Ingrese Apellido
Telefono Ingrese Telefono
DPI Ingrese DPI
Email Ingrese email
Fecha de nacimiento
Nombre de usuario
Crear password Ingrese Password
Repetir password Confirmar Password
Su contraseña debe contener:
Al menos 8 caracteres
Al menos 1 digito
Al menos 1 simbolo
Al menos 1 mayuscula
Registrar

Figura 49: Enter Caption

Cirugía Pediátrica
Nombre: Cirugía Pediátrica
Slug: cirugia-pediatrica
Costo: 500 ⓘ
Horario: 8:00 - 12:00 AM
Cupo: 4 ⓘ

Figura 52

N. Pagina con informacion de cada curso



Figura 53

Ñ. Cerrar Sesión



Figura 54

O. Ocultar Curso con cupo lleno



Figura 55

III. ANEXOS

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