# CATEDRA: Programación III / Diseño e software III

# RECUPERATORIO Segundo Parcial Práctico

**CONSIGNA GENERAL:**

Ud. tendrá la opción de acceder al código del repositorio GIT (<https://github.com/unca-programacion3/proyecto_parcial2>), hacer un FORK del proyecto del parcial para añadirlo a su repositorio personal. Luego, puede trabajar sobre el proyecto y subir los cambios a su repositorio e informar el link. Al finalizar, debe OBLIGATORIAMENTE completar el código solicitado en este archivo.

Si Ud. no desea operar de la manera anterior, puede solamente completar el código solicitado en este archivo.

**CONSIGNAS DEL PARCIAL:**

**Tomando el caso de estudio de la aplicación web para gestión de correos electrónicos, se piden desarrollar las siguientes consignas, algunas de las cuales corresponden al parcial anterior, y otras son nuevas:**

1. Dentro de la app “mail” del proyecto de Django dado, generar los modelos y relaciones necesarias para registrar información de emails enviados por el usuario logueado y las distintas categorías a las que luego se puede asignar un mail.
   1. Generar un modelo para registrar los datos de una “categoria”, que consta solo de un “nombre”.
   2. Generar otro modelo para registrar información de un “mail” enviado por el usuario logueado en el sistema, con los siguientes datos:
   * **usuario\_origen**, correspondiente al Usuario que está logueado en el sistema
   * **mail\_origen**, que corresponde al mail del Usuario que está logueado en el sistema
   * **mail\_destinatario,** correspondiente al mail destino de otra persona.
   * **asunto**, que sería el título del mail.
   * **cuerpo**, que corresponde al texto dentro del cuerpo del mail.
   * **fecha\_envio**, correspondiente a la fecha en que se envía el mail (debe ser por defecto la fecha de creación del registro).
   * **categoria,** que corresponde a un objeto del modelo anterior, en principio debe ser vacía y los registros de mail no se deben eliminar si se elimina una categoría (atributo **on\_delete**)
2. class Categoria(models.Model):  
    categoria = models.CharField(max\_length=100, null=True)  
    pass  
     
     
   class Mail(models.Model):  
    usuario\_origen = models.ForeignKey(User, on\_delete=models.CASCADE)  
    mail\_origen = models.EmailField(max\_length=254)  
    destinatario = models.EmailField(max\_length=254)  
    asunto = models.CharField(max\_length=200)  
    cuerpo = models.TextField()  
    fecha\_envio = models.DateTimeField(auto\_now\_add=True)  
    categoria = models.ForeignKey(Categoria, on\_delete=models.PROTECT, default=7)#La categoría 7 es mi categoría “None”  
    pass

**NOTAS SI SE UTILIZA EL CODIGO DEL REPOSITORIO GIT:**

(1) Completar con el código correspondiente en el lugar especificado por la leyenda: “**#Completar aquí”** dentro del archivo **mail/models.py**

(2) No hacer ninguna modificación en la definición del modelo **User**

(3) Generar y aplicar las migraciones a la Base de datos con el comando correspondiente.

(4) Respetar los nombres de campos para los modelos solicitados

class Categoria(models.Model):  
 categoria = models.CharField(max\_length=100)  
 pass  
  
  
class Mail(models.Model):  
 usuario\_origen = models.ForeignKey(User, on\_delete=models.CASCADE)  
 mail\_origen = models.EmailField(max\_length=254)  
 destinatario = models.EmailField(max\_length=254)  
 asunto = models.CharField(max\_length=200)  
 cuerpo = models.TextField()  
 fecha\_envio = models.DateTimeField(auto\_now\_add=True)  
 categoria = models.ForeignKey(Categoria, on\_delete=models.SET\_NULL, null=True)  
 pass

1. Completar la definición del método **“clean”** de la clase “MailForm” dentro del archivo “forms.py”. Se debe controlar aquí, que el campo “mail\_destinatario” no sea igual al “mail\_origen”

class MailForm(forms.ModelForm):  
 class Meta:  
 model = Mail  
 fields = ('mail\_origen', 'destinatario', 'asunto', 'cuerpo')  
  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
  
 # Asigna la clase CSS a todos los widgets del formulario  
 for field\_name, field in self.fields.items():  
 field.widget.attrs['class'] = 'form-control'  
  
 self.fields['mail\_origen'].widget.attrs['readonly'] = 'true'  
  
 def clean(self):  
 super().clean()  
 mail\_origen = self.cleaned\_data["mail\_origen"]  
 mail\_destinatario = self.cleaned\_data["destinatario"]  
 if mail\_origen == mail\_destinatario:  
 raise forms.ValidationError("El mail de origen no puede ser igual que el del destinatario")  
 pass

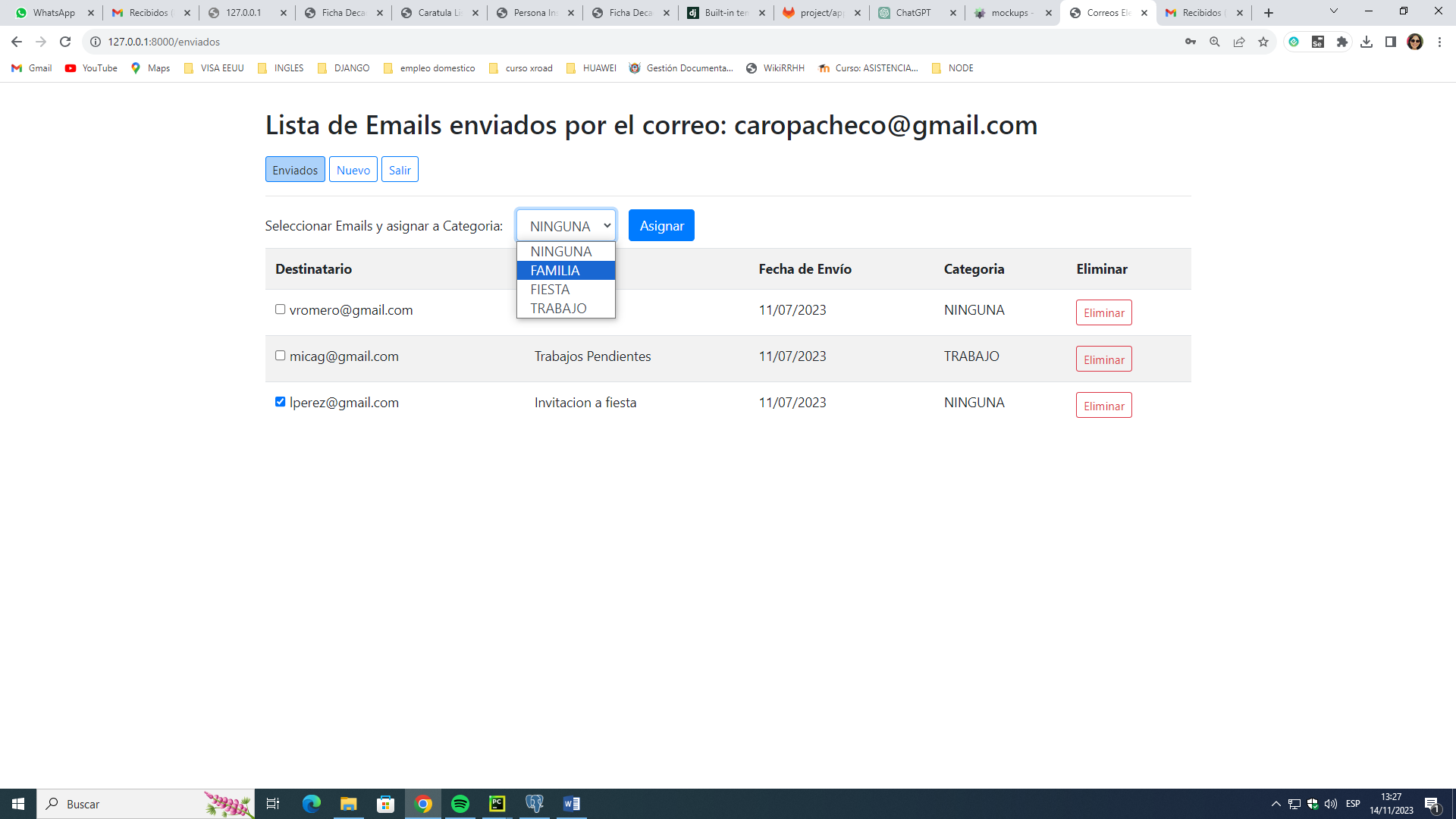
1. Completar la definición de la vista **“nuevo”** dentro del archivo “views.py”, según lo indicado en el archivo mediante los comentarios seguidos a la leyenda “# Completar aquí”.

@csrf\_exempt  
@login\_required  
def nuevo(request):  
 nuevo\_email = None  
 if request.method == 'POST':  
 email\_form = MailForm(request.POST)  
 if email\_form.is\_valid():  
 nuevo\_email = email\_form.save(commit=False)  
  
 nuevo\_email.usuario\_origen = request.user  
 nuevo\_email.mail\_origen = request.user.email  
 nuevo\_email.save()  
  
 return redirect('mail:enviados')  
 else:  
 email\_form = MailForm(initial={'mail\_origen': request.user.email})  
   
 return render(request, 'mail/nuevo.html', {'form': email\_form})

1. Completar la definición de la vista **“enviados”** dentro del archivo “views.py”, según lo indicado en el archivo mediante los comentarios seguidos a la leyenda “# Completar aquí”.

@login\_required  
def enviados(request):  
 usuario\_logueado = request.user  
 messages = Mail.objects.filter(usuario\_origen=usuario\_logueado).order\_by("-fecha\_envio")  
 categorias = Categoria.objects.all().order\_by("categoria")  
 return render(request, 'mail/enviados.html', {'messages': messages, 'categorias': categorias})  
 pass

1. Completar la definición del template **“mail/enviados.html”** según lo indicado en el archivo mediante los comentarios seguidos a la leyenda “Completar aquí”. El resultado debe ser como el de la Figura 1:



*Figura 1. Template que muestra lista de mails enviados por el usuario logueado*

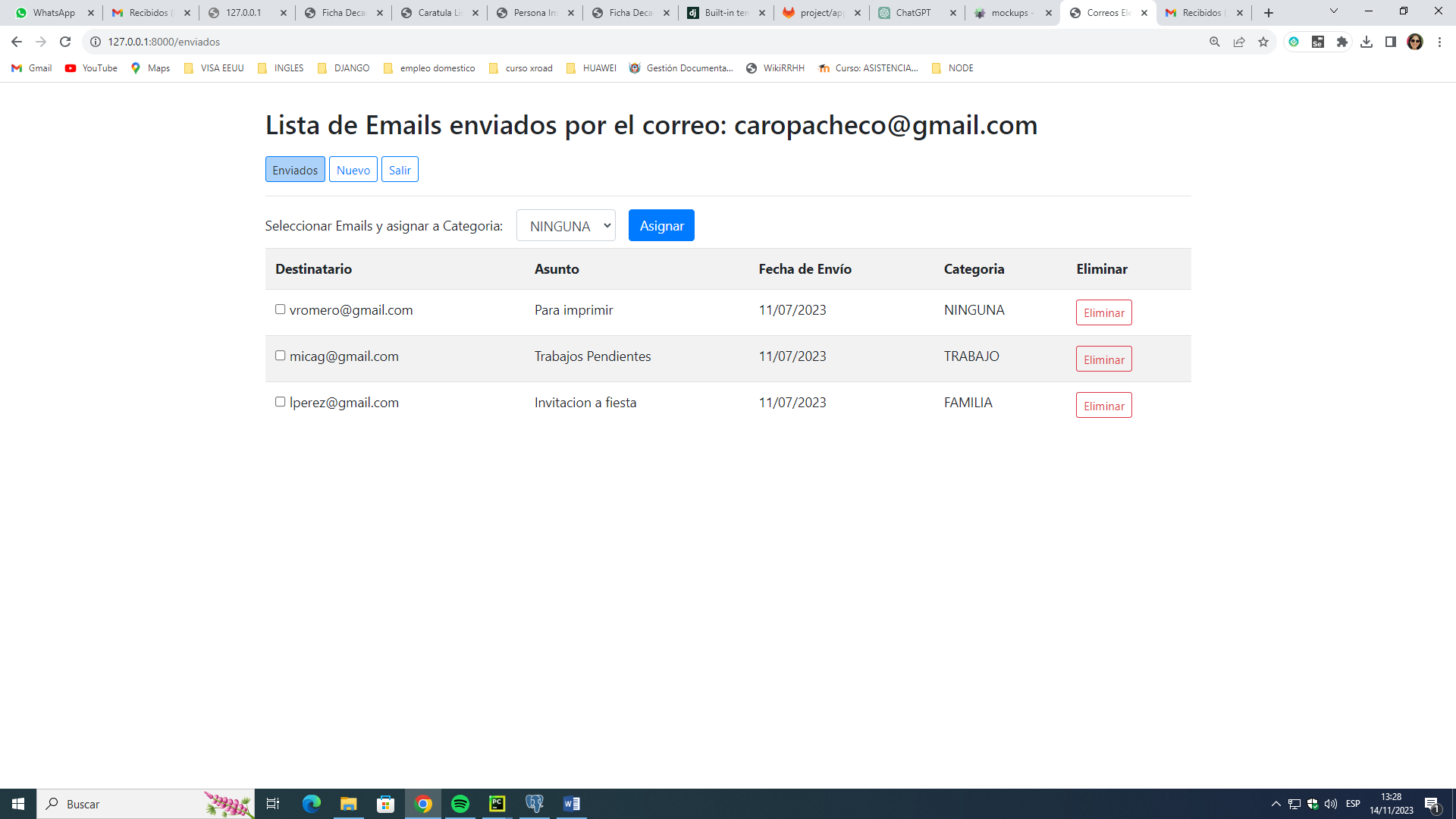
Parte 1 – Renderizacion de combo de “categorías”:

<<div class="form-group mx-sm-3 mb-2">  
 <select name="categoria" class="form-control">  
 <option value="0">NINGUNA</option>  
  
 {% for categoria in categorias %}  
 <option value="{{ categoria.id }}">{{ categoria.categoria }}</option>  
 {% endfor %}  
 </select>  
</div>

Parte 2 – Renderizacion de tabla de mails enviados:

<table class="table table-striped">  
 <tr>  
 <th>Destinatario</th>  
 <th>Asunto</th>  
 <th>Fecha de Envío</th>  
 <th>Categoria</th>  
 <th>Eliminar</th>  
 </tr>  
 {% for mail in messages %}  
 <tr>  
 <td><input type="checkbox" name="mail" value="{{ mail.id }}">{{mail.destinatario}}</td>  
 <td>{{mail.asunto}}</td>  
 <td>{{mail.fecha\_envio}}</td>  
 <td>{{mail.categoria.categoria|default\_if\_none:"Ninguna"}}</td>  
 <td><a href="{% url 'mail:eliminar\_mail' mail.id %}" class="btn btn-danger">Eliminar</a></td>  
 </tr>  
 {% endfor %}  
</table>

1. Completar la definición de la vista **“asignar\_categoria”** dentro del archivo “views.py”, según lo indicado en el archivo mediante los comentarios seguidos a la leyenda “# Completar aquí”. Una vez que se ejecute la vista mencionada, se debe volver a ver el template de la Figura 1, con el valor de categoría actualizado para los mails seleccionados, tal como la Figura 2:



*Figura 2. Lista de mails enviados donde se refresca el valor del campo “Categoría”*

@login\_required  
def asignar\_categoria(request):  
 if request.method == 'POST':  
 ids\_emails\_seleccionados = request.POST.getlist('mail')  
 categoria\_selecionada = int(request.POST.get('categoria'))  
 for id\_email in ids\_emails\_seleccionados:  
 email = Mail.objects.get(id=id\_email)  
 if categoria\_selecionada == 0:  
 email.categoria = None  
 else:  
 email.categoria = Categoria.objects.get(id=categoria\_selecionada)  
 email.save()  
 return redirect('mail:enviados')  
 else:  
 return redirect('mail:enviados')

1. Desarrollar la vista “eliminar\_mail” que permita eliminar un email según el ID enviado en la URL. Tener en cuenta los comentarios en el código fuente.
2. @login\_required  
   def eliminar\_mail(request, pk):  
    mail = get\_object\_or\_404(Mail, pk=pk)  
    if request.user == mail.usuario\_origen:  
    mail.delete()  
    return redirect(reverse("mail:enviados"))  
    else:  
    messages.error = "El mail que se quiere eliminar no pertenece al usuario logueado"  
    return redirect(reverse("mail:enviados"))
3. Completar la definición de las urls dentro del archivo “mail/urls.py”, agregando los patrones de url para acceder a las vistas de "enviados", "nuevo", “eliminar\_mail” y "asignar\_categoria".

app\_name = 'mail'  
  
urlpatterns = [  
 path("", views.login\_view, name="login"),  
 path("enviados", views.enviados, name="enviados"),  
 path("asignar\_categoria",views.asignar\_categoria, name="asignar\_categoria"),  
 path("eliminar\_mail/<int:pk>", views.eliminar\_mail, name="eliminar\_mail"),  
 path("nuevo", views.nuevo, name="nuevo"),  
 path("login", views.login\_view, name="login"),  
 path("logout", views.logout\_view, name="logout"),  
 path("registro", views.register, name="registro"),  
]

**NOTAS FINALES SI SE UTILIZA CODIGO DE GIT:**

* No hacer ninguna modificación a las vistas y templates correspondientes al **login, logout** y **registro** de usuarios. Esto servirá para registrar usuarios y probar la funcionalidad del sistema.
* Generar un usuario de tipo “superadmin” para poder cargar registros de “Categorías” desde el Admin de Django.
* Documentar el código agregando comentarios para aquellas sentencias más significativas.
* Subir al repositorio personal de GIT los cambios realizados al proyecto.