



Tecnológico de Monterrey

STC0204

Desarrollo de componentes de software

César Ignacio Saucedo Rodríguez A01712245

04 / 06 / 2025

Construcción de software y toma de decisiones (Gpo 501)

Cuaderno de apuntes:

Resumen Integrador del Curso: Construcción de Software y Toma de Decisiones

Este curso abordó los fundamentos del desarrollo de software en equipo con un enfoque ágil y orientado a la toma de decisiones basada en datos. Se integraron prácticas de ingeniería de software, diseño de sistemas, documentación, pruebas, despliegue y trabajo colaborativo.

Bosquejo de Conceptos Clave

- **Metodologías ágiles**
 - SCRUM (sprints, backlog, daily meetings, retrospectivas)
 - Roles: Product Owner, Scrum Master, Development Team
- **Requerimientos de Software**
 - Funcionales y no funcionales
 - Técnicas de recolección (entrevistas, observación, historias de usuario)
- **Modelado y diseño**
 - Diagramas UML: Casos de uso, clases, actividad
 - Prioridades: Complejidad, valor, riesgo y estabilidad
 - Mapas mentales para visualización temprana de módulos

- **Desarrollo colaborativo**

- Uso de Git y GitHub (branching, merge, pull requests)
- Manejo de conflictos y flujos de trabajo centralizados o por feature

- **Arquitectura del sistema**

- Arquitectura cliente-servidor
- Separación de responsabilidades por app

- **Pruebas y calidad**

- Pruebas unitarias y funcionales
- Validación de casos de uso
- Buenas prácticas de documentación técnica

- **Despliegue**

- Railway para hosting en la nube
- Variables de entorno y seguridad
- Creación de superusuarios, migraciones y población de datos

Convención de código del equipo

A continuación se enlistan las principales reglas que seguimos como equipo para mantener un código limpio, coherente y fácil de mantener:

Estructura general

- **Framework:** Django 5 con estructura modular por apps
- **Base de datos:** MariaDB
- **Frontend:** TailwindCSS + HTMX para interactividad

Nombres y formato

- Archivos Python: snake_case.py
- Clases: PascalCase
- Métodos y variables: snake_case
- Templates: nombre_funcionalidad.html (por ejemplo: detalle_campana.html)
- Estáticos: ubicados en static/, organizados por tipo (js/, css/, img/)

Organización del repositorio

- Cada funcionalidad se encapsula en una app: OBD_users, OBD_project, OBD_items, OBD_campaigns
- Separación clara de templates:

- templates/OBD/ para vistas generales
- templates/account/ para autenticación
- partials/ y includes/ para componentes reutilizables

Buenas prácticas

- Cada vista nueva se acompaña de su prueba unitaria (cuando aplique)
- Se evita lógica compleja en templates (preferencia por vistas y contextos)
- Se usan modelos personalizados con Abstract User para flexibilidad

Los commits siguen la convención:

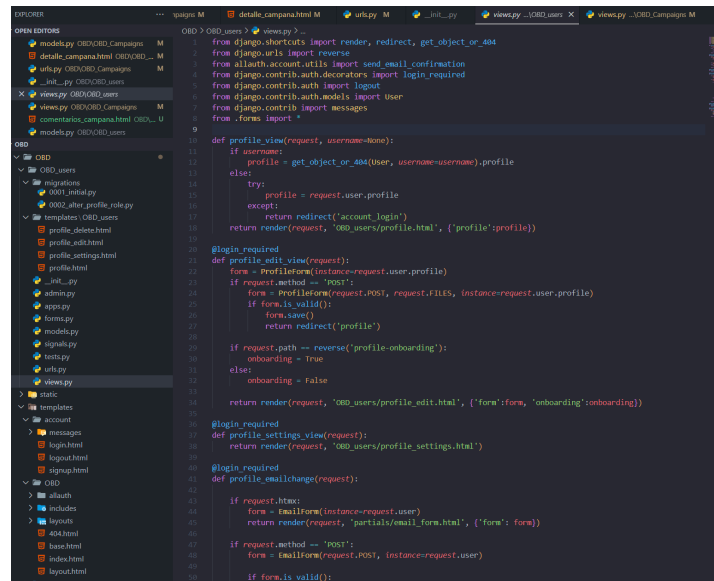
- Corrige bug
- Mejora código sin cambiar funcionalidad
- Cambios en la documentación
- Añade o mejora pruebas
- Cambios de formato/estilo

Contribuciones individuales de frontend y backend en el repositorio del proyecto

Al liderar el equipo de software, dividí en partes diferentes las responsabilidades del backend y frontend. En este punto en semana 9, llevamos aproximadamente el 70% de desarrollo en nuestro proyecto, esto es porque ya tenemos implementados los modelos en Django y las apps necesarias dentro del proyecto para el correcto funcionamiento.

El proyecto en backend y frontend se dividió de la siguiente manera:

El comienzo del proyecto backend con el sistema de autenticación de usuarios que se hizo con allauth, Django y HTMX fue implementado por mí, por otra parte aksel, fue el que creó las vistas para inicio de sesión y menú principal. Alex por otro lado fue el que implementó el modelo de usuario.



```

1 from django.shortcuts import render, redirect, get_object_or_404
2 from django.urls import reverse
3 from allauth.account.utils import send_email_confirmation
4 from django.contrib.auth.decorators import login_required
5 from django.contrib.auth import login, logout
6 from django.contrib.auth.models import User
7 from django.contrib import messages
8 from forms import *
9
10 def profile_view(request, username=None):
11     if username:
12         profile = get_object_or_404(User, username=username).profile
13     else:
14         try:
15             profile = request.user.profile
16         except:
17             return redirect('account_login')
18     return render(request, 'OBD_users/profile.html', {'profile': profile})
19
20 @login_required
21 def profile_edit_view(request):
22     form = ProfileForm(instance=request.user.profile)
23     if request.method == 'POST':
24         form = ProfileForm(request.POST, request.FILES, instance=request.user.profile)
25         if form.is_valid():
26             form.save()
27             return redirect('profile')
28
29     if request.path == reverse('profile-onboarding'):
30         onboarding = True
31     else:
32         onboarding = False
33     return render(request, 'OBD_users/profile_edit.html', {'form': form, 'onboarding': onboarding})
34
35 @login_required
36 def profile_settings_view(request):
37     return render(request, 'OBD_users/profile_settings.html')
38
39 @login_required
40 def profile_emailchange(request):
41     if request.method == 'POST':
42         form = EmailForm(instance=request.user)
43         return render(request, 'partials/email_form.html', {'form': form})
44     if request.method == 'POST':
45         form = EmailForm(request.POST, instance=request.user)
46         if form.is_valid():
47

```

Figura 1. [Views.py](#) de OBD_users (backend de la gestión de usuarios)

La parte de OBD_project que corresponde a vistas básicas como home, contacto y about, fue el backend implementado por Aksel, pero la parte de frontend fue hecha por mi basada en los bosquejos elegidos por el equipo y el diseño que ya había implementado Aksel.

```

OBD > OBD_project > templates > OBD_project > about.html > div.p-10.text-center
1 {% extends 'OBD/layouts/blank.html' %}
2
3 {% block layout %}
4
5 <div class="p-10 text-center">
6     <h1 class="text-4xl font-bold mb-4">Acerca de Nosotros</h1>
7     <p class="text-lg text-gray-600 max-w-2xl mx-auto">
8         En Leadly, creemos que la organización y eficiencia son clave para el éxito. Nuestro objetivo es ayudarte a
9     </p>
10    <div class="mt-6 text-gray-500 text-sm">
11        <p>Fundado en 2024, Leadly ha ayudado a más de 100 equipos a alcanzar sus metas.</p>
12        <p>Nuestra misión es simplificar la gestión de proyectos, brindando herramientas funcionales y accesibles p
13    </div>
14 </div>
15
16 {% endblock %}

```

Figura 2. Frontend de Django OBD_project

La parte de OBD_campaigns es donde las responsabilidades empezaron a tomar más complejidad, como Aksel y Alex fueron los encargados de crear los modelos y que encajaran con el backend que yo implementé, de similar manera entre ellos dos gestionaron la parte del frontend de esa parte del sistema, basado en los bosquejos que ellos mismos hicieron.

```
OBD > OBD_Campaigns > views.py > vista_campanas
1 from django.contrib.auth.decorators import login_required, user_passes_test
2 from django.views.decorators.http import require_http_methods
3 from django.shortcuts import render, redirect
4 from .models import Campana, Item, Archivolive, Comentario
5 from django.shortcuts import get_object_or_404, render
6 from .forms import CampanaForm, ItemForm, ArchivoliveForm, CSVUploadForm
7 from django.http import HttpResponseRedirect
8 from django.views.decorators.csrf import csrf_exempt
9 from django.views.decorators.cache import never_cache
10 from django.urls import reverse
11 from django.contrib import messages
12 import csv
13
14 # Dashboard de campañas
15 @login_required
16 def vista_campanas(request):
17     campanas = Campana.objects.order_by('-fecha_creacion_campana')
18     es_admin = request.user.is_superuser
19     return render(request, 'OBD_Campaigns/campana.html', {
20         'campanas': campanas,
21         'es_admin': es_admin
22     })
23
24 # Vista a una campaña específica
25 @login_required
26 def detalle_campana(request, pk):
27     campana = get_object_or_404(Campana, pk=pk)
28     return render(request, 'OBD_Campaigns/campana.html', {
29         'campana': campana
30     })
31
32 # Vista para confirmar la eliminación de una campaña
33 @login_required
34 def borrar_confirmacion_campana(request, pk):
35     campana = get_object_or_404(Campana, pk=pk)
36     return render(request, 'OBD_Campaigns/partial_confirm_delete.html', {
37         'campana': campana
38     })
39
40 # Vista para eliminar una campaña
41 @csrf_exempt
42 @require_http_methods(["DELETE"])
43 def borrar_campana(request, pk):
44     campana = get_object_or_404(Campana, pk=pk)
45     campana.delete()
46     return HttpResponseRedirect("") # Devuelve string vacío en lugar de 204
47
48 # formulario para crear una nueva campaña
49 @never_cache
50 @require_http_methods(["GET", "POST"])
51 @login_required
52 @user_passes_test(lambda u: u.is_superuser)
```

Figura 3. [Views.py](#) de OBD_campaigns (backend de la gestión de campañas)