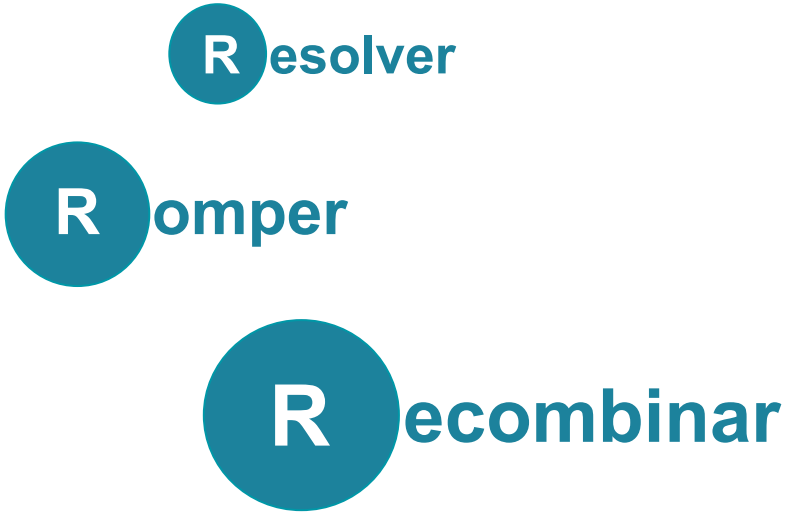


Las tres Rs para programar apps Django reutilizables

Fran Muñoz & Sonia Alhama
PyConES Granada 2022
APSL

Sonia Alhama

- Graduada en Física
- Mentora DjangoGirls
- Desarrollo back y data
- Gestión equipo

salhama@apsl.net

@sonia_alhama

https://www.linkedin.com/in/soniaalhama/

Fran Muñoz

- Ingeniero informático
- Teletrabajando en Ibiza desde 2015
- Referente desarrollo back y front
- Creador programa Akademy

fmunoz@apsl.net

@Franmc1

https://es.linkedin.com/in/francisco-munoz-cuevas

apsl

Agenda

1. Presentación SPONSOR TIME
2. Software reutilizable
3. Diseño de software
4. Herramientas de Django

apsl

SPONSOR TIME

APSL/django-yubin

Send e-mails asyncronously using cron

python django mail mailer

☆ 43 Python Updated 27 days ago

Asiduos asistentes y sponsors de PyConES y DjangoGirls.

Organizadores de DjangoGirls Mallorca.

Creantbits

Ligados a la comunidad Python desde los inicios en 2009.

Colaboramos con nuestro GitHub público.

Organizadores de charlas de Python a nivel local como los Creantbits.

apsl

SPONSOR TIME

Ofrecer al cliente una solución lo más completa posible y a medida

1. **Consultoría**

   Diagnóstico y análisis de requerimientos

2. **Desarrollo web a medida**

   Python + Django
   React - Vue
   Odoo

3. **Ingeniería de sistemas**

   Despliegues en cloud (AWS - Google)
   Alta disponibilidad
   Backups y monitorización

4. **Data e inteligencia artificial**

   Data analysis y pruebas de concepto
   Data mining y diseño de algoritmos
   Machine learning

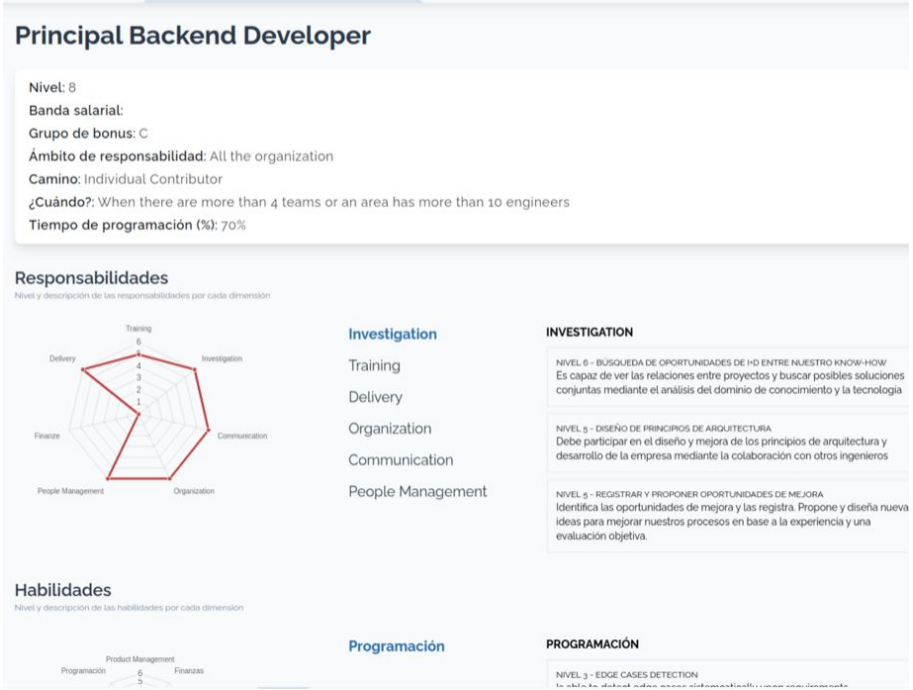
apsl

SPONSOR TIME

apsl

SPONSOR TIME

YedAI & IntraForce



Principal Backend Developer

Nivel: 8
Banda salarial:
Grupo de bonus: C
Ámbito de responsabilidad: All the organization
Camino: Individual Contributor
¿Cuándo?: When there are more than 4 teams or an area has more than 10 engineers
Tiempo de programación (%): 70%

Responsabilidades

Nivel y descripción de las responsabilidades por cada dimensión

Investigation
Training
Delivery
Organization
Communication
People Management

INVESTIGATION

NIVEL 6 - BÚSQUEDA DE OPORTUNIDADES DE I+D ENTRE NUESTRO KNOW-HOW
Es capaz de ver las relaciones entre proyectos y buscar posibles soluciones conjuntas mediante el análisis del dominio de conocimiento y la tecnología

NIVEL 5 - DISEÑO DE PRINCIPIOS DE ARQUITECTURA
Debe participar en el diseño y mejora de los principios de arquitectura y desarrollo de la empresa mediante la colaboración con otros ingenieros

NIVEL 5 - REGISTRAR Y PROPONER OPORTUNIDADES DE MEJORA
Identifica las oportunidades de mejora y las registra. Propone y diseña nueva ideas para mejorar nuestros procesos en base a la experiencia y una evaluación objetiva.

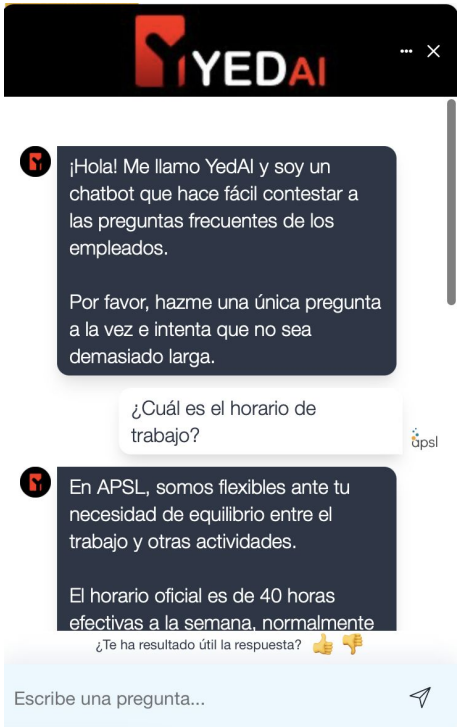
Habilidades

Nivel y descripción de las habilidades por cada dimensión

Programación

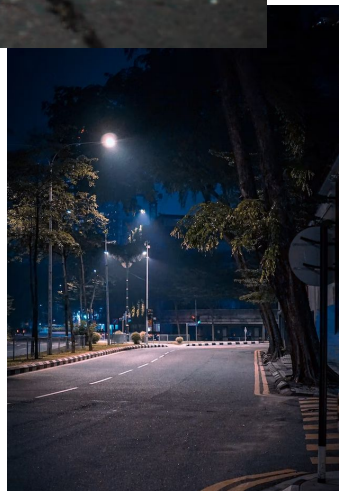
PROGRAMACIÓN

NIVEL 3 - EDGE CASES DETECTION



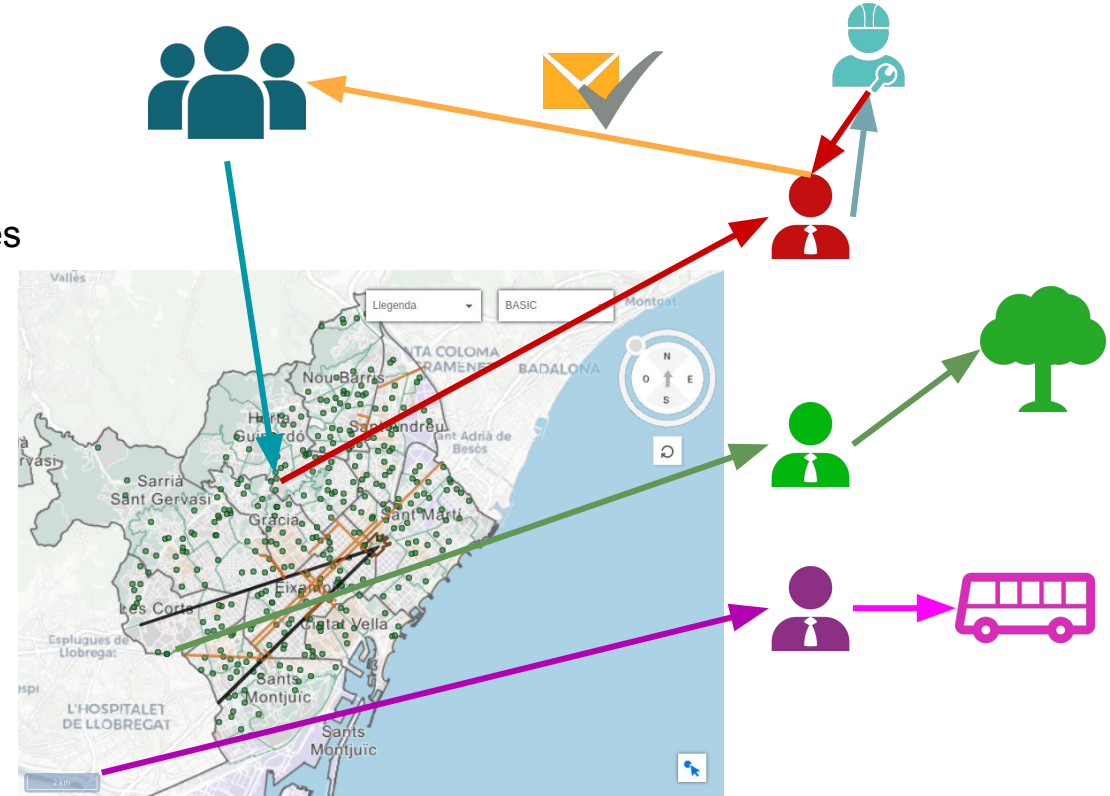
Nuestro ejemplo: IRIS2

- Gestión de comunicaciones ciudadanas
- Todas las peticiones, incidencias y mantenimientos de la ciudad son gestionados aquí
- Implementado para el Ajuntament de Barcelona.
- Reimplementación de IRIS(1)
- Futuro Open Source



Nuestro ejemplo: IRIS2

- Sistema de envío de SMS
- API del padrón
- Servicios de gestión externa
- Procesos y flujos
- Asignación de tareas con diferentes estrategias
- Proveedor de mapas y geocoding
- Almacenamiento de ficheros
- Lógica de permisos



apsl

EXPECTATIVA

- Nueva funcionalidad
- Programar 1 vez --> usar N veces
- Colaborar
- Solución global a la primera

apsl

REALIDAD

- Deadlines
- No entendemos el problema
- Parálisis por análisis
- Complejidad software genérico
- Técnicas de programación

Acabamos desistiendo

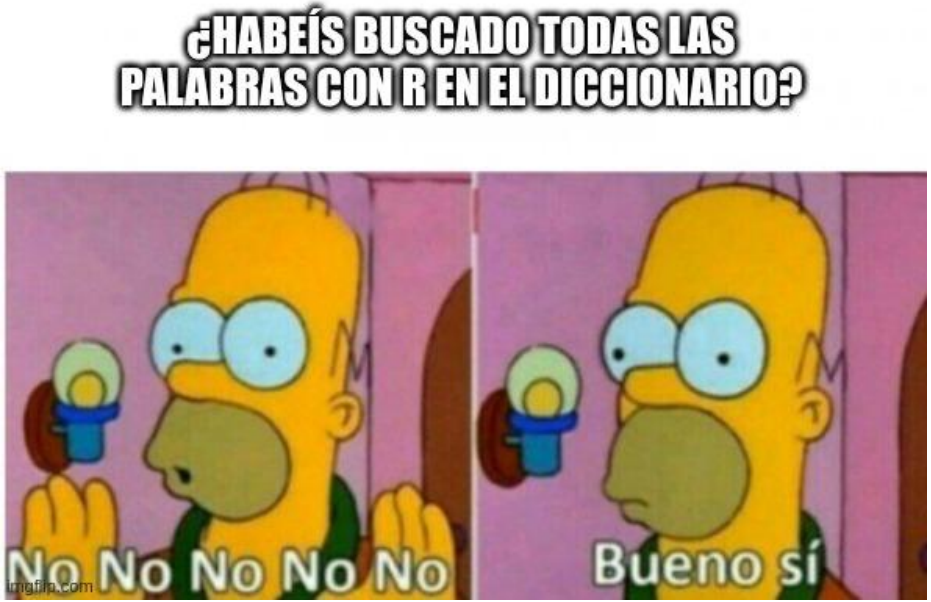
Esta estrategia no funciona…

apsl

MEJOR SEGUIR OTRO CAMINO:

RESOLVER, ROMPER Y RECOMBINAR

Disclaimer #1, #2

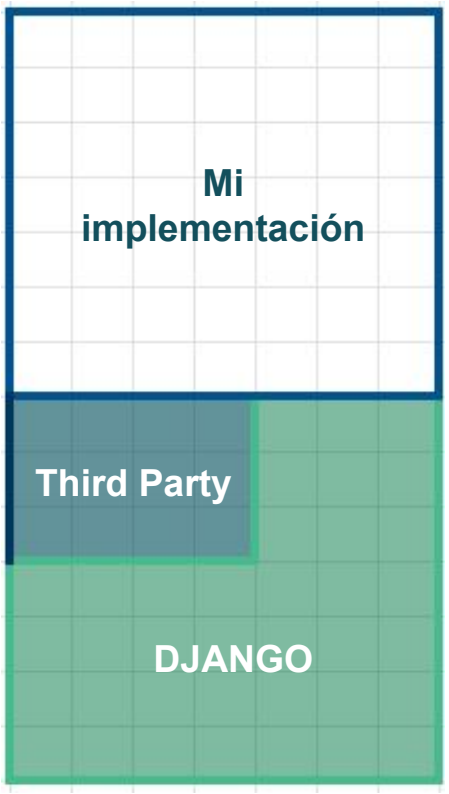


Resolver

- Menos presión de entrega
- Resolver problemas de uno en uno

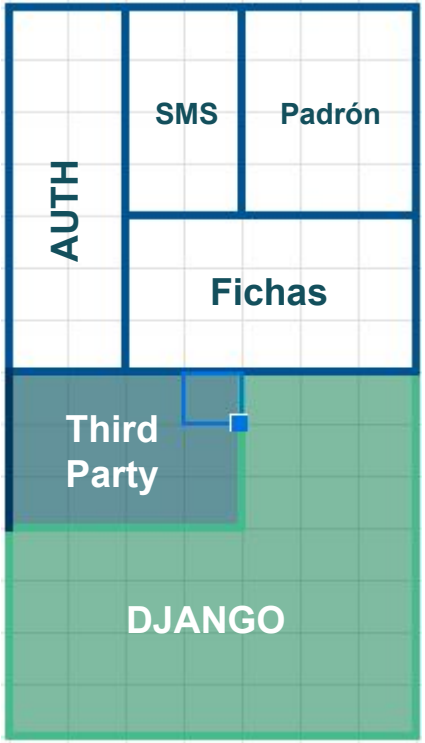
- Entender mejor
- Foto global + detalles

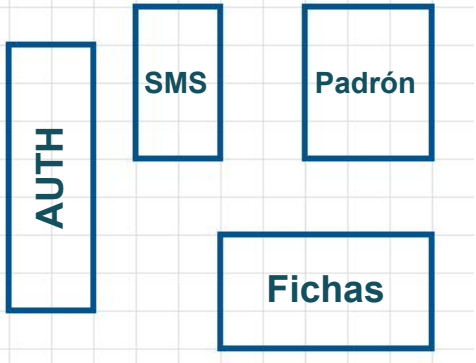
- A la gente le resulta más fácil pensar en abstracto si partimos de lo concreto



Romper

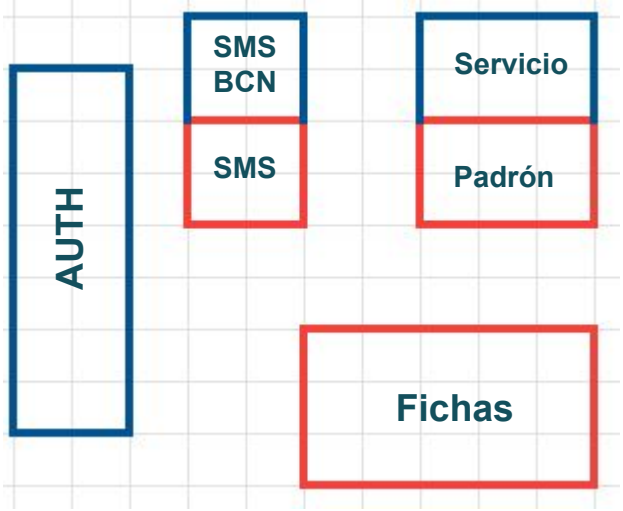
- Separar piezas
- Revisar unión
- Módulos Plug&Play
- Principios de diseño
  - separar responsabilidades
  - estructurar código
  - controlar dependencias

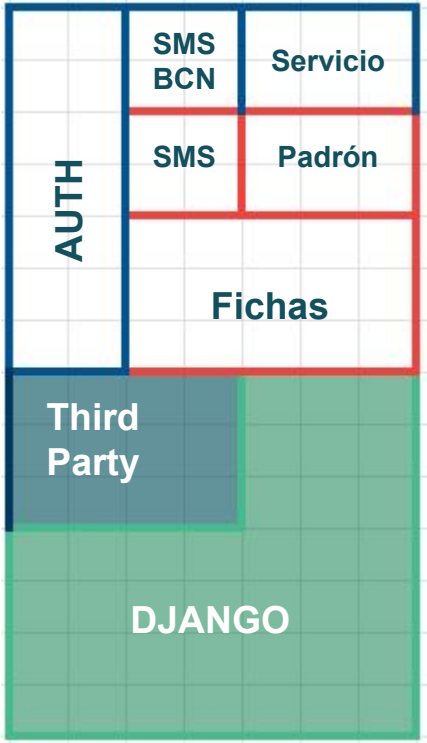




Recombinar

- Alcance:
  - ¿Qué resolvemos?
  - Restricciones.
- De módulos fijos a otros tipos de conexiones
- Puntos para extender





apsl

PRINCIPIOS DE DISEÑO

apsl

Potencial de reutilización no equivale a código reutilizable

CONFIGURABLE

PLUG & PLAY

EXTENSIBLE

FÁCIL AÑADIR FUNCIONALIDAD

INTEROPERABLE

OTROS SISTEMAS SE PUEDEN INTEGRAR

Identificar las responsabilidades

- **Pensar** qué requisitos o flujos podrían cambiar
- **Identificar** qué unidad de código es la responsable
- **Definir** la interfaz (Clases Base abstracta o firma de una función)
- **Permitir** inyectar la dependencia

Responsabilidad:

Asignar la tarea a la persona que la tiene que resolver.

Clase base: Derivator
Acción principal:
Derivator.derivate(record_card)

Estrategias:

- Directa: DirectDerivator
- Distrito: DistrictDerivator
- Territorial: GisDerivator

Invertir dependencias

- Clases abstractas:
  - **QUÉ** hace la aplicación
- Clases concretas:
  - Para cada **QUÉ** puede haber varios **CÓMO**

Construye sobre el framework

- Construye especializaciones sobre el **framework**.
- Aprende las **estrategias de librerías open source**.
- Bajar la barrera de entrada
- Código predecible.

Cuidado con la herencia

- No abusar de la herencia múltiple
- Evitar “clases dios”
- Evitar “clases frankenstein”.
- La flexibilidad a veces significa equivocarse.

apsl

HERRAMIENTAS

Apps

- Unidad básica de Django
- Eventos del ciclo de vida

- Debemos utilizar otros elementos básicos de Django
- Podemos añadir admin.py, templates, etc.

```
class MyLibraryConfig(AppConfig):
    name = 'mylibrary'

    def ready(self):
        # Register custom signal events
        # Initialization...
```

Inyectar dependencias directas: import_string

Librería:

```
if settings.SMS_BACKEND is not None:
    try:
        send_sms = import_string(settings.SMS_BACKEND)
        send_sms(record_card.pk, send_real_sms=True)
    except ImportError:
        logger.info(f"Unable to locate the module
                    {settings.SMS_BACKEND}")
```

Inyectar dependencias directas: import_string

Implementación:

```
def sms_send(record_card_id):
    record_card = RecordCard.objects.get(pk=record_card_id)
    message = render_record_response(record_card)
    result = SmsService().send_sms(
                    message,
                    record_card.get_telephone())
    return result
```

```
SMS_BACKEND = 'integrations.services.sms.sms_send'
```

Registro / Inyección

- Una dependencia se puede resolver con muchas implementaciones.
- Se decide qué clase usar en base a algún criterio.

- En Django encontramos el Django Admin.
- En IRIS: conexión con proveedores.

```
@admin.register(WorkflowPlan)
class WorkflowPlanAdmin(admin.ModelAdmin):
    list_display = ("workflow", "responsible_profile",
                    "start_date_process", "action_required")
```

apsl

Registro / Inyección

Conexión con N proveedores

1. Clase base

```python
class ExternalValidator(metaclass=ABCMeta):
    """
    This validators are meant for send and validate a card to an external service.
    If the service responds OK, then the record card is validated and set to its
next
    state, typically in "External Management".
    """
    def __init__(self, record_card):
        self.record_card = record_card

    @abstractmethod
    def validate(self, **kwargs):
        """
        :return: True if the record card is validated within the external service.
        """
        pass

    def handle_state_change(self, **kwargs):
        return False
```

apsl

2. Crear un registro para inyectar implementaciones (por ejemplo un dict)

```python
registry = {}


def register(cls, codename, override=False):
    if codename in registry and not override:
        raise Exception(f"Already registered class with codename {codename}")
    registry[codename] = cls
    return cls


def get_external_validator(record: RecordCard) -> ExternalValidator:
    sender_codename = getattr(record,'external_sender', None)
    validator_cls = external_validator_registry.get(sender_codename, None)
    return validator_cls(record_card, external_service) if validator_cls else None
```

3. El usuario de la librería implementa su clase

```python
from mylibrary import external_validators

@external_validators.register(codename="MY_SERVICE_IMPLEMENTATION")
class ExternalServiceSender(ExternalValidator):

    def validate(self, **kwargs):
        # DO SOMETHING TO SEND
        self.send_to_service(self.get_parameters(), self.get_headers(), **kwargs)
        return self.external_id
```

Inyectar dependencias: Autodiscover

- Estilo django-admin
- Cada app django podrá añadir un <tu_concepto>.py (igual que con admin.py)

```
class MyLibraryApp(AppConfig):

    def ready(self):
        # Importa en cada APP de INSTALLED_APPS el fichero
        # external_senders.py
        autodiscover_modules('external_senders',
                             register_to=registry)
```

Pipeline

- En ocasiones se deben probar varias estrategias.
  - Authentication
  - Template Loaders
- O se aplican varios pasos de un proceso.
  - Middlewares
- En base a una prioridad.

Pipeline: buscar hasta que uno dé resultado

```
def _authenticate(self):
    for authenticator in self.authenticators:
        try:
            user_auth_tuple = authenticator.authenticate(self)
        except exceptions.APIException:
            self._not_authenticated()
            raise

        if user_auth_tuple is not None:
            self._authenticator = authenticator
            self.user, self.auth = user_auth_tuple
            return

    self._not_authenticated()
```

Chain of responsibility: middlewares

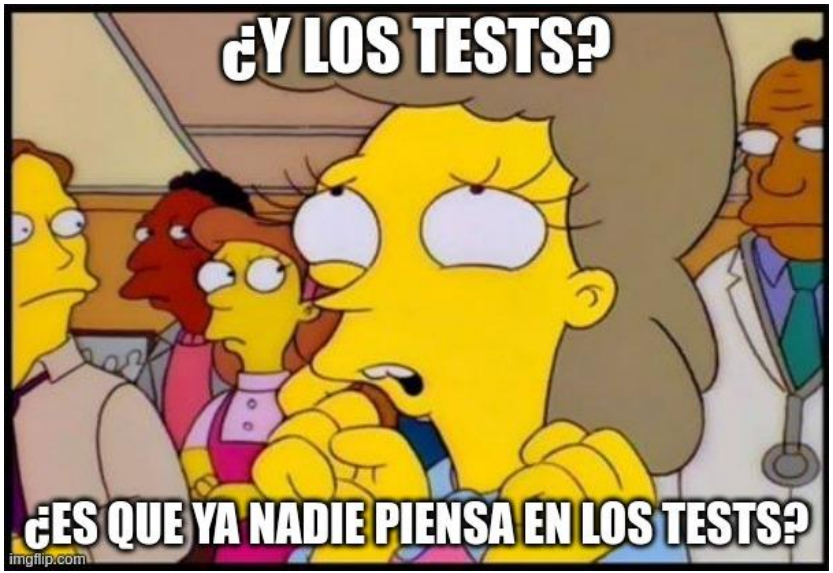
```
MIDDLEWARE = [
    "django_prometheus.middleware.PrometheusBeforeMiddleware",
    "django.middleware.security.SecurityMiddleware",
    "django.middleware.locale.LocaleMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "corsheaders.middleware.CorsMiddleware",  # Django Cors Headers
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
    "django_prometheus.middleware.PrometheusAfterMiddleware",
    "main.middleware.ApplicationMiddleware",
]
```

Eventos (Signals)

- Ofrecer señales en los eventos más importantes de la lógica.
- Evitamos a los usuarios hacer malabares para sobreescribir las clases.

```
record_card_created = Signal(providing_args=["record_card"])
record_card_state_changed = Signal(providing_args=["record_card"])
record_card_directly_closed = Signal(providing_args=["record_card"])
record_card_resend_answer = Signal(providing_args=["record_card"])
```

Pero además puedes ofrecer clases base para que los usuarios de tu librería implementen sus tests más rápido



Dev Experience

- Out of the box → caso común.
- Clases fáciles de usar.
- Clases base de testing
  - Hacer la vida más fácil
  - Fomentar buenas prácticas
- Presentar los requisitos y el problema que queremos resolver.
  - Documentación

Conclusiones

- No te líes, **resuelve el problema** y haz varias iteraciones, nada mejor que software funcionando.
- Si sientes **seguridad** y dominas las técnicas, adelante, **hazlo desde el inicio**. Lo importante es avanzar.
- Piensa en definir las **responsabilidades**, diseña abstracciones y ofrece implementaciones de serie.
- **Django ofrece herramientas** para implementar tus propios módulos reutilizables que el propio framework usa.
- Para una buena **DevExperience** toma de referencia a los paquetes que más éxito tienen o el propio django.


SPONSOR TIME
¡Estamos contratando!
apsl
apsl.tech

apsl

¡Gracias!