Atualização: models.py, admin.py e views.py

Incluí abaixo as versões sugeridas e prontas dos ficheiros `models.py`, `admin.py` e `views.py`. Cada secção está marcada com o nome do ficheiro e contém código que podes copiar directamente para o teu projecto `lab`.

`models.py`

```
from datetime import date
from django.db import models
from django.conf import settings
from django.core.exceptions import ValidationError
from django.utils.html import format_html
from django.utils import timezone
from django.db import transaction

# -------------------- Paciente -----------------------------
class Paciente(models.Model):
    id = models.BigAutoField(primary_key=True, verbose_name='NID')
    nome = models.CharField(max_length=255, verbose_name='Nome')
    data_nascimento = models.DateField(null=True, blank=True,
verbose_name='Data de nascimento')
    genero = models.CharField(max_length=32, choices=[('M', 'Masculino'), ('F',
'Feminino')], verbose_name='Sexo')
    telefone = models.CharField(max_length=32, blank=True,
verbose_name='Telefone')
    residencia = models.CharField(max_length=255, blank=True,
verbose_name='Residência')
    proveniencia = models.CharField(max_length=128, blank=True,
verbose_name='Proveniência')
    nacionalidade = models.CharField(max_length=64, blank=True,
verbose_name='Nacionalidade')
    numero_id = models.CharField(max_length=64, unique=True,
verbose_name='Identidade')
    historico_medico = models.TextField(blank=True, null=True,
verbose_name='Histórico')
    created_at = models.DateTimeField(auto_now_add=True, verbose_name='Entrada')

    class Meta:
        verbose_name = "Paciente"
        verbose_name_plural = "Pacientes"
        ordering = ['nome']
```

```
    @property
    def idade(self):
        if not self.data_nascimento:
            return None
        hoje = date.today()
        idade_em_anos = hoje.year - self.data_nascimento.year
        if (hoje.month, hoje.day) < (self.data_nascimento.month,
self.data_nascimento.day):
            idade_em_anos -= 1
        return idade_em_anos

    def __str__(self):
        idade_display = self.idade_display()
        return f"{self.nome} ({self.numero_id}) - Idade: {idade_display}"

    def idade_display(self):
        if self.idade is None:
            return "—"

        # Menores de 2 anos: exibir detalhe meses/dias
        if self.idade < 2:
            hoje = date.today()
            idade_total_dias = (hoje - self.data_nascimento).days
            idade_em_anos = idade_total_dias // 365
            idade_em_meses = (idade_total_dias % 365) // 30
            idade_em_dias = idade_total_dias % 30

            if idade_em_anos > 0:
                return f"{idade_em_anos} ano{'s' if idade_em_anos > 1 else ''},
{idade_em_meses} mês{'es' if idade_em_meses != 1 else ''} e {idade_em_dias}
dia{'s' if idade_em_dias != 1 else ''}"
            elif idade_em_meses > 0:
                return f"{idade_em_meses} mês{'es' if idade_em_meses != 1 else
''} e {idade_em_dias} dia{'s' if idade_em_dias != 1 else ''}"
            else:
                return f"{idade_em_dias} dia{'s' if idade_em_dias != 1 else ''}"

        if self.idade == 1:
            return f"{self.idade} ano"

        return f"{self.idade} anos"

    idade_display.short_description = "Idade"


# -------------------- Exame --------------------------------
class Exame(models.Model):
    id = models.AutoField(primary_key=True)
```

```
    nome = models.CharField(max_length=255, verbose_name='Exames')
    descricao = models.TextField(blank=True, help_text="Descrição",
verbose_name='Descrição')
    valor_ref = models.CharField(max_length=64, blank=True,
verbose_name='Referência')
    unidade = models.CharField(max_length=32, blank=True,
verbose_name='Unidades')
    trl_horas = models.PositiveIntegerField(default=24,
verbose_name='TRL(horas)')

    class Meta:
        verbose_name = "Exame"
        verbose_name_plural = "Exames"
        ordering = ['nome']

    def __str__(self):
        return f'#{self.id} - {self.nome}'

    def display_valor_ref(self):
        return self.valor_ref or "—"
    display_valor_ref.short_description = "Valor de referência"

    def tempo_resposta_display(self):
        if self.trl_horas > 1:
            return f'{self.trl_horas} horas'
        else:
            return f"{self.trl_horas} hora"
    tempo_resposta_display.short_description = "TRL"


# -------------------- Requisição de Análises -----------------
class RequisicaoAnalise(models.Model):
    STATUS_CHOICES = [
        ('PENDENTE', 'Pendente'),
        ('POR_PREENCHER', 'Por preencher'),
        ('PREENCHIDO', 'Preenchido'),
        ('VALIDADO', 'Validado'),
    ]

    paciente = models.ForeignKey(Paciente, on_delete=models.CASCADE,
verbose_name="Paciente")
    exames = models.ManyToManyField('Exame', through='ItemRequisicao',
blank=True, verbose_name='Exames requisitados')
    analista = models.ForeignKey(settings.AUTH_USER_MODEL, null=True,
blank=True, on_delete=models.SET_NULL, verbose_name='Técnico de Laboratório')
    observacoes = models.TextField(blank=True, verbose_name="Observações
adicionais")
    created_at = models.DateTimeField(auto_now_add=True, verbose_name='Criado
```

```
em')
    status = models.CharField(max_length=24, choices=STATUS_CHOICES,
default='PENDENTE')

    @property
    def exames_list(self):
        return Exame.objects.filter(itemrequisicao__requisicao=self)

    @property
    def exames_count(self):
        return self.exames_list.count()

    def exames_summary(self):
        names = list(self.exames_list.values_list('nome', flat=True)[:5])
        summary = ", ".join(names)
        if self.exames_count > 5:
            summary += " …"
        return summary or "—"
    exames_summary.short_description = "Resumo dos Exames"

    def observacoes_short(self):
        return (self.observacoes[:80] + '…') if self.observacoes and
len(self.observacoes) > 80 else (self.observacoes or "—")
    observacoes_short.short_description = "Observações"

    def __str__(self):
        return f"Req - #{self.id} — {self.paciente.nome.upper()} | Exames:
{self.exames_summary()}"

    class Meta:
        verbose_name = "Requisição de Análises"
        verbose_name_plural = "Requisições de Análises"
        ordering = ['-id']

    # ----------------- geração de placeholders -----------------
    def generate_result_placeholders(self, force=False):
        """
        Cria Resultados e ResultadoItems placeholders para cada exame desta
requisição.
        Se force == True, tenta não duplicar; se precisa recriar, passar
force=True apaga existentes (usar com cuidado).
        """
        from .models import Resultado, ExameCampoResultado, ResultadoItem,
ItemRequisicao, Exame

        exames =
Exame.objects.filter(itemrequisicao__requisicao=self).distinct()
```

```
        with transaction.atomic():
            if force:
                # Atenção: apagar resultados existentes apaga histórico — usar
com auditoria se necessário
                Resultado.objects.filter(requisicao=self).delete()

            resultados_to_create = []
            resultado_items_to_create = []

            # criar objetos Resultado apenas se não existirem
            for exame in exames:
                if not Resultado.objects.filter(requisicao=self,
exame=exame).exists():
                    resultados_to_create.append(Resultado(requisicao=self,
exame=exame))

            if resultados_to_create:
                Resultado.objects.bulk_create(resultados_to_create)

            # criar ResultadoItem para cada campo de cada exame
            resultados = Resultado.objects.filter(requisicao=self,
exame__in=exames).select_related('exame')
            for resultado in resultados:
                campos =
ExameCampoResultado.objects.filter(exame=resultado.exame)
                for campo in campos:
                    exists = ResultadoItem.objects.filter(requisicao=self,
exame_campo=campo).exists()
                    if not exists:
                        resultado_items_to_create.append(
                            ResultadoItem(
                                requisicao=self,
                                exame_campo=campo,
                                unidade=campo.exame.unidade or "",
                                valor_referencia=campo.exame.valor_ref or ""
                            )
                        )

            if resultado_items_to_create:
                ResultadoItem.objects.bulk_create(resultado_items_to_create)

            # atualizar status se havia placeholders criados
            if resultados_to_create or resultado_items_to_create:
                if self.status == 'PENDENTE':
                    self.status = 'POR_PREENCHER'
                    self.save(update_fields=['status'])
```

```python
# -------------------- Item da Requisição ------------------
class ItemRequisicao(models.Model):
    requisicao = models.ForeignKey(RequisicaoAnalise, on_delete=models.CASCADE,
verbose_name='Requisição')
    exame = models.ForeignKey(Exame, on_delete=models.CASCADE,
verbose_name='Exame')

    def __str__(self):
        return f"{self.requisicao} | {self.exame}"

    class Meta:
        verbose_name = "Item de Requisição"
        verbose_name_plural = "Itens de Requisição"


# -------------------- Resultado ---------------------------
class Resultado(models.Model):
    requisicao = models.ForeignKey(RequisicaoAnalise, on_delete=models.CASCADE,
related_name='resultados', verbose_name='Requisição')
    exame = models.ForeignKey(Exame, on_delete=models.CASCADE,
verbose_name='Exame')
    resultado = models.TextField(null=True, blank=True,
verbose_name="Resultado")
    unidade = models.CharField(max_length=32, blank=True,
verbose_name='Unidade')
    valor_referencia = models.CharField(max_length=64, blank=True,
verbose_name='Valor de referência')
    data_insercao = models.DateTimeField(auto_now=True, verbose_name='Data de
inserção')
    validado = models.BooleanField(default=False, verbose_name='Validado')
    validado_por = models.ForeignKey(settings.AUTH_USER_MODEL, null=True,
blank=True, related_name='resultados_validados', on_delete=models.SET_NULL,
verbose_name='Validado por')
    data_validacao = models.DateTimeField(null=True, blank=True,
verbose_name='Data de validação')

    class Meta:
        verbose_name = "Resultado de Exame"
        verbose_name_plural = "Resultados de Exames"
        unique_together = (('requisicao', 'exame'),)
        ordering = ['requisicao', 'exame']

    def save(self, *args, **kwargs):
        if self.exame:
            if not self.unidade:
                self.unidade = self.exame.unidade
            if not self.valor_referencia:
                self.valor_referencia = self.exame.valor_ref
```

```
        if self.validado and not self.data_validacao:
            self.data_validacao = timezone.now()
        super().save(*args, **kwargs)

    def is_valid_display(self):
        return format_html('<b style="color:{};">{}</b>', 'green' if
self.validado else 'red', 'Sim' if self.validado else 'Não')
    is_valid_display.short_description = "Validado"

    def formatted_data_insercao(self):
        return self.data_insercao.strftime("%d/%m/%Y %H:%M") if
self.data_insercao else "—"
    formatted_data_insercao.short_description = "Inserido em"

    def __str__(self):
        return f"Resultado: {self.requisicao} - {self.exame}"


# -------------------- Campos dinâmicos por exame ----------------
class ExameCampoResultado(models.Model):
    exame = models.ForeignKey(Exame, on_delete=models.CASCADE,
related_name='campos')
    nome_campo = models.CharField(max_length=255)
    tipo_campo = models.CharField(max_length=50, choices=[('text', 'Texto'),
('number', 'Número'), ('bool', 'Sim/Não'), ('percent', 'Percentual')])
    obrigatorio = models.BooleanField(default=True)

    def __str__(self):
        return f"{self.exame.nome} | {self.nome_campo}"


# -------------------- ResultadoItem -------------------------
class ResultadoItem(models.Model):
    requisicao = models.ForeignKey(RequisicaoAnalise, on_delete=models.CASCADE,
related_name='resultado_items')
    exame_campo = models.ForeignKey(ExameCampoResultado,
on_delete=models.CASCADE, verbose_name='Campo do exame')
    resultado = models.CharField(max_length=512, blank=True)
    unidade = models.CharField(max_length=32, blank=True)
    valor_referencia = models.CharField(max_length=64, blank=True)

    def save(self, *args, **kwargs):
        if self.exame_campo and self.exame_campo.exame:
            self.unidade = self.exame_campo.exame.unidade or self.unidade
            self.valor_referencia = self.exame_campo.exame.valor_ref or
self.valor_referencia
        super().save(*args, **kwargs)
```

```
    def __str__(self):
        return f"{self.exame_campo.exame.nome} | {self.exame_campo.nome_campo}
| {self.resultado or '—'}"
```

---

admin.py

```
from django.contrib import admin
from django import forms
from django.http import HttpResponse, HttpResponseRedirect
from django.urls import path
from django.utils import timezone
from django.contrib import messages
from django.contrib.auth.models import Group
from .models import Paciente, Exame, RequisicaoAnalise, ItemRequisicao,
Resultado, ResultadoItem, ExameCampoResultado
from .utils.pdf_generator import gerar_pdf_requisicao, gerar_pdf_resultados


# ============================================================
@admin.register(Paciente)
class PacienteAdmin(admin.ModelAdmin):
    list_display = ('id', 'nome', 'genero', 'idade_display', 'telefone',
'proveniencia', 'nacionalidade', 'numero_id', 'created_at')
    search_fields = ('nome', 'numero_id', 'telefone')
    list_filter = ('genero', 'nacionalidade', 'proveniencia')
    ordering = ('nome',)
    list_per_page = 25
    readonly_fields = ('created_at',)

    def has_view_permission(self, request, obj=None):
        return request.user.is_authenticated

    def has_add_permission(self, request):
        user = request.user
        return (
            user.is_superuser or
            user.groups.filter(name__in=['Administrador', 'Administrativo',
'Técnico de Laboratório']).exists()
        )

    def has_change_permission(self, request, obj=None):
        user = request.user
        return (
            user.is_superuser or
```

```
            user.groups.filter(name__in=['Administrador', 'Administrativo',
'Técnico de Laboratório']).exists()
        )

    def has_delete_permission(self, request, obj=None):
        user = request.user
        return user.is_superuser or
user.groups.filter(name='Administrador').exists()


# ============================================================
@admin.register(Exame)
class ExameAdmin(admin.ModelAdmin):
    list_display = ('id', 'nome', 'descricao', 'valor_ref', 'unidade',
'trl_horas', 'tempo_resposta_display')
    search_fields = ('nome', 'descricao')
    ordering = ('nome',)
    list_per_page = 30

    def has_view_permission(self, request, obj=None):
        return request.user.is_authenticated

    def has_add_permission(self, request):
        user = request.user
        return user.is_superuser or
user.groups.filter(name='Administrador').exists()

    def has_change_permission(self, request, obj=None):
        user = request.user
        return user.is_superuser or
user.groups.filter(name='Administrador').exists()

    def has_delete_permission(self, request, obj=None):
        user = request.user
        return user.is_superuser or
user.groups.filter(name='Administrador').exists()


# ============================================================
class ItemRequisicaoInline(admin.TabularInline):
    model = ItemRequisicao
    extra = 3
    autocomplete_fields = ['exame']
    show_change_link = True
    verbose_name = "Item de Exame"
    verbose_name_plural = "Itens de Exame"
```

```
class RequisicaoAnaliseAdminForm(forms.ModelForm):
    class Meta:
        model = RequisicaoAnalise
        fields = '__all__'
        widgets = {
            'exames': forms.CheckboxSelectMultiple()
        }


@admin.register(RequisicaoAnalise)
class RequisicaoAnaliseAdmin(admin.ModelAdmin):
    form = RequisicaoAnaliseAdminForm
    inlines = [ItemRequisicaoInline]
    list_display = ('id', 'paciente', 'exames_count', 'analista', 'status',
'observacoes_short')
    search_fields = ('paciente__nome', 'paciente__numero_id',
'analista__username')
    list_filter = ('status', 'analista', 'paciente')
    autocomplete_fields = ('paciente', 'analista')
    readonly_fields = ('analista',)
    actions = ['baixar_pdf_requisicao']
    list_per_page = 20

    def get_urls(self):
        urls = super().get_urls()
        custom_urls = [
            path('<int:requisicao_id>/preencher-resultados/',
self.admin_site.admin_view(self.preeencher_resultados_view),
name='lab_requisicao_preencher'),
        ]
        return custom_urls + urls

    def preeencher_resultados_view(self, request, requisicao_id):
        # redireciona para a view de app normal onde o preenchimento acontece
        return HttpResponseRedirect(f'/lab/requisicao/{requisicao_id}/preencher-
resultados/')

    def save_model(self, request, obj, form, change):
        if not obj.analista:
            obj.analista = request.user
        super().save_model(request, obj, form, change)

    def has_view_permission(self, request, obj=None):
        return request.user.is_authenticated

    def has_add_permission(self, request):
        user = request.user
        return (
```

```
            user.is_superuser or
            user.groups.filter(name__in=['Administrador', 'Administrativo',
'Técnico de Laboratório']).exists()
        )

    def has_change_permission(self, request, obj=None):
        user = request.user
        return (
            user.is_superuser or
            user.groups.filter(name__in=['Administrador', 'Administrativo',
'Técnico de Laboratório']).exists()
        )

    def has_delete_permission(self, request, obj=None):
        user = request.user
        return user.is_superuser or
user.groups.filter(name='Administrador').exists()

    def baixar_pdf_requisicao(self, request, queryset):
        if queryset.count() != 1:
            self.message_user(request, "Selecione exactamente uma requisição
para baixar o PDF.", level=messages.WARNING)
            return
        requisicao = queryset.first()
        pdf_content = gerar_pdf_requisicao(requisicao)
        response = HttpResponse(pdf_content, content_type='application/pdf')
        response['Content-Disposition'] = f'attachment;
filename="requisicao_{requisicao.id}.pdf"'
        return response
    baixar_pdf_requisicao.short_description = "Baixar PDF da requisição
selecionada"


# ============================================================
@admin.register(Resultado)
class ResultadoAdmin(admin.ModelAdmin):
    list_display = (
        'requisicao', 'exame', 'resultado', 'unidade', 'valor_referencia',
        'is_valid_display', 'formatted_data_insercao',
        'validado_por', 'data_validacao'
    )
    search_fields = ('requisicao__paciente__nome',
'requisicao__paciente__numero_id', 'exame__nome')
    list_filter = ('validado', 'data_insercao', 'data_validacao')
    autocomplete_fields = ('requisicao', 'exame')
    readonly_fields = ('data_insercao', 'data_validacao')
    actions = ['validar_resultados', 'baixar_pdf_resultados']
    list_per_page = 40
```

```
    def has_view_permission(self, request, obj=None):
        return request.user.is_authenticated

    def has_add_permission(self, request):
        user = request.user
        return user.is_superuser or
user.groups.filter(name__in=['Administrador', 'Técnico de
Laboratório']).exists()

    def has_change_permission(self, request, obj=None):
        user = request.user
        return user.is_superuser or
user.groups.filter(name__in=['Administrador', 'Técnico de
Laboratório']).exists()

    def has_delete_permission(self, request, obj=None):
        user = request.user
        return user.is_superuser or
user.groups.filter(name='Administrador').exists()

    def validar_resultados(self, request, queryset):
        for res in queryset:
            if res.resultado and not res.validado:
                res.validado = True
                res.data_validacao = timezone.now()
                res.validado_por = request.user
                res.save()
    validar_resultados.short_description = "Validar resultados selecionados"

    def baixar_pdf_resultados(self, request, queryset):
        requisicoes = set(r.requisicao for r in queryset)
        if len(requisicoes) != 1:
            self.message_user(request, "Selecione resultados de uma única
requisição para gerar o PDF.", level=messages.WARNING)
            return
        requisicao = list(requisicoes)[0]
        pdf_content = gerar_pdf_resultados(requisicao)
        response = HttpResponse(pdf_content, content_type='application/pdf')
        response['Content-Disposition'] = f'attachment;
filename="resultados_{requisicao.id}.pdf"'
        return response
    baixar_pdf_resultados.short_description = "Baixar PDF dos resultados
selecionados"


# Registrar modelos auxiliares
@admin.register(ExameCampoResultado)
```

```
class ExameCampoResultadoAdmin(admin.ModelAdmin):
    list_display = ('exame', 'nome_campo', 'tipo_campo', 'obrigatorio')
    search_fields = ('exame__nome', 'nome_campo')


@admin.register(ResultadoItem)
class ResultadoItemAdmin(admin.ModelAdmin):
    list_display = ('requisicao', 'exame_campo', 'resultado', 'unidade',
'valor_referencia')
    search_fields = ('requisicao__paciente__nome', 'exame_campo__nome_campo')
    list_filter = ('exame_campo__exame',)
```

`views.py` (nova view para preencher / revisar / validar)

```
from django.shortcuts import get_object_or_404, render, redirect
from django import forms
from django.db import transaction
from django.contrib.auth.decorators import login_required, permission_required
from django.urls import reverse
from django.http import HttpResponse
from .models import RequisicaoAnalise, ResultadoItem, Resultado


class ResultadosDinamicosForm(forms.Form):
    def __init__(self, resultado_items_qs, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # adicionar campos dinamicamente
        for ri in resultado_items_qs:
            field_name = f"ri_{ri.id}"
            tipo = ri.exame_campo.tipo_campo
            required = ri.exame_campo.obrigatorio
            label = f"{ri.exame_campo.exame.nome} — {ri.exame_campo.nome_campo}"
            help_text = f"Unidade: {ri.unidade} | Ref: {ri.valor_referencia}" if
(ri.unidade or ri.valor_referencia) else ''

            if tipo == 'text':
                self.fields[field_name] = forms.CharField(required=required,
label=label, help_text=help_text)
            elif tipo == 'number':
                self.fields[field_name] = forms.DecimalField(required=required,
label=label, help_text=help_text)
            elif tipo == 'percent':
                self.fields[field_name] = forms.DecimalField(required=required,
label=label, help_text=help_text, max_value=100)
```

```
            elif tipo == 'bool':
                self.fields[field_name] = forms.ChoiceField(choices=[('', '—'),
('Sim', 'Sim'), ('Nao', 'Não')], required=required, label=label,
help_text=help_text)
            else:
                self.fields[field_name] = forms.CharField(required=required,
label=label, help_text=help_text)


@login_required
def preencher_resultados(request, requisicao_id):
    requisicao = get_object_or_404(RequisicaoAnalise, id=requisicao_id)
    # garante placeholders
    requisicao.generate_result_placeholders()

    resultado_items =
ResultadoItem.objects.filter(requisicao=requisicao).select_related('exame_campo',
'exame_campo__exame')

    if request.method == 'POST':
        form = ResultadosDinamicosForm(resultado_items, request.POST)
        if form.is_valid():
            with transaction.atomic():
                updates = []
                for ri in resultado_items:
                    key = f"ri_{ri.id}"
                    val = form.cleaned_data.get(key)
                    ri.resultado = str(val) if val is not None else ''
                    updates.append(ri)
                ResultadoItem.objects.bulk_update(updates, ['resultado'])

                # quando items preenchidos, atualiza status da requisição
                requisicao.status = 'PREENCHIDO'
                requisicao.save(update_fields=['status'])

            return redirect(reverse('lab:revisar_resultados',
args=[requisicao.id]))
    else:
        # inicializar formulário com valores existentes
        initial = {f"ri_{ri.id}": ri.resultado for ri in resultado_items}
        form = ResultadosDinamicosForm(resultado_items, initial=initial)

    # agrupar por exame para o template
    grouped = {}
    for ri in resultado_items:
        grouped.setdefault(ri.exame_campo.exame.nome, []).append(ri)

    return render(request, 'lab/preencher_resultados.html', {
```

```
        'requisicao': requisicao,
        'form': form,
        'grouped': grouped,
    })


@login_required
def revisar_resultados(request, requisicao_id):
    requisicao = get_object_or_404(RequisicaoAnalise, id=requisicao_id)
    resultado_items =
ResultadoItem.objects.filter(requisicao=requisicao).select_related('exame_campo',
'exame_campo__exame')

    # agrupado para facilitar a visualizacao
    grouped = {}
    for ri in resultado_items:
        grouped.setdefault(ri.exame_campo.exame.nome, []).append(ri)

    return render(request, 'lab/revisar_resultados.html', {
        'requisicao': requisicao,
        'grouped': grouped,
    })


@login_required
@permission_required('lab.change_resultado', raise_exception=True)
def validar_resultados_view(request, requisicao_id):
    requisicao = get_object_or_404(RequisicaoAnalise, id=requisicao_id)
    # validar todos os resultados associados
    resultados = Resultado.objects.filter(requisicao=requisicao)
    count = 0
    for r in resultados:
        if not r.validado:
            r.validado = True
            r.validado_por = request.user
            r.data_validacao = timezone.now()
            r.save()
            count += 1

    requisicao.status = 'VALIDADO'
    requisicao.save(update_fields=['status'])

    return redirect(reverse('lab:revisar_resultados', args=[requisicao.id]))
```

Notas finais / passos para integrar

1. Copia cada bloco para o ficheiro correspondente (faz backup antes).
2. Cria templates mínimos: `lab/preencher_resultados.html` e `lab/revisar_resultados.html` (posso gerar exemplos se quiseres).
3. Regista URLs do app `lab` no `lab/urls.py` e inclui no `backend/urls.py`:

```
# lab/urls.py
from django.urls import path
from . import views

app_name = 'lab'

urlpatterns = [
    path('requisicao/<int:requisicao_id>/preencher-resultados/',
views.preencher_resultados, name='preencher_resultados'),
    path('requisicao/<int:requisicao_id>/revisar-resultados/',
views.revisar_resultados, name='revisar_resultados'),
    path('requisicao/<int:requisicao_id>/validar-resultados/',
views.validar_resultados_view, name='validar_resultados'),
]
```

1. Executa `python manage.py makemigrations` e `migrate`.
2. Atenção: se alterares PKs (ex.: converter campos id existentes), pode ser necessário ajuste manual de migrações.
3. Testa o fluxo: criar paciente → criar requisição com exames → abrir `preencher-resultados` → preencher → revisar → validar → gerar PDF.

Se queres, eu gero também os templates básicos (`preencher_resultados.html` e `revisar_resultados.html`) e um `lab/urls.py` completo, ou posso aplicar pequenas alterações conforme preferires. Qual o próximo passo?