🚀Turbo Models Pack — Corpo Humano Holístico v1.0

Abaixo está um **pacote completo de modelos interligados** para dar um salto quântico no seu projeto. Inclui:

- Expansões em **Anatomia**, **Metabolismo**, **Fisiologia**, **Sistema Endócrino** e **Psicologia**
- Relacionamentos ricos (M2M com `through`) para **efeitos de alimentos, nutrientes, hormônios e emoções** sobre **órgãos** e **parâmetros fisiológicos**
- **Scripts de importação** de nutrientes, alimentos, parâmetros e biomarcadores
- Um **serviço de insights** para a Aura explicar *"Alimento → Nutrientes → Órgãos → Parâmetros → Sinais/Sintomas"*
- Dicas de **admin** e **migrações**

> **Observação**: Todos os FKs que referenciam modelos de outros apps usam a string do app (ex.: `'anatomia.Orgao'`) para evitar import circular.

---

1) anatomia/models.py (extensão)

> **Adicione** as classes abaixo ao final do arquivo, mantendo as suas classes já existentes (`SistemaCorporal`, `Orgao`, `Tecido`).

```
# anatomia/models.py (trecho novo)
from django.db import models

class CelulaTipo(models.Model):
    nome = models.CharField(max_length=120)
    orgao = models.ForeignKey('anatomia.Orgao', on_delete=models.CASCADE,
related_name='celulas')
    funcoes = models.TextField(blank=True, null=True, help_text="Funções
principais deste tipo celular no órgão.")

    class Meta:
        unique_together = ('nome', 'orgao')

    def __str__(self):
        return f"{self.nome} ({self.orgao.nome})"

class Receptor(models.Model):
    nome = models.CharField(max_length=120)
    descricao = models.TextField(blank=True, null=True)
    orgaos = models.ManyToManyField('anatomia.Orgao', blank=True,
related_name='receptores', help_text="Órgãos que expressam este receptor")

    class Meta:
        unique_together = ('nome',)
```

```
    def __str__(self):
        return self.nome
```

---

2) metabolismo/models.py (expansão forte)

Substitua o conteúdo atual por este (mantém `RegistroMetabolico` e expande `Alimento`).

```
# metabolismo/models.py
from django.db import models
from django.core.validators import MinValueValidator, MaxValueValidator

TIPO_NUTRIENTE = (
    ('MACRO', 'Macronutriente'),
    ('MICRO', 'Micronutriente'),
)

FAMILIA_NUTRIENTE = (
    ('CARB', 'Carboidrato'),
    ('PROT', 'Proteína/Aminoácido'),
    ('LIP', 'Lipídio/Ácido Graxo'),
    ('VIT', 'Vitamina'),
    ('MIN', 'Mineral'),
    ('FIB', 'Fibra'),
    ('OUT', 'Outro'),
)

TIPO_EFEITO = (
    ('ESTIMULA', 'Estimula'),
    ('INIBE', 'Inibe'),
    ('PROTEGE', 'Protege'),
    ('LESIONA', 'Lesiona'),
    ('MODULA', 'Modula'),
)

class Alimento(models.Model):
    nome = models.CharField(max_length=100, unique=True, help_text="Nome do
alimento. Ex: 'Maçã'")
    calorias_por_100g = models.IntegerField()
    efeito = models.CharField(max_length=255, blank=True, null=True,
help_text="Efeito principal do alimento. Ex: 'Antioxidante'")
    orgaos = models.ManyToManyField('anatomia.Orgao',
through='EfeitoAlimentoOrgao', blank=True, related_name='alimentos')

    def __str__(self):
        return self.nome
```

```
class Nutriente(models.Model):
    nome = models.CharField(max_length=120, unique=True)
    tipo = models.CharField(max_length=6, choices=TIPO_NUTRIENTE)
    familia = models.CharField(max_length=4, choices=FAMILIA_NUTRIENTE,
default='OUT')
    descricao = models.TextField(blank=True, null=True)
    unidade_padrao = models.CharField(max_length=10, default='mg',
help_text="mg, µg, g, IU, etc.")

    def __str__(self):
        return self.nome

class ComposicaoAlimento(models.Model):
    alimento = models.ForeignKey('metabolismo.Alimento',
on_delete=models.CASCADE, related_name='composicao')
    nutriente = models.ForeignKey('metabolismo.Nutriente',
on_delete=models.CASCADE, related_name='alimentos')
    quantidade_por_100g = models.DecimalField(max_digits=10,
decimal_places=3)
    unidade = models.CharField(max_length=10, default='mg')
    biodisponibilidade = models.PositiveSmallIntegerField(default=100,
validators=[MinValueValidator(1), MaxValueValidator(100)])

    class Meta:
        unique_together = ('alimento', 'nutriente')

    def __str__(self):
        return f"{self.alimento.nome} → {self.nutriente.nome}:
{self.quantidade_por_100g}{self.unidade}/100g"

class EfeitoAlimentoOrgao(models.Model):
    alimento = models.ForeignKey('metabolismo.Alimento',
on_delete=models.CASCADE)
    orgao = models.ForeignKey('anatomia.Orgao', on_delete=models.CASCADE)
    tipo_efeito = models.CharField(max_length=10, choices=TIPO_EFEITO,
default='MODULA')
    intensidade = models.SmallIntegerField(default=0,
validators=[MinValueValidator(-3), MaxValueValidator(3)],
help_text="-3 forte negativo, 0 neutro, +3 forte positivo")
    mecanismo = models.TextField(blank=True, null=True, help_text="Resumo do
mecanismo proposto (ex.: antioxidante, anti-inflamatório, lipotrópico)")
    evidencia = models.CharField(max_length=50, blank=True, null=True,
help_text="Nível resumido: observacional, RCT, revisão, etc.")

    class Meta:
        unique_together = ('alimento', 'orgao', 'tipo_efeito')

    def __str__(self):
        return f"{self.alimento.nome} {self.tipo_efeito} {self.orgao.nome}
({self.intensidade})"
```

```
class RegistroMetabolico(models.Model):
    TIPO_CHOICES = [('ENTRADA', 'Entrada'), ('SAIDA', 'Saída')]
    EMOcoes_CHOICES = [
        ('FELIZ', 'Feliz'), ('GRATO', 'Grato'), ('PRODUTIVO', 'Produtivo'),
        ('NORMAL', 'Normal'), ('CANSADO', 'Cansado'), ('ESTRESSADO',
'Estressado'),
        ('ANSIOSO', 'Ansioso'),
    ]

    perfil = models.ForeignKey('usuarios.PerfilUsuario',
on_delete=models.CASCADE, help_text="Conecta o registro ao PerfilUsuario
específico.")
    tipo = models.CharField(max_length=7, choices=TIPO_CHOICES,
default='ENTRADA')
    descricao = models.CharField(max_length=255)
    calorias = models.IntegerField()
    estado_emocional = models.CharField(max_length=15,
choices=EMOcoes_CHOICES, default='NORMAL')
    data_hora = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.perfil.user.username} - {self.tipo}: {self.calorias}
kcal"
```

---

3) fisiologia/models.py (expansão forte)

Substitua o conteúdo atual por este (mantém `ProcessoBiologico` e adiciona parâmetros, biomarcadores, vias e condições clínicas).

```
# fisiologia/models.py
from django.db import models
from django.core.validators import MinValueValidator, MaxValueValidator

TIPO_PARAM = (
    ('SINAL', 'Sinal Vital'),
    ('LAB', 'Laboratorial/Biomarcador'),
    ('FUNC', 'Função Orgânica'),
)

TIPO_EFEITO = (
    ('ESTIMULA', 'Estimula'),
    ('INIBE', 'Inibe'),
    ('PROTEGE', 'Protege'),
    ('LESIONA', 'Lesiona'),
    ('MODULA', 'Modula'),
)

class ProcessoBiologico(models.Model):
```

```
    titulo = models.CharField(max_length=200, help_text="Nome do processo.
Ex: 'Ciclo de Krebs', 'Resposta Inflamatória'")
    descricao = models.TextField()
    orgaos_envolvidos = models.ManyToManyField('anatomia.Orgao',
help_text="Órgãos que participam deste processo.")

    def __str__(self):
        return self.titulo

class ParametroFisiologico(models.Model):
    nome = models.CharField(max_length=120)
    orgao = models.ForeignKey('anatomia.Orgao', on_delete=models.CASCADE,
related_name='parametros')
    tipo = models.CharField(max_length=5, choices=TIPO_PARAM, default='FUNC')
    unidade = models.CharField(max_length=20, blank=True, null=True)
    referencia_min = models.FloatField(blank=True, null=True)
    referencia_max = models.FloatField(blank=True, null=True)
    descricao = models.TextField(blank=True, null=True)

    class Meta:
        unique_together = ('nome', 'orgao')

    def __str__(self):
        return f"{self.nome} ({self.orgao.nome})"

class Biomarcador(models.Model):
    nome = models.CharField(max_length=120, unique=True)
    unidade = models.CharField(max_length=20, blank=True, null=True)
    referencia_min = models.FloatField(blank=True, null=True)
    referencia_max = models.FloatField(blank=True, null=True)
    orgao_relacionado = models.ForeignKey('anatomia.Orgao',
on_delete=models.SET_NULL, null=True, blank=True)

    def __str__(self):
        return self.nome

class ExameLaboratorial(models.Model):
    nome = models.CharField(max_length=120, unique=True)
    biomarcadores = models.ManyToManyField('fisiologia.Biomarcador',
related_name='exames')

    def __str__(self):
        return self.nome

class ViaMetabolica(models.Model):
    nome = models.CharField(max_length=200, unique=True)
    descricao = models.TextField(blank=True, null=True)
    orgaos = models.ManyToManyField('anatomia.Orgao', blank=True)

    def __str__(self):
        return self.nome
```

```
class CondicaoClinica(models.Model):
    nome = models.CharField(max_length=160, unique=True)
    descricao = models.TextField(blank=True, null=True)
    orgaos_afetados = models.ManyToManyField('anatomia.Orgao', blank=True)
    biomarcadores_associados =
models.ManyToManyField('fisiologia.Biomarcador', blank=True)

    def __str__(self):
        return self.nome

class Sintoma(models.Model):
    nome = models.CharField(max_length=160, unique=True)
    descricao = models.TextField(blank=True, null=True)
    orgaos_associados = models.ManyToManyField('anatomia.Orgao', blank=True)
    condicoes = models.ManyToManyField('fisiologia.CondicaoClinica',
blank=True, related_name='sintomas')

    def __str__(self):
        return self.nome

# Liga Nutriente → Parâmetro (efeito fisiológico direto)
class EfeitoNutrienteParametro(models.Model):
    nutriente = models.ForeignKey('metabolismo.Nutriente',
on_delete=models.CASCADE)
    parametro = models.ForeignKey('fisiologia.ParametroFisiologico',
on_delete=models.CASCADE)
    tipo_efeito = models.CharField(max_length=10, choices=TIPO_EFEITO,
default='MODULA')
    intensidade = models.SmallIntegerField(default=0,
validators=[MinValueValidator(-3), MaxValueValidator(3)])
    mecanismo = models.TextField(blank=True, null=True)

    class Meta:
        unique_together = ('nutriente', 'parametro', 'tipo_efeito')

    def __str__(self):
        return f"{self.nutriente.nome} {self.tipo_efeito}
{self.parametro.nome} ({self.intensidade})"
```

---

4) sistema_endocrino/models.py (expansão)

Substitua por este conteúdo (mantém `Glandula` e `Hormonio` e adiciona alvos e eixos).

```
# sistema_endocrino/models.py
from django.db import models
```

```
TIPO_HORMONIO = (
    ('PEPTIDEO', 'Peptídeo'),
    ('ESTEROIDE', 'Esteróide'),
    ('AMINA', 'Amina'),
)

TIPO_EFEITO = (
    ('ESTIMULA', 'Estimula'),
    ('INIBE', 'Inibe'),
    ('PROTEGE', 'Protege'),
    ('LESIONA', 'Lesiona'),
    ('MODULA', 'Modula'),
)

class Glandula(models.Model):
    orgao = models.OneToOneField('anatomia.Orgao', on_delete=models.CASCADE)

    def __str__(self):
        return f"Glândula: {self.orgao.nome}"

class Hormonio(models.Model):
    nome = models.CharField(max_length=100, help_text="Ex: 'Insulina',
'Cortisol'")
    glandula_produtora = models.ForeignKey('sistema_endocrino.Glandula',
on_delete=models.CASCADE)
    funcao_principal = models.TextField()
    tipo = models.CharField(max_length=8, choices=TIPO_HORMONIO,
default='PEPTIDEO')
    meia_vida_min = models.FloatField(blank=True, null=True, help_text="Meia-
vida aproximada em minutos")

    class Meta:
        unique_together = ('nome', 'glandula_produtora')

    def __str__(self):
        return self.nome

class EfeitoHormonalOrgao(models.Model):
    hormonio = models.ForeignKey('sistema_endocrino.Hormonio',
on_delete=models.CASCADE, related_name='efeitos_orgaos')
    orgao_alvo = models.ForeignKey('anatomia.Orgao',
on_delete=models.CASCADE, related_name='efeitos_hormonais')
    tipo_efeito = models.CharField(max_length=10, choices=TIPO_EFEITO,
default='MODULA')
    descricao = models.TextField(blank=True, null=True)

    class Meta:
        unique_together = ('hormonio', 'orgao_alvo', 'tipo_efeito')

    def __str__(self):
        return f"{self.hormonio.nome} → {self.orgao_alvo.nome}
```

```
({self.tipo_efeito})"

class EfeitoHormonalParametro(models.Model):
    hormonio = models.ForeignKey('sistema_endocrino.Hormonio',
on_delete=models.CASCADE)
    parametro = models.ForeignKey('fisiologia.ParametroFisiologico',
on_delete=models.CASCADE)
    tipo_efeito = models.CharField(max_length=10, choices=TIPO_EFEITO,
default='MODULA')
    intensidade = models.SmallIntegerField(default=0)
    mecanismo = models.TextField(blank=True, null=True)

    class Meta:
        unique_together = ('hormonio', 'parametro', 'tipo_efeito')

class EixoEndocrino(models.Model):
    nome = models.CharField(max_length=120, unique=True,
help_text="Ex.: Eixo HPA (Hipotálamo-Hipófise-Adrenal)")
    descricao = models.TextField(blank=True, null=True)
    glandulas = models.ManyToManyField('sistema_endocrino.Glandula',
blank=True)

    def __str__(self):
        return self.nome

class RitmoCircadiano(models.Model):
    hormonio = models.OneToOneField('sistema_endocrino.Hormonio',
on_delete=models.CASCADE, related_name='ritmo')
    pico_hora_local = models.PositiveSmallIntegerField(help_text="Hora local
aproximada do pico (0-23)")
    nadir_hora_local =
models.PositiveSmallIntegerField(help_text="Hora local aproximada do nadir
(0-23)")

    def __str__(self):
        return f"Ritmo de {self.hormonio.nome}"
```

---

5) psicologia/models.py (extensão enxuta)

Adicione ao final do arquivo (mantendo suas classes).

```
# psicologia/models.py (trecho novo)
from django.db import models
from django.core.validators import MinValueValidator, MaxValueValidator

TIPO_EFEITO = (
    ('ESTIMULA', 'Estimula'),
    ('INIBE', 'Inibe'),
```

```
    ('PROTEGE', 'Protege'),
    ('LESIONA', 'Lesiona'),
    ('MODULA', 'Modula'),
)

class EfeitoEmocaoParametro(models.Model):
    emocao = models.ForeignKey('psicologia.EstadoEmocional',
on_delete=models.CASCADE)
    parametro = models.ForeignKey('fisiologia.ParametroFisiologico',
on_delete=models.CASCADE)
    tipo_efeito = models.CharField(max_length=10, choices=TIPO_EFEITO,
default='MODULA')
    intensidade = models.SmallIntegerField(default=0,
validators=[MinValueValidator(-3), MaxValueValidator(3)])
    mecanismo = models.TextField(blank=True, null=True)

    class Meta:
        unique_together = ('emocao', 'parametro', 'tipo_efeito')

    def __str__(self):
        return f"{self.emocao.nome} {self.tipo_efeito} {self.parametro.nome}
({self.intensidade})"
```

---

6) Admin — registre os novos modelos (opcional, mas recomendado)

anatomia/admin.py (adicione imports e registros):

```
from .models import SistemaCorporal, Orgao, Tecido, CelulaTipo,
Receptor
admin.site.register(CelulaTipo)
admin.site.register(Receptor)
```

metabolismo/admin.py (substitua por):

```
from django.contrib import admin
from .models import Alimento, RegistroMetabolico, Nutriente,
ComposicaoAlimento, EfeitoAlimentoOrgao

admin.site.register(Alimento)
admin.site.register(Nutriente)
admin.site.register(ComposicaoAlimento)
admin.site.register(EfeitoAlimentoOrgao)
admin.site.register(RegistroMetabolico)
```

fisiologia/admin.py:

```
from django.contrib import admin
from .models import (
    ProcessoBiologico, ParametroFisiologico, Biomarcador,
ExameLaboratorial,
    ViaMetabolica, CondicaoClinica, Sintoma,
EfeitoNutrienteParametro
)

admin.site.register(ProcessoBiologico)
admin.site.register(ParametroFisiologico)
admin.site.register(Biomarcador)
admin.site.register(ExameLaboratorial)
admin.site.register(ViaMetabolica)
admin.site.register(CondicaoClinica)
admin.site.register(Sintoma)
admin.site.register(EfeitoNutrienteParametro)
```

sistema_endocrino/admin.py:

```
from django.contrib import admin
from .models import Glandula, Hormonio, EfeitoHormonalOrgao,
EfeitoHormonalParametro, EixoEndocrino, RitmoCircadiano

admin.site.register(Glandula)
admin.site.register(Hormonio)
admin.site.register(EfeitoHormonalOrgao)
admin.site.register(EfeitoHormonalParametro)
admin.site.register(EixoEndocrino)
admin.site.register(RitmoCircadiano)
```

psicologia/admin.py:

```
from django.contrib import admin
from .models import EventoGatilho, EstadoEmocional,
ModificadorFisiologico, EfeitoEmocaoParametro

admin.site.register(EfeitoEmocaoParametro)
```

---

7) Scripts de importação (dados iniciais)

7.1) `metabolismo/scripts/import_nutrientes_e_alimentos.py`

```
# metabolismo/scripts/import_nutrientes_e_alimentos.py
from metabolismo.models import Alimento, Nutriente, ComposicaoAlimento,
EfeitoAlimentoOrgao
from anatomia.models import Orgao


def get_orgao(nome):
    try:
        return Orgao.objects.get(nome=nome)
    except Orgao.DoesNotExist:
        print(f"⚠️ Órgão não encontrado: {nome}")
        return None


def seed_nutrientes():
    nutrientes = [
        dict(nome='Vitamina A (retinol)', tipo='MICRO', familia='VIT',
unidade_padrao='µg', descricao='Essencial para visão e epitélios'),
        dict(nome='Vitamina C (ác. ascórbico)', tipo='MICRO', familia='VIT',
unidade_padrao='mg', descricao='Antioxidante; síntese de colágeno'),
        dict(nome='Ômega-3 (EPA/DHA)', tipo='MACRO', familia='LIP',
unidade_padrao='g', descricao='Anti-inflamatório, membranas neuronais'),
        dict(nome='Fibra Solúvel', tipo='MACRO', familia='FIB',
unidade_padrao='g', descricao='Fermentação colônica; microbiota'),
        dict(nome='Ferro', tipo='MICRO', familia='MIN', unidade_padrao='mg',
descricao='Hemoglobina; transporte de O2'),
        dict(nome='Sódio', tipo='MICRO', familia='MIN', unidade_padrao='mg',
descricao='Balanço hídrico; PA'),
        dict(nome='Frutose', tipo='MACRO', familia='CARB',
unidade_padrao='g', descricao='Metabolismo hepático preferencial'),
        dict(nome='Proteína', tipo='MACRO', familia='PROT',
unidade_padrao='g', descricao='Aminoácidos essenciais para síntese
proteica'),
    ]
    for n in nutrientes:
        Nutriente.objects.get_or_create(nome=n['nome'], defaults=n)


def seed_alimentos_e_efeitos():
    dados = {
        'Cenoura': {
            'calorias_por_100g': 41,
            'efeito': 'Antioxidante para pele/retina',
            'composicao': [('Vitamina A (retinol)', 835.0, 'µg', 90)],
            'orgaos': [
                ('Olhos', 'PROTEGE', +2, 'Retinal/rhodopsina para
```

```
fotorrecepção'),
                ('Pele', 'PROTEGE', +1, 'Suporte a epitélio'),
                ('Fígado', 'MODULA', +1, 'Armazenamento de retinol'),
            ]
        },
        'Salmão': {
            'calorias_por_100g': 208,
            'efeito': 'Rico em ômega-3 (EPA/DHA)',
            'composicao': [('Ômega-3 (EPA/DHA)', 2.0, 'g', 95), ('Proteína',
20.0, 'g', 100)],
            'orgaos': [
                ('Cérebro', 'PROTEGE', +2, 'Fluidez de membrana; sinapses'),
                ('Coração', 'PROTEGE', +2, 'Anti-inflamatório;
triglicérides'),
            ]
        },
        'Maçã': {
            'calorias_por_100g': 52,
            'efeito': 'Fibra solúvel e polifenóis',
            'composicao': [('Fibra Solúvel', 2.4, 'g', 70)],
            'orgaos': [
                ('Intestino Grosso', 'MODULA', +2,
'Butirato por fermentação; microbiota'),
                ('Coração', 'PROTEGE', +1, 'Perfil lipídico via fibras'),
            ]
        },
        'Brócolis': {
            'calorias_por_100g': 34,
            'efeito': 'Indutores de enzimas de fase II',
            'composicao': [('Vitamina C (ác. ascórbico)', 89.0, 'mg', 90),
('Proteína', 2.8, 'g', 100)],
            'orgaos': [
                ('Fígado', 'ESTIMULA', +1, 'Detox (Nrf2/Glutationa)'),
                ('Baço', 'MODULA', +1, 'Suporte imune'),
            ]
        },
        'Mel': {
            'calorias_por_100g': 304,
            'efeito': 'Carboidrato simples (frutose > glicose)',
            'composicao': [('Frutose', 38.0, 'g', 100)],
            'orgaos': [
                ('Fígado', 'LESIONA', -1, 'Excesso crônico → lipogênese de
novo'),
            ]
        },
        'Sal': {
            'calorias_por_100g': 0,
            'efeito': 'Fonte de sódio',
            'composicao': [('Sódio', 38758.0, 'mg', 100)],
            'orgaos': [
                ('Rins', 'MODULA', +1, 'Homeostase hídrica'),
```

```
                ('Coração', 'LESIONA', -1, 'PA ↑ em suscetíveis (excesso)'),
            ]
        },
    }

    for nome, d in dados.items():
        alimento, created = Alimento.objects.get_or_create(
            nome=nome,
            defaults=dict(calorias_por_100g=d['calorias_por_100g'],
efeito=d.get('efeito'))
        )
        if not created:
            alimento.calorias_por_100g = d['calorias_por_100g']
            alimento.efeito = d.get('efeito')
            alimento.save()

        # composição
        for (nut_nome, qtd, un, bio) in d.get('composicao', []):
            try:
                nut = Nutriente.objects.get(nome=nut_nome)
                ComposicaoAlimento.objects.update_or_create(
                    alimento=alimento, nutriente=nut,
                    defaults=dict(quantidade_por_100g=qtd, unidade=un,
biodisponibilidade=bio)
                )
            except Nutriente.DoesNotExist:
                print(f"⚠️ Nutriente não encontrado: {nut_nome}")

        # efeitos em órgãos
        for (org_nome, t, intensidade, mecanismo) in d.get('orgaos', []):
            org = get_orgao(org_nome)
            if org:
                EfeitoAlimentoOrgao.objects.update_or_create(
                    alimento=alimento, orgao=org, tipo_efeito=t,
                    defaults=dict(intensidade=intensidade,
mecanismo=mecanismo)
                )


def run():
    seed_nutrientes()
    seed_alimentos_e_efeitos()
    print('✅ Nutrientes e alimentos importados com sucesso!')
```

7.2) `fisiologia/scripts/import_parametros_biomarcadores.py`

```
# fisiologia/scripts/import_parametros_biomarcadores.py
from fisiologia.models import ParametroFisiologico, Biomarcador,
ExameLaboratorial
from anatomia.models import Orgao
```

```
def org(nome):
    from anatomia.models import Orgao as O
    try:
        return O.objects.get(nome=nome)
    except O.DoesNotExist:
        print(f"⚠️ Órgão não encontrado: {nome}")
        return None


def run():
    mapa_param = [
        dict(nome='Frequência Cardíaca', orgao='Coração', tipo='SINAL',
unidade='bpm', referencia_min=60, referencia_max=100,
             descricao='Batimentos por minuto em repouso.'),
        dict(nome='Pressão Arterial Sistólica', orgao='Coração',
tipo='SINAL', unidade='mmHg', referencia_min=90, referencia_max=120),
        dict(nome='Glicemia de Jejum', orgao='Pâncreas', tipo='LAB',
unidade='mg/dL', referencia_min=70, referencia_max=99),
        dict(nome='Taxa de Filtração Glomerular (eTFG)', orgao='Rins',
tipo='FUNC', unidade='mL/min/1.73m²', referencia_min=60, referencia_max=999),
        dict(nome='ALT (TGP)', orgao='Fígado', tipo='LAB', unidade='U/L',
referencia_min=0, referencia_max=41),
        dict(nome='TSH', orgao='Tireoide', tipo='LAB', unidade='mIU/L',
referencia_min=0.27, referencia_max=4.2),
    ]

    for p in mapa_param:
        o = org(p['orgao'])
        if not o:
            continue
        ParametroFisiologico.objects.update_or_create(
            nome=p['nome'], orgao=o,
            defaults=dict(tipo=p['tipo'], unidade=p.get('unidade'),
referencia_min=p.get('referencia_min'),
                          referencia_max=p.get('referencia_max'),
descricao=p.get('descricao'))
        )

    biomarcadores = [
        dict(nome='Glicose', unidade='mg/dL', referencia_min=70,
referencia_max=99, orgao_relacionado='Pâncreas'),
        dict(nome='Creatinina', unidade='mg/dL', referencia_min=0.6,
referencia_max=1.3, orgao_relacionado='Rins'),
        dict(nome='ALT (TGP)', unidade='U/L', referencia_min=0,
referencia_max=41, orgao_relacionado='Fígado'),
        dict(nome='TSH', unidade='mIU/L', referencia_min=0.27,
referencia_max=4.2, orgao_relacionado='Tireoide'),
    ]
```

```
    for b in biomarcadores:
        o = org(b['orgao_relacionado']) if b.get('orgao_relacionado') else
None
        Biomarcador.objects.update_or_create(
            nome=b['nome'],
            defaults=dict(unidade=b.get('unidade'),
referencia_min=b.get('referencia_min'),
                          referencia_max=b.get('referencia_max'),
orgao_relacionado=o)
        )

    # exames básicos
    exames = {
        'Glicemia de Jejum': ['Glicose'],
        'Função Renal': ['Creatinina'],
        'Perfil Hepático': ['ALT (TGP)'],
        'Perfil Tireoidiano': ['TSH'],
    }
    for ex, biom_list in exames.items():
        exame, _ = ExameLaboratorial.objects.get_or_create(nome=ex)
        for b in biom_list:
            try:
                bio = Biomarcador.objects.get(nome=b)
                exame.biomarcadores.add(bio)
            except Biomarcador.DoesNotExist:
                print(f"⚠️ Biomarcador não encontrado: {b}")

    print('✅ Parâmetros, biomarcadores e exames importados!')
```

---

8) Serviço de insights — core/insights.py

Permite a Aura explicar o caminho *Alimento* → *Nutrientes* → *Órgãos* → *Parâmetros*.

```
# core/insights.py
from metabolismo.models import Alimento, ComposicaoAlimento
from fisiologia.models import ParametroFisiologico, EfeitoNutrienteParametro
from metabolismo.models import EfeitoAlimentoOrgao


def explicar_alimento(nome_alimento: str) -> str:
    try:
        alimento = Alimento.objects.get(nome__iexact=nome_alimento)
    except Alimento.DoesNotExist:
        return f"Não encontrei o alimento '{nome_alimento}'."

    partes = [f"Alimento: {alimento.nome} ({alimento.calorias_por_100g} kcal/
100g)"]
    if alimento.efeito:
```

```
        partes.append(f"Efeito geral: {alimento.efeito}.")

    # Nutrientes
    comp =
ComposicaoAlimento.objects.filter(alimento=alimento).select_related('nutriente')
    if comp.exists():
        partes.append("Nutrientes-chave:")
        for c in comp[:8]:
            partes.append(f" - {c.nutriente.nome}: {c.quantidade_por_100g}
{c.unidade}/100g (biodisp. {c.biodisponibilidade}%)")

    # Órgãos-alvo
    efeitos_org =
EfeitoAlimentoOrgao.objects.filter(alimento=alimento).select_related('orgao')
    if efeitos_org.exists():
        partes.append("Efeitos por órgão:")
        for e in efeitos_org:
            sinal = '↑' if e.intensidade > 0 else ('↓' if e.intensidade < 0
else '•')
            partes.append(f" - {e.orgao.nome}: {e.tipo_efeito} {sinal}
({e.intensidade}). {e.mecanismo or ''}")

    # Parâmetros influenciados (via nutrientes → parâmetros)
    efeitos_param =
EfeitoNutrienteParametro.objects.filter(nutriente__alimentos__alimento=alimento).select_relate
'nutriente').distinct()
    if efeitos_param.exists():
        partes.append("Parametros possivelmente afetados:")
        for ep in efeitos_param[:10]:
            partes.append(f" - {ep.parametro.nome}
({ep.parametro.orgao.nome}): {ep.tipo_efeito} ({ep.intensidade}) via
{ep.nutriente.nome}.")

    return "\n".join(partes)
```

---

9) Passos para migrar e popular

1. **Salvar os modelos** (acima) nos respectivos apps.
2. Rodar migrações:

```
python manage.py makemigrations anatomia metabolismo fisiologia
sistema_endocrino psicologia
python manage.py migrate
```

3. Popular **nutrientes/alimentos**:

```
python manage.py shell -c "from
metabolismo.scripts.import_nutrientes_e_alimentos import run; run()"
```

4. Popular **parâmetros/biomarcadores/exames**:

```
python manage.py shell -c "from
fisiologia.scripts.import_parametros_biomarcadores import run; run()"
```

5. Testar **insights** no shell:

```
python manage.py shell -c "from core.insights import explicar_alimento;
print(explicar_alimento('Cenoura'))"
```

---

10) Como a Aura pode usar isso (ideia rápida)

No `core/ai_service.py`, você pode chamar `explicar_alimento('Cenoura')` quando o usuário registra uma refeição com esse alimento, e incorporar o texto no prompt da Aura. Assim ela explica **o que o alimento faz no corpo**, com base **nos modelos e dados do seu próprio sistema**.

Resultado prático

Com esse pack você consegue visualizar, por exemplo: - **Salmão** → (Ômega-3) → **Cérebro/Coração** → **Parâmetros** (triglicerídeos, inflamação) - **Cenoura** → (Vitamina A) → **Olhos/Pele/Fígado** → **Parâmetros** (visão, integridade do epitélio) - **Sal/Sódio** → **Rins/Coração** → **PA, TFG**

E a estrutura está pronta para você incrementar com **doenças, sintomas, hormônios, emoções** e gerar painéis causais do tipo *"por que isso aconteceu?"*.

Se quiser, no próximo passo eu preparo **consultas Django prontas** para: - dado um **órgão**, listar *alimentos/nutrientes* mais protetores e mais lesivos; - dado um **sintoma**, inferir quais **parâmetros e órgãos** investigar primeiro; - dado um **hormônio**, ver **órgãos-alvo** e **parâmetros** modulados.