Escopo: **Demandas (org_jobs)** + **Detalhe da demanda** + **Lista (demandas/lista)** + **Ciclo de vida** + **Integração com leilões** + **Categorias (catálogo do site)**.
Não inclui pagamentos/assinaturas (fora do combinado).

---

1) Visão de negócio do Módulo de Demandas

O que é uma "Demanda" no Wokaly

Uma **Demanda** é a entidade que representa a necessidade de contratação/serviço da empresa (buyer). Ela é a base para:

- registrar o escopo (título, descrição, alcance geográfico, unidade/filial responsável)
- controlar governança interna (responsável / owner da demanda, status, prazos)
- habilitar um processo competitivo via **Leilão/Participação** (job_auction) quando a demanda estiver apta.

Em termos de produto:

- **Demanda** = "o quê precisamos e quem é responsável"
- **Leilão** = "como vamos receber propostas e selecionar fornecedor"

Por que esse módulo é central

Ele conecta:

- Operação diária do comprador (member/owner/admin)
- Controle organizacional por **unidade/filial (branch)** (multitenant + branch-level RBAC)
- Mercado (fornecedores) via leilões publicados no Mundo Wokaly
- Métricas/Histórico do comprador (dashboard "Empresa / Histórico")

---

2) Modelo de dados e tabelas envolvidas

2.1 public.org_jobs (Demandas)

Campos-chave (nível produto):

- id, org_id, branch_id
- job_code (ex.: JOB-000217)
- title, description
- category (**catálogo oficial** em código)
- target_scope (all_br | custom_states)
- target_states (array de UFs quando escopo custom)

- status (draft/published/in_evaluation/awarded/cancelled/closed)
- payment_term_days (15/20/25/30/45/50/60)
- budget_max (orçamento interno máximo da demanda)
- **Governança interna**:
    - opened_by_profile_id (quem abriu)
    - owner_profile_id (responsável atual)
- **Integração com leilão**:
    - is_biddable (apta para leilão)
    - budget_mode (none/reference_hidden/reference_visible)
    - has_active_auction, last_auction_id

2.2 public.job_auctions (Leilões vinculados à demanda)

- id, org_id, job_id
- status (draft/published/running/finished/cancelled/desert)
- budget_mode, reference_budget, starts_at, ends_at
- mundo_display_format (hero/compact/rich_list/simple_list/mini)
- first_published_at (evento canônico de publicação)
- **timestamps canônicos** de encerramento (implementados no projeto):
    - finished_at, cancelled_at, desert_at (via trigger)

2.3 public.job_auction_bids (Lances)

- auction_id, bidder_profile_id, amount, status (placed/shortlisted/accepted/rejected)
- created_at
- accepted é a fonte do “vencedor selecionado” pela empresa.

---

3) Segurança e governança (RBAC, branch-level e fail-safe)

3.1 Fonte de verdade do acesso

- Menu não garante segurança.
- O acesso real é garantido por **RPCs SECURITY DEFINER** + validações por get_current_org_membership() + regras branch-level.

3.2 Papéis

- owner / admin: acessam e editam demandas da org inteira (respeitando branch_id válido)

- member: acessa e edita demandas **somente** no escopo de filial em que atua:
    - na criação: p_branch_id obrigatório e precisa estar em org_branch_members
    - na edição: permissão baseada na filial atual da demanda (v_job.branch_id)

---

4) RPCs do Módulo de Demandas (back-end)

4.1 upsert_org_job_self(...) — RPC principal do módulo (criar/editar)

Função: cria ou atualiza uma demanda (org_jobs) sob o contexto da organização atual.

Responsabilidades por etapa (como está no SQL)

1) Membership e RBAC

- exige get_current_org_membership()
- permite owner/admin/member

2) Normalizações

- currency default 'R' (legado do seu DB — mantém)
- visibility default 'public'
- target_scope default 'all_br'
- converte 'states' → 'custom_states'
- status default 'draft'
- payment_term_days default 30

3) Validações

- title obrigatório
- description obrigatório
- status deve estar na lista permitida
- visibility deve estar na lista permitida
- target_scope deve estar na lista permitida
- payment_term_days deve estar em (15,20,25,30,45,50,60)

4) Carrega job quando edição

- garante que o job pertence à org

5) Branch-level RBAC

- owner/admin: valida se p_branch_id pertence à org
- member:

- novo: exige branch e membership naquela branch
- edição: exige membership na branch atual do job

6) Normalização de is_biddable e budget_mode

- define valores finais:
  - se não biddable → budget_mode = NULL
  - se biddable → budget_mode = coalesce(..., 'none')
- preserva valores do banco na edição quando front não manda

7) INSERT

- cria demanda
- grava opened_by_profile_id e owner_profile_id inicialmente como auth.uid()

8) UPDATE

- atualiza campos
- category = coalesce(p_category, j.category) (categoria opcional)

Ponto crítico: categoria validada por constraint

A RPC aceita p_category text, mas a tabela tem constraint (ver item 6).
Ou seja: **o banco decide o catálogo permitido**.

4.2 get_org_job_identity_self(p_job_id) — detalhe da demanda (front [id])

Função: retorna o "shape" completo para a página de detalhe da demanda, incluindo:

- campos base do job
- category, target_scope, target_states
- flags de leilão (is_biddable, budget_mode, has_active_auction, last_auction_id)
- identidades de criador e responsável.

4.3 list_org_jobs_identity_self() — lista de demandas (front /lista)

Função: retorna as demandas visíveis ao usuário (RBAC aplicado), com:

- job_code, title, branch_id, status, created_at
- flags adicionais: is_biddable, has_active_auction
- status do leilão vinculado (quando aplicável): job_auction_status / auction_status
- vitrine: mundo_display_format / auction_display_format (conforme RPC).

Essa lista é usada na página /dashboard/empresa/demandas/lista.

4.4 list_org_job_owner_candidates_self(p_job_id) + update_org_job_owner_self(p_job_id, p_new_owner_profile_id)

Função (produto): manter o responsável atual da demanda.

- O detalhe da demanda carrega candidatos reais para dono/responsável
- A troca é feita via RPC específica (RBAC no banco)

4.5 Integração com leilões a partir da demanda

create_org_job_auction_self(p_job_id)

Cria um leilão a partir da demanda, respeitando:

- governança da org
- regra de limitação e integridade (incluindo requisitos de orçamento/planejamento em outras camadas)

list_job_auctions_for_job_company_self(p_job_id)

Usada no JobLifecycleCard para exibir **histórico de leilões** vinculados à demanda.

---

5) Front-end do módulo de Demandas (arquivos e comportamento)

5.1 Lista de demandas

Arquivo: /app/dashboard/empresa/demandas/lista/page.tsx

Características:

- carrega dados via list_org_jobs_identity_self
- filtros em memória: busca por código/título/responsável, status e filial
- paginação local (pageSize / currentPage)
- mostra “badge” do estado do leilão vinculado e vitrine Mundo Wokaly

Vitrine Mundo Wokaly (na lista)

Helpers:

- normalizeMundoDisplayFormat
- getMundoDisplayFormatLabel
- getJobVitrineInfo

Resultado:

- exibe “Formato no Mundo Wokaly”
- badge “Exibido / Não exibido” conforme status do leilão (published/running).

5.2 Detalhe da demanda (editar)

Arquivo: /app/dashboard/empresa/demandas/[id]/page.tsx

Fluxo de UX:

- Carrega via get_org_job_identity_self
- Exibe dados em cards
- Botões:
    - “Editar demanda” → habilita edição
    - “Salvar alterações” → chama upsert_org_job_self + (se necessário) update_org_job_owner_self
    - “Cancelar edição” → reset de form via resetFormFromJob(job)

Branch change para member (regra de UX)

Se member mudar a filial:

- redireciona para /dashboard/empresa/demandas/lista (porque pode perder acesso ao próprio registro, de acordo com branch-level RBAC).

5.3 Ciclo de vida da demanda + leilões vinculados

Arquivo: /app/dashboard/empresa/demandas/[id]/JobLifecycleCard.tsx

Mostra:

- Status atual da demanda
- Se está apta para leilão (is_biddable)
- Resumo do vínculo com leilões:
    - existe leilão ativo?
    - existe leilão encerrado?
    - histórico de leilões da demanda via list_job_auctions_for_job_company_self

Regra importante implementada:

- Se já houve leilão finished para essa demanda, não cria novos leilões: “para novo processo competitivo, criar nova demanda”.

---

6) Catálogo oficial de categorias (produto + DB + front)

6.1 Lista de categorias

Foi criado/expandido o catálogo com base na landing, com códigos canônicos (snake_case) e labels PT-BR.

6.2 Front: contrato de categorias

Arquivos:

- /lib/central-leiloes/types.ts
    - JobCategory (union type com o catálogo completo)
    - JOB_CATEGORY_OPTIONS (na ordem do site)
- /lib/central-leiloes/format.ts
    - normalizeJobCategory(...) atualizado para suportar todas
    - labelForJobCategory(...) usa JOB_CATEGORY_OPTIONS

6.3 i18n

Chaves:

- jobs.detail.job_kind.<code> para cada categoria (source de verdade de label do sistema)

---

7) Banco: validação de categoria (constraints)

Problema encontrado

Havia **duas constraints** em org_jobs.category:

- org_jobs_category_chk (antiga, só 4 categorias)
- org_jobs_category_allowed_chk (nova, catálogo completo)

Como as duas coexistiam, a antiga bloqueava as categorias novas.

Solução aplicada

- Removida org_jobs_category_chk
- Mantida somente org_jobs_category_allowed_chk com o catálogo completo
- Validado via query de auditoria (e testado salvando categorias novas)

---

8) Busca por categoria no histórico (server-side, sem acento)

Para suportar busca por “Consultoria”/“Treinamentos” etc. de forma robusta:

Helper no banco

- public.job_category_label_ptbr(p_cat text) (immutable)
- Retorna label PT-BR para qualquer categoria do catálogo.

Uso

- RPC list_company_history_auctions_self usa:
    - j.category raw
    - job_category_label_ptbr(j.category) para label PT-BR
    - ambos normalizados com translate(lower(...)) para busca sem acentos

---

9) Regras de consistência e padrões do projeto (aplicados)

- **Menu ≠ segurança**: back sempre é "fonte final"
- **RBAC por org + branch** nas RPCs
- Mudanças incrementais, auditáveis, sem refatorões
- i18n com t("key") || "fallback"
- Separação clara:
    - Demanda: governança e escopo
    - Leilão: processo competitivo, status e operação
    - Histórico: relatórios e visão por eventos canônicos

---

10) O que foi entregue / estado atual (baseline)

✅ Módulo de Demandas está consistente com:

- catálogo completo do site
- DB validando categoria corretamente
- tela de detalhe com select completo e salvando
- lista mostrando vitrine/leilão e filtros
- integração com ciclo de vida e criação de leilão

---

11) Próximos passos recomendados

1. **Aplicar o mesmo catálogo completo** nas telas de "Nova demanda" (se houver um form separado do detalhe) para evitar divergência.
2. Garantir que qualquer RPC que filtre por categoria use o helper job_category_label_ptbr (evita "mapa duplicado" de labels).
3. Espelhar a mesma experiência de "lista padrão Wokaly" e filtros no lado fornecedor (onde o DataGrid também tende a virar BI).
4. Documentar o catálogo em /docs/db.md (fonte única + regra da constraint).

/docs/db.md — Catálogo de Categorias (org_jobs.category)

Este documento descreve o **catálogo oficial de categorias** do Wokaly e as regras de banco relacionadas ao campo `public.org_jobs.category`.

> Objetivo: garantir **consistência** entre Banco, RPCs e Front-end (UI/landing), evitando divergências e constraints duplicadas.

---

1) Campo `org_jobs.category`

- **Tabela:** `public.org_jobs`
- **Coluna:** `category` (`text`, opcional)
- **Semântica:** representa a categoria/tipo de serviço da demanda (buyer).
- **Regra:** pode ser `NULL` (não informado) ou um **código canônico** (`snake_case`) pertencente ao catálogo oficial.

1.1 Catálogo oficial (códigos)

Os valores válidos atuais são:

- `admin_support`
- `auditing`
- `bpo_operations`
- `consulting`
- `design_creative`
- `engineering_maintenance`
- `events_production`
- `finance`
- `quality_management`
- `headhunting`

- `legal_compliance`
- `cleaning_facilities`
- `logistics_supply`
- `digital_marketing`
- `project`
- `recruitment_selection`
- `human_resources`
- `writing_translation`
- `security_surveillance`
- `workplace_safety`
- `technology_data`
- `training`

> Nota: a **UI** deve exibir labels em PT-BR via i18n, mas o banco armazena o **código**.

---

2) Constraint de validação (fonte de verdade do banco)

2.1 Constraint única

O banco deve manter **apenas uma** constraint para validar o catálogo:

- **Constraint:** `org_jobs_category_allowed_chk`

> **Não manter** outras constraints similares (ex.: `org_jobs_category_chk`), pois causam bloqueios silenciosos e conflitos.

2.2 SQL (referência)

```sql
```

```
alter table public.org_jobs
add constraint org_jobs_category_allowed_chk
check (
  category is null
  or category in (
    'admin_support',
    'auditing',
    'bpo_operations',
    'consulting',
    'design_creative',
    'engineering_maintenance',
    'events_production',
    'finance',
    'quality_management',
    'headhunting',
    'legal_compliance',
    'cleaning_facilities',
    'logistics_supply',
    'digital_marketing',
    'project',
    'recruitment_selection',
    'human_resources',
    'writing_translation',
    'security_surveillance',
    'workplace_safety',
    'technology_data',
    'training'
  )
);
```

2.3 Auditoria rápida (garantia de duplicidade zero)

```
select conname, pg_get_constraintdef(oid) as definition
from pg_constraint
where conrelid = 'public.org_jobs'::regclass
  and contype = 'c'
  and pg_get_constraintdef(oid) ilike '%category%'
order by conname;
```

Resultado esperado: apenas org_jobs_category_allowed_chk.

3) Helper de label PT-BR (busca/relatórios)

Para permitir busca por termos em PT-BR (ex.: “Consultoria”, “Treinamentos”), existe o helper:

- **Função:** public.job_category_label_ptbr(p_cat text)
- **Propriedade:** IMMUTABLE
- **Uso:** filtros server-side (RPCs), buscas sem acento, relatórios.

3.1 SQL (referência)

```
create or replace function public.job_category_label_ptbr(p_cat text)
returns text
language plpgsql
immutable
set search_path to 'public', 'pg_temp'
as $$
declare
 v text := coalesce(lower(trim(p_cat)), '');
begin
 if v = '' then return ''; end if;

 v := replace(v, '-', '_');

 case v
  when 'admin_support' then return 'administrativo e suporte';
  when 'auditing' then return 'auditoria';
  when 'bpo_operations' then return 'bpo operacao';
  when 'consulting' then return 'consultoria';
  when 'design_creative' then return 'design grafico e criativo';
  when 'engineering_maintenance' then return 'engenharia e manutencao';
  when 'events_production' then return 'eventos e producao';
  when 'finance' then return 'financas';
  when 'quality_management' then return 'gestao da qualidade';
  when 'headhunting' then return 'headhunter';
```

```
    when 'legal_compliance' then return 'juridico e compliance';
    when 'cleaning_facilities' then return 'limpeza e facilities';
    when 'logistics_supply' then return 'logistica e suprimentos';
    when 'digital_marketing' then return 'marketing digital';
    when 'project' then return 'projetos';
    when 'recruitment_selection' then return 'recrutamento e selecao';
    when 'human_resources' then return 'recursos humanos';
    when 'writing_translation' then return 'redacao e traducao';
    when 'security_surveillance' then return 'seguranca e vigilancia';
    when 'workplace_safety' then return 'seguranca do trabalho';
    when 'technology_data' then return 'tecnologia e dados';
    when 'training' then return 'treinamentos';
    else return v;
  end case;
end;
$$;
```

3.2 Uso recomendado em buscas (sem acento)

Exemplo de normalização (remove acento):

```
translate(
 lower(public.job_category_label_ptbr(j.category::text)),
 'áàãâäéèêëíìîïóòõôöúùûüç',
 'aaaaaeeeeiiiiooooouuuuc'
)
```

---

4) Checklist para adicionar uma NOVA categoria (futuro)

Quando o catálogo crescer, seguir este fluxo **sem pular etapas**.

4.1 Banco (DB)

1. **Escolher código canônico**

- snake_case
- preferir inglês/neutral (não usar acentos)
- exemplo: customer_success, data_science, etc.

2. **Atualizar constraint do catálogo**

- alterar a lista do org_jobs_category_allowed_chk para incluir o novo código.
- garantir que **não existe** outra constraint concorrente.

3. **Atualizar helper de label PT-BR**

- adicionar when '<novo_code>' then return '<label pt-br normalizado>'.

4. **Auditar constraints**

```
select conname, pg_get_constraintdef(oid)
from pg_constraint
where conrelid='public.org_jobs'::regclass
 and contype='c'
 and pg_get_constraintdef(oid) ilike '%category%';
```

5. **Teste rápido**

- atualizar uma demanda existente para a nova categoria via front (ou SQL controlado)
- garantir que não há erro 23514 (check constraint).

4.2 Front-end (UI)

6. **Atualizar contrato do catálogo**

- arquivo: lib/central-leiloes/types.ts
    - adicionar no JobCategory (union)
    - adicionar em JOB_CATEGORY_OPTIONS (na ordem do site)

7. **Atualizar normalização**

- arquivo: lib/central-leiloes/format.ts
    - incluir o novo case em normalizeJobCategory

8. **Atualizar i18n**

- adicionar chave:
    - jobs.detail.job_kind.<novo_code> = <Label em PT-BR>

9. **Validar telas que usam categoria**

- Detalhe da demanda (select)
- Central de leilões (filtro por categoria)
- Histórico (lista/relatórios) — busca por label deve funcionar via helper

4.3 Busca/relatórios (RPCs)

10. **Garantir que RPCs que filtram por categoria usam:**

- j.category raw
- job_category_label_ptbr(j.category) para label PT-BR
- normalização translate(lower(...)) para busca sem acento

---

5) Padrões e guardrails

- Banco valida catálogo via **uma** constraint (fonte final).
- UI renderiza labels via i18n, mas armazena **código** no banco.
- Busca server-side deve incluir:
    - raw code
    - label PT-BR via helper
- Evitar duplicidade de constraints (causa falhas difíceis de rastrear).