■# Sistema de Gestão de Frotas – Especificação Técnica Integral

Versão: 1.0.0
Data: 2025-11-02
Fonte primária do código: `C:\Users\Geovane\Documents\Transpontual\sistema_gestão_frotas`
Destinatários: Equipe de engenharia, arquitetos de soluções, sistemas de IA generativa.

Sumário

Visão Geral
O sistema de Gestão de Frotas Transpontual provê aplicações web e mobile para controle de veíc
motoristas, checklists veiculares, serviços de manutenção e fornecedores. A solução é composta

- **API REST (FastAPI)** responsável pelas regras de negócio, persistência PostgreSQL e auten
via JWT/SSO.
- **Dashboard administrativo (Flask + Bootstrap)** para times operacionais, com módulos de
checklist, motoristas, veículos, fornecedores, ordens de serviço e relatórios.
- **PWA Mobile** (HTML/JS) focado em checklists offline-first para motoristas.
- **Ferramentas auxiliares** (scripts, ETLs, seeds SQL, templates compartilhados).

Todo o ecossistema foi desenvolvido em Python, com suporte oficialmente documentado para execu
via Docker Compose, Render e Railway.

Capacidades de Negócio
Domínio	Descrição	Componentes principais
Autenticação & Autorização	Login com JWT e suporte a perfis (admin, gestor, operador)
`backend_fastapi/app/api_v1.py`, `app/core/security.py`, modelos `Usuario`, `UsuarioPerfil`.
Gestão de Frota	Cadastro de veículos, motoristas, fornecedores, ordens de serviço e co
de manutenção.	Modelos `Veiculo`, `Motorista`, `Fornecedor`, `OrdemServico`; templates Flask
`templates/vehicles`, `motoristas`, `fornecedores`, `maintenance`.
Checklist Veicular	Modelos de checklist, itens, execução, respostas e monitoramento de
status/avarias.	Rotas FastAPI `/api/v1/checklist`, service `checklist_service.py`, PWA mobil
telas Flask `templates/checklists`.
Relatórios & KPIs	Painel com controles de aprovação, filtros, DataTables com exportaçã
CSV/PDF, gráficos (Chart.js/Plotly).	`flask_dashboard/app/templates/dashboard*.html`,
`reports/*.html`, endpoints `/reports/*`, rota FastAPI `/checklist/kpis`.
Integração & ETL	Scripts SQL (`sql/*.sql`), jobs de manutenção (`etl_jobs/` placeholde
exportação de dados e proxies Nginx.	`scripts/`, `nginx/`, `etl_jobs/`, `shared_templates/`
Mobile Offline	Aplicativo PWA com cache, manifest e service worker para execução de
checklists sem conectividade constante.	`mobile_pwa/index.html`, `manifest.json`, `service-
worker.js`.

Arquitetura do Sistema

Componentes e Responsabilidades
Componente	Tecnologia	Responsabilidade	Dependências
FastAPI Backend (`backend_fastapi/app`)	Python 3.11+, FastAPI, SQLAlchemy, Pydantic	REST, regras de negócio, segurança, upload de mídia.	PostgreSQL, storage local (`uploads/`)
Flask Dashboard (`flask_dashboard/app`)	Flask, Bootstrap 5, Jinja2, DataTables, Chart Plotly	UI administrativa responsiva, relatórios, filtros, exports.	API FastAPI (`API_BASE` autenticação JWT (Flask session).
PWA Checklist (`mobile_pwa/`)	HTML5, CSS, Vanilla JS, Service Worker	Execução de checklists em campo, modo offline, sincronização via API.	API FastAPI (REST), LocalStorage, Geolocation.
ETL/Jobs (`etl_jobs/`)	Python scripts (placeholder)	Automação de agregações, manute dados.	PostgreSQL.
Nginx configs (`nginx/`)	Nginx reverse proxy	Publicação composta (API + Dashboard), TLS/headers.	Docker, containers backend/frontend.
Seeds & SQL (`sql/`, `seeds/`)	SQL scripts, Python seeding	Inicialização de schema, básicos, correções.	PostgreSQL.

Fluxo Lógico
1. Usuário autentica no backend (`/api/v1/auth/login`), recebe JWT.
2. Dashboard/PWA adicionam `Authorization: Bearer` em chamadas subsequentes.
3. Operações CRUD manipulam entidades (veículos, motoristas, checklists etc.) persistidas em PostgreSQL.
4. Dashboard consome endpoints REST via `requests` (server-to-server). PWA consome diretamente fetch.
5. Uploads de imagens/checklists são gravados em `uploads/` (configurável via `STORAGE_DIR`).
6. Opcionalmente, Nginx unifica hostnames e aplica TLS; Render/Gunicorn fazem o deploy server

Deploy Topologias
Ambiente	Stack	Observações
Dev local	Docker Compose (`docker-compose.yml`) sobe Postgres, FastAPI (uvicorn), Flas Dashboard (gunicorn) e worker opcional.	Porta padrão API 8005, dashboard 8050, PWA servido manualmente.
Render	`render.yaml` executa `gunicorn app:app` com `PYTHONPATH=/opt/render/project/s	Ajustar ordem de `PYTHONPATH` para priorizar `app.py`.
Railway	Procfiles separados (`Procfile.backend`, `Procfile.frontend`).	`railway*.js definem variáveis, builds análogos ao Render.

Ambientes e Configuração
Variáveis principais (`.env`)
Variável	Descrição	Default
`DATABASE_URL`	URL PostgreSQL (Supabase ou local).	`postgresql://...`
`JWT_SECRET`, `JWT_EXPIRES_MINUTES`	Segredo e expiração do token.	`dev-jwt-secret-chang production`, `1440`
`STORAGE_DIR`	Diretório para uploads.	`./uploads`
`API_PORT`, `PORT`	Porta FastAPI (local).	`8005`
`FLASK_SECRET_KEY`	Chave de sessão Flask.	`dev-secret-key`
`API_BASE`	Base URL consumida pela dashboard/PWA.	`http://localhost:8005`
Credenciais seed	`API_LOGIN_EMAIL=admin@transpontual.com`, `API_LOGIN_PASSWORD=123456`	

Variáveis para Render (`render.yaml`)
- `ENV`, `DEBUG`, `LOG_LEVEL`, `PORT` (10000), `API_BASE=http://127.0.0.1:8005`, `ALLOWED_ORI
JSON, `PYTHONPATH=/opt/render/project/src`.
- Ajustar `PYTHONPATH` para priorizar o módulo `app.py` (dashboard) quando usar Gunicorn.

Dependências
- **Backend**: FastAPI, SQLAlchemy 2.x, Pydantic, Passlib/Bcrypt, python-jose, Uvicorn, Reques
Pillow, ReportLab (exports), Pandas, OpenPyXL.
- **Dashboard**: Flask 3.0+, requests, DataTables, Bootstrap 5, Chart.js, Plotly, Font Awesom
Bootstrap Icons.
- **Mobile PWA**: Sem build (HTML/CSS/JS puros).

Backend FastAPI
Stack
- Estrutura modular em `backend_fastapi/app`, com `api_v1.py` agregando rotas, `routers/` para
domínios (`checklist.py`, `fornecedores.py`), `models.py` com SQLAlchemy declarativo.
- Middlewares: CORS (`app/main.py`), logging de request (`X-Process-Time`), handlers 404/500
personalizados.
- Autenticação: `POST /api/v1/auth/login` gera JWT, `GET /api/v1/auth/me` retorna usuário
autenticado. `app/core/security.py` lida com bcrypt, SSO fallback e criação de tokens.

Módulos Funcionais (Principais Endpoints)
Módulo	Endpoints chave (prefix `/api/v1`)	Observações
Auth & Users	`POST /auth/login`, `GET /auth/me`, `POST /users` (seed/test via scripts
Suporta modo offline (demo) se DB indisponível.
Checklists (`routers/checklist.py`)	`GET /checklist`, `POST /checklist/start`, `POST
/checklist/answer`, `POST /checklist/finish`, `GET /checklist/{id}`, `GET /checklist/modelos`
/checklist/modelos`, `POST /checklist/modelos/{id}/itens`.	Controle completo de ciclo (pré/
viagem), respostas com severidade, upload opcional de evidências.
Veículos	`GET /vehicles`, `POST /vehicles`, `PUT /vehicles/{id}`, `GET /vehicles/{id}
(definidos em `api_v1.py`).	Modelo `Veiculo` com campos técnicos (placa, renavam, tipo,
capacidade).
Motoristas	`GET /drivers`, `POST /drivers`, `PUT /drivers/{id}`.	Campos: CNH, valida
categoria, status ativo.
Fornecedores (`routers/fornecedores.py`)	CRUD completo (`/fornecedores`).	Usado por
de manutenção.
Ordens de Serviço	`GET /service-orders`, `POST /service-orders`, `PATCH /service-
orders/{id}`, `POST /service-orders/{id}/items`.	Modelo `OrdemServico` com itens
(`OrdemServicoItem`).
Relatórios / KPIs	`GET /checklist/kpis`, `GET /checklist/stats/summary`, `GET
/checklist/download/{id}` (PDF/CSV).	Retorna agregações para dashboards.
Saúde	`GET /health`, `GET /debug/db/info`.	Utilizados por Render e dashboard.

Modelagem de Dados (principais tabelas)
- `users`: dados de usuário, permissões e status.
- `motoristas`, `veiculos`, `fornecedores`: cadastros operacionais.
- `checklist_modelos`, `checklist_itens`, `checklist`, `checklist_respostas`: estrutura e exec
de checklists.
- `ordens_servico`, `ordem_servico_itens`: manutenção corretiva/preventiva.
- `logs_acesso`, `sessoes_usuarios`, `usuario_permissoes`, `perfis_acesso`, `usuario_perfis`:
auditoria e segurança.
- Scripts SQL em `sql/ddl.sql` e `sql/seed.sql` definem campos, índices e dados iniciais (cons
para lista completa).

Regras e Validações
- Checklist: `odometro_fim >= odometro_ini`, itens obrigatórios (ordem, severidade), avaria po
bloquear viagem.
- Ordens de serviço: `numero_os` auto-gerado (`OS{yyyyMMdd}{uuid8}`) se não informado.
- Fornecedores: campos opcionais removidos antes de persistir (limpeza).

- Autenticação: bcrypt hash; modo DEV aceita senhas em texto para seeds.

Armazenamento e Uploads
- Arquivos no diretório `uploads/` (configurável). Service `storage.py` define utilitários.
- `STORAGE_DIR` deve estar acessível para API e dashboard (quando rodando no mesmo host/conta

Testes
- `backend_fastapi/tests/` contém testes PyTest (fixtures em `tests/conftest.py`) cobrindo autenticação e checklist básico.
- `scripts/test_*` e `test_*.py` complementam testes manuais/integração.

Dashboard Flask
Stack e Layout
- Base em `flask_dashboard/app/dashboard.py` + `routes.py`.
- Template raiz `templates/base.html`: barra superior, offcanvas lateral, footer com status d conexão.
- Ferramentas: Bootstrap 5.3, Bootstrap Icons, Font Awesome 6, DataTables (Buttons, Responsiv Chart.js, Plotly, jQuery.
- Autenticação: decorators em `auth_decorators.py` garantem roles (admin, gestor, operador).

Paleta de Cores & Design Tokens
Nome	Valor principal
Primária (navbar/botões)	`#0d47a1` (`--bs-primary`)
Sucesso	`#1b5e20`
Perigo	`#c62828`
Alerta	`#ef6c00`
Background claro	`#f8f9fa` (gradientes usados em cards)
HSL tokens avançados	Definidos em `static/styles.css` (modo claro/escuro).
Tipografia	Stack Bootstrap: `-apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, "Helv Neue", Arial, sans-serif`.

Páginas/Módulos (principais templates)
Rota Flask	Template	Função	Exportações
`/`	`dashboard.html`	KPIs de checklist, status da API, gráficos Plotly.	CSV/PDF via r `/reports/*`.
`/checklists`	`checklists/list.html`	DataTable com filtros, status badges e botões de a Exportar CSV (`/reports/checklists/csv`), PDF (`/reports/checklists/pdf`).
`/checklists/<id>`	`checklists/detail.html`	Detalhes, timeline, exportar relatório indi (`/reports/checklist/{id}/pdf`).	PDF.
`/drivers`, `/vehicles`, `/fornecedores`	`drivers/list.html`, `vehicles/list.html`, `fornecedores/list.html`	CRUD com formulários modais e DataTables.	CSV (via DataTables Bu
`/reports/*`	`reports/*.html`	Relatórios (abastecimentos, ordens de serviço).	Exporta CSV, XLSX e PDF (dependendo do módulo).
`/maintenance`	`maintenance/*.html`	Painéis de OS, peças, agendamentos preventivos.	Exportações CSV / Excel.
`/settings/users`	`users/list.html`	Gestão de contas e permissões.	CSV (DataTables).

Componentes UI
- **Cards**: `checklist-card`, `stat-card` com gradientes; hover states (`transform translate
- **Badges**: `status-*`, `sev-*` definem cores e gradientes.
- **Tabelas**: DataTables com `responsive` + `buttons` (CSV, Excel, Print, PDF). Configuraçõe definidas em JS de cada tela.
- **Modais**: formularios `form-control` com bordas `--input`.
- **Exports**: Botões com `bi-filetype-csv`, `bi-file-earmark-pdf`. Export runners usam `window.open` para endpoints de relatório.

JS utilitários (resumo)
- `static/js/app.js`: setup Axios (JWT via cookies), alertas, loading overlay.
- `static/js/calculadora.js`, `csc_viagens.js`, `csc_dashboard.js`: módulos específicos com integrações AJAX.
- `refreshData()` global usado para auto-refresh do template base (disparado a cada 5 min).

PWA Mobile
Stack & Organização
- Arquivos raiz em `mobile_pwa/`: `index.html`, `manifest.json`, `service-worker.js`, `sw.js`
- Design mobile-first com variáveis CSS (`--primary`, `--bg`, `--muted`). Tipografia: `system
- Service worker implementa cache (`sw.js`) e fallback offline.
- Manifest define nome, ícones, tema (`#0d6efd`).

Fluxos de Uso
1. Motorista informa API base (persistida em `localStorage`), veículo, motorista e modelo de checklist.
2. App tenta obter coordenadas via Geolocation API (`geoInit`).
3. `btnStart` chama `POST /checklist/start`, renderiza itens dinamicamente (chips para avarias radio buttons para status).
4. Cada resposta dispara `POST /checklist/answer`; finalização chama `POST /checklist/finish`
5. Toasts (`.toast`) dão feedback; barra de progresso calcula percentual de itens respondidos
6. Footer fixo com botão “Finalizar” e campo de odômetro final.

Estilo
- Cartões com bordas arredondadas (14px), sombra leve.
- Botões com `border-radius: 10px`, `background: var(--primary)`.
- Chips para avarias mudam para `background:#ffe3e3` quando ativos.
- Modo offline: avisos no header e mensagens toast.

Dados, Relatórios e Exportações
Exportações Frontend
- **Checklists**: CSV/PDF via `exportData(format)` (abre `/reports/checklists/{format}?filters
- **Checklist individual**: `exportChecklist(id)` ⇒ `/reports/checklist/{id}/pdf`.
- **Relatórios de abastecimentos, OS**: dropdowns de exportação (`reports/abastecimentos.html `reports/service_orders.html`) com CSV/Excel/PDF.
- **DataTables Buttons**: Em `list.html` (checklists), `drivers/list.html` e outros, DataTabl carrega recursos `buttons.bootstrap5.min.css/js` para exportar CSV/Excel/Print.

Relatórios Backend
- PDFs gerados via ReportLab (instalado) ou Jinja + WeasyPrint (dependendo do módulo).
- CSVs/Pandas: conversão em endpoints `/reports/*` utilizando Pandas/OpenPyXL conforme necessi
- KPI endpoints alimentam dashboards (taxa de aprovação, totais).

Banco de Dados
- DDL completo em `sql/ddl.sql` inclui chaves estrangeiras, índices e views.
- Seeds (`sql/seed.sql`, `maintenance_seed.sql`) populam dados de teste (usuários, veículos, checklists).
- Scripts `scripts/*.py` tratam migrações pontuais (`fix_sequence.py`, `create_admin.py`).

Integrações Externas
- **Supabase**: conector PostgreSQL (hosts `aws-0-sa-east-1.pooler.supabase.com`). Ajustar `DATABASE_URL` e `sslmode=require`.
- **Render/Railway**: Deploy serverless com Gunicorn (`app:app`). Necessário garantir import de `app.py` e `dashboard_app.py`.

- **SSO/Transpontual Auth**: módulo `transpontual_auth/` (no backend) para integrações futuras
SSO corporativo.
- **Plotly/Chart.js CDN**: dashboards consomem libs via CDN (recomenda-se pin de versão e fal
local em ambientes restritos).

Requisitos Não Funcionais
Categoria	Detalhes
Segurança	JWT assinado (`HS256`), força de senha via bcrypt, logs de acesso (`logs_ac
controle de sessões (`sessoes_usuarios`). Limitar upload a tipos esperados, sanitização de in
Disponibilidade	Endpoints `/health` e `/api/csc/viagens` com retries no dashboard.
Render/Gunicorn com 2 workers, 4 threads (config em `render.yaml`).
Performance	Cabeçalho `X-Process-Time`, DataTables com paginação server-side (impleme
Projeção de < 200ms para endpoints críticos em rede interna.
Escalabilidade	API stateless; usar Postgres escalável (Supabase). Separar storage de
(S3) em produção.
Observabilidade	Logging `logging.basicConfig(level=INFO)`, prints formatados. Recomen
integrar com Prometheus e Sentry futuramente.
Acessibilidade	Contraste elevado nas badges, botões com largura mínima 32px (mobile).
Recomendado adicionar aria-labels nos ícones e checar WCAG 2.1 AA.

Build, Deploy e Operação
Build Local
```bash
cp .env.example .env         # ajustar variáveis
docker compose up -d --build # sobe postgres + api + dashboard
# API docs: http://localhost:8005/docs
# Dashboard: http://localhost:8050/
```

Execução Manual (sem Docker)
```bash
# Backend
cd backend_fastapi
pip install -r requirements.txt
uvicorn app.main:app --reload --port 8005

# Dashboard
cd ../flask_dashboard
pip install -r requirements.txt
python run.py  # roda flask com debug na porta 8050
```

Deploy Render
1. Subir repositório para Git.
2. Configurar serviço web com build `pip install ... && pip install gunicorn`.
3. `startCommand: gunicorn app:app --workers 2 --threads 4`. Ajustar `PYTHONPATH` e variáveis
ambiente conforme `render.yaml`.
4. Garantir diretório `uploads/` persistente (Render Disk).

Deploy Railway
1. Projetos distintos para backend (`Procfile.backend`) e dashboard (`Procfile.frontend`).
2. `requirements-dashboard.txt` e `requirements.txt` (backend).
3. `railway*.json` definem variáveis (DATABASE_URL, JWT_SECRET etc.).

Tarefas Operacionais

- Seeds: `python scripts/apply_sql.py` (executa `sql/seed.sql`).
- Migrações manuais: executar scripts em `sql/maintenance_system.sql`, `scripts/fix_*`.
- Jobs ETL: agendar scripts em `etl_jobs/` (placeholder) para rodar em cron/K8s.

Apêndices
A. Paleta de Cores (Dashboard)
Token	HSL	Hex aproximado	Uso
`--primary`	`222.2° 47.4% 11.2%`	`#0d1b3b`	Navbar/botões principais
`--primary-foreground`	`210° 40% 98%`	`#f8fbff`	Texto em botões primários
`--secondary`	`210° 40% 96.1%`	`#e9eff8`	Cards secundários/fundos
`--muted-foreground`	`215.4° 16.3% 46.9%`	`#6c7a8d`	Textos auxiliares
`--destructive`	`0° 84.2% 60.2%`	`#f05454`	Alertas críticos
Badges severidade	Gradientes `#ef4444 → #dc2626`, `#f59e0b → #fbbf24`, `#3b82f6 → #60a5
Status de checklist

B. Fontes
- Base: família Bootstrap (`-apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, "Helvetica
Arial, sans-serif`).
- PWA: `system-ui, -apple-system, Segoe UI, Roboto, Ubuntu`.
- Recomendações: usar peso 400/500 no body, 600+ em cabeçalhos.

C. Estrutura de Pastas (síntese)
```
backend_fastapi/
  app/
    api_v1.py
    core/ (config, database, security)
    routers/ (checklist, fornecedores)
    models.py, schemas.py
    services/ (checklist_service, ...)
flask_dashboard/
  app/
    dashboard.py, routes.py
    templates/ (...)
    static/styles.css, js/*
mobile_pwa/
sql/
scripts/
nginx/
docs/
```

D. Checklist de Configuração Inicial
1. Definir `DATABASE_URL`, `JWT_SECRET`, `FLASK_SECRET_KEY`.
2. Executar seeds/metadados (`scripts/apply_sql.py`).
3. Criar usuário admin via `python backend_fastapi/app/create_admin.py`.
4. Ajustar `API_BASE` no `.env` do dashboard e PWA (via UI).
5. Configurar storage compartilhado para uploads.
6. Validar endpoints de exportação (CSV/PDF) e geração de relatórios.
7. Habilitar monitoramento (logs, health checks) no provedor de deploy.

> **Observação**: Esta especificação descreve o estado atual do repositório e serve como cont
para reconstrução do sistema por uma IA ou equipe de engenharia. Recomenda-se manter o docume
sincronizado com alterações futuras (módulos, esquemas, paleta de cores, dependências).