Ниже — полный набор файлов для монорепозитория `dispatcher-monorepo/` без Docker Compose. Скопируйте структуру и содержимое 1:1.

`/README.md`

````
# Dispatcher Monorepo

Frontend: Next.js 14 (Vercel-ready)
Backend: FastAPI (Railway/Timeweb-ready)
DB: PostgreSQL (TimewebCloud)

## Быстрый старт
1) Примените SQL в БД: `backend/db.sql` (схема), затем `backend/extras.sql`
(триггеры/поиск).
2) Backend:
   ```bash
   cd backend
   pip install -r requirements.txt
   # Windows: setx DATABASE_URL "postgresql://user:pass@host:5432/dispatcher"
   # Linux/Mac: export DATABASE_URL="..."
   uvicorn main:app --host 0.0.0.0 --port 8000
   ```
3) Frontend:
   ```bash
   cd frontend
   cp .env.example .env.local  # укажите NEXT_PUBLIC_API_URL
   npm install
   npm run dev
   ```

## Деплой
- Backend → Railway: корень = `backend/`, переменные: `DATABASE_URL`,
`CORS_ORIGINS`.
- Frontend → Vercel: корень = `frontend/`, переменная: `NEXT_PUBLIC_API_URL`
(URL бэка).
````

`/.gitignore`

```
# node/next
node_modules/
.next/
out/
*.log

# python
```

```
__pycache__/
.venv/
.env

# misc
.DS_Store
```

/.env.example

```
# Backend DB
DATABASE_URL=postgresql://dispatcher:password@db-host:5432/dispatcher
# Разрешённые источники для CORS (через запятую)
CORS_ORIGINS=https://your-frontend.vercel.app,http://localhost:3000

# Frontend API URL (пример)
NEXT_PUBLIC_API_URL=https://your-backend.up.railway.app
```

/backend/requirements.txt

```
fastapi==0.111.0
uvicorn[standard]==0.30.0
psycopg2-binary==2.9.9
```

/backend/Procfile

```
web: uvicorn main:app --host 0.0.0.0 --port $PORT
```

/backend/README.md

````
# Backend (FastAPI)

## Запуск
```bash
pip install -r requirements.txt
# Windows
setx DATABASE_URL "postgresql://dispatcher:password@db-host:5432/dispatcher"
setx CORS_ORIGINS "https://your-frontend.vercel.app,http://localhost:3000"
````
````

```
# Новый PowerShell
uvicorn main:app --host 0.0.0.0 --port 8000
```

Деплой Railway

- Root Directory: `backend/`
- Variables: `DATABASE_URL`, `CORS_ORIGINS`
- Procfile уже добавлен.

SQL

1) `db.sql` — базовая схема.
2) `extras.sql` — триггеры, аудит, поиск.

```
---
## `/backend/main.py`
```python
import os, json
from typing import Optional
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
import psycopg2
from psycopg2.extras import RealDictCursor
from pydantic import BaseModel

DATABASE_URL = os.getenv("DATABASE_URL")
CORS_ORIGINS = os.getenv("CORS_ORIGINS", "http://localhost:3000").split(",")

app = FastAPI(title="Dispatcher API", version="1.0.0")

app.add_middleware(
    CORSMiddleware,
    allow_origins=[o.strip() for o in CORS_ORIGINS if o.strip()],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

def get_conn():
    if not DATABASE_URL:
        raise RuntimeError("DATABASE_URL is not set")
    return psycopg2.connect(DATABASE_URL, cursor_factory=RealDictCursor)

class SearchParams(BaseModel):
    from_city: int
    to_city: int
    mode: str = "exact"  # exact | partial

@app.get("/health")
```
```

```
def health():
    return {"status": "ok"}

@app.get("/cities")
def cities(q: str):
    sql = """
      SELECT city_id, name_display
      FROM cities
      WHERE name_norm ILIKE %(p)s OR name_display ILIKE %(p)s
      ORDER BY name_display
      LIMIT 20
    """
    with get_conn() as c, c.cursor() as cur:
        cur.execute(sql, {"p": f"%{q.lower()}%"})
        return cur.fetchall()

@app.post("/search")
def search(params: SearchParams):
    if params.mode not in ("exact", "partial"):
        raise HTTPException(status_code=400, detail="mode must be exact or
partial")
    with get_conn() as c, c.cursor() as cur:
        if params.mode == "exact":
            cur.execute(
                "SELECT * FROM search_performers_exact(%(a)s,%(b)s)",
                {"a": params.from_city, "b": params.to_city},
            )
        else:
            cur.execute(
                "SELECT * FROM search_performers_partial(%(a)s,%(b)s)",
                {"a": params.from_city, "b": params.to_city},
            )
        return cur.fetchall()

@app.get("/deals")
def deals(limit: int = 100, offset: int = 0, performer_id:
Optional[int]=None):
    q = "SELECT * FROM deals ORDER BY deal_id DESC LIMIT %(l)s OFFSET %(o)s"
    args = {"l": limit, "o": offset}
    if performer_id:
        q = "SELECT * FROM deals WHERE performer_id=%(pid)s ORDER BY deal_id
DESC LIMIT %(l)s OFFSET %(o)s"
        args["pid"] = performer_id
    with get_conn() as c, c.cursor() as cur:
        cur.execute(q, args)
        return cur.fetchall()

class DealPatch(BaseModel):
    status: Optional[str] = None
    cost_rub: Optional[float] = None
    payload: Optional[dict] = None
```

```
@app.patch("/deals/{deal_id}")
def update_deal(deal_id: int, body: DealPatch):
    sets, params = [], {"id": deal_id}
    if body.status is not None:
        sets.append("status=%(status)s")
        params["status"] = body.status
    if body.cost_rub is not None:
        sets.append("cost_rub=%(cost)s")
        params["cost"] = body.cost_rub
    if body.payload is not None:
        sets.append("payload=%(payload)s")
        params["payload"] = json.dumps(body.payload)
    if not sets:
        raise HTTPException(400, "Nothing to update")
    sql = f"UPDATE deals SET {', '.join(sets)} WHERE deal_id=%(id)s RETURNING
*"
    with get_conn() as c, c.cursor() as cur:
        cur.execute(sql, params)
        row = cur.fetchone()
        c.commit()
        return row

# === Дополнительные CRUD (минимальные) ===
from fastapi import Body

class Performer(BaseModel):
    fio: str
    phone_norm: str
    geo_zone: Optional[str] = ""
    note: Optional[str] = ""

@app.get("/performers")
def list_performers(query: Optional[str]=None, limit: int=50, offset: int=0):
    where = ""
    params = {"l": limit, "o": offset}
    if query:
        where = "WHERE fio ILIKE %(q)s OR phone_norm ILIKE %(q)s"
        params["q"] = f"%{query}%"
    sql = f"SELECT * FROM performers {where} ORDER BY fio LIMIT %(l)s OFFSET
%(o)s"
    with get_conn() as c, c.cursor() as cur:
        cur.execute(sql, params)
        return cur.fetchall()

@app.post("/performers")
def create_performer(p: Performer):
    sql = """
      INSERT INTO performers(fio, phone_norm, geo_zone, note)
      VALUES (%(fio)s, %(phone)s, %(geo)s, %(note)s)
      RETURNING *
```

```
    """
    with get_conn() as c, c.cursor() as cur:
        cur.execute(sql, {"fio": p.fio, "phone": p.phone_norm, "geo":
p.geo_zone or "", "note": p.note or ""})
        row = cur.fetchone()
        c.commit()
        return row

@app.put("/performers/{performer_id}")
def update_performer(performer_id: int, p: Performer):
    sql = """
      UPDATE performers
      SET fio=%(fio)s, phone_norm=%(phone)s, geo_zone=%(geo)s, note=%(note)s
      WHERE performer_id=%(id)s
      RETURNING *
    """
    with get_conn() as c, c.cursor() as cur:
        cur.execute(sql, {"fio": p.fio, "phone": p.phone_norm, "geo":
p.geo_zone or "", "note": p.note or "", "id": performer_id})
        row = cur.fetchone()
        c.commit()
        return row

class RouteVariant(BaseModel):
    name: Optional[str] = ""
    stops: list[int]

@app.get("/route-variants")
def list_route_variants(limit: int=50, offset: int=0):
    sql = "SELECT * FROM route_variants ORDER BY variant_id DESC LIMIT %(l)s
OFFSET %(o)s"
    with get_conn() as c, c.cursor() as cur:
        cur.execute(sql, {"l": limit, "o": offset})
        return cur.fetchall()

@app.post("/route-variants")
def create_route_variant(rv: RouteVariant):
    sql = """
      INSERT INTO route_variants(name, stops)
      VALUES (%(name)s, %(stops)s)
      RETURNING *
    """
    with get_conn() as c, c.cursor() as cur:
        cur.execute(sql, {"name": rv.name or "", "stops": rv.stops})
        row = cur.fetchone()
        # пересоберём позиции
        cur.execute("SELECT rebuild_variant_positions(%s)",
(row["variant_id"],))
        c.commit()
        return row
```

```
@app.put("/route-variants/{variant_id}")
def update_route_variant(variant_id: int, rv: RouteVariant):
    sql = """
      UPDATE route_variants SET name=%(name)s, stops=%(stops)s WHERE
variant_id=%(id)s RETURNING *
    """
    with get_conn() as c, c.cursor() as cur:
        cur.execute(sql, {"name": rv.name or "", "stops": rv.stops, "id":
variant_id})
        row = cur.fetchone()
        cur.execute("SELECT rebuild_variant_positions(%s)", (variant_id,))
        c.commit()
        return row

@app.post("/performers/{performer_id}/variants")
def attach_variant(performer_id: int, body: dict = Body(...)):
    variant_id = int(body.get("variant_id"))
    with get_conn() as c, c.cursor() as cur:
        cur.execute(
            "INSERT INTO performer_variants(performer_id, variant_id) VALUES
(%s,%s) ON CONFLICT DO NOTHING",
            (performer_id, variant_id),
        )
        c.commit()
        return {"ok": True}

@app.delete("/performers/{performer_id}/variants/{variant_id}")
def detach_variant(performer_id: int, variant_id: int):
    with get_conn() as c, c.cursor() as cur:
        cur.execute(
            "DELETE FROM performer_variants WHERE performer_id=%s AND
variant_id=%s",
            (performer_id, variant_id),
        )
        c.commit()
        return {"ok": True}
```

/backend/db.sql

```
-- Расширения
CREATE EXTENSION IF NOT EXISTS pg_trgm;
CREATE EXTENSION IF NOT EXISTS btree_gin;

-- Города
CREATE TABLE IF NOT EXISTS cities (
  city_id      SERIAL PRIMARY KEY,
  name_norm    TEXT UNIQUE NOT NULL,
  name_display TEXT NOT NULL
```

```
);

-- Псевдонимы городов
CREATE TABLE IF NOT EXISTS city_aliases (
  alias_norm TEXT PRIMARY KEY,
  city_id    INT NOT NULL REFERENCES cities(city_id) ON DELETE CASCADE
);
CREATE INDEX IF NOT EXISTS city_aliases_city_id_idx ON city_aliases(city_id);

-- Исполнители
CREATE TABLE IF NOT EXISTS performers (
  performer_id SERIAL PRIMARY KEY,
  fio          TEXT NOT NULL,
  phone_norm   TEXT NOT NULL,
  geo_zone     TEXT DEFAULT '',
  note         TEXT DEFAULT '',
  created_at   TIMESTAMPTZ DEFAULT now(),
  updated_at   TIMESTAMPTZ DEFAULT now()
);
CREATE UNIQUE INDEX IF NOT EXISTS performers_phone_fio_uidx ON
performers(phone_norm, fio);
CREATE INDEX IF NOT EXISTS performers_phone_gin ON performers USING GIN
(phone_norm gin_trgm_ops);

-- Варианты маршрутов
CREATE TABLE IF NOT EXISTS route_variants (
  variant_id SERIAL PRIMARY KEY,
  name       TEXT DEFAULT '',
  stops      INT[] NOT NULL,
  created_at TIMESTAMPTZ DEFAULT now(),
  updated_at TIMESTAMPTZ DEFAULT now(),
  CONSTRAINT stops_min_two CHECK (cardinality(stops) >= 2)
);

-- Позиции городов в маршруте
CREATE TABLE IF NOT EXISTS route_variant_positions (
  variant_id INT NOT NULL REFERENCES route_variants(variant_id) ON DELETE
CASCADE,
  city_id    INT NOT NULL REFERENCES cities(city_id) ON DELETE CASCADE,
  pos        INT NOT NULL,
  PRIMARY KEY (variant_id, city_id)
);
CREATE INDEX IF NOT EXISTS rvp_city_variant_pos_idx ON
route_variant_positions(city_id, variant_id, pos);

-- Привязка маршрут ↔ исполнитель
CREATE TABLE IF NOT EXISTS performer_variants (
  performer_id INT NOT NULL REFERENCES performers(performer_id) ON DELETE
CASCADE,
  variant_id   INT NOT NULL REFERENCES route_variants(variant_id) ON DELETE
CASCADE,
```

```
  PRIMARY KEY (performer_id, variant_id)
);

-- Сделки
CREATE TABLE IF NOT EXISTS deals (
  deal_id      BIGSERIAL PRIMARY KEY,
  created_at   TIMESTAMPTZ DEFAULT now(),
  updated_at   TIMESTAMPTZ DEFAULT now(),
  performer_id INT REFERENCES performers(performer_id),
  city_from    INT REFERENCES cities(city_id),
  city_to      INT REFERENCES cities(city_id),
  variant_id   INT REFERENCES route_variants(variant_id),
  cost_rub     NUMERIC(12,2) DEFAULT 0,
  status       TEXT DEFAULT 'new',
  payload      JSONB DEFAULT '{}'::jsonb
);
CREATE INDEX IF NOT EXISTS deals_perf_idx ON deals(performer_id);
CREATE INDEX IF NOT EXISTS deals_route_idx ON deals(city_from, city_to);

-- Журнал аудита
CREATE TABLE IF NOT EXISTS audit_log (
  audit_id   BIGSERIAL PRIMARY KEY,
  table_name TEXT NOT NULL,
  op         TEXT NOT NULL,
  pk         JSONB,
  old_row    JSONB,
  new_row    JSONB,
  changed_at TIMESTAMPTZ DEFAULT now(),
  changed_by TEXT DEFAULT current_user,
  txid       BIGINT DEFAULT txid_current()
);

-- Нормализация/ensure
CREATE OR REPLACE FUNCTION norm_text(t TEXT) RETURNS TEXT AS $$
BEGIN
  RETURN trim(regexp_replace(lower(coalesce(t,'')), '\s+', ' ', 'g'));
END; $$ LANGUAGE plpgsql IMMUTABLE;

CREATE OR REPLACE FUNCTION norm_phone(t TEXT) RETURNS TEXT AS $$
BEGIN
  RETURN regexp_replace(coalesce(t,''), '\\D', '', 'g');
END; $$ LANGUAGE plpgsql IMMUTABLE;

CREATE OR REPLACE FUNCTION ensure_city(t TEXT) RETURNS INT AS $$
DECLARE
  nn TEXT := norm_text(t);
  cid INT;
BEGIN
  IF nn = '' THEN RETURN NULL; END IF;
  SELECT c.city_id INTO cid
  FROM city_aliases a JOIN cities c ON c.city_id = a.city_id
```

```
  WHERE a.alias_norm = nn;
  IF cid IS NOT NULL THEN RETURN cid; END IF;

  SELECT city_id INTO cid FROM cities WHERE name_norm = nn;
  IF cid IS NOT NULL THEN RETURN cid; END IF;

  INSERT INTO cities(name_norm, name_display) VALUES (nn, initcap(nn))
RETURNING city_id INTO cid;
  INSERT INTO city_aliases(alias_norm, city_id) VALUES (nn, cid) ON CONFLICT
DO NOTHING;
  RETURN cid;
END; $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION ensure_performer(p_fio TEXT, p_phone TEXT)
RETURNS INT AS $$
DECLARE
  nn_phone TEXT := right(norm_phone(p_phone), 11);
  pid INT;
BEGIN
  IF coalesce(p_fio,'') = '' OR coalesce(nn_phone,'') = '' THEN RETURN NULL;
END IF;
  SELECT performer_id INTO pid FROM performers WHERE phone_norm = nn_phone
AND fio = p_fio;
  IF pid IS NOT NULL THEN
    UPDATE performers SET updated_at = now() WHERE performer_id = pid;
    RETURN pid;
  END IF;
  INSERT INTO performers(fio, phone_norm) VALUES (p_fio, nn_phone) RETURNING
performer_id INTO pid;
  RETURN pid;
END; $$ LANGUAGE plpgsql;
```

/backend/extras.sql

```
-- updated_at триггеры
CREATE OR REPLACE FUNCTION trg_touch_updated_at() RETURNS TRIGGER AS $$
BEGIN
  NEW.updated_at := now();
  RETURN NEW;
END; $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS performers_touch ON performers;
CREATE TRIGGER performers_touch BEFORE UPDATE ON performers
FOR EACH ROW EXECUTE FUNCTION trg_touch_updated_at();

DROP TRIGGER IF EXISTS route_variants_touch ON route_variants;
CREATE TRIGGER route_variants_touch BEFORE UPDATE ON route_variants
FOR EACH ROW EXECUTE FUNCTION trg_touch_updated_at();
```

```
DROP TRIGGER IF EXISTS deals_touch ON deals;
CREATE TRIGGER deals_touch BEFORE UPDATE ON deals
FOR EACH ROW EXECUTE FUNCTION trg_touch_updated_at();

-- Аудит
CREATE OR REPLACE FUNCTION audit_trigger() RETURNS TRIGGER AS $$
BEGIN
  IF TG_OP = 'INSERT' THEN
    INSERT INTO audit_log(table_name, op, pk, old_row, new_row)
    VALUES (TG_TABLE_NAME, TG_OP, NULL, NULL, to_jsonb(NEW));
    RETURN NEW;
  ELSIF TG_OP = 'UPDATE' THEN
    INSERT INTO audit_log(table_name, op, pk, old_row, new_row)
    VALUES (TG_TABLE_NAME, TG_OP, NULL, to_jsonb(OLD), to_jsonb(NEW));
    RETURN NEW;
  ELSIF TG_OP = 'DELETE' THEN
    INSERT INTO audit_log(table_name, op, pk, old_row, new_row)
    VALUES (TG_TABLE_NAME, TG_OP, NULL, to_jsonb(OLD), NULL);
    RETURN OLD;
  END IF;
  RETURN NULL;
END; $$ LANGUAGE plpgsql;

DO $$
BEGIN
  IF NOT EXISTS (SELECT 1 FROM pg_trigger WHERE tgname='audit_performers')
THEN
    CREATE TRIGGER audit_performers AFTER INSERT OR UPDATE OR DELETE ON
performers
    FOR EACH ROW EXECUTE FUNCTION audit_trigger();
  END IF;
  IF NOT EXISTS (SELECT 1 FROM pg_trigger WHERE
tgname='audit_route_variants') THEN
    CREATE TRIGGER audit_route_variants AFTER INSERT OR UPDATE OR DELETE ON
route_variants
    FOR EACH ROW EXECUTE FUNCTION audit_trigger();
  END IF;
  IF NOT EXISTS (SELECT 1 FROM pg_trigger WHERE
tgname='audit_performer_variants') THEN
    CREATE TRIGGER audit_performer_variants AFTER INSERT OR UPDATE OR DELETE
ON performer_variants
    FOR EACH ROW EXECUTE FUNCTION audit_trigger();
  END IF;
  IF NOT EXISTS (SELECT 1 FROM pg_trigger WHERE tgname='audit_deals') THEN
    CREATE TRIGGER audit_deals AFTER INSERT OR UPDATE OR DELETE ON deals
    FOR EACH ROW EXECUTE FUNCTION audit_trigger();
  END IF;
END$$;

-- Пересборка позиций маршрута
```

```
CREATE OR REPLACE FUNCTION rebuild_variant_positions(p_variant_id INT)
RETURNS VOID AS $$
BEGIN
  DELETE FROM route_variant_positions WHERE variant_id = p_variant_id;
  INSERT INTO route_variant_positions(variant_id, city_id, pos)
  SELECT p_variant_id, city_id, ord
  FROM unnest((SELECT stops FROM route_variants WHERE variant_id =
p_variant_id)) WITH ORDINALITY AS t(city_id, ord);
END; $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION trg_variants_positions() RETURNS TRIGGER AS $$
BEGIN
  PERFORM rebuild_variant_positions(NEW.variant_id);
  RETURN NEW;
END; $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS route_variants_after_insupd ON route_variants;
CREATE TRIGGER route_variants_after_insupd
AFTER INSERT OR UPDATE OF stops ON route_variants
FOR EACH ROW EXECUTE FUNCTION trg_variants_positions();

-- Частичный поиск
CREATE OR REPLACE FUNCTION search_performers_partial(from_id INT, to_id INT)
RETURNS TABLE(performer_id INT, fio TEXT, phone_norm TEXT, variant_id INT)
AS $$
BEGIN
  RETURN QUERY
  SELECT pf.performer_id, pf.fio, pf.phone_norm, pv.variant_id
  FROM performer_variants pv
  JOIN route_variant_positions a ON a.variant_id = pv.variant_id AND
a.city_id IN (from_id, to_id)
  JOIN route_variant_positions b ON b.variant_id = pv.variant_id AND
b.city_id IN (from_id, to_id) AND a.city_id <> b.city_id
  JOIN performers pf ON pf.performer_id = pv.performer_id
  WHERE a.pos < b.pos;
END; $$ LANGUAGE plpgsql;

-- Точный поиск
CREATE OR REPLACE FUNCTION search_performers_exact(from_id INT, to_id INT)
RETURNS TABLE(performer_id INT, fio TEXT, phone_norm TEXT, variant_id INT)
AS $$
BEGIN
  RETURN QUERY
  WITH exact_variants AS (
    SELECT rv.variant_id
    FROM route_variants rv
    JOIN route_variant_positions p1 ON p1.variant_id = rv.variant_id AND
p1.pos = 1
    JOIN route_variant_positions pN ON pN.variant_id = rv.variant_id
    WHERE pN.pos = (SELECT cardinality(stops) FROM route_variants WHERE
variant_id = rv.variant_id)
```

```
      AND ((p1.city_id = from_id AND pN.city_id = to_id) OR (p1.city_id =
to_id AND pN.city_id = from_id))
  )
  SELECT pf.performer_id, pf.fio, pf.phone_norm, pv.variant_id
  FROM performer_variants pv
  JOIN exact_variants ev ON ev.variant_id = pv.variant_id
  JOIN performers pf ON pf.performer_id = pv.performer_id
  ORDER BY pf.fio;
END; $$ LANGUAGE plpgsql;
```

/frontend/package.json

```
{
  "name": "dispatcher-frontend",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start"
  },
  "dependencies": {
    "classnames": "2.5.1",
    "next": "14.2.4",
    "react": "18.2.0",
    "react-dom": "18.2.0",
    "swr": "2.2.5"
  },
  "devDependencies": {
    "autoprefixer": "10.4.19",
    "postcss": "8.4.38",
    "tailwindcss": "3.4.9",
    "typescript": "5.5.4"
  }
}
```

/frontend/tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2020",
    "lib": ["dom", "dom.iterable", "esnext"],
    "strict": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "noEmit": true,
```

```
    "module": "esnext",
    "moduleResolution": "bundler",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "jsx": "preserve",
    "baseUrl": ".",
    "paths": { "@/*": ["./*"] }
  },
  "include": ["**/*.ts", "**/*.tsx"],
  "exclude": ["node_modules"]
}
```

/frontend/tailwind.config.ts

```
import type { Config } from "tailwindcss";
const config: Config = {
  content: ["./app/**/*.{ts,tsx}", "./components/**/*.{ts,tsx}"],
  theme: { extend: {} },
  plugins: []
};
export default config;
```

/frontend/postcss.config.js

```
module.exports = { plugins: { tailwindcss: {}, autoprefixer: {} } };
```

/frontend/next.config.js

```
/** @type {import('next').NextConfig} */
const nextConfig = { reactStrictMode: true };
module.exports = nextConfig;
```

/frontend/.env.example

```
NEXT_PUBLIC_API_URL=http://127.0.0.1:8000
```

/frontend/app/globals.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;

:root { color-scheme: light; }
body { @apply bg-gray-50 text-gray-900; }
```

/frontend/app/layout.tsx

```
import "./globals.css";
import type { Metadata } from "next";

export const metadata: Metadata = { title: "Диспетчер перевозок",
description: "Поиск исполнителей и маршрутов" };
export default function RootLayout({ children }: { children:
React.ReactNode }) {
  return (
    <html lang="ru">
      <body className="min-h-screen">
        <div className="border-b bg-white">
          <div className="max-w-7xl mx-auto px-6 py-4 flex items-center
justify-between">
            <div className="flex items-center gap-3">
              <div className="text-blue-600 text-2xl">✈️</div>
              <div className="text-xl font-semibold">Диспетчер перевозок</
div>
            </div>
            <a href="/" className="text-sm text-blue-600
hover:underline">Главная</a>
          </div>
        </div>
        {children}
      </body>
    </html>
  );
}
```

/frontend/components/ui.tsx

```
import React from "react";
import cx from "classnames";
```

```
export const Input = (p: React.InputHTMLAttributes<HTMLInputElement>) => (
  <input {...p} className={cx("w-full px-4 py-2 rounded-xl border border-
gray-300 focus:outline-none focus:ring-2 focus:ring-blue-500",
p.className)} />
);
export const Button = ({ className, ...p }:
React.ButtonHTMLAttributes<HTMLButtonElement>) => (
  <button {...p} className={cx("px-4 py-2 rounded-xl bg-blue-600 text-white
hover:bg-blue-700 disabled:opacity-50", className)} />
);
export const Card: React.FC<{ children: React.ReactNode; className?: string }
> = ({ children, className }) => (
  <div className={cx("bg-white border border-gray-200 rounded-2xl p-4 shadow-
sm", className)}>{children}</div>
);
```

/frontend/lib/api.ts

```
export const API_BASE = process.env.NEXT_PUBLIC_API_URL || "http://
127.0.0.1:8000";
async function j<T>(res: Response): Promise<T> { if (!res.ok) throw new
Error(`HTTP ${res.status}`); return await res.json(); }
export async function getHealth() { return j(await fetch(`${API_BASE}/
health`, { cache: "no-store" })); }
export async function searchCities(q: string, opts: RequestInit = {}) {
return j(await fetch(`${API_BASE}/cities?q=${encodeURIComponent(q)}`, {
cache: "no-store", ...opts })); }
export async function searchPerformers(payload: { from_city: number;
to_city: number; mode: "exact" | "partial" }) {
  return j(await fetch(`${API_BASE}/search`, { method: "POST", headers: {
"Content-Type": "application/json" }, body: JSON.stringify(payload) }));
}
export async function getDeals(params: { limit?: number; offset?: number;
performer_id?: number } = {}) {
  const usp = new URLSearchParams();
  if (params.limit) usp.set("limit", String(params.limit));
  if (params.offset) usp.set("offset", String(params.offset));
  if (params.performer_id) usp.set("performer_id",
String(params.performer_id));
  return j(await fetch(`${API_BASE}/deals?${usp.toString()}`, { cache: "no-
store" }));
}
```

/frontend/components/CityCombobox.tsx

```
"use client";
import React, { useState, useEffect, useMemo } from "react";
import { Input } from "./ui";
import { searchCities } from "@/lib/api";

type City = { city_id: number; name_display: string };
export default function CityCombobox({ label, onSelect }: { label: string;
onSelect: (c: City | null) => void }) {
  const [q, setQ] = useState("");
  const [options, setOptions] = useState<City[]>([]);
  const [open, setOpen] = useState(false);
  const [loading, setLoading] = useState(false);

  // простейший дебаунс
  const debounced = useMemo(() => q, [q]);

  useEffect(() => {
    if (!debounced) { setOptions([]); return; }
    const ctrl = new AbortController();
    setLoading(true);
    searchCities(debounced, { signal: ctrl.signal })
      .then(setOptions)
      .catch(() => {})
      .finally(() => setLoading(false));
    return () => ctrl.abort();
  }, [debounced]);

  return (
    <div className="relative">
      <div className="mb-2 text-sm text-gray-700">{label}</div>
      <Input value={q} onChange={(e)=>{ setQ(e.target.value);
setOpen(true); }} placeholder="Начните вводить город…" />
      {open && (loading || options.length>0) && (
        <div className="absolute z-10 mt-1 w-full bg-white border border-
gray-200 rounded-xl shadow-lg max-h-64 overflow-auto">
          {loading && <div className="px-3 py-2 text-sm text-
gray-500">Загрузка…</div>}
          {!loading && options.map((c)=> (
            <div key={c.city_id} onClick={()=>{ onSelect(c);
setQ(c.name_display); setOpen(false); }} className="px-3 py-2 text-sm
hover:bg-blue-50 cursor-pointer">
              {c.name_display}
            </div>
          ))}
        </div>
      )}
    </div>
```

```
  );
}
```

/frontend/app/page.tsx

```
"use client";
import React, { useMemo, useState } from "react";
import useSWR from "swr";
import { Button, Card } from "@/components/ui";
import CityCombobox from "@/components/CityCombobox";
import { API_BASE, getDeals, searchPerformers } from "@/lib/api";

export default function Page() {
  const [fromCity, setFrom] = useState<{city_id:number; name_display:string}
| null>(null);
  const [toCity, setTo] = useState<{city_id:number; name_display:string} |
null>(null);
  const [mode, setMode] = useState<"exact"|"partial">("exact");
  const [queryKey, setQueryKey] = useState(0);

  const { data: deals } = useSWR(["deals", queryKey], () => getDeals({
limit: 25 }), { revalidateOnFocus: false });

  const [loading, setLoading] = useState(false);
  const [results, setResults] = useState<any[]|null>(null);

  async function runSearch(){
    if(!fromCity || !toCity) return;
    setLoading(true);
    try {
      const r = await searchPerformers({ from_city: fromCity.city_id,
to_city: toCity.city_id, mode });
      setResults(r as any[]);
    } finally { setLoading(false); }
  }

  const title = useMemo(()=> fromCity && toCity ? `${fromCity.name_display}
→ ${toCity.name_display}` : "Поиск исполнителей", [fromCity,toCity]);

  return (
    <main className="max-w-7xl mx-auto px-6 py-6 space-y-6">
      <div className="grid grid-cols-1 md:grid-cols-4 gap-4">
        <Card>
          <div className="text-sm text-gray-700 mb-3">API</div>
          <div className="text-sm">{API_BASE}</div>
        </Card>
        <Card>
          <div className="text-sm text-gray-700 mb-3">Найти исполнителя</div>
```

```
          <div className="grid grid-cols-1 md:grid-cols-2 gap-3">
            <CityCombobox label="Откуда" onSelect={setFrom} />
            <CityCombobox label="Куда" onSelect={setTo} />
          </div>
          <div className="flex items-center gap-3 mt-3">
            <div className="flex rounded-xl border p-1 bg-gray-50">
              <button onClick={()=>setMode("exact")} className={`px-3 py-1
text-sm rounded-lg ${mode==="exact"?"bg-white shadow":""}`}>Точное</button>
              <button onClick={()=>setMode("partial")} className={`px-3 py-1
text-sm rounded-lg ${mode==="partial"?"bg-white shadow":""}`}>Частичное</
button>
            </div>
            <Button onClick={runSearch} disabled={!fromCity||!toCity||
loading}>{loading?"Ищу…":"Найти"}</Button>
          </div>
        </Card>
        <Card className="md:col-span-2">
          <div className="text-sm text-gray-700 mb-2">Подсказка</div>
          <div className="text-sm text-gray-600">Выберите города из
автодополнения. В режиме «Частичное» попадут водители с длинными маршрутами,
проходящими через оба города в нужном порядке.</div>
        </Card>
      </div>

      <section className="grid grid-cols-1 lg:grid-cols-3 gap-6">
        <Card className="lg:col-span-2">
          <div className="flex items-center justify-between mb-3">
            <div className="text-lg font-semibold">{title}</div>
            {results && <div className="text-sm text-gray-600">Найдено:
<b>{results.length}</b></div>}
          </div>
          {!results && <div className="text-sm text-gray-500">Выберите
города и нажмите «Найти».</div>}
          {results && results.length===0 && <div className="text-sm text-
gray-500">Совпадений не найдено.</div>}
          {results && results.length>0 && (
            <div className="grid sm:grid-cols-2 gap-3">
              {results.map((r:any)=> (
                <div key={`${r.performer_id}-${r.variant_id||"nv"}`}
className="p-4 rounded-2xl border hover:shadow-sm transition">
                  <div className="flex items-center justify-between">
                    <div>
                      <div className="font-semibold">{r.fio}</div>
                      <div className="text-sm text-gray-600">{r.phone_norm}</
div>
                    </div>
                    {r.variant_id && <span className="text-xs bg-blue-100
text-blue-700 px-2 py-1 rounded-full">вариант #{r.variant_id}</span>}
                  </div>
                </div>
              ))}
```

```
            </div>
          )}
        </Card>

        <Card>
          <div className="text-lg font-semibold mb-2">Последние сделки</div>
          {!deals && <div className="text-sm text-gray-500">Загружаю…</div>}
          {deals && deals.length===0 && <div className="text-sm text-
gray-500">Пока нет данных</div>}
          <div className="space-y-2 max-h-[460px] overflow-auto pr-1">
            {deals && deals.map((d:any)=> (
              <div key={d.deal_id} className="p-3 rounded-xl border bg-
white">
                <div className="flex items-center justify-between">
                  <div className="font-medium text-sm">#{d.deal_id}</div>
                  <div className="text-xs text-gray-500">{new
Date(d.created_at).toLocaleDateString()}</div>
                </div>
                <div className="text-sm text-gray-700">{d.city_from} →
{d.city_to}</div>
                <div className="text-sm"><span className="text-gray-500">₽</
span> {Number(d.cost_rub||0).toLocaleString()}</div>
              </div>
            ))}
          </div>
        </Card>
      </section>

      <footer className="text-xs text-gray-500">API:
<code>NEXT_PUBLIC_API_URL</code>. Деплой: Vercel (frontend) + Railway/
Timeweb (backend).</footer>
    </main>
  );
}
```

/frontend/tests/config.test.mjs

```
import fs from "node:fs";
import path from "node:path";

function assert(cond, msg) { if (!cond) throw new Error(msg); }
const file = path.join(process.cwd(), "vercel.json");
if (fs.existsSync(file)) {
  const raw = fs.readFileSync(file, "utf8");
  const json = JSON.parse(raw);
  assert(json && typeof json === "object", "vercel.json must be object");
  assert(json.version === 2, "vercel.json must set version=2");
  assert(json.env && typeof json.env.NEXT_PUBLIC_API_URL === "string" &&
```

```
json.env.NEXT_PUBLIC_API_URL.length>0,
    "env.NEXT_PUBLIC_API_URL must be non-empty");
  console.log("✓ vercel.json ok");
} else {
  console.log("(i) vercel.json not present — ok");
}
```

`/frontend/vercel.json` (опционально)

```
{
  "version": 2,
  "env": {
    "NEXT_PUBLIC_API_URL": "https://api.your-domain.tld"
  }
}
```

`/tools/import_deals.py`

```
# import_deals.py — импорт CSV в PostgreSQL
import argparse, json, re, sys
import chardet
import pandas as pd
import psycopg2

def detect_encoding(path):
    with open(path, 'rb') as f:
        raw = f.read(1000000)
    enc = chardet.detect(raw)['encoding'] or 'utf-8'
    return enc

def read_csv_any(path):
    enc = detect_encoding(path)
    for sep in [None, ';', ',']:
        try:
            df = pd.read_csv(path, sep=sep, engine='python', encoding=enc)
            if len(df.columns) > 1:
                return df
        except Exception:
            continue
    raise RuntimeError('Не удалось прочитать CSV. Сохраните в UTF-8 с ;
или ,')

def find_col(cols, candidates):
    cl = {c.lower().strip(): c for c in cols}
    for name in candidates:
```

```
        if name.lower() in cl:
            return cl[name.lower()]
    for c in cols:
        lo = c.lower()
        if any(x.lower() in lo for x in candidates):
            return c
    return None

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument('--db', required=True, help='PostgreSQL URL')
    ap.add_argument('--csv', required=True, help='Path to CSV')
    args = ap.parse_args()

    df = read_csv_any(args.csv)
    cols = list(df.columns)

    col_from  = find_col(cols, ['Откуда полный','Откуда','origin','город
отправления'])
    col_to    = find_col(cols, ['Куда полный','Куда','destination','город
назначения'])
    col_fio   = find_col(cols, ['ФИО','fullname','ФИО клиента','контактное
лицо'])
    col_phone = find_col(cols, ['Номер телефона','телефон','тел.','phone'])
    col_cost  = find_col(cols, ['СЕБЕСТОИМОСТЬ
МАРШРУТА','Стоимость','Цена','Итоговая цена'])
    col_route = find_col(cols, ['Маршрут','Путь','трек'])

    miss = [k for k,v in
{'from':col_from,'to':col_to,'fio':col_fio,'phone':col_phone}.items() if v is
None]
    if miss:
        print('Не найдены обязательные колонки:', miss, file=sys.stderr)
        sys.exit(1)

    conn = psycopg2.connect(args.db)
    conn.autocommit = False
    cur = conn.cursor()

    # ensure helper functions exist
    cur.execute("""
    CREATE OR REPLACE FUNCTION norm_text(t TEXT) RETURNS TEXT AS $$
    BEGIN RETURN trim(regexp_replace(lower(coalesce(t,'')), '\s+', ' ',
'g')); END; $$ LANGUAGE plpgsql IMMUTABLE;
    CREATE OR REPLACE FUNCTION norm_phone(t TEXT) RETURNS TEXT AS $$
    BEGIN RETURN regexp_replace(coalesce(t,''), '\D', '', 'g'); END; $$
LANGUAGE plpgsql IMMUTABLE;
    CREATE OR REPLACE FUNCTION ensure_city(t TEXT) RETURNS INT AS $$
    DECLARE nn TEXT := norm_text(t); cid INT;
    BEGIN
      IF nn = '' THEN RETURN NULL; END IF;
```

```
      SELECT c.city_id INTO cid FROM city_aliases a JOIN cities c ON
c.city_id = a.city_id WHERE a.alias_norm = nn;
      IF cid IS NOT NULL THEN RETURN cid; END IF;
      SELECT city_id INTO cid FROM cities WHERE name_norm = nn;
      IF cid IS NOT NULL THEN RETURN cid; END IF;
      INSERT INTO cities(name_norm, name_display) VALUES (nn, initcap(nn))
RETURNING city_id INTO cid;
      INSERT INTO city_aliases(alias_norm, city_id) VALUES (nn, cid) ON
CONFLICT DO NOTHING;
      RETURN cid;
    END; $$ LANGUAGE plpgsql;
    CREATE OR REPLACE FUNCTION ensure_performer(p_fio TEXT, p_phone TEXT)
RETURNS INT AS $$
    DECLARE nn TEXT := right(norm_phone(p_phone), 11); pid INT;
    BEGIN
      IF coalesce(p_fio,'') = '' OR coalesce(nn,'') = '' THEN RETURN NULL;
END IF;
      SELECT performer_id INTO pid FROM performers WHERE phone_norm = nn AND
fio = p_fio;
      IF pid IS NOT NULL THEN UPDATE performers SET updated_at = now() WHERE
performer_id = pid; RETURN pid; END IF;
      INSERT INTO performers(fio, phone_norm) VALUES (p_fio, nn) RETURNING
performer_id INTO pid; RETURN pid;
    END; $$ LANGUAGE plpgsql;
    """)
    conn.commit()

    inserted = 0
    for _, r in df.iterrows():
        from_name = str(r.get(col_from, '') or '').strip()
        to_name   = str(r.get(col_to, '') or '').strip()
        fio       = str(r.get(col_fio, '') or '').strip()
        phone     = str(r.get(col_phone, '') or '').strip()
        if not (from_name and to_name and fio and phone):
            continue
        route_str = str(r.get(col_route, '') or '')
        cost_raw  = str(r.get(col_cost, '') or '').replace(' ',
'').replace(',', '.')
        try:
            cost = float(cost_raw) if cost_raw else 0.0
        except:
            cost = 0.0

        cur.execute("SELECT ensure_city(%s), ensure_city(%s)", (from_name,
to_name))
        cf, ct = cur.fetchone()
        cur.execute("SELECT ensure_performer(%s,%s)", (fio, phone))
        (pid,) = cur.fetchone()

        cur.execute(
            """INSERT INTO deals(performer_id, city_from, city_to, cost_rub,
```

```
payload)
                    VALUES (%s,%s,%s,%s, %s)""",
                (pid, cf, ct, cost, json.dumps({'route_str': route_str}))
            )
            inserted += 1
            if inserted % 1000 == 0:
                conn.commit()

    conn.commit()
    print(f'Inserted deals: {inserted}')

if __name__ == '__main__':
    main()
```

/tools/SQL/deals_import_helpers.sql

```
-- Нормализация/ensure для импорта через psql/Adminer
CREATE OR REPLACE FUNCTION norm_text(t TEXT) RETURNS TEXT AS $$
BEGIN
  RETURN trim(regexp_replace(lower(coalesce(t,'')), '\s+', ' ', 'g'));
END; $$ LANGUAGE plpgsql IMMUTABLE;

CREATE OR REPLACE FUNCTION norm_phone(t TEXT) RETURNS TEXT AS $$
BEGIN
  RETURN regexp_replace(coalesce(t,''), '\\D', '', 'g');
END; $$ LANGUAGE plpgsql IMMUTABLE;

CREATE OR REPLACE FUNCTION ensure_city(t TEXT) RETURNS INT AS $$
DECLARE
  nn TEXT := norm_text(t);
  cid INT;
BEGIN
  IF nn = '' THEN RETURN NULL; END IF;
  SELECT c.city_id INTO cid FROM city_aliases a JOIN cities c ON c.city_id =
a.city_id WHERE a.alias_norm = nn;
  IF cid IS NOT NULL THEN RETURN cid; END IF;
  SELECT city_id INTO cid FROM cities WHERE name_norm = nn;
  IF cid IS NOT NULL THEN RETURN cid; END IF;
  INSERT INTO cities(name_norm, name_display) VALUES (nn, initcap(nn))
RETURNING city_id INTO cid;
  INSERT INTO city_aliases(alias_norm, city_id) VALUES (nn, cid) ON CONFLICT
DO NOTHING;
  RETURN cid;
END; $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION ensure_performer(p_fio TEXT, p_phone TEXT)
RETURNS INT AS $$
DECLARE
```

```
  nn_phone TEXT := right(norm_phone(p_phone), 11);
  pid INT;
BEGIN
  IF coalesce(p_fio,'') = '' OR coalesce(nn_phone,'') = '' THEN RETURN NULL;
END IF;
  SELECT performer_id INTO pid FROM performers WHERE phone_norm = nn_phone
AND fio = p_fio;
  IF pid IS NOT NULL THEN
    UPDATE performers SET updated_at = now() WHERE performer_id = pid;
    RETURN pid;
  END IF;
  INSERT INTO performers(fio, phone_norm) VALUES (p_fio, nn_phone) RETURNING
performer_id INTO pid;
  RETURN pid;
END; $$ LANGUAGE plpgsql;
```

/tools/SQL/deals_import_psql.sql

```
-- Temp staging table
DROP TABLE IF EXISTS raw_deals;
CREATE TEMP TABLE raw_deals(
  from_text TEXT,
  to_text   TEXT,
  fio       TEXT,
  phone     TEXT,
  cost      TEXT,
  route_str TEXT
);

-- После \copy переложим в deals
WITH normed AS (
  SELECT
    from_text, to_text, fio, phone,
    NULLIF(replace(replace(cost, ' ', ''), ',', '.'), '')::numeric(12,2) AS
cost_rub,
    route_str
  FROM raw_deals
  WHERE coalesce(from_text,'') <> '' AND coalesce(to_text,'') <> '' AND
coalesce(fio,'') <> '' AND coalesce(phone,'') <> ''
), ids AS (
  SELECT
    ensure_city(from_text) AS cf,
    ensure_city(to_text)   AS ct,
    ensure_performer(fio, phone) AS pid,
    cost_rub,
    route_str
  FROM normed
)
```

```
INSERT INTO deals(performer_id, city_from, city_to, cost_rub, payload)
SELECT pid, cf, ct, cost_rub, jsonb_build_object('route_str', route_str)
FROM ids
WHERE pid IS NOT NULL AND cf IS NOT NULL AND ct IS NOT NULL;
```