Ниже — готовый каркас репозитория под MVP (2D DXF на входе). Включает: - backend/ (Java Spring Boot, REST-API, job-менеджмент) - planner/ (Python FastAPI + Shapely + OR-Tools + ezdxf) - infra: docker-compose с Postgres+PostGIS и MinIO - sample data и правила

Дерево проекта

```
lv-wiring-planner/
├─ docker-compose.yml
├─ .env.example
├─ README.md
├─ infra/
│  ├─ db/
│  │  ├─ init.sql
│  │  └─ postgis-indexes.sql
│  └─ minio/
│     └─ create-buckets.sh
├─ backend/
│  ├─ pom.xml
│  ├─ src/main/java/com/example/planner/PlannerApplication.java
│  ├─ src/main/java/com/example/planner/config/S3Config.java
│  ├─ src/main/java/com/example/planner/config/DbConfig.java
│  ├─ src/main/java/com/example/planner/controller/ProjectController.java
│  ├─ src/main/java/com/example/planner/controller/JobController.java
│  ├─ src/main/java/com/example/planner/dto/
│  │  ├─ CreateProjectRequest.java
│  │  ├─ UploadFileResponse.java
│  │  ├─ PlanPreferences.java
│  │  └─ JobStatusResponse.java
│  ├─ src/main/java/com/example/planner/entity/
│  │  ├─ Project.java
│  │  ├─ Room.java
│  │  ├─ Device.java
│  │  └─ Route.java
│  ├─ src/main/java/com/example/planner/repo/
│  │  ├─ ProjectRepository.java
│  │  └─ RoomRepository.java
│  ├─ src/main/resources/
│  │  ├─ application.yml
│  │  └─ rules-default.json
│  └─ Dockerfile
├─ planner/
│  ├─ app/
│  │  ├─ main.py
│  │  ├─ models.py
│  │  ├─ rules.py
│  │  ├─ geometry.py
```

```
│  │  ├─ placement.py
│  │  ├─ routing.py
│  │  ├─ export_dxf.py
│  │  └─ export_pdf.py
│  ├─ requirements.txt
│  └─ Dockerfile
└─ samples/
   ├─ rules/rules_ru_gost_example.json
   ├─ geojson/simple_apartment.geojson
   └─ dxf/README.txt
```

---

docker-compose.yml

```
version: "3.9"
services:
  db:
    image: postgis/postgis:15-3.4
    environment:
      POSTGRES_DB: planner
      POSTGRES_USER: planner
      POSTGRES_PASSWORD: planner
    ports: ["5432:5432"]
    volumes:
      - db_data:/var/lib/postgresql/data
      - ./infra/db/init.sql:/docker-entrypoint-initdb.d/10_init.sql:ro
      - ./infra/db/postgis-indexes.sql:/docker-entrypoint-initdb.d/
20_indexes.sql:ro

  minio:
    image: quay.io/minio/minio:RELEASE.2024-08-03T04-33-23Z
    environment:
      MINIO_ROOT_USER: minio
      MINIO_ROOT_PASSWORD: minio12345
    command: server /data --console-address ":9001"
    ports: ["9000:9000", "9001:9001"]
    volumes:
      - minio_data:/data

  backend:
    build: ./backend
    environment:
      SPRING_PROFILES_ACTIVE: docker
      DB_URL: jdbc:postgresql://db:5432/planner
      DB_USER: planner
      DB_PASS: planner
      S3_ENDPOINT: http://minio:9000
      S3_ACCESS_KEY: minio
```

```
      S3_SECRET_KEY: minio12345
      S3_BUCKET_RAW: raw-plans
      S3_BUCKET_EXPORTS: exports
      PLANNER_URL: http://planner:8000
    ports: ["8080:8080"]
    depends_on: [db, minio, planner]

  planner:
    build: ./planner
    environment:
      RULES_PATH: /app/rules.json
    volumes:
      - ./samples/rules/rules_ru_gost_example.json:/app/rules.json:ro

volumes:
  db_data: {}
  minio_data: {}
```

---

infra/db/init.sql (минимум для геометрии и проектов)

```
CREATE EXTENSION IF NOT EXISTS postgis;

CREATE TABLE IF NOT EXISTS project (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT now()
);

CREATE TABLE IF NOT EXISTS room (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  project_id UUID NOT NULL REFERENCES project(id) ON DELETE CASCADE,
  name TEXT,
  room_type TEXT,
  geom GEOMETRY(POLYGON, 3857)
);

CREATE TABLE IF NOT EXISTS device (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  project_id UUID NOT NULL REFERENCES project(id) ON DELETE CASCADE,
  room_id UUID REFERENCES room(id) ON DELETE SET NULL,
  type TEXT NOT NULL,
  elevation_cm INT,
  props JSONB,
  geom GEOMETRY(POINT, 3857)
);
```

```
CREATE TABLE IF NOT EXISTS route (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  project_id UUID NOT NULL REFERENCES project(id) ON DELETE CASCADE,
  cable_type TEXT,
  length_m DOUBLE PRECISION,
  geom GEOMETRY(LINESTRING, 3857)
);
```

---

backend/Dockerfile

```
FROM eclipse-temurin:21-jdk as build
WORKDIR /app
COPY pom.xml .
RUN mvn -q -e -B -DskipTests dependency:go-offline
COPY src ./src
RUN mvn -q -e -B -DskipTests package

FROM eclipse-temurin:21-jre
WORKDIR /app
COPY --from=build /app/target/backend-*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java","-jar","/app/app.jar"]
```

---

backend/pom.xml (минимальный)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>backend</artifactId>
  <version>0.1.0</version>
  <properties>
    <java.version>21</java.version>
    <spring.boot.version>3.3.4</spring.boot.version>
  </properties>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-dependencies</artifactId>
        <version>${spring.boot.version}</version>
        <type>pom</type>
```

```
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk-s3</artifactId>
      <version>1.12.772</version>
    </dependency>
    <dependency>
      <groupId>org.locationtech.jts</groupId>
      <artifactId>jts-core</artifactId>
      <version>1.20.0</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
          <finalName>backend-${project.version}</finalName>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

---

backend/src/main/resources/application.yml

```
server:
  port: 8080
spring:
  datasource:
```

```
    url: ${DB_URL}
    username: ${DB_USER}
    password: ${DB_PASS}
  jpa:
    hibernate:
      ddl-auto: none
    properties:
      hibernate.dialect: org.hibernate.dialect.PostgreSQLDialect
s3:
  endpoint: ${S3_ENDPOINT}
  accessKey: ${S3_ACCESS_KEY}
  secretKey: ${S3_SECRET_KEY}
  bucketRaw: ${S3_BUCKET_RAW}
  bucketExports: ${S3_BUCKET_EXPORTS}
planner:
  url: ${PLANNER_URL}
```

---

backend/src/main/java/com/example/planner/PlannerApplication.java

```
package com.example.planner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class PlannerApplication {
  public static void main(String[] args) {
    SpringApplication.run(PlannerApplication.class, args);
  }
}
```

---

backend/src/main/java/com/example/planner/controller/ProjectController.java

```
package com.example.planner.controller;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import java.util.Map;

@RestController
```

```
@RequestMapping("/api/projects")
public class ProjectController {

  @PostMapping
  public ResponseEntity<?> create(@RequestParam String name){
    // TODO: save Project to DB, return id
    return ResponseEntity.ok(Map.of("id",
"00000000-0000-0000-0000-000000000001", "name", name));
  }

  @PostMapping("/{id}/upload")
  public ResponseEntity<?> upload(@PathVariable String id, @RequestParam
MultipartFile file){
    // TODO: put to MinIO raw-plans, return key
    return ResponseEntity.ok(Map.of("projectId", id, "key",
file.getOriginalFilename()));
  }
}
```

---

backend/src/main/java/com/example/planner/controller/JobController.java

```
package com.example.planner.controller;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.Map;

@RestController
@RequestMapping("/api/jobs")
public class JobController {

  @PostMapping("/run")
  public ResponseEntity<?> run(@RequestBody Map<String,Object> req){
    // req: { projectId, preferencesJsonKey, rulesProfile }
    // TODO: call planner /process pipeline (ingest->place->route->export)
    return ResponseEntity.accepted().body(Map.of("jobId", "JOB-123"));
  }

  @GetMapping("/{jobId}")
  public ResponseEntity<?> status(@PathVariable String jobId){
    // TODO: read job status
    return ResponseEntity.ok(Map.of("jobId", jobId, "status", "DONE",
"exports", new String[]{"exports/drawings/project-1.pdf", "exports/drawings/
project-1.dxf"}));
```

```
  }
}
```

---

backend/src/main/resources/rules-default.json (черновик)

```
{
  "version": 1,
  "locale": "RU",
  "heights_cm": {"socket": 30, "switch": 90, "worktop_socket": 110,
"ceiling_light": 250},
  "min_offsets_cm": {"corner": 10, "door": 15, "wet_zone": 60},
  "per_room_requirements": {
    "BEDROOM": {"socket_per_wall_meter": 0.3, "min_switches": 1, "lights":
1},
    "LIVING": {"socket_per_wall_meter": 0.4, "min_switches": 1, "lights": 1},
    "KITCHEN": {"socket_per_wall_meter": 0.6, "worktop_zone": true,
"min_switches": 1, "lights": 2},
    "BATH": {"socket_per_wall_meter": 0.1, "min_switches": 1, "lights": 1}
  },
  "routing": {"h_trunk_height_cm": 20, "v_only_vertical": true,
"avoid_doors": true, "penalties": {"door_cross": 1000, "bearing_wall": 500}}
}
```

---

planner/Dockerfile

```
FROM python:3.11-slim
WORKDIR /app
COPY planner/requirements.txt /app/requirements.txt
RUN pip install --no-cache-dir -r requirements.txt
COPY planner/app /app
ENV PYTHONUNBUFFERED=1
EXPOSE 8000
CMD ["python","-m","app.main"]
```

---

planner/requirements.txt

```
fastapi==0.115.0
uvicorn==0.30.6
```

```
shapely==2.0.4
geojson==3.1.0
networkx==3.3
ezdxf==1.3.4
reportlab==4.2.2
ortools==9.10.4067
pydantic==2.8.2
```

---

planner/app/models.py (Pydantic-схемы)

```python
from pydantic import BaseModel
from typing import List, Literal, Optional

class IngestRequest(BaseModel):
    project_id: str
    src_s3_key: str  # путь к DXF в raw-plans

class PlaceRequest(BaseModel):
    project_id: str
    preferences: dict

class RouteRequest(BaseModel):
    project_id: str

class ExportRequest(BaseModel):
    project_id: str
    formats: List[Literal['PDF','DXF','PNG']] = ['PDF']
```

---

planner/app/rules.py (загрузка правил)

```python
import json, os

_rules = None

def get_rules():
    global _rules
    if _rules is None:
        path = os.getenv('RULES_PATH', '/app/rules.json')
        with open(path, 'r', encoding='utf-8') as f:
            _rules = json.load(f)
    return _rules
```

---

planner/app/geometry.py (упрощённая геометрия с GeoJSON sample)

```
from shapely.geometry import shape, Polygon, LineString, Point
import json, os

DB = {
    # для MVP храним в памяти: project_id -> rooms/devices/routes
    'rooms': {}, 'devices': {}, 'routes': {}
}

def load_sample_rooms(project_id: str, path: str = '/app/../samples/geojson/
simple_apartment.geojson'):
    with open(path, 'r', encoding='utf-8') as f:
        gj = json.load(f)
    rooms = [shape(feat['geometry']) for feat in gj['features']]
    DB['rooms'][project_id] = rooms
    return rooms

def room_walls(room: Polygon):
    xs, ys = room.exterior.coords.xy
    return [LineString([ (xs[i], ys[i]), (xs[i+1], ys[i+1]) ]) for i in
range(len(xs)-1)]

def along_wall_points(wall: LineString, step: float=1.5, offsets: float=0.2):
    # генерирует точки через step метров, с отступами от углов
    L = wall.length
    t = offsets
    pts = []
    while t < L - offsets:
        pts.append(wall.interpolate(t))
        t += step
    return pts
```

---

planner/app/placement.py (генерация кандидатов + CP-SAT выбор)

```
from .geometry import DB, room_walls, along_wall_points
from .rules import get_rules
from shapely.geometry import Point
from ortools.sat.python import cp_model

def generate_candidates(project_id: str):
    rules = get_rules()
```

```
    rooms = DB['rooms'][project_id]
    candidates = []  # (type, room_idx, point)
    for idx, room in enumerate(rooms):
        per_meter = rules['per_room_requirements'].get('LIVING',
{}).get('socket_per_wall_meter', 0.3)
        step = 1.0 / max(per_meter, 0.2)
        for wall in room_walls(room):
            for p in along_wall_points(wall, step=step, offsets=0.3):
                candidates.append(('SOCKET', idx, p))
        # Добавим выключатель возле одной из дверей — упростим: первая
вершина
        x, y = room.exterior.coords[0]
        candidates.append(('SWITCH', idx, Point(x, y)))
    return candidates


def select_devices(project_id: str, candidates):
    # Простейшая модель: берём до N сокетов на комнату + 1 выключатель
    rules = get_rules()
    model = cp_model.CpModel()
    xs = []
    for i, (t, room_idx, p) in enumerate(candidates):
        v = model.NewBoolVar(f"x_{i}")
        xs.append(v)
    # Ограничения по комнатам
    for room_idx in set(r for _,r,_ in candidates):
        idxs_socket = [i for i,(t,r,_) in enumerate(candidates) if
r==room_idx and t=='SOCKET']
        idxs_switch = [i for i,(t,r,_) in enumerate(candidates) if
r==room_idx and t=='SWITCH']
        # минимум 1 выключатель
        model.Add(sum(xs[i] for i in idxs_switch) >= 1)
        # максимум 6 сокетов для MVP
        model.Add(sum(xs[i] for i in idxs_socket) <= 6)
    # Цель: минимизировать количество устройств (как заглушка стоимости)
    model.Minimize(sum(xs))
    solver = cp_model.CpSolver()
    solver.parameters.max_time_in_seconds = 2.0
    res = solver.Solve(model)
    chosen = []
    if res in (cp_model.OPTIMAL, cp_model.FEASIBLE):
        for i, var in enumerate(xs):
            if solver.Value(var) == 1:
                chosen.append(candidates[i])
    DB['devices'][project_id] = chosen
    return chosen
```

---

planner/app/routing.py (грубый граф и кратчайшие маршруты)

```
from shapely.geometry import LineString, Point
from shapely.ops import unary_union
import networkx as nx
from .geometry import DB

# Примем точку щита фиксированной для MVP
PANEL = Point(0.5, 0.5)


def build_corridor_graph(rooms):
    # Граф: вершины — все вершины полигонов + PANEL; рёбра — стороны комнат
    G = nx.Graph()
    for room in rooms:
        coords = list(room.exterior.coords)
        for i in range(len(coords)-1):
            a = coords[i]; b = coords[i+1]
            G.add_node(a); G.add_node(b)
            length = Point(a).distance(Point(b))
            G.add_edge(a, b, weight=length)
    G.add_node((PANEL.x, PANEL.y))
    # соединим PANEL с ближайшей вершиной
    best = None; best_d = 1e9
    for n in G.nodes:
        d = Point(n).distance(PANEL)
        if d < best_d:
            best_d, best = d, n
    if best is not None:
        G.add_edge(best, (PANEL.x, PANEL.y), weight=best_d)
    return G


def route_all(project_id: str):
    rooms = DB['rooms'][project_id]
    devices = DB['devices'][project_id]
    G = build_corridor_graph(rooms)
    routes = []
    for t, room_idx, p in devices:
        start = (p.x, p.y)
        # привязываем к ближайшей вершине графа
        closest = min(G.nodes, key=lambda n: Point(n).distance(p))
        # путь до панели
        path = nx.shortest_path(G, closest, (PANEL.x, PANEL.y),
weight='weight')
        coords = [start] + path
        ls = LineString(coords)
        length = ls.length
```

```
            routes.append((t, ls, length))
    DB['routes'][project_id] = routes
    return routes
```

---

planner/app/export_dxf.py (минимальный DXF)

```
import ezdxf
from shapely.geometry import Polygon, Point, LineString

SYMBOLS = {
    'SOCKET': {'layer': 'EL_SOCKET'},
    'SWITCH': {'layer': 'EL_SWITCH'},
}


def export_dxf(project_id: str, rooms, devices, routes, out_path: str):
    doc = ezdxf.new()
    msp = doc.modelspace()
    # Комнаты (контуры)
    for room in rooms:
        pts = list(room.exterior.coords)
        msp.add_lwpolyline(pts, dxfattribs={'layer': 'ROOMS'})
    # Устройства
    for t, room_idx, p in devices:
        msp.add_circle((p.x, p.y), 0.05, dxfattribs={'layer': SYMBOLS[t]
['layer']})
    # Маршруты
    for t, ls, _ in routes:
        msp.add_lwpolyline(list(ls.coords), dxfattribs={'layer':
'EL_CABLES'})
    doc.saveas(out_path)
```

---

planner/app/export_pdf.py (очень простой PDF-чертёж)

```
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import A4
from reportlab.lib.units import cm

# Примитив: просто точки и линии в миллиметрах, без масштабов (MVP)

def export_pdf(project_id: str, rooms, devices, routes, out_path: str):
    c = canvas.Canvas(out_path, pagesize=A4)
```

```
    w, h = A4
    c.setLineWidth(0.5)
    # Рисуем комнаты
    c.setStrokeColorRGB(0,0,0)
    for room in rooms:
        pts = [(x*cm, y*cm) for x,y in list(room.exterior.coords)]
        for i in range(len(pts)-1):
            c.line(pts[i][0], pts[i][1], pts[i+1][0], pts[i+1][1])
    # Устройства
    for t, _, p in devices:
        x,y = p.x*cm, p.y*cm
        c.circle(x, y, 0.15*cm, stroke=1, fill=0)
    # Маршруты
    c.setStrokeColorRGB(0.2,0.2,0.8)
    for _, ls, _ in routes:
        coords = [(x*cm, y*cm) for x,y in list(ls.coords)]
        for i in range(len(coords)-1):
            c.line(coords[i][0], coords[i][1], coords[i+1][0], coords[i+1]
[1])
    c.showPage(); c.save()
```

---

planner/app/main.py (оркестрация FastAPI)

```
from fastapi import FastAPI
from .models import IngestRequest, PlaceRequest, RouteRequest, ExportRequest
from .geometry import load_sample_rooms, DB
from .placement import generate_candidates, select_devices
from .routing import route_all
from .export_dxf import export_dxf
from .export_pdf import export_pdf
import os

app = FastAPI()

@app.post('/ingest')
async def ingest(req: IngestRequest):
    # MVP: игнорируем DXF и берём samples/geojson
    rooms = load_sample_rooms(req.project_id)
    return {"project_id": req.project_id, "rooms": len(rooms)}

@app.post('/place')
async def place(req: PlaceRequest):
    cands = generate_candidates(req.project_id)
    chosen = select_devices(req.project_id, cands)
    return {"project_id": req.project_id, "devices": len(chosen)}

@app.post('/route')
async def route(req: RouteRequest):
```

```
    routes = route_all(req.project_id)
    total = sum(l for _,_,l in routes)
    return {"project_id": req.project_id, "routes": len(routes),
"length_sum": total}

@app.post('/export')
async def export(req: ExportRequest):
    rooms = DB['rooms'][req.project_id]
    devices = DB['devices'][req.project_id]
    routes = DB['routes'][req.project_id]
    os.makedirs('/tmp/exports', exist_ok=True)
    out = []
    if 'DXF' in req.formats:
        dxf_path = f"/tmp/exports/{req.project_id}.dxf"
        export_dxf(req.project_id, rooms, devices, routes, dxf_path)
        out.append(dxf_path)
    if 'PDF' in req.formats:
        pdf_path = f"/tmp/exports/{req.project_id}.pdf"
        export_pdf(req.project_id, rooms, devices, routes, pdf_path)
        out.append(pdf_path)
    return {"project_id": req.project_id, "files": out}
```

---

samples/geojson/simple_apartment.geojson (две комнаты, условные метры)

```
{
  "type": "FeatureCollection",
  "features": [
    {"type": "Feature", "properties": {"name": "living"},
     "geometry": {"type": "Polygon", "coordinates": [[[0,0],[6,0],[6,4],[0,
4],[0,0]]]}}
    ,{"type": "Feature", "properties": {"name": "bedroom"},
     "geometry": {"type": "Polygon", "coordinates": [[[6,0],[10,0],[10,4],[6,
4],[6,0]]]}}
  ]
}
```

---

samples/rules/rules_ru_gost_example.json (пример правил)

```
{
  "version": 1,
```

```
  "locale": "RU",
  "heights_cm": {"socket": 30, "switch": 90, "ceiling_light": 250},
  "min_offsets_cm": {"corner": 10, "door": 15, "wet_zone": 60},
  "per_room_requirements": {
    "LIVING": {"socket_per_wall_meter": 0.4, "min_switches": 1, "lights": 1},
    "BEDROOM": {"socket_per_wall_meter": 0.3, "min_switches": 1, "lights": 1}
  },
  "routing": {"h_trunk_height_cm": 20, "v_only_vertical": true,
"avoid_doors": true, "penalties": {"door_cross": 1000}}
}
```

README.md (шпаргалка)

```
## Запуск

```bash
docker compose up --build
```

Проверка планировщика (использует samples/geojson):

```bash
# 1) Ingest (поднимаем комнаты)
curl -X POST http://localhost:8000/ingest -H 'Content-Type: application/
json'
  -d '{"project_id":"proj-1", "src_s3_key":"raw-plans/plan1.dxf"}'

# 2) Расстановка устройств
curl -X POST http://localhost:8000/place -H 'Content-Type: application/
json'
  -d '{"project_id":"proj-1", "preferences":{}}'

# 3) Маршрутизация
curl -X POST http://localhost:8000/route -H 'Content-Type: application/
json'
  -d '{"project_id":"proj-1"}'

# 4) Экспорт DXF+PDF
curl -X POST http://localhost:8000/export -H 'Content-Type: application/
json'
  -d '{"project_id":"proj-1", "formats":["DXF","PDF"]}'
```

Файлы появятся в контейнере planner по пути `/tmp/exports/`.

Следующие шаги

- [ ] Реальный парсинг DXF: ezdxf → полилинии → полигоны комнат
```

- [ ] Слои и типы помещений (семантика)
- [ ] Жёсткие правила (ГОСТ/ПУЭ) из БД вместо захардкоженных
- [ ] Точная шкала и листы (PDF с легендой)
- [ ] Слив результатов в MinIO, отдача ссылок из backend
- [ ] Редактор 2D во фронтенде (перетаскивание устройств) ```

Пояснения

- Сейчас planner упрощённо берёт `samples/geojson/simple_apartment.geojson` вместо реального DXF — это ускоряет старт. Заменим на парсер DXF, когда проверим пайплайн end-to-end.
- Маршрутизация — через граф по вершинам полигонов; на практике нужно строить «линию штробы» на фиксированных высотах и добавлять штрафы за пересечения дверей/несущих стен.
- Экспорт DXF/PDF — примитивный, но рабочий. Символы можно заменить на DXF-блоки.