Отчётность по квартирам

Модуль отчётности позволяет собирать агрегированные данные об объекте из таблицы `users.flat_reports` и формировать PDF с кратким описанием квартиры, рынка и аналогов. Ниже краткое описание архитектуры, API и CLI-скрипта.

1. Что должно быть в окружении

1. PostgreSQL DSN – задаётся через `FLAT_REPORTS_DSN` (предпочтительно) или `DATABASE_URL`. Именно этот DSN используется как FastAPI, так и CLI-скриптом.
2. ReportLab – библиотека для генерации PDF (`pip install reportlab`).
3. Шрифты – чтобы не получать "квадратики", положите TTF/OTF в `server/Inter` или `server/fonts`. Можно указать путь через `REPORT_FONT_PATH` или `REPORT_FONT_EXTRA_PATHS`.

2. HTTP API

Endpoint: `POST /api/reports/flat`

```
{
  "flat_id": 77,
  "report_date": null,
  "output_path": null,
  "regenerate": true
}
```

Поведение: 1. API получает `flat_id` и читает строку из `users.user_flats`. Оттуда берутся `tg_user_id`, `house_id`, `floor`, `rooms`, `radius_m`, `area_ratio`, `floor_delta` и `days_limit`. Если `radius_m` NULL — используется 1500, аналоги получают значения по умолчанию `0.15/2/30`. 2. Если в `users.user_flats.house_id` ещё нет значения, API вызывает внутренне слой подготовки (`ReportPipeline.prepare(flat_id, run_parser=False)`), чтобы автоматически определить дом (`house_id`) и записать его в `user_flats`. 3. Если `regenerate=true`, API вызывает `users.build_flat_report(...)` и `users.build_flat_report_analogs(...)` с параметрами из `user_flats`. 4. Берёт свежий `report_json` из `users.flat_reports` (`fetch_latest_report_json`). 5. Собирает PDF `build_flat_report_pdf` и возвращает JSON:

```
{
  "success": true,
  "pdf_path": "server/flat_report_92207_7_2_20241218120000.pdf",
  "file_size": 2912,
  "params": {
    "flat_id": 77,
    "tg_user_id": 123456789,
    "house_id": 92207,
    "floor": 7,
    "rooms": 2,
    "radius_m": 1500,
    "analogs_area_ratio": 0.15,
    "analogs_floor_delta": 2,
    "analogs_days_limit": 30,
    "report_date": null,
    "output_path": "server/flat_report_92207_7_2_20241218120000.pdf",
    "regenerate": true
  },
  "message": "PDF-[redacted]"
}
```

Чтобы сохранить файл в другом месте, передайте `output_path` (путь должен быть доступен процессу API). По умолчанию PDF сохраняется рядом с `server/realty_parser_server.py` (например, `server/flat_report_92207_7_2_20241218120000.pdf`).

3. CLI-скрипт

Файл: `server/rep.py`

```
.venv/bin/python server/rep.py
```

Параметры прошиты в начале файла:

```
tg_user_id = 123456789
house_id = 92207
floor = 7
rooms = 2
radius_m = 1500
report_date = None
regenerate = True
analogs_area_ratio = 0.15
analogs_floor_delta = 2
analogs_days_limit = 30
output_path = Path("flat_report.pdf")
```

Логика такая же, как у API: скрипт вызывает SQL-процедуры (если `regenerate=True`), берёт JSON и рендерит PDF в `flat_report.pdf` (можно изменить путь).

4. Подготовка отчёта вручную

Если требуется собственный пайплайн:

1. Вызовите функцию `users.build_flat_report(tg_user_id, house_id, floor, rooms, radius_m)`.
2. Вызовите `users.build_flat_report_analogs(tg_user_id, house_id, floor, rooms, radius_m, area_ratio, floor_delta, days_limit)`.
3. Прочитайте `report_json`:

```
SELECT report_json
FROM users.flat_reports
WHERE tg_user_id = ... AND house_id = ... AND floor = ... AND rooms = ... AND radius_m = ...
ORDER BY updated_at DESC
LIMIT 1;
```

1. Передайте JSON в `server.reportlab.build_flat_report_pdf(...)`.

5. Подготовка данных users.ads для user_flats

Регенерация данных для отчёта начинается с заполнения `users.ads`, чтобы `users.build_flat_report*` чувствовали себя уверенно:

1. Берём `users.user_flats` (например, `id=77` с `tg_user_id`, `rooms`, `floor`). Если в таблице есть поле `radius_m`, оно используется; иначе читаем `meters` (оставшуюся для старых строк), а при отсутствии обоих — по умолчанию `1000` метров. Поле сохраняется обратно в `users.user_flats.radius_m`, чтобы следующая подготовка сразу знала значение.
2. Через `public.get_house_id_by_address(p_address)` находим `house_id`. Если дом не определён, операция останавливается с ошибкой.
3. Первый найденный `house_id` сохраняется в `users.user_flats.house_id`, так что последующие запуски пропускают второй `get_house_id`.
4. Скачиваем строки из `public.flats_history` по найденному `house_id` и подставляем их в `users.ads` (помечая `ads.from=0`, `source='flats_history'`, `distance_m=0`). Это связывает «собственную» квартиру с пользователем.
5. Вызываем `public.find_nearby_apartments(address, rooms, current_price, area, kitchen_area, radius)` и добавляем результат в `users.ads` с `ads.from=2`, чтобы собрать конкурентов из окрестностей.
6. По каждому URL (если он ведёт на Cian) последовательно делаем `GET /api/parse/ext?url=...` в пределах вашего FastAPI (`REPORT_PARSER_BASE_URL`, по умолчанию `http://localhost:8008`). Это тот же расширенный парсер, но без адреса и фото, чтобы запросы были легче; ответ применяем к `users.ads`, пополняя поля цены, площади, этажа и статуса, чтобы `users.build_flat_report*` получил самые свежие данные.

> Если FastAPI слушает на другом адресе, настройте `REPORT_PARSER_BASE_URL` в `.env`, чтобы pipeline знал, куда шлать запросы.

Эту подготовку можно запускать вручную через `POST /api/reports/prepare` с `flat_id` + необязательными настройками `radius_m`, `max_history`, `max_nearby`, `run_parser`. Результат содержит `result.history_ads`, `result.nearby_ads`, `result.persisted_ads` и список успешно распарсенных ссылок.

6. Проверка изменений объявлений

Endpoint `POST /api/reports/flats_state` пробегает все строки `users.ads` с `status = true`, делает `GET /api/parse/flat_state?url=...`, сравнивает цену/статус и, если отличается, записывает snapshot в `users.ad_history` (поля `price`, `status`, `views_today`, `created_at`). Ответ содержит число проверенных объявлений, сколько из них обновилось и краткий список изменений.

```
{
  "success": true,
  "checked": 42,
  "updated": 3,
  "changes": [
    {
      "ad_id": 123,
      "url": "https://www.cian.ru/sale/flat/320454077/",
      "price_changed": true,
      "status_changed": false
    }
  ],
  "errors": []
}
```

Это позволяет регулярно синхронизировать `users.ad_history` и видеть, какой ад открылся/перешёл в архив.

7. Отладка шрифтов

Если текст в PDF отображается квадратиками:

- Убедитесь, что в `server/Inter` лежит TTF (не только OTF). Для Inter можно использовать `Inter-V.ttf` или экспортировать TTF из Google Fonts.
- Установите переменную `REPORT_FONT_PATH=/absolute/path/to/your.ttf`.
- Запустите `python server/rep.py` – в логах появится сообщение, если шрифт не растёрлся.

8. Примеры SQL

```
SELECT users.build_flat_report(123456789, 92207, 7, 2, 1500);
SELECT users.build_flat_report_analogs(123456789, 92207, 7, 2, 1500, 0.15, 2, 30);
```

Эти вызовы можно делать вручную, если нужно подготовить отчёт задолго до формирования PDF.

cd /Users/pavellebedev/Desktop/pyton/realty/mrealty .venv/bin/python -m uvicorn server.realty_parser_server:app --reload --port 8008

curl -X POST http://localhost:8008/api/reports/flat \ -H "Content-Type: application/json" \ -d '{"flat_id":77,"regenerate":true}'

9. Генерация PDF документации

Файл `server/build_reporting_pdf.py` переводит `server/reporting.md` в PDF и кладёт его рядом с серверными ресурсами (`server/[redacted] ([redacted]).pdf`). Скрипт использует те же шрифты, что и основной генератор отчётов, и опирается на `markdown` + `bs4` для простого форматирования.

```
python server/build_reporting_pdf.py
```

Параметры:

- `--input` (`-i`) — путь к Markdown-файлу (по умолчанию `server/reporting.md`).

- `--output` (`-o`) — итоговый PDF (по умолчанию `server/[redacted] ([redacted]).pdf`).

Если вы вносите изменения в `reporting.md`, запустите скрипт заново, чтобы обновить PDF-копию. Убедитесь, что в окружении установлены зависимости: `pip install reportlab markdown beautifulsoup4`.

10. AI-комментарии к таблицам

PDF теперь снабжается краткими выводами рядом с таблицами ([redacted], [redacted], [redacted]). Комментарии формируются через OpenRouter (модель по умолчанию `nex-agi/deepseek-v3.1-nex-n1:free`) и берутся из `server/report_ai_commentary.py`. Чтобы текст генерировался автоматически:

- Задайте `OPENROUTER_API_KEY` в окружении (такой же ключ, что используется для остальных ботов).
- При необходимости поменяйте модель с помощью `OPENROUTER_MODEL` и добавьте запасной через `OPENROUTER_FALLBACK_MODEL`.

Если ключа нет, комментарии заменяются простыми описаниями (функция `server/report_ai_commentary.py` сама умеет создавать fallback). Отдельных действий для чтения PDF не требуется, всё происходит во время вызова `build_flat_report_pdf`.