Поехали к пользовательским отчетам

Меню "Пользовательские"

Попадаем на список отчетов ksk_report_header
Таблица отфильтрована так что видим только пользовательские отчеты.

Пользователю доступна кнопка "Создать"
По нажатию на кнопку открывается форма, там пользователя выбирает тип отчета, диапазон дат. Нажимает создать. Система создает запись в ksk_report_header и отправляет отчет на создание.
UI регулярно опрашивает бакенд для получения статуса отчета

Т.е. нам нужны функции REST API

1. список отчетов
2. создание отчета
3. Удаление отчета
4. получение статуса отчета

Выведи в том же виде. Предложи улучшения.

Отлично! Проанализировал структуру БД. Теперь сформирую REST API для пользовательских отчетов с улучшениями.

REST API для пользовательских отчетов КСК

Версия: 1.0
Дата: 26 октября 2025, 20:48

СПИСОК ФУНКЦИЙ API:

1. `GET /api/reports/user?status=all&limit=50&offset=0` - список пользовательских отчетов
2. `POST /api/reports/user` - создание пользовательского отчета
3. `DELETE /api/reports/user/{report_id}` - удаление пользовательского отчета
4. `GET /api/reports/user/{report_id}/status` - получение статуса отчета
5. ✨ **УЛУЧШЕНИЕ:** `GET /api/reports/user/types` - список доступных типов отчетов
6. ✨ **УЛУЧШЕНИЕ:** `GET /api/reports/user/{report_id}/data` - получение данных готового отчета

7. ✨ **УЛУЧШЕНИЕ:** `GET /api/reports/user/{report_id}/export/{format}` - экспорт отчета

ФУНКЦИЯ 1: Список пользовательских отчетов

REST API:

```
GET /api/reports/user?status=all&limit=50&offset=0
```

Параметры запроса:

- `status` (опционально): `all` (по умолчанию), `created`, `in_progress`, `done`, `error`
- `limit` (опционально): количество записей (по умолчанию 50, максимум 100)
- `offset` (опционально): смещение для пагинации (по умолчанию 0)

Источник данных:

- **Таблица:** `ksk_report_header`
- **JOIN:** с `ksk_report_orchestrator`
- **Условие:** WHERE `initiator = 'user'` AND `user_login = {current_user}`

Пример JSON ответа:

```
{
  "data": [
    {
      "id": 123,
      "report_code": "totals",
      "report_name": "Статистика",
      "status": "done",
      "created_datetime": "2025-10-25T14:30:00",
      "finished_datetime": "2025-10-25T14:30:45",
      "start_date": "2025-10-01",
      "end_date": "2025-10-25",
      "remove_date": "2025-11-08",
      "user_login": "ivanov",
      "parameters": {
        "list_codes": ["4200", "4204"]
      }
    },
    {
      "id": 124,
      "report_code": "figurants",
      "report_name": "Отчёт по фигурантам",
      "status": "in_progress",
      "created_datetime": "2025-10-26T10:15:00",
      "finished_datetime": null,
      "start_date": "2025-10-20",
      "end_date": "2025-10-26",
```

```
      "remove_date": "2025-11-02",
      "user_login": "ivanov",
      "parameters": null
    },
    {
      "id": 125,
      "report_code": "list_totals",
      "report_name": "Итоги по спискам",
      "status": "error",
      "created_datetime": "2025-10-26T09:00:00",
      "finished_datetime": "2025-10-26T09:00:15",
      "start_date": "2025-10-15",
      "end_date": "2025-10-25",
      "remove_date": "2025-11-02",
      "user_login": "ivanov",
      "parameters": null
    }
  ],
  "pagination": {
    "total_records": 15,
    "limit": 50,
    "offset": 0,
    "has_more": false
  }
}
```


ФУНКЦИЯ 2: Создание пользовательского отчета

REST API:

```
POST /api/reports/user
```

Тело запроса:

```
{
  "report_code": "figurants",
  "start_date": "2025-10-01",
  "end_date": "2025-10-25",
  "parameters": {
    "list_codes": ["4200", "4204", "2254"]
  }
}
```

Источник данных:

- **Функция:** `ksk_report_create()`
- **Действие:** Создаёт запись в `ksk_report_header` со статусом `created`
- **Триггер:** Асинхронно запускается фоновая задача для выполнения отчета

Пример JSON ответа:

```
{
  "id": 126,
  "report_code": "figurants",
  "report_name": "Отчёт по фигурантам",
  "status": "created",
  "created_datetime": "2025-10-26T20:48:30",
  "start_date": "2025-10-01",
  "end_date": "2025-10-25",
  "remove_date": "2025-11-02",
  "ttl": 7,
  "message": "Отчет создан и отправлен на выполнение"
}
```

HTTP коды:

- `201 Created` - отчет успешно создан
- `400 Bad Request` - некорректные параметры
- `404 Not Found` - неизвестный report_code

ФУНКЦИЯ 3: Удаление пользовательского отчета

REST API:

```
DELETE /api/reports/user/{report_id}
```

Источник данных:

- **Функция:** `ksk_cleanup_reports(specific_report_id)`
- **Действие:** Удаляет запись из `ksk_report_header` (каскадно удаляются данные отчета)

Пример JSON ответа:

```
{
  "message": "Отчет успешно удален",
  "deleted_report_id": 123
}
```

HTTP коды:

- `200 OK` - отчет удален
- `403 Forbidden` - попытка удалить чужой отчет
- `404 Not Found` - отчет не найден

ФУНКЦИЯ 4: Получение статуса отчета

REST API:

```
GET /api/reports/user/{report_id}/status
```

Источник данных:

- **Таблица:** `ksk_report_header`
- **Условие:** WHERE `id = report_id` AND `initiator = 'user'` AND `user_login = {current_user}`

Пример JSON ответа (в процессе):

```
{
  "id": 124,
  "status": "in_progress",
  "created_datetime": "2025-10-26T10:15:00",
  "finished_datetime": null,
  "progress_percentage": null,
  "message": "Отчет формируется..."
}
```

Пример JSON ответа (готов):

```
{
  "id": 123,
  "status": "done",
  "created_datetime": "2025-10-25T14:30:00",
  "finished_datetime": "2025-10-25T14:30:45",
  "duration_seconds": 45,
  "message": "Отчет готов"
}
```

Пример JSON ответа (ошибка):

```
{
  "id": 125,
  "status": "error",
  "created_datetime": "2025-10-26T09:00:00",
  "finished_datetime": "2025-10-26T09:00:15",
  "error_message": "Ошибка выполнения: timeout при обработке данных",
  "message": "Произошла ошибка при формировании отчета"
}
```

✨ ФУНКЦИЯ 5: Список доступных типов отчетов (УЛУЧШЕНИЕ)

REST API:

```
GET /api/reports/user/types
```

Источник данных:

- **Таблица:** `ksk_report_orchestrator`
- **Описание:** Возвращает список типов отчетов, доступных для пользовательского создания

Пример JSON ответа:


```
{
  "data": [
    {
      "report_code": "totals",
      "name": "Статистика",
      "description": "Общая статистика по транзакциям",
      "user_ttl": 14,
      "supports_parameters": false
    },
    {
      "report_code": "totals_by_payment_type",
      "name": "Статистика по типам платежей",
      "description": "Статистика с разбивкой по типам платежей",
      "user_ttl": 14,
      "supports_parameters": false
    },
    {
      "report_code": "list_totals",
      "name": "Итоги по спискам",
      "description": "Сводка по санкционным спискам",
      "user_ttl": 14,
      "supports_parameters": false
    },
    {
      "report_code": "list_totals_by_payment_type",
      "name": "Итоги по спискам и типам платежей",
      "description": "Детализация по спискам и типам платежей",
      "user_ttl": 14,
      "supports_parameters": false
    },
    {
      "report_code": "figurants",
      "name": "Отчёт по фигурантам",
      "description": "Детальная информация по фигурантам санкционных списков",
      "user_ttl": 7,
      "supports_parameters": true,
      "parameters_schema": {
        "list_codes": {
```

```
          "type": "array",
          "items": "string",
          "description": "Коды санкционных списков для фильтрации",
          "optional": true
        }
      }
    }
  ]
}
```

Зачем: Чтобы UI мог динамически отображать доступные типы отчетов без хардкода.

✨ ФУНКЦИЯ 6: Получение данных готового отчета (УЛУЧШЕНИЕ)

REST API:

```
GET /api/reports/user/{report_id}/data?limit=100&offset=0
```

Источник данных:

- **Зависит от типа отчета:**
  - `totals` → `ksk_report_totals_data`
  - `figurants` → `ksk_report_figurants_data`
  - и т.д.

Пример JSON ответа:

```
{
  "report_id": 123,
  "report_code": "figurants",
  "report_name": "Отчёт по фигурантам",
  "start_date": "2025-10-01",
  "end_date": "2025-10-25",
  "data": [
    {
      "list_code": "4200",
      "name_figurant": "Иванов Иван Иванович",
      "president_group": "Group A",
      "auto_login": "false",
      "exclusion_phrase": "не является",
      "total": 5,
      "total_allow": 3,
      "total_review": 1,
      "total_deny": 1,
      "total_bypass": 0
    }
  ],
  "pagination": {
    "total_records": 150,
```

```
    "limit": 100,
    "offset": 0,
    "has_more": true
  }
}
```

Зачем: Для отображения данных отчета в UI без скачивания файла.

✨ ФУНКЦИЯ 7: Экспорт отчета (УЛУЧШЕНИЕ)

REST API:

```
GET /api/reports/user/{report_id}/export/{format}
```

Параметры:

- `format`: `xlsx`, `csv`, `pdf` (опционально)

Примеры:

```
GET /api/reports/user/123/export/xlsx
GET /api/reports/user/123/export/csv
```

Ответ:

- **Content-Type:** зависит от формата
- **Content-Disposition:** `attachment; filename="report_figurants_2025-10-01_2025-10-25.xlsx"`

Зачем: Скачивание готового отчета в нужном формате.

ПРЕДЛОЖЕНИЯ ПО УЛУЧШЕНИЯМ:

1. ✅ Polling (опрос) статуса:

Проблема: UI регулярно опрашивает бэкенд
Решение: Используй короткие интервалы (2-3 секунды) только пока отчет в статусе `created` или `in_progress`. После `done`/`error` — прекращай опрос.

Пример логики на фронтенде:

```
function pollReportStatus(reportId) {
  const intervalId = setInterval(async () => {
    const response = await fetch(`/api/reports/user/${reportId}/status`);
    const data = await response.json();
```

```
    if (data.status === 'done' || data.status === 'error') {
      clearInterval(intervalId);
      updateUI(data);
    }
  }, 3000); // каждые 3 секунды
}
```

2. ✅ WebSocket (вместо polling):

Альтернатива: Использовать WebSocket для real-time обновления статуса.

```
const ws = new WebSocket('wss://api.example.com/reports/ws');

ws.onmessage = (event) => {
  const data = JSON.parse(event.data);
  if (data.report_id === 123) {
    updateReportStatus(data);
  }
};
```

Плюсы: Меньше нагрузка на сервер, мгновенные обновления.

3. ✅ Фильтрация по датам в списке отчетов:

```
GET /api/reports/user?status=all&created_from=2025-10-01&created_to=2025-10-26
```

4. ✅ Массовое удаление отчетов:

```
DELETE /api/reports/user/batch
```

Тело запроса:

```
{
  "report_ids": [123, 124, 125]
}
```

5. ✅ Валидация дат при создании:

Проверки на бэкенде:

- `start_date <= end_date`
- `end_date <= CURRENT_DATE` (нельзя создать отчет на будущее)
- `end_date - start_date <= 365 days` (ограничение диапазона)

6. ✅ Уведомления:

Опция: Отправлять email/Telegram уведомление когда отчет готов.

Дополнительное поле в запросе:

```
{
  "report_code": "figurants",
  "start_date": "2025-10-01",
  "end_date": "2025-10-25",
  "notify_on_complete": true
}
```

ИТОГОВАЯ ТАБЛИЦА API:

Метод	Endpoint	Описание	Пагинация
GET	`/api/reports/user`	Список отчетов	✅ Да
POST	`/api/reports/user`	Создать отчет	❌ Нет
DELETE	`/api/reports/user/{id}`	Удалить отчет	❌ Нет
GET	`/api/reports/user/{id}/status`	Статус отчета	❌ Нет
GET	`/api/reports/user/types`	Типы отчетов	❌ Нет
GET	`/api/reports/user/{id}/data`	Данные отчета	✅ Да
GET	`/api/reports/user/{id}/export/{format}`	Экспорт	❌ Нет

Готово к реализации!

⁂

1. ksk_reports_system.sql