Все данные

TL;DR: по документации песочницы OpenBanking Russia вы можете легально и технически собрать в БД *всё*, что возвращают API:

1. идентификаторы пользователя и связку с банками,
2. счета, балансы и транзакции,
3. продукты и договоры (кредиты, карты, вклады),
4. согласия (на счета и на продукты),
5. свои аналитические метрики поверх всего этого.
   Ниже — по блокам, «что именно» можно складировать.

---

0. Что я считаю «данными о юзере»

В контексте хакатона у вас есть:

- ваш **внутренний пользователь** (user_id в вашем приложении);
- один или несколько **тестовых клиентов песочницы** (`client_id` вида `team260-3` и т.п.);
- набор **банков** (VBank, ABank, SBank) с OpenBanking API.

«Данные о юзере» — это всё, что позволяет:

- идентифицировать этого клиента в песочнице и банках;
- описать его финансовое состояние: счета, балансы, транзакции, продукты, долги;
- хранить историю согласий и действий.

---

1. Идентификация и техсвязка с банками

1.1. Идентификаторы

Из документации и API вы точно можете хранить:

- `client_id` клиента в песочнице — то, что вы передаёте в `client_id` при межбанковских запросах (`GET /accounts?client_id=...`). (abank.open.bankingapi.ru)
- ID вашей команды `team260` как **запрашивающий банк** (`X-Requesting-Bank`). (abank.open.bankingapi.ru)
- Внутренний `user_id` в вашей системе и маппинг `user_id (app) ⇔ client_id (sandbox) ⇔ bank_id (vbank/abank/sbank)`.

Это всё имеет смысл класть в отдельную таблицу «профили пользователя / банковские идентификаторы».

1.2. Токены и аутентификация

По `POST /auth/bank-token` вы получаете: (abank.open.bankingapi.ru)

- `access_token`
- `token_type`
- `client_id` (ID вашей команды)
- `expires_in`

Технически вы можете их хранить в БД, но по хорошей практике:

- `client_secret` **не храните** в открытом виде (это из памятки команды, не из API);
- `access_token` либо не сохраняете, либо шифруете/кладёте в KV-хранилище, а в основной БД храните только метаданные (время истечения, тип, банк).

---

2. Счета клиента (Accounts API)

Из `GET /accounts` документация явно говорит, что в ответе есть минимум: (abank.open.bankingapi.ru)

- `account_id` — уникальный идентификатор счёта;
- `currency` — валюта (RUB, USD, EUR);
- `account_type` — тип счёта (Personal, Business и т.п.);
- `nickname` — «имя» счёта для UI;
- `servicer` — информация о банке (идентификатор, название и т.п.).

Плюс из логов песочницы (которые вы уже видели у себя) там ещё есть, например:

- статус счёта (`status`);
- дата открытия (`openingDate`);
- детальные реквизиты (`account` → `schemeName`, `identification`, `name` – ФИО клиента и номер счёта).

Что точно можно и нужно хранить в БД по счетам:

- Привязку к юзеру: `user_id`, `client_id`, `bank_id`;
- Идентификаторы счёта: `account_id`, внутренний номер счёта (`identification`);
- Тип и характеристики: `account_type`, возможно подтип, `currency`, `status`, `openingDate`, `nickname`;
- Данные о банке (из `servicer`): BIC/код банка, название (`bank_name`), возможно тип банка;
- Имя владельца счёта (ФИО) — приходит в `name` внутри реквизитов счёта (это фактически единственный кусок PII в песочнице уровня ФИО).

---

3. Баланс по счёту (Balances)

`GET /accounts/{account_id}/balances` возвращает данные по балансу конкретного счёта. Из описания видно, что это типичный OB Russia баланс: (abank.open.bankingapi.ru)

Как минимум вы можете хранить:

- `account_id` (связка с таблицей счетов);
- `amount` — размер остатка;
- `currency`;
- `balanceType` — тип баланса (обычно что-то вроде ClosingBooked, Expected и т.п.);
- `referenceDate` / `referenceDateTime` — дата, на которую актуален баланс;
- возможно, `creditLine` / лимиты (если банк их отдаёт).

Практически:

- Делаете таблицу `balances` или нормализуете в `accounts_balances_history`;
- Сохраняете **историю** балансов по датам — это база для расчёта динамики, STS и индекса фин.здоровья.

---

4. История транзакций (Transactions)

`GET /accounts/{account_id}/transactions` описан как выдающий **список транзакций** по счёту с пагинацией по `page` и `limit`. (abank.open.bankingapi.ru)

Документация не расписывает поля транзакции прямо в description, но по стандарту OpenBanking Russia и типовой модели OB-транзакций вы там увидите (через Swagger / фактический JSON):

- `transaction_id` — уникальный ID операции;
- даты: `bookingDateTime`, возможно `valueDateTime`;
- `amount` + `currency`;
- направление: `credit_debit_indicator` (списание/зачисление);
- текст описания / `remittanceInformation` (назначение платежа);
- коды/категории: `bankTransactionCode`, возможно MCC и др.;
- данные контрагента (обычно сильно обезличенные: маскированный счёт/карта, имя).

В БД вы можете хранить:

- всё вышеперечисленное как есть;
- ссылку на счёт `account_id` и, через него, на `user_id`;
- технические поля: источник (VBank/ABank/SBank), время загрузки, номер страницы при импорте.

И уже поверх этого строить свои категории, кластеры расходов, регулярные платежи и т.д.

---

5. Продукты и договоры (Products / Product Agreements)

В документации есть отдельный блок по **каталогу продуктов** и **договорам**:

- `/products` — каталог продуктовых предложений (депозиты, кредиты, карты и т.д.);

- `/products/{product_id}` — детали конкретного продукта; (abank.open.bankingapi.ru)
- `/product-agreements` и `/product-agreements/{agreement_id}` — договоры клиента по продуктам (вклад, кредит, карта). (abank.open.bankingapi.ru)

5.1. Каталог продуктов (`/products`)

В ответе по каждому продукту у вас типично будут:

- `product_id`;
- `product_type` (deposit / credit / card и т.п.);
- название продукта;
- базовые условия: ставка, срок, минимальная сумма, комиссия, кэшбэк и т.п.

Это **не персональные** данные, но вы можете хранить:

- справочник всех продуктов с полными условиями;
- маппинг, какие продукты доступны как предложения конкретному юзеру (если реализуете персонизацию).

5.2. Договоры (`/product-agreements`)

Это уже чисто пользовательская история: открытые вклады, кредиты, карты. Из описаний видно, что:

- договор создаётся через `POST /product-agreements` с телом `ProductAgreementRequest`;
- кредитный скоринг и лимиты зависят от сегмента клиента, суммарного долга и т.д. (abank.open.bankingapi.ru)

По договору вы сможете хранить (по факту ответа):

- `agreement_id`;

- `product_id` + `product_type`;
- `client_id` и, через него, `user_id`;
- ссылку на счёт договора (`account_id`);
- статус договора (active, closed, overdue и т.п.);
- ключевые параметры продукта для этого клиента:
  - сумма основного долга / лимита;
  - текущий остаток долга;
  - ставка (%);
  - дата начала / окончания;
  - график платежей (дата ближайшего платежа, сумма, минимум и т.д.);
  - признак просрочки и данные о ней (если отдаются API).

Это именно те данные, которые вам нужны для анализа долговой нагрузки, PDN, STS и всего, что вы строите в своём приложении.

---

6. Согласия (Consents) — на счета и на продукты

У вас два больших семейства согласий:

1. **Согласия на доступ к счетам** — `/account-consents` и `/account-consents/request`;
2. **Согласия на управление договорами** — `/product-agreement-consents`. (abank.open.bankingapi.ru)

6.1. Account consents

Тело запроса на согласие по счетам: (abank.open.bankingapi.ru)

```
{
  "client_id": "team200-1",
  "permissions": [
    "ReadAccountsDetail",
    "ReadBalances",
    "ReadTransactionsDetail"
  ],
  "reason": "",
  "requesting_bank": "test_bank",
  "requesting_bank_name": "Test Bank"
}
```

И ответ по согласиям (`GET /account-consents/{consent_id}` / список) содержит:

- `consentId`;
- `status` (pending, approved, rejected, revoked);
- `creationDateTime`, `statusUpdateDateTime`;
- `permissions` — набор прав;
- `expirationDateTime`.

Все эти поля можно и нужно логировать в БД:

- история выданных юзером согласий;
- какие права он дал;
- до какой даты живёт согласие;
- откуда вы к нему обращались (каким банком/командой).

6.2. Product-agreement consents

Из описания `POST /product-agreement-consents/request`: (abank.open.bankingapi.ru)

Вы можете хранить по каждому согласованию:

- `client_id`;
- `requesting_bank` (ваша команда);
- уровень прав:
  - `read_product_agreements`;
  - `open_product_agreements`;
  - `close_product_agreements`;
- ограничения:
  - `allowed_product_types` (deposit/card/credit и т.п.);
  - `max_amount` — максимальная сумма продукта;
  - `valid_until`;
- `reason` — зачем вы это спрашивали.

И ровно так же — `consent_id`, статус, дата создания/обновления.

---

7. Технические и поведенческие данные вашего приложения

Это уже **ваши** данные, но они напрямую связаны с юзером и разрешены документацией:

- история логинов/сессий (по вашему `user_id`);
- какие банки он подключил (флаг `is_connected` по каждому bank_id);
- когда вы в последний раз обновляли:
  - список счетов;
  - историю транзакций;
  - список договоров;
- какие действия совершал в UI:
  - клик по офферу продукта;

- подача заявки на новый продукт (связь с `product_id` / `agreement_id`);
- изменение настроек (лимиты, цели, уведомления).

Документация API никак это не запрещает — это логика вашего сервиса.

---

8. Аналитика иDerived-метрики

Самое вкусное — это **производные** данные, которые вы можете посчитать на основе всего выше:

- агрегированные балансы по всем банкам (общий кэш, активы, долги);
- ваш **коэффициент финансового здоровья**;
- **Save-to-Spend (STS)** — «сколько можно безопасно потратить сегодня»;
- прогнозы поступлений и расходов по категориям;
- ближайшие обязательные платежи (кредиты, ЖКХ и т.п.);
- PDN/отношение платежей по долгам к доходу (даже если доход вы оцениваете по транзакциям).

Эти метрики вы можете смело хранить в отдельных таблицах (например, `user_metrics_daily`), с привязкой к дате пересчёта и версии алгоритма.

---

9. Что *лучше* не хранить или хранить очень аккуратно

Чисто чтобы не подставиться ни перед судьями, ни перед здравым смыслом:

- `client_secret` вашей команды — только в env/secret-хранилище, **не** в обычной БД;
- `access_token` и refresh-токены — по возможности не хранить, либо шифровать и ограничивать доступ;
- полные реквизиты (полный PAN карты, полный номер счёта) — в песочнице это и так, как правило, маскировано, но если где-то «проскочит» — лучше маскировать перед сохранением.

Если хочешь, дальше могу:

- превратить это в конкретную схему БД (таблицы + поля),
- или разложить: какие из этих данных реально дают вам преимущество на питче перед жюри, а что можно не делать в MVP.