



PG BootCamp Russia 2025 Ekaterinburg

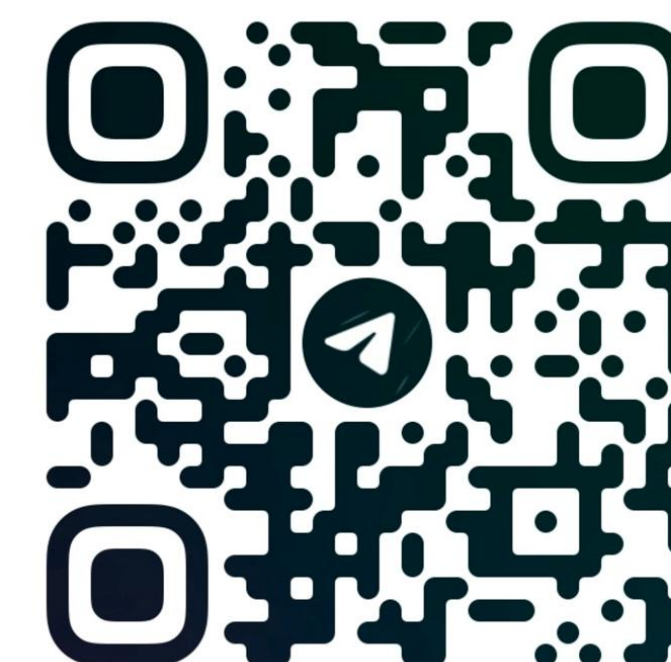
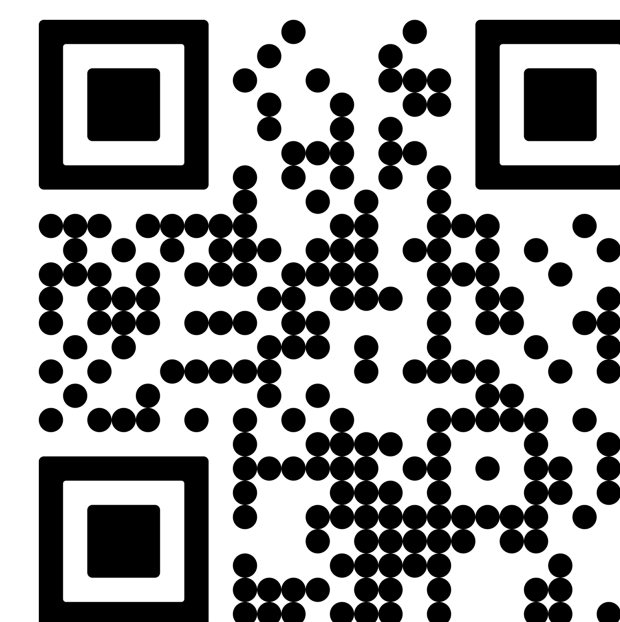
PGBootCamp.ru

Превращаем PostgreSQL в RestFull API Server

Константин Ващенко

kv@xsquare.ru

xsquare.ru



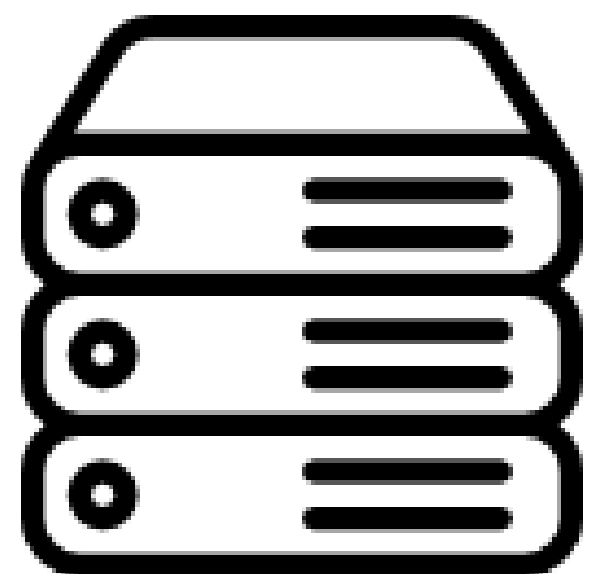
@XSQUARE365

Разработка приложений

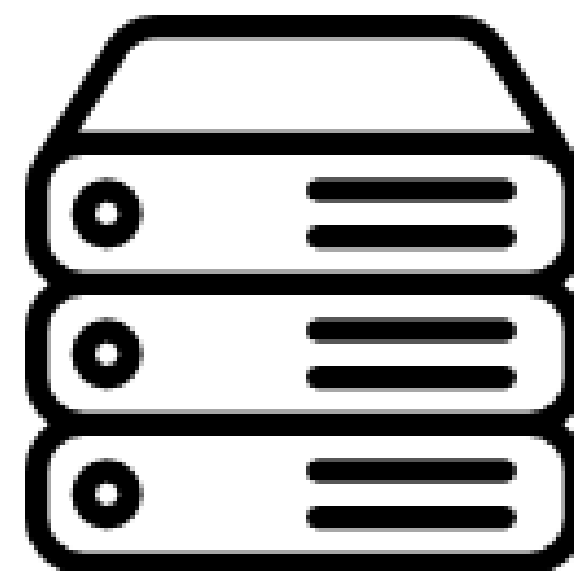
Web



HTTP
SERVER



APPLICATION
SERVER



DB
SERVER



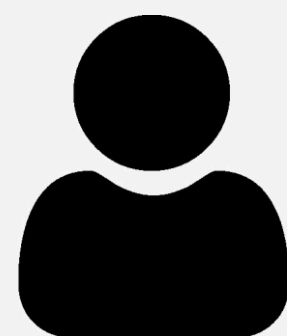
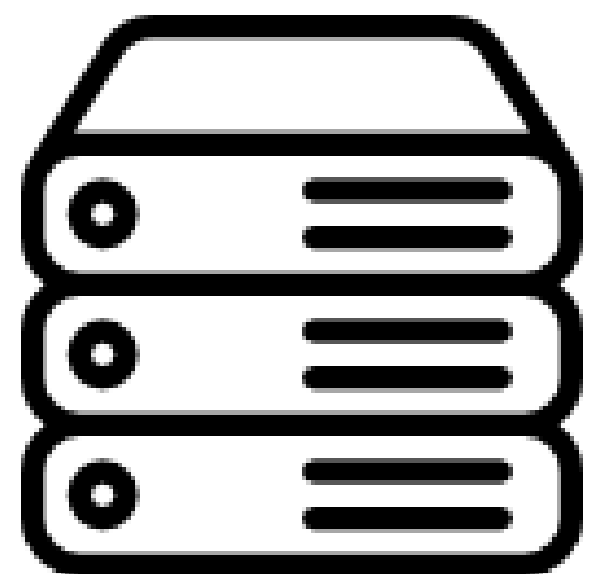
Разработка приложений

Web

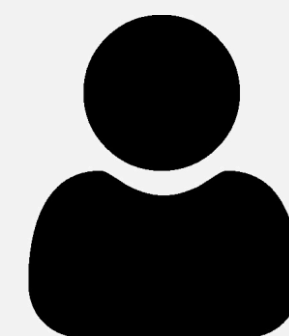
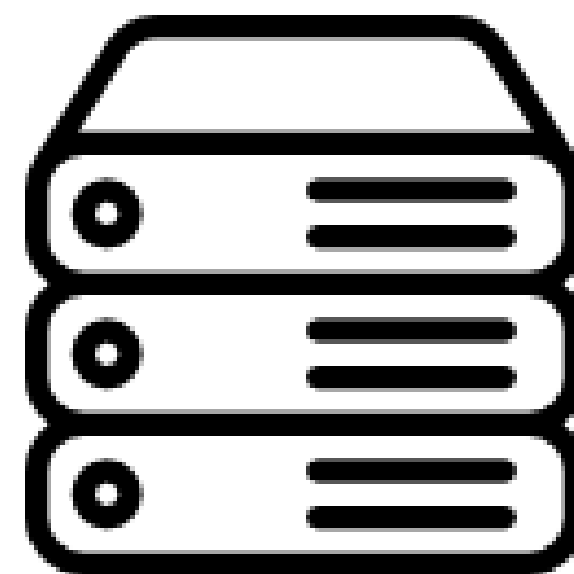


Разработка:

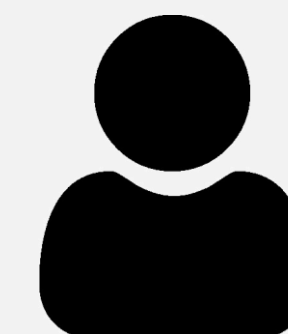
HTTP
SERVER



APPLICATION
SERVER



DB
SERVER



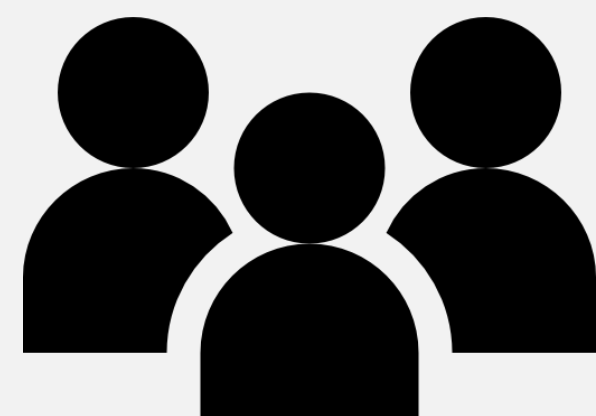
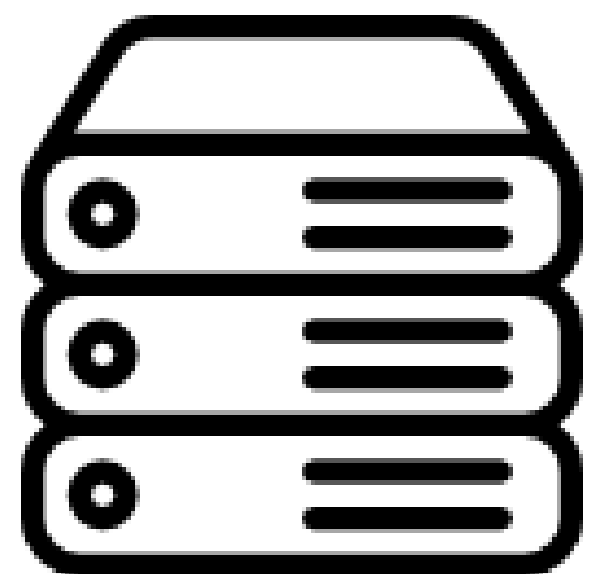
Разработка web-приложений

Web

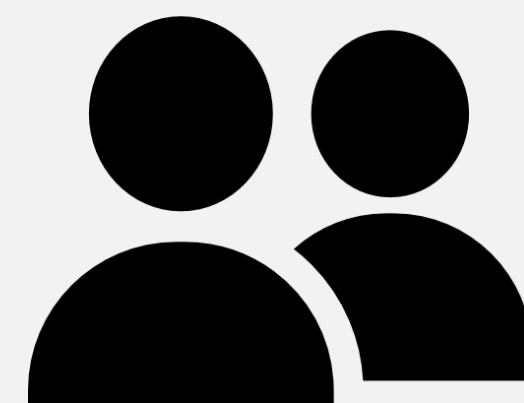
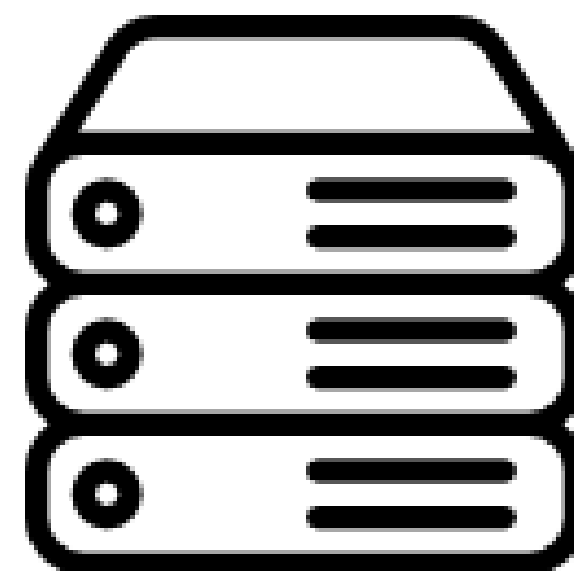


Разработка:

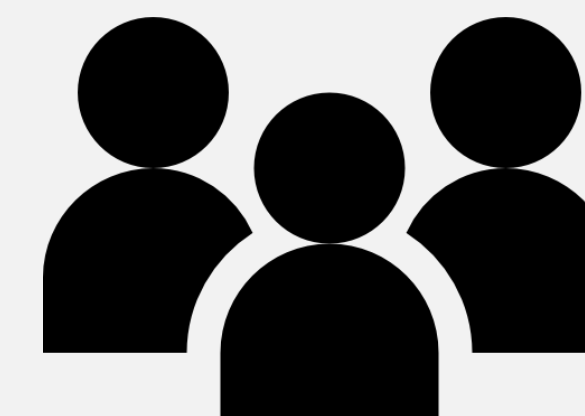
HTTP
SERVER



APPLICATION
SERVER



DB
SERVER



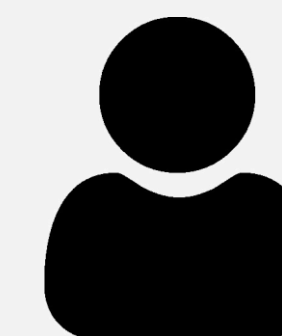
PostgreSQL и внешний мир



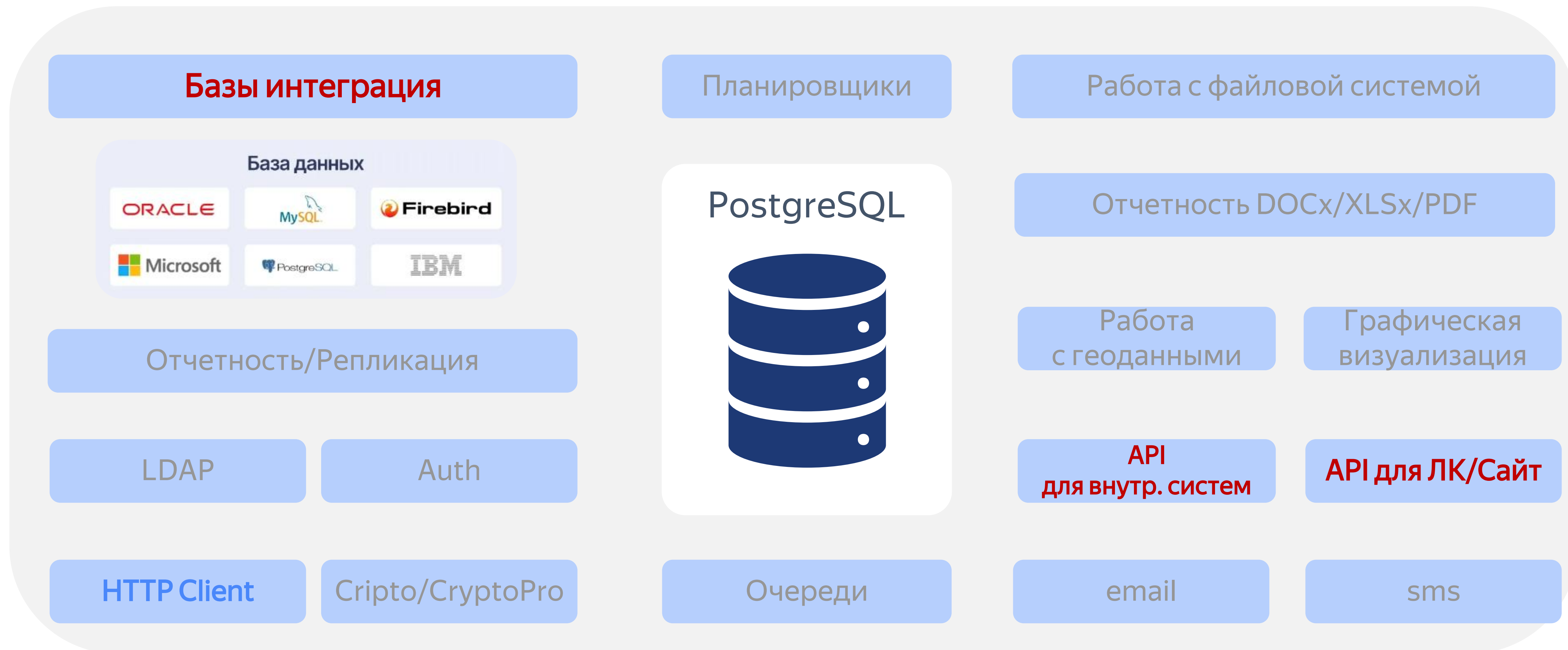
Как
подключать



PostgreSQL
SERVER



Типовые задачи OLTP-приложения



Расширения. Сила PostgreSQL в расширяемости!

Миграция

- ora2pg
- orafce
- ora_migrator

Планировщики

- pg_cron
- pg_timetable

Расширения

- http
- pgmail
- pgcrypto
- postGIS
- uuid-oss
- hstore
- pg_trgm
- pg_stat_statements

и др.

Обёртки внешних данных

- dblink_fdw
- oracle_fdw
- tds_fdw

Загружаемые процедурные языки

- PL/Python
- PL/Perl
- PL/Tcl

Расширения. Проблема №1

Обёртки внешних данных

- `dblink_fdw`
- `oracle_fdw`
- `tds_fdw`
- и т.д.

1. Базы все разные

2. Зависимость от библиотек

Просто не можем обновиться.
Нужно компилировать снова и снова

3. Отсутствие пула соединений

4. Кодировки

tds_FDW for MS SQL. Кодировки

```
int tds_err_handler(DBPROCESS *dbproc, int severity, int dberr,
{
    #ifdef DEBUG
        ereport(NOTICE,
            (errmsg("----> starting tds_err_handler")
            ));
    #endif

    ereport(ERROR,
        (errcode(ERRCODE_FDW_UNABLE_TO_CREATE_EXECUTION),
        errmsg("%s", tds_err_msg(severity, dberr, oserr, dberrstr, c
        ));

    return INT_CANCEL;
}
```

```
4088
4089 int tds_err_handler(DBPROCESS *dbproc, int severity, int dberr, int
4090 {
4091     #ifdef DEBUG
4092         ereport(NOTICE,
4093             (errmsg("----> starting tds_err_handler")
4094             ));
4095     #endif
4096
4097+     if dberr == 2403
4098+     {
4099+         return 0;
4100+     }
4101+     else
4102+     {
4103         ereport(ERROR,
4104             (errcode(ERRCODE_FDW_UNABLE_TO_CREATE_EXECUTION),
4105             errmsg("%s", tds_err_msg(severity, dberr, oserr, dberrstr, c
4106             ));
4107+     }
4108
4109     return INT_CANCEL;
4110 }
```

Расширения PL/Языки. Проблема №2

PL/Python – Все может. Но безумно медленный и ресурсоемкий.

PL/Perl – Все может. Работает на уровне ОС и опасен

PostgreSQL и JSON

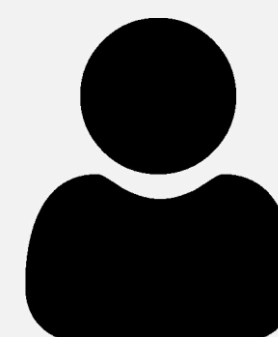
Web



от SQL к

JSON

Postgres



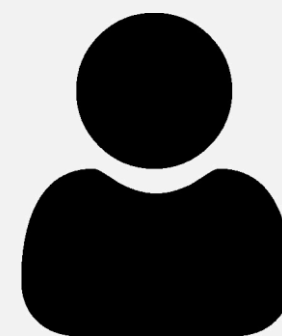
Расширения. Решение №1

Postgres



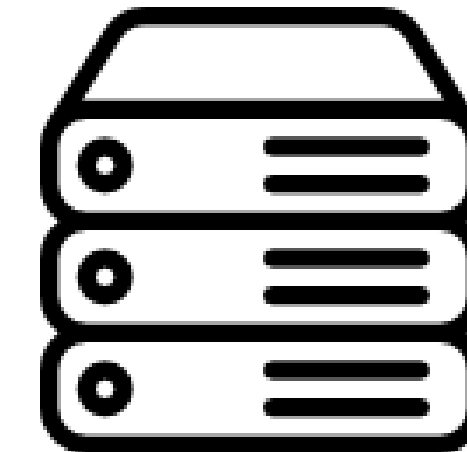
решаем транспортную задачу
исходящего трафика

pgsql-http



только SQL+PL/pgSQL

API



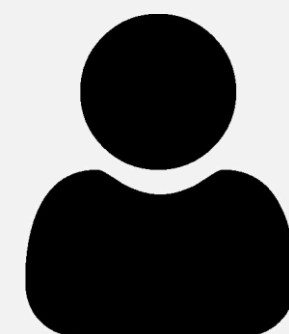
Расширения. Решение №2

Postgres



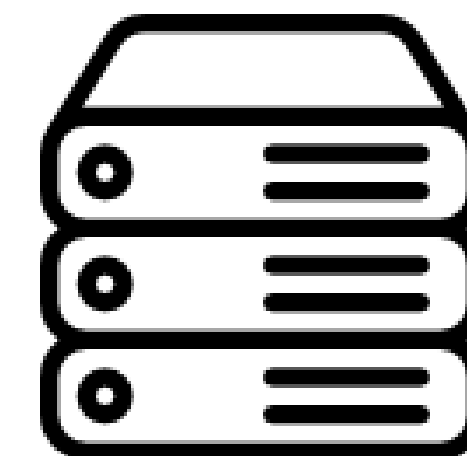
решаем транспортную задачу
входящего трафика

xDAC



только SQL+PL/pgSQL

API



Публикуем данные. Пример N°1

```
-- Функция в PostgreSQL
CREATE OR REPLACE FUNCTION api.f_hello()
  RETURNS json LANGUAGE plpgsql
AS $function$
BEGIN
  RETURN '{"GET_RETURN": "Hello world!!"}'::json;
END;
$function$;
```

```
curl "http://127.0.0.1:8887/xdac/rpc/f_hello" -X GET
```

Публикуем данные. Пример №2

```
--Функция в PostgreSQL с передачей параметров
CREATE OR REPLACE FUNCTION api.f_concat_str(p_first_str character varying DEFAULT '',
                                             p_second_str character varying DEFAULT '')
    RETURNS json LANGUAGE plpgsql
AS $function$
DECLARE
    l_result VARCHAR;
BEGIN
    l_result := CONCAT(p_first_str,p_second_str);

    RETURN ('{"CONCAT_RESULT": "' || l_result || '"}'):JSON;
END;
$function$;
```

```
curl -H "Content-Type: application/json" -d '{"p_first_str": "", "p_second_str": ""}' "http://127.0.0.1:8887/xdac/rpc/f_concat_str"
```


Публикуем данные. Пример №3

```
-- Функция в PostgreSQL с передачей JSON или JSONB
CREATE OR REPLACE FUNCTION api.f_json_parse(p_json json)
  RETURNS json LANGUAGE plpgsql
AS $function$
DECLARE
BEGIN
  RETURN ('{"JSON_OUTPUT": ' || p_json::TEXT || '}')::JSON;

EXCEPTION WHEN OTHERS
THEN
  RETURN 'Некорректный формат JSON!';
END;
$function$;
```

```
curl -H "Content-Type: application/json" -H "Prefer: params=single-object"
-d '{"first_value": 1, "second_value": "two", "third_value": [1, "two", "3"]}' "http://127.0.0.1:8887/xdac/rpc/f_json_parse"
```


Публикуем данные. Пример N°4

```
--Функция в PostgreSQL. Получаем в JSON сведения о клиенте по ID из списка клиентов

CREATE OR REPLACE FUNCTION api.f_get_client(p_id bigint)
  RETURNS json LANGUAGE plpgsql
AS $function$
  DECLARE
    l_client_info json;
  BEGIN
    SELECT json_build_object('fi',          CONCAT(c.last_name, ' ', c.first_name),
                             'date_of_birth', to_char(c.date_of_birth, 'dd.mm.yyyy'),
                             'phone',        c.phone,
                             'email',        c.email)
    INTO l_client_info
    FROM cli.t_clients c
    WHERE c.id = p_id;

    RETURN l_client_info;
  END;
$function$;
```

```
curl "http://127.0.0.1:8887/xdac/rpc/f_get_client?p_id=2" -X GET
```

Публикуем данные. Пример N°5

```
-- Функция в PostgreSQL. Пример массовой выгрузки из таблицы
CREATE OR REPLACE FUNCTION api.f_get_mass()
  RETURNS json LANGUAGE plpgsql
AS $function$
  DECLARE
    l_mass_clients JSON;
  BEGIN

    WITH tbl AS (
      SELECT c.last_name, c.first_name, c.date_of_birth, c.email, c.phone
      FROM cli.t_clients c
      LIMIT 3
    )
    SELECT JSON_AGG(tbl.*) as mass_clients
    INTO l_mass_clients
    FROM tbl;

    RETURN l_mass_clients;

  END;
$function$;
```

```
curl "http://127.0.0.1:8887/xdac/rpc/f_get_mass" -X GET
```

Публикуем данные. Пример N°6 или для ленивых

```
1  -- Функция в PostgreSQL или для ленивых
2  ✓ create or replace function api.f_get_user_xml() returns bytea
3      language plpgsql
4      as
5      $$
6      declare
7          L_mass_clients text;
8      begin
9          select query_to_xml('select * from users.t_user_auth_log LIMIT 8',false,false,'')::text into L_mass_clients;
10         return L_mass_clients::bytea;
11     end;
12     $$;
```

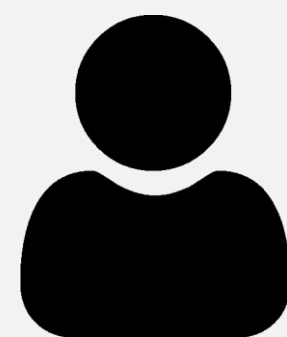
```
curl "http://127.0.0.1:8887/xdac/rpc/ f_get_user_xml" -X GET
```



Получаем данные. Пример N°1

решаем транспортную задачу
исходящего трафика. HTTP GET

pgsql-http



```
select content::json data  
from  
http_get('http://my_ip/xdac/f_hello') t
```

Получаем данные. Пример №2

Метод POST
и
Timeout

```
1  create function f_get_message_status(p_message_id character varying) returns json
2      language plpgsql
3  as
4  $$
5  DECLARE
6      l_session varchar(500);
7      l_data     varchar;
8      l_resp     http_response;
9      l_url      varchar;
10 BEGIN
11     l_session := sms.f_login();
12     l_data := 'sessionId=' || l_session ||
13             '&messageId=' || (p_message_id::json->>0);
14     l_url := pref.f$c1(p_code 'SMS_ADDRESS') || '/Sms/State';
15
16     perform http_set_curl_opt('CURLOPT_TIMEOUT', '300');
17     l_resp := http_post(l_url, l_data, 'application/x-www-form-urlencoded');
18     return l_resp.content::json;
19 END;
20 $$;
```

Получаем данные. Пример №3

```
root@lcdp4-edu.xsquare:~# curl -X POST -H "X-Forwarded-For: 127.0.0.1" -H "Cookie: session=123456789" -H 'Content-Type: application/json' --data '{
  "id": "000000000-0000-0000-0000-000000000000",
  "fileName": "example.txt",
  "fileContent": "UEsDBBBQACAgIA...",
  "mimeType": "text/plain",
  "type": "File"
}' http://localhost:9000/add_file
```

Передаёт **хедеры** через параметры конфигурации

```
CALL files.add_file(p_id      => (p_json->>'id')::uuid,
                    p_file_name => (p_json->>'fileName'),
                    p_file_content => decode((p_json->>'fileContent'),
                                             'base64'),
                    p_mime_type  => (p_json->>'mimeType'),
                    p_file_type  => (p_json->>'type'),
                    p_id_user    => (l_cookie->'session'));
```

HTTP-сервис принимает запрос и устанавливает **cookies**

```
l_header := current_setting('request.headers', true)::json;

l_ext := l_header->>'X-Forwarded-For';

l_arr := string_to_array(l_headers->>'Cookie', ';');

FOREACH l_entry IN ARRAY l_arr
LOOP
  l_arr_ent := string_to_array(l_entry, '=');
  l_cookie := l_cookie || hstore(trim(l_arr_ent[1]), trim(l_arr_ent[2]));
END LOOP;
```

Тело запроса как параметр вызова функции

FWD. Кандидат на вылет

Postgres



~~fdw~~

Postgres



Oracle



DB2



MS SQL



MySQL



MS Excel



✓ FWD. Postgre select * from Oracle

```
1  -- превращаем Oracle а таблицу в Postgres
2  ✓ WITH json_data AS (
3      select content::json data
4      from http_get( uri 'http://10.150.117.205:8890/xdac/rpc/F_GET_ORA_CLIENTS') t
5  )
6  SELECT
7      elem.value->>'id',
8      elem.value->>'last_name',
9      elem.value->>'first_name',
10     elem.value->>'email'
11 FROM
12     json_data,
13     json_array_elements(data) elem;
```

```
1  -- Делаем таблицу в Oracle в виде JSON
2  create or replace function xdac_demo.f_get_ora_clients return clob as
3      l_client_info clob;
4      begin
5          -- FUNCTION IN ORACLE
6
7          select JSON_ARRAYAGG(
8              json_object(
9                  key 'id' value to_char(id),
10                 key 'first_name' value first_name,
11                 key 'last_name' value last_name||' tantor',
12                 key 'email' value email
13             )
14          )
15          into l_client_info
16          from t_clients;
17
18
19
20     return l_client_info;
21 end;
```

```
curl "http://127.0.0.1:8887/xdac/rpc/ f_get_ora_clients" -X GET
```


FWD. Postgre select * from Postgre

```
2 ✓ WITH json_data AS (  
3     select content::json data  
4     | from http_get(uri 'http://10.150.117.205:8890/xdac/rpc/F_GET_ORA_CLIENTS') t  
5 )  
6 SELECT  
7     elem.value->>'id',  
8     elem.value->>'last_name',  
9     elem.value->>'first_name',  
10    elem.value->>'email'  
11 FROM  
12     json_data,  
13     json_array_elements(data) elem;
```

f_get_user_xml

```
1 -- Функция в PostgreSQL или для ленивых  
2 ✓ create or replace function api.f_get_user_xml() returns bytea  
3     language plpgsql  
9     select query_to_xml('select * from users.t_user_auth_log LIMIT 8',false,false,'')::text into L_mass_clients;  
10    return L_mass_clients::bytea;  
11 end;
```

FWD. select * from Postgre

```
2 ✓ WITH json_data AS (  
3     select content::json data  
4     | from http_get(uri 'http://10.150.117.205:8890/xdac/rpc/F_GET_ORA_CLIENTS') t  
5 )  
6 SELECT  
7     elem.value->>'id',  
8     elem.value->>'last_name',  
9     elem.value->>'first_name',  
10    elem.value->>'email'  
11 FROM  
12     json_data,  
13     json_array_elements(data) elem;
```

f_get_user_xml

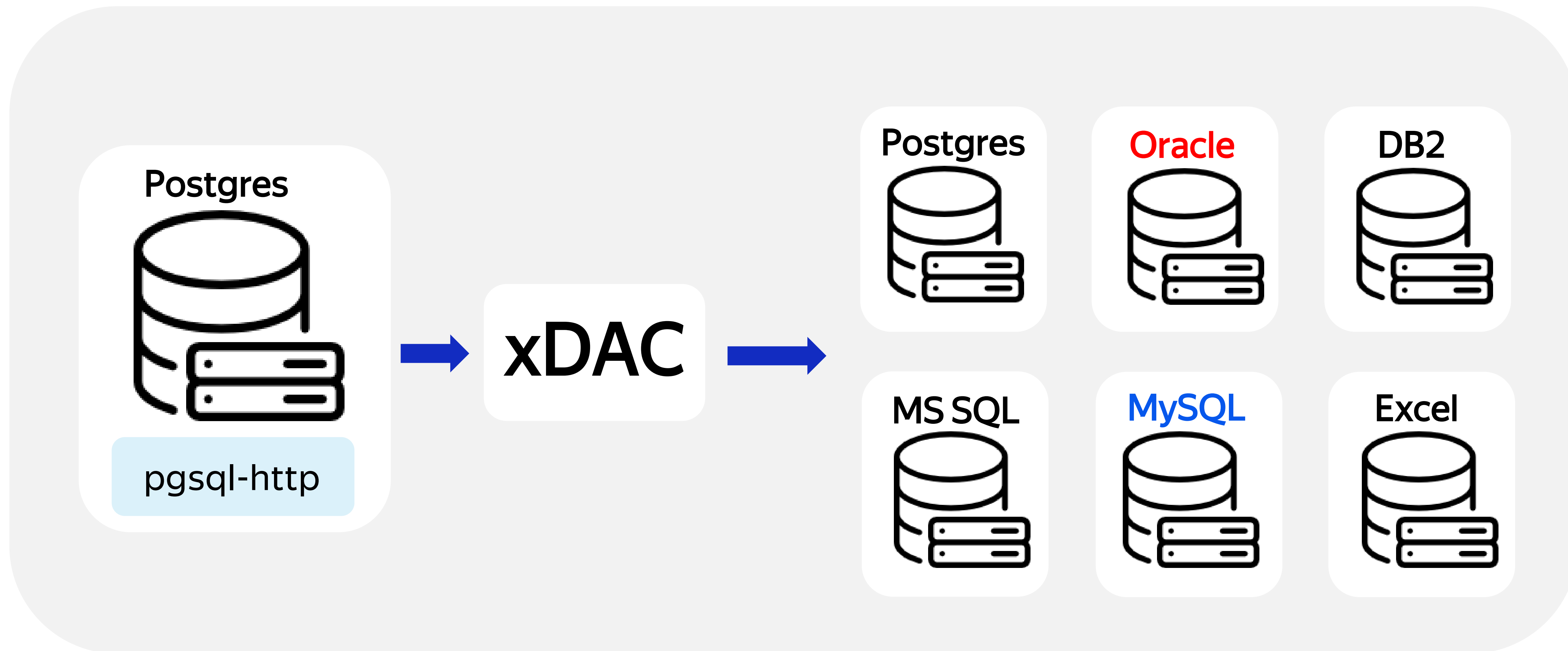
```
1 -- Функция в PostgreSQL или для ленивых  
2 ✓ create or replace function api.f_get_user_xml() returns bytea  
3     language plpgsql  
9     select query_to_xml('select * from users.t_user_auth_log LIMIT 8',false,false,'')::text into L_mass_clients;  
10    return L_mass_clients::bytea;  
11 end;
```


/ FWD. Поможем Oracle)))

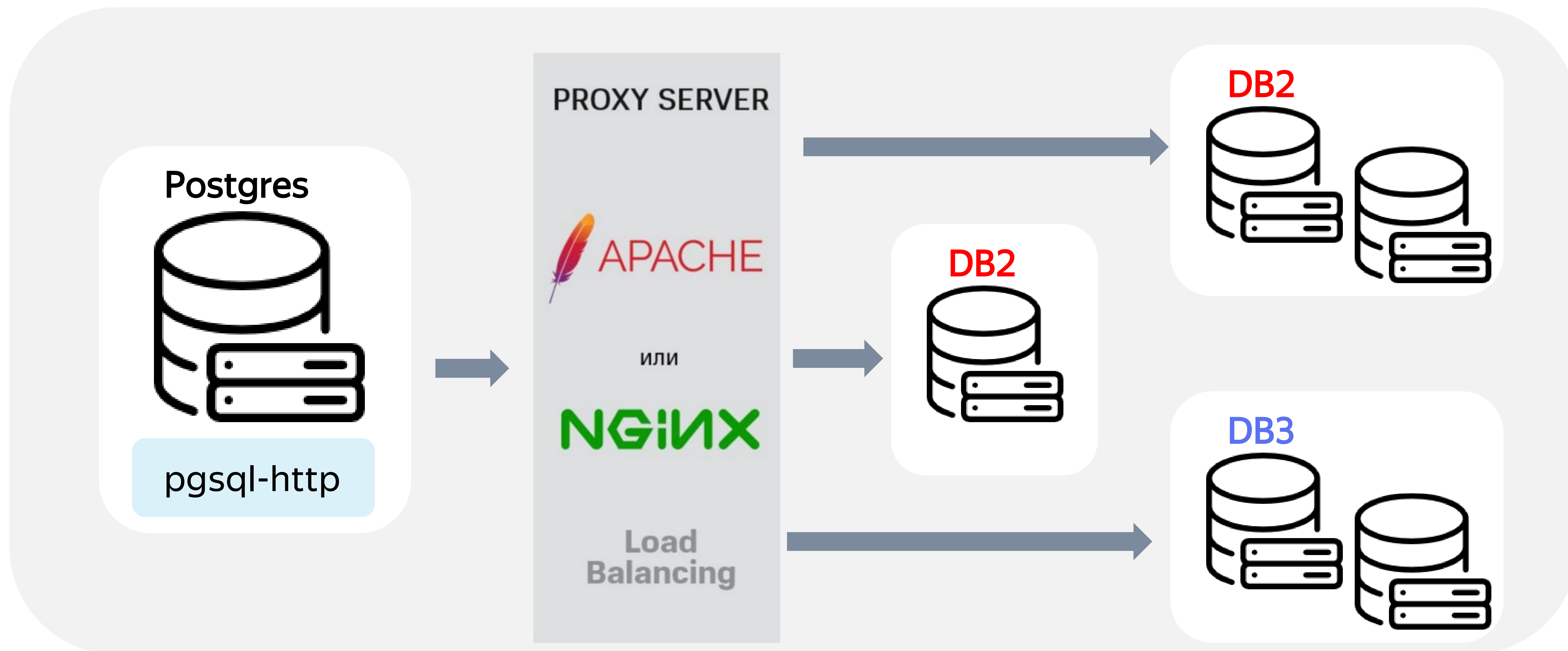
```
SQL Output Statistics
-- превращаем Postgres в таблицу в PostgreSQL
select x.*
from xmltable (
    '/table/row'
    passing xmltype(httpuritype.createUri('http://10.150.117.205:8887/xdac/rpc/f_get_user_xml').getblob(),nls_charset_id('AL32UTF8'))
    columns
        id varchar2(100) path 'id'
        ,id_user varchar2(100) path 'id_user'
        ,date_created varchar2(100) path 'date_created'
        ,id_addr varchar2(100) path 'id_addr'
) x
;
```

```
1 -- Функция в PostgreSQL или для ленивых
2 ✓ create or replace function api.f_get_user_xml() returns bytea
3     language plpgsql
9     select query_to_xml('select * from users.t_user_auth_log LIMIT 8',false,false,'')::text into L_mass_clients;
10    return L_mass_clients::bytea;
11 end;
```

FWD. Один ко многим

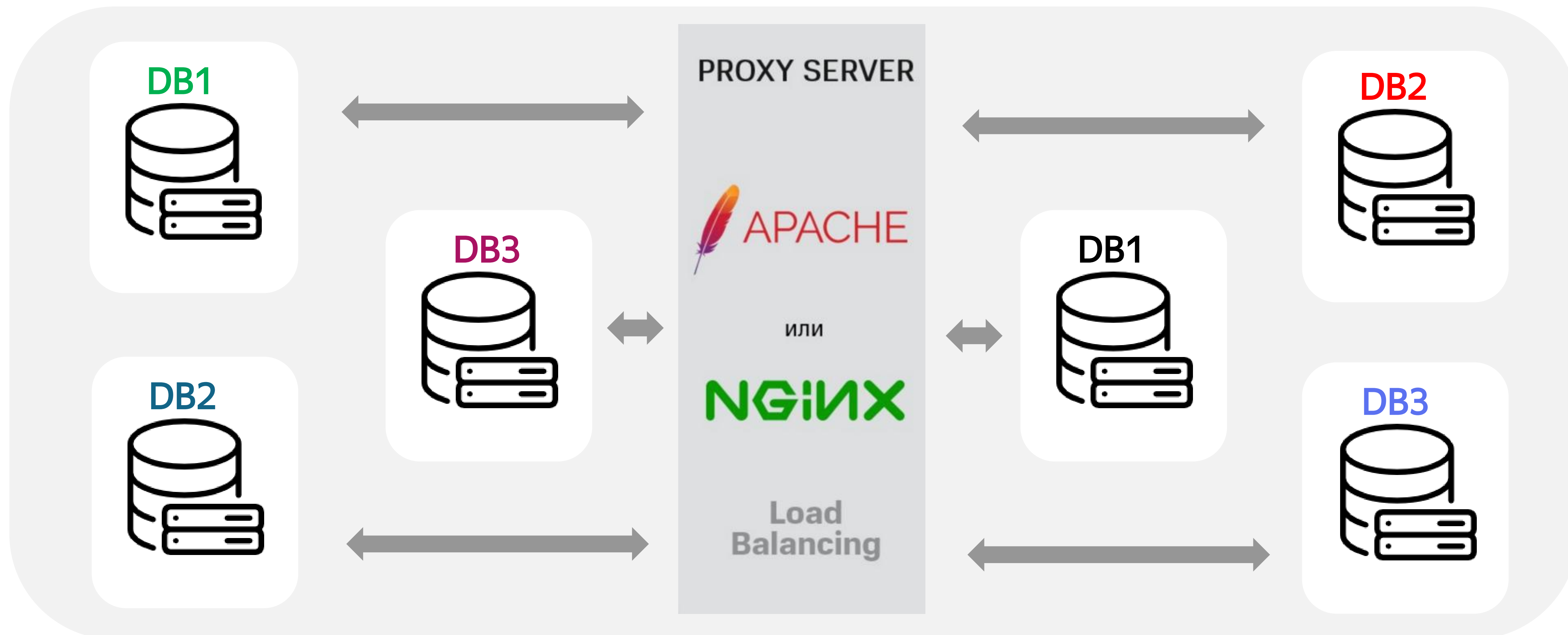


✓ FWD. Один ко многим с балансировкой



✓ FWD. Многие ко многим с балансировкой.

или как мы победили шардиннг

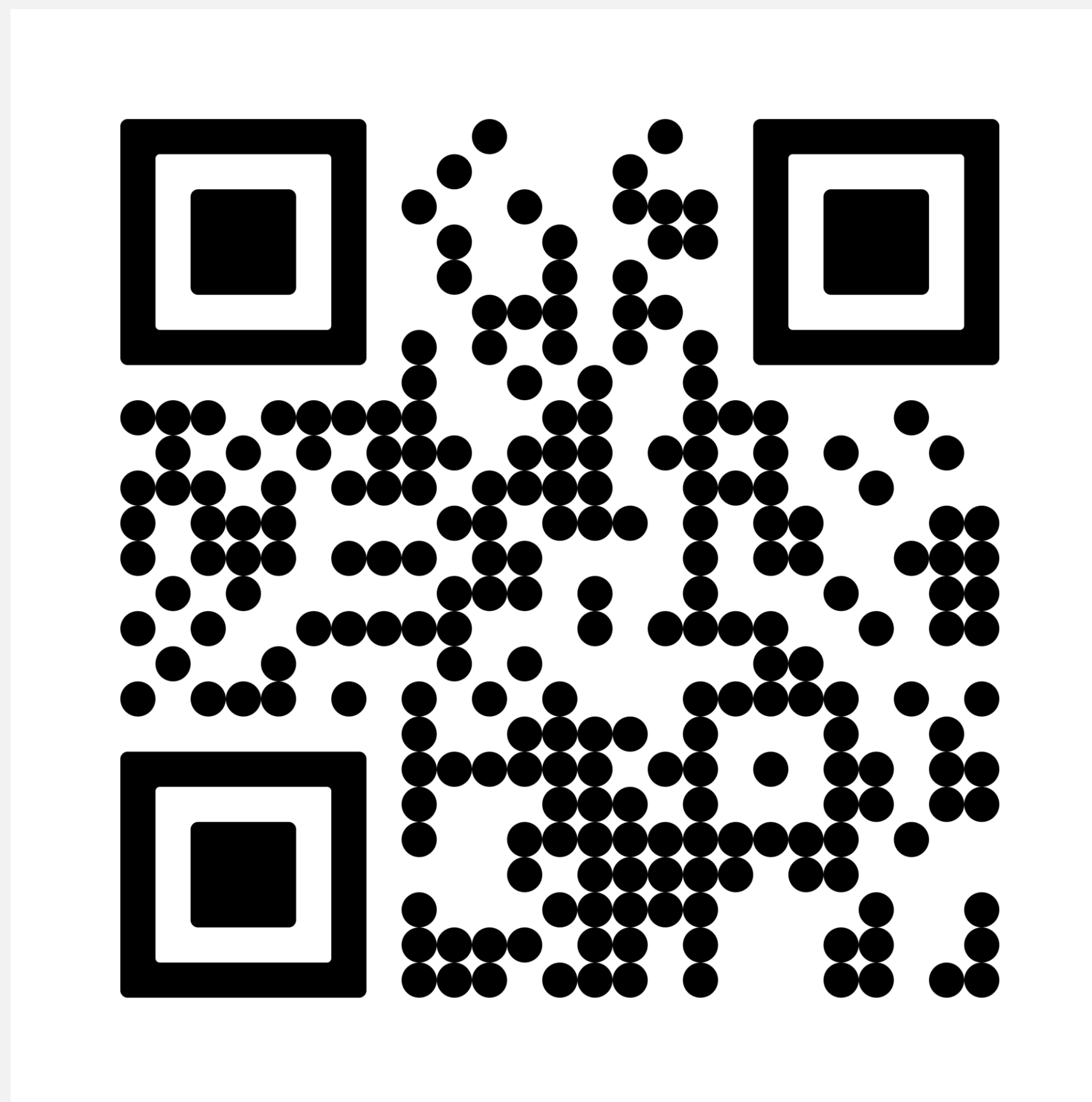


Практика. Подключаем Гос Услуги/ЕСИА

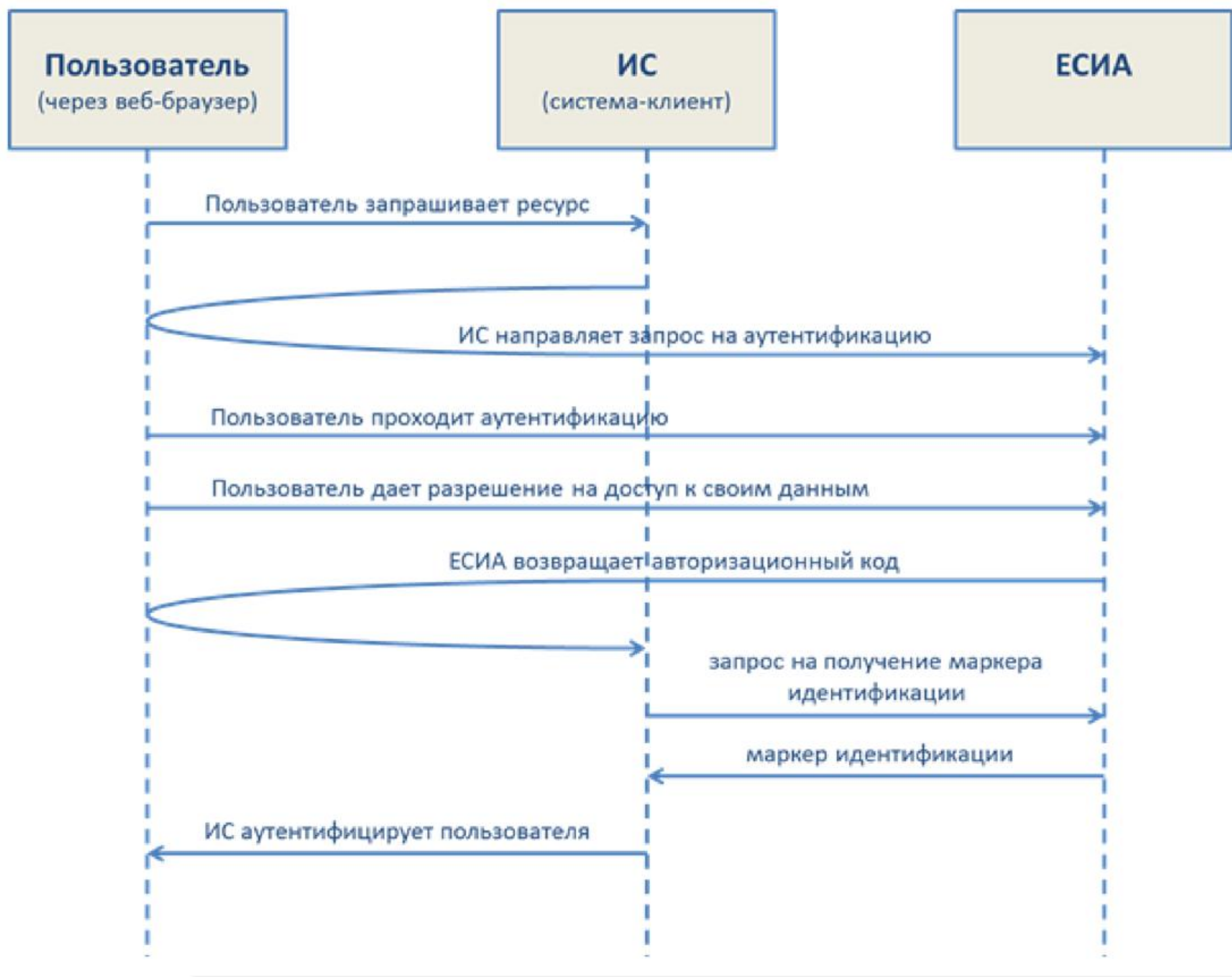
ЕДИНАЯ СИСТЕМА
ИДЕНТИФИКАЦИИ И АУТЕНТИФИКАЦИИ

Методические рекомендации
по использованию
Единой системы идентификации и аутентификации
Версия 2.84

Практика. Раскрываем коды клиента к ЕСИА



Практика. Протокол. Стр. 47



```
esia
├── tables 1
│   └── t_list
│       ├── columns 9
│       ├── keys 1
│       └── indexes 2
├── routines 9
│   ├── f_esia (esia.esia_resp): esia.t_list
│   ├── f_esia_create (timestamp): esia.t_list
│   ├── f_get_auth_url (uuid, varchar): text
│   ├── f_get_doc_info (varchar, varchar): http_response
│   ├── f_get_doc_list (varchar, varchar): http_response
│   ├── f_get_personal_info (varchar, varchar): http_response
│   ├── f_get_token (uuid, varchar): http_response
│   ├── f_gost_sign (bytea): text
│   └── p_set_error (integer, varchar, text)
├── sequences 1
│   └── 123 t_list_id_seq integer
├── object types 1
│   └── esia_resp
│       └── object attributes 4
│           ├── code varchar
│           ├── state uuid
│           ├── error varchar
│           └── error_description varchar
```


Практика. ЕСИА. AUTH

```
create function f_get_auth_url(p_state uuid, p_url character varying DEFAULT NULL::character varying) returns text
    language plpgsql
as
$$
DECLARE
    l_timestamp    VARCHAR(25);
    l_secret        TEXT;
    l_ret           TEXT;
BEGIN
    l_timestamp := TO_CHAR(CURRENT_TIMESTAMP,'YYYY.MM.DD HH24:MI:SS TZHTZM');
    l_secret := esia.f_gost_sign( p_data (CONCAT( pref.f$c1( p_code 'ESIA_SCOPE'),
                                                l_timestamp,pref.f$c1( p_code 'ESIA_CLIENT_ID')
                                                ,p_state::VARCHAR))::BYTEA);
    l_ret := CONCAT(pref.f$c1( p_code 'ESIA_AUTH')
                    , '?scope=',urlencode(pref.f$c1( p_code 'ESIA_SCOPE'))
                    , '&redirect_uri=',urlencode(COALESCE(p_url,pref.f$c1( p_code 'ESIA_RETURN_URL'))))
                    , '&client_id=',pref.f$c1( p_code 'ESIA_CLIENT_ID')
                    , '&timestamp=',urlencode(l_timestamp)
                    , '&state=',p_state::VARCHAR
                    , '&response_type=code'
                    , '&client_secret=',urlencode(l_secret)
                    , '&access_type=online');

    RETURN l_ret;
EXCEPTION
.....
END;
$;
```


Практика. ЕСИА. get_Token

```
create function f_get_token(p_state uuid, p_code character varying) returns http_response
BEGIN
    l_timestamp := TO_CHAR(CURRENT_TIMESTAMP, 'YYYY.MM.DD HH24:MI:SS TZHTZM');
    l_secret := esia.f_gost_sign( p_data (CONCAT(pref.f$c1('ESIA_SCOPE'), l_timestamp, pref.f$c1('ESIA_CLIENT_ID'), p_state::VARCHAR))::BYTEA);

    l_body := CONCAT('client_id=', pref.f$c1('ESIA_CLIENT_ID')
        , '&code=' , p_code
        , '&grant_type=authorization_code'
        , '&client_secret=', urlencode(l_secret)
        , '&state=' , p_state::VARCHAR
        , '&redirect_uri=', urlencode(pref.f$c1('ESIA_RETURN_URL'))
        , '&scope=', urlencode(pref.f$c1('ESIA_SCOPE'))
        , '&timestamp=', urlencode(l_timestamp)
        , '&token_type=Bearer'
        , '&access_type=online');

    PERFORM http_set_curlopt('CURLOPT_TIMEOUT', '300');
    PERFORM http_set_curlopt('CURLOPT_CONNECTTIMEOUT', '5');
    l_req.method := 'POST';
    l_req.uri := pref.f$c1('ESIA_TOKEN');

    l_req.content_type := 'application/x-www-form-urlencoded';
    l_req.content := l_body;
    l_resp := http(l_req);
    RETURN l_resp;
```

Практика. ЕСИА. get_Personal_Info

```
1 create function f_get_personal_info(p_token character varying, p_client_id character varying) returns http_response_
2 DECLARE
3     l_req      http_request;
4     l_resp     http_response;
5 BEGIN
6     PERFORM http_set_curlpt('CURLOPT_TIMEOUT','300');
7     PERFORM http_set_curlpt('CURLOPT_CONNECTTIMEOUT','5');
8
9     l_req.method := 'GET';
10    l_req.uri     := pref.f$c1('ESIA_PRNS')||'/'||p_client_id;
11
12    l_req.headers := array_append(l_req.headers
13                                ,http_header('authorization','Bearer '||p_token));
14    l_resp := http(l_req);
15    RETURN l_resp;
```



PG BootCamp Russia 2025 Ekaterinburg

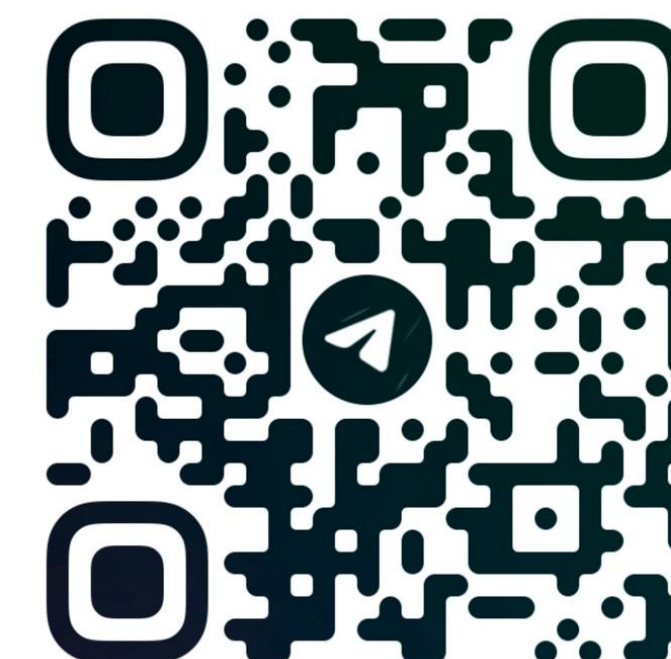
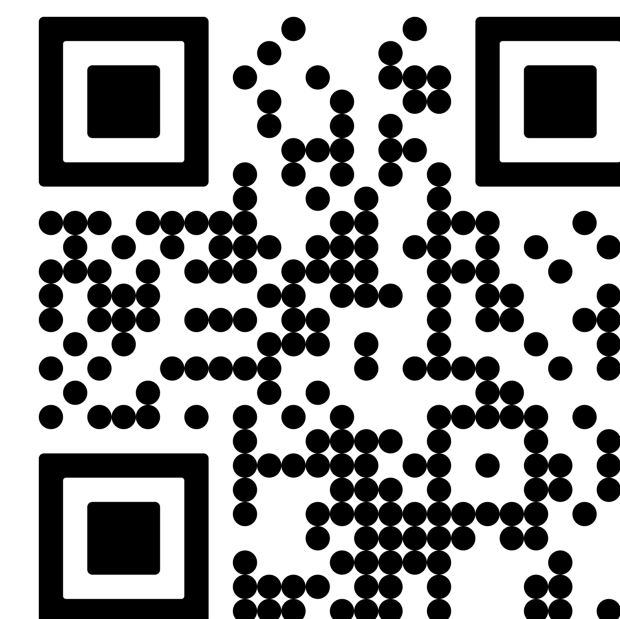
PGBootCamp.ru

Спасибо за внимание!

Константин Ващенко

kv@xsquare.ru

xsquare.ru



@XSQUARE365