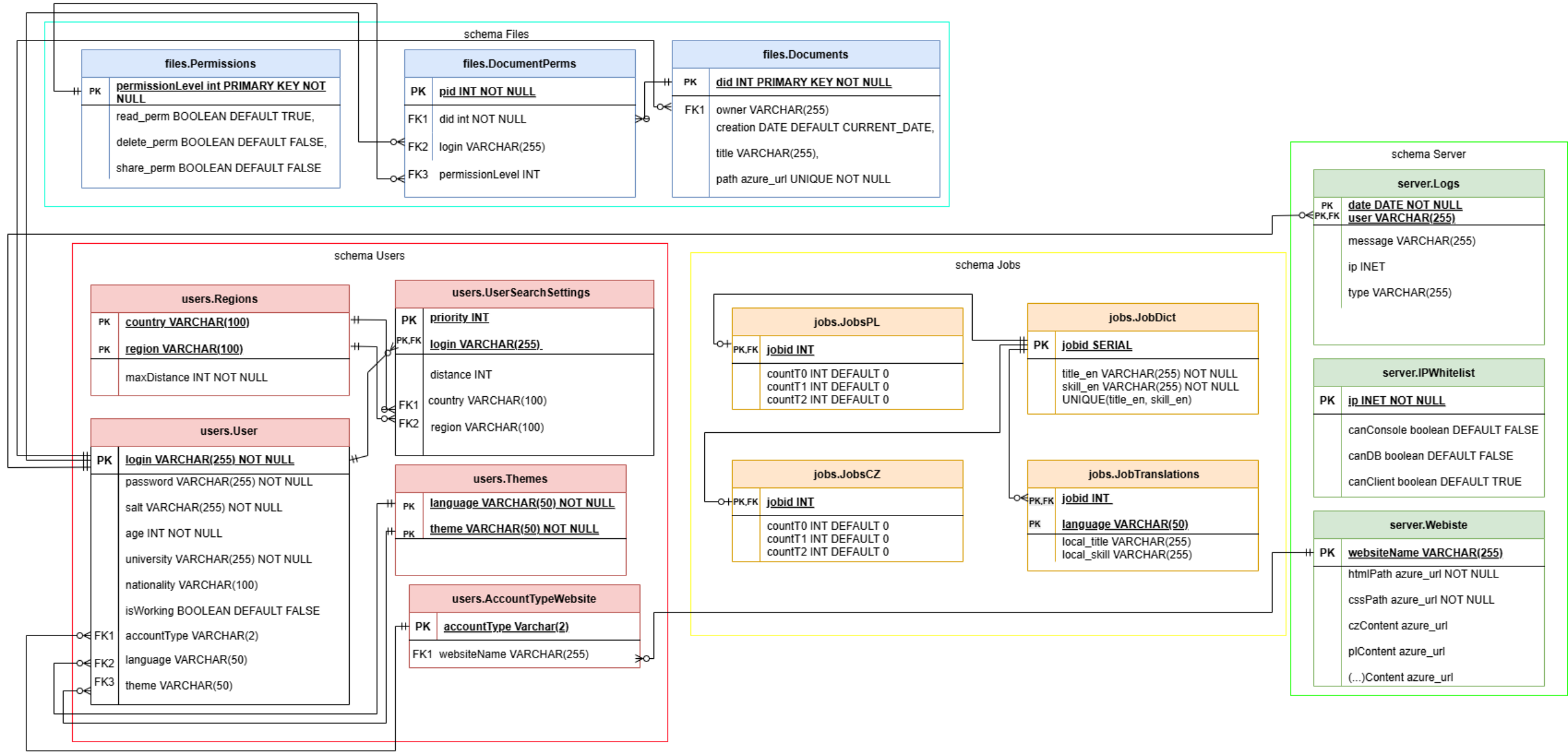
Obsługa aplikacji do tworzenia CV

Mateusz Maślanka, Kacper Rosner,
Filip Trzciński

Schemat bazy ERD / relacje



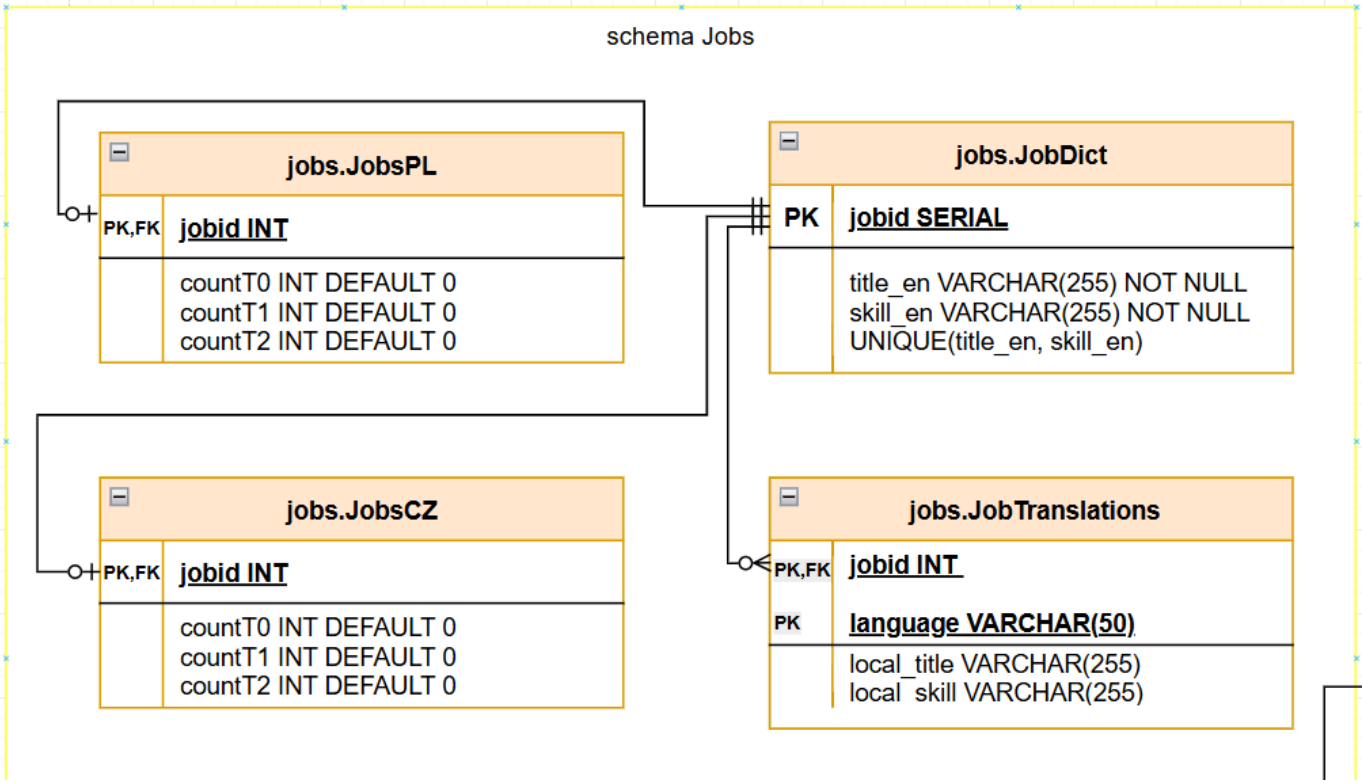
Schema Jobs

Schemat Jobs służy do magazynacji statystyk dotyczących wystąpienia danego stanowiska I umiejętności w serwisie np. Pracuj.pl.

Dana oferta pracy jest analizowana pod kątem lokalizacji, umiejętności I stanowiska, jakie jest wymienione. W momencie gdy analiza pojedynczego stanowiska zostaje zakończona, updateowane zostają odpowiednie rekordy w jobs.JobsPL.

Crawler:

1. Znalezienie id stanowiska w słowniku
2. Zwiększenie countera w odpowiedniej bazie (JobsPL / JobsCZ)



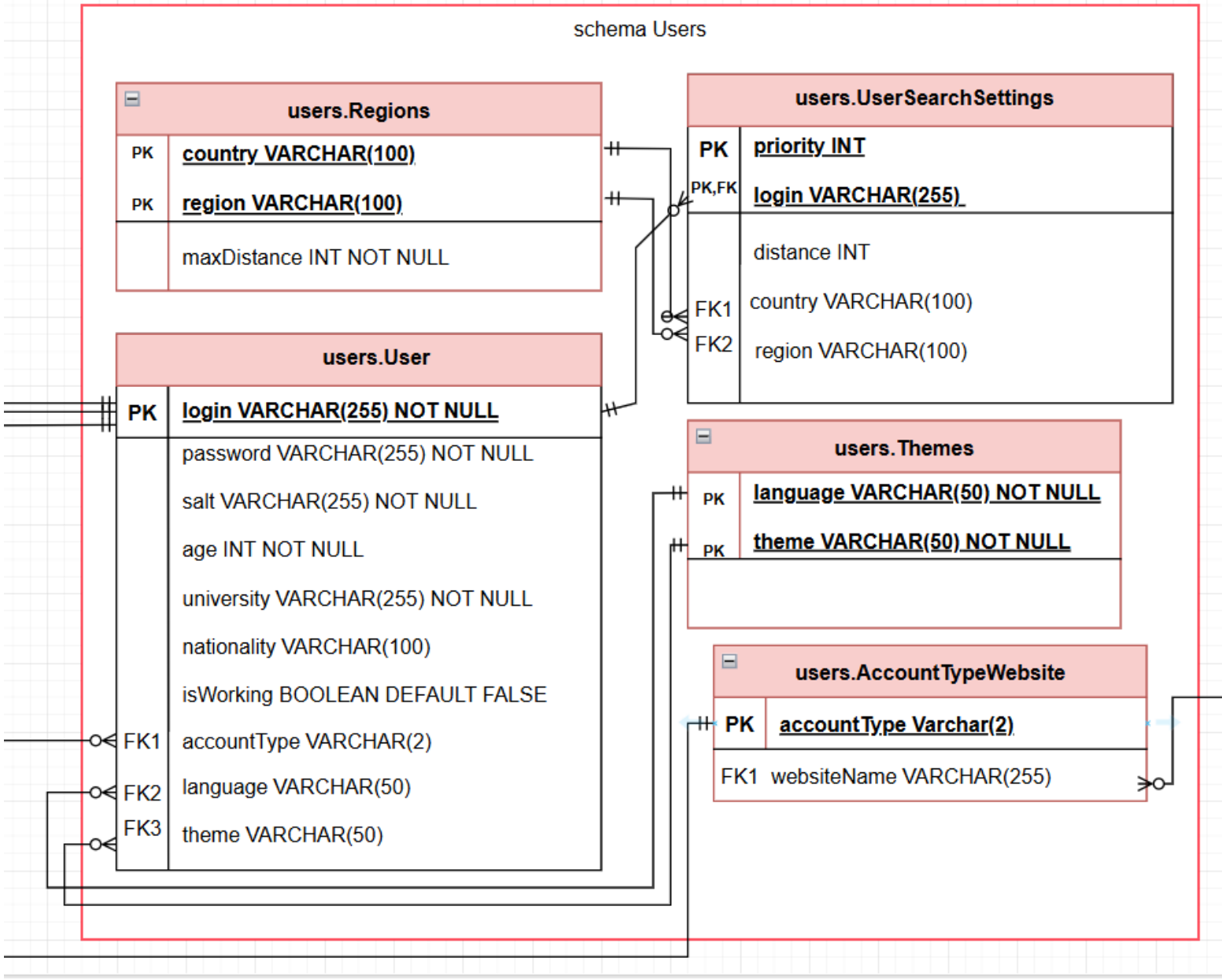
Wyszukiwanie stanowisk:

1. Znalezienie id stanowiska w słowniku
2. Sprawdzenie czy istnieje tłumaczenie dla danej pozycjiw konkretnym języku - jeśli tak – zwrot danych w lokalnym jęzuku - jeśli nie – zwrot w języku angielskim (title_en, skill_en)

Wyszukiwanie najbardziej opłacalnych stanowisk dla umiejętności I na odwrót

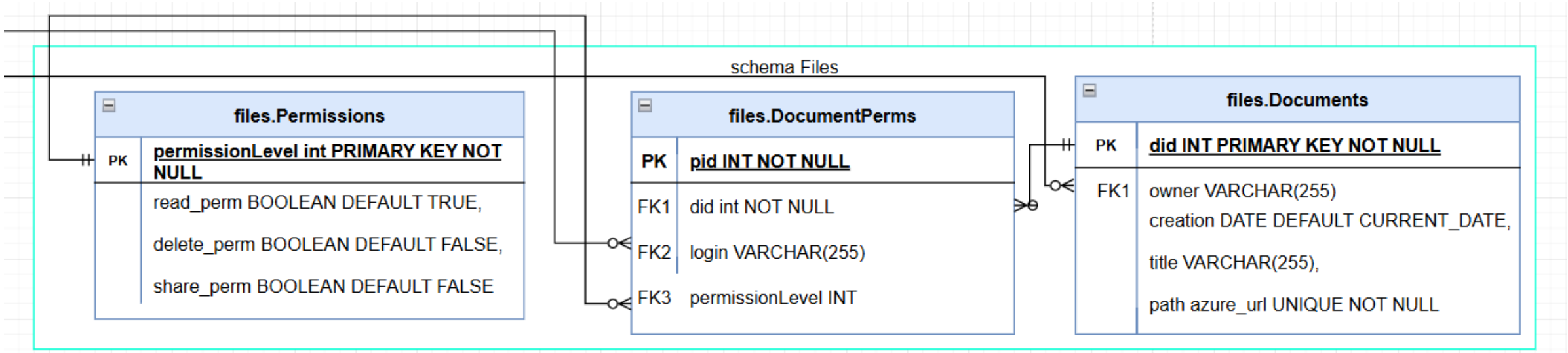
Schema Users

- Możliwość rejestracji, logowania, automatycznego sprawdzenia hasła
- Typ konta – sprawdzenie dostępności konkretnych stron dla danego typu konta
- Wybranie danego motywu - możliwość tworzenia motywów unikalnych dla danego języka - np. Na okazję specyficzną w danym państwie
- Unikalne rozpoznanie użytkownika pojedynczym primary key
- Możliwość dopisania do swojego konta regionu, który posłużyć może do dopasowaniakonta Klient do konta Rekruter.
- Możliwość wybrania (policzenia) użytkowników, którzy szukają pracy w danym regionie - funkcjonalność dla rekruterów
- Możliwość szukania pracy w różnych regionach z różnym priorytetem
- Szukanie pracy w odległości zgodnej z maksymalną odległością danego rejonu - Kraków - max 30km, Małoplska - max 0km



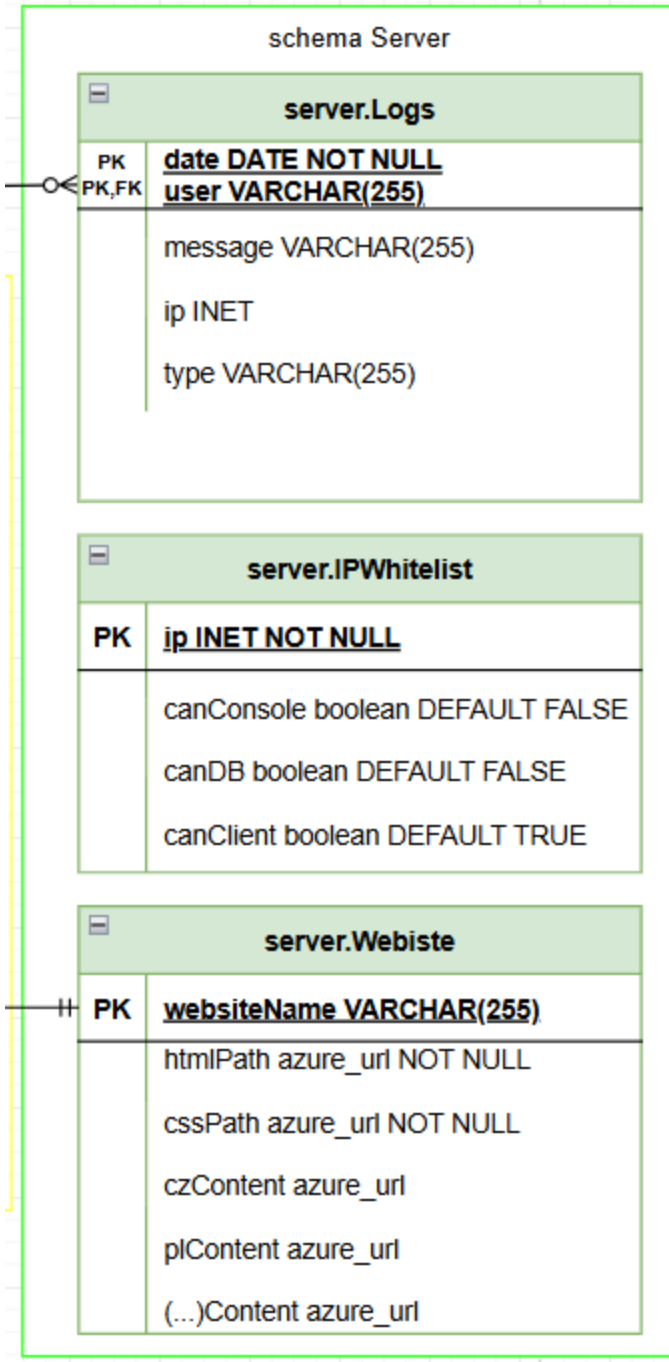
Schema Files

- Możliwość tworzenia CV przystosowanych pod różne stanowiska I zapisanie ich w bazie danych
- Możliwość udostępnienia innym (wielu) użytkownikom swojego CV na platformie
- Ze względu na typ pliku – pdf – brak problemu z jednoczesnym edytowaniem
- Predefiniowane permisje
- Przy wielokrotnym udostępnianiu, ślad po właścicieli pliku pozostaje



Schema Server

- (Poza bazą) 3 typy entity podłączalnych do serwera – konsola, kontroler bazy danych, klient
- Możliwość banowania użytkowników
- Logowanie zmian dokonywanych na bazie danych, prób włamania I komunikatów od konkretnych użytkowników w formie logów - błędów, warningów
- Możliwość tworzenia dynamicznych stron internetowych z fallbackiem w razie braku posiadania kontentu w danym języku



Baza danych obsługuje backend platformy rekrutacyjnej, która łączy **kandydatów**, **rekruterów/klientów** oraz **crawler** zbierający dane o rynku pracy.

System przechowuje dane użytkowników, ich preferencje wyszukiwania, dokumenty (np. CV) i statystyki popularności ofert pracy.

- **Rejestracja i logowanie** użytkowników (bezpieczne hashowanie haseł + salt).
- **Zarządzanie dokumentami (CV/PDF)**: dodawanie dokumentów, nadawanie i cofanie uprawnień (read/share/delete).
- **Wyszukiwanie / filtrowanie użytkowników** pod kątem regionu i ustawień (dla rekruterów).
- **Analiza rynku pracy (Jobs)**: słownik stanowisk i umiejętności + zliczanie trendów (T0/T1/T2) osobno dla krajów (PL/CZ).
- **Wielojęzyczność**: tłumaczenia stanowisk z mechanizmem fallback do EN, gdy brak lokalizacji.
- **Audyt i bezpieczeństwo**: automatyczne logowanie zmian w kluczowych tabelach (kto/co/kiedy/z jakiego IP).
- **User/Kandydat** – zakłada konto, dodaje CV, udostępnia rekruterom.
- **Client/Rekruter** – ma dostęp do udostępnionych dokumentów, filtruje kandydatów po regionie.
- **Crawler** – aktualizuje statystyki występowania ofert/umiejętności w kraju.
- **Co gwarantuje baza:**
- Brak przechowywania haseł plaintext (hash+salt).
- Uprawnienia do plików są kontrolowane poziomami i akcjami (read/share/delete).
- Każda zmiana danych jest audytowana w server.logs.

Test całego flow dodawania:

Dodanie tłumaczenia

Znalezienie po tłumaczeniu

Znalezienie najnowszego okresu

Powiększenie count

Sprawdzenie rekordu

```
postgres=> SELECT jobs.upsert_jobdict(
postgres(>     'Billing Supervisor',
postgres(>     'Data Entry',
postgres(>     'Kierownik ds. rozliczen',
postgres(>     'Wprowadzanie danych',
postgres(>     'PL'
postgres(> );
 upsert_jobdict
----------------
          28006
(1 row)


postgres=>
postgres=> SELECT * FROM jobs.get_en_by_local('Kierownik ds. rozliczen', 'Wprowadzanie danych');
      title_en      |  skill_en
--------------------+-------------
 Billing Supervisor |  Data Entry
(1 row)


postgres=>
postgres=>
postgres=> SELECT jobs.get_latest_period_code('pl') AS obecny_okres;
 obecny_okres
--------------
 2
(1 row)


postgres=>
postgres=> SELECT jobs.bump_counter_auto('pl', 'Billing Supervisor', 'Data Entry', 'T2', 50);
 bump_counter_auto
-------------------

(1 row)


postgres=>
postgres=> SELECT * FROM jobs.get_stats_by_text('Billing Supervisor', 'Data Entry', 'pl');
 job_id |      title_en      |  skill_en   | t0_count | t1_count | t2_count
--------+--------------------+-------------+----------+----------+----------
  28006 | Billing Supervisor |  Data Entry |      450 |        0 |       50
(1 row)
```

```
postgres=> \sf jobs.upsert_jobdict
CREATE OR REPLACE FUNCTION jobs.upsert_jobdict(p_title_en text, p_skill_en text, p_local_title text, p_local_skill text, p_language character varying)
 RETURNS integer
 LANGUAGE plpgsql
AS $function$
DECLARE
    v_jobid int;
BEGIN
    SELECT jobid INTO v_jobid
    FROM jobs.jobdict
    WHERE title_en ~* ('^\s*' || p_title_en || '\s*$')
      AND skill_en ~* ('^\s*' || p_skill_en || '\s*$')
    LIMIT 1;

    IF v_jobid IS NULL THEN
        INSERT INTO jobs.jobdict (title_en, skill_en)
        VALUES (btrim(p_title_en), btrim(p_skill_en))
        RETURNING jobid INTO v_jobid;
    END IF;

    INSERT INTO jobs.jobtranslations (jobid, language, local_title, local_skill)
    VALUES (v_jobid, p_language, btrim(p_local_title), btrim(p_local_skill))
    ON CONFLICT (jobid, language) DO UPDATE
    SET local_title = EXCLUDED.local_title,
        local_skill = EXCLUDED.local_skill;

    RETURN v_jobid;
END;
$function$
```

```
postgres=> \sf jobs.get_en_by_local
CREATE OR REPLACE FUNCTION jobs.get_en_by_local(p_title text, p_skill text)
 RETURNS TABLE(title_en character varying, skill_en character varying)
 LANGUAGE plpgsql
AS $function$
BEGIN
  RETURN QUERY
  SELECT jd.title_en, jd.skill_en
  FROM jobs.jobdict jd
  JOIN jobs.jobtranslations jt USING (jobid)
  WHERE jt.local_title ~* ('^\s*' || p_title || '\s*$')
    AND jt.local_skill ~* ('^\s*' || p_skill || '\s*$');

  IF NOT FOUND THEN
    RAISE EXCEPTION 'Nie znaleziono tlumaczenia dla: %, %', p_title, p_skill;
  END IF;
END;
$function$
```

```
postgres=> \sf jobs.bump_counter_auto
CREATE OR REPLACE FUNCTION jobs.bump_counter_auto(p_country text, p_title_en text, p_skill_en text, p_bucket text, p_amount integer DEFAULT 1)
 RETURNS void
 LANGUAGE plpgsql
AS $function$
DECLARE
  v_bucket text := upper(p_bucket);
  v_country text := lower(p_country);
  v_jobid int;
BEGIN

  SELECT jobid INTO v_jobid
  FROM jobs.jobdict
  WHERE title_en ~* ('^\s*' || p_title_en || '\s*$')
    AND skill_en ~* ('^\s*' || p_skill_en || '\s*$');

  IF v_jobid IS NULL THEN
    RAISE EXCEPTION 'Job with title % and skill % not found', p_title_en, p_skill_en;
  END IF;

  IF v_bucket NOT IN ('T0','T1','T2') THEN
    RAISE EXCEPTION 'bucket must be T0, T1 or T2';
  END IF;

  IF v_country IN ('pl','poland') THEN
    IF v_bucket = 'T0' THEN
      UPDATE jobs.jobspl SET countt0 = countt0 + p_amount WHERE jobid = v_jobid;
    ELSIF v_bucket = 'T1' THEN
      UPDATE jobs.jobspl SET countt1 = countt1 + p_amount WHERE jobid = v_jobid;
    ELSE
      UPDATE jobs.jobspl SET countt2 = countt2 + p_amount WHERE jobid = v_jobid;
    END IF;

  ELSIF v_country IN ('cz','czech','czechia') THEN
    IF v_bucket = 'T0' THEN
      UPDATE jobs.jobscz SET countt0 = countt0 + p_amount WHERE jobid = v_jobid;
    ELSIF v_bucket = 'T1' THEN
      UPDATE jobs.jobscz SET countt1 = countt1 + p_amount WHERE jobid = v_jobid;
    ELSE
      UPDATE jobs.jobscz SET countt2 = countt2 + p_amount WHERE jobid = v_jobid;
    END IF;

  ELSE
    RAISE EXCEPTION 'Unknown country: % (use pl/cz)', p_country;
  END IF;

  IF NOT FOUND THEN
    RAISE EXCEPTION 'Job counters row not found for country % and ID %', p_country, v_jobid;
  END IF;
END;
$function$
```

```
postgres=> \sf jobs.get_latest_period_code
CREATE OR REPLACE FUNCTION jobs.get_latest_period_code(p_country text)
 RETURNS text
 LANGUAGE plpgsql
AS $function$
DECLARE
  v_table text;
  v_latest_col text;
BEGIN
  v_table := CASE WHEN lower(p_country) IN ('pl', 'poland') THEN 'jobspl' ELSE 'jobscz' END;

  SELECT column_name INTO v_latest_col
  FROM information_schema.columns
  WHERE table_schema = 'jobs'
    AND table_name = v_table
    AND column_name LIKE 'countt%'
  ORDER BY substring(column_name FROM '[0-9]+')::int DESC
  LIMIT 1;

  RETURN upper(substring(v_latest_col FROM 'countt(.*)'));
END;
$function$
```

```
postgres=> \sf jobs.get_stats_by_text
CREATE OR REPLACE FUNCTION jobs.get_stats_by_text(p_title_en text, p_skill_en text, p_country text)
 RETURNS TABLE(job_id integer, title_en character varying, skill_en character varying, t0_count integer, t1_count integer, t2_
count integer)
 LANGUAGE plpgsql
AS $function$
DECLARE
  v_c text := lower(p_country);
BEGIN
  RETURN QUERY
  SELECT
    jd.jobid,
    jd.title_en,
    jd.skill_en,

    CASE WHEN v_c IN ('pl', 'poland') THEN jp.countt0 ELSE jc.countt0 END,
    CASE WHEN v_c IN ('pl', 'poland') THEN jp.countt1 ELSE jc.countt1 END,
    CASE WHEN v_c IN ('pl', 'poland') THEN jp.countt2 ELSE jc.countt2 END
  FROM jobs.jobdict jd

  LEFT JOIN jobs.jobspl jp ON jd.jobid = jp.jobid
  LEFT JOIN jobs.jobscz jc ON jd.jobid = jc.jobid

  WHERE jd.title_en ~* ('^\s*' || p_title_en || '\s*$')
    AND jd.skill_en ~* ('^\s*' || p_skill_en || '\s*$')

    AND (
      (v_c IN ('pl', 'poland') AND jp.jobid IS NOT NULL) OR
      (v_c IN ('cz', 'czech', 'czechia') AND jc.jobid IS NOT NULL)
    );
END;
$function$
```

Widok v_user_documents_permissions

Widok zwraca listę dokumentów użytkownika wraz z aktualnymi uprawnieniami prawnymi: read / share / delete oraz flagą is_owner.

Tu pokazujemy wynik dla użytkownika alice. Widzimy dokument "CV Alice" oraz flagi t, czyli ma pełne prawa (czytanie, udostępnianie i usuwanie) oraz is_owner = t, bo jest właścicielem. Ten widok działa jak "API" dla systemu: jednym SELECT-em dostajemy pełny stan uprawnień

```
postgres=> SELECT *
postgres-> FROM files.v_user_documents_permissions
postgres-> WHERE login = 'alice'
postgres-> ORDER BY creation_date DESC, did;
 login | did | owner | creation_date |  title   |                path                | permissionlevel | can_read | can_share | can_delete | is_owner
-------+-----+-------+---------------+----------+------------------------------------+-----------------+----------+-----------+------------+----------
 alice |   1 | alice | 2026-01-24    | CV Alice | azure.database.com/files/cv_alice.pdf |               3 | t        | t         | t          | t
(1 row)

postgres=>
```

```
postgres=> SELECT set_config('app.user','admin', true);
 set_config
------------
 admin
(1 row)
```

Ustawienie kontekstu app.user, z którego korzystają triggery audytu i walidacji.

app.user to wartość ustawiana na czas sesji która mówi bazie jaki użytkownik wykonuje operacje. Triggery (np. audyt) odczytują ten kontekst i zapisują go w logach aby wiedzieć kto zrobił zmianę.Dzięki temu logi i walidacje nie są anonimowe i da się prześledzić historię zmian.

Używamy set_config('app.user','admin', true), żeby zasymulować użytkownika aplikacji. Wynik admin potwierdza, że kontekst został ustawiony i kolejne operacje DML będą podpisywane tym użytkownikiem

Czy wszystkie wymagane triggery są aktywne na tabelach w schematach users/files/jobs/server.

Po tym teście oczekiwanym wynikiem ma być Lista ok. 11 triggerów:

- trg_users_hash_password, trg_users_block_login_change
- trg_files_auto_owner_perms, trg_files_validate_documentperms
- trg_jobs_create_counters, trg_jobs_translation_fallback
- trg_audit_* dla users/files/jobs.

```
postgres=> SELECT
postgres->   n.nspname AS schema,
postgres->   c.relname AS table_name,
postgres->   t.tgname  AS trigger_name
postgres-> FROM pg_trigger t
postgres-> JOIN pg_class c ON c.oid = t.tgrelid
postgres-> JOIN pg_namespace n ON n.oid = c.relnamespace
postgres-> WHERE NOT t.tgisinternal
postgres->   AND n.nspname IN ('users','files','jobs','server')
postgres-> ORDER BY 1,2,3;
 schema |   table_name    |           trigger_name
--------+-----------------+-----------------------------------
 files  | documentperms   | trg_audit_files_documentperms
 files  | documentperms   | trg_files_validate_documentperms
 files  | documents       | trg_audit_files_documents
 files  | documents       | trg_files_auto_owner_perms
 jobs   | jobdict         | trg_audit_jobs_jobdict
 jobs   | jobdict         | trg_jobs_create_counters
 jobs   | jobtranslations | trg_audit_jobs_jobtranslations
 jobs   | jobtranslations | trg_jobs_translation_fallback
 users  | user            | trg_audit_users_user
 users  | user            | trg_users_block_login_change
 users  | user            | trg_users_hash_password
(11 rows)

postgres=>
```

Czy users.usersearchsettings.distance<users.region.maxDistance ?

- Trigger waliduje poprawność ustawień wyszukiwania użytkownika. Przy INSERT lub UPDATE w tabeli users.usersearchsettings sprawdza, czy podany dystans jest mniejszy niż maksymalny limit zdefiniowany dla danego kraju i regionu w tabeli users.regions.
Jeśli region nie istnieje lub dystans przekracza limit, zapis jest blokowany, a baza zwraca błąd.
Dzięki temu dane są spójne i użytkownicy nie mogą ustawić nierealnego zasięgu wyszukiwania.

```
CREATE OR REPLACE FUNCTION users.fn_validate_search_distance()
RETURNS trigger AS $$
DECLARE
    v_max_dist int;
BEGIN
    SELECT r.maxdistance INTO v_max_dist
    FROM users.regions r
    WHERE r.country = NEW.country
      AND r.region = NEW.region;

    IF v_max_dist IS NULL THEN
        RAISE EXCEPTION 'Region % w kraju % nie istnieje', NEW.region, NEW.country;
    END IF;

    IF NOT (NEW.distance < v_max_dist) THEN
        RAISE EXCEPTION 'Dystans (%) musi byc mniejszy niz limit regionu (%)',
            NEW.distance, v_max_dist;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CTION

DROP TRIGGER IF EXISTS trg_users_validate_search_distance ON users.usersearchsettings;
ER
CREATE TRIGGER trg_users_validate_search_distance
BEFORE INSERT OR UPDATE OF distance, country, region ON users.usersearchsettings
FOR EACH ROW
EXECUTE FUNCTION users.fn_validate_search_distance();
```

Test - userSearchSettings

```
INSERT INTO users."user" (login, password, age, university, nationality)
VALUES ('test_user', 'placeholder_pass', 25, 'AGH', 'Polish')
ON CONFLICT (login) DO NOTHING;


INSERT INTO users.regions (country, region, maxdistance)
VALUES ('Poland', 'Krakow', 50)
ON CONFLICT (country, region) DO NOTHING;
```

Trigger sprawdza, czy ustawiona przez użytkownika odległość wyszukiwania nie przekracza limitu zdefiniowanego dla regionu.

```
INSERT INTO users.usersearchsettings (login, priority, country, region, distance)
VALUES ('test_user', 1, 'Poland', 'Krakow', 30);
```

```
postgres=> select * from users.usersearchsettings;
   login   | priority | country | region | distance
-----------+----------+---------+--------+----------
 test_user |        1 | Poland  | Krakow |       30
(1 row)


postgres=> UPDATE users.usersearchsettings SET distance = 60 WHERE login = 'test_user';
ERROR:  Dystans (60) musi byc mniejszy niz limit regionu (50)
CONTEXT:  PL/pgSQL function users.fn_validate_search_distance() line 15 at RAISE
postgres=> select * from users.usersearchsettings;
   login   | priority | country | region | distance
-----------+----------+---------+--------+----------
 test_user |        1 | Poland  | Krakow |       30
(1 row)
```

Funkcjonalność updatowania jobs.jobsXY

```
CREATE OR REPLACE FUNCTION jobs.get_en_by_local(p_title text, p_skill text)
 RETURNS TABLE(title_en character varying, skill_en character varying)
 LANGUAGE plpgsql
AS $function$
BEGIN
  RETURN QUERY
  SELECT jd.title_en, jd.skill_en
  FROM jobs.jobdict jd
  JOIN jobs.jobtranslations jt USING (jobid)
  WHERE jt.local_title ~* ('^\s*' || p_title || '\s*$')
    AND jt.local_skill ~* ('^\s*' || p_skill || '\s*$');

  IF NOT FOUND THEN
    RAISE EXCEPTION 'Nie znaleziono tlumaczenia dla: %, %', p_title, p_skill;
  END IF;
END;
```

- Funkcja mapuje lokalne nazwy stanowisk i umiejętności na wersję angielską używaną wewnętrznie przez system.

```
postgres=> SELECT login, password, salt
postgres-> FROM users."user"
postgres-> WHERE login='doc_user_1';
   login    |             password             |               salt
------------+----------------------------------+----------------------------------
 doc_user_1 | 48484830807207b1c5e3595dcc922953 | 3d9fbb93aaec07896ab30f7f0425b8e9
(1 row)
```

```
postgres=> SELECT users.register_user(
postgres(>   'doc_user_1', 'Pass123!', 22, 'AGH', 'Polish', false, false,
postgres(>   'STUDENT', 'EN', 'DARK'
postgres(> );
 register_user
---------------

(1 row)
```

Funkcja users.register_user(...)

Rejestruje użytkownika, a hasło nie jest zapisywane plaintext tylko jest haszowane + salt.

Testujemy czy hasło nie jest przechowywane jako plaintext i czy salt jest ustawiony oraz, wyniki pokazują że funkcja działa poprawnie

```
postgres=> UPDATE users."user"
postgres-> SET password='NewPass!234'
postgres-> WHERE login='doc_user_1';
UPDATE 1
postgres=>
postgres=> SELECT users.check_login('doc_user_1','NewPass!234') AS should_be_true_after_update;
 should_be_true_after_update
-----------------------------
 t
(1 row)
```

Funkcja
Trigger trg_users_hash_password
(hash + salt)

Automatycznie haszuje hasło przy INSERT i przy UPDATE hasła.

Testujemy czy trigger hashujący działa też przy zmianie Hasła.

Poprawność działania tej funkcji widzimy również na poprzednim slajdzie

Funkcja users.check_login(login, password)

Sprawdza logowanie użytkownika na podstawie hasha + salt, zwraca true/false.

```
postgres=> SELECT users.check_login('doc_user_1','wrong') AS should_be_false;
 should_be_false
-----------------
 f
(1 row)
```

```
postgres=> SELECT users.check_login('doc_user_1','Pass123!') AS should_be_true;
 should_be_true
----------------
 t
(1 row)
```

```
postgres=> SELECT * FROM files.permissions ORDER BY permissionlevel;
 permissionlevel | read_perm | delete_perm | share_perm
-----------------+-----------+-------------+------------
               1 | t         | f           | f
               2 | t         | t           | f
               3 | t         | t           | t
(3 rows)
```

Tabela files.permissions (poziomy 1/2/3)

Słownik poziomów uprawnień i mapowanie na akcje (read/delete/share).

Testujemy czy tabela files.permissions jest wypełniona i jakie akcje odpowiadają poziomom.

Trigger trg_files_auto_owner_perms

```
postgres=> INSERT INTO files.documents(did, owner, title, path)
postgres-> VALUES (101, 'alice', 'CV Doc Test', 'azure.database.com/files/cv_doc_test.pdf');
INSERT 0 1
```

```
postgres=> SELECT * FROM files.documentperms
postgres-> WHERE did = 101
postgres-> ORDER BY pid;
 pid | did | login | permissionlevel
-----+-----+-------+-----------------
   3 | 101 | alice |               3
(1 row)
```

```
postgres=> SELECT
postgres->   files.has_permission('alice', 101, 'read')   AS owner_read,
postgres->   files.has_permission('alice', 101, 'share')  AS owner_share,
postgres->   files.has_permission('alice', 101, 'delete') AS owner_delete;
 owner_read | owner_share | owner_delete
------------+-------------+--------------
 t          | t           | t
(1 row)
```

Po dodaniu dokumentu automatycznie dopisuje ownera do files.documentperms z maksymalnym poziomem (zwykle 3).

Tutaj testujemy czy działa dodawnie INSERT a następnie czy dokument dostał automatycznie ownera

Owner powinien dostać poziom 3 czyli prawo do read share oraz delate no i to też przetestowalismy

Funkcja files.grant_permission (granto.did, target, level)

Nadaje uprawnienie do dokumentu innemu użytkownikowi i dopisuje rekord do files.documentperms.

W tych testach sprawdzamy czy pojawi się wiersz dla usera z permissionlevel=1, czy permisja dziala i pokaze się blad oraz to czy user z share może użyć grant_permission

Zasada bezpieczeństwa:

Uprawnienia może nadawać tylko użytkownik, który ma do danego dokumentu prawo SHARE (np. owner / osoba z share).

```
postgres=> DO $$
postgres$> BEGIN
postgres$>   PERFORM files.grant_permission('bob_doc_1', 101, 'bob_doc_1', 1);
postgres$>   RAISE EXCEPTION 'FAIL: should require share permission';
postgres$> EXCEPTION WHEN others THEN
postgres$>   RAISE NOTICE 'OK expected: %', SQLERRM;
postgres$> END $$;
NOTICE:  OK expected: User bob_doc_1 has no share permission for document 101
DO
```

```
postgres=> SELECT files.grant_permission('alice', 101, 'bob_doc_1', 3);
 grant_permission
------------------

(1 row)

postgres=>
postgres=> -- bob teraz daje read ownerowi (albo komuś innemu istniejącemu)
postgres=> SELECT files.grant_permission('bob_doc_1', 101, 'alice', 1);
 grant_permission
------------------

(1 row)

postgres=>
postgres=> SELECT * FROM files.documentperms
postgres-> WHERE did = 101
postgres-> ORDER BY pid;
 pid | did |   login   | permissionlevel
-----+-----+-----------+-----------------
   3 | 101 | alice     |               1
   4 | 101 | bob_doc_1 |               3
(2 rows)
```

```
postgres=> SELECT files.grant_permission('alice', 101, 'bob_doc_1', 1);
 grant_permission
------------------

(1 row)

postgres=>
postgres=> SELECT * FROM files.documentperms
postgres-> WHERE did = 101
postgres-> ORDER BY pid;
 pid | did |   login   | permissionlevel
-----+-----+-----------+-----------------
   3 | 101 | alice     |               3
   4 | 101 | bob_doc_1 |               1
(2 rows)
```

Funkcja files.has_permission(login, did, action)

Sprawdza czy dany użytkownik ma prawo do danej akcji (read/delete/share) na dokumencie.

Tutaj użytkownik miał level=1 czyli tylko read i to się sprawdza bo user nie może uzywać share

```
postgres=> SELECT
postgres->   files.has_permission('bob_doc_1', 101, 'read')  AS bob_read_after,
postgres->   files.has_permission('bob_doc_1', 101, 'share') AS bob_share_after;
 bob_read_after | bob_share_after
----------------+-----------------
 t              | f
(1 row)
```

```
postgres=> SELECT users.register_user(
postgres(>   'charlie_doc_3', 'Char#123', 25, NULL, 'Polish', false, false,
postgres(>   'STUDENT', 'EN', 'DARK'
postgres(> );
 register_user
---------------

(1 row)

postgres=>
postgres=> -- aktor = bob (ma share=3)
postgres=> SELECT set_config('app.user','bob_doc_1', true);
 set_config
------------
 bob_doc_1
(1 row)

postgres=>
postgres=> INSERT INTO files.documentperms(pid, did, login, permissionlevel)
postgres-> VALUES (12001, 101, 'charlie_doc_1', 3);
ERROR:  duplicate key value violates unique constraint "documentperms_pkey"
DETAIL:  Key (pid)=(12001) already exists.
postgres=>
postgres=> SELECT * FROM files.documentperms
postgres-> WHERE did = 101
postgres-> ORDER BY pid;
  pid  | did |     login     | permissionlevel
-------+-----+---------------+-----------------
     3 | 101 | alice         |               1
     4 | 101 | bob_doc_1     |               3
 12001 | 101 | charlie_doc_1 |               3
(3 rows)
```

Trigger trg_files_validate_documentperms

Waliduje bezpośrednie modyfikacje files.documentperms. Zabezpiecza przed "ręcznym" dopisywaniem uprawnień niezgodnie z zasadami.

W naszej aktualnej imprelentacji INSERT przejdzie bo nasz user ma level=3, user z share może bezpośrednio modyfikować documentperms do swojego poziomu

Trigger trg_jobs_create_counters
+
funkcja fn_create_job_counters

Po dodaniu nowej pozycji do jobs.jobdictautomatycznie tworzy rekordy liczników w jobs.jobspl i jobs.jobscz (start od 0).

Testujemy INSERT do jobs.jobdict + uruchomienie triggera trg_jobs_create_counters.

Testujemy czy Trigger fn_create_job_counters utworzył wiersze w obu tabelach liczników.

```
postgres=> INSERT INTO jobs.jobdict(title_en, skill_en)
postgres-> VALUES ('Data Engineer', 'SQL')
postgres-> RETURNING jobid;
 jobid
--------
 344249
(1 row)

INSERT 0 1
```

```
postgres=> SELECT * FROM jobs.jobspl WHERE jobid = 344249;
 jobid  | countt0 | countt1 | countt2
--------+---------+---------+---------
 344249 |       0 |       0 |       0
(1 row)

postgres=> SELECT * FROM jobs.jobscz WHERE jobid = 344249;
 jobid  | countt0 | countt1 | countt2
--------+---------+---------+---------
 344249 |       0 |       0 |       0
(1 row)
```

Funkcja jobs.bump_counter(country, jobid, bucket)

```
postgres=> SELECT jobs.bump_counter('pl', 344249, 'T2');
 bump_counter
--------------

(1 row)

postgres=> SELECT jobs.bump_counter('cz', 344249, 'T1');
 bump_counter
--------------

(1 row)

postgres=>
postgres=> SELECT * FROM jobs.jobspl WHERE jobid = 344249;
 jobid  | countt0 | countt1 | countt2
--------+---------+---------+---------
 344249 |       0 |       0 |       1
(1 row)

postgres=> SELECT * FROM jobs.jobscz WHERE jobid = 344249;
 jobid  | countt0 | countt1 | countt2
--------+---------+---------+---------
 344249 |       0 |       1 |       0
(1 row)
```

Funkcja inkrementuje odpowiedni licznik (PL/CZ) i odpowiedni koszyk T0/T1/T2.

Testujemy Funkcję jobs.bump_counter oraz poprawne mapowanie bucketów T0/T1/T2.

Testujemy również Obsługę błędu "unknown country" oraz obsługę błędu “bucket must be T0, T1 or T2”.

```
postgres=> DO $$
postgres$> BEGIN
postgres$>   PERFORM jobs.bump_counter('xx', 344249, 'T1');
postgres$>   RAISE EXCEPTION 'FAIL: unknown country should fail';
postgres$> EXCEPTION WHEN others THEN
postgres$>   RAISE NOTICE 'OK expected: %', SQLERRM;
postgres$> END $$;
NOTICE:  OK expected: Unknown country: xx (use pl/cz)
DO
```

```
postgres=> DO $$
postgres$> BEGIN
postgres$>   PERFORM jobs.bump_counter('pl', 344249, 'BAD');
postgres$>   RAISE EXCEPTION 'FAIL: BAD bucket should fail';
postgres$> EXCEPTION WHEN others THEN
postgres$>   RAISE NOTICE 'OK expected: %', SQLERRM;
postgres$> END $$;
NOTICE:  OK expected: bucket must be T0, T1 or T2
DO
```

Trigger trg_jobs_translation_fallback

Jeśli brak tłumaczenia (local_title/local_skill puste), to ustawia fallback z title_en/skill_en.

Po testach tej funkcji możemy zazważyć że wynik to local_title = 'Data Engineer' oraz local_skill = 'SQL'.

Czyli trigger działa poprawnie

Późniejszy test pokazuje nam że widzimy dokładnie swoje wartości bez powrotu do angielskich

```
postgres=> INSERT INTO jobs.jobtranslations(jobid, language, local_title, local_skill)
postgres-> VALUES (344249, 'EN', NULL, NULL)
postgres-> ON CONFLICT (jobid, language) DO UPDATE
postgres-> SET local_title = EXCLUDED.local_title,
postgres->     local_skill = EXCLUDED.local_skill;
INSERT 0 1
postgres=>
postgres=> SELECT * FROM jobs.jobtranslations
postgres-> WHERE jobid = 344249 AND language = 'EN';
 jobid  | language |  local_title  | local_skill
--------+----------+---------------+-------------
 344249 | EN       | Data Engineer | SQL
(1 row)
```

```
postgres=> UPDATE jobs.jobtranslations
postgres-> SET local_title = 'Inżynier Danych', local_skill = 'SQL / ETL'
postgres-> WHERE jobid = 344249 AND language = 'EN';
UPDATE 1
postgres=>
postgres=> SELECT * FROM jobs.jobtranslations
postgres-> WHERE jobid = 344249 AND language = 'EN';
 jobid  | language |   local_title    | local_skill
--------+----------+------------------+-------------
 344249 | EN       | Inżynier Danych | SQL / ETL
(1 row)
```

Funkcja server.is_ip_allowed(ip, scope)

Sprawdza, czy IP ma dostęp do danego "scope" (console/db/client) na podstawie server.ipwhitelist.

Po tym teście możemy zauważyć że widać rekordy w whitelist + funkcja zwraca t dla scope ustawionych na true

```
postgres=> SELECT * FROM server.ipwhitelist ORDER BY ip;
    ip     | canconsole | candb | canclient
-----------+------------+-------+-----------
 127.0.0.1 | t          | t     | t
(1 row)

postgres=>
postgres=> SELECT server.is_ip_allowed('127.0.0.1','console') AS console_ok;
 console_ok
------------
 t
(1 row)

postgres=> SELECT server.is_ip_allowed('127.0.0.1','db')      AS db_ok;
 db_ok
-------
 t
(1 row)

postgres=> SELECT server.is_ip_allowed('127.0.0.1','client')  AS client_ok;
 client_ok
-----------
 t
(1 row)
```

Funkcja server.fn_audit_log+ triggery trg_audit_

Audyt zapisuje każdą operację INSERT/UPDATE/DELETE do server.logs wraz z app.user i IP. W testach wykonujemy zmiany na tabelach i widzimy, że w logach pojawiają się nowe wpisy z poprawnym typem operacji, użytkownikiem i adresem IP potwierdzenie widzimy na testach obok

```
postgres=> SELECT
postgres->   n.nspname AS schema,
postgres->   c.relname AS table_name,
postgres->   t.tgname  AS trigger_name
postgres-> FROM pg_trigger t
postgres-> JOIN pg_class c ON c.oid = t.tgrelid
postgres-> JOIN pg_namespace n ON n.oid = c.relnamespace
postgres-> WHERE NOT t.tgisinternal
postgres->   AND t.tgname LIKE 'trg_audit_%'
postgres-> ORDER BY 1,2,3;
 schema |   table_name    |          trigger_name
--------+-----------------+--------------------------------
 files  | documentperms   | trg_audit_files_documentperms
 files  | documents       | trg_audit_files_documents
 jobs   | jobdict         | trg_audit_jobs_jobdict
 jobs   | jobtranslations | trg_audit_jobs_jobtranslations
 users  | user            | trg_audit_users_user
(5 rows)
```

```
postgres=> SELECT date, "user", ip, message
postgres-> FROM server.logs
postgres-> ORDER BY date DESC, "user";
    date    |   user    |       ip       |
                        message
------------+-----------+----------------+-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-----------------------
 2026-01-24 | azureuser | 91.150.222.250 | INSERT on users.user; INSERT on users.user; INSERT on files.documents; INSERT on files.document
perms; INSERT on jobs.jobdict; INSERT on users.user; UPDATE on users.user; INSERT on jobs.jobtranslations; INSERT on users.user; INSERT on
 files.documentperms;
(1 row)
```

```
postgres=> SELECT *
postgres-> FROM server.logs
postgres-> ORDER BY date DESC, "user";
    date    |   user    |
                        message
                                |       ip       |  type
------------+-----------+-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-----------------------------------+----------------+--------
 2026-01-24 | azureuser | INSERT on users.user; INSERT on users.user; INSERT on files.documents; INSERT on files.documentper
ms; INSERT on jobs.jobdict; INSERT on users.user; UPDATE on users.user; INSERT on jobs.jobtranslations; INSERT on users.user
; INSERT on files.documentperms;  | 91.150.222.250 | INSERT
(1 row)
```

```
postgres=> -- 1) UPDATE usera (powinien dać audit UPDATE on users.user)
postgres=> UPDATE users."user"
postgres-> SET age = age
postgres-> WHERE login = 'doc_user_1';
UPDATE 1
postgres=>
postgres=> -- 2) INSERT/DELETE na jobtranslations (audit na jobs.jobtranslations)
postgres=>
postgres=> INSERT INTO jobs.jobtranslations(jobid, language, local_title, local_skill)
postgres-> VALUES (344249, 'PL', 'Tester', 'SQL')
postgres-> ON CONFLICT (jobid, language) DO UPDATE
postgres-> SET local_title = EXCLUDED.local_title,
postgres->     local_skill = EXCLUDED.local_skill;
INSERT 0 1
```

Publiczne Repozytorium Github

https://github.com/kacper-rosner/cvDB