Politechnika Rzeszowska im. Ignacego Łukasiewicza
Wydział Matematyki i Fizyki Stosowanej

Aplikacje Bazodanowe – Projekt

Mini USOS

Nad projektem pracowali:
Norbert Cyran (gr. nr 1, nr albumu: 166307)
Vitalii Morskyi (gr. nr 4, nr albumu: 166731)

Rzeszów
21 stycznia 2023

1 Definicja zadania projektowego

W tym rozdziale zdefiniowano zadanie projektowe, wybrany temat projektu oraz plan działania przyszłej aplikacji.

1.1 Wymagania projektowe

Większość wymagań ostatecznej aplikacji dotyczą języka PL/pgSQL. Rozwiązanie powinno zawierać co najmniej:

- 8 procedur w proceduralnym języku programowania PL/pgSQL;
- 8 funkcji w proceduralnym języku programowania PL/pgSQL.

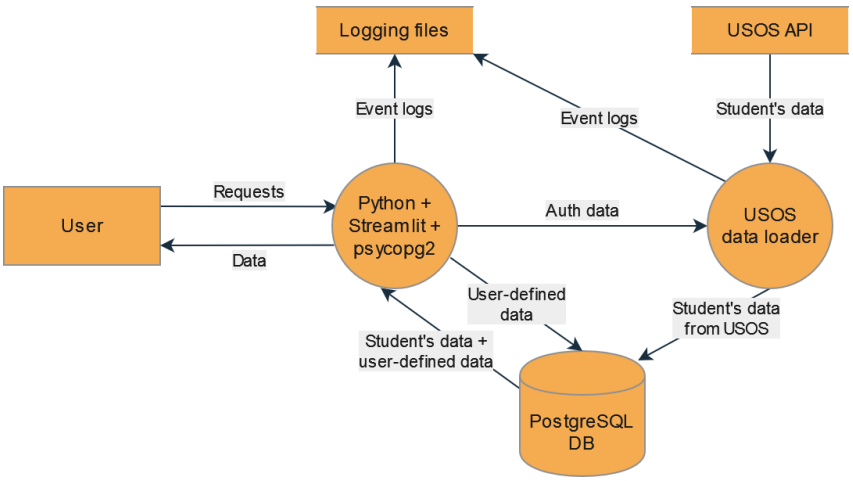
Obowiązkowym wymaganiem jest również przygotowanie przyjaznego interfejsu aplikacji, na którym będą zamieszczone różnego rodzaju raporty, formularzy oraz deski rozdzielcze.

1.2 Opis wybranego tematu projektu

Postanowiono przygotować aplikację, której działalność nieco przypomina system USOS. Pozwala ona na zarządzanie uczelnianym planem zajęć, studentami i nawet grupami studenckimi. Aplikacja również zawiera panel z wybranymi statystykami.

Czynności, które można dokonać w aplikacji:

- Zalogować się za pomocą konta uczelnianego, tym samym automatycznie dodając swoje zajęcia do bazy.
- Przenieść wybranego studenta do innej grupy dla poszczególnych zajęć.
- Przenieść wybranego studenta do innej grupy dla wszystkich zajęć tego typu.
- Automatycznie stworzyć i dodać nowego studenta do grup o najmniejszej ilości studentów.
- Zmienić prowadzącego wybranej grupy.
- Dodać nowe zajęcia dla wybranej grupy.
- Zmienić czas wybranych zajęć.
- Usunąć wybrane zajęcia.
- Dodać nowego prowadzącego.
- Dodać nowy budynek uczelni.
- Zobaczyć ilość godzin pracy z IiAD każdego prowadzącego.
- Zobaczyć odległości pomiędzy poszczególnymi budynkami PRz



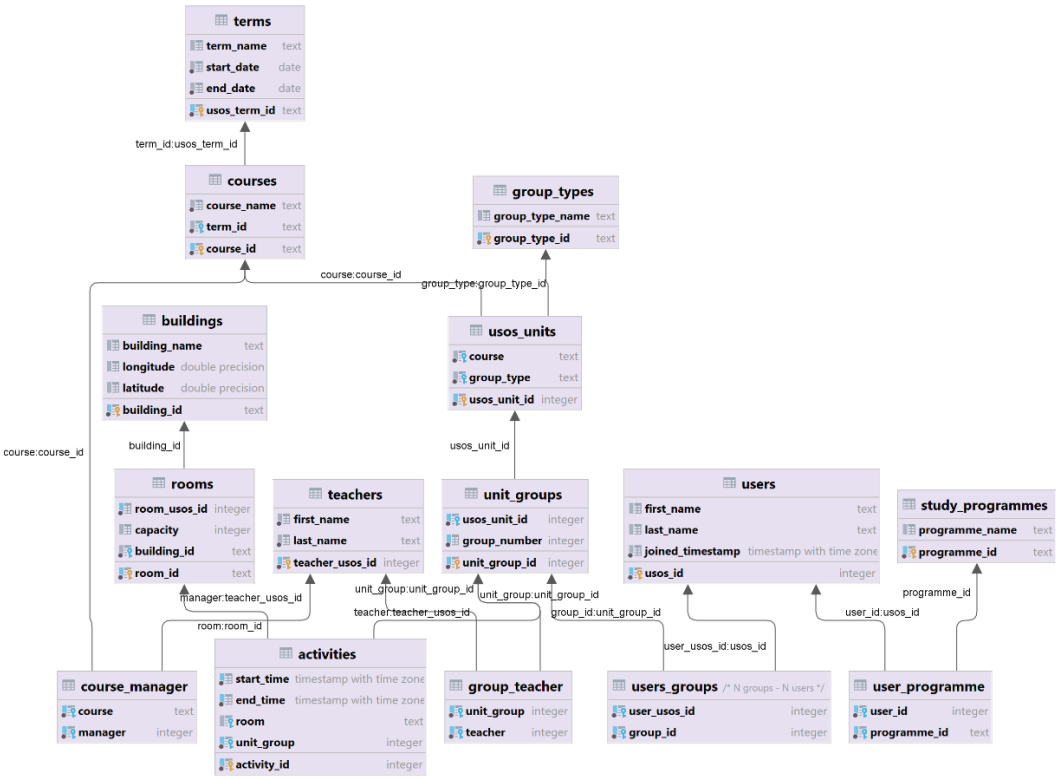
Rysunek 1.1 Diagram przepływu danych w aplikacji

1.3 Plan aplikacji

Podstawę aplikacji pełni system bazodanowy PostgreSQL. Została w nim utworzona baza danych oparta o system USOS. Dane do systemu bazodanowego wgrywane są w sposób automatyczny z użyciem USOS API. Sama aplikacja napisana jest w języku Python. Interfejs wizualny użytkownika pełni strona internetowa napisana w bibliotece Pythona o nazwie Streamlit. Na Rysunek 1.1 pokazano diagram przepływu danych w naszej aplikacji.

W systemie bazodanowym PostgreSQL zostaną przygotowane procedury i funkcje, które będą wywoływane z poziomu aplikacji. Więcej informacji na ten temat w następnym rozdziale.

2 Funkcji i procedury w PostgreSQL



Rysunek 2.1 Diagram relacyjnej bazy danych

W naszej aplikacji funkcje przeważnie pełnią bardziej pomocniczą rolę i są wykorzystywane w ramach innych procesów. Natomiast procedury zwykle stanowią już całkowity proces, który jest uruchamiany przez użytkownika. Są jednak wyjątki z tej reguły. Do przykładu funkcja „Back to the future” jest często używana wewnątrz innych procedur w celu sprawdzenia podanych przez użytkownika parametrów. Na Rysunek 2.2 podano diagram, który pokazuje w jaki sposób funkcje i procedury komunikują między sobą w aplikacji.

W następnych podrozdziałach dokładnie opisano działalność każdej funkcji i procedury. Szczególną uwagę zwróciliśmy na opis użytych bardziej skomplikowanych typów danych i struktur języka PL/pgSQL. W celach ułatwienia zrozumienia poniższych funkcji i procedur na Rysunek 2.1 podano diagram relacyjnej bazy danych w PostgreSQL.

2.1 Przygotowane funkcje

- **„Grupa ma wolne miejsca”**
  Funkcja sprawdza czy w podanej grupie zajęciowej znajduje się wolne miejsce. Wykonywane jest to poprzez porównanie najmniejszej ilości miejsc w jednej sali, a ilości studentów w grupie. Jeżeli ilość osób w grupie jest większa, niż ilość miejsc w sali to funkcje zwraca błąd z komunikatem „Grupa nie mieści się już w niektórych salach!”. W przypadku gdy ilość miejsc jest taka sama jak ilość studentów funkcja zwróci *„False”,* w przypadku istnienia wolnych miejsc uzyskamy *„True”*.

```
CREATE OR REPLACE FUNCTION grupa_ma_wolne_miejsca(id_grupy unit_groups.unit_group_id%TYPE)
    RETURNS BOOLEAN
    LANGUAGE plpgsql
AS
$$
DECLARE
    liczebnosc_grupy INTEGER;
    min_rozmiar_sali public.rooms.capacity%TYPE;
BEGIN
    liczebnosc_grupy = ilosc_studentow_w_grupie(id_grupy);
    SELECT MIN(r.capacity)
    INTO min_rozmiar_sali
    FROM unit_groups ung
             INNER JOIN activities a ON ung.unit_group_id = a.unit_group
             INNER JOIN rooms r ON r.room_id = a.room
    WHERE ung.unit_group_id = id_grupy;

    CASE
        WHEN min_rozmiar_sali < liczebnosc_grupy
            THEN RAISE DATA_EXCEPTION USING MESSAGE = 'Grupa już nie mieści się w niektórych
salach!';
        WHEN min_rozmiar_sali = liczebnosc_grupy THEN RETURN FALSE;
        ELSE RETURN TRUE;
        END CASE;
END
$$;
```

- **„Zajęcia studenta w danym czasie”**
  Funkcja posiada 3 wersje zależnie od podanych danych. Pierwsza wersja pobiera od użytkownika id studenta, godzinę rozpoczęcia i zakończenia zajęć. Pozostałe dwie wersje pobierają dodatkowe dane w postaci ID starych zajęć. Do zmiennej *„liczba_kolidujacych_zajec”* zapisuje ilość zajęć, które student ma w podanym wcześniej przedziale czasowym. Jeżeli wartość zmiennej jest większa niż zero, funkcja zwraca „False”, w przeciwnym wypadku uzyskamy „True”.

```
-- Wersja pierwsza
CREATE OR REPLACE FUNCTION zajecia_studenta_w_danym_czasie(
    student users.usos_id%TYPE,
    czas_poczatkowy activities.start_time%TYPE,
    czas_koncowy activities.end_time%TYPE
)
    RETURNS bool
    LANGUAGE plpgsql
AS
$$
DECLARE
    liczba_kolidujacych_zajec INT;
BEGIN
    SELECT COUNT(activity_id)
    INTO liczba_kolidujacych_zajec
    FROM activities act
             INNER JOIN unit_groups ung ON act.unit_group = ung.unit_group_id
             INNER JOIN users_groups usg ON ung.unit_group_id = usg.group_id
             INNER JOIN users u ON u.usos_id = usg.user_usos_id
    WHERE (start_time >= czas_poczatkowy AND start_time < czas_koncowy
        OR end_time > czas_poczatkowy AND end_time <= czas_koncowy)
      AND u.usos_id = student;

    IF liczba_kolidujacych_zajec > 0 THEN
        RETURN FALSE;
    ELSIF liczba_kolidujacych_zajec = 0 THEN
        RETURN TRUE;
    END IF;
END;
$$;
```

Wersja druga i trzecia posiadają możliwość „zignorowania” starych zajęć studenta. Są one prawi identyczne, różnią się tylko formatem zmiennej *„ignoruj_usos_unit_id”*. W jednej to *„integer”,* a w drugiej *„array”*.

```
-- Wersja druga
CREATE OR REPLACE FUNCTION zajecia_studenta_w_danym_czasie(
    student users.usos_id%TYPE,
    czas_poczatkowy activities.start_time%TYPE,
    czas_koncowy activities.end_time%TYPE,
    ignoruj_usos_unit_id usos_units.usos_unit_id%TYPE
)
    RETURNS bool
    LANGUAGE plpgsql
AS
$$
DECLARE
    liczba_kolidujacych_zajec INT;
BEGIN
    SELECT COUNT(activity_id)
    INTO liczba_kolidujacych_zajec
    FROM activities act
             INNER JOIN unit_groups ung ON act.unit_group = ung.unit_group_id
             INNER JOIN users_groups usg ON ung.unit_group_id = usg.group_id
             INNER JOIN users u ON u.usos_id = usg.user_usos_id
    WHERE (start_time >= czas_poczatkowy AND start_time < czas_koncowy
        OR end_time > czas_poczatkowy AND end_time <= czas_koncowy)
      AND u.usos_id = student
      AND ung.usos_unit_id != ignoruj_usos_unit_id;

    IF liczba_kolidujacych_zajec > 0 THEN
        RETURN FALSE;
    ELSIF liczba_kolidujacych_zajec = 0 THEN
        RETURN TRUE;
    END IF;
END;
$$;

-- Wersja trzecia
CREATE OR REPLACE FUNCTION zajecia_studenta_w_danym_czasie(
    student users.usos_id%TYPE,
    czas_poczatkowy activities.start_time%TYPE,
    czas_koncowy activities.end_time%TYPE,
    ignoruj_liste_usos_unit_ids INTEGER ARRAY
)
    RETURNS bool
    LANGUAGE plpgsql
AS
$$
DECLARE
    liczba_kolidujacych_zajec INT;
BEGIN
    SELECT COUNT(activity_id)
    INTO liczba_kolidujacych_zajec
    FROM activities act
             INNER JOIN unit_groups ung ON act.unit_group = ung.unit_group_id
             INNER JOIN users_groups usg ON ung.unit_group_id = usg.group_id
             INNER JOIN users u ON u.usos_id = usg.user_usos_id
    WHERE (start_time >= czas_poczatkowy AND start_time < czas_koncowy
        OR end_time > czas_poczatkowy AND end_time <= czas_koncowy)
      AND u.usos_id = student
      AND ung.usos_unit_id != ALL (ignoruj_liste_usos_unit_ids);

    IF liczba_kolidujacych_zajec > 0 THEN
        RETURN FALSE;
    ELSIF liczba_kolidujacych_zajec = 0 THEN
        RETURN TRUE;
    END IF;
END;
$$;
```

- **„Zajęcia wykładowcy w danym czasie”**
  Celem funkcji jest sprawdzenie czy podany wykładowca nie ma zajęć w podanym przez użytkownika przedziale czasowym. Tak jak w poprzedniej funkcji zliczane są zajęcia odbywające są w danym okresie czasowym. Jeżeli ilość zajęć danego wykładowcy jest większa niż 0 to funkcja zwraca *„False”*, w przeciwnym wypadku dostaniemy *„True”*.

```
CREATE OR REPLACE FUNCTION zajecia_wykladowcy_w_danym_czasie(wykladowca
teachers.teacher_usos_id%TYPE,
                                                             czas_poczatkowy
activities.start_time%TYPE,
                                                             czas_koncowy
activities.end_time%TYPE)
    RETURNS bool
    LANGUAGE plpgsql
AS
$$
DECLARE
    wynik INT;
BEGIN
    SELECT COUNT(activity_id)
    INTO wynik
    FROM activities act
             INNER JOIN unit_groups ug ON ug.unit_group_id = act.unit_group
             INNER JOIN group_teacher gt ON ug.unit_group_id = gt.unit_group
             INNER JOIN teachers t ON t.teacher_usos_id = gt.teacher
    WHERE (start_time >= czas_poczatkowy AND start_time < czas_koncowy
        OR end_time > czas_poczatkowy AND end_time <= czas_koncowy)
      AND t.teacher_usos_id = wykladowca;
    IF wynik > 0 THEN
        RETURN FALSE;
    ELSIF wynik = 0 THEN
        RETURN TRUE;
    END IF;
END;
$$;
```

- **„Zajęcia grupy w danym czasie”**
  Działanie tej funkcji jest identyczne jak dwóch poprzednich funkcji w tym, że sprawdzamy ilość zajęć dla całej grupy, a nie samego studenta czy wykładowcy.

```
CREATE OR REPLACE FUNCTION zajecia_grupy_w_danym_czasie(grupa unit_groups.unit_group_id%TYPE,
                                                        czas_poczatkowy
activities.start_time%TYPE,
                                                        czas_koncowy activities.end_time%TYPE)
    RETURNS bool
    LANGUAGE plpgsql
AS
$$
DECLARE
    wynik INT;
BEGIN
    SELECT COUNT(activity_id)
    INTO wynik
    FROM activities act
             INNER JOIN unit_groups ug ON ug.unit_group_id = act.unit_group
    WHERE (start_time >= czas_poczatkowy AND start_time < czas_koncowy
        OR end_time > czas_poczatkowy AND end_time <= czas_koncowy)
      AND ug.unit_group_id = grupa;
    IF wynik > 0 THEN
        RETURN FALSE;
    ELSIF wynik = 0 THEN
        RETURN TRUE;
    END IF;
END;
$$;
```

- **„Wolna sala”**
  Zadaniem funkcji jest sprawdzenie z w podanym okresie czasowym sala jest wolna. Funkcja działa identycznie jak funkcja poprzednia.

```
CREATE OR REPLACE FUNCTION wolna_sala(sala rooms.room_id%TYPE,
                                      czas_poczatkowy activities.start_time%TYPE,
                                      czas_koncowy activities.end_time%TYPE)
    RETURNS bool
    LANGUAGE plpgsql
AS
$$
DECLARE
    wynik INT;
BEGIN
    SELECT COUNT(activity_id)
    INTO wynik
    FROM activities act
             INNER JOIN rooms r ON r.room_id = act.room
    WHERE (start_time >= czas_poczatkowy AND start_time < czas_koncowy
        OR end_time > czas_poczatkowy AND end_time <= czas_koncowy)
      AND r.room_id = sala;
    IF wynik > 0 THEN
        RETURN FALSE;
    ELSIF wynik = 0 THEN
        RETURN TRUE;
    END IF;
END;
$$;
```

- **„Ilość godzin wykładowcy”**
  Funkcja pobiera ID wykładowcy podane przez użytkownika i zwraca łączną ilość godzin pracy tego wykładowcy. Za pomocą podwójnej pętli *„For”* zlicza łączną ilość godzin pracy wykładowcy. Wynik jest zwracany w godzinach zegarowych (nie akademicznych)

```
CREATE OR REPLACE FUNCTION ilosc_godzin_wykladowcy(wykladowca teachers.teacher_usos_id%TYPE)
RETURNS DOUBLE PRECISION AS
$$
DECLARE
    zajecie     RECORD;
    grupa       INT;
    laczny_czas DOUBLE PRECISION := 0.0;
BEGIN
    FOR grupa IN SELECT unit_group FROM group_teacher WHERE teacher = wykladowca
        LOOP
            FOR zajecie IN SELECT start_time, end_time FROM activities WHERE unit_group =
grupa
                LOOP
                    laczny_czas := laczny_czas +
                                   EXTRACT(EPOCH FROM AGE(zajecie.end_time,
zajecie.start_time)) / 3600 AS difference;
                END LOOP;
        END LOOP;
    RETURN laczny_czas;
END;
$$ LANGUAGE plpgsql;
```

- **„Ilość godzin studenta”**
  Funkcja działa identycznie jak funkcja poprzednia. Różni się tylko faktem, że podawane jest ID studenta, którego chcemy policzyć ilość godzin studiów, a nie wykładowcy.

```
CREATE OR REPLACE FUNCTION ilosc_godzin_studenta(student users.usos_id%TYPE) RETURNS DOUBLE
PRECISION AS
$$
DECLARE
    zajecie     RECORD;
    grupa       INT;
    laczny_czas DOUBLE PRECISION := 0.0;
BEGIN
    FOR grupa IN SELECT group_id FROM users_groups WHERE user_usos_id = student
        LOOP
            FOR zajecie IN SELECT start_time, end_time FROM activities WHERE unit_group =
grupa
                LOOP
                    laczny_czas := laczny_czas +
                                   EXTRACT(EPOCH FROM AGE(zajecie.end_time,
zajecie.start_time)) / 3600 AS difference;
                END LOOP;
        END LOOP;
    RETURN laczny_czas;
END;
$$ LANGUAGE plpgsql;
```

- **„Odległość między budynkami”**
  Funkcja pobiera ID dwóch różnych budynków i liczy występującą miedzy nimi odległość w metrach. Do obliczenia odległości wykorzystywany jest wzór Haversin, który wygląda następująco:

$$2 \cdot r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Źródło: https://pl.frwiki.wiki/wiki/Formule_de_haversine

Gdzie:
r – promień Ziemi w metrach (6371000m)
φ_1, φ_2 – szerokości geograficzne punktu 1 i punktu 2 w radianach
λ_1, λ_2 – długości geograficzne punktu 1 i punktu 2 w radianach

- **„Ilość studentów w grupie”**
  Funkcja pobiera ID grupy jako zmienną i zwraca ilość studentów w grupie. Wykonywane jest to poprzez zliczanie unikalnych ID studentów w danej grupie zajęciowej.

```
CREATE OR REPLACE FUNCTION ilosc_studentow_w_grupie(IN id_grupy
public.unit_groups.unit_group_id%TYPE)
    RETURNS INTEGER
    LANGUAGE plpgsql
AS
$$
DECLARE
    liczebnosc_grupy INTEGER;
BEGIN
    SELECT COUNT(DISTINCT usg.user_usos_id)
    INTO liczebnosc_grupy
    FROM public.unit_groups ung
             INNER JOIN users_groups usg ON ung.unit_group_id = usg.group_id
    WHERE ung.unit_group_id = id_grupy;
    RETURN liczebnosc_grupy;
END;
$$;
```

- **„Grupa z minimalną ilością studentów”**
  Użytkownik podaje rodzaj grupy zajęciowej (lab, pro). Funkcja wyznacza średnią ilość studentów wśród grup tego typu na kierunku 3 roku IiAD. Zatem funkcja zwraca numer grupy z najmniejszą średnią ilością studentów (średnia jest potrzebna w przypadku, gdy na laboratorium jednego przedmiotu jest mniej studentów niż na laboratorium innego przedmiotu tej samej grupy laboratoryjnej).

```
CREATE OR REPLACE FUNCTION grupa_z_min_iloscia_studentow(IN typ_grupy
public.group_types.group_type_id%TYPE)
    RETURNS DOUBLE PRECISION
    LANGUAGE plpgsql
AS
$$
DECLARE
    najmniej_liczebna_grupa RECORD;
BEGIN
    SELECT ung.group_number numer_grupy, AVG(ilosc_studentow_w_grupie(ung.unit_group_id))
srednia_ilosc_studentow
    INTO najmniej_liczebna_grupa
    FROM unit_groups ung
             INNER JOIN (SELECT DISTINCT uu.usos_unit_id, uu.group_type
                         FROM usos_units uu
                                  INNER JOIN unit_groups ung2 ON uu.usos_unit_id =
ung2.usos_unit_id
                                  INNER JOIN users_groups usg ON ung2.unit_group_id =
usg.group_id
                         -- Nie jest to błąd, po prostu ograniczamy się do 3 roku IiAD,
                         -- podany user_usos_id to ID Vitalii'a Morskyiego
                         WHERE usg.user_usos_id = 234394) y3uu ON ung.usos_unit_id =
y3uu.usos_unit_id
    WHERE y3uu.group_type = typ_grupy
    GROUP BY ung.group_number
    ORDER BY srednia_ilosc_studentow, numer_grupy
    LIMIT 1;
    RETURN najmniej_liczebna_grupa.numer_grupy;
END;
$$;
```

2.2 Przygotowane procedury

- **„Back to the future”**
  Procedura ma za zadanie sprawdzić czy podany czas zakończenia zajęć jest mniejszy od czasu rozpoczęcia. Jeżeli czas początkowy jest większy niż czas końcowy to otrzymamy wiadomość: „Czas początkowy jest większy od czasu końcowego.”. Posiadamy 2 wersje procedury, które różnią się tylko typem danych wejściowych. W pierwszej wersji mamy typ danych „Timestamp without time zone”, a w drugiej „Timestamp with time zone”.

```
-- Wersja pierwsza
CREATE OR REPLACE PROCEDURE back_to_the_future(czas_a TIMESTAMP WITHOUT TIME ZONE, czas_b
TIMESTAMP WITHOUT TIME ZONE)
    LANGUAGE plpgsql AS
$$
BEGIN
    IF czas_a > czas_b THEN
        RAISE INVALID_PARAMETER_VALUE USING MESSAGE = 'Czas początkowy jest większy od czasu
końcowego.';
    END IF;
END;
$$;

-- Wersja druga
CREATE OR REPLACE PROCEDURE back_to_the_future(czas_a TIMESTAMP WITH TIME ZONE, czas_b
TIMESTAMP WITH TIME ZONE)
    LANGUAGE plpgsql AS
$$
BEGIN
    IF czas_a > czas_b THEN
        RAISE INVALID_PARAMETER_VALUE USING MESSAGE = 'Czas początkowy jest większy od czasu
końcowego.';
    END IF;
END;
$$;
```

- **„Zmiana prowadzącego”**
  Ta procedura odpowiada za zmianę prowadzącego dla danej grupy zajęciowej z danego przedmiotu. Procedura pobiera ID wykładowcy i ID grupy. Przed aktualizacją prowadzącego sprawdzane jest czy wykładowca w tym czasie nie prowadzi innych zajęć. Do sprawdzenia wykorzystywana jest funkcja „Zajęcia wykładowcy w danym czasie”. Jeżeli funkcja zwróci wartość *„True”* prowadzący zostanie zmieniony, a procedura wyświetli wiadomość „Prowadzący został zmieniony”. W przeciwnym wypadku otrzymamy informacje, że prowadzący nie został zmieniony, ponieważ prowadzi w tym czasie inne zajęcia.

```
CREATE OR REPLACE PROCEDURE zmien_prowadzacego(
    IN grupa_id unit_groups.unit_group_id%TYPE,
    IN wykladowca_id group_teacher.teacher%TYPE,
    OUT result TEXT)
    LANGUAGE plpgsql
AS
$$
DECLARE
    poczatek TIMESTAMP;
    koniec   TIMESTAMP;
BEGIN
    SELECT start_time, end_time
    INTO poczatek, koniec
    FROM activities
    WHERE unit_group = grupa_id;
    IF zajecia_wykladowcy_w_danym_czasie(wykladowca_id, poczatek, koniec) = TRUE
    THEN
        UPDATE group_teacher
        SET teacher = wykladowca_id
        WHERE unit_group = grupa_id;
        result = 'Wykładowca został zmieniony.';
    ELSE
        RAISE EXCEPTION 'Wykładowca nie został zmieniony, ponieważ ma wtedy zajęcia.';
    END IF;
END
$$;
```

- **„Dodanie nowych zajęć”**
  Ta procedura odpowiedzialna jest za dodawanie nowych zajęć. Przyjmuje ID grupy, dla której mają odbywać się nowe zajęcia, ID sali, czas rozpoczęcia i czas zakończenia zajęć. Z pomocą procedury „Back to the future” sprawdza czy podane godziny są prawidłowe. Za pomocą polecenia „SELECT” uzyskuje ID wykładowcy, który ma prowadzić zajęcia. Po spełnieniu warunków: uzyskanie wartości *„True”* z funkcji „Zajęcia grupy w danym czasie”, „Zajęcia wykładowcy w danym czasie” i „Wolna sala” zostają dodane nowe zajęcia. W przypadku niespełnienia któregoś z warunków wyrzucany jest błąd z odpowiednią wiadomością.

```
CREATE OR REPLACE PROCEDURE dodaj_nowe_zajecia(grupa_id unit_groups.unit_group_id%TYPE,
                                               sala rooms.room_id%TYPE,
                                               czas_rozpoczecia activities.start_time%TYPE,
                                               czas_zakonczenia activities.end_time%TYPE,
                                               OUT result TEXT)
    LANGUAGE plpgsql
AS
$$
DECLARE
    wykladowca_id group_teacher.teacher%TYPE;
    warunek1      BOOLEAN;
    warunek2      BOOLEAN;
    warunek3      BOOLEAN;
BEGIN
    CALL back_to_the_future(czas_rozpoczecia, czas_zakonczenia);
    SELECT t.teacher_usos_id
    INTO wykladowca_id
    FROM teachers t
             INNER JOIN group_teacher gt ON t.teacher_usos_id = gt.teacher
             INNER JOIN unit_groups ug ON ug.unit_group_id = gt.unit_group
    WHERE ug.unit_group_id = grupa_id;
    warunek1 = zajecia_grupy_w_danym_czasie(grupa_id, czas_rozpoczecia, czas_zakonczenia);
    warunek2 = zajecia_wykladowcy_w_danym_czasie(wykladowca_id, czas_rozpoczecia,
czas_zakonczenia);
    warunek3 = wolna_sala(sala, czas_rozpoczecia, czas_zakonczenia);
    IF warunek1 = TRUE AND warunek2 = TRUE AND warunek3 = TRUE
    THEN
        INSERT INTO activities (start_time, end_time, room, unit_group)
        VALUES (czas_rozpoczecia, czas_zakonczenia, sala, grupa_id);
        result = 'Dodano nowe zajęcia.';
    END IF;
    IF warunek1 = FALSE
    THEN
        RAISE EXCEPTION 'Nie można dodać nowych zajec, poniewaz w danym czasie grupa ma inne
zajecia.';
    END IF;
    IF warunek2 = FALSE
    THEN
        RAISE EXCEPTION 'Nie można dodać nowych zajec, poniewaz w danym czasie wykładowca ma
inne zajecia.';
    END IF;
    IF warunek3 = FALSE
    THEN
        RAISE EXCEPTION 'Nie można dodać nowych zajec, poniewaz w danym czasie sala jest
zajeta.';
    END IF;
END;
$$;
```

- **„Przenieś zajęcia w czasie”**
  Procedura otrzymuje ID zajęć do przeniesienia, a także nowe godziny rozpoczęcia i zakończenia. Przy użyciu procedury „Back to the future” sprawdza poprawność podanych godzin. Procedura wyszukuje wszystkie dane powiązane z podanymi zajęciami i za pomocą użytych w poprzedniej procedurze funkcji sprawdza możliwość przeniesienia zajęć. Jeżeli jest to możliwe, to czas rozpoczęcia i zakończenia podanych zajęć zostaje zmieniony na nowy. W przeciwnym wypadku wyrzucany jest błąd z odpowiednim komunikatem.

```
CREATE OR REPLACE PROCEDURE przenieś_zajecia_w_czasie(id_zajec activities.activity_id%TYPE,
                                                      nowy_czas_rozpoczecia
activities.start_time%TYPE,
                                                      nowy_czas_zakonczenia
activities.end_time%TYPE,
                                                      OUT result TEXT)
    LANGUAGE plpgsql
AS
$$
DECLARE
    grupa_id      INTEGER;
    wykladowca_id INTEGER;
    sala          TEXT;
    warunek1      BOOLEAN;
    warunek2      BOOLEAN;
    warunek3      BOOLEAN;
BEGIN
    CALL back_to_the_future(nowy_czas_rozpoczecia, nowy_czas_zakonczenia);
    SELECT unit_group_id, gt.teacher, r.room_id
    INTO grupa_id, wykladowca_id, sala
    FROM unit_groups ug
             INNER JOIN activities a ON ug.unit_group_id = a.unit_group
             INNER JOIN group_teacher gt ON ug.unit_group_id = gt.unit_group
             INNER JOIN rooms r ON r.room_id = a.room
    WHERE activity_id = id_zajec;

    warunek1 = zajecia_grupy_w_danym_czasie(grupa_id, nowy_czas_rozpoczecia,
nowy_czas_zakonczenia);
    warunek2 = zajecia_wykladowcy_w_danym_czasie(wykladowca_id, nowy_czas_rozpoczecia,
nowy_czas_zakonczenia);
    warunek3 = wolna_sala(sala, nowy_czas_rozpoczecia, nowy_czas_zakonczenia);
    IF warunek1 = TRUE AND warunek2 = TRUE AND warunek3 = TRUE
    THEN
        UPDATE activities
        SET start_time = nowy_czas_rozpoczecia,
            end_time   = nowy_czas_zakonczenia
        WHERE activity_id = id_zajec;
        result = 'Przeniesiono zajęcia na wskazany czas.';
    END IF;
    IF warunek1 = FALSE
    THEN
        RAISE EXCEPTION 'Nie mozna przeniesc zajec, poniewaz w danym czasie grupa ma inne
zajecia.';
    END IF;
    IF warunek2 = FALSE
    THEN
        RAISE EXCEPTION 'Nie mozna przeniesc zajec, poniewaz w danym czasie wykładowca ma inne
zajecia.';
    END IF;
    IF warunek3 = FALSE
    THEN
        RAISE EXCEPTION 'Nie mozna przeniesc zajec, poniewaz w danym czasie sala jest
zajeta.';
    END IF;
END;
$$;
```

- **„Przenieś studenta do innej grupy na jedne zajęcia”**
  Użytkownik podaje procedurze ID Studenta do przeniesienia, ID obecnej grupy zajęciowej i ID nowej grupy zajęciowej. Za pomocą funkcji „Zajęcia studenta w danym czasie” procedura sprawdza czy zajęcia nowej grupy nie będą kolidować z pozostałymi zajęciami studenta. Funkcja „Grupa ma wolne miejsca” sprawdza czy nowa grupa ma wolne miejsce dla studenta. Jeżeli obie funkcje zwrócą pozytywny wynik, to student zostanie przeniesiony między grupami. W przeciwnym wypadku wyrzucany jest błąd, zawierający informacje o czynnikach przeszkadzających przeniesieniu studenta do innej grupy.

```
CREATE OR REPLACE PROCEDURE przenies_studenta_do_innej_grupy_na_jedne_zajecia(student_id
users.usos_id%TYPE,
                                                                              obecna_grupa_id
unit_groups.unit_group_id%TYPE,
                                                                              nowa_grupa_id
unit_groups.unit_group_id%TYPE,
                                                                              OUT result TEXT)
    LANGUAGE plpgsql
AS
$$
DECLARE
    sala                TEXT;
    student_nie_ma_zajec bool = TRUE;
    x                   RECORD;
BEGIN
    IF grupa_ma_wolne_miejsca(nowa_grupa_id) THEN
        SELECT room_id
        INTO sala
        FROM rooms
                 INNER JOIN activities a ON rooms.room_id = a.room
                 INNER JOIN unit_groups ug ON ug.unit_group_id = a.unit_group
        WHERE unit_group_id = nowa_grupa_id;
        FOR x IN SELECT start_time, end_time
                 FROM activities a
                          INNER JOIN unit_groups u ON u.unit_group_id = a.unit_group
                 WHERE unit_group_id = nowa_grupa_id
            LOOP
                IF zajecia_studenta_w_danym_czasie(student_id, x.start_time, x.end_time,
obecna_grupa_id) = FALSE
                THEN
                    student_nie_ma_zajec = FALSE;
                END IF;
            END LOOP;
        IF student_nie_ma_zajec
        THEN
            UPDATE users_groups
            SET group_id = nowa_grupa_id
            WHERE user_usos_id = student_id
              AND group_id = obecna_grupa_id;
            result = 'Student został przepisany do innej grupy.';
        ELSE
            RAISE EXCEPTION 'Student nie może zmienić grupy, ponieważ w wybranym czasie on ma
inne zajęcia.';
        END IF;
    ELSE
        RAISE EXCEPTION 'Student nie może być przypisany do wkazanej grupy, ponieważ niema w
niej wolnych miejsc';
    END IF;
EXCEPTION
    WHEN DATA_EXCEPTION THEN
        BEGIN
            RAISE NOTICE 'Some assigned rooms to the group % are too small.', nowa_grupa_id;
            RAISE EXCEPTION 'Student nie może być przypisany do wkazanej grupy, ponieważ niema
w niej wolnych miejsc';
        END;
END;
$$;
```

- **„Przenieś studenta na wszystkie zajęcia”**
  Procedura przyjmuje ID studenta, ID rodzaju zajęć i ID nowej grupy. Do tymczasowej tabeli „Unit groups to move” pobierane są wszystkie dane na temat dotychczasowych grup danego studenta dla danego rodzaju zajęć. Dalej procedura działa podobnie do poprzedniej procedury. Różnica polega na tym, że ostatnia przenosiła studenta do innej grupy tylko dla jednego typu zajęć jednego przedmiotu, gdyż ta procedura stara się przenieść studenta do innej grupy dla wszystkich przedmiotów, w których jest dany typ grupy. Np. przenieść studenta do grupy laboratoryjnej nr 4 na wszystkich przedmiotach, na których są zajęcia laboratoryjne. Procedura ograniczona jest do grup zajęciowych 3 roku Inżynierii i Analizy Danych (czyli nie zadziała ona dla studentów 2 roku IiAD).

```
CREATE OR REPLACE PROCEDURE przenies_studenta_na_wszystkie_zajecia(id_studenta
users.usos_id%TYPE,
                                                                   rodzaj_zajec
group_types.group_type_id%TYPE,
                                                                   nr_nowej_grupy
unit_groups.group_number%TYPE,
                                                                   OUT result TEXT)
    LANGUAGE plpgsql
AS
$$
DECLARE
    single_group_to_move           RECORD;
    nie_ma_zajec                   BOOLEAN = TRUE;
    activities_of_the_moving_group RECORD;
    id_nowej_grupy                 unit_groups.unit_group_id%TYPE;
    lista_grup_studenta            INTEGER ARRAY;
BEGIN
    CREATE TEMP TABLE unit_groups_to_move AS
    SELECT unit_groups.usos_unit_id, unit_groups.group_number, unit_groups.unit_group_id
    FROM unit_groups
             INNER JOIN users_groups ug ON unit_groups.unit_group_id = ug.group_id
             INNER JOIN usos_units uu ON uu.usos_unit_id = unit_groups.usos_unit_id
             INNER JOIN users u ON u.usos_id = ug.user_usos_id
    WHERE id_studenta = u.usos_id
      AND uu.group_type = rodzaj_zajec;
    SELECT INTO lista_grup_studenta ARRAY(SELECT usos_unit_id FROM unit_groups_to_move);
    FOR single_group_to_move IN SELECT * FROM unit_groups_to_move
        LOOP
            FOR activities_of_the_moving_group IN
                SELECT a.start time,
                       a.end_time
                FROM activities a
                         INNER JOIN unit_groups u ON u.unit_group_id = a.unit_group
                WHERE u.usos_unit_id = single_group_to_move.usos_unit_id
                  AND u.group_number = nr_nowej_grupy
                LOOP
                    IF
                            zajecia_studenta_w_danym_czasie(
                                    id_studenta,
                                    activities_of_the_moving_group.start_time,
                                    activities_of_the_moving_group.end_time,
                                    lista_grup_studenta) = FALSE
                    THEN
                        nie_ma_zajec = FALSE;
                    END IF;
                END LOOP;
        END LOOP;
    IF nie_ma_zajec
    THEN
        FOR single_group_to_move IN SELECT * FROM unit_groups_to_move
            LOOP
                SELECT unit_group_id
                INTO id_nowej_grupy
                FROM unit_groups
                WHERE usos_unit_id = single_group_to_move.usos_unit_id
                  AND group_number = nr_nowej_grupy;
                UPDATE users_groups
                SET group_id = id_nowej_grupy
                WHERE user_usos_id = id_studenta
                  AND group_id = single_group_to_move.unit_group_id;
            END LOOP;
        result = 'Student został przeniesiony do nowej grupy.';
    ELSE
        RAISE EXCEPTION 'Student nie został przeniesiony do nowej grupy';
    END IF;
    DROP TABLE IF EXISTS unit_groups_to_move;
END;
$$;
```

- **„Dodaj nowego prowadzącego”**
  Procedura odpowiedzialna za dodanie nowego prowadzącego przyjmuje z klawiatury imię i nazwisko nowego wykładowcy. Sprawdza czy wprowadzone imię i nazwisko spełniają warunki takie jak: czy składają się tylko ze znaków ASCII i czy zaczynają się na wielką literę, a także czy nie ma ich już w bazie danych.

```
CREATE OR REPLACE PROCEDURE dodaj_nowego_prowadzacego(
    IN imie public.teachers.first_name%TYPE,
    IN nazwisko public.teachers.last_name%TYPE,
    OUT result TEXT
)
    LANGUAGE plpgsql
AS

$$
BEGIN
    CASE
        WHEN imie !~ '^[A-Z][a-z]+$'
            THEN RAISE EXCEPTION 'Imię powinno zawierać tylko znaki ASCII, pierwsza litera
powinna być duża.';
        WHEN nazwisko !~ '^[A-Z][a-z]+$'
            THEN RAISE EXCEPTION 'Nazwisko powinno zawierać tylko znaki ASCII, pierwsza litera
powinna być duża.';
        WHEN (SELECT COUNT(t.teacher_usos_id) FROM teachers t WHERE t.first_name = imie AND
t.last_name = nazwisko) > 0
            THEN RAISE EXCEPTION 'Nowy prowadzący nie został dodany, ponieważ podane dane
prowadzącego już znajdują się w bazie.';
        ELSE INSERT INTO public.teachers (teacher_usos_id, first_name, last_name)
             VALUES (NEXTVAL('public.teachers_custom_usos_id_seq'), imie, nazwisko);
             result = 'Nowy prowadzący został pomyślnie dodany do bazy.';
        END CASE;
END;
$$;
```

- **„Dodaj nowy budynek”**
  Procedura przyjmuje wszystkie potrzebne dane na temat nowego budynku, czyli: jego ID, nazwę i współrzędne geograficzne. Za pomocą składni „CASE” procedura sprawdza czy budynek nie znajduje się już w bazie danych, a także czy nie znajduje się poza terenem Rzeszowa bądź Stalowej Woli.

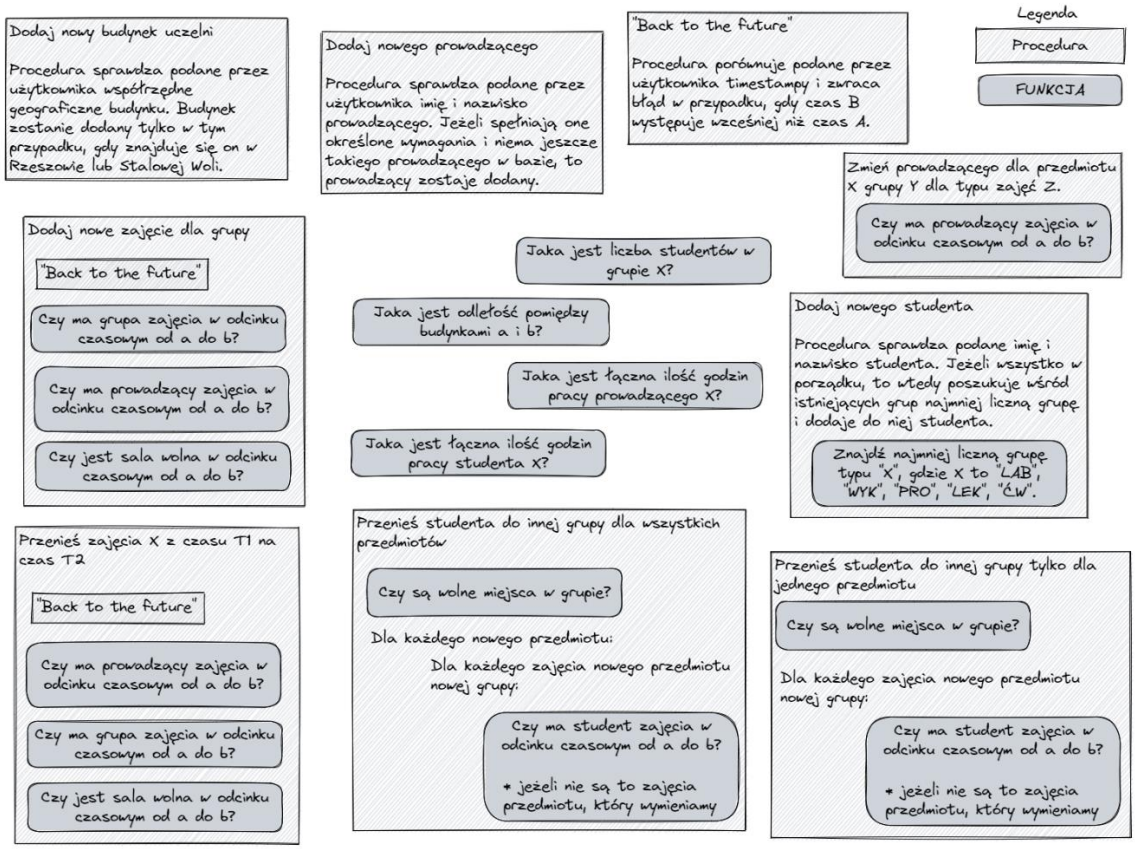
```
CREATE OR REPLACE PROCEDURE dodaj_nowy_budynek(
    IN id_budynku public.buildings.building_id%TYPE,
    IN nazwa_budynku public.buildings.building_name%TYPE,
    IN dlugosc_geo public.buildings.longitude%TYPE,
    IN szerokosc_geo public.buildings.latitude%TYPE,
    OUT result TEXT
)
    LANGUAGE plpgsql
AS

$$
BEGIN
    CASE
        WHEN (SELECT COUNT(b.building_id)
              FROM buildings b
              WHERE b.building_id = id_budynku
                 OR b.building_name = nazwa_budynku) > 0
            THEN RAISE EXCEPTION 'Nowy budynek nie został dodany, ponieważ podana nazwa lub id
budynku już znajduje się w bazie.';
        WHEN (SELECT LENGTH(id_budynku)) = 0 OR (SELECT LENGTH(nazwa_budynku)) = 0
            THEN RAISE EXCEPTION 'Nazwa lub id budynku nie może być pustym';
        WHEN NOT (
                (dlugosc_geo > 21.903 AND dlugosc_geo < 22.080 AND szerokosc_geo > 49.977 AND
szerokosc_geo < 50.136)
                OR
                (dlugosc_geo > 22.003 AND dlugosc_geo < 22.150 AND szerokosc_geo > 50.486 AND
szerokosc_geo < 50.665)
            )
            THEN RAISE EXCEPTION 'Budynek nie został dodany, ponieważ podane współrzędne
geograficzne nie znajdują się w Rzeszowie lub Stalowej Woli.';
        ELSE INSERT INTO public.buildings (building_id, building_name, longitude, latitude)
             VALUES (id_budynku, nazwa_budynku, dlugosc_geo, szerokosc_geo);
             result = 'Nowy budynek został pomyślnie dodany do bazy.';
        END CASE;
END;
$$;
```

- **„Automatycznie dodaj studenta”**
  Procedura przyjmuję imię i nazwisko nowego studenta jako dane wejściowe. Najpierw sprawdza czy dane studenta składają się ze znaków ASCII i czy zaczynają się na wielką literę. Następnie student zostaje dodany do bazy danych, a procedura poszukuje grupy z najmniejszą ilością studentów za pomocą funkcji „grupa z minimalną ilością studentów”. Po odnalezieniu dla studenta odpowiednich grup procedura zwraca wiadomość z informacjami do jakich grup został przypisany student.

```
CREATE OR REPLACE PROCEDURE auto_dodaj_studenta(
    IN imie public.users.first_name%TYPE,
    IN nazwisko public.users.last_name%TYPE,
    OUT result TEXT
)
    LANGUAGE plpgsql
AS
$$
DECLARE
    new_user_usos_id   public.users.usos_id%TYPE;
    gr_lek             public.unit_groups.group_number%TYPE;
    gr lab pro         public.unit groups.group number%TYPE;
    gr_cw              public.unit_groups.group_number%TYPE;
    gr_wyk             public.unit_groups.group_number%TYPE;
    cur_usos_units CURSOR FOR SELECT DISTINCT uu.usos_unit_id, uu.group_type
                              FROM public.usos_units uu
                                       INNER JOIN unit_groups ung ON uu.usos_unit_id =
ung.usos_unit_id
                                       INNER JOIN users_groups usg ON ung.unit_group_id =
usg.group_id
                                       INNER JOIN users u ON usg.user_usos_id = u.usos_id
                              WHERE u.usos_id = 234394;
    single_usos_unit   RECORD;
    temp_unit_group_id public.unit_groups.unit_group_id%TYPE;
    temp gr number     public.unit groups.group number%TYPE;
BEGIN
    CASE
        WHEN imie !~ '^[A-Z][a-z]+$'
            THEN RAISE EXCEPTION 'Imię powinno zawierać tylko znaki ASCII, pierwsza litera
powinna być duża.';
        WHEN nazwisko !~ '^[A-Z][a-z]+$'
            THEN RAISE EXCEPTION 'Nazwisko powinno zawierać tylko znaki ASCII, pierwsza litera
powinna być duża.';
        ELSE BEGIN
            -- Dodajemy nowego studenta do bazy.
            new_user_usos_id = NEXTVAL('public.users_custom_usos_id_seq');
            INSERT INTO public.users (usos id, first name, last name)
            VALUES (new_user_usos_id, imie, nazwisko);

            -- Poszukiwanie grup o najmniejszej ilości studentów
            gr_lek = grupa_z_min_iloscia_studentow('LEK');
            gr_lab_pro = grupa_z_min_iloscia_studentow('LAB');
            CASE gr_lab_pro
                WHEN 1, 2 THEN gr_cw = 1;
                WHEN 3, 4 THEN gr_cw = 2;
                WHEN 5, 6 THEN gr_cw = 3;
                ELSE RAISE EXCEPTION 'Lab/pro group (%) is unknown.', gr_lab_pro;
                END CASE;
            gr_wyk = grupa_z_min_iloscia_studentow('WYK');
            -- Dodawanie studenta do każdego typu grupy każdego przedmiotu
            OPEN cur_usos_units;
            LOOP
                FETCH cur_usos_units INTO single_usos_unit;
                EXIT WHEN NOT found;
                CASE single_usos_unit.group_type
                    WHEN 'WYK' THEN temp_gr_number = gr_wyk;
                    WHEN 'LAB', 'PRO' THEN temp_gr_number = gr_lab_pro;
                    WHEN 'ĆW' THEN temp_gr_number = gr_cw;
                    WHEN 'LEK' THEN temp_gr_number = gr_lek;
                    ELSE RAISE EXCEPTION 'Group type (%) is unknown.',
single_usos_unit.group_type;
                    END CASE;
                SELECT ung.unit_group_id
                INTO temp_unit_group_id
                FROM public.unit_groups ung
                WHERE ung.group_number = temp_gr_number
                  AND ung.usos_unit_id = single_usos_unit.usos_unit_id;
                INSERT INTO public.users_groups (user_usos_id, group_id) VALUES
(new_user_usos_id, temp_unit_group_id);
                result = FORMAT('Studnet został został dodany do grup WYK %s, ĆW %s, LAB-PRO
%s i LEK %s.',
                                gr_wyk, gr_cw, gr_lab_pro, gr_lek);
            END LOOP;
            CLOSE cur_usos_units;
        END;
        END CASE;
```

2.3 Całkowity model wewnętrzny

Na poniższym rysunku widzimy już wszystkie procedury i funkcji zebrane w jednym miejscu.



Rysunek 2.2 Wizualna reprezentacja przygotowanych funkcji i procedur w PostgreSQL.

2.4 Wykorzystane struktury i typy danych języka PL/pgSQL

Głównym celem przedmiotu Aplikacje Bazodanowe jest nauczenie się korzystać z bardziej skomplikowanych struktur i typów danych języka PL/pgSQL. Dlatego w tym podrozdziale zostaną wylistowane użyte przez nas elementy z krótkim opisem do czego służą.

a) If'y

Wyrażenie „if" stosowane jest w celu wyegzekwowania alternatywnych poleceń bazując na pewnych warunkach. Przykład:

```
IF liczba_kolidujacych_zajec > 0 THEN
    RETURN FALSE;
ELSIF liczba_kolidujacych_zajec = 0 THEN
    RETURN TRUE;
END IF;
```

b) Case'y

Case pozwala w prosty sposób zarządzać dużą ilością warunków. Dobrze widoczne jest to na przykładzie:

```
CASE single_usos_unit.group_type
    WHEN 'WYK' THEN temp_gr_number = gr_wyk;
    WHEN 'LAB', 'PRO' THEN temp_gr_number = gr_lab_pro;
    WHEN 'ĆW' THEN temp_gr_number = gr_cw;
    WHEN 'LEK' THEN temp_gr_number = gr_lek;
    ELSE RAISE EXCEPTION 'Group type (%) is unknown.',
single_usos_unit.group_type;
    END CASE;
```

c) For'y

„For" pozwala na powtórzeniu serii poleceń

```
FOR single_group_to_move IN SELECT * FROM unit_groups_to_move
    LOOP
        FOR activities_of_the_moving_group IN
            SELECT a.start_time,
                   a.end_time
            FROM activities a
                     INNER JOIN unit_groups u ON u.unit_group_id = a.unit_group
            WHERE u.usos_unit_id = single_group_to_move.usos_unit_id
              AND u.group_number = nr_nowej_grupy
            LOOP
                IF
                        zajecia_studenta_w_danym_czasie(
                                id_studenta,
                                activities_of_the_moving_group.start_time,
                                activities_of_the_moving_group.end_time,
                                lista_grup_studenta) = FALSE
                THEN
                    nie_ma_zajec = FALSE;
                END IF;
            END LOOP;
    END LOOP;
```

d) Loop'y

„Loop" tak samo jak „for" pozwala na wykonanie serii poleceń

```
OPEN cur_usos_units;
LOOP
    FETCH cur_usos_units INTO single_usos_unit;
    EXIT WHEN NOT found;

    CASE single_usos_unit.group_type
        WHEN 'WYK' THEN temp_gr_number = gr_wyk;
        WHEN 'LAB', 'PRO' THEN temp_gr_number = gr_lab_pro;
        WHEN 'ĆW' THEN temp_gr_number = gr_cw;
        WHEN 'LEK' THEN temp_gr_number = gr_lek;
        ELSE RAISE EXCEPTION 'Group type (%) is unknown.', single_usos_unit.group_type;
        END CASE;

    SELECT ung.unit group id
    INTO temp_unit_group_id
    FROM public.unit_groups ung
    WHERE ung.group_number = temp_gr_number
      AND ung.usos_unit_id = single_usos_unit.usos_unit_id;

    INSERT INTO public.users_groups (user_usos_id, group_id) VALUES (new_user_usos_id,
temp_unit_group_id);
    result = FORMAT('Studnet został został dodany do grup WYK %s, ĆW %s, LAB-PRO %s i LEK
%s.',
                    gr_wyk, gr_cw, gr_lab_pro, gr_lek);
END LOOP;
CLOSE cur_usos_units;
```

e) **Exception**

Konstrukcja „Exception” pozwala wyłapywać błędy i dokonywać różnych działań w zależności od tego jakiego typu błąd został wyrzucony.

```
BEGIN
-- ...
EXCEPTION
    WHEN DATA_EXCEPTION THEN
        BEGIN
            RAISE NOTICE 'Some assigned rooms to the group % are too small.', nowa_grupa_id;
            RAISE EXCEPTION 'Student nie może być przypisany do wkazanej grupy, ponieważ niema
w niej wolnych miejsc';
        END;
END;
```

f) **Raise**

Polecenie „Raise” jest używane by wyświetlić ostrzeżenia lub błędy.

```
IF czas_a > czas_b THEN
    RAISE INVALID_PARAMETER_VALUE USING MESSAGE = 'Czas początkowy jest większy od czasu
końcowego.';
END IF;
```

g) **Record**

„Record” to typ danych przyjmujący strukturę dowolnego wiersza tabeli, który został do niego przypisany. Użyty w celu przechowania w sobie kolejnych wierszy z tabeli „activities” by móc zliczyć godziny pracy wykładowcy w ciągu semestru.

```
DECLARE
    zajecie      RECORD;
    grupa        INT;
    laczny_czas DOUBLE PRECISION := 0.0;
BEGIN
    FOR grupa IN SELECT unit_group
                 FROM group_teacher
                 WHERE teacher = wykladowca
        LOOP
            FOR zajecie IN SELECT start_time, end_time
                           FROM activities
                           WHERE unit_group = grupa
                LOOP
                    laczny_czas := laczny_czas +
                                   EXTRACT(EPOCH FROM AGE(zajecie.end_time, zajecie.start_time))
                                       / 3600 AS difference;
                END LOOP;
        END LOOP;
    RETURN laczny_czas;
END;
```

h) **Cursor**

Typ danych „Cursor” pozwala pobierać wiersze z tabeli pojedynczo jeden po drugim. Umożliwia to pracę na dużych zbiorach danych, ponieważ nie musimy wgrywać do pamięci komputerowej całej tabeli. Wykorzystany został w celu demonstracyjnym w następnym odcinku kodu:

```
OPEN cur_usos_units;
LOOP
    FETCH cur_usos_units INTO single_usos_unit;
    EXIT WHEN NOT found;

    -- ...

END LOOP;
CLOSE cur_usos_units;
```

i) **Array**

Pozwala kolumnom tabel być zdefiniowanymi jako tablice wielowymiarowe o zmiennej długości. W przykładzie został użyty do zapisania listy ID starych zajęć, które mają być zignorowane przy sprawdzaniu czy student ma zajęcia w danym czasie:

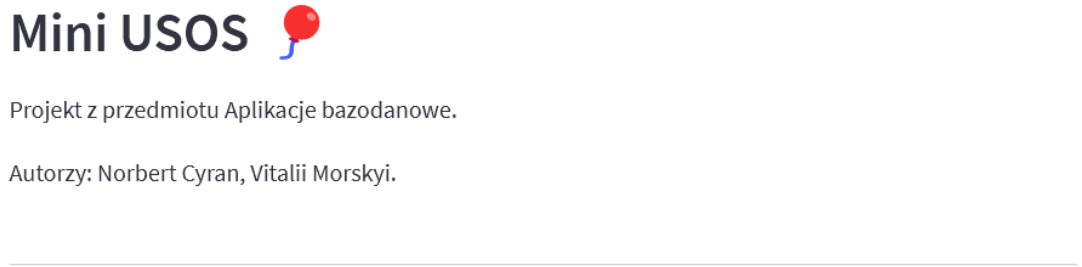
```
CREATE OR REPLACE FUNCTION zajecia_studenta_w_danym_czasie(
    student users.usos_id%TYPE,
    czas_poczatkowy activities.start_time%TYPE,
    czas_koncowy activities.end_time%TYPE,
    ignoruj_liste_usos_unit_ids INTEGER ARRAY
)
    RETURNS bool
    LANGUAGE plpgsql
AS
$$
DECLARE
    liczba_kolidujacych_zajec INT;
BEGIN
    SELECT COUNT(activity_id)
    INTO liczba_kolidujacych_zajec
    FROM activities act
             INNER JOIN unit_groups ung ON act.unit_group = ung.unit_group_id
             INNER JOIN users_groups usg ON ung.unit_group_id = usg.group_id
             INNER JOIN users u ON u.usos_id = usg.user_usos_id
    WHERE (start_time >= czas_poczatkowy AND start_time < czas_koncowy
        OR end_time > czas_poczatkowy AND end_time <= czas_koncowy)
      AND u.usos_id = student
      AND ung.usos_unit_id != ALL (ignoruj_liste_usos_unit_ids);

    IF liczba_kolidujacych_zajec > 0 THEN
        RETURN FALSE;
    ELSIF liczba_kolidujacych_zajec = 0 THEN
        RETURN TRUE;
    END IF;
END;
$$;
```

3 Interfejs aplikacji oraz podłączenie się do PostgreSQL

3.1 Streamlit jako podstawa interfejsu

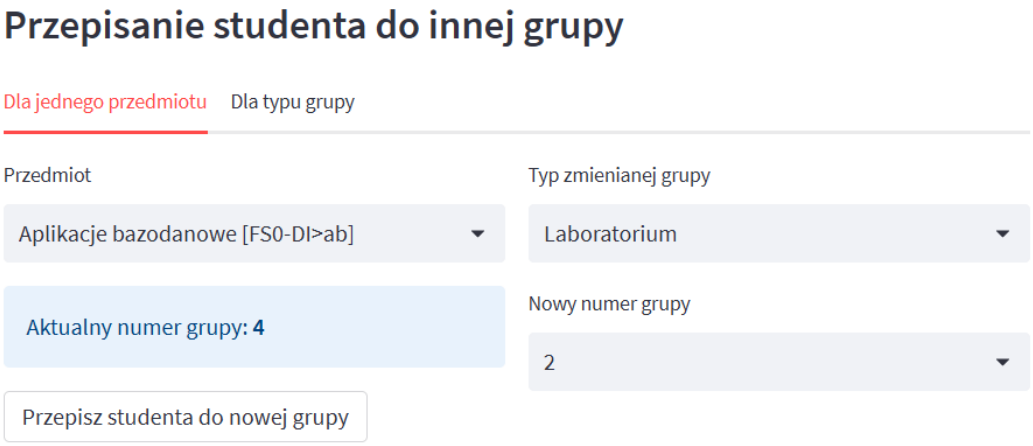
Streamlit – biblioteka w języku Python, która pozwala w bardzo łatwy sposób tworzyć interaktywne dashbordy. Biblioteka w pełni wspiera język znaczników „Markdown", co pozwala w bardzo łatwy sposób formatować tekst.

```
st.header("Mini USOS 🎈")

st.markdown("Projekt z przedmiotu Aplikacje bazodanowe.")
st.markdown("Autorzy: Norbert Cyran, Vitalii Morskyi.")

st.markdown("---")
```



Tworzenie elementów sterowania lub wprowadzania danych też nie stanowi żadnego problemu: Streamlit posiada dużą liczbę predefiniowanych narzędzi.

```
tab1_one_subject, tab2_all_groups_of_type = st.tabs(["Dla jednego przedmiotu", "Dla typu
grupy"])
with tab1_one_subject:
    col1_group_selection, col2_group_selection = st.columns(2)
    with col1_group_selection:
        unique_readable_course_id = st.selectbox(label="Przedmiot",
options=courses_df.unique_readable_course_id)
    with col2_group_selection:
        group_type_name = st.selectbox(label="Typ zmienianej grupy",
options=group_types_df.group_type_name,
                                       key="single_subject_group_type")
    if user_unit_group is not None:
        with col1_group_selection:
            group_display = st.info(f"Aktualny numer grupy: **{old_group_number}**")
        with col2_group_selection:
            unit_group_number = st.selectbox(label="Nowy numer grupy",
options=unit_groups_df.group_number,
                                             key="single_subject_group_number")
        with col1_group_selection:
            if st.button("Przepisz studenta do nowej grupy",
key="single_subject_accept_button"):
                if old_unit_group_id != new_unit_group_id:
                    if change_single_subject_group_answer[1]:
                        tab1_one_subject.success(change_single_subject_group_answer[0][0])
                        group_display.info(f"Aktualny numer grupy: **{unit_group_number}**")
                    else:
                        tab1_one_subject.error(change_single_subject_group_answer[0])
                else:
                    tab1_one_subject.warning(
                        "Nowa grupa nie różni się od już aktualnej grupy. Nie wprowadzono
zmian.")
    else:
        st.warning("Osoba nie jest przypisana do żadnej grupy dla wybranego przedmiotu.")
```



Biorąc pod uwagę powyższe fakty, Streamlit jest idealnym narzędziem do przygotowania interfejsu użytkownika w danym projekcie.

3.2 Podłączenie się do PostgreSQL

Aby podłączyć system bazodanowy PostgreSQL do Pythona wykorzystujemy biblioteki psycopg2. Pozwala ona dokonywać następujące czynności:

- Tworzyć połączenie z bazą danych PostgreSQL.
- Tworzyć kursory w bazie danych, w których można uruchamiać dowolne kwerendy lub procedury.

```
def connect(self):
    """Connect to the PostgreSQL database server."""
    try:
        logging.info("Connecting to the PostgreSQL database...")
        self.conn = psycopg2.connect(**app_config.raw_sections["postgresql"])
        self.sqlalchemy_engine = sqlalchemy.create_engine(
            "postgresql+psycopg2://", creator=lambda: self.conn
        )

        # Display PostgreSQL version
        cur = self.conn.cursor()
        cur.execute("SELECT version()")
        logging.info(f"PostgreSQL version:\t{cur.fetchone()}")
        cur.close()

    except (Exception, psycopg2.DatabaseError):
        log_exception()

    if self.conn is not None:
        self.is_connected = True
```

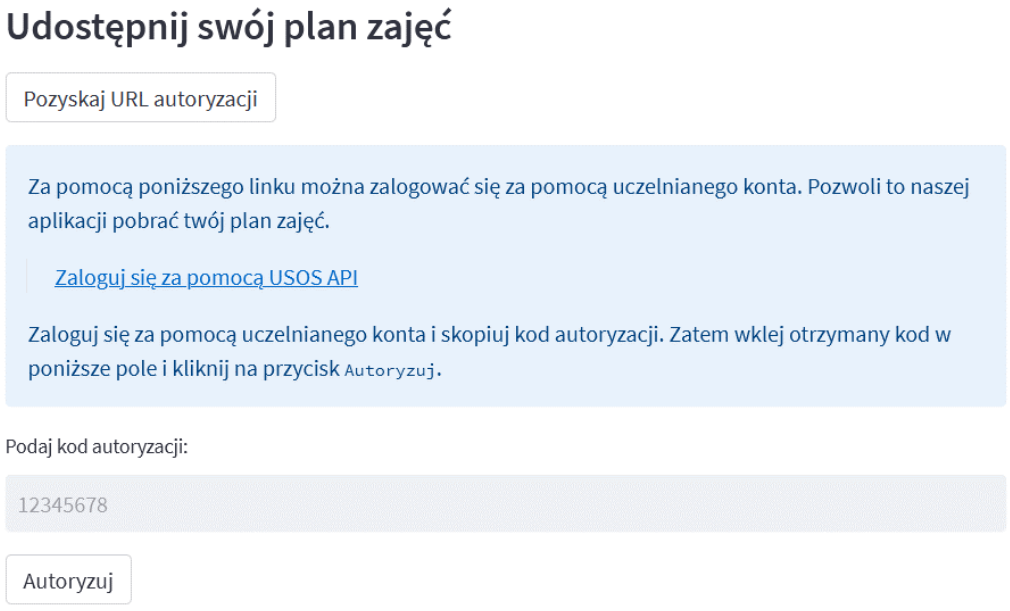
Ciekawy fakt: łącznie cała aplikacja tworzy 28 różnych kursorów, z których 27 wykonywane są z poziomu Pythona i jeden przykładowy został dodany w procedurze.

Do komunikacji z bazą danych została również użyta biblioteka Pythona o nazwie pandas. Ona również korzysta z biblioteki psycopg2, ale zwracany przez nią wynik jest już ładnie przekształcony na ramkę danych – szeroko stosowaną w Pyrhonie reprezentację tabeli. Znacznie ułatwia to pracę z bazą danych w przypadku, gdy zapytanie zwraca nie jedną wartość, a od razu kilka.

```python
def get_unit_group_activities(self, unit_group_id: int):
    query = (
        "SELECT * "
        "FROM activities a "
        "         INNER JOIN unit_groups ug ON a.unit_group = ug.unit_group_id "
        "WHERE ug.unit_group_id = %(unit_group_id)s;"
    )
    df = pd.read_sql(
        query, self.sqlalchemy_engine, params={"unit_group_id": unit_group_id}
    )
    logging.debug(f"Get unit group activities {unit_group_id=}.")
    return df
```

3.3 Pobieranie danych za pomocą USOS API

Na pierwszym widoku aplikacji dostępny jest przycisk pozwalający na zalogowanie się w systemie USOS. W takim przypadku aplikacja otrzymuje możliwość pobrać wszystkie dane, które dotyczą planu zajęć użytkownika oraz jego kolegów na roku. Nie jest jednak widoczny skład grup zajęciowych, w których nie uczestniczy zalogowany użytkownik. W związku z tym dla większości studentów dostępne są tylko grupy wykładowe.



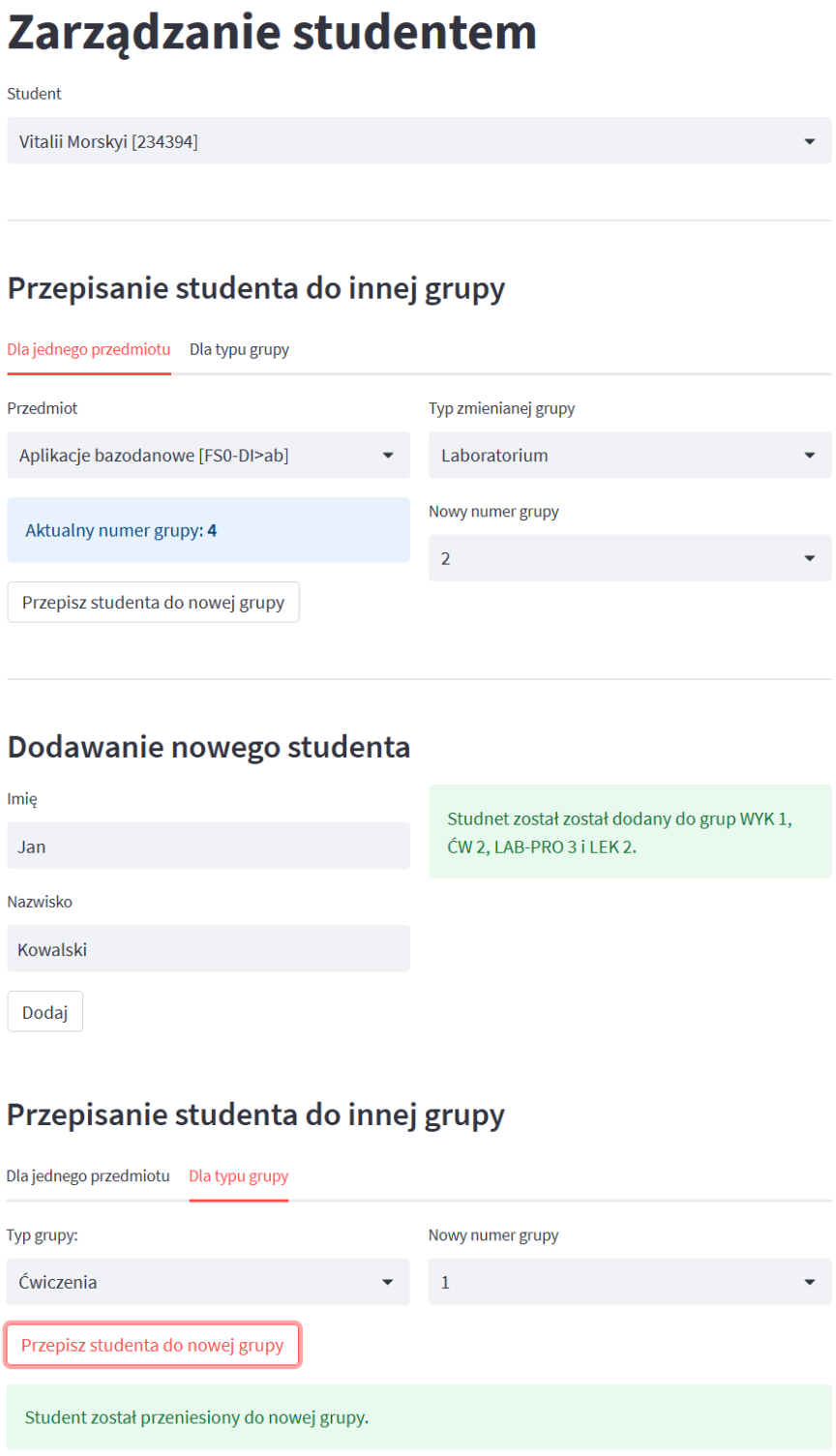
4 Wynikająca aplikacja końcowa oraz wnioski

4.1 Widoki i możliwości aplikacji

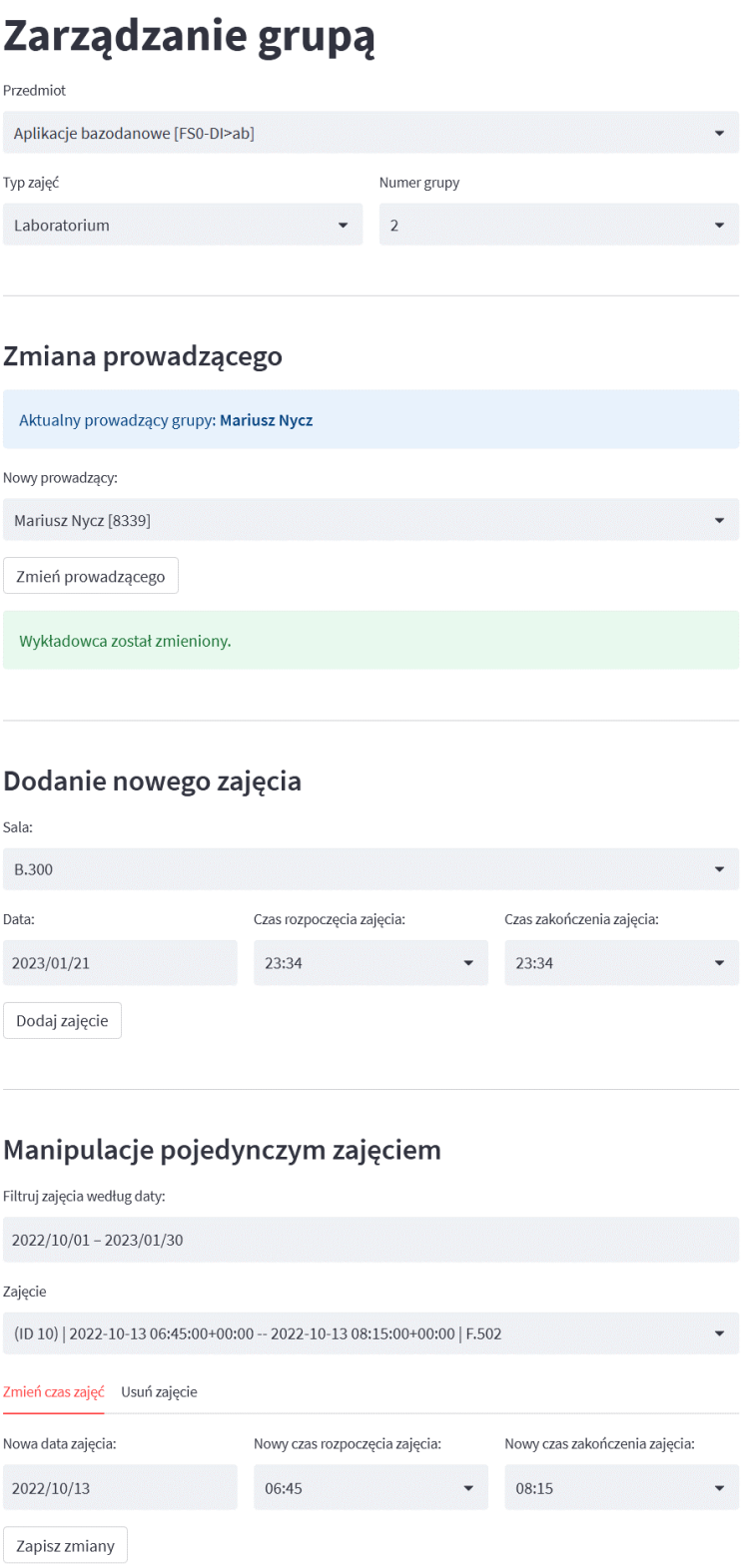
Aplikacja składa się z pięciu podstron internetowych.

Na pierwszej stronie podane są ogólne informacje na temat aplikacji, niektóre diagramy, pokazujące jej wewnętrzny układ oraz pole do logowania się za pomocą konta uczelnianego (w tym celu trzeba kliknąć na przycisk).

Druga strona pozwala zarządzać poszczególnymi studentami, które znajdują się w bazie. Takiemu studentowi można zmienić grupę zajęciową dla jednego przedmiotu lub dla całego typu zajęć (np przenieść studenta do 3 grupy laboratoryjnej dla wszystkich przedmiotów, które posiadają taki typ zajęć). Dostępna również jest opcja dodawania nowego studenta do bazy. Zostanie jemu automatycznie przydzielona grupa dla każdego typu zajęć w taki sposób, aby grupa ta była jak najmniejsza.



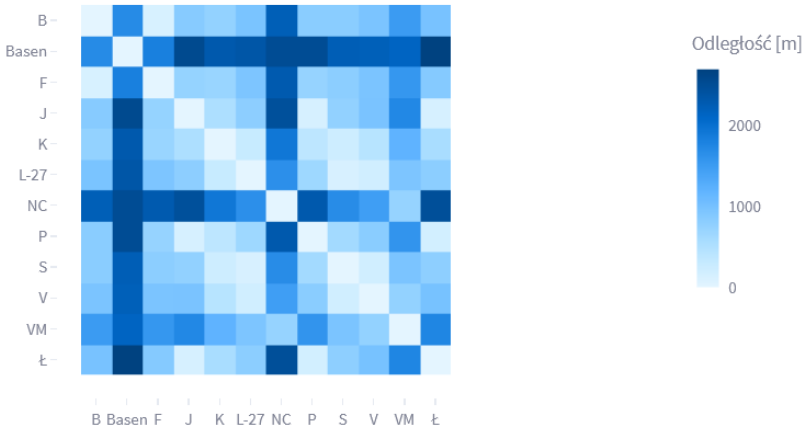
Na trzeciej stronie możemy właśnie zarządzać grupami studenckimi. Dostępne są opcję zmiany prowadzącego grupy, dodania nowego zajęcia, usunięcia lub przesunięcia istniejącego zajęcia grupy.



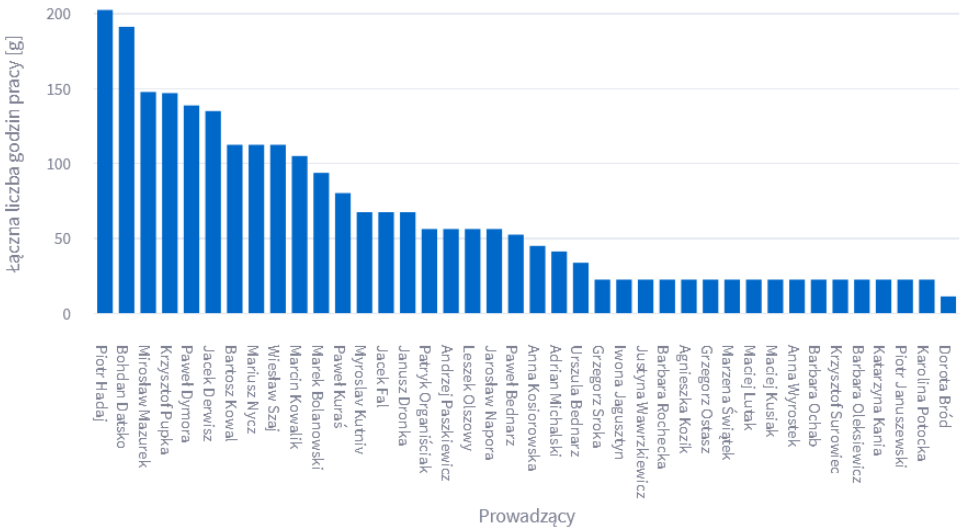
Czwarta strona zawiera dwa przykładowe wykresy. Pierwszy pokazuje odległość pomiędzy poszczególnymi budynkami, gdyż drugi – łączną liczbę godzin pracy prowadzących z drugim i trzecim rocznikami Inżynierii i Analizy Danych.

Statystyki

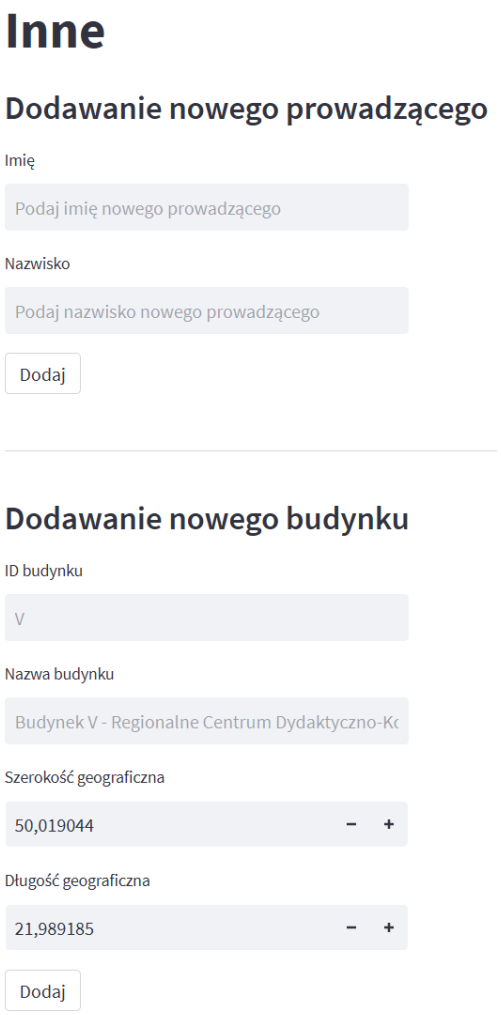
Odległości między budynkami Politechniki Rzeszowskiej



Spędzony czas prowadzących z IiAD



Na ostatniej stronie podano dwie funkcjonalności: dodawanie nowego budynku uczelnianego oraz dodawanie nowego prowadzącego. Funkcjonalności te trudno było przydzielić do innych stron.



4.2 Podsumowanie doświadczeń wynikających z wykonania projektu

Wynikiem danego projektu powstała aplikacja oparta o system bazodanowy PostgreSQL i język programowania Python. Z pewnością możemy powiedzieć, że najbardziej kształcącą częścią było przygotowanie procedur i funkcji w języku PL/pgSQL. Dość ciekawym doświadczeniem było przygotowanie funkcji liczącej odległość pomiędzy budynkami uczelni za pomocą współrzędnych geograficznych. Najwięcej problemów napotkaliśmy przy pierwszych próbach uruchomienia procedur PL/pgSQL z poziomu języka Python. Jak się okazało, bardzo rzadko są używane w taki sposób i dlatego było dość mało dokumentacji na ten temat, ale poradziliśmy sobie.

Gdybyśmy chcieli naprawdę używać tej aplikacji, to potrzebowalibyśmy jeszcze w znacznym stopniu ulepszyć ją. Do przykładu, aplikacja nie posiada żadnego systemu uwierzytelniania, a więc każdy może zobaczyć treści wyświetlane na tej stronie. Przydatnymi będzie również odporności aplikacji na błędy użytkownika (chociaż to już jest zrobiono w dość dobrym stopniu, nadal pojawiają się przypadki, gdy użytkownik robi jakieś nieprzewidziane czynności niepożądane).