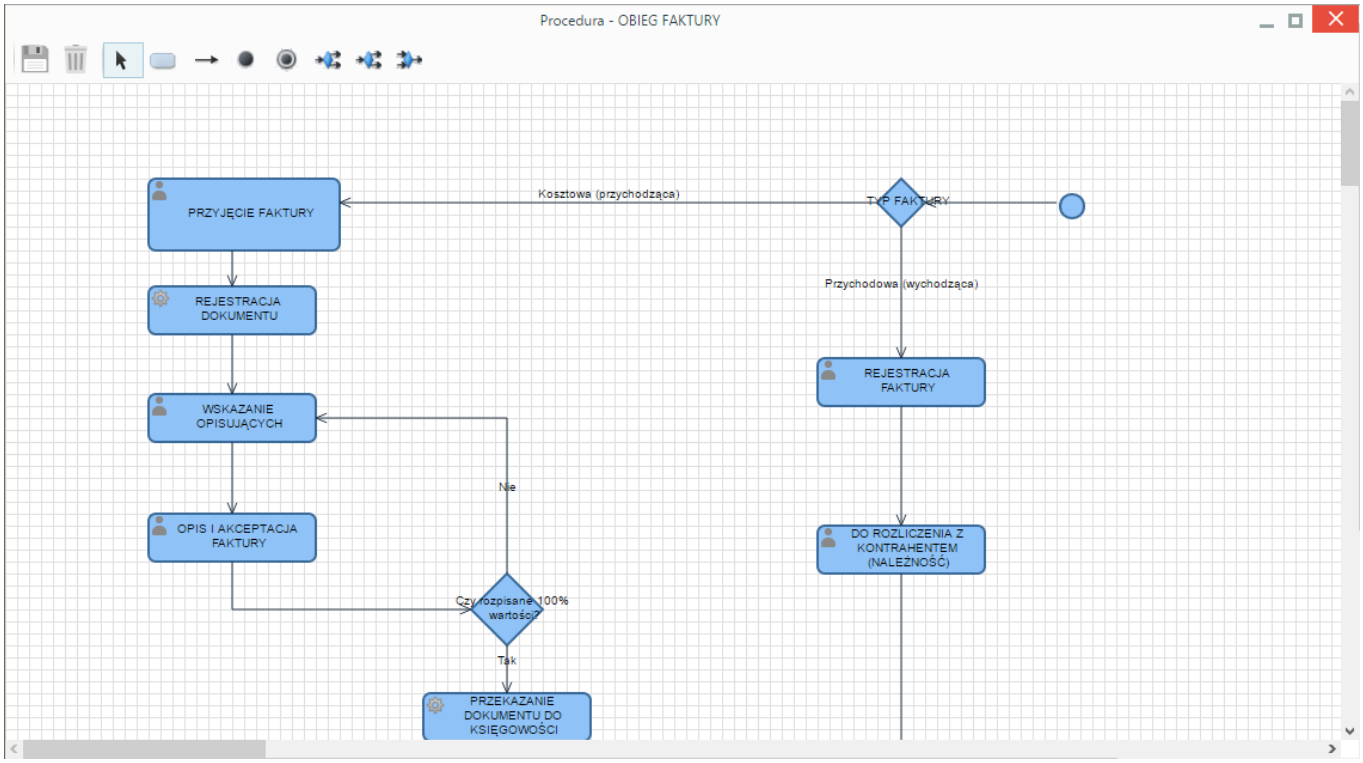
Automatyzacja procesów workflow

Podstawowe informacje

Procedury workflow oparte są o notację BPMN i uwzględniają wszystkie najważniejsze elementy tej notacji. Składają się na nią:

- Etapy (Czynności - bloczki)
- Przejścia (strzałki)
- Decyzje (diament powodujący wyświetlenie decyzji dla użytkownika)
- Warunki (diament dokonujący ewaluacji warunków SQL)
- Złączenia (JOIN - w przypadku wymagania spełnienia poprzednich etapów)
- Pętle multi-instance z podprocesami (etapy ze znakiem +)
- Dane wejściowe (możliwość pobrania od użytkownika danych różnego typu: znakowe, liczbowe, listy pracowników, listy wyboru pobierane ze słowników kwerendami SQL itp).
- Właściwości (definiowalne zmienne)
- Przypisania (możliwość operowania na zmiennych)



Konfiguracja procedur pozwala tworzyć mapy procesów odnoszące się zarówno do dokumentów jak i spraw. Przykłady wykorzystania dostępne są tutaj: Wykorzystanie procedur

Komendy

Komendy mogą być wywoływane na aktywacji lub zakończeniu etapu. Opis komend i lista parametrów

Opis tworzenia własnych komend

Dla zaawansowanych

W workflow biorą udział następujące tabele:

- procedures_def - tabela procedur - przechowuje informacje o procedurze np. Zatwierdzenie faktury kosztowej
- stages_def - tabela etapów - przechowuje definicje poszczególnych etapów np. Akceptacja Prezesa
- stages - instancje etapów - przechowuje informacje o zapisanych etapach konkretnych procesów: spraw, dokumentów

- proc_actions - akcje powiązane z procedurami lub z etapami, wykonują się przed lub po zapisie np. beforeStageChange
- action_commands - komendy wykonywane przez system na akcjach - wybierane spośród zawartych w katalogu commands - można dodać parametry, które dodają się do standardowych dwóch Obiektu Akcji oraz obiektu encji powiązanej z wykonywaną akcją np. Dokument albo Sprawa

Wykorzystanie własności, danych wejściowych i przypisań

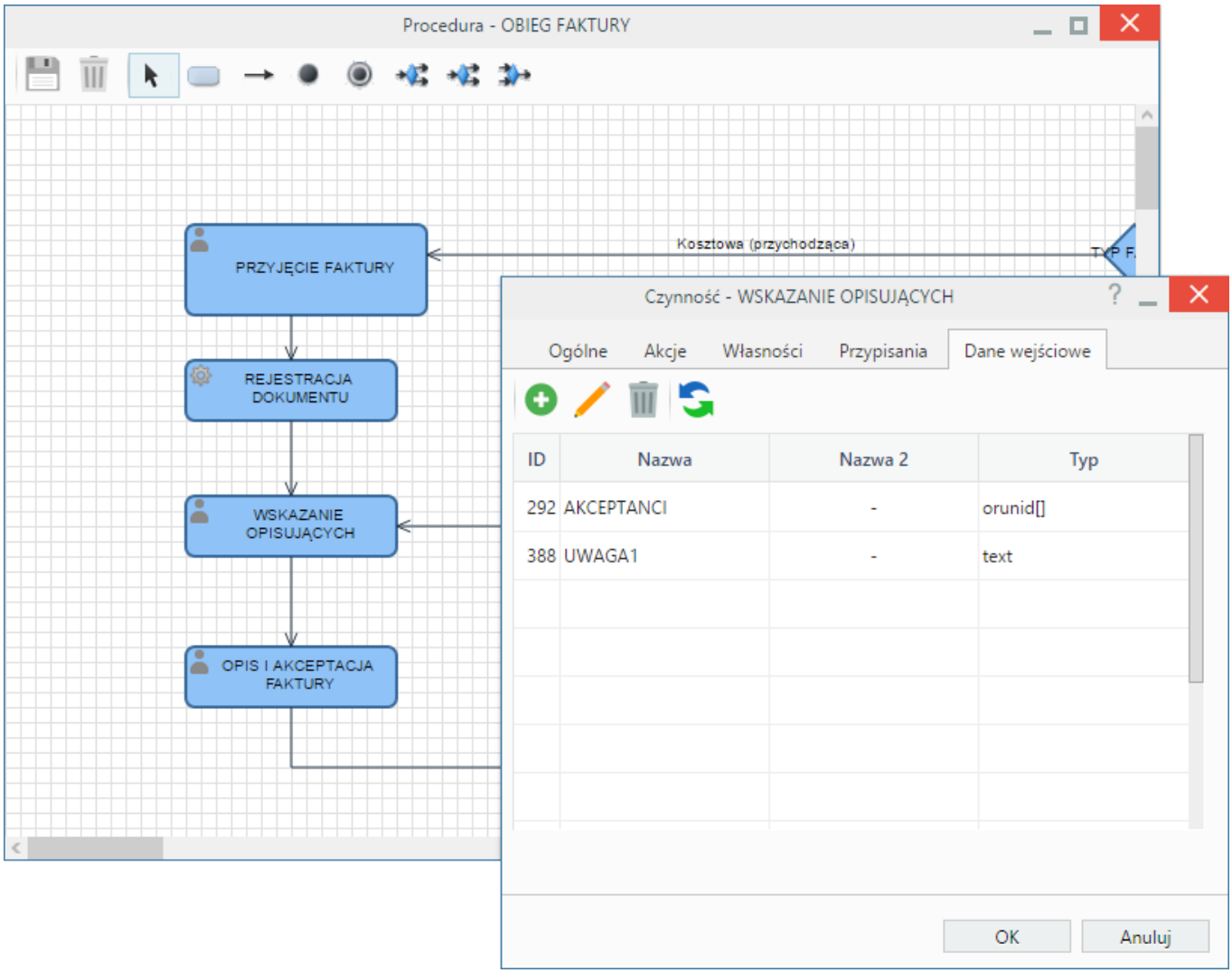
Potężne możliwości silnika workflow systemu eDokumenty możliwe są m.in. dzięki wykorzystaniu parametrów i zmiennych które mogą być dynamicznie przetwarzane podczas wykonywania procedury. Dane mogą być pobierane od użytkownika, ale również przetwarzane przez sam workflow.
Do danej wejściowej i własności odwołujemy się (w warunkach lub przypisaniach) poprzez nazwę poprzedzoną znakiem "$" oraz całość zamykamy w nawiasy "{}" (np. {$Akceptant}).

Dane wejściowe

Dane wejściowe służą tym samym czym odczyt standardowego wejścia w konsoli czy programie (czyli pobraniu od użytkownika znaków). Można je pobierać z różnych formantów (pól tekstowych, list wyboru, list pracowników). Najciekawszą opcją jest opcja SELECT która pozwala zdefiniować dowolną kwerendę SQL zwracającą potrzebną nam w danym etapie listę (np. kierowników, księgowych, zasobów itp). Przykładowa lista dla atrybutu CZŁONEK ZARZĄDU potrzebna do wyboru osoby podpisującej umowę:

```
{"sql":"SELECT orunid as value, fullnm || ' - ' || ndenam as caption FROM orgtree_view WHERE orunid IN (3,14,15,16)"}
```

Inny przykład to pobranie identyfikatora stanowiska, wystarczy w tym celu wybrać opcję orunid[].



Rodzaje danych wejściowych

Dane wejściowe mogą przyjmować różne typy, m.in: listę. Wówczas w polu Typ należy wybrać "select" a w polu parametry należy wpisać kwerendę zwracającą dwie kolumny:

```
{"sql":"SELECT povcid AS value, place_ || ' ' || dscrpt AS caption
FROM places_of_vcosts
WHERE year__ = (EXTRACT(year FROM CURRENT_DATE))::int AND is_del IS FALSE
ORDER BY caption ASC"}
```

jeśli lista jest na tyle długa że spowalniałaby działanie przeglądarki przy ładowaniu danych, lepiej zastosować pole typu lookup. Poniżej przykład pola dla wyboru projektów:

```
{"sql":"SELECT projid, number || ' ' || projnm AS caption, 'PROJECT' as clsnam FROM projects WHERE is_del IS FALSE AND {FI
```

UWAGA

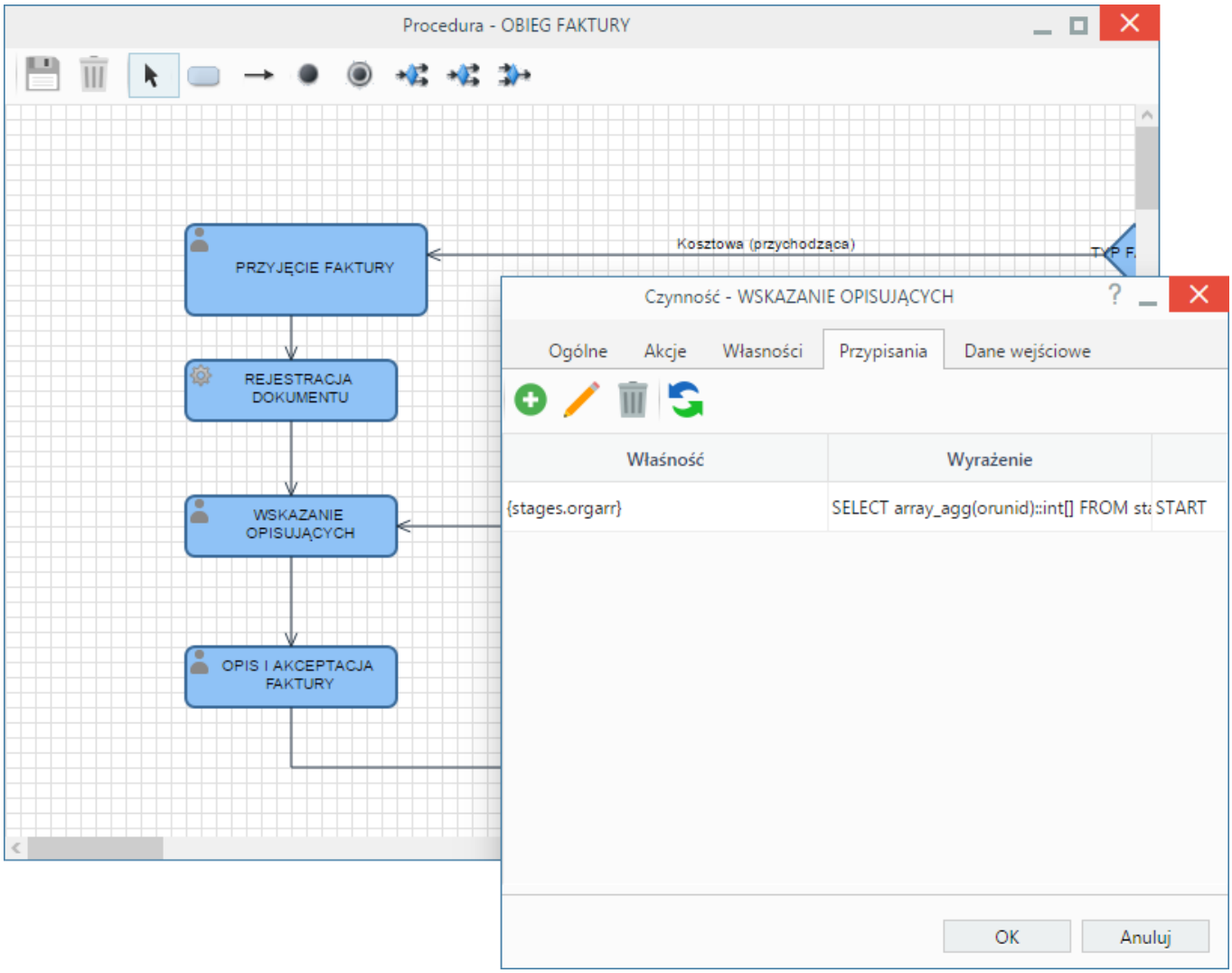
Dane wejściowe jeśli nie są wypełnione zwracają zawsze ciąg znaków 'NULL' oprócz listy wartości która zwraca pusty string . *W przypisaniach i parametrach do komend należy więc używać konstrukcji NULLIF (param1, param2) która zwraca wartość NULL (bazodanową) jeśli param1 jest równe param2. Przykładowo:*

```
-- na formatce pobierane są parametry z listy i pola tekstowego.
-- aby użyć to w przypisaniu do komendy "Ustaw wartość cechy" należy wpisać:
SQL::SELECT COALESCE(NULLIF('{$LISTA2}',''), NULLIF('{$TEXT}','NULL'))
```

Przypisania

Przypisania służą nadaniu wartości dla zmiennych procedury jak również nadaniu wartości atrybutom etapu którego dotyczą. Najczęściej wykorzystuje się przypisanie stanowisk wykonujących etap poprzez przypisanie do własności {stages.orgarr} tablicy (UWAGA! dane muszą być typem tablicowym, w kwerendach należy pamiętać o rzutowaniu).

Patrz przykład:



Tak więc dane wejściowe typu array o nazwie "Akceptant" zostały przypisane do własności {stages.orgarr} (czyli tablicy wykonujących zadanie workflow).

Przypisanie też możemy użyć bez konieczności pobierania danych od użytkownika, możemy je pobrać z bazy danych. Dla tego przykładu gdybyśmy chcieli pobrać Opiekuna klienta którego dotyczy sprzedaż (ze sprawy) dodalibyśmy Przypisanie własności {stages.orgarr} wartości wyrażenia SQL:

```
SELECT ARRAY[o.orunid] FROM contacts c  JOIN processes USING(contid) JOIN orgtree_view o ON o.usr_id = c.macrtk
WHERE prc_id = {processes.prc_id}
```

Przy przypisywaniu danej z danych wejściowych pobranych w etapie należy zwrócić uwagę żeby ustawić czas przypisania na KONIEC.

Przypisania również można stosować do ustawienia własności obiektów których dotyczy workflow. Np aby ustawić własność dokumentu, sprawy czy zapotrzebowania. Odwołanie ma postać {nazwa_tabeli.nazwa_pola_z_bazy} np:

```
{documents.dscrpt} - opis dokumentu
{processes.fxtrid} - termin sprawy
{demand.acorid} - jednostka rozliczeniowa w zapotrzebowaniu
```

Własności

Własności służą do zdefiniowania dodatkowych atrybutów procedury - można je traktować jako zmienne procedury dostępne we wszystkich etapach jak również w parametrach akcji(komend).

Najczęściej zdefiniujemy własność kiedy chcemy aby nadać jej określoną wartość a później wykorzystywać np. w warunkach do sterowania przebiegiem workflow. Np. Zdefiniujmy własność "Czy jest przedpłata", którą napełnimy wartością zależną od wyniku zapytania SQL. Następnie wykorzystamy tą własność w warunku.

Kilka słów o zmiennych

W zapytaniach SQL można używać następujących wyrażeń, które zostaną zastąpione odpowiednimi wartościami:

- {PRC_ID} - prc_id sprawy której dotyczy procedura
- {DOC_ID} - doc_id dokumentu którego dotyczy procedura
- {SOP_ID} - id etapu/czynnności
- {STAGES.PTSTID} - id definicji etapu
- również zawartości obiektów podlegających workflow sprawy i dokumentu np.:
  - {processes.rspuid} - id osoby odpowiedzialnej za sprawę
  - {documents.adduid} - id osoby tworzącej dokument

W dalszej części umieszczone zostały użyteczne konstrukcje przy budowaniu workflow:

Przykłady zapytań

"Magiczne" własności dla procedury

Własności należy najpierw zadeklarować (np. we właściwościach procedury).

{$__orgarrVisible} - (bool) Wartość TRUE spowoduje ukrycie informacji (na panelu workflow) "Do wykonania przez:".
UWAGA! Ustawienie to odnosi się do każdego aktywnego etapu/czynności w procedurze (po wykonaniu danej czynności najpewniej będziemy chcieli przywrócić domyślne ustawienie, czyli pokazać informację o osobach wykonujących).
Przykład:
Na START czynności ukrywamy informację o osobach definiując wartość przypisania tak aby zwróciło FALSE::boolean.. np.:

```
SELECT array_upper({stages.orgarr}::int[], 1) > 1
```

Na zakończenie czynności przywracamy widoczność poprzez przypisanie TRUE do {$__orgarrVisible}:

```
SELECT TRUE
```

1. **{$__panelColor}** - (string) kolor panelu/szarfy workflow (np. #ff0000)

Przykład: Na start przypisujemy np:

```
SELECT '#e74c3c';
```

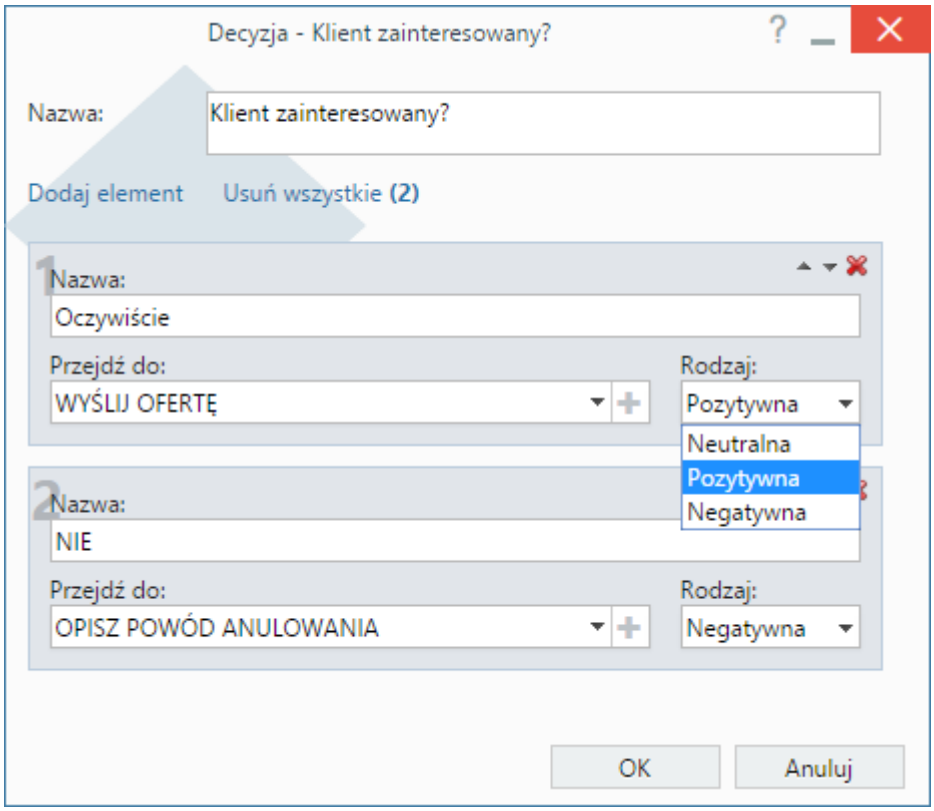
i na koniec etapu

```
SELECT ''
```

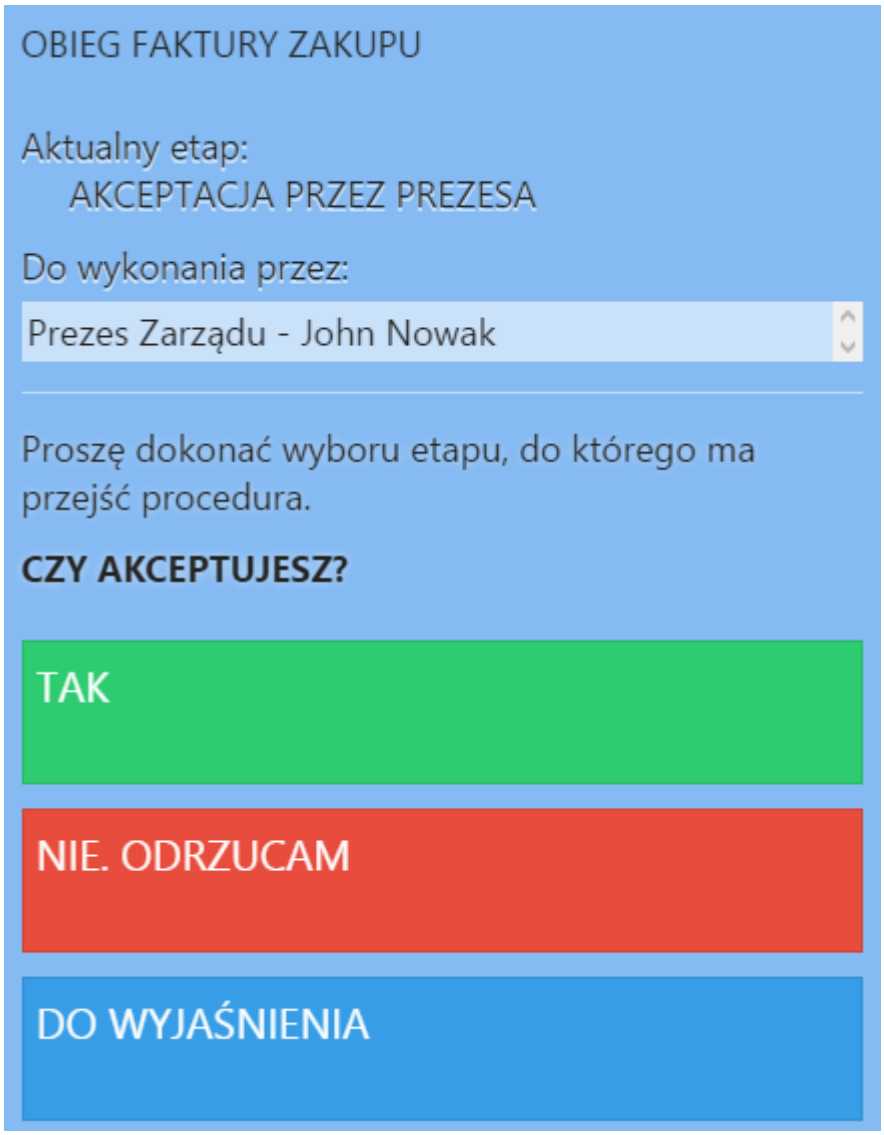
Kolorowe przyciski

Definiując opcje wyświetlające się na szarfie procedury dla decyzji, możemy skorzystać z kodu kolorów, które wizualnie wspomagają ich interpretację, np. jeśli jakaś decyzja ma charakter negatywny, przycisk będzie czerwony, pozytywne akcje będą zielone. Kolor możemy nadać każdemu przyciskowi po wyborze z listy opcji:

- Neutralna - przycisk niebieski
- Pozytywna - przycisk zielony
- Negatywna - przycisk czerwony.



Okno definicji decyzji



Kolorowe przyciski w szarfie procedury

Trochę teorii

Tworzenie prostych procesów workflow nie wymaga dużego przygotowania, ale do tworzenia bardziej zaawansowanych modeli konieczna jest minimalna znajomość teoretycznych zasad rządzących przepływem procesów.

Podstawy teoretyczne

Tips & Tricks

1. Ponieważ zmienne zadeklarowane we danych wejściowych oraz we własnościach zwracają string, dlatego jeśli użyjemy ich w parametrach komend - np. dodaj komentarz wówczas w komentzru pojawi się ciąg znaków wraz z niepożądanymi pojedynczymi cudzysłowami. Aby temu zapobiec w komendzie należy użyć konstrukcji :

```
SQL::SELECT {$KOMENTARZ}
```

Dzięki temu w komentarzu uzyskamy sam oryginalny ciąg znaków wprowadzony w danej wejściowej.

1. Deklarując zmienne globalne przechowujące użytkownika którego identyfikator chcemy zapamiętać gdyż będzie on wykorzystywany w innych etapach (np. do wysłania powiadomienia o przyjęciu wniosku), zaleca się deklarować zmienne globalne jako "int" i zachowywać w nich zawsze identyfikator usr_id. Wówczas w kolejnych przypisaniach lub komendach najłatwiej będzie operować tym parametrem.

Przykładowo chcąc przypisać kolejny etap dla tego użytkownika stosujemy:

```
{stages.orgarr} - SELECT ARRAY(SELECT orunid FROM orgtree_view WHERE usr_id={$OSOBA_WNIOSKUJACA})
```

Optymalizacja bazy danych

Jeżeli tabela stages zawiera pokaźną ilość rekordów (np. powyżej 1M) to można usunąć niepotrzebne rekordy z bazy.
Poniższe zapytanie usunie wszystkie aktywne lub załatwione elementy (czynności, decyzje itp.), dla których cały proces został już zakończony.

```
DELETE FROM stages
USING procedures p
WHERE p.procid = stages.procid AND NOT (stages.is_act OR stages.is_fix) AND (p.comple OR p.cancel);
```

Jeśli chcemy pozostawić historię wykonanych czynności na procedurze możemy usunąć etapy, które nie były użyte w procedurze i pozostawić tylko wykonane etapy.

```
DELETE
FROM stages
WHERE sop_id = any
   (SELECT sop_id
    FROM stages s
    JOIN procedures p USING(procid)
    WHERE NOT (s.is_act
               OR s.is_fix)
      AND (p.comple
           OR p.cancel)
    ORDER BY p.adddat ASC)
```