Informacje o wersji 1.14.0.13
z dn. 20-12-2024

W TEJ WERSJI

- Nowe typy akcji wywoływanych z akcji Algorytm C#
- Zmiany funkcjonalności eksperymentalnych dotyczących procesów BPMN
- Aktualizacja modułu Telerik Reporting
- Poprawki błędów...

PRZED AKTUALIZACJĄ

> Nie jest wymagane wykonanie żadnych dodatkowych czynności.

KOMPATYBILNOŚĆ APLIKACJI

Zaleca się przenoszenie aplikacji między wersjami nAxiom kompatybilnymi na poziomie bazy danych.

Bieżąca wersja bazy danych: 20241030081548

Wersje nAxiom kompatybilne na poziomie bazy danych:

- 1.14.0.12
- 1.14.0.11

NOWE I ZMODERNIZOWANE FUNKCJE

1. Rozszerzenie akcji Algorytm C#

Akcję Algorytm C# rozbudowano o możliwości wywołania akcji raportu oraz definiowania i wykonywania akcji systemowych zapisu rekordu i zmiany statusu dokumentu. Akcje zapisu i zmiany statusu są wykonywane z id użytkownika, który zainicjował wykonanie macierzystej akcji Algorytm C#.

AKCJA GENEROWANIA RAPORTU

Akcja wywołana w ten sposób musi mieć włączony przełącznik `Zapisz wygenerowany plik do załączników`, pobranie wygenerowanego raportu przez przeglądarkę nie jest możliwe.

Wywołanie:

```
var result =  await CSharpScript.ActionRunner.ReportsAction.Instance.RunActionAsync({&BaseApp.Actions.BaseModule.AkcjaRapor
```

Parametr model jest przekazywany jako ciąg znaków. Aby wygenerowany raport został dodany do załącznika, w modelu należy wskazać RecordId (instancję dokumentu) oraz BusinessDocumentId (definicję dokumentu). Dla poprawnego działania uprawnień konieczne jest przekazanie parametru UserId. Parametry używane w zapytaniu raportu należy przekazać w polu Model parametru model. To pole nie może mieć wartości NULL, jeśli parametry nie są przekazywane, musi ono być pustym obiektem JSON ({}):

```
using System;
using System.Linq;
using System.Text;
using Shared.Exceptions;
using Newtonsoft.Json;
using System.Collections.Generic;

var recordId = 122;
var model = new Dictionary<string,object>()
{
  { "Model", @"{""Test"" : 1}" },
  { "RecordId", recordId},
  { "BusinessDocumentId", BusinessDocumentId},
  { "UserId", UserId }
};

var serializedModel = JsonConvert.SerializeObject(model);
var result =  await CSharpScript.ActionRunner.ReportsAction.Instance.RunActionAsync({&BaseApp.Actions.BaseModule.AkcjaRapor

if(!result.IsValid)
{
  throw new TranslatedException("Błąd akcji raportu:", result.ValidationResult);
}

return result.IsValid;
```

W parametrze model można przekazać do akcji raportów wartość kontekstu (pole Model) samej akcji Algorytm C#:

```
var model = new Dictionary<string,object>()
{
  { "Model", Model,
  { "RecordId", recordId},
  { "BusinessDocumentId", BusinessDocumentId},
```

```
  { "UserId", UserId }
};
```

Przy tworzeniu modelu można również pominąć krok serializacji słownika i ręcznie stworzyć odpowiedni ciąg znaków:

```
var model = @$"""  ""RecordId"": ""{recordId}"", ""BusinessDocumentId"": {BusinessDocumentId}}}";
```

AKCJA ZAPISU REKORDU

Wywołanie:

```
var result = await CSharpScript.ActionRunner.SaveRecordAction.Instance.RunActionAsync(request);
```

Definicja typu parametru request:

```
public class SaveDocumentRequest
{
  public int? BusinessDocumentId { get; set; }
  public Dictionary<string, object> Model { get; set; }
  public string RecordId { get; set; }
}
```

- BusinessDocumentId (int): id definicji dokumentu
- RecordId (string): id dokumentu (aktualizacja); w przypadku nowego rekordu należy przekazać wartość 0
- Model (Dictionary<string, object>): słownik par klucz: wartość, wartości pól dokumentu biznesowego

Przykład zapisu nowego dokumentu z przekazaniem wartości Email. Id dokumentu jest przypisane do zmiennej newId:

```
using System;
using System.Linq;
using System.Text;
using Shared.Exceptions;
using Newtonsoft.Json;
using System.Collections.Generic;
using CSharpScript.ActionRunner;

var request = new SaveDocumentRequest()
{
  RecordId = "0",
  BusinessDocumentId = BusinessDocumentId,
```

```
  Model = new Dictionary<string, object>()
  {
    {"Email", "nowe c# nowy record"},
  }
};

var result =  await  CSharpScript.ActionRunner.SaveRecordAction.Instance.RunActionAsync(request);

if(!result.IsValid)
{
  throw new TranslatedException("Tworzenie rekordu nie powiodło się", result.ValidationResult);
}

var newId = result.Result;
```

AKCJA ZMIANY STATUSU

Wywołanie:

```
var result =  await  CSharpScript.ActionRunner.ChangeStatusAction.Instance.RunActionAsync(request);
```

Definicja typu parametru request:

```
public class ChangeStatusRequest
{
  public int? BusinessDocumentId { get; set; }
  public string TransitionInternalId { get; set; }
  public IDictionary<string, object> Model { get; set; }
  public string RecordId { get; set; }
}
```

- TransitionInternalId (GUID): wartość pola InternalId przejścia, które ma zostać wykonane (core.BusinessTransitions.InternalId)
- BusinessDocumentId (int): id definicji dokumentu
- RecordId (string): id dokumentu (aktualizacja); w przypadku nowego rekordu należy przekazać wartość 0
- Model (Dictionary<string, object>): słownik par klucz: wartość, umożliwia przekazanie kontekstu danych dla akcji przed i po wykonywanych podczas zmiany statusu

Przykład zmiany statusu dla bieżącego rekordu (otworzonego w formularzu, na którym znajduje się przycisk wywołujący akcję):

```
using System;
using System.Linq;
```

```
using System.Text;
using Shared.Exceptions;
using Newtonsoft.Json;
using System.Collections.Generic;
using CSharpScript.ActionRunner;

var request = new ChangeStatusRequest()
{
  RecordId = RecordId,
  BusinessDocumentId = BusinessDocumentId,
  TransitionInternalId = "c70a1fd2-32b2-7d0b-c470-d42eba506425",
  Model = JsonConvert.DeserializeObject<Dictionary<string, object>>(Model)
};

var result =  await  CSharpScript.ActionRunner.ChangeStatusAction.Instance.RunActionAsync(request);

if(!result.IsValid)
{
  throw new TranslatedException("Zmiana statusu nie powiodła się", result.ValidationResult);
}
```

2. Pliki appsettings.json

Do plików konfiguracyjnych appsettings.json serwisów taskservice i publicapi dodano adresy URL serwisów reportsapi i syncfusion. Ta zmiana umożliwi korzystanie z nowych możliwości akcji Algorytm C# w zadaniach cyklicznych oraz żądaniach do PublicAPI.

FUNKCJONALNOŚCI EKSPERYMENTALNE

1. Akcja Uruchom BPMN

W akcji Uruchom BPMN dodano oczekiwanie na odpowiedź z silnika BPMN w przypadku, gdy akcja była wywoływana w inny sposób niż z przeglądarki (np. wyzwalana przejściem workflow). Zmiana dotyczy tylko akcji wykonywanej w trybie synchronicznym.

2. Migracja procesów BPMN

W migratorze aplikacji dodano mechanizm automatycznego publikowania w środowisku docelowym migrowanych procesów BPMN, który były opublikowane w środowisku źródłowym.

POPRAWKI I USUNIĘTE BŁĘDY

1. Raporty Telerik

Usunięto błąd w działaniu akcji raportów, który powodował, że plik PDF z wygenerowanym raportem dodawany do sekcji załączników był nieprawidłowy.

2. Raporty Telerik

Usunięto problem z uruchamianiem modułu designera raportów w środowisku dockerowym.

3. Migrator aplikacji

Poprawiono migrację triggerów bazodanowych dla schematów innych niż domyślny (dbo).

INFORMACJE UZUPEŁNIAJĄCE

1. Aktualizacja Telerik Reporting

Zaktualizowano moduł Telerik Reporting do wersji 18.3.24.1112.