# Programowanie komponentowe

Komponent Powiadomień

**Prowadzący: dr inż.** Lucjan Pelc
**Wykonanie**: Kinga Jaworska
**Kierunek**: Informatyka, rok III
**Nr albumu**: s36560

# Spis treści

# Specyfikacja aplikacji

## Tematyka projektu:

Tematem projektu jest aplikacja internetowa, będąca listą zadań. Daje ona użytkownikowi możliwość zarządzania zadaniami. Głównym komponentem jest element, generujący powiadomienia o zbliżających się zadaniach w formie wiadomości przesyłanej na adres email.

## Wykorzystywane technologie:

Platforma- *ASP .Net*

Biblioteka- *Hangfire, Postal V4*

## Zakres aplikacji:

- Zarządzanie zadaniami (podstawowe operacje CRUD)
- Zarządzanie kontem użytkownika (autoryzacja, logowanie, rejestracja)
- Zarządzanie powiadomieniami- do danego zadania użytkownik przypisuje konkretną datę, wedle której generowane jest powiadomienie.

# Struktura komponentu powiadomień:

### Dane dostarczane do komponentu:

- Email użytkownika
- Opcja powiadomień
- Zawartość powiadomienia

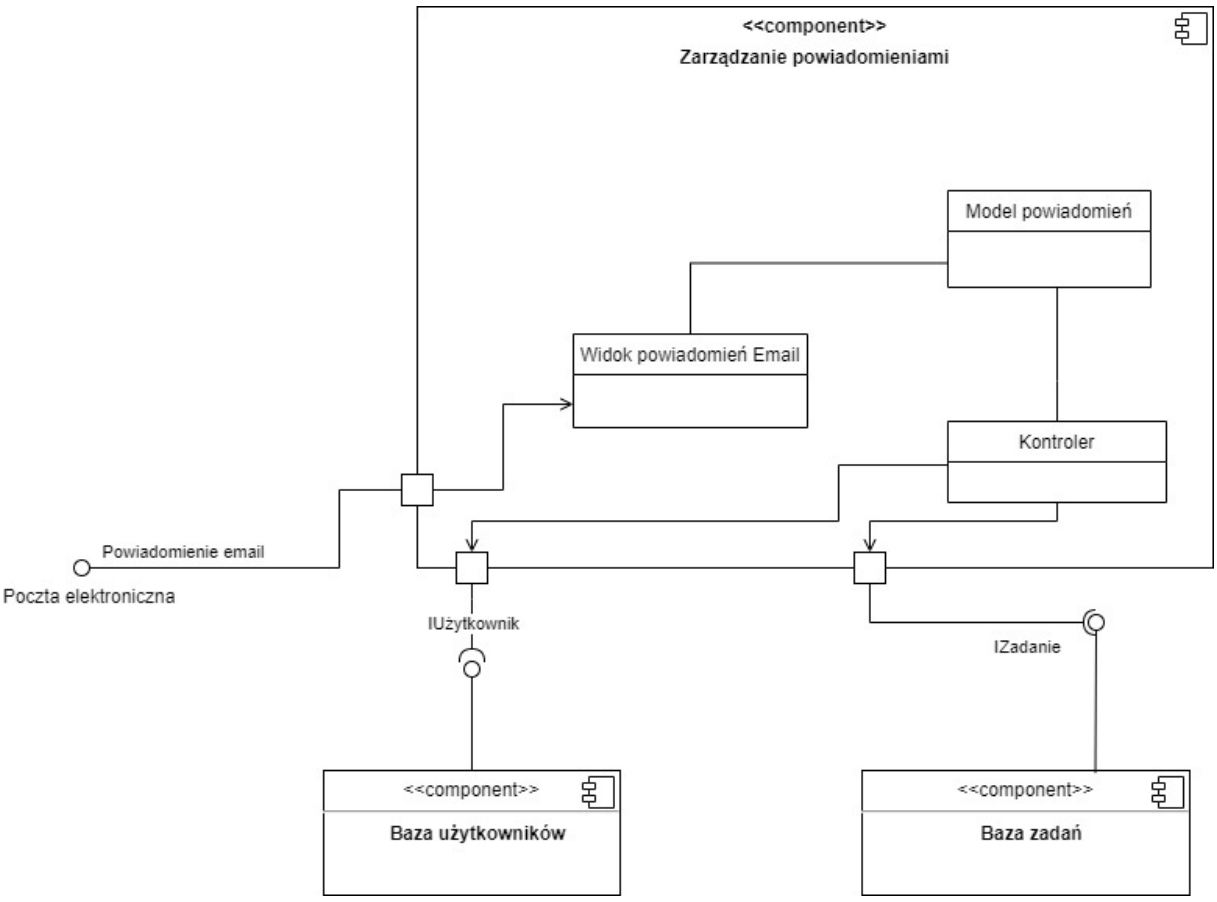
### Komponent generuje:

- Powiadomienia Email

### Opcje powiadomień (konfigurowane komponentu przez użytkownika):

- Informowanie o zadaniach na bieżący dzień
- Informowanie o przyszłych zadaniach
- Informowanie o zapomnianych zadaniach (domyślnie)

## Diagram komponentów



- **Model powiadomień**- pozwala na pobranie danych z bazy
- **Kontroler**- usługa obsługująca komponent (wszystkie dostępne metody, łączenie z bazą, tworzenie powiadomień na podstawie dostarczonych interfejsów oraz sposób ich wyświetlania)
- **Widok powiadomień Email** - graficzny sposób prezentacji powiadomień

# Wnętrze komponentu

## Metody zawarte w NotiController

1. *TurnEmail* – przekazuje potrzebne interfejsy z zewnątrz i uruchamia metodę MailInBackground.

```
1 odwołanie
public void turnEmail(string con, string columnName, string query, string useremail, string option, int days)
{
    try
    {
        MailInBackground(con, columnName, query, useremail,option, days);
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
1 odwołanie
```

2. *MailInBackground* – włącza usługę uruchamiającą dane zadanie (metodę) co określony czas oraz ustawia dla niej id jako email użytkownika. Dzięki temu można ją modyfikować i jest ona wykonywana oddzielnie dla każdego użytkownika, posiadającego konto.

```
1 odwołanie
public void MailInBackground(string con, string columnName, string query, string useremail, string option, int days)
{
    RecurringJob.AddOrUpdate(useremail, () => Mailing(con,columnName, query, useremail, option, days), Cron.Daily); // w ramach testów mozliwosc us
}
1 odwołanie
```

3. *Mailing* – metoda uruchamiana w *MailInBackground*. Tworzy i wysyła wiadomość, jeśli użytkownik posiada nieukończone zadania z przeszłości. Filtruje powiadomienia wedle wybranej opcji.

```
1 odwołanie
public void Mailing(string con, string columnName, string query, string useremail, string option, int days)
{
    List<String> NotiList2 = new List<String>();

    //spr czy notiBase mozna scalic z Noti

    switch(option)
    {
        case "1":
            NotiList2 = GetNotiToEmail(con, columnName, query,option, days);

            if (NotiList2.Count > 0)
            {
                //NotiBase notif = new NotiBase();
                Noti notif = new Noti();
                notif.Info = "Misje, o ktorych zapomniales :(";
                notif.NotiName = "Zadanie";
                notif.UserEmail = useremail;  //getCurrentUser przesylane do komponentu

                var email = new Noti()
                {
                    Info = notif.Info,
                    UserEmail = notif.UserEmail,
                    NotiName = notif.NotiName,
                    TaskList = NotiList2   //pobierane z bazy nazwy zadań do powiadomienia
                };

                email.Send();
            }
            break;

        case "2":
```

4. *GetNotiToEmail* – pobiera z bazy o podanym połączeniu wszystkie zaległe zadania na podstawie kwerendy wysłanej do komponentu przez użytkownika.

```
1 odwołanie
public List<String> GetNotiToEmail(string con, string columnName, string query, string option, int days) //zadania, ktore sa zalegle- EMAIL
{
    string currentTime = DateTime.Now.ToString("yyyy-MM-dd");
    try
    {
        using (var connection = new SqlConnection(con))
        {
            connection.Open();

            // var currentTime = DateTime.Now.AddDays(1).ToString("yyyy-MM-dd");
            //@"SELECT [NotiName], [TimeTask], [UserEmail] from [dbo].[Table_Task] where [TimeTask]<'" + currentTime + "' AND [UserEmail]='" + use

            using (command = new SqlCommand(query  + "< '"+ currentTime + "'" /*+ " <'" + currentTime + "'"*/, connection)) //zaległe zadania
            {
                var reader = command.ExecuteReader();

                while (reader.Read())
                {
                    NotiList.Add(reader[columnName] != DBNull.Value ? (string)reader[columnName] : "");
                }
            }
        }

        return NotiList;
    }
    catch (Exception ex)
    {
        return null;
    }
}
```

5. *GetNotiFuture* – pobiera z bazy o podanym połączeniu wszystkie zadania, których termin przypada na maksymalnie 2 dni od bieżącego dnia.

```
1 odwołanie
public List<String> GetNotiFuture(string con, string columnName, string query, string option, int days) //zadania, ktore sa na przyszlosci (max 2
{
    var futureTime = DateTime.Now.AddDays(days).ToString("yyyy-MM-dd");                    //mozliwosc konfuguracji na ile dni przed wyznaczo
    string currentTime = DateTime.Now.AddDays(1).ToString("yyyy-MM-dd");

    try
    {
        using (var connection = new SqlConnection(con))
        {
            connection.Open();

            using (command = new SqlCommand(query + " BETWEEN '" + currentTime + "' AND '"+ futureTime +"'", connection))
            {
                var reader = command.ExecuteReader();

                while (reader.Read())
                {
                    NotiList.Add(reader[columnName] != DBNull.Value ? (string)reader[columnName] : "");
                }
            }
        }

        return NotiList;
    }
    catch (Exception ex)
    {
        return null;
    }
```

6. *GetNotiToday* – pobiera z bazy o podanym połączeniu wszystkie zadania z obecnego dnia na podstawie kwerendy wysłanej do komponentu przez użytkownika.

```
1 odwołanie
public List<String> GetNotiToday(string con, string columnName, string query, string option) //zadania, ktore sa na przyszlosc- EMAIL
{
    string currentTime = DateTime.Now.ToString("yyyy-MM-dd");
    try
    {
        using (var connection = new SqlConnection(con))
        {
            connection.Open();

            using (command = new SqlCommand(query + "='" + currentTime + "'", connection)) //zaległe zadania
            {
                var reader = command.ExecuteReader();

                while (reader.Read())
                {
                    NotiList.Add(reader[columnName] != DBNull.Value ? (string)reader[columnName] : "");
                }
            }
        }

        return NotiList;
    }

    catch (Exception ex)
    {
        return null;
    }
}
```

# Interfejs programistyczny – sposób przekazania komponentu

## Konfiguracja

Wpierw wymagana jest podstawowa konfiguracja, która umożliwia aplikacji działanie w tle, zarządzania częstotliwością powiadomień oraz ustalenia adresu, z którego będą wysyłane emaile do użytkowników aplikacji.

Instalacja wymaganych pakietów NuGet:

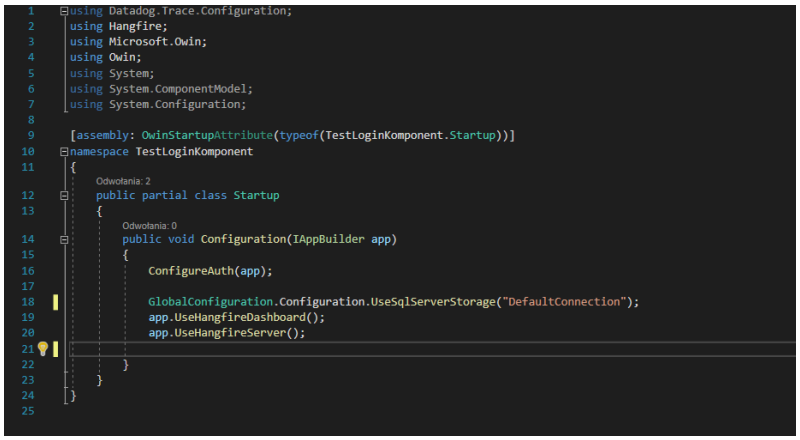
- Hangfire
- Postal v4 (dla wersji 4 .Net Framework)

Następnie ustawienie w pliku konfiguracyjnym web.conf serwera poczty:

```
</system.web>
<system.net>
  <mailSettings>
    <smtp from="[redacted]@gmail.com">
      <network host="smtp.gmail.com" port="587" userName="[redacted]gmail.com" password="[redacted]" enableSsl="true"
    </smtp>
  </mailSettings>
</system.net>
```

Uruchomienie pakietu *Hangfire* w klasie startowej OWIN:

W pierwszej linii pliku *Noti.cshtml* należy umieścić ścieżkę do klasy Noti.cs:



```
using Datadog.Trace.Configuration;
using Hangfire;
using Microsoft.Owin;
using Owin;
using System;
using System.ComponentModel;
using System.Configuration;

[assembly: OwinStartupAttribute(typeof(TestLoginKomponent.Startup))]
namespace TestLoginKomponent
{
    Odwołania: 2
    public partial class Startup
    {
        Odwołania: 0
        public void Configuration(IAppBuilder app)
        {
            ConfigureAuth(app);

            GlobalConfiguration.Configuration.UseSqlServerStorage("DefaultConnection");
            app.UseHangfireDashboard();
            app.UseHangfireServer();

        }
    }
}
```

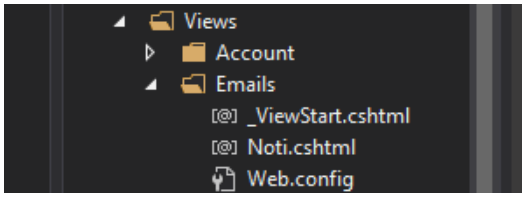
@model NazwaProjektu.NazwyFolderow.Noti

```
@model TestKomponent5.ViewModels.Users.Noti
To: @Model.UserEmail
Subject: @Model.Info

<p>Te zadania czekają na odhaczenie, włącz aplikację, by móc je ukończyć.</p>
@foreach (string item in @Model.TaskList)
{
    <p> @item</p>
    <br>
}
```

## Przekazany komponent

Użytkownik otrzymuje komponent w postaci kontrolera *NotiController.cs*, klasę *Noti.cs* (stanowiącej model wiadomości Email) oraz widok wiadomości Email *Noti.cshtml*, którą należy umieścić w folderze wygenerowanym przez pakiety NuGet (w ścieżce Views/Emails).




## Implementacja

Programista w dowolny sposób uzyskuje połączenie do bazy, a także email obecnie zalogowanego użytkownika. Następnie w miejscu, w którym chce uruchomić komponent tworzy instancję klasy NotificationController. Wywołuje na niej metodę *turnEmail* przekazując jako argumenty połącznie z bazą, email zalogowanego użytkownika, kwerendę uzyskującą zadania z bazy oraz nazwę kolumny pod jaką znajdują się zawartości powiadomień. Programista w dowolny sposób przekazuje również informację o sposobie generowania powiadamiania. Umożliwia to dostarczenie do komponentu numeru opcji. Może on być spersonalizowany dla każdego użytkownika, bądź wybrany domyślnie przez programistę.

## Wymagania

Kwerenda powinna zawierać:

- nazwę lub zawartość powiadomienia typu String
- email użytkownika
- czas, wedle którego będą generowane powiadomienia

## Opcje- własna konfiguracja

Powiadomienia mogą być dostarczane na trzy sposoby. Informować o:

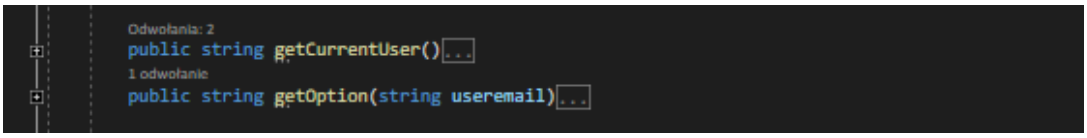
- zadaniach, których termin upłynął (opcja = 1)
- zadaniach, których termin wykonania przypada na maksymalnie 2 dzień od bieżącego dnia. (opcja = 2)
- zadaniach, których termin wykonania przypada na obecny dzień (opcja = 3)

Możliwe jest także nadanie opcji, określającej na ile dni przed mają być generowane powiadomienia, których termin upływa w przyszłości. W przypadku nieużywania tej opcji można przekazać do paametru days dowolną wartość.

## Użycie komponentu w przykładowej aplikacji

Komponent został zastosowany w aplikacji typu *„To Do”* . Do jej stworzenia zostały wykorzystane gotowe komponenty logowania i rejestracji przy pomocy pakietu *Microsoft.Asp.Net.Identity*.

W klasie HomeController w dowolny sposób został pozyskany obecnie zalogowany użytkownik oraz opcja powiadomień. W tym wypadku na pobranie emaila zalogowanego użytkownika pozwala dodatkowo użyty komponent logowania. Opcja zaś pobierana jest z bazy zawierającej użytkowników.




### Komponent w głównym kodzie aplikacji

W głównej klasie, zostaje stworzony obiekt, przez który wywołuje się komponent. Przekazano do niego połączenie z bazą, nazwę kolumny zwierającej treść powiadomień, kwerendę pobierającej zadania, email użytkownika, opcję, decydującą o sposobie powiadamiania oraz liczbę dni, przed którą mają być wysyłane powiadomienia.

```
string columnName= "NotiName";
useremail = getCurrentUser();
NotiController notiController2 = new NotiController();
```

```
    //Dostarczane do komponentu:
    option = getOption(useremail);
    string currentTime = DateTime.Now.ToString("yyyy-MM-dd");
    string query = @"SELECT [NotiName], [TimeTask], [UserEmail] from [dbo].[Table_Task] where [UserEmail]='" + useremail + "' AND [TimeTask]"; //pob

    //Wyzwalanie komponentu:
    int  days = 4;
    notiController2.turnEmail(con, columnName, query, useremail,option, days);
    return RedirectToAction("TaskList");
}
```

### Baza Danych

Struktura tabeli z zadaniami z zawartością powiadomień i terminem ich wykonania.

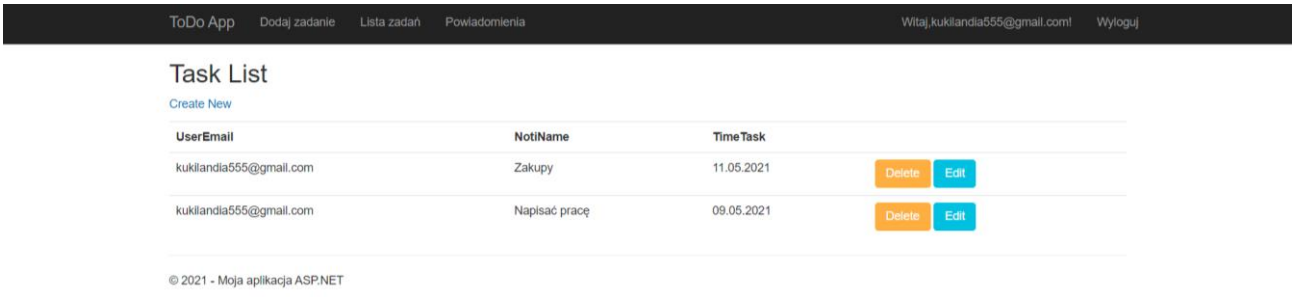
| Name | Data Type | Allow Nulls | Default |
|---|---|---|---|
| TaskID | int | ☐ | |
| UserEmail | nvarchar(250) | ☐ | |
| NotiName | nvarchar(250) | ☐ | |
| TimeTask | date | ☐ | |
| Opcja | nvarchar(250) | ☑ | |
| | | ☐ | |

Tabela wygenerowana przez gotowy komponent rejestracji, rozszerzona o kolumnę z opcją powiadomień.

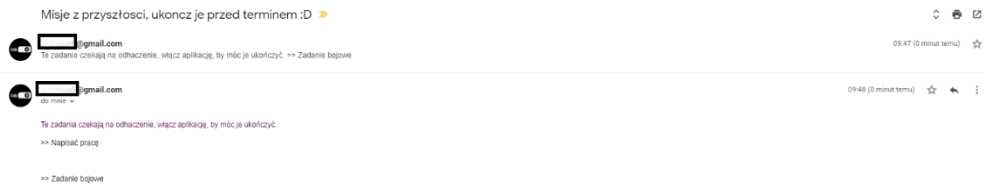
| | Name | Data Type | Allow Nulls | Default |
|---|---|---|---|---|
| 🔑 | Id | nvarchar(128) | ☐ | |
| | Email | nvarchar(256) | ☑ | |
| | EmailConfirmed | bit | ☐ | |
| | PasswordHash | nvarchar(MAX) | ☑ | |
| | SecurityStamp | nvarchar(MAX) | ☑ | |
| | PhoneNumber | nvarchar(MAX) | ☑ | |
| | PhoneNumberConfirmed | bit | ☐ | |
| | TwoFactorEnabled | bit | ☐ | |
| | LockoutEndDateUtc | datetime | ☑ | |
| | LockoutEnabled | bit | ☐ | |
| | AccessFailedCount | int | ☐ | |
| | UserName | nvarchar(256) | ☐ | |
| | Option | nvarchar(1) | ☐ | ((1)) |
| | | | ☐ | |

## Wynik działania

Lista zadań danego użytkownika:




Widok otrzymanego powiadomienia email:




Pod adresem https://localhost:44353/hangfire można uruchomić dashboard dla usługi hangfire. Panel pakietu wyświetla uruchomione procesy oraz częstoliwość ich wykonywania. Zadanie (w tym wypadku wysłanie emaila) jest obecnie uruchomione dla dwóch użytkowników i wykonywane raz dziennie.