Advanced Programming C#

Lecture 13

dr hab. inż. Małgorzata Janik

malgorzata.janik@pw.edu.pl


WYDZIAŁ FIZYKI
POLITECHNIKA WARSZAWSKA
POLITECHNIKA WARSZAWSKA
WARSAW UNIVERSITY OF TECHNOLOGY

Project

Project part III

- Final Date: 26.01.2020 (next weeks!)
- Presentations (max 5 min per project!):
  - 1 poster/slide that advertises your project
  - 1 slide listing implemented functionalities
  - presentation of the program
    - real-time presentation of the application
  - to be shown to the whole group
- Code will be evaluated
  - Project **must** be shared via git repository as well
  - If application does not meet basic requirements/functionality, the project is not graded

Programowanie asynchroniczne

Asynchroniczność w C# (async / await)

Jak wykonywać długie operacje bez blokowania interfejsu użytkownika?

(problem w aplikacjach okienkowych)

Aplikacje Windows Forms mają jeden główny wątek UI.

Ten wątek:

- rysuje okno
- obsługuje kliknięcia
- reaguje na użytkownika

Jeśli ten wątek zostanie zablokowany → aplikacja przestaje reagować.

Idea asynchroniczności

Zamiast:

- „zatrzymaj program i czekaj”

chcemy:

- „zacznij operację i wróć do UI”

Asynchroniczność:

- oddaje sterowanie systemowi
- pozwala UI działać normalnie
- wraca do kodu, gdy operacja się zakończy

Słowa kluczowe: async i await

async:

→ oznacza metodę asynchroniczną

await:

→ czeka na zakończenie operacji, nie blokuje wątku UI

Przykład async / await

```
private async void button1_Click(object sender, EventArgs e)
{
    await Task.Delay(3000);
    label1.Text = "Gotowe";
}
```

Efekt:
- tekst pojawia się po 3 sekundach
- okno cały czas reaguje

await:
- zatrzymuje wykonanie metody
- zapisuje jej stan
- oddaje sterowanie UI
- wraca do metody po zakończeniu operacji

(To nie jest Sleep!)

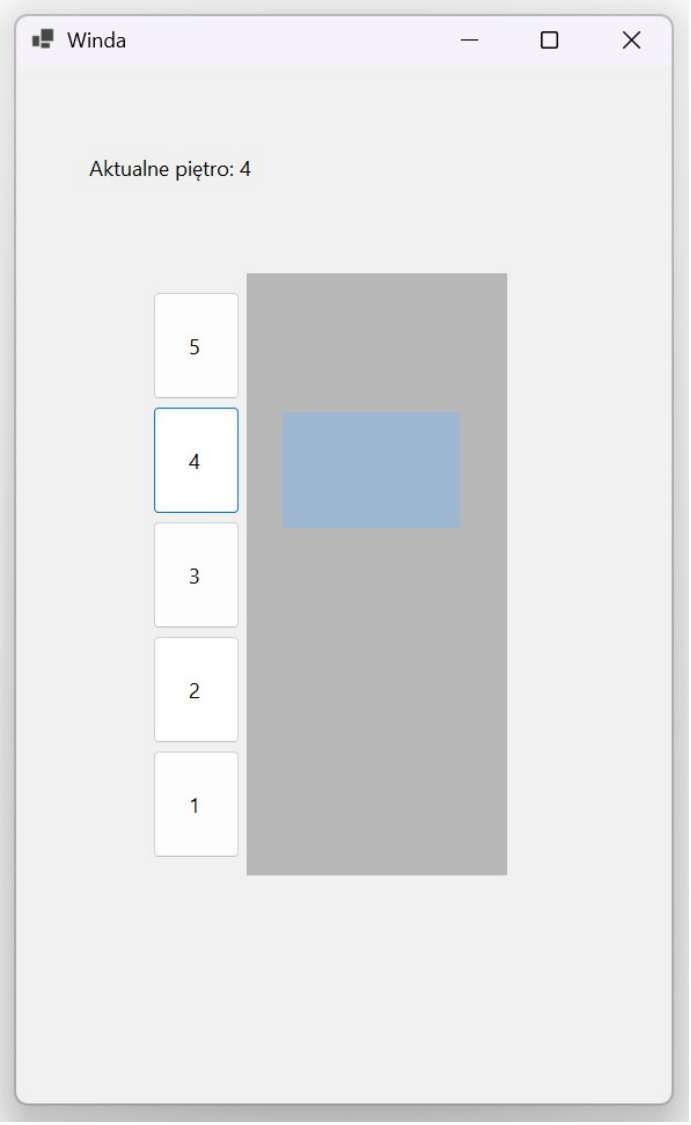
Co można „awaitować”?
Można użyć await dla metod zwracających:
- Task
- Task<T>

Przykłady:
- Task.Delay(...)
- ReadAllTextAsync(...)
- GetStringAsync(...)
- własne metody async

Zadanie: Winda

Aplikacja **Windows Forms** przedstawia jedną windę obsługującą 5 pięter.

Użytkownik wybiera piętra za pomocą przycisków, a winda porusza się między nimi w czasie, piętro po piętrze.



Kontrolki

- Panel panelShaft – szyb windy
- Panel panelElevator – sama winda (prostokąt)
- Label labelFloor – aktualne piętro
- 5 × Button – piętra 1–5

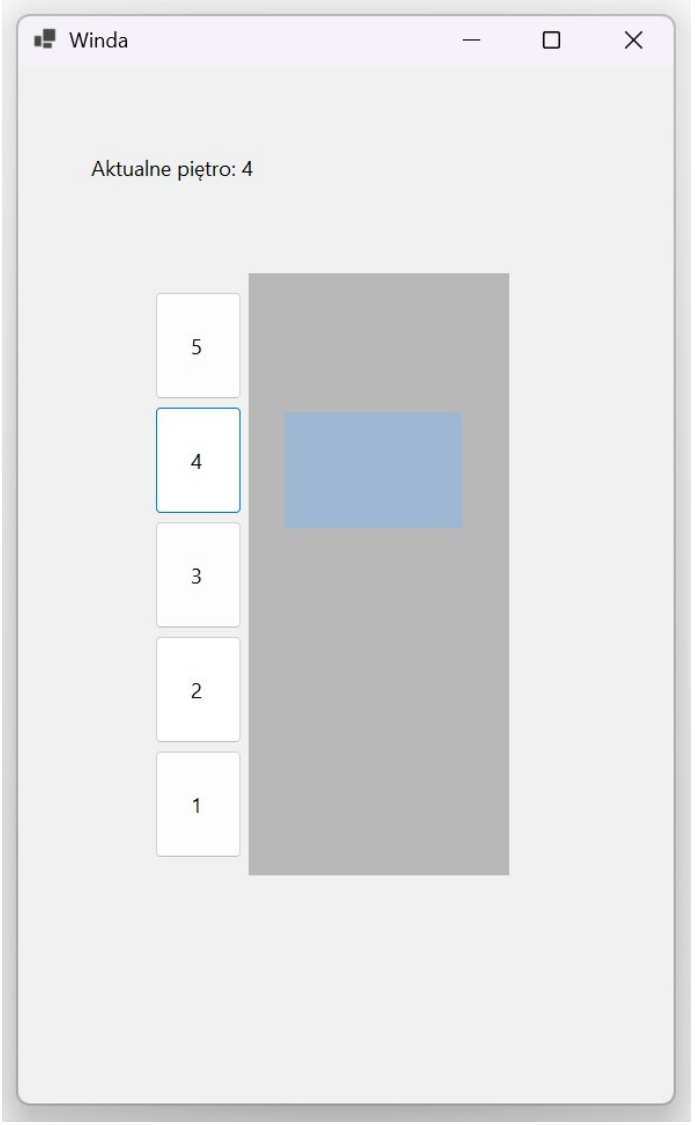
Układ:

- panelShaft wysoki, np. 300 px
- panelElevator wewnątrz, wysokość np. 40 px

Zadanie: Winda

Aplikacja przedstawia jedną windę obsługującą 5 pięter.

Użytkownik wybiera piętra za pomocą przycisków, a winda porusza się między nimi w czasie, piętro po piętrze.



Pola Form1:

- private int currentFloor = 1; //aktualne piętro
- private int targetFloor = 1; //piętro do którego jedzie
- private const int floorsCount = 5;
- private const int floorHeight = 60; // px na piętro

Ustawienie windy na odpowiednim miejscu:

```
private void UpdateUI()
{
    //Odśwież label „Aktualne piętro”

    // Liczymy pozycję windy
    int bottomOffset = ….. ;

    //Ustawiamy pozycję windy
    panelElevator.Top = panelShaft.Height -
                panelElevator.Height - bottomOffset;
}
```

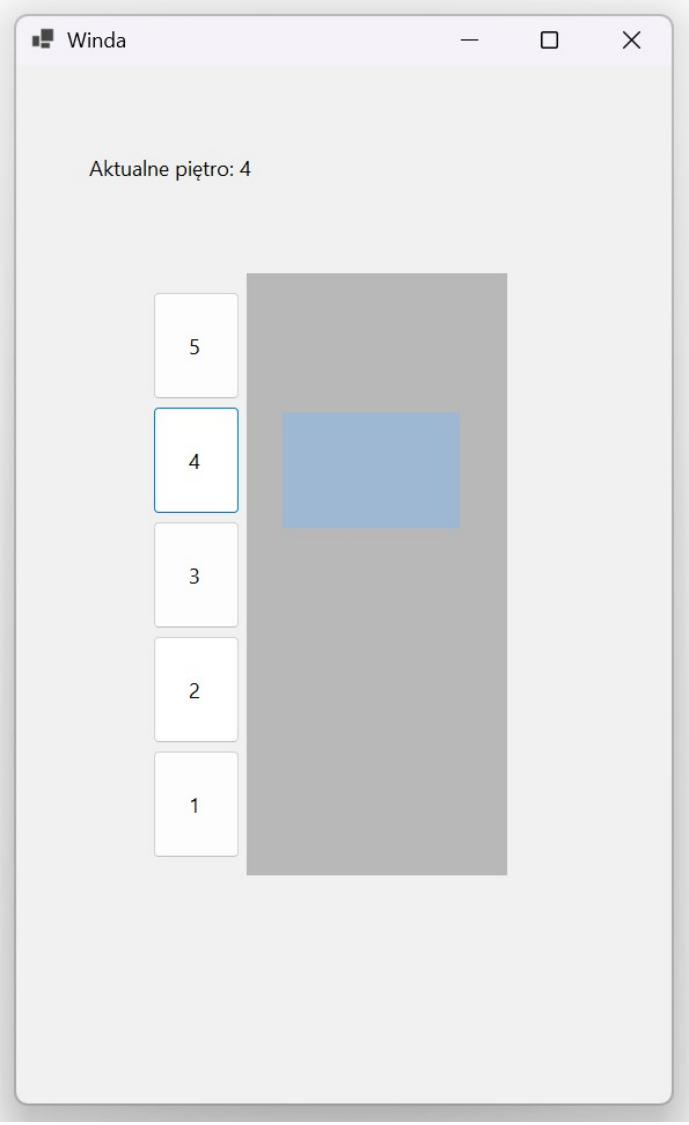
Zadanie: Winda

Obsługa przycisków:

Metoda
FloorButton_Click(object sender, EventArgs e)
wspólna dla wszystkich przycisków.

- Jak wyciągnąć który przycisk był naciśnięty?
  Button button = sender as Button;

Powinna być/mieć:
- async
- Pobierać numer piętra z nazwy przycisku.
- Wywoływać metodę MoveElevatorAsync(selectedFloor) w trybie await.



Zadanie: Winda

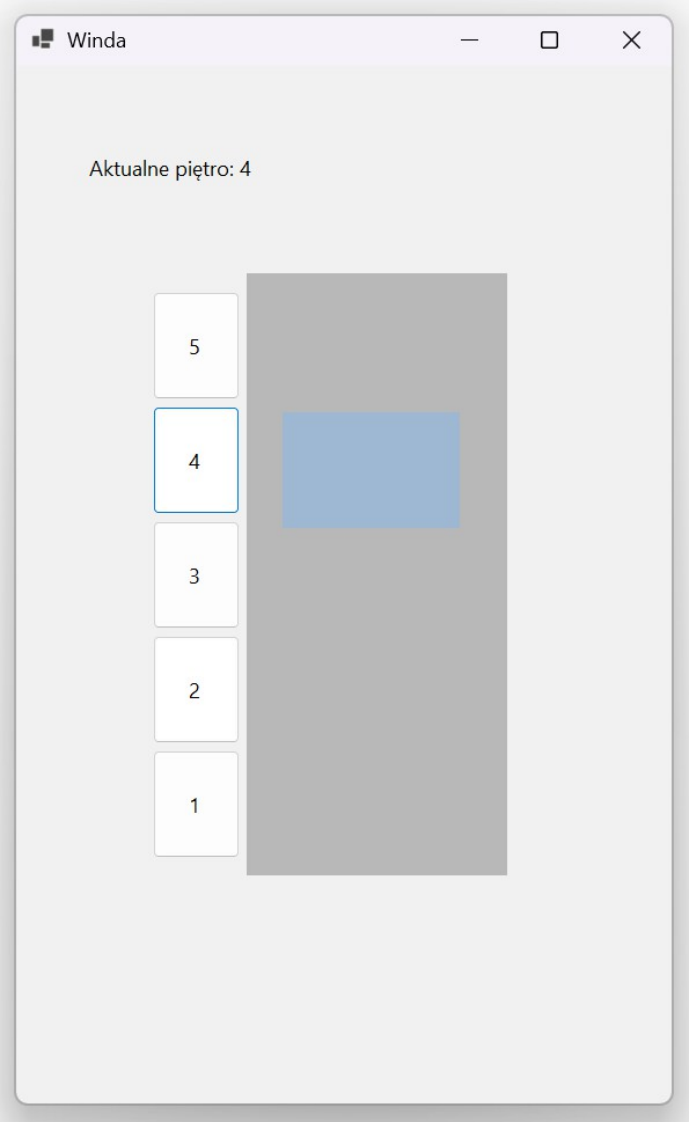
Ruch windy (kluczowy fragment):

```
private async Task MoveElevatorAsync(int target)
{
    ...
}
```

Po kolei:
- Ustaw targetFloor.

Dopóki currentFloor != targetFloor:
- odczekaj 500 ms (Task.Delay)
- zwiększ lub zmniejsz currentFloor o 1
- UpdateUI()

To symuluje:
- ruch windy w czasie,
- przejazd piętro po piętrze.



Zadanie: Winda

Dodanie **zdarzenia zmiany piętra**
(pozwala rozdzielić logikę i UI) zamiast obecnego wywoływania metody UpdateUI:

```
public event Action FloorChanged;
```

Metoda obsługująca zdarzenie FloorChanged to po prostu UpdateUI (void, nie przyjmuje argumentów), najlepiej zmienić jej nazwę na:

private void OnFloorChanged()

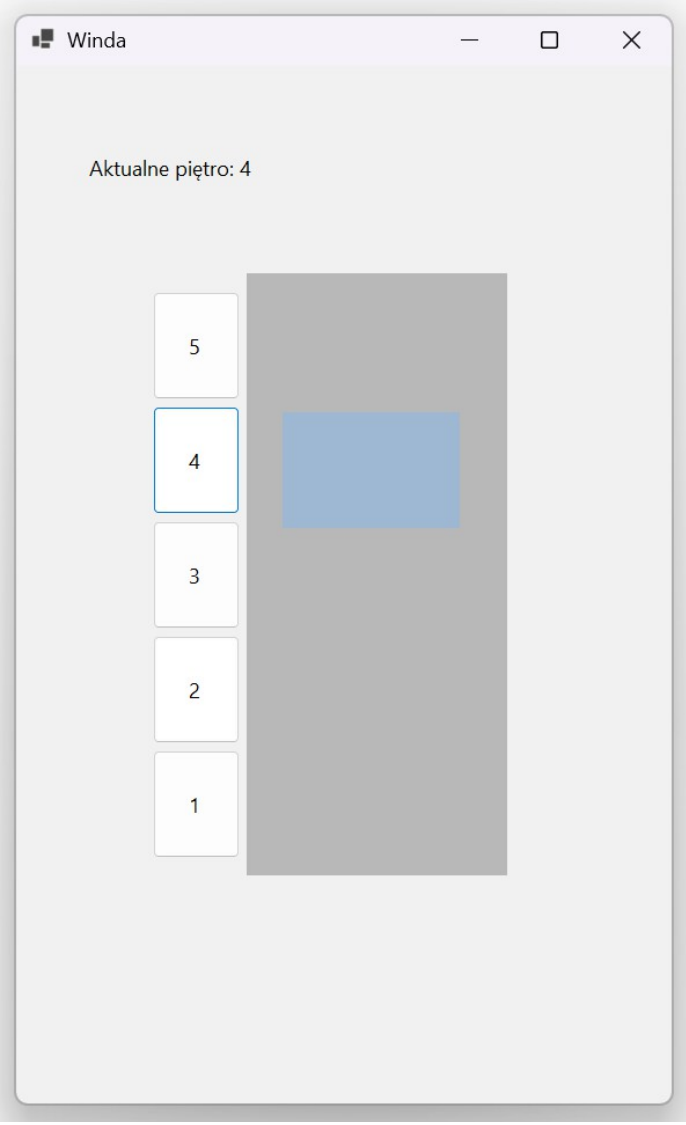
To jest jedyna metoda, która modyfikuje UI!

Subskrypcja w konstuktorze Form1:

```
FloorChanged += OnFloorChanged;
```

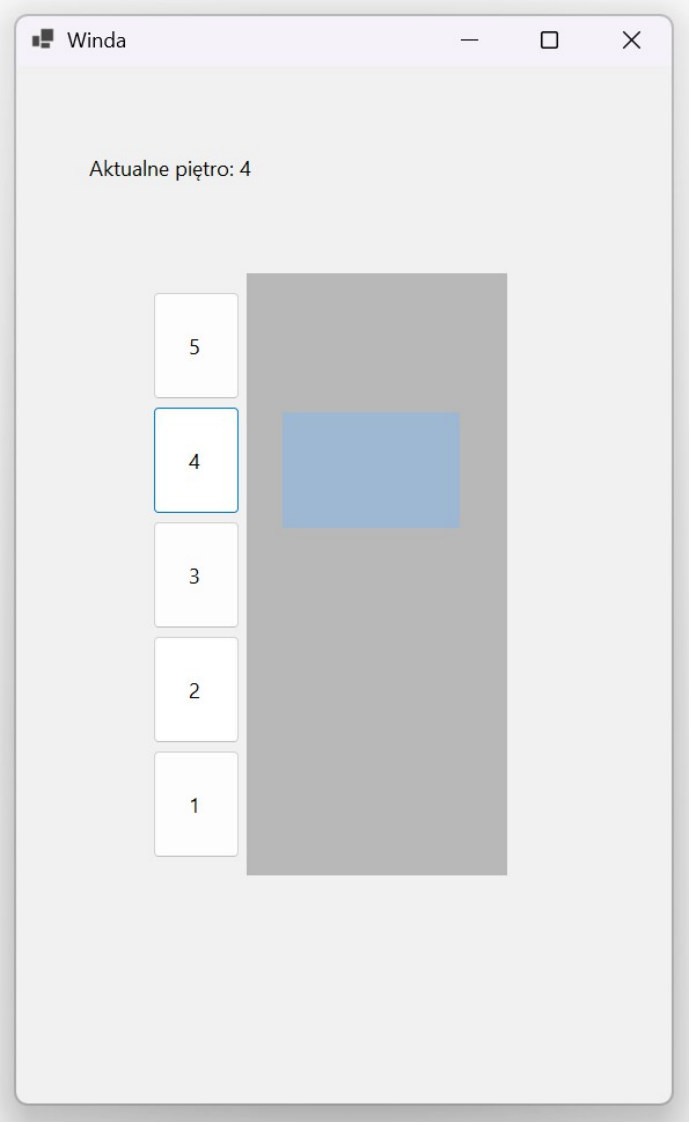
Wywołanie eventu:

```
FloorChanged?.Invoke();
```



Zadanie: Winda

Możliwe rozszerzenia:
+ kolorowanie przycisków
+ kolejka pięter



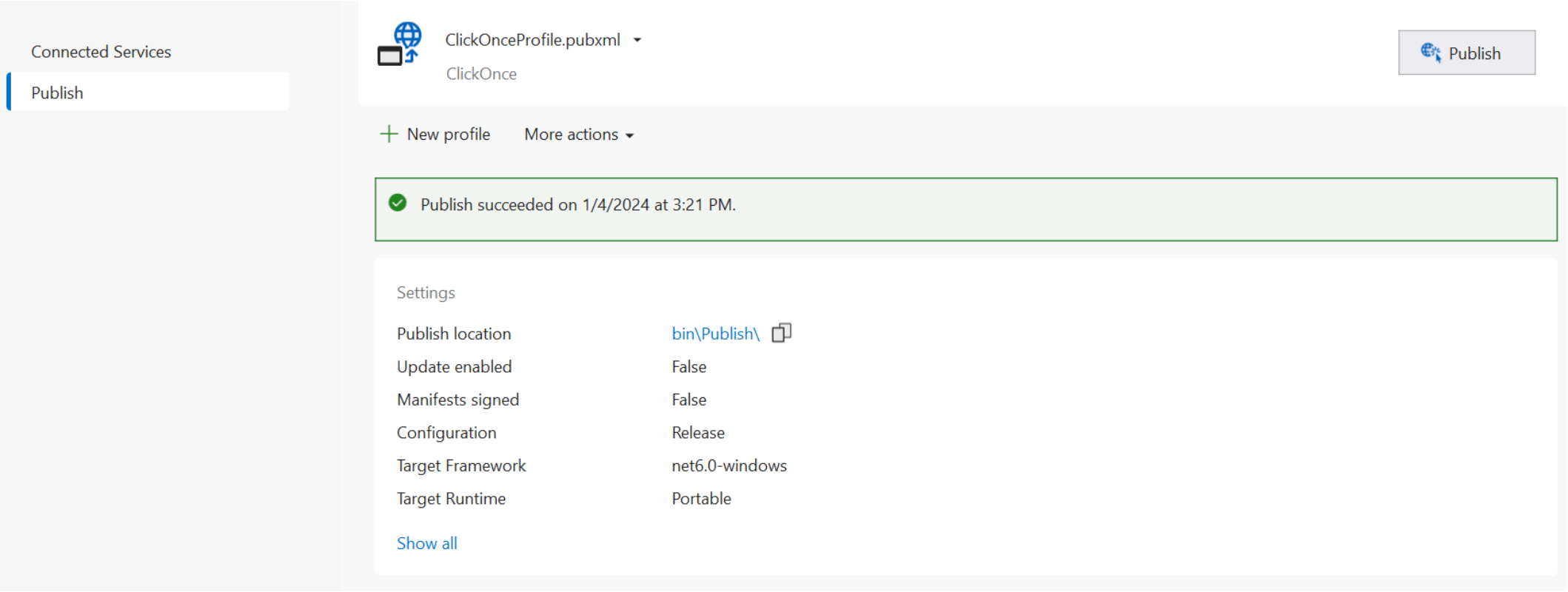
THE END

Dr hab inż. Małgorzata Janik
malgorzata.janik@pw.edu.pl

Creating EXE

Creating EXE

- Click on the Project → Publish... → Folder → ClickOnce... → CD, DVD or USB... → Next → Next → Finish
- Publish



Creating EXE

- With Installer extension:

  https://www.youtube.com/watch?v=NOkBUoP54b8

- Simplest:

  https://www.youtube.com/watch?v=rMr3ejOEiDY

  - Click on the Project → Publish... → Folder → Folder... → choose place to publish → Finish