

# Platformy Programistyczne .NET i Java

## Laboratorium 4

### *Aplikacja webowa w technologii ASP.NET Core*

prowadzący: *Dr inż. Radosław Idzikowski, mgr inż. Michał Jaroszczuk*

---

## 1 Cel laboratorium

Celem laboratorium jest zapoznanie się z podstawami projektowania i implementacji aplikacji webowych w technologii **ASP.NET Core**. W ramach zajęć należy w pierwszej kolejności zapoznać się z technologią **Blazor** w ramach platformy **ASP.NET Core**, następnie stworzyć aplikację bazodanową oraz w ostatnim korku opublikować ją w ramach chmurze **Azure**.

Praca będzie oceniana na bieżąco podczas zajęć. **Ukończenie każdego etapu powinno być zgłoszone prowadzącemu w celu akceptacji i odnotowania postępów**. Sam program należy umieścić na repozytorium **github** i wysłać zaproszenie do prowadzącego. Czas na wykonanie zadania to dwa zajęcia laboratoryjne. Podczas pierwszego spotkania trzeba wykonać co najmniej jedno zadanie zdefiniowane w Rozdziale 3.

## 2 Zadania

W ramach zajęć należy w zespołach wykonać następujące zadania:

1. Poznanie platformy **Blazor**.
2. Bazodanowa aplikacja webowa.
3. Wdrożenie aplikacji w chmurze **Azure**.

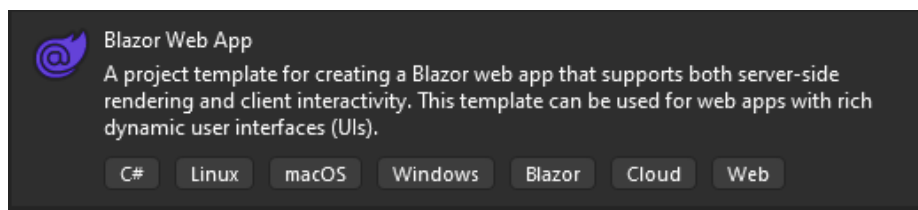
Za wykonanie zadania nr 1 jest ocena dostateczna, za każde kolejne zadanie jest +1 do oceny. Na ocenę bardzo dobrą (5.0) należy wykonać wszystkie trzy zadania. Link do repozytorium należy przesłać dopiero po oddaniu i ocenieniu pracy na laboratorium.

## 3 Opis Zadań

W tej sekcji zostaną kolejno omówione wszystkie zadania do wykonania podczas laboratorium. Komputery w laboratorium zostały przygotowane do przeprowadzenia zajęć.

### 3.1 Zadanie 1

W ostatnich latach widać trend polegający na wypieraniu aplikacji desktopowych przez aplikacje webowe. Wymaga to innego niż w przypadku aplikacji desktopowych podejścia stąd powstanie framework'ów takich jak **MVC** (implementujący wzorzec Model - View - Controller) czy **Razor** (pozwalający na osadzanie kodu platformy **.NET** na stronach internetowych). Technologie te stały się jednak niewystarczające ze względu na konieczność kontroli procesu zarówno po stronie klienta jak



Rysunek 1: Tworzenie nowego projektu aplikacji webowej Blazor

i serwera. Stąd też niedawne wypuszczenie na rynek technologii **Blazor** przez firmę Microsoft, która obsługuje renderowanie zarówno po stronie klienta jak i serwera w jednym modelu programowania.

W ramach zadania należy utworzyć projekt **Blazor Web App** wedle szablonu. Następnie trzeba zapoznać się z budową tego typu aplikacji na utworzonym przykładzie. Poniżej opisano krok po kroku kolejne elementy. Na koniec będzie trzeba wprowadzić dwie drobne modyfikacje: (1) prosty licznik oraz (2) filtr danych.

### Aplikacja Blazor

W pierwszej kolejności należy utworzyć nowy projekt **Blazor Web App** (aplikacji webowej **Blazor** jak na Rys. 1). W kolejnym kroku należy wybrać zgodność z **.NET 8.0 (Long Term Support)** oraz:

- `Authentication type: None,`
- `Configure for HTTPS : True,`
- `Interactive render mode: Server,`
- `Interactivity location: Per page/component,`
- `Include sample pages : True,`
- `Do not use top-level statements : False,`

Nowo utworzony projekt zawiera niezbędne pliki do uruchomienia prostej aplikacji **Blazor**. Plik `Program.cs` jest punktem wejścia dla aplikacji, który uruchamia serwer oraz w którym skonfigurowane są usługi i oprogramowanie pośredniczące aplikacji. W folderze `Components` znajdują się elementarne pliki aplikacji jak:

- `App.razor` – główny komponent dla aplikacji,
- `Routes.razor` – odpowiedzialny za konfigurację routera **Blazor**,
- pozostałe pliki odpowiedzialne za `Layout` (wygląd) strony.

Ponadto katalog `Pages` zawiera kilka już przygotowanych, przykładowych stron internetowych dla aplikacji. Plik `launchSettings.json` wewnątrz katalogu `Properties` definiuje różne ustawienia profilu dla środowiska lokalnego. Numer portu jest automatycznie przypisywany podczas tworzenia projektu i zapisywany w tym pliku.

Niestety tak przygotowana aplikacja nie uruchomi się w każdej przeglądarce, co skutkuje kodem błędu `NET::ERR_CERT_INVALID`. Rozwiązaniem jest skonfigurowanie uwierzytelniania za pomocą certyfikatu. W tym celu należy dodać pakiet `Microsoft.AspNetCore.Authentication.Certificate` z wykorzystaniem menadżera pakietów `NuGet`. Następnie możemy dodać prosty domyślny certyfikat, wystarczający na potrzeby zadania (ale niezalecany w komercyjnych aplikacjach) W pierwszej kolejności trzeba skonfigurować odpowiedni serwis odpowiedzialny uwierzytelniane, koniecznie jeszcze przed zbudowaniem aplikacji `var app = builder.Build();`.

```

1 builder.Services.AddAuthentication(
2     CertificateAuthenticationDefaults.AuthenticationScheme)
3     .AddCertificate();

```

Następnie należy wywołać metody odpowiedzialne za uwierzytelnianie i autoryzacji. Koniecznie przed uruchomieniem już zbudowanej aplikacji `app.Run()`;

```

1 app.UseAuthentication();
2 app.UseAuthorization();

```

Tak przygotowaną Aplikację uruchamiamy standardowo zieloną strzałką *Start Debugging* przy użyciu protokołu `https`. Aplikacja może być oczywiście uruchomiona również z wykorzystaniem protokołu `http`. Zmiany w kodzie możemy aplikować bez wyłączania programu stosując przycisk *Hot Reload* (symbol czerwonego płomienia). W przypadku utworzenia projektu z włączoną opcją `Include sample pages`, otrzymamy trzy przykładowe strony, każda o rozszerzeniu `.razor`. Najprostszą konstrukcją ma strona startowa `Home.razor`.

```

1 @page "/"
2 <PageTitle>Home</PageTitle>
3 <h1>Hello, world!</h1>
4 Welcome to your new app.

```

Aplikacje Blazor opierają się na komponentach. Komponent w Blazor to element interfejsu użytkownika, taki jak strona, okno dialogowe czy formularz wprowadzania danych. Klasa komponentu jest zwykle pisana w formie strony znaczników Razor z rozszerzeniem pliku `.razor`. Razor to składnia do łączenia znaczników HTML z kodem C#. Blazor wykorzystuje naturalne znaczniki HTML do komponowania interfejsu użytkownika. Lepiej to jest widocznie na przykładzie z licznikiem kliknięć przy użyciu przycisku w komponencie `Counter.razor`.

```

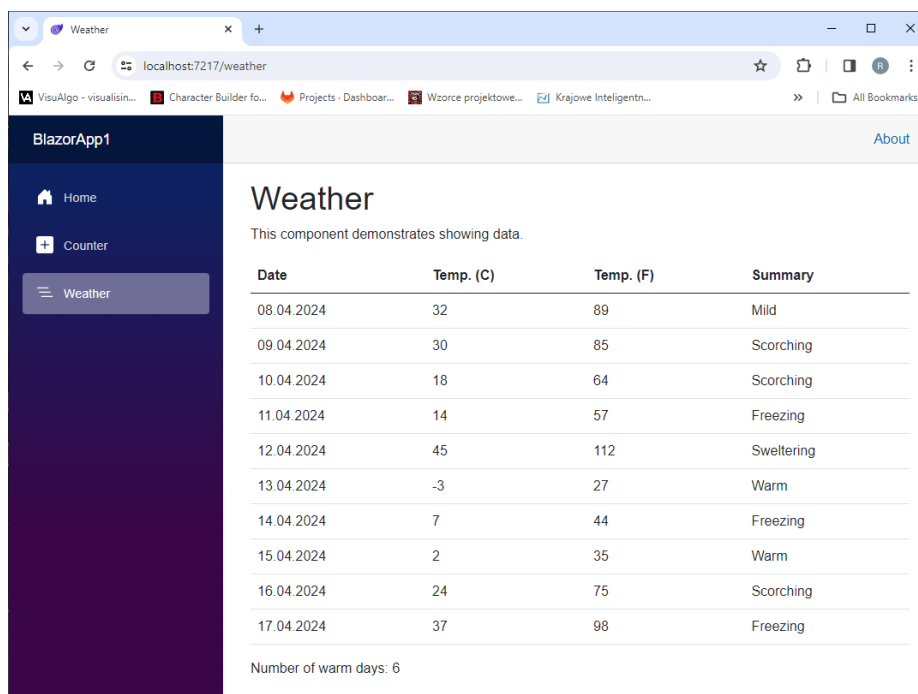
1 @page "/counter"
2 @rendermode InteractiveServer
3 <PageTitle>Counter</PageTitle>
4 <h1>Counter</h1>
5 <p role="status">Current count: @currentCount</p>
6 <button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
7 @code {
8     private int currentCount = 0;
9
10    private void IncrementCount()
11    {
12        currentCount++;
13    }
14 }

```

Dyrektywa `@rendermode` umożliwia interaktywne renderowanie komponentu na serwerze, dzięki czemu może obsługiwać zdarzenia interfejsu użytkownika przesyłane z przeglądarki. Przy każdym wejściu na stronę `counter` licznik będzie zerowany. Dużo ciekawszym i bardziej rozbudowanym przykładem jest strona `weather`, gdzie jest prognozowana (w tym wypadku losowo) pogoda na pięć najbliższych dni, a następnie wyświetlana w formie tabeli.

W ramach zadania należy w pierwszej kolejności wydłużyć termin prognozy do 10 dni. Następnie trzeba dodać w sekcji `@code` zmienną prywatną `warmDays` odpowiedzialną za liczbę ciepłych dni (powyżej  $15^{\circ}\text{C}$ ). W funkcji `OnInitializedAsync()` należy dopisać fragment kodu odpowiedzialny za ich zliczenie (bezpośrednio po wygenerowaniu). Na koniec jeszcze należy w części z kodem HTML dodać akapit z wyświetloną sumą (analogicznie jak status licznika) jak na rysunku 2. Dodanie nowych lub zmiana nazwy zmiennych czy funkcji wymusi reset aplikacji.

Kolejnym krokiem jest dodanie przycisku, który pozwoli odfiltrować dni z temperaturą poniżej  $15^{\circ}\text{C}$ . W pierwszej kolejności należy dodać funkcję (może być typu `void`), która przy użyciu składni LINQ pozwoli odfiltrować tablicę `forecasts`. Na koniec trzeba pamiętać, aby wymusić utworzenie tablicy metodą `ToArray()`. Teraz już wystarczy dodać przycisk na dole strony oraz na początku



Rysunek 2: Strona z prognozą pogody uzupełniona o licznik ciepłych dni

dyrektywę komponentu `@rendermode InteractiveServer`. Przycisk dodajemy używając znaczników HTML `<button>Text</button>`, do zdefiniowania jego wyglądu możemy użyć, np.: klasy `btn btn-primary`. Zdefiniowaną wcześniej metodą należy podpiąć pod wydarzenie odpowiedzialne za kliknięcie `@onclick="WarmDaysFilter"` lub w przypadku funkcji z parametrem za pomocą delegata `@onclick="() => WarmDaysFilter(15)"`. Następnie należy dodać również przycisk `Restore`, który pozwoli przywrócić tabelę do pierwotnego stanu.

Ostatnim etapem jest filtrowanie po nazwie. W tym celu można zdefiniować funkcję `private void Input(ChangeEventArgs arg)`, gdzie parametr `arg` pozwoli na przechwycenie wprowadzonego tekstu, ale trzeba jego pole `Value` zrzutować na `string` lub wywołać metodę `ToString()`. Do samego filtrowania tekstu warto skorzystać z LINQ oraz metody `Contains`. Na koniec należy dodać jeszcze komponent odpowiedzialny za wprowadzanie danych oraz podpiąć napisaną funkcję `<input class="form-control" @oninput="@Input" />`., alternatywnie możemy skorzystać z wydarzenia `@onchange` jeśli chcemy potwierdzić wprowadzenie frazy enterem. Powyższe funkcje zostały opisane jako typu `void`, dla funkcji, które zwracają wynik natychmiastowo takie rozwiązanie jest dopuszczalne. Jednak według zasad poprawnego projektowania funkcje powinny być asynchroniczne, co wymusza typ `Task`. Przykład działającej aplikacji zamieszczono pod [linkiem](#) w portalu [Azure](#).

## 3.2 Zadanie 2

W ramach zadania należy wykonać prostą aplikację bazodanową w technologii `Blazor` z wykorzystaniem `EntityFramework`. Wykonana aplikacja musi zawierać system logowania oraz powinna pozwalać na wprowadzanie ocen dla filmów, gier czy książek.

Do autoryzacji użytkowników można skorzystać z gotowego rozwiązania jakie oferuje nam projekt `Blazor Web App`, wystarczy podczas tworzenia nowego projektu wybrać opcję `Individual Accounts` w polu `Authentication type`. W efekcie otrzymamy prawie gotowy system do autoryzacji użytkowników (potwierdzenie rejestracji poprzez kliknięcie w link po jej wykonaniu). Przed uruchomieniem

mieniem aplikacji należy jeszcze stworzyć migrację oraz wykonać aktualizację bazy danych z poziomu `Package Manager Console`. Sposób obsługi bazy danych (jej definicji, tworzenia czy migracji) nie różni się od poznanego już sposobu przy okazji laboratorium nr 2. Docelowo dostęp do części stron powinien zostać zablokowany z użyciem atrybutu `Authorize` (sposób użycia pokazano w komponencie `Auth.razor`). Ponadto część podstron w menu nawigacyjnym również powinna być niewidoczna dla niezalogowanych użytkowników (komponent `NavMenu.razor`, sekcja `<AuthorizeView>`). Alternatywnie dla chętnych, może spróbować zaimplementować autoryzację z wykorzystaniem konta `Google` lub `Microsoft`, co może być przydatne w przyszłych studenckich bądź komercyjnych projektach.

W przypadku części bazodanowej w pierwszej kolejności należy utworzyć nową klasę pod elementy, które będziemy przechowywać. Można to zrobić bezpośrednio w folderze `Components` lub stworzyć wewnątrz niego dedykowany folder. Poniżej przedstawiono przykładową klasę przechowującą informację o filmach. Do poprawnego działania `EntityFramework` składowe klasy powinny być w formie własności. Ponadto składowe powinny być o różnych typach. Składowa `Id` jest obowiązkowa, pozostałe mogą przyjąć wartość `null`. Przy zmiennej `DateTime?` warto dodać adnotację, która wymusi format daty, widoczne będzie to w późniejszym etapie.

```
1 public class Movie
2 {
3     public int Id { get; set; }
4     public string? Title { get; set; }
5     public string? Description { get; set; }
6     [DataType(DataType.Date)]
7     public DateTime? ReleaseDate { get; set; }
8     public float? Rate { get; set; }
9 }
```

Następnie należy wygenerować odpowiednie podstrony do obsługi bazy danych. W tym celu skorzystać można z opcji dodania nowego elementu szkieletowego (`New Scaffolded Item`), następnie `Razor Components using Entity Framework (CRUD)`. Jako `Template` wybieramy `CRUD`, w `Model class` wybieramy naszą przygotowaną klasę, a jako `DbContext` wybieramy domyślnie utworzoną bazę danych przy tworzeniu kont. W efekcie otrzymamy podstrony z podstawowymi funkcjonalnościami (wylistowanie wszystkich elementów, dodawanie, edycja, usuwanie). W `NavMenu.razor` należy dodać odnośnik tylko do głównej podstrony `Index.razor`. Przed uruchomieniem aplikacji ponownie należy stworzyć nową migrację oraz zaktualizować bazę danych. W celu otrzymania maksymalnej oceny za to zadanie, należy dodać następujące modyfikacje i funkcjonalności:

- w widoku `Details` dodać możliwość dodania oceny, ocena musi zostać zaktualizowana poprzez przeliczenie z już istniejącą oceną a nie podmieniona,
- w widoku `Index` dodać możliwość sortowania po różnych kolumnach,
- w widoku `Index` nie wszystkie informacje powinny być widoczne (np. brak opisu),
- zablokowanie dostępu do części stron bez zalogowania,
- poprawienie menu nawigacyjnego,
- dodanie obrazka, w bazie danych przechować `url`, wyświetlenie w `Details`.

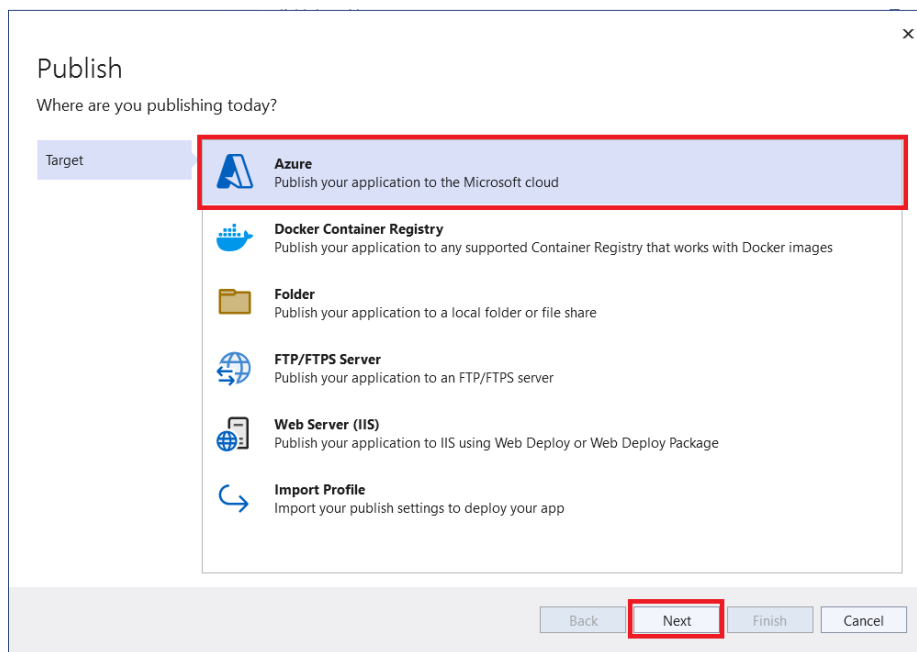
Po konsultacji z prowadzącym jest możliwość zastąpienia części funkcjonalności innymi. Przykładowe inspiracje do rozwoju aplikacji, które będą dodatkowo punktowane:

- wcześniej wspomniane wykonanie autoryzacji użytkowników z wykorzystaniem konta `Google` lub `Microsoft`,
- zakotwiczenie webowego komponentu (np. mapy `Google`, `Twitter`, pogoda, itp.),

- przesył danych pomiędzy dwoma niezależnymi aplikacjami webowymi (swojej i kolegi lub koleżanki) zrobionymi podczas zajęć,
- możliwość wgrania pliku (z dysku, zdjęcia z kamery komputerowej, nagrania dźwięku, itp).

### 3.3 Zadanie 3

Celem zadania jest wdrożenie aplikacji w ramach zasobów Microsoft Azure. Dzięki temu możliwy będzie dostęp do naszej aplikacji z innego dowolnego urządzenia z dostępem do sieci Internet. Zanim zaczniemy, musimy się zalogować na studenckie konto Microsoft w programie Visual Studio w celu użycia subskrypcji, której aktywacja opisana będzie w dalszej części zadania. Jeśli Visual Studio było instalowane z licencji przypisanej do studenckiego adresu e-mail, to nie powinno to stanowić większego problemu. Proces publikowania aplikacji należy rozpocząć od wybrania odpowiedniej akcji z poziomu Solution Explorer, klikając prawym przyciskiem myszy na projekt aplikacji webowej i wybierając opcję Publish... lub z zakładki Build, również wybierając opcję Publish... Interesująca nas opcja to Azure - Host your application to the Microsoft cloud, pokazana na Rysunku 3 (jak widać nie jest to jedyna możliwość).



Rysunek 3: Wybór miejsca publikacji projektu

Następnie należy wybrać opcję Azure App Service (Windows) lub alternatywnie Azure App Service (Linux), definiuje ona pod jakim systemem uruchomiona będzie aplikacja webowa na serwerze. Do wykonania zadania będzie trzeba aktywować darmową subskrypcję Azure w ramach studenckiego konta. Jeśli subskrypcja została wcześniej aktywowana, powinna się automatycznie podpowiedzieć, w przeciwnym wypadku pojawi się link do portalu Azure, gdzie subskrypcję można aktywować. Co ważne, w momencie dodawania nowej subskrypcji, nie należy wybierać opcji Start free, która wymagać będzie podpięcia karty płatniczej, ale wybrać opcję Azure for Students. Po poprawnym aktywowaniu subskrypcji, należy utworzyć nowy AppService. Wiązać się to będzie z:

- nadaniem nowej nazwy serwisu,

- wyborem subskrypcji,
- utworzeniem nowej grupy zasobów,
- stworzeniem nowego planu hostingu, w którym należy nadać nazwę, wybrać lokalizację **Poland Central** oraz wybrać rozmiar **Free**. Pokazano to na Rysunku 4, który przedstawia okno, mające się wyświetlić po wyborze opcji **Create new** w zakładce **App Service**.

Po wpisaniu wszystkich wymaganych opcji należy wybrać opcję **Create**, odczekać chwilę potrzebną na utworzenie serwisu a następnie zakończyć tworzenie i zamknąć okno. Po utworzeniu nowego elementu **App Service**, należy na stronie **Publish** wybrać opcję **Publish**. Pod stworzonym linkiem powinna być dostępna nasza witryna. Aby opublikować zmiany dokonane w aplikacji już po jej opublikowaniu, należy ponownie wybrać opcję **Publish**.

The screenshot shows a dialog box titled "App Service (Windows) Create new". It features a Microsoft account dropdown menu in the top right. The main content area contains four input fields, each highlighted with a red border:

- Name:** MyFirstAzureWebApp20230327113754
- Subscription name:** Visual Studio Enterprise Subscription
- Resource group:** myResourceGroup\* (with a "New..." link to the right)
- Hosting Plan:** MyFirstAzureWebAppPlan\* (West Europe, F1) (with a "New..." link to the right)

At the bottom of the dialog, there are three buttons: "Export...", "Create" (highlighted with a red border), and "Cancel".

Rysunek 4: Tworzenie nowego serwisu aplikacji

Warunkiem zaliczenia zadania, jest zaprezentowanie prowadzącemu Zadania 2, uruchomionego z poziomu platformy **Azure**, a nie lokalnie.