

Grupa .NET Politechnika Krakowska

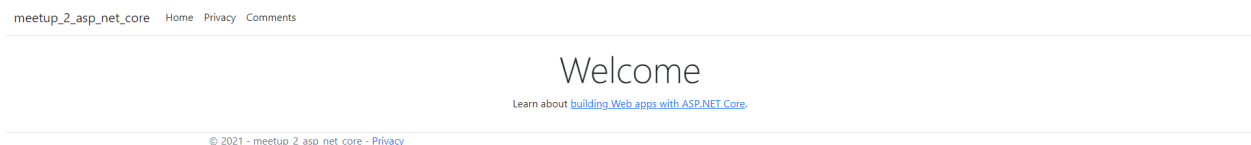
Wprowadzenie do ASP.NET Core

Przypomnienie z poprzednich labów

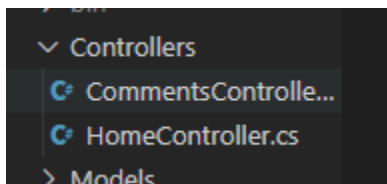
Polecenie **dotnet run** wpisane wewnątrz katalogu projektu uruchamia aplikację. Po każdej zmianie w kodzie trzeba na nowo uruchomić program.

```
PS C:\Users\Jarek\Desktop\NET PK\meetup-2-asp-net-core> dotnet run
Trwa kompilowanie...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7193
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5154
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\Jarek\Desktop\NET PK\meetup-2-asp-net-core\
█
```

Po wejściu w przeglądarce na adres <https://localhost:7193> albo <http://localhost:5154>



W katalogu Controllers są zdefiniowane wszystkie kontrolery. Każdy kontroler ma swoje metody, które nazywane są akcjami. Są to kolejne endpointy w aplikacji.



```
using Microsoft.AspNetCore.Mvc;

You, a day ago | 1 author (You)
namespace meetup_1_asp_net_core.Controllers
{
    0 references | You, a day ago | 1 author (You)
    public class CommentsController : Controller
    {
        0 references
        public IActionResult Index()
        {
            return View();
        }

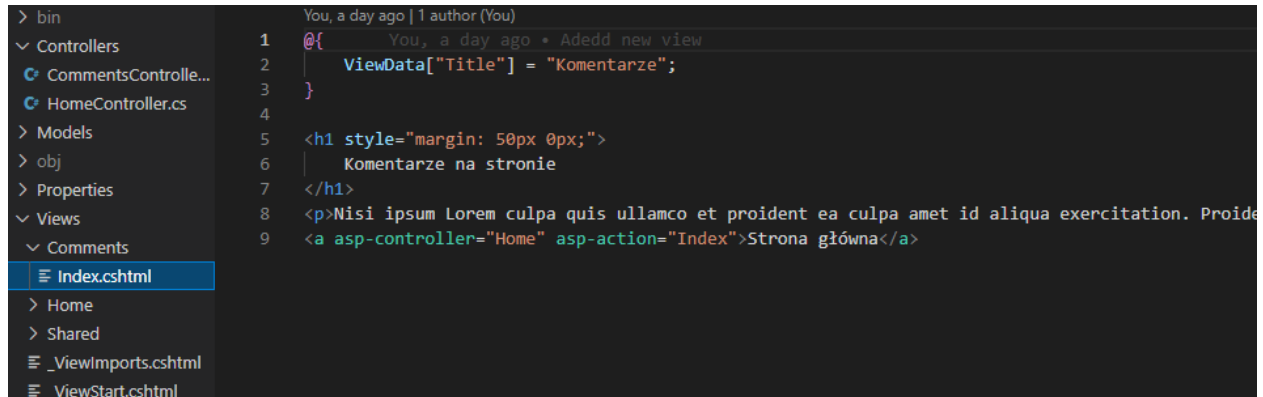
        0 references
        public IActionResult Welcome()
        {
            return StatusCode(503);
        }

        0 references
        public IActionResult Name(string name, int age)
        {
            return Ok(new {
                success = true,
                response = "Ok",
                firstName = name,
                age
            });
        }
    }
}
```

Aby wywołać daną akcję z kontrolera wystarczy wejść na adres
localhost:{port}/{controller}/{akcja}

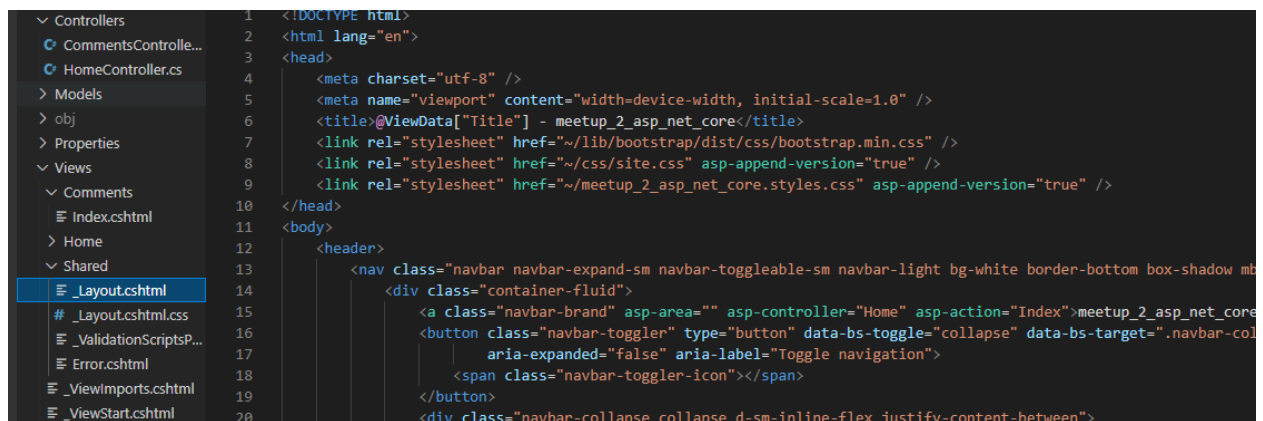
np. localhost:7193/Comments/Index

W katalogu Views znajdują się pliki, które będą zwracane bezpośrednio do klienta. Struktura nie jest przypadkowa. Nazwa katalogu to nazwa kontrolera, natomiast nazwa pliku to nazwa akcji z danego kontrolera. Rozszerzeniem pliku musi być **cshtml**.



```
1 You, a day ago | 1 author (You)
2 @{
3     ViewData["Title"] = "Komentarze";
4 }
5 <h1 style="margin: 50px 0px;">
6     Komentarze na stronie
7 </h1>
8 <p>Nisi ipsum Lorem culpa quis ullamco et proident ea culpa amet id aliqua exercitation. Proident
9 <a asp-controller="Home" asp-action="Index">Strona główna</a>
```

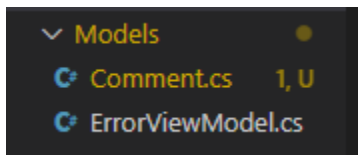
Główny szablon aplikacji (menu, stopka, ułożenie zawartości, itp.) znajduje się w pliku Views/Shared.



```
1 <DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="utf-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>@ViewData["Title"] - meetup_2_asp_net_core</title>
7     <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
8     <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
9     <link rel="stylesheet" href="~/meetup_2_asp_net_core.styles.css" asp-append-version="true" />
10 </head>
11 <body>
12     <header>
13         <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
14             <div class="container-fluid">
15                 <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">meetup_2_asp_net_core</a>
16                 <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse"
17                     aria-expanded="false" aria-label="Toggle navigation">
18                     <span class="navbar-toggler-icon"></span>
19                 </button>
20                 <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
```

Dodanie nowego serwisu

Na początku dodamy model, który będzie zwracany z aplikacji i wyświetlany na ekranie. W katalogu Models utwórzcie sobie plik **Comment.cs**



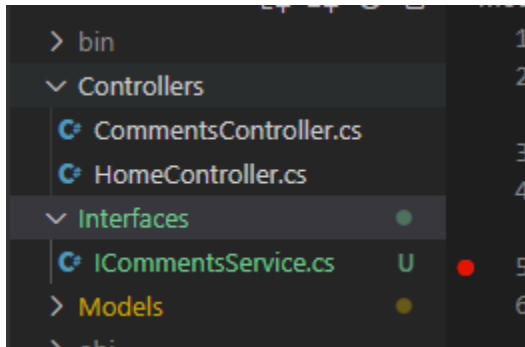
A jako jego zawartość wpiszcie poniższy kod. Ogólnie jest to reprezentacja pojedynczego komentarza, jak chcecie to możecie sobie dodać jakieś nowe pola jak np. **Author**, itp.

```
namespace meetup_2_asp_net_core.Models
{
    1 reference
    public class Comment
    {
        0 references
        public int Id { get; set; }

        0 references
        public string Message { get; set; }

        0 references
        public DateTime AddedAt { get; set; }
    }
}
```

Teraz dodamy sobie nowy interfejs, który będzie opisywał wszystkie dostępne metody dla serwisu, który później utworzymy. Ten serwis będzie odpowiedzialny za dodawanie i wyświetlanie komentarzy.



A jego implementacja będzie wyglądała tak.

```
using meetup_2_asp_net_core.Models;

...

namespace meetup_2_asp_net_core.Interfaces
{
    4 references | ...

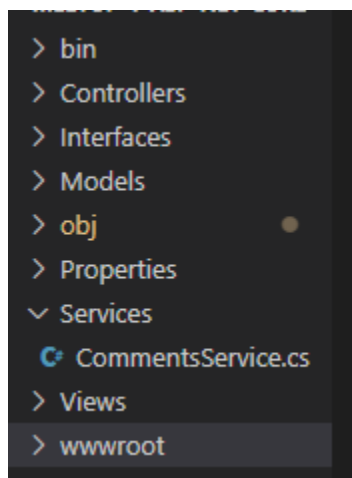
    public interface ICommentsService
    {
        1 reference
        Comment[] GetAllComments();

        0 references
        void CreateComment(string message);
    }
}
```

Ważne jest aby dodać **using meetup_2_asp_net_core.Models;**

Jest to informacja, w którym miejscu znajduje się definicja poszczególnych klas (w naszym przypadku Comment, który wcześniej utworzyliśmy).

Teraz możemy dodać serwis, który de facto będzie już coś robił. Żeby to zrobić należy dodać sobie plik **CommentsService.cs** do katalogu **Services** (jak go nie ma to trzeba go utworzyć).



Sam kod odpowiedzialny za pobieranie i dodawanie komentarzy znajduje się wewnątrz tego pliku. Możecie sobie wpisać takie coś. Definiujemy tutaj nazwę klasy (**CommentsService**) oraz ustalamy jakie metody powinna mieć. Interfejs gwarantuje nam to, że metody zostaną w jakiś sposób zaimplementowane. Bez tego pojawi nam się błąd i aplikacja się nie uruchomi (jak chcecie to sobie to sprawdźcie co się stanie bez dodania metod)

```
using meetup_2_asp_net_core.Models;
using meetup_2_asp_net_core.Interfaces;

namespace meetup_2_asp_net_core.Services
{
    0 references
    class CommentsService : ICommentsService
    {
        0 references
        public Comment[] GetAllComments()
        {
            return new Comment[] {
                new Comment()
                {
                    Id = 1,
                    AddedAt = DateTime.Now,
                    Message = "Komentarz 1"
                },
                new Comment()
                {
                    Id = 2,
                    AddedAt = DateTime.Now.AddHours(-2),
                    Message = "Komentarz 2"
                }
            };
        }

        0 references
        public void CreateComment(string message)
        {
            // Composite formatting
            Console.WriteLine("Comment created with content {0} at {1}", message, DateTime.Now);
        }
    }
}
```


Dependency Injection

Aby udostępnić nasz serwis globalnie do całej funkcji, należy dodać go do scope'a serwisów. Robi to się w pliku **Program.cs**. Dodajcie sobie linijki, które zaczynają się od zielonej obramówki. Linijka 8 dodaje nasz nowy serwis, ale potrzebujemy informacji skąd ma sobie je wziąć i dlatego musimy zrobić importy.

```
You, seconds ago | 1 author (You)
1  using meetup_2_asp_net_core.Interfaces;
2  using meetup_2_asp_net_core.Services;
3
4  var builder = WebApplication.CreateBuilder(args);
5
6  // Add services to the container.
7  builder.Services.AddControllersWithViews();
8  builder.Services.AddScoped<ICommentsService, CommentsService>();
9  
```

Teraz możemy wykorzystać metody, które utworzyliśmy. Aby to zrobić trzeba wykonać 3 następujące kroki:

1. Utworzyć nową zmienną, której typem jest interfejs z metodami
2. W konstruktorze wstrzyknąć zależność (ICommentsService commentsService)
3. Wywołać metodę w dowolnym miejscu w naszym kontrolerze

```

You, seconds ago | 1 author (You)
1 using Microsoft.AspNetCore.Mvc;
2 using meetup_2_asp_net_core.Interfaces;
3
You, seconds ago | 1 author (You)
4 namespace meetup_2_asp_net_core.Controllers
5 {
6     0 references | You, seconds ago | 1 author (You)
7     public class CommentsController : Controller
8     {
9         2 references
10        private readonly ICommentsService _commentsService;
11
12        0 references
13        public CommentsController(ICommentsService commentsService)
14        {
15            _commentsService = commentsService;
16        }
17
18        0 references
19        public IActionResult Index()
20        {
21            var comments = _commentsService.GetAllComments();
22
23            return View(comments);
24        }
25
26        You, 2 days ago • Added new view

```

Teraz wystarczy, że wyświetlimy nasze komentarze na stronie. W katalogu Views/Comments/Index dodajemy następujący kod:

```

1 @model IEnumerable<Comment>
2
3 @{
4     ViewData["Title"] = "Komentarze";
5 }
6
7 <h1 style="margin: 50px 0px;">
8     Komentarze na stronie
9 </h1>
10 <p>Nisi ipsum Lorem culpa quis ullamco et proident ea culpa amet id aliqua exercitation. Proident officia magna ipsum id mollit velit elit laboris sint. Culpa labore minima labore nostrud adip.
11 <a asp-controller="Home" asp-action="Index">Strona główna</a>
12
13 @foreach (var comment in Model)
14 {
15     <div style="margin-top: 50px;">
16         <i>@comment.AddedAt</i>
17         <hr />
18         @comment.Message
19     </div>
20 }

```

Na samym początku definiujemy zwracany typ obiektu z kontrolera

```
You, seconds ago | I author (You)  
1 | @model IEnumerable<Comment>  
2 |
```

Wypisanie komentarzy zwróconych z kontrolera na ekranie znajduje się w pętli foreach

```
0  
1 | @foreach (var comment in Model)  
2 | {  
3 |     <div style="margin-top: 50px;">  
4 |         <i>@comment.AddedAt</i>  
5 |         <hr />  
6 |         @comment.Message  
7 |     </div>  
8 | }  
9 |
```

Model jest to zmienna, która przechowuje wszystko to co zostało zwrócone z kontrolera. Dodanie typów znajduje się w pierwszej linijce widoku (**@model ...**)

Dzięki temu możemy zobaczyć listę komentarzy na stronie:

```
01.12.2021 15:43:40  
Komentarz 1  
  
01.12.2021 13:43:40  
Komentarz 2
```

Dodawanie nowych komentarzy

Teraz dodamy sobie możliwość dodawania nowych komentarzy.

Pierwszą rzeczą jest dodanie formularza na naszej stronie (**Views/Comments/Index.cshtml**)

```
21
22 <form method="post" asp-controller="Comments" asp-action="Send" style="margin-top: 100px;">
23     <label>
24         Dodaj komentarz <br/>
25         <input name="Message" maxlength="10" placeholder="Treść komentarza" required />
26     </label>
27     <input type="submit" value="Dodaj" />
28 </form>
29
```

Teraz musimy dodać nowy endpoint, który będzie nasłuchiwał zapytań pod adresem **/Comments/Send**. Dane przesyłane są metodą POST i dlatego też będziemy musieli to zaimplementować.

Na samym początku zdefiniujemy sobie obiekt, który będzie przesyłany z formularza.

```
namespace meetup_2_asp_net_core.Models
{
    0 references
    public class NewCommentRequest
    {
        0 references
        public string Message { get; set; }
    }
}
```

W pliku **Models/NewCommentRequest** utwórzcie sobie następującą treść. Pola tej klasy są nazwami inputów z formularza powyżej (**Message**)

Teraz w pliku CommentsController dodajemy nowy import, w którym znajduje się definicja naszego requesta.

```
using meetup_2_asp_net_core.Interfaces;  
using meetup_2_asp_net_core.Models;
```

Dodajemy nową metodę. Adnotacja **[HttpPost]** oznacza, że ten endpoint będzie nasłuchiwał zapytań typu POST (domyślnie nasłuchuje GET)

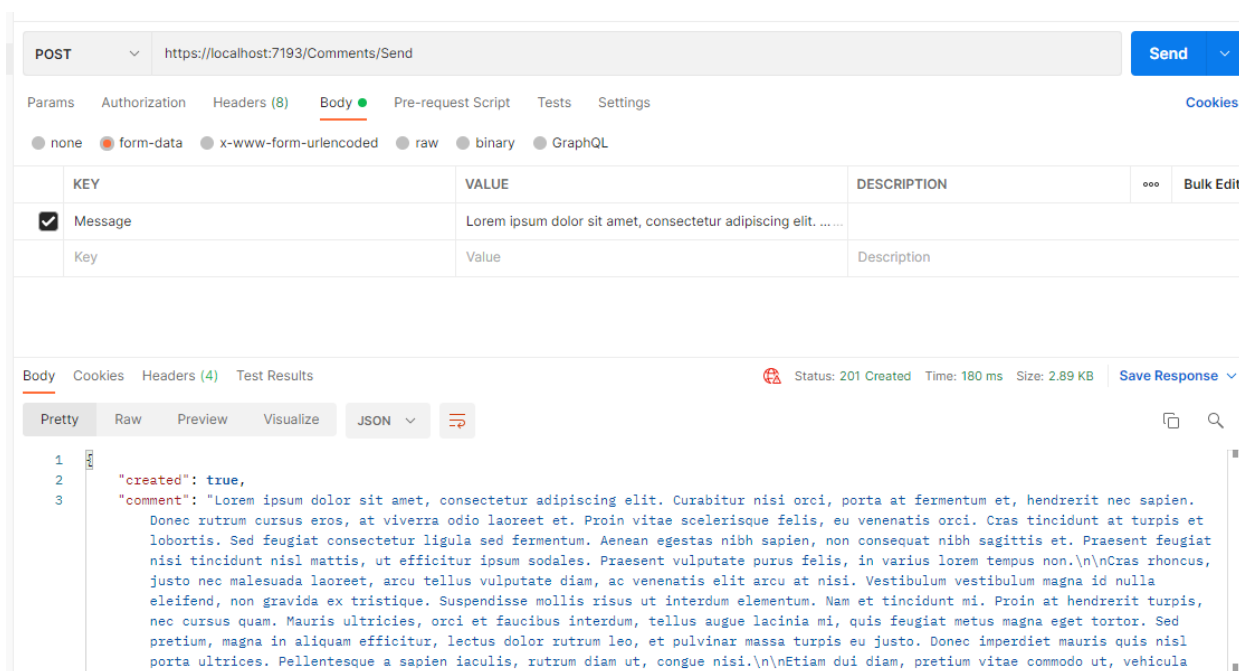
```
32  
33 | [HttpPost]  
    0 references  
34 | public IActionResult Send([FromForm] NewCommentRequest request)  
35 | {  
36 |     _commentsService.CreateComment(request.Message);  
37 |  
38 |     return StatusCode(201, new {  
39 |         Created = true,  
40 |         Comment = request.Message,  
41 |     });  
42 | }  
43 |
```

Po zapisaniu. Możemy zauważyć, że akcja się poprawnie wykonuje. W logach aplikacji znajduje się info dotyczące nowego komentarza.

```
Content root path: C:\Users\Jarek\Desktop\NET PK\meetu  
Comment created with content Test at 01.12.2021 16:03:49
```

Model Validation

Jak narazie dodaliśmy zabezpieczenie od strony frontendowej (maksymalna długość ciągu znaków). Jednak nie jest to żadne rozwiązanie. Po wejściu w dowolne narzędzie do wysyłania zapytań HTTP (np. Postman), możecie wysłać request z dużo większą ilością znaków niż ta, która jest dozwolona w formularzu.



Tak więc, zabezpieczenia frontendowe to żadne zabezpieczenia. Wciąż można zepsuć naszą aplikację.

Najprostszą warstwą bezpieczeństwa, jest dodanie walidacji modelu (Model Validation), który sprawdza czy dane wejściowe do kontrolera są poprawne. Robi to się bardzo prosto przy użyciu atrybutów.

W modelu NewCommentRequest wystarczy dodać sobie kilka reguł, które muszą zostać spełnione, aby potraktować nasz request jako prawidłowy.

```

1 reference | You, 6 minutes ago | 1 author (You)
public class NewCommentRequest
{
    [Required]
    [MaxLength(10)]
    [MinLength(0)]
    // [StringLength(100)]
    2 references
    public string Message { get; set; }
}

```

Teraz wystarczy wywołać tylko metodę walidującą nasz request i cała obsługa poprawności zostanie obsłużona.

```

0 references
public IActionResult Send([FromForm] NewCommentRequest request)
{
    if (!ModelState.IsValid)
        return BadRequest();

    _commentsService.CreateComment(request.Message);
}

```

Teraz nie jesteśmy w stanie dodać niepoprawnego pliku.

POST https://localhost:7193/Comments/Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> Message	Lorem ipsum dolor sit amet, consectetur adipiscing elit. ...	
Key	Value	Description

Body Cookies Headers (3) Test Results

Status: 400 Bad Request Time: 209 ms Size: 101 B Save Response

Pretty Raw Preview Visualize Text

1

Zapisywanie danych w pamięci aplikacji

Na dzisiejszym spotkaniu zapiszemy sobie dane w pamięci aplikacji. Na samym początku poprawimy UX, i dodamy przekierowanie na główną stronę komentarzy po dopisaniu komentarza.

```
[HttpPost]
0 references
public IActionResult Send([FromForm] NewCommentRequest request)
{
    if (!ModelState.IsValid)
        return BadRequest();

    _commentsService.CreateComment(request.Message);

    return Redirect("/Comments");
}
```

Teraz zmieniamy implementację dodawania i odczytywania komentarzy w taki sposób, aby te metody operowały na statycznej liście **List<Comment> Comments**

Statyczne zmienne zapewniają nam to, że w obrębie całej aplikacji mamy zawsze tylko jedną zmienną, która jest dostępna do użycia w każdym miejscu. W naszym przypadku w tej zmiennej przechowywać będziemy nasze wszystkie komentarze.


```

You, seconds ago | 1 author (You)
using meetup_2_asp_net_core.Models;
using meetup_2_asp_net_core.Interfaces;
You, 16 hours ago • Added DI

You, seconds ago | 1 author (You)
namespace meetup_2_asp_net_core.Services
{
    1 reference | You, seconds ago | 1 author (You)
    class CommentsService : ICommentsService
    {
        2 references
        public static List<Comment> Comments = new List<Comment>() {
            new Comment()
            {
                Id = 1,
                AddedAt = DateTime.Now,
                Message = "Komentarz 1"
            },
            new Comment()
            {
                Id = 2,
                AddedAt = DateTime.Now.AddHours(-2),
                Message = "Komentarz 2"
            }
        };

        0 references
        public List<Comment> GetAllComments()
        {
            return Comments;
        }

        1 reference
        public void CreateComment(string message)
        {
            Comments.Add(new Comment
            {
                AddedAt = DateTime.Now,
                Id = 3,
                Message = message
            });

            // Composite formatting
            Console.WriteLine("Comment created with content {0} at {1}", message, DateTime.Now);
        }
    }
}

```

Jak możecie zauważyć zwracamy teraz listę a nie tablicę jak to miało miejsce wcześniej. Aby poprawić ten błąd musimy zmienić sygnatury w interfejsie.

```
You, seconds ago | 1 author (You)
using meetup_2_asp_net_core.Models;

You, seconds ago | 1 author (You)
namespace meetup_2_asp_net_core.Interfaces
{
    4 references | You, seconds ago | 1 author (You)
    public interface ICommentsService
    {
        1 reference
        List<Comment> GetAllComments();

        1 reference
        void CreateComment(string message);
    }
}
```

Teraz już wszystko działa jak należy. Żadne błędy się nie pokazują.

```
You, 4 minutes ago | 1 author (You)
namespace meetup_2_asp_net_core.Services
{
    1 reference | You, 4 minutes ago | 1 author (You)
    class CommentsService : ICommentsService
    {
        2 references
        public static List<Comment> Comments = new List<Comment>() {
            new Comment()
            {
                Id = 1,
                AddedAt = DateTime.Now,
                Message = "Komentarz 1"
            }
        }
    }
}
```

Przetestowanie aplikacji

Teraz możecie wejść sobie na stronę i dodać nowy komentarz, podczas działania aplikacji nowe komentarze będą zapisywane. W chwili kiedy zresetujemy aplikację, utracimy wszystkie nowo wprowadzone komentarze.

Komentarze na stronie

Nisi ipsum Lorem culpa quis ullamco et proident ea culpa amet id aliqua exercitation. Proident officia magna ipsum id mollit velit elit laboris sint. Culpa labore minim labore nostrud adipiscing nostrud laborum ad est dolor sint consequat duis. Occaecat velit ex amet excepteur esse consectetur eu ex duis amet reprehenderit culpa. Incidunt deserunt mollit amet deserunt ad duis aliqua eiusmod veniam. Mollit eu occaecat aliquip aute minim eu aliquip minim id excepteur.

[Strona główna](#)

01.12.2021 16:45:51

Komentarz 1

01.12.2021 14:45:51

Komentarz 2

01.12.2021 16:45:53

Test

01.12.2021 16:45:55

Dwa

Dodaj komentarz

Treść komentarza

Dodaj



Zadanko do wykonania

Dodajcie do apki możliwość wyświetlenia użytkowników systemu (nowy widok i kontroler).

Po kliknięciu w użytkownika przeniesie nas do osobnej podstrony ze szczegółami danej osoby (ścieżka np. /Users/14). Gdzie 14 to Id użytkownika.