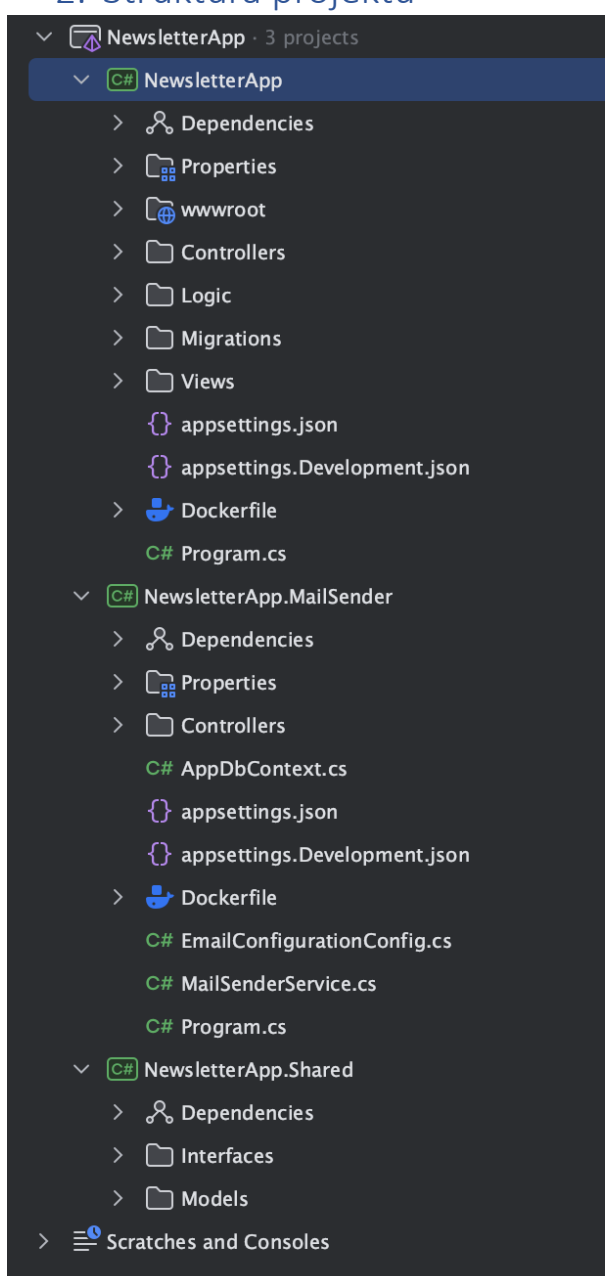


# Dokumentacja NewsLetterApp

## 1. Opis projektu

NewsLetterApp to system do tworzenia oraz rozsyłania wiadomości z zawartością między osoby, które zasubskrybują wybrany przez siebie temat. System został napisany w technologii .NET Core oraz zbudowany z trzech kontenerów dockerowych. Pierwszym z nich jest baza danych, w której trzymane są dane użytkowników, oraz naszych tematów wraz z zawartością. Drugim kontenerem jest główna aplikacja, która odpowiada za logowanie, oraz tworzenie naszych tematów, subskrypcję ich i zarządzanie nimi. Trzeci kontener to serwis, który po otrzymaniu odpowiedniego zapytania wysyła maile do osób, które subskrybują dany newsletter.

## 2. Struktura projektu



Struktura projektu to 2 aplikacje i 1 projekt, w którym są przechowywane klasy, które są wspólne dla obu aplikacji. NewsLetterApp to główna aplikacja, NewsLetterApp.MailSender to aplikacja-serwis, do rozsyłania maili, a NewsLetterApp.Shared to biblioteka ze wspólną zawartością obu aplikacji.

### 3. Połączenie aplikacji z bazą danych

Do połączenia z bazą danych został użyty ORM Entity Framework Core, który poza połączeniem z bazą danych odpowiada również za obsługę autoryzacji i autentykacji użytkowników. Została napisana klasa AppDbContext, która jest odpowiednikiem naszej bazy danych, jej tabeli oraz przekształca kod C# na zapytania SQL w celu zarządzania naszymi danymi w bazie.

```
public class AppDbContext : IdentityDbContext<ApplicationUser>, IAppDbContext
{
    private readonly IHttpContextAccessor _httpContextAccessor;

    public AppDbContext(DbContextOptions<AppDbContext> options, IHttpContextAccessor httpContextAccessor) : base(options)
    {
        _httpContextAccessor = httpContextAccessor;
    }

    public DbSet<Newsletter> NewsLetters { get; set; }
    public DbSet<NewsContent> NewsContents { get; set; }
    public DbSet<NewsletterSubscriber> NewsletterSubscribers { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Newsletter>(x =>
        {
            x.HasKey(x => x.Id);

            x.HasMany(x => x.NewsContents)
                .WithOne(x => x.NewsLetter)
                .HasForeignKey(x => x.NewsLetterId)
                .OnDelete(DeleteBehavior.Cascade);

            x.HasMany(x => x.NewsLetterSubscribers)
                .WithMany(x => x.NewsLetters);
        });

        modelBuilder.Entity<NewsContent>(x =>
        {
            x.HasKey(x => x.Id);
        });

        modelBuilder.Entity<NewsletterSubscriber>(x =>
        {
            x.HasKey(x => x.Id);
        });

        base.OnModelCreating(modelBuilder);
    }

    public override Task<int> SaveChangesAsync(CancellationToken cancellationToken = new CancellationToken())
    {
        foreach (var entry in ChangeTracker.Entries<AuditableEntity>())
        {
            switch (entry.State)
            {
                case EntityState.Deleted:
                    entry.Entity.ModifiedBy = GetCurrentUser();
                    entry.Entity.Modified = DateTime.Now;
                    entry.Entity.InactivatedBy = GetCurrentUser();
                    entry.Entity.InactivatedBy = GetCurrentUser();
                    entry.Entity.StatusId = 0;
                    entry.State = EntityState.Modified;
                    break;
                case EntityState.Modified:
                    entry.Entity.ModifiedBy = GetCurrentUser();
                    entry.Entity.Modified = DateTime.Now;
                    break;
                case EntityState.Added:
                    entry.Entity.CreatedBy = GetCurrentUser();
                    entry.Entity.Created = DateTime.Now;
                    entry.Entity.StatusId = 1;
                    break;
                default:
                    break;
            }
        }

        return base.SaveChangesAsync(cancellationToken);
    }

    private string GetCurrentUser()
    {
        return _httpContextAccessor.HttpContext?.User?.Identity?.Name ?? "Anonymous";
    }
}
```

## 4. Rejestracja, logowanie i zarządzanie kontem użytkownika

W celu zarządzania newsletterami niezbędne było zabezpieczenie systemu przed osobami nieuprawnionymi do jego obsługi. W tym celu powstało logowanie, rejestracja oraz zarządzanie kontem administratora. Tylko osoba z uprawnieniami administratora może rejestrować nowe osoby, które będą zarządzać newsletterami. Poniżej przedstawiono klasę do zarządzania użytkownikami.

```
public class UserAuthenticationController : Controller
{
    private readonly IUserAuthenticationService _authService;

    public UserAuthenticationController(IUserAuthenticationService authService)
    {
        _authService = authService;
    }

    [AllowAnonymous]
    public async Task<IActionResult> RegisterAdmin()
    {
        RegistrationModel model = new RegistrationModel
        {
            Username = "admin",
            Email = "admin@gmail.com",
            FirstName = "Adam",
            LastName = "Ludwiczak",
            Password = "Pass123$"
        };
        model.Role = "admin";
        var result = await _authService.RegisterAsync(model);
        return RedirectToAction("Login");
    }

    public IActionResult Login()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Login(LoginModel model)
    {
        if (!ModelState.IsValid)
            return View(model);
        var result = await _authService.LoginAsync(model);
        if (result.StatusCode == 1)
        {
            return RedirectToAction("Index", "Home");
        }
        else
        {
            TempData["msg"] = result.Message;
            return RedirectToAction(nameof(Login));
        }
    }

    public IActionResult Registration()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Registration(RegistrationModel model)
    {
        if (!ModelState.IsValid)
        {
            return View(model);
        }

        model.Role = "user";
        var result = await _authService.RegisterAsync(model);
        TempData["msg"] = result.Message;
        return RedirectToAction(nameof(Registration));
    }

    [Authorize]
    public async Task<IActionResult> Logout()
    {
        await _authService.LogoutAsync();
        return RedirectToAction(nameof(Login));
    }

    [Authorize]
    public IActionResult ChangePassword()
    {
        return View();
    }

    [Authorize]
    [HttpPost]
    public async Task<IActionResult> ChangePassword(ChangePasswordModel model)
    {
        if (!ModelState.IsValid)
            return View(model);
        var result = await _authService.ChangePasswordAsync(model, User.Identity.Name);
        TempData["msg"] = result.Message;
        return RedirectToAction(nameof(ChangePassword));
    }
}
```

A oto jak wygląda interfejs dla podanego kontrolera.

## Logowanie

NewsletterApp Home Login

**Username**

**Password**

Login

[Not a user? Signup](#)

## Rejestracja

NewsletterApp News Letters Registration ChangePassword Logout

### Registration

FName\*

LName\*

Email

Username

Password (eg. John@123)

Password confirm

Registration

Login

## Zmiana hasła

NewsletterApp News Letters Registration ChangePassword Logout

### Registration

Current Password\*

New Password\*

Confirm New Password\*

Save

## 5. Strona główna

Na stronie głównej użytkownik widzi listę wszystkich newsletterów, do których może się zapisać i otrzymywać wiadomości z nich na skrzynkę pocztową którą podał podczas subskrypcji.

NewsletterApp News Letters Registration ChangePassword Logout

News Letters List	
Name	Subscribe
.Net	<button>Subscribe</button>
Docker nowości	<button>Subscribe</button>
Python - wąż dla początkujących	<button>Subscribe</button>
SQL - Krótkie porady optymalnych zapytań	<button>Subscribe</button>

Kod klasy kontrolera strony głównej

```
[AllowAnonymous]
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;
    private readonly INewLettersService _newLettersService;

    public HomeController(ILogger<HomeController> logger, INewLettersService newLettersService)
    {
        _logger = logger;
        _newLettersService = newLettersService;
    }

    public async Task<IActionResult> Index()
    {
        var response = await _newLettersService.GetList();

        return View(response);
    }

    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
    public IActionResult Error()
    {
        return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
    }
}
```

Po wybraniu interesującego nas tematu klikamy przycisk Subscribe i przenosi nas do strony, gdzie wpiszemy nasz email, na który będą przychodzić newsy.

NewsletterApp Home Login

### Subscribe

Email

## 6. Panel zarządzania newsami

Administrator posiada dostęp do panelu, gdzie może tworzyć tematy newsletterów oraz ich treści. Panel wygląda następująco.

NewsletterApp News Letters Registration ChangePassword Logout

News Letter List

Create news letter

Name	Edit	Delete
.Net	Edit	Delete
Docker nowości	Edit	Delete
Python - wąż dla początkujących	Edit	Delete
SQL - Krótkie porady optymalnych zapytań	Edit	Delete

W tym panelu administrator zarządza istniejącymi newsletterami oraz może utworzyć nowy.

NewsletterApp News Letters Registration ChangePassword Logout

Create News Letter

Name

Create

Po stworzeniu mamy możliwość przejścia do panelu zarządzania wątkiem, gdzie możemy tworzyć newsy oraz wysyłać je do subskrybujących nasz newsletter.

NewsletterApp News Letters Registration ChangePassword Logout

SQL - Krótkie porady optymalnych zapytań

Create news

Title	Send	Edit	Delete
Wstęp do pythona - środowisko oraz IDE	Send	Edit	Delete

NewsletterApp News Letters Registration ChangePassword Logout

Title

Wstęp do pythona - środowisko oraz IDE

Content

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam et pulvinar sapien, et rhoncus lacus. In a quam non urna lobortis molestie. Donec maximus fringilla pharetra. Vivamus tempor mattis pharetra. Curabitur ultricies pretium augue et dapibus. Pellentesque metus nisi, varius sit amet felis ac, fermentum venenatis sapien. Fusce suscipit elit eu mollis mollis. In sodales congue libero, vel tempus velit luctus vitae. Praesent lacinia hendrerit sollicitudin. Donec non leo velit. Integer diam nisi, dapibus at velit eu, malesuada lobortis urna.

Ut efficitur dapibus quam, sit amet accumsan elit efficitur et. Nam pellentesque arcu a massa luctus venenatis. Donec et risus ultrices, scelerisque erat at, blandit quam. Pellentesque condimentum porta vulputate. Quisque in lorem aliquet, elementum justo eget, pretium ex. Aliquam aliquam egestas dui, id laoreet ligula ultrices id. Etiam non aliquam quam. Vivamus est magna, eleifend eu molestie non, convallis et arcu. Nulla suscipit sagittis molestie.

Vivamus eget porttitor erat. In ut lacus eu lectus commodo commodo eu eu risus. Duis suscipit, neque ut ultricies tempor, metus turpis ultricies orci, quis congue eros lacus sit amet leo. Morbi nec vulputate tortor. Pellentesque ac dui vestibulum, finibus urna non, feugiat magna. Nullam risus lorem, suscipit vel lobortis vel, consectetur a purus. Quisque sit amet erat cursus, placerat est sit amet, fringilla massa. Fusce eu orci velit. Aenean interdum efficitur justo id interdum.

Nam in maximus augue, at suscipit nisi. Donec euismod augue non orci porttitor, non pharetra odio accumsan. Nulla cursus ipsum et risus feugiat efficitur. Pellentesque congue ante malesuada, blandit nunc vitae, elementum est. Integer sit amet ullamcorper justo. Suspendisse feugiat magna nunc, dictum hendrerit lectus finibus laoreet. Curabitur in rutrum massa. Etiam interdum ut urna eget luctus. Nunc ac nunc molestie orci dapibus placerat in ut ipsum. Ut condimentum nisl malesuada massa venenatis malesuada.

Donec turpis tortor, consectetur a tempor quis, feugiat vitae metus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Duis efficitur dolor nec tortor viverra sollicitudin. Maecenas id diam fringilla, rutrum ex vitae, placerat sem. Nullam vitae metus lectus. Nam viverra, quam vitae imperdiet volutpat, risus felis malesuada diam, non ullamcorper felis orci ac massa. Maecenas vel commodo ligula. Suspendisse varius at felis non tincidunt. Curabitur vitae justo arcu. Nunc tortor felis, tempor eget elementum in, tristique a nunc. Donec nec magna scelerisque, suscipit purus id, lobortis ligula. Proin quam turpis, commodo ac dictum et, malesuada quis lorem.

Create

## Kod klas kontrolerów panelu administratora.

```
[Authorize]
public class NewsletterController : Controller
{
    private readonly INewLettersService _newLettersService;

    public NewsletterController(INewLettersService newLettersService)
    {
        _newLettersService = newLettersService;
    }

    public async Task<IActionResult> Index()
    {
        var model = await _newLettersService.GetList();

        return View(model);
    }

    [HttpGet]
    [AllowAnonymous]
    public async Task<IActionResult> Subscribe(Guid id)
    {
        ViewBag.NewsLetterId = id;

        return View();
    }

    [HttpPost]
    [AllowAnonymous]
    public async Task<IActionResult> Subscribe(SubscribeNewsletterDto dto)
    {
        var result = await _newLettersService.Subscribe(dto.NewsLetterId, dto.Email);

        return RedirectToAction("Index", "Home");
    }

    [AllowAnonymous]
    public async Task<IActionResult> Unsubscribe(Guid subscriberId, Guid newsLetterId)
    {
        var result = await _newLettersService.Unsubscribe(newsLetterId, subscriberId);

        ViewBag.Message = result.Message;
        return View("UnsubscribeInfo");
    }

    public IActionResult Create()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Create(CreateNewsletterDto model)
    {
        var result = await _newLettersService.Create(model);

        if (result.StatusCode == 1)
        {
            return RedirectToAction("Index");
        }

        return View(model);
    }

    public async Task<IActionResult> Delete(Guid id)
    {
        var result = await _newLettersService.Delete(id);

        return RedirectToAction("Index");
    }

    public async Task<IActionResult> Edit(Guid id)
    {
        var result = await _newLettersService.GetEditDto(id);

        if (result == null)
        {
            return RedirectToAction("Index");
        }

        return View(result);
    }

    public async Task<IActionResult> SaveEdit(EditNewsletterDto dto)
    {
        var result = await _newLettersService.SaveEditedNewsletter(dto);

        if (result.StatusCode == 1)
        {
            return RedirectToAction("Index");
        }

        return RedirectToAction("Edit", dto);
    }
}
```

```

[Authorize]
public class NewsContentController : Controller
{
    private readonly INewsLetterContentsService _newsLetterContentsService;
    private readonly IHttpService _httpService;
    private readonly ILogger<NewsContentController> _logger;

    public NewsContentController(INewsLetterContentsService newsLetterContentsService,
        IHttpService httpService,
        ILogger<NewsContentController> logger)
    {
        _newsLetterContentsService = newsLetterContentsService;
        _httpService = httpService;
        _logger = logger;
    }

    public async Task<IActionResult> Index(Guid id)
    {
        var response = await _newsLetterContentsService.GetNewsLettersContentForNewsLetter(id);

        return View(response);
    }

    public async Task<IActionResult> CreateGetView(Guid newsLetterId)
    {
        ViewBag.NewsLetterId = newsLetterId;

        return View("Create");
    }

    [HttpPost]
    public async Task<IActionResult> Create(CreateNewsLetterContentDto model)
    {
        var response = await _newsLetterContentsService.Create(model);

        if (response.StatusCode == 1)
        {
            return RedirectToAction("Index", new { id = model.NewsLetterId });
        }

        ViewBag.NewsLetterId = model.NewsLetterId;

        return View(model);
    }

    [HttpGet]
    public async Task<IActionResult> Edit(Guid id)
    {
        var response = await _newsLetterContentsService.GetUpdateModel(id);

        return View(response);
    }

    [HttpPost]
    public async Task<IActionResult> Edit(UpdateNewsLetterContentDto model)
    {
        var response = await _newsLetterContentsService.Update(model);

        if (response.StatusCode == 1)
        {
            return RedirectToAction("Index", new { id = model.NewsLetterId });
        }

        return View(model);
    }

    public async Task<IActionResult> Delete(Guid id, Guid newsLetterId)
    {
        var response = await _newsLetterContentsService.Delete(id);

        return RedirectToAction("Index", new { id = newsLetterId });
    }

    [HttpGet]
    public async Task<IActionResult> Send(Guid id, Guid newsLetterId)
    {
        var url = $"{Environment.GetEnvironmentVariable("MAIL_SENDER_URL")}/MailSender/SendMails?id={id}";

        try
        {
            var response = await _httpService.GetAsync(url);
        }
        catch (Exception e)
        {
            _logger.LogError(e.Message);
        }

        return RedirectToAction("Index", "Newsletter", new { id = newsLetterId });
    }
}

```



## 7. Serwis do wysyłania maili

Serwis do wysyłania maili jest prostą aplikacją w postaci API, która po otrzymaniu odpowiedniego zapytania, które składa się z linku end-pointa oraz id newsa pobiera wszystkie dane i przygotowuje treść maila i listę maili do których należy ją wysłać.

```
[ApiController]
[Route("[controller]")]
[AllowAnonymous]
public class MailSenderController : ControllerBase
{
    private readonly IAppDbContext _ctx;
    private readonly IMailSenderService _mailSenderService;
    private readonly ILogger<MailSenderController> _logger;

    public MailSenderController(IAppDbContext ctx, IMailSenderService mailSenderService, ILogger<MailSenderController> logger)
    {
        _ctx = ctx;
        _mailSenderService = mailSenderService;
        _logger = logger;
    }

    [HttpGet]
    public async Task<IActionResult> SendMails(Guid id)
    {
        var newsletter = await _ctx.NewsContents
            .Include(x => x.NewsLetter)
            .ThenInclude(x => x.NewsLetterSubscribers)
            .FirstOrDefaultAsync(x => x.Id == id);

        if (newsletter == null)
            return NotFound("Newsletter not found");

        var subscribers = newsletter.NewsLetterSubscribers.Select(x => new {x.Email, x.Id}).ToList();
        var title = newsletter.Title;
        var content = newsletter.Content;

        foreach (var subscriber in subscribers)
        {
            _logger.LogInformation("Sending mail to {subscriber}");
            await _mailSenderService.SendMail(subscriber.Email, title, ContentBuilder(content, newsletter.NewsLetterId, subscriber.Id));
        }

        return Ok();
    }

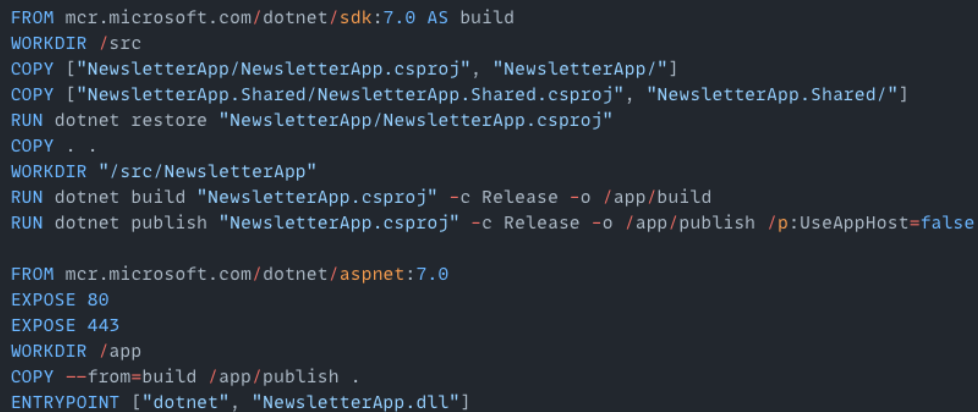
    private string ContentBuilder(string content, Guid newsletterId, Guid subscriberId)
    {
        content += '\n';
        content += '\n';
        content += $"Unsubscribe: {Environment.GetEnvironmentVariable("")}/Newsletter?subscriberId={subscriberId}&newsletterId={newsletterId}";

        return content;
    }
}
```

## 8. Utworzenie Dockerfile oraz docker-compose

Do obu aplikacji powstały dwa pliki Dockerfile z instrukcjami do utworzenia obrazów dockerowych tych aplikacji.

NewsletterApp – aplikacja główna

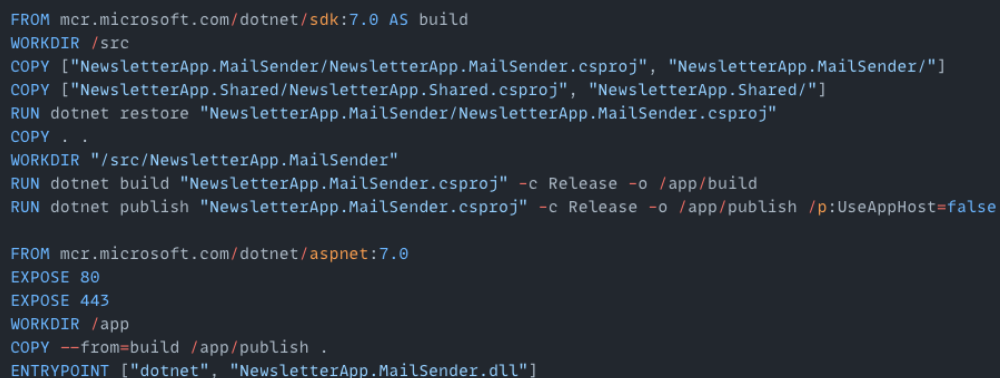


```
FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /src
COPY ["NewsletterApp/NewsletterApp.csproj", "NewsletterApp/"]
COPY ["NewsletterApp.Shared/NewsletterApp.Shared.csproj", "NewsletterApp.Shared/"]
RUN dotnet restore "NewsletterApp/NewsletterApp.csproj"
COPY . .
WORKDIR "/src/NewsletterApp"
RUN dotnet build "NewsletterApp.csproj" -c Release -o /app/build
RUN dotnet publish "NewsletterApp.csproj" -c Release -o /app/publish /p:UseAppHost=false

FROM mcr.microsoft.com/dotnet/aspnet:7.0
EXPOSE 80
EXPOSE 443
WORKDIR /app
COPY --from=build /app/publish .
ENTRYPOINT ["dotnet", "NewsletterApp.dll"]
```

copygify.com

NewsletterApp.MailSender – serwis do wysyłania maili



```
FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /src
COPY ["NewsletterApp.MailSender/NewsletterApp.MailSender.csproj", "NewsletterApp.MailSender/"]
COPY ["NewsletterApp.Shared/NewsletterApp.Shared.csproj", "NewsletterApp.Shared/"]
RUN dotnet restore "NewsletterApp.MailSender/NewsletterApp.MailSender.csproj"
COPY . .
WORKDIR "/src/NewsletterApp.MailSender"
RUN dotnet build "NewsletterApp.MailSender.csproj" -c Release -o /app/build
RUN dotnet publish "NewsletterApp.MailSender.csproj" -c Release -o /app/publish /p:UseAppHost=false

FROM mcr.microsoft.com/dotnet/aspnet:7.0
EXPOSE 80
EXPOSE 443
WORKDIR /app
COPY --from=build /app/publish .
ENTRYPOINT ["dotnet", "NewsletterApp.MailSender.dll"]
```

copygify.com

Plik docker compose jest plikiem, który automatyzuje za nas większość operacji, które trzeba zrobić, aby odpalić obrazy w środowisku dockerowym.

```
version: "3.8"

services:
  sqlserver:
    image: mcr.microsoft.com/azure-sql-edge
    container_name: sqlserver-newsletter
    environment:
      - ACCEPT_EULA=1
      - MSSQL_SA_PASSWORD=Pass12345
    ports:
      - 18000:1433

  newsletter:
    image: newsletter
    container_name: newsletter
    build:
      context: .
      dockerfile: ./NewsletterApp/Dockerfile
    ports:
      - 12000:80
      - 12001:443
    environment:
      - ASPNETCORE_URLS=https://+:http://+
      - ASPNETCORE_HTTPS_PORT=12001
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_Kestrel__Certificates__Default__Password=p455word!
      - ASPNETCORE_Kestrel__Certificates__Default__Path=https://newsletterapp.pfx
      - CONNECTION_STRING=Server=host.docker.internal;18000;Initial Catalog=NewsletterDatabase;User ID=sa;Password=Pass12345;MultipleActiveResultSets=True;Encrypt=True;TrustServerCertificate=True;Connection Timeout=30;Persist Security Info=False;
      - MAIL_SENDER_URL=http://172.21.0.4:18000
    volumes:
      - $(HOME)/.aspnet/https:/https/

  mail-sender:
    image: mail-sender
    container_name: mail-sender
    build:
      context: .
      dockerfile: ./NewsletterApp/MailSender/Dockerfile
    ports:
      - 13000:80
      - 13001:443
    environment:
      - ASPNETCORE_URLS=https://+:http://+
      - ASPNETCORE_HTTPS_PORT=13001
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_Kestrel__Certificates__Default__Password=p455word!
      - ASPNETCORE_Kestrel__Certificates__Default__Path=https://newsletterapp/MailSender.pfx
      - CONNECTION_STRING=Server=host.docker.internal;18000;Initial Catalog=NewsletterDatabase;User ID=sa;Password=Pass12345;MultipleActiveResultSets=True;Encrypt=True;TrustServerCertificate=True;Connection Timeout=30;Persist Security Info=False;
      - EMAIL_FROM=adam.ludwiczak90@gmail.com
      - EMAIL_SMTP_SERVER=smtp.gmail.com
      - EMAIL_SMTP_PORT=465
      - EMAIL_USERNAME=adam.ludwiczak90@gmail.com
      - EMAIL_PASSWORD=*****
    volumes:
      - $(HOME)/.aspnet/https:/https/
```

Pobiera on obraz bazy danych oraz przygotowuje jej kontener wraz z utworzeniem konta dostępu do niej. Buduje i konfiguruje obie aplikacje na podstawie przygotowanych wcześniej plików Dockerfile.