Mariusz Wieczorek

mariusz.wieczorek@kabat.pl

Streszczenie

Aplikację webowa w ASP.NET MVC, która wysyła maile.  
Wymagania minimum: możliwość wpisania tytułu, treści wiadomości, adresu lub adresatów, nazwę nadawcy, która będzie wyświetlana adresatowi oraz przycisk „Wyślij”.  
 Po kliknięciu tego przycisku wiadomość e-mail zostanie wysłana.  
Dodatkowo: wysyłanie załączników, zapis wysłanych maili w bazie danych.

asp.net mvc

Zadanie 2: MailSender

**1. Tworzymy i aktualizujemy aplikację, aby uzyskać szkielet pustej aplikacji gotowej do dalszego rozwoju.**

Tworzymy aplikację **MailSender** korzystając z szablonu :   
**ASP.NET Web Application** (.NET Framework) z c#

Utworzenie w katalogu Model podkatalogów Domains i ViewModels, oraz przeniesienie plików zawierających modele widoków do ViewModels i wyodrębnienie klas do osobnych plików.

Aktualizujemy w wygenerowanej aplikacji pakiety NuGet.

Poprawiamy szablony w katalogu View\Shared\ \_Layout.cshtml i \_LoginPartial.cshtml tak, aby były zgodne z wersją 4 bootstrap’a.

Czyścimy z niepotrzebnego kodu plik Index.cshtml

W **Web.config** wpisujemy odpowiedni dla naszej bazy danych **connectionstring**.

|  |
| --- |
| **Web.config** |
| <connectionStrings>  <add name="DefaultConnection"  connectionString="Server=127.0.0.1;Database=MailSender;Uid=user1;Pwd=alamakota;"  providerName="System.Data.SqlClient"  />  </connectionStrings> |

Uwaga na klucz musi być taki sam jak w ApplicationDbContext.cs

Założenia

Tabela z emailami, każdy użytkownik widzi tylko swoje maile,  
Tabela z adresami, każdy użytkownik widzi tylko wysłane przez siebie adresy,  
  
ponadto przy zalogowanym użytkowniku zapamiętany jest jego e-mail i nazwa

Definiujemy Klasy Modeli

Klasa **Email**  
Jeden e-mail może być wystawiony przez jednego użytkownika  
Jeden e-mail może mieć wiele pozycji na liście odbiorców

|  |
| --- |
| public class **Email**  {  public **Email**()  {  EmailRecipients = new Collection<EmailRecipient>();  }  public int Id { get; set; }  [Required]  [Display(Name = "Temat")]  public string Subject { get; set; }  [Display(Name = "Data Utworzenia")]  public DateTime CreatedDate { get; set; }  [Display(Name = "Data Wysłania")]  public DateTime SentDate { get; set; }  [Display(Name = "Treść")]  public string Message { get; set; }    [Required]  [ForeignKey("User")]  public string UserId { get; set; }    public ApplicationUser User { get; set; }    public ICollection<EmailRecipient> EmailRecipients { get; set; }  } |

Klasa **EmailRecipient** – pozycja na liście odbiorców maila  
Jeden e-mail może mieć wiele pozycji na liście odbiorców, relacja podobna jak faktura->pozycje  
co zdefiniowano już w klasie **Email** .  
Jedna pozycja na liście odbiorców może należeć tylko do jednego maila.  
Właściwość EmailId i dodatkowo właściwość Email.  
Jedna pozycja na liście odbiorców może zawierać tylko jeden adres.  
Natomiast jeden adres może wystąpić w wielu pozycjach odbiorców.

|  |
| --- |
| public class EmailRecipient  {  public int Id { get; set; }  public int EmailId { get; set; }  [Display(Name = "Odbiorca")]  public int AddressId { get; set; }  public Email Email { get; set; }  public Address Address { get; set; }  } |

Klasa **Address** z adresem e-mail,  
Zalogowany użytkownik aplikacji widzi tylko wpisane przez siebie adresy, więc  
jeden adres należy tylko do jednego użytkownika – klucz obcy.  
Jeden adres może być na wielu listach odbiorców.  
Natomiast na jednej liście w pozycji odbiorców może być jeden adres  
co zdefiniowano w klasie **EmailRecipient**

|  |
| --- |
| public class Address  {  public Address()  {  EmailRecipients = new Collection<EmailRecipient>();  }  public int Id { get; set; }  [Required]  [Display(Name = "Nazwa")]  public string Name { get; set; }  [Required]  [Display(Name = "Adres")]  public string Email { get; set; }    [Required]  [ForeignKey("User")]  public string UserId { get; set; }  public ApplicationUser User { get; set; }  public ICollection<EmailRecipient> EmailRecipients { get; set; }  } |

Klasa **ApplicationUser**

Modyfikujemy klasę wygenerowaną przez szablon.  
Jeden użytkownik może wystawić wiele e-maili.  
Jeden email może być wystawiony przez jednego użytkownika – zdefiniowano w klasie **Email**  
Jeden użytkownik może mieć dostęp do wielu adresów e-mail.   
Jeden adres należy do jednego użytkownika - zdefiniowano w klasie **Address**

|  |
| --- |
| public class ApplicationUser : IdentityUser  {  public ApplicationUser()  {  Emails = new Collection<Email>();  Addresses = new Collection<Address>();  }  public ICollection<Email> Emails { get; set; }  public ICollection<Address> Addresses { get; set; }  [. . .]  } |

W klasie ApplicationDbContext musimy zdefiniować wszystkie tabele

|  |
| --- |
| public class ApplicationDbContext : IdentityDbContext<ApplicationUser>  {  public ApplicationDbContext()  : base("DefaultConnection", throwIfV1Schema: false)  {  }  public DbSet<Address> Addresses { get; set; }  public DbSet<Email> Emails { get; set; }  public DbSet<EmailRecipient> EmailRecipients { get; set; }    public static ApplicationDbContext Create()  {  return new ApplicationDbContext();  }  } |

enable-migrations

add-migration init

update-database ( pojawia się błąd związany z kaskadowym usuwaniem)

|  |
| --- |
| public class ApplicationDbContext : IdentityDbContext<ApplicationUser>  {  public ApplicationDbContext()  : base("DefaultConnection", throwIfV1Schema: false)  {  }  public DbSet<Address> Addresses { get; set; }  public DbSet<Email> Emails { get; set; }  public DbSet<EmailRecipient> EmailRecipients { get; set; }    public static ApplicationDbContext Create()  {  return new ApplicationDbContext();  } |
| protected override void OnModelCreating(DbModelBuilder modelBuilder)  {  // jak nie chcemy aby przy usunięciu usera  // zostały również usunięte przypisane do niego maile  modelBuilder.Entity<ApplicationUser>()  .HasMany(x => x.Emails)  .WithRequired(x => x.User)  .HasForeignKey(x => x.UserId)  .WillCascadeOnDelete(false);  // jak nie chcemy aby przy usunięciu usera  //zostali również usunięci przypisani do niego adresy e-mail  modelBuilder.Entity<ApplicationUser>()  .HasMany(x => x.Addresses)  .WithRequired(x => x.User)  .HasForeignKey(x => x.UserId)  .WillCascadeOnDelete(false);  base.OnModelCreating(modelBuilder);  } |
| } |

Usuwamy migracje, dodajemy ją ponownie i aktualizujemy bazę

Autoryzacja

Oznaczmy całą klasę HomeController jako [Authorize]

Oznaczamy Akcje About() i Contact() jako [AllowAnonymous].