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

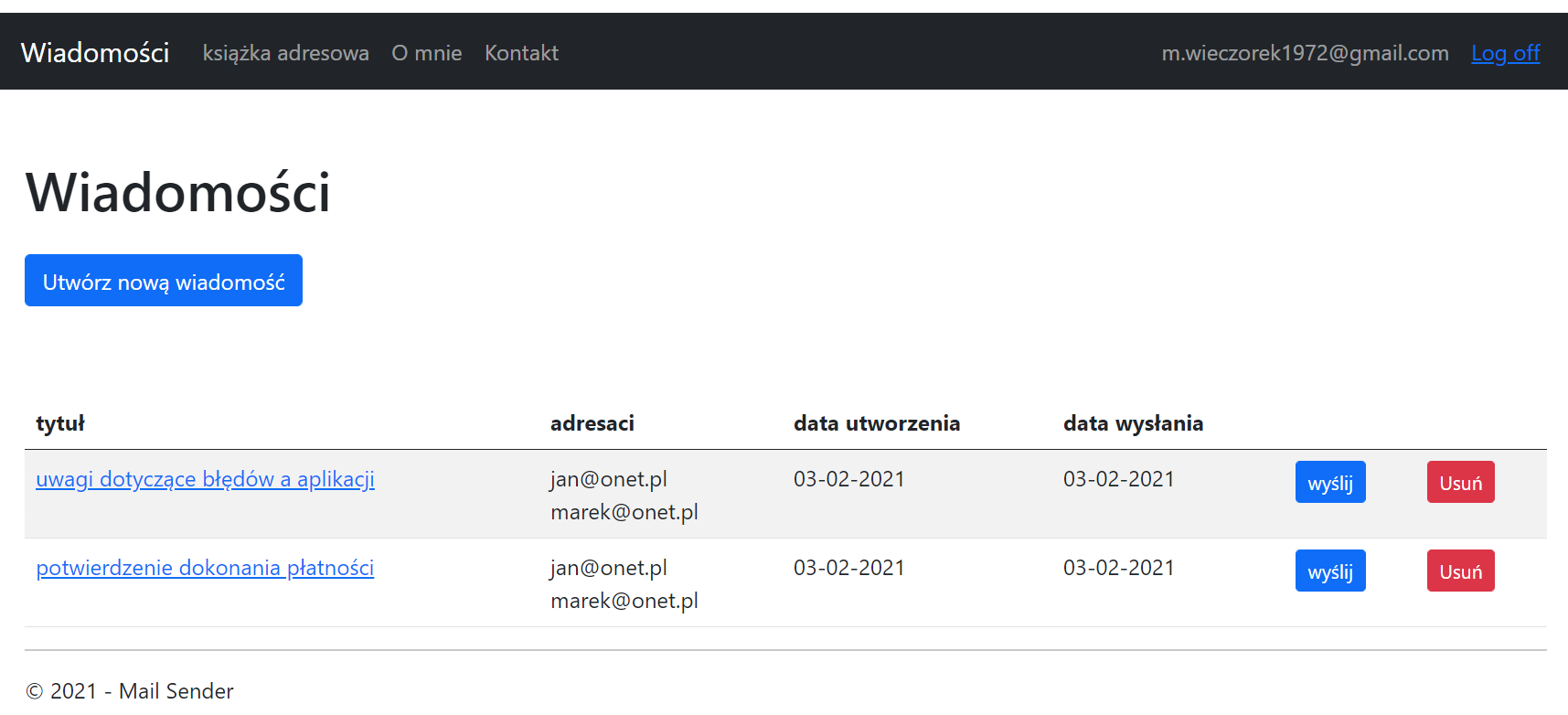
Oznaczmy całą klasę HomeController jako [Authorize]

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

**Tworzenie Widoków**

**Widok z listą wiadomości email**

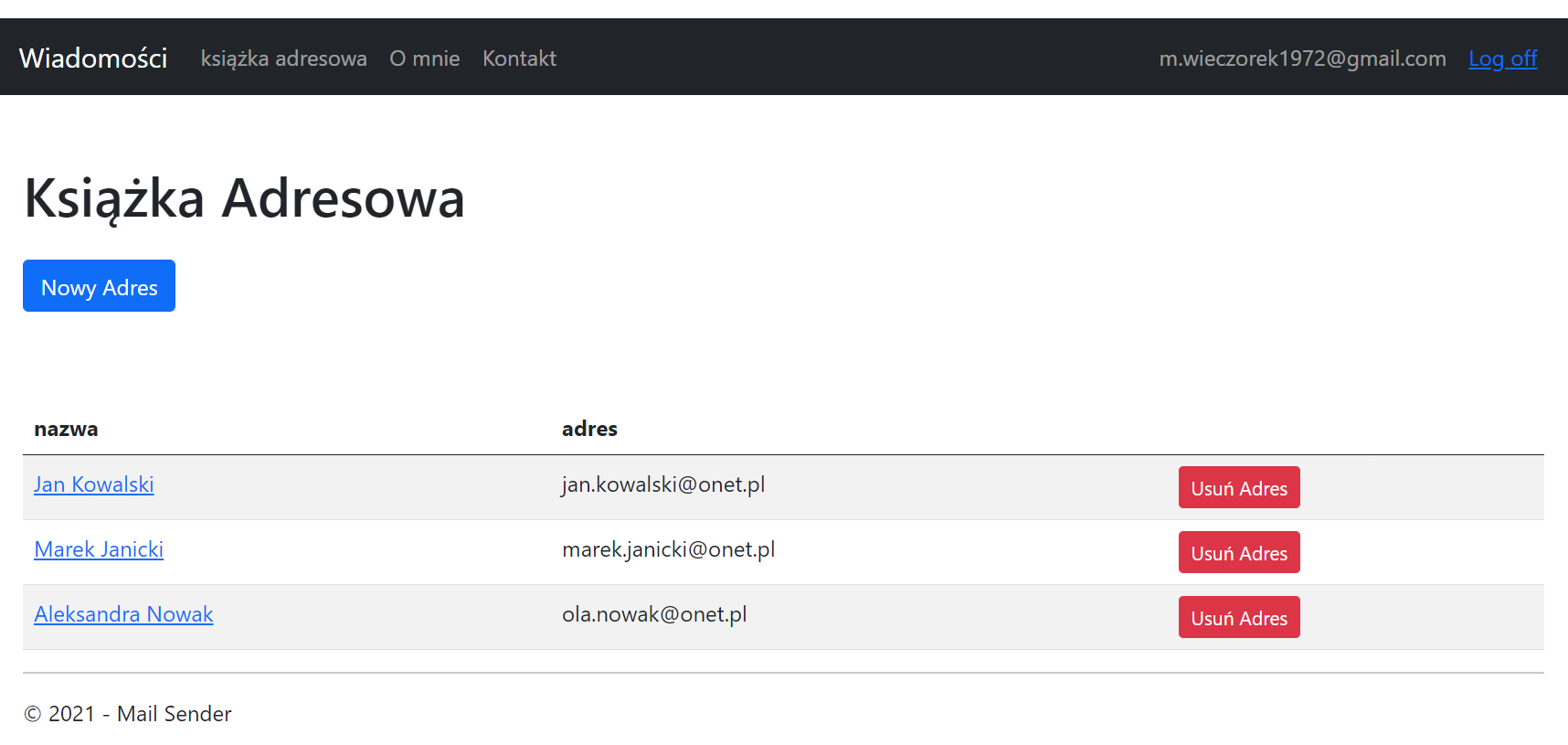
**Views\Home\Index.cshtml**



|  |
| --- |
| @model IEnumerable<MailSender.Models.Domains.Email>  @{  ViewBag.Title = "e-mail list";  }  <h1>Wiadomości</h1>  <button class="btn btn-primary mb-3 mt-3"  onclick='window.location.href = "@Url.Action("Email","Home")"'>  Utwórz nową wiadomość  </button>  @if (Model != null && Model.Any())  {  <table class="table table-hover table-striped mt-3 mt-5">  <thead>  <tr>  <th scope="col">tytuł</th>  <th scope="col">adresaci</th>  <th scope="col">data utworzenia</th>  <th scope="col">data wysłania </th>  <th scope="col"> </th>  <th scope="col"> </th>  </tr>  </thead>  <tbody>  @foreach (var email in Model)  {  <tr>  <td>  <a class="font-weight-bold"  href="@Url.Action("Email","Home", new { id = email.Id })">  @email.Subject  </a>  </td>  <td>  @foreach (var recipient in email.EmailRecipients)  {  <div>@recipient.Address.Email</div>  }  </td>  <td>@email.CreatedDate.ToString("dd-MM-yyyy")</td>  <td>@email.SentDate.ToString("dd-MM-yyyy")</td>  <td>  <button class="btn btn-primary btn-sm"  onclick="sendEmail('@email.Id',this)">  wyślij  </button>  </td>  <td>  <button class="btn btn-danger btn-sm"  onclick="deleteEmail('@email.Id',this)">  Usuń  </button>  </td>  </tr>  }  </tbody>  </table>  }  else  {  <div>  Brak poczty e-mail  </div>  }  @section scripts  {  <script type="text/javascript">  function deleteEmail() {  if (!confirm("Czy na pewno chcesz usunąć wiadomość ?"))  return;  }  function sendEmail() {  if (!confirm("Czy na pewno chcesz wysłać wiadomość ?"))  return;  }  </script>  } |

**Widok z listą adresów**

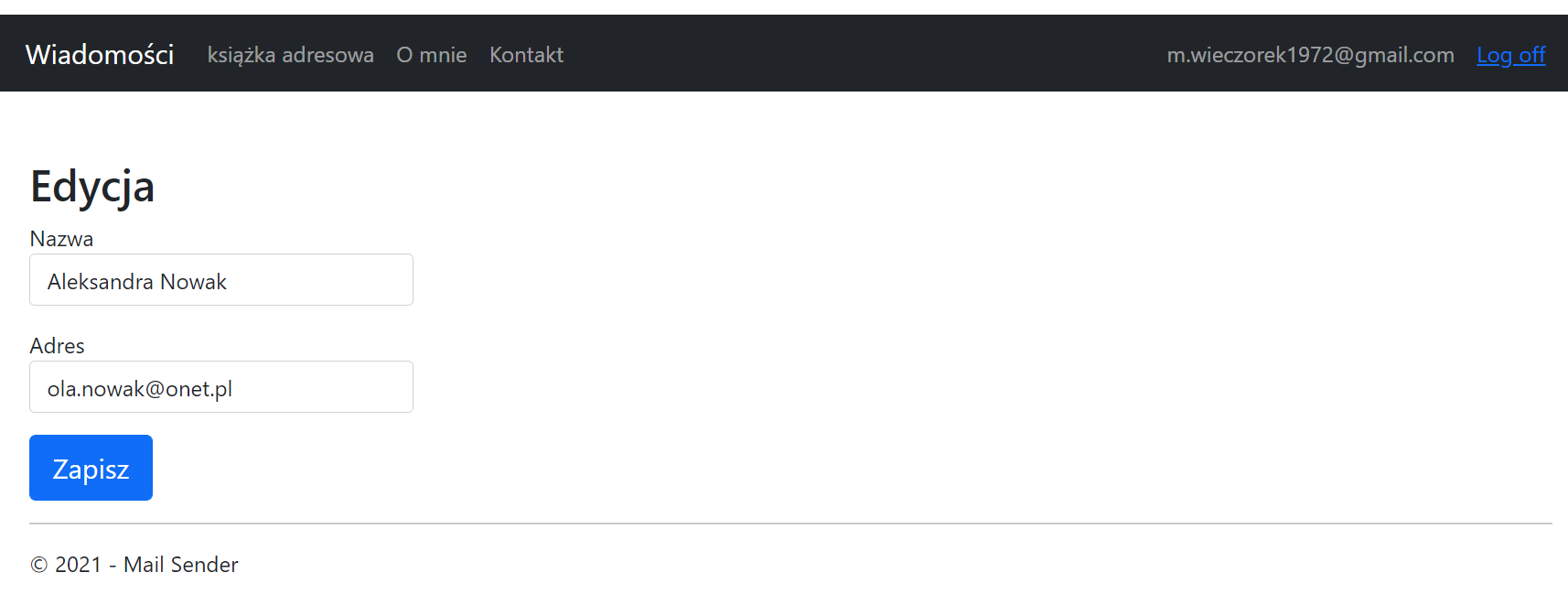
**Views\Home\Addresses.cshtml**



|  |
| --- |
| @model MailSender.Models.ViewModels.EditAddressViewModel  @{  ViewBag.Title = Model.Heading;  }  <h2>@Model.Heading</h2>  @using (Html.BeginForm("Address", "Home"))  {  @Html.AntiForgeryToken()  if (Model.Address.Id == 0)  {  <p class="alert alert-info mb-4 mt-4">  Pola: Lp, Produkt, Ilość i Wartość są  <strong>wymagane</strong>  </p>  }  @Html.HiddenFor(x => x.Address.Id)  <div class="form-group mb-3">  @Html.LabelFor(x => x.Address.Name)  @Html.TextBoxFor(x => x.Address.Name,  new { @class = "form-control" })  @Html.ValidationMessageFor(x => x.Address.Name)  </div>  <div class="form-group mb-3">  @Html.LabelFor(x => x.Address.Email)  @Html.TextBoxFor(x => x.Address.Email,  new { @class = "form-control" })  @Html.ValidationMessageFor(x => x.Address.Email)  </div>  <button class="btn btn-primary btn-lg" type="submit">  Zapisz  </button>  } |

**Widok dodawania/edycji adresu e-mail**

**Views\Home\Address.cshtml**



|  |
| --- |
| @model MailSender.Models.ViewModels.EditAddressViewModel  @{  ViewBag.Title = Model.Heading;  }  <h2>@Model.Heading</h2>  @using (Html.BeginForm("Address", "Home"))  {  @Html.AntiForgeryToken()  if (Model.Address.Id == 0)  {  <p class="alert alert-info mb-4 mt-4">  Pola: Lp, Produkt, Ilość i Wartość są  <strong>wymagane</strong>  </p>  }  @Html.HiddenFor(x => x.Address.Id)  <div class="form-group mb-3">  @Html.LabelFor(x => x.Address.Name)  @Html.TextBoxFor(x => x.Address.Name,  new { @class = "form-control" })  @Html.ValidationMessageFor(x => x.Address.Name)  </div>  <div class="form-group mb-3">  @Html.LabelFor(x => x.Address.Email)  @Html.TextBoxFor(x => x.Address.Email,  new { @class = "form-control" })  @Html.ValidationMessageFor(x => x.Address.Email)  </div>  <button class="btn btn-primary btn-lg" type="submit">  Zapisz  </button>  } |

**Modele Widoku**

Do widoku w którym będziemy edytować i dodawać nowy adres musimy przekazać  
Obiekt klasy **Address** , oraz nagłówek z informacją czy dodajemy, czy też edytujemy adres.

|  |
| --- |
| namespace MailSender.Models.ViewModels  {  public class EditAddressViewModel  {  public Address Address { get; set; }  public string Heading { get; set; }  }  } |

Do widoku, w którym będziemy edytować lub dodawać nowego maila,  
Będziemy przekazywać listę obiektów **List<Address>,** która będzie naszą książką adresową,  
czyli listę z adresami e-mail dostępnymi dla zalogowanego użytkownika, tymi które sam dodał.  
Będziemy też przekazywać obiekt typu **Email**, który w zależności od tego, czy dodajemy e-maila, czy też go edytujemy, będzie zawierał rekord odczytany z bazy lub pusty obiekt.  
Oraz string **Heading** z informacją czy edytujemy, czy też dodajemy nową wiadomość.

|  |
| --- |
| namespace MailSender.Models.ViewModels  {  public class EditEmailViewModel  {  public Email Email { get; set; }  public List<Address> Addresses { get; set; }  public string Heading { get; set; }  }  } |

Do widoku w którym będziemy dodawać/edytować odbiorcę maila

Będziemy przekazywać listę obiektów **List<Address>,** która będzie naszą książką adresową

|  |
| --- |
| namespace MailSender.Models.ViewModels  {  public class EmailRecipientViewModel  {  public EmailRecipient EmailRecipient { get; set; }  public List<Address> Addresses { get; set; }  public string Heading { get; set; }  }  } |

**Kontroler**

|  |
| --- |
| namespace MailSender.Controllers  {  [Authorize]  public class HomeController : Controller  {  private FakeRepositories \_fakeRepositories = new FakeRepositories();  public ActionResult Index()  {  var userId = User.Identity.GetUserId();  var emails = \_fakeRepositories.GetFakeEmails(userId);  return View(emails);  }  public ActionResult Email(int id = 0)  {  var userId = User.Identity.GetUserId();  var email = id == 0 ?  \_fakeRepositories.GetNewEmail(userId) :  \_fakeRepositories.GetFakeEmail(userId,id);      var editEmail = PrepareEmailVm(email,userId);  return View(editEmail);  }  public ActionResult Address(int id = 0)  {  var userId = User.Identity.GetUserId();  var address = id == 0 ?  \_fakeRepositories.GetNewAddress(userId) :  \_fakeRepositories.GetFakeAddress(userId,id);  var editAddress = PrepareEditAddressVm(address, userId);  return View(editAddress);  }  public ActionResult Addresses()  {  var userId = User.Identity.GetUserId();  var addresses = \_fakeRepositories.GetFakeAddresses(userId);  return View(addresses);  }  }  } |

W kontrolerze używamy testowego repozytorium, które używamy go do utworzenia Modelu Widoku, który następnie jest przekazany do widoku. Model Widoku tworzą **metody pomocnicze**

|  |
| --- |
| private EditAddressViewModel PrepareEditAddressVm(Address address, string userId)  {  return new EditAddressViewModel  {  Address = address,  Heading = address.Id == 0 ? "Dodawanie Nowego Adresu" : "Edycja",  };  }  private EditEmailViewModel PrepareEmailVm(Email email, string userId)  {  return new EditEmailViewModel  {  Email = email,  Heading = email.Id == 0 ? "Nowa Wiadomość" : "Edycja Wiadomości",  Addresses = \_fakeRepositories.GetFakeAddresses(userId)  };  } |

**Testowe Repozytorium**

|  |
| --- |
| using MailSender.Models.Domains;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  namespace MailSender.Models.Repositories  {  public class FakeRepositories  {  public List<Address> GetFakeAddresses(string userId)  {  return new List<Address>  {  new Address{Id = 1, Name="Jan Kowalski",Email="jan.kowalski@onet.pl" },  new Address{Id = 2, Name="Marek Janicki",Email="marek.janicki@onet.pl"},  new Address{Id = 3, Name="Aleksandra Nowak",Email="ola.nowak@onet.pl" },  new Address{Id = 4, Name="Marek Nowak",Email="marek.nowak@gmial.com" },  new Address{Id = 5, Name="Wioletta Krysiak",Email="wiola.k@gmial.com" },  };  }  // Testowa lista wiadomości Email  public List<Email> GetFakeEmails(string userId)  {  var addreses = GetFakeAddresses("1");  return new List<Email>  {  new Email  {  Id = 1,  Subject = "uwagi dotyczące błędów a aplikacji",  CreatedDate = DateTime.Now,  SentDate = DateTime.Now,  EmailRecipients = new List<EmailRecipient>  {    new EmailRecipient {Address = addreses.First(x => x.Id == 1) },  new EmailRecipient {Address = addreses.First(x => x.Id == 2) },  }  },  new Email  {  Id = 2,  Subject = "potwierdzenie dokonania płatności",  CreatedDate = DateTime.Now,  SentDate = DateTime.Now,  EmailRecipients = new List<EmailRecipient>  {  new EmailRecipient {Address = addreses.First(x => x.Id == 4) },  new EmailRecipient {Address = addreses.First(x => x.Id == 2) },  new EmailRecipient {Address = addreses.First(x => x.Id == 1) },  }  },  new Email  {  Id = 3,  Subject = "potwierdzenie wysłania zamówienia",  CreatedDate = DateTime.Now,  SentDate = DateTime.Now,  EmailRecipients = new List<EmailRecipient>  {  new EmailRecipient {Address = addreses.First(x => x.Id == 5) },  new EmailRecipient {Address = addreses.First(x => x.Id == 4) },  }  }  };  }  }  } |