Mariusz Wieczorek

[Adres e-mail]

Streszczenie

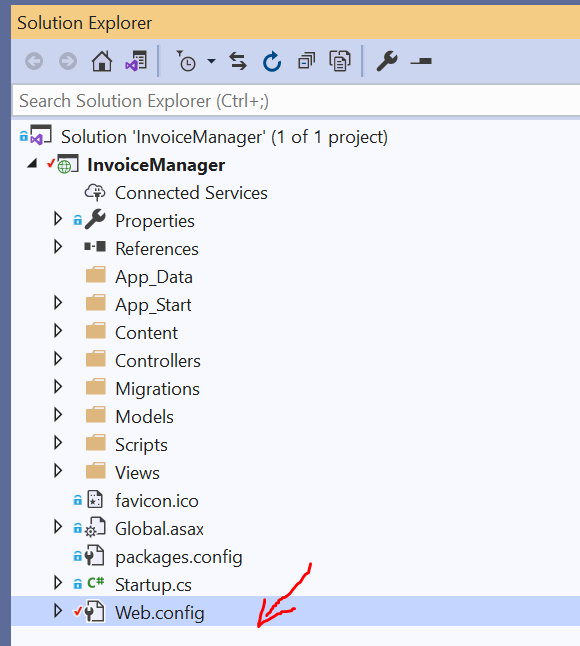
Web.config – connectionstring

ASP.NET MVC

Tworzenie bazy danych

**Tworzenie Bazy Danych**

Zaczynamy od modyfikacji domyślnego connectionstring’a.  
Connectionstring definiujemy w pliku Web.config



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

Parametr **name** musi być taki sam jak zdefiniowany w klasie ApplicationDbContext.cs

|  |
| --- |
| public class ApplicationDbContext : IdentityDbContext<ApplicationUser>  {  public ApplicationDbContext()  : base("DefaultConnection", throwIfV1Schema: false)  {  } |

**Model**

**Konfiguracja tabel za pomocą DataAnnotations.**

Aby skonfigurować tabele w przypadku aplikacji **WPF** wykorzystywaliśmy **fluent API** i tworzyliśmy oddzielne dedykowane klasy dla każdej z tabel np. InvoiceConfiguration.cs.

W przypadku aplikacji **ASP.NET MVC,** nie konfigurujemy tabel za pomocą **FluentAPI**, ale za pomocą adnotacji **DataAnnotations** , czyli będziemy dodawać atrybuty dla pól, które chcemy dodatkowo skonfigurować.

Aby działały adnotacje musimy załączyć przestrzeń nazw:

|  |
| --- |
| using System.ComponentModel.DataAnnotations;  using System.ComponentModel.DataAnnotations.Schema; |

Przykładowe adnotacje

|  |  |
| --- | --- |
| [Display(Name = "Klient")] | Wyświetlana nazwa |
| [Required] | Pole wymagane |
| [Required(ErrorMessage ="Pole Tytuł jest wymagane !!!")] | Pole wymagane, w przypadku nie wypełnienia komunikat określony jako ErrorMessage |
| [ForeignKey("User")] | Klucz obcy |

**Tworzenie klas modeli**

Klasa **Invoice** reprezentuje nagłówek faktury.  
Jedna faktura może zawierać jedną metodę płatności, klucz obcy **MethodOfPaymentId**.  
Natomiast jedna metoda płatności może występować na wielu fakturach   
co zdefiniowano w klasie **MethodOfPayment**.

Jedna faktura może zostać wystawiona przez jednego użytkownika klucz obcy **UserId**.  
Jeden użytkownik natomiast może wystawić wiele faktur, co zdefiniowano  
w klasie ApplicationUser.

|  |
| --- |
| **UserId** musi być typu string, bo takiego typu jest id użytkownika w wygenerowanej klasie ApplicationUser. Z tego też powodu musi ta właściwość mieć atrybut **[ForeignKey("User")]**, gdyby pole to było typu int nie musiało by mieć tego atrybutu. |

Jedna faktura może mieć jednego klienta klucz obcy **ClientId**.  
Jeden klient może natomiast wystąpić na wielu fakturach co zdefiniowano w klasie **Client**.

Pozycja faktury może należeć tylko do jednego nagłówka faktury,  
co zdefiniowano w klasie InvoicePossition.  
Natomiast jedna faktura może zawierać wiele pozycji.

|  |
| --- |
| public class Invoice  {  public Invoice()  {  InvoicePositions = new Collection<InvoicePossition>();  }  public int Id { get; set; }  [Required(ErrorMessage ="Pole Tytuł jest wymagane !!!")]  [Display(Name = "Tytuł faktury")]  public string Title { get; set; }  [Display(Name = "Wartość")]  public decimal Value { get; set; }  [Display(Name = "Sposób płatności")]  public int MethodOfPaymentId { get; set; }  [Display(Name = "Termin Płatności")]  public DateTime PaymentDate { get; set; }  [Display(Name = "Data Wystawienia")]  public DateTime CreatedDate { get; set; }  [Display(Name = "Uwagi")]  public string Comments { get; set; }  [Display(Name = "Klient")]  public int ClientId { get; set; }  [Required]  [ForeignKey("User")]  public string UserId { get; set; }  [Display(Name = "Sposób płatności")]  public MethodOfPayment MethodOfPayment { get; set; }  public Client Client { get; set; }  public ApplicationUser User { get; set; }  public ICollection<InvoicePossition> InvoicePositions { get; set; }  } |

|  |
| --- |
| Przy UserId musimy pokazać, że jest to klucz obcy, ponieważ jest to pole typu string Pól „nienulowalnych” typu **int** nie musimy oznaczać jako wymagane, nie musimy też ich oznaczać jako kluczy obcych . |

Klasa **MethodOfPayment** reprezentuje sposoby płatności

Jeden typ płatności może wystąpić na wielu fakturach.  
Jedna faktura może zawierać tylko jeden sposób płatności zdefiniowano w klasie Invoice.

|  |
| --- |
| public class MethodOfPayment  {  public MethodOfPayment()  {  Invoices = new Collection<Invoice>();  }  public int Id { get; set; }  [Required]  [Display(Name = "Klient")]  public string Name { get; set; }  public ICollection<Invoice> Invoices { get; set; }  } |

|  |
| --- |
| **Relacje jeden do wielu**  Zawsze jak definiujemy relacje jeden do wielu, powinniśmy to robić tak: Jeżeli na fakturze **Invoice** mamy klucz obcy **MethodOfPaymentId** (id sposobu płatności) to w klasie z metodami płatności **MethodOfPayment** powinniśmy zdefiniować listę faktur, na których ta metoda płatności występuje.  Wykorzystujemy do tego najprostszą implementację kolekcji **ICollection<T>**.  W przypadku faktur tworzymy właściwość: public ICollection<Invoice> Invoices { get; set; }  A w konstruktorze inicjujemy tą właściwość Invoices = new Collection<Invoice>(); |

Klasa **ApplicationUser** reprezentuje użytkownika aplikacji.

Jedna faktura może należeć do jednego użytkownika co zdefiniowano w klasie Invoice   
Jeden użytkownik może wystawić wiele faktur.  
  
Jeden klient może należeć do jednego użytkownika co zdefiniowano w klasie Client.  
Użytkownik może mieć przypisanych wielu klientów.

Jeden użytkownik może mieć jeden adres.

|  |
| --- |
| public class ApplicationUser : IdentityUser  {  public ApplicationUser()  {  Invoices = new Collection<Invoice>();  Clients = new Collection<Client>();  }  [Required]  public string Name { get; set; }  public int AddressId { get; set; }  public Address Address { get; set; }  public ICollection<Invoice> Invoices { get; set; }  public ICollection<Client> Clients { get; set; }  } |

Klasa Address reprezentuje wszelakie adresy zarówno użytkowników aplikacji jak i klientów

Jeden klient ma jeden adres co zdefiniowano w klasie **Client**,  
Natomiast jeden adres może mieć wielu klientów, np. mieszkających w tym samym domu.

Jeden użytkownik może mieć jeden adres co zdefiniowano w klasie **ApplicationUser**,  
Natomiast jeden adres może mieć wielu użytkowników.

|  |
| --- |
| public class Address  {  public Address()  {  Clients = new Collection<Client>();  ApplicationUsers = new Collection<ApplicationUser>();  }  public int Id { get; set; }  [Required]  [Display(Name = "Ulica")]  public string Street { get; set; }  [Required]  [Display(Name = "Numer Domu")]  public string Number { get; set; }  [Required]  [Display(Name = "Miasto")]  public string City { get; set; }  [Required]  [Display(Name = "Kod Pocztowy")]  public string PostalCode { get; set; }  public ICollection<Client> Clients { get; set; }  public ICollection<ApplicationUser> ApplicationUsers { get; set; }  } |

Klasa **Client** reprezentuje klienta na fakturze

Jeden klient może wystąpić na wielu fakturach.  
Na jednej fakturze może być tylko jeden klient co zdefiniowano w klasie **Invoice**.

Klient może mieć jeden adres.  
Jeden adres może należeć do wielu klientów co zdefiniowano w klasie **Address**.

Klient może być założony przez jednego użytkownika.  
Jeden użytkownik może założyć wielu klientów co zdefiniowano w klasie **ApplicationUser**.

|  |
| --- |
| public class Client  {  public Client()  {  Invoices = new Collection<Invoice>();  }    public int Id { get; set; }  [Required]  public string Name { get; set; }  public string Email { get; set; }  public int AddressId { get; set; }  [Required]  [ForeignKey("User")]  public string UserId { get; set; }  public Address Address { get; set; }  public ApplicationUser User { get; set; }  public ICollection<Invoice> Invoices { get; set; }  } |

|  |
| --- |
| **Uwaga**  Klient ma właściwość adres, nie będziemy trzymać całego adresu, ale klucz obcy do tabeli adres. pomocniczo definiujemy też klasę adres. |

Klasa **InvoicePossition** reprezentuje pojedynczą pozycję faktury.

Jedna faktura może zawierać wiele pozycji co zdefiniowano w klasie Invoice   
Jedna pozycja faktury może należeć do jednej faktury .

Jedna pozycja faktury może zawierać jeden produkt.  
Jeden produkt może wystąpić we wielu pozycjach faktury co zdefiniowano w klasie .

|  |
| --- |
| public class InvoicePossition  {  public int Id { get; set; }  public int Lp { get; set; }  public int InvoiceId { get; set; }  [Display(Name = "Wartość")]  public decimal Value { get; set; }  [Display(Name = "Produkt")]  public int ProductId { get; set; }  [Display(Name = "Ilosć")]  public int Quantity { get; set; }  public Invoice Invoice { get; set; }  public Product Product { get; set; }  } |

Klasa **Product**.

Jeden produkt może wystąpić na wielu pozycjach.  
Jedna pozycja może tylko jeden produkt co zdefiniowano w klasie **InvoicePossition**.

|  |
| --- |
| public class Product  {  public Product()  {  InvoicePossitions = new Collection<InvoicePossition>();  }  public int Id { get; set; }  [Required]  public string Name { get; set; }  public decimal Value { get; set; }  public ICollection<InvoicePossition> InvoicePossitions { get; set; }  } |

W klasie ApplicationDbContext musimy zdefiniować wszystkie tabele

|  |
| --- |
| using System.Data.Entity;  using InvoiceManager.Models.Domains;  using Microsoft.AspNet.Identity.EntityFramework;  namespace InvoiceManager.Models  {  public class ApplicationDbContext : IdentityDbContext<ApplicationUser>  {  public ApplicationDbContext()  : base("DefaultConnection", throwIfV1Schema: false)  {  }  public DbSet<Address> Addresses { get; set; }  public DbSet<Client> Clients { get; set; }  public DbSet<Invoice> Invoices { get; set; }  public DbSet<InvoicePossition> InvoicePossitions { get; set; }  public DbSet<MethodOfPayment> MethodOfPayments { get; set; }  public DbSet<Product> Products { get; set; }  public static ApplicationDbContext Create()  {  return new ApplicationDbContext();  }  }  } |

W Package Manager Console

|  |
| --- |
| PM> **enable-migrations**  Checking if the context targets an existing database... |

Dodajemy migracje o nazwie init

|  |
| --- |
| PM> **add-migration init**  Scaffolding migration 'init'.  The Designer Code for this migration file includes a snapshot of your current Code First model. This snapshot is used to calculate the changes to your model when you scaffold the next migration. If you make additional changes to your model that you want to include in this migration, then you can re-scaffold it by running 'Add-Migration init' again.  PM> |

Wykonujemy migracje

|  |
| --- |
| PM> **update-database**  Specify the '-Verbose' flag to view the SQL statements being applied to the target database.  Applying explicit migrations: [202101061331228\_init].  Applying explicit migration: 202101061331228\_init.  … |

pojawia się błąd

EF działa tak, że gdy usuniemy użytkownika, usunięte zostaną też wszystkie rekordy, gdzie użyty jest dany użytkownik.

Aby temu zapobiec musimy w klasie ApplicationDbContext nadpisać klasę

|  |
| --- |
| using System.Data.Entity;  using InvoiceManager.Models.Domains;  using Microsoft.AspNet.Identity.EntityFramework;  namespace InvoiceManager.Models  {  public class ApplicationDbContext : IdentityDbContext<ApplicationUser>  {  public ApplicationDbContext()  : base("DefaultConnection", throwIfV1Schema: false)  {  }  public DbSet<Address> Addresses { get; set; }  public DbSet<Client> Clients { get; set; }  public DbSet<Invoice> Invoices { get; set; }  public DbSet<InvoicePossition> InvoicePossitions { get; set; }  public DbSet<MethodOfPayment> MethodOfPayments { get; set; }  public DbSet<Product> Products { 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 faktury  modelBuilder.Entity<ApplicationUser>()  .HasMany(x => x.Invoices)  .WithRequired(x => x.User)  .HasForeignKey(x => x.UserId)  .WillCascadeOnDelete(false);  // jak nie chcemy aby przy usunięciu usera  //zostali również usunięci przypisani do niego klienci  modelBuilder.Entity<ApplicationUser>()  .HasMany(x => x.Clients)  .WithRequired(x => x.User)  .HasForeignKey(x => x.UserId)  .WillCascadeOnDelete(false);  base.OnModelCreating(modelBuilder);  } |
| }  } |

ALTER TABLE wsp.userupraw

ADD constraint fk\_wsp\_userupraw\_wsp\_users

FOREIGN KEY(userid)

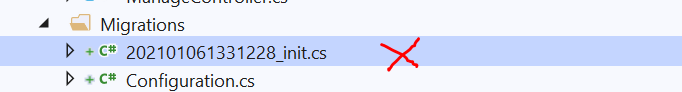
REFERENCES wsp.users(pozid)

ON DELETE CASCADE

ON UPDATE CASCADE ;

NO ACTION | CASCADE | SET NULL | SET DEFAULT

|  |
| --- |
| Uwaga  **Aby wykonać jeszcze raz migrację musimy usunąć starą:** |



Wykonujemy

|  |
| --- |
| PM> **add-migration init** |

I

|  |
| --- |
| PM> **update-database**  Specify the '-Verbose' flag to view the SQL statements being applied to the target database.  Applying explicit migrations: [202101061356087\_init].  Applying explicit migration: 202101061356087\_init.  Running Seed method.  PM> |