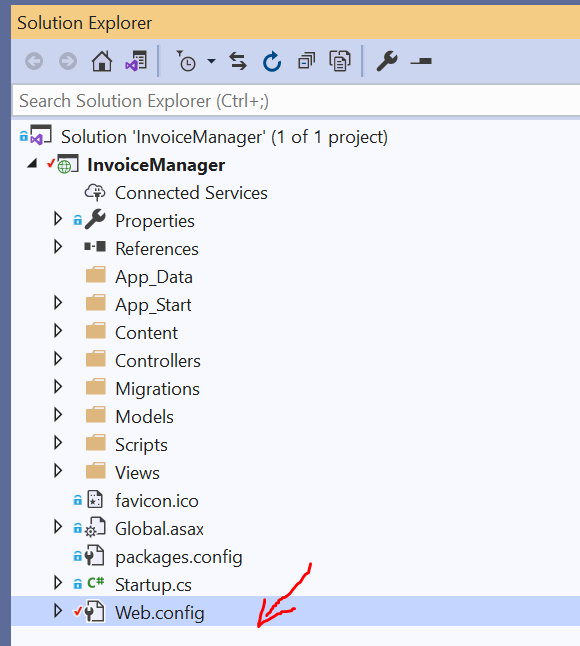
**Tworzenie Bazy Danych**

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)  {  } |

**Tworzenie Tabel**

Tabela z fakturami.  
UserId musi być typu string, bo takiego typu jest id użytkownika w wygenerowanym szablonie  
aplikacji.  
Tworzymy pola ClientID, UserId, MethodOdPaymentId, oraz odpowiadające im pola  
Client, User, MethodOfPayment

|  |
| --- |
| namespace InvoiceManager.Models.Domains  {  public class Invoice  {  public int Id { get; set; }  public string Title { get; set; }  public decimal Value { get; set; }  public int MethodOfPaymentId { get; set; }  public DateTime PaymentDate { get; set; }  public DateTime CreatedDate { get; set; }  public string Comments { get; set; }  public int ClientId { get; set; }  public string UserId { get; set; }  public MethodOfPayment MethodOfPayment { get; set; }  public Client Client { get; set; }  public ApplicationUser User { get; set; }  }  } |

Aby skonfigurować tabele w przypadku aplikacji WPF wykorzystaliśmy fluent Api i tworzyliśmy oddzielne dedykowane klasy np. InvoiceConfiguration.cs.

W tym przypadku nie zrobimy tego za pomocą FluentApi, ale za pomocą DataAnnotations czyli będziemy dodawać atrybuty dla niektórych pól.

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

|  |
| --- |
| public class Invoice  {  public int Id { get; set; }  [Required]  public string Title { get; set; }  public decimal Value { get; set; }  public int MethodOfPaymentId { get; set; }  public DateTime PaymentDate { get; set; }  public DateTime CreatedDate { get; set; }  public string Comments { get; set; }  public int ClientId { get; set; }  [Required]  [ForeignKey("User")]  public string UserId { get; set; }  public MethodOfPayment MethodOfPayment { get; set; }  public Client Client { get; set; }  public ApplicationUser User { get; set; }  } |

Przy UserId musimy pokazać, że jest to klucz obcy, ponieważ jest to pole typu string  
Pola nienulowalne typu int nie musimy oznaczać jako wymagane.

|  |
| --- |
| namespace InvoiceManager.Models.Domains  {  public class MethodOfPayment  {  public MethodOfPayment()  {  Invoices = new Collection<Invoice>();  }  public int Id { get; set; }  [Required]  public string Name { get; set; }  public ICollection<Invoice> Invoices { get; set; }  }  } |

Relacje jeden do wielu  
Jeżeli w fakturze mamy klucze obce np. MethodOfPaymentId czy ClientId, to w tej drugiej klasie  
powinniśmy zdefiniować listę tych faktur.  
ICollection – to najprostsza implementacje kolekcji.

Podobnie robimy z klientami, jeden klient może na wielu fakturach więc w definicję klasy klienta wstawiamy listę faktur.

|  |
| --- |
| namespace InvoiceManager.Models.Domains  {  public class Client  {  public Client()  {  Invoices = new Collection<Invoice>();  }  public int Id { get; set; }  [Required]  public string Name { get; set; }  public ICollection<Invoice> Invoices { get; set; }  }  } |

Połączenia z adresami ( jeden do wielu ) Jeden klient może mieć jeden adres, ale pod tym samym adresem może mieszkać wielu klientów – mieszkających w tym samym mieszkaniu, domu.

W ApplicationDbContext musimy zdefiniować wszystkie tabele

|  |
| --- |
| 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

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

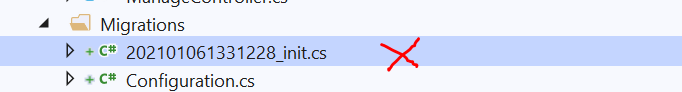
REFERENCES wsp.users(pozid)

ON DELETE CASCADE

ON UPDATE CASCADE ;

NO ACTION | CASCADE | SET NULL | SET DEFAULT

Aby wykonać jeszcze raz migrację usuwamy 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> |