dopolnitelno
microsoft sql server management studio da simnam?

blank solution EShopApplication
so desen klik na solution dodava nov folder so ime EShop
so desen klik na folderot kreira web aplikacija kojashto vsushnost kje ja kodirame -> add new project: asp .net core web app so ime EShop.Web i ja stava vo folderot kojshto go kreiravme (vo EShop folderot) bira Web App MVC - posle vo authentication bira Individual User accounts
koristi .net core 3.1
klika create

AppDbContext – ja povrzuva aplikacijata so bazata
Vo migrations se site tabeli koi ni trebaat

Home controller
Sekoj kontroler ima end points – akcii, gi primaat baranjata od korisnicite I im vrakjaat soodvetni odgovori!
Index, privacy , error akcii – sekoja akcija vrakja html!

Za sekoja akcija mora da ima soodvetno view!!! So isto ime.
Za da mozhe da se napishe return View();
I vo samiot Views folder ima podfolder za sekoj kontroler!! A vnatre vo toj podfolder se view-ata koi kje imaat isto ime so akciite vo kontrolerot na koj se odnesuva toj podfolder.

Sekogash koga imame nekoj migracija, taa migracija mora da se primeni vrz nekoja baza so koja rabotime.
Odi vo appsetting.json I gleda Connection Strings: I tamu ima nekoja lokalna baza na podatoci I go menuva parametarot
Database=ShopDb
So ova znaeme deka imame baza I treba samo migracijata da ja primenime vrz bazata.
Preku nugget packet manager console pishuva enable-migration , add-migration Initial (ova se dodava vo Data/Migrations)
I na kraj pravi update-database so cel da se primenat promenite vrz bazata na podatoci.

Kako da ja videme bazata – View/SQL Server Object Explorer

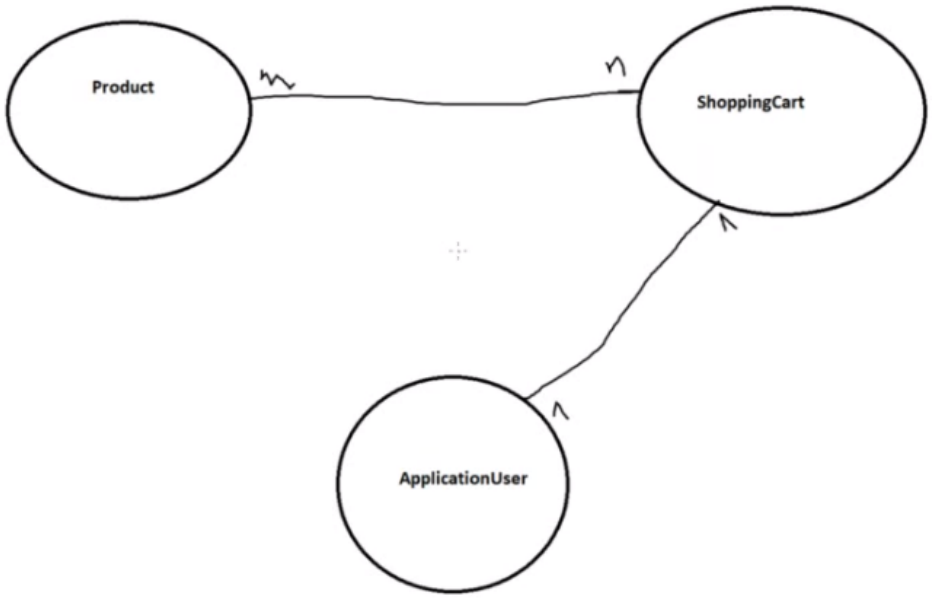
So refresh ja gleda napravenata baza

Program.cs prviot kod koj se izvrshuva pri start na app. A tuka vo main metodot povikuva da se izvrshat konfiguraciite od Startup.cs klasata – tuka ima razlichni konfiguracii – so koj db kontekst kje se povrzuvame so bazata, koj e konekciskiot string.
Pattern za rutiranje: controller-name/action-name/{id?}
Pr. Home/Index I ovaa akcija ne prima parametri I zatoa ne dodavame id vo url to

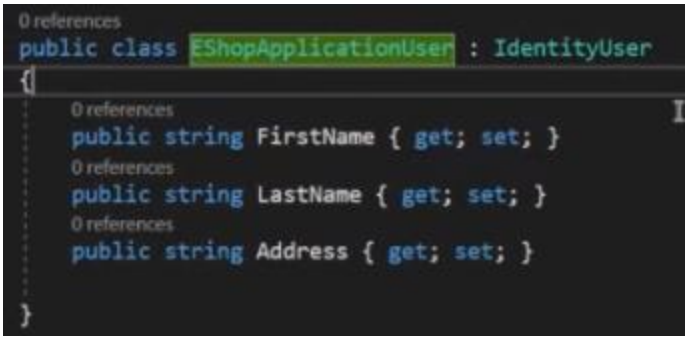
Moze da dodademe I nov pattern so endpoints.mapGet(..) I sl.

43min. dodavanje lokalna baza vo sql server management studio
** preku github ako prevzemame aud, togash samo so update-database

2 del
Prvo gi kodirame modelite, pa posle gi pravime tabelite vo bazata.
ER model:



Fluent API
Postavuvanje razlichni ogranichuvanja vrz svojstvata od modelite.
Vo Models dodava folder Domain I folder Identity vo koj kje se sodrzhi korisnikot.
Vo startup.cs vekje ima definirano deka klasata koja kje se koristi za User e IdentityUser – vekje e predefinirana ovaa klasa. Nie od ovaa klasa kje nasleduvame za da I dodademe dopolnitelni svojstva, vrski so drugi entiteti I sl.



I sledno vo startup.cs go menuva korisnikot takashto services.AddDefaultIdentity<EShopApplicationUser>
Treto, vo ApplicationDbContext se pravi promena takashto : IdentityDbContext<EShopApplicationUser> ova znachi deka sme ja preoptovarile IdentityUser klasata so nasha klasa!

Isto taka, bidejki go promenivme user-ot koj kje go koristi aplikacijata, ova mora da go zabelezhime I vo bazata! Znachi potrebno e migracija, pa update-database ???

Dodava novi domain modeli:
Product, ShoppingCart

Treba da se kreira M:N relacija megju Product I ShoppingCart – za ovaa cel e potrebna nova tabela koja kako PK kje gi ima FK na dvata entiteti koi vleguvaat vo relacijata.
Za ovaa cel, kreirame dopolnitelen Model klasa so ime ProductInShoppingCart I ovde gi ima dvata PK na entitetite koi vleguvaat vo relacijata.
Bidejki ova e klasa koja proizleguva od M:N relaciajta, nie gi dodavame I samite reference kon entitetite koi vleguvaat vo relacijata

Do tuka, ja kreiravme tabelata/modelot za M:N relacijata I sega treba da se dodade svojstvo vo dvete original tabeli koi vleguvaat vo ovaa relacija, takashto vo sekoja od tabelite Product I ShoppingCart kje dodademe svojstvo od tip ProductInShoppingCart. Ova svojstvo kje bide virtual I sekako ova kje bide lista!

Virtual – se odnesuva na lazy/eager loading za podatocite koi gi sodrzhi eden object. Primer: ako imame lazy loading za ShoppingCart koga kje povlecheme informacii za dadena SC nie kje go dobieme samo nejziniot id, ne I celata lista od proizvodi. A dokolku e eager, naednash kje gi dobieme site podatoci od koi e sostaven eden object vo kojshto se chuvaat I svojstva koi se odnesuvaat na relacija.
Virtual znachi lazy loading!

Sledno, konfiguracii so Fluent API: vo appDbContext.cs
Potrebno e da se preoptovari metodot OnModelCreating – bidejki vo ovoj metod se pravat konfiguraciite. Istite konfiguracii mozhe da se napravat I preku anotacii vrz svojstvata vnatre vo model klasite.
a)

```
builder.Entity<Product>().Property(z => z.ProductName).IsRequired();
```

b)

```
[Required]
1 reference
public string ProductName { get; set; }
```

Vo ovoj sluchaj go ostavame pod b, a vo onModelCreating gi konfigurirame samite relacii megju entitetite.

1. Generiranje na id na modelite – za sekoja klasa treba da definirame, avtomatski da se generiraat id-ata.

```
builder.Entity<Product>()
    .Property(z => z.Id)
    .ValueGeneratedOnAdd();
```

2. Konfiguracija na Many many relacijata/klasata ProductInShoppingCart – ovaa tabela ima kompoziten kluch od dvata FK. Za da go definirame ovoj PK, nie go konfigurirame I ova preku Fluent API:

```
//definiranje na kompoziten PK za klasata/modelot ProductInShoppingCart
builder.Entity<ProductInShoppingCart>()
    .HasKey(z => new { z.ProductId, z.ShoppingCartId });
```

Bidejki e slozhen kluchot, mora da go definirame preku anonimen object so lambda expression.

3. Dopolnitelno, sledno treba da se anotiraat I samite relacii, na primer listite koi gi ima vo Product I ShoppingCart, a koi se od tip ProductInShoppingCart mora da gi definirame deka vsushnost pretstavuvaat svojstva za relacii!
   Eden od nachinite e da ja definirame samata klasa-relacija ProductInShoppingCart, bidejki taa imashe po edno svojstvo koe pokazhuva kon Product I kon ShoppingCart, nie mozhe da gi definirame ovie svojstva kako .hasOne .withMany a ovde kje go navedeme imeto na svojstvoto vo samata tabela Product/ShoppinCart koe se odnesuva na listata od tip ProductInShoppingCart.

- Ova mozhe I na drug nachin da se konfigurira??

4. Sledno, da ja definirame One to One relacijata megju ShoppingCart I User

Vo EShoppApplicationUser dodava novo svojstvo od tip ShoppingCart I e virtual
Istoto go pravime I vo ShoppingCart -- *tuka ne e virtual??*

```
3 references
public class ShoppingCart
{
    1 reference
    public Guid Id { get; set; }
    1 reference
    public virtual ICollection<ProductInShoppingCart>
    0 references
    public EShopApplicationUser Owner { get; set; }
}
```

Nie odreduvame, vo ShoppingCart da dodademe FK od tip string OwnerId za da imame pokazhuvach na koj korisnik pripagja daden object od tip ShoppingCart.
(

One-to-one

> One to one relationships have a reference navigation property on both sides. They follow the same conventions as one-to-many relationships, but a unique index is introduced on the foreign key property to ensure only one dependent is related to each principal.

When configuring the relationship with the Fluent API, you use the `HasOne` and `WithOne` methods.

When configuring the foreign key you need to specify the dependent entity type - notice the generic parameter provided to `HasForeignKey` in the listing below. In a one-to-many relationship it is clear that the entity with the reference navigation is the dependent and the one with the collection is the principal. But this is not so in a one-to-one relationship - hence the need to explicitly define it.

Pr.

```
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .HasOne(b => b.BlogImage)
            .WithOne(i => i.Blog)
            .HasForeignKey<BlogImage>(b => b.BlogForeignKey);
    }


ublic class Blog

    public int BlogId { get; set; }
    public string Url { get; set; }

    public BlogImage BlogImage { get; set; }


ublic class BlogImage

    public int BlogImageId { get; set; }
    public byte[] Image { get; set; }
    public string Caption { get; set; }

    public int BlogForeignKey { get; set; }
    public Blog Blog { get; set; }
```

)

Konfiguracija na One-to-one vo Fluent API:

```
//konfiguriranje na one to one relacija megju SC i EshopUser
builder.Entity<ShoppingCart>()
    .HasOne(sc => sc.Owner)
    .WithOne(o => o.UserCart)
    .HasForeignKey<ShoppingCart>(sc => sc.OwnerId);
```

Ova e po konvencija, istoto e dadeno I vo dokumentacijata pogore!

Sledno, vo ApplicationDbContext sekogash mora da imame svojstva od tip modelite/tabelite koi sakame da se mapiraat vo bazata!
Tie se **virtual** I od tip **DbSet**<> a vnatre se naveduva konkretniot model//tabela koja sakame da se dodade vo bazata.

```
public virtual DbSet<Product> Products { get; set; }
0 references
public virtual DbSet<ShoppingCart> ShoppingCarts { get; set; }
0 references
public virtual DbSet<ProductInShoppingCart> ProductInShoppingCarts { get; set; }
```

Bidejki dodadovme novi modeli, pred se mora da se napravi **add-migration**
Add-migration mora za da se **kreiraat** tabelite koi gi registriravme vo appDbContext toa se vsushnost svojstvata od tip DBSet
No posle mora I update-database za da se zabelezhi toa vo samata baza.

Vsushnost svojstvata od tip DbSet<> se tabelite vo bazata koi svojstva nie direktno kje gi pristapuvame za da prebaruvame, zapishuvame I slichno!
Znachi go koristeme AppDbContext kako svojstvo preku DI vo nashite klasi (pr. Kontroler) I od toa svojstvo gi pristapuvame site tabeli koi ni trebaat za dadena logika!