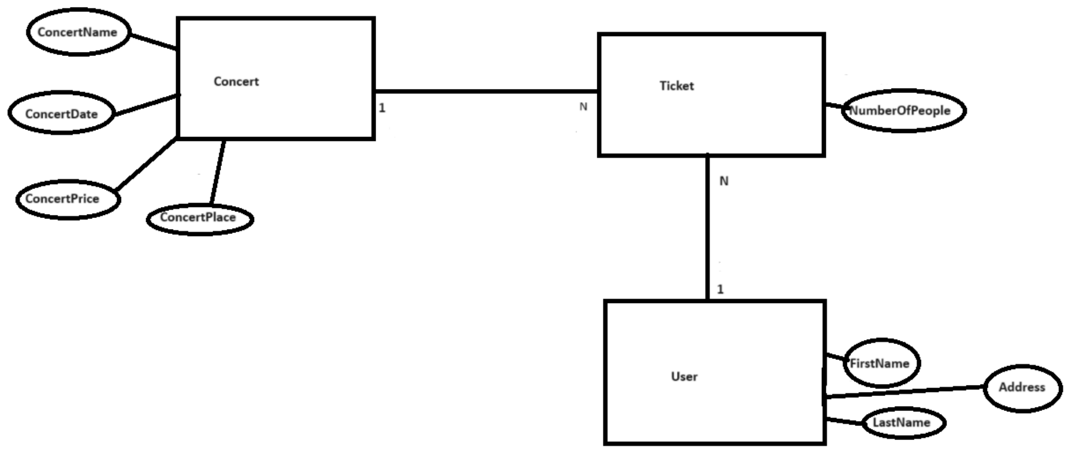
Лабораториска вежба 1 (Група А) / Laboratory exercise 1 (Group A)

Да се изработи веб апликација за купување на билети за Концерти со користење на Onion Architecture. Апликацијата треба да ги опфаќа следните карактеристики:

- Слоеви на Onion Architecture
- Класата User треба да наследува од IdentityUser (Напомена: Направете замена на IdentityUser со User во Program.cs, AppdbContext и во LoginPartial)
- Генерирање на модели во Web слојот според наведената шема
- Најава на корисник и регистрација на корисник
- 2 Контролери со scaffolding: ConcertController и ConcertTicketsController
- Сите CRUD операции (Додавање, Едитирање, Бришење) поврзани со Концерти и со Билетите.
- Преглед на сите достапни Концерти и Билети.
- Бонус: Концертите да се прикажуваат во форма на Grid Layout по 3 во ред.

Дизајнот на базата на податоци треба да изгледа вака/The database design should look like this::



REGISTER/LOGIN USER

1. Create Solution

2. Create a ApplicationUser class that will inherit from IdentityUser

```
public class ApplicationUser : IdentityUser
 {
public string FirstName { get; set; } public
    string LastName { get; set; } public
    string UserAddress { get; set; }
 }
```

3. Replace IdentityUser with ApplicationUser in Program.cs, AppdbContext and in LoginPartial

Program.cs:

```
builder.Services.AddDefaultIdentity<ApplicationUser>(options =>
 options.SignIn.RequireConfirmedAccount = true)
.AddEntityFrameworkStores<ApplicationDbContext>();
```

AppdbContext.cs

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
```

LoginPartial.cshtml

```
@inject SignInManager<ConcertShop.Models.Identity.ApplicationUser>
SignInManager
@inject UserManager<ConcertShop.Models.Identity.ApplicationUser>
 UserManager
```

4. Add Migration Through Nuget Packet Manager Console

`add-migration`	`update-database`

5. Create Scaffolding For Identity (On Scaffolding Select Login & Register)

6. On Register.cshtml.cs Add Input Models For Your Properties

```
public class InputModel
 {
[Required]
[Display(Name = "FirstName")]
public string FirstName { get; set; }

[Required]
[Display(Name = "LastName")]
public string LastName { get; set; }

[Required]
[Display(Name = "UserAddress")]
public string UserAddress { get; set; }

. . . Here are other Identity Default Input Models
```

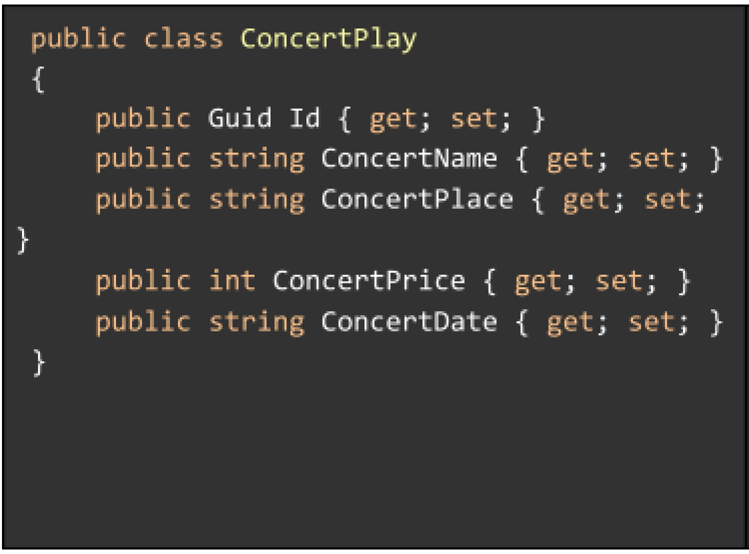
7. On same file below register your properties when a user is created

```
if (ModelState.IsValid)
 {
var user = CreateUser();

user.FirstName = Input.FirstName;
     user.LastName = Input.LastName;
     user.UserAddress = Input.UserAddress;

. . . Other Identity Defauls Stuff
```

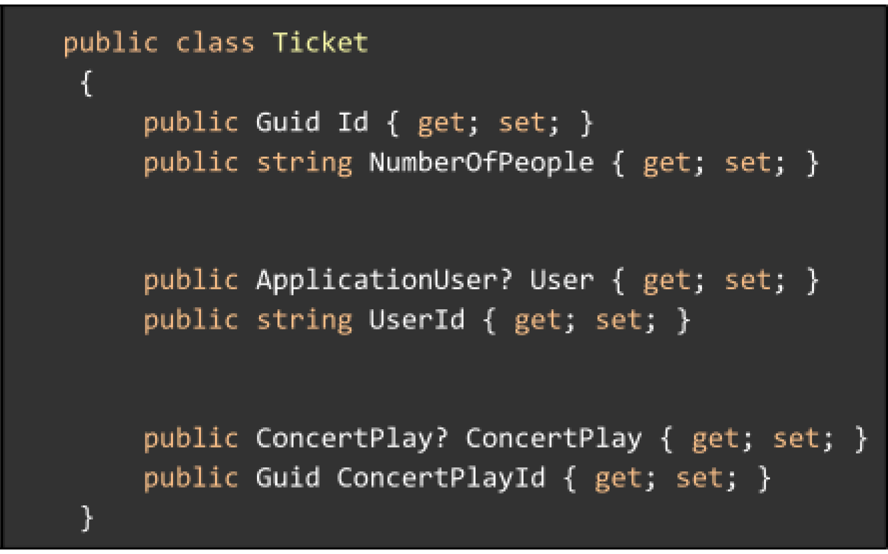
You now have User Registration and User Login

CRUD OPERATIONS (Add, Edit, Delete)

1. Create model classes



```
public class ConcertPlay
{
    public Guid Id { get; set; }
    public string ConcertName { get; set; }
    public string ConcertPlace { get; set;
}
    public int ConcertPrice { get; set; }
    public string ConcertDate { get; set; }
}
```

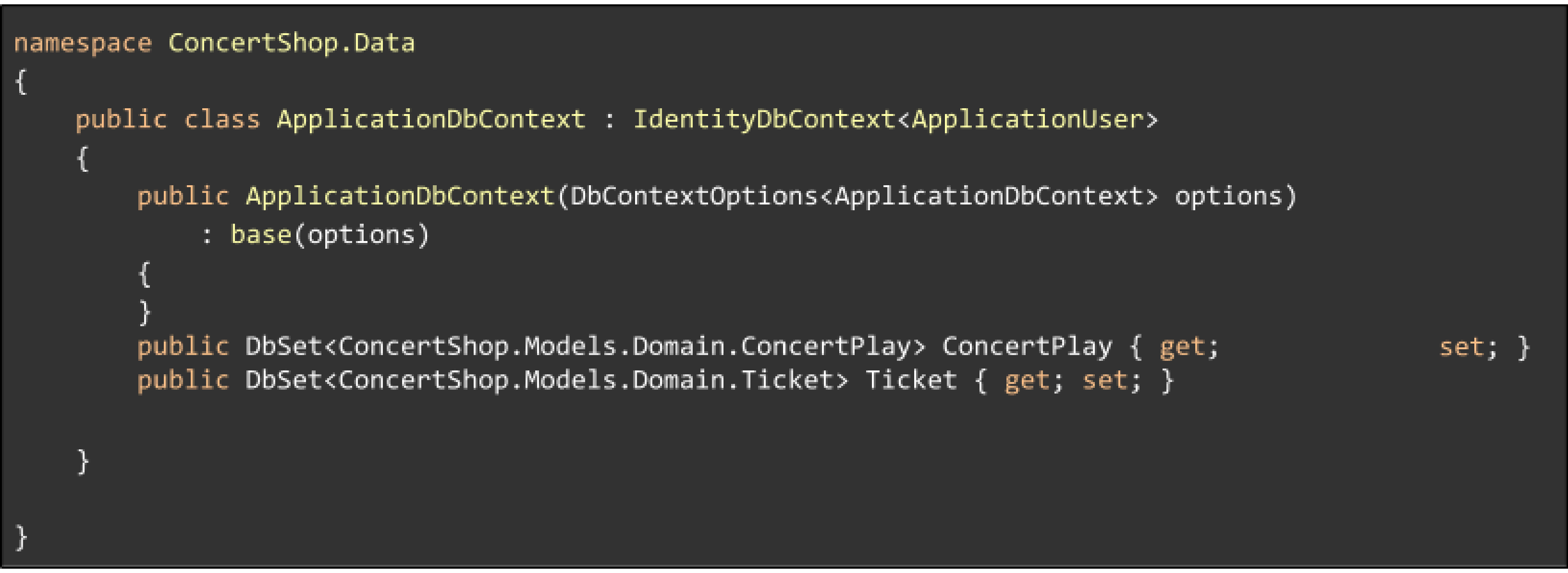


```
public class Ticket
 {
     public Guid Id { get; set; }
     public string NumberOfPeople { get; set; }


     public ApplicationUser? User { get; set; }
     public string UserId { get; set; }


     public ConcertPlay? ConcertPlay { get; set; }
     public Guid ConcertPlayId { get; set; }
 }
```

GUID (Globally Unique Identifier): A GUID is a 128-bit integer used as a unique identifier in software development. It ensures uniqueness across space and time, making it suitable for generating primary keys in databases.

Nullable Types (?): The ? indicates that the property can be assigned a null value. In this case, User and ConcertPlay properties can be null, meaning a ticket may not necessarily be associated with a user or a concert

2. Add The New Classes to ApplicationDbContext

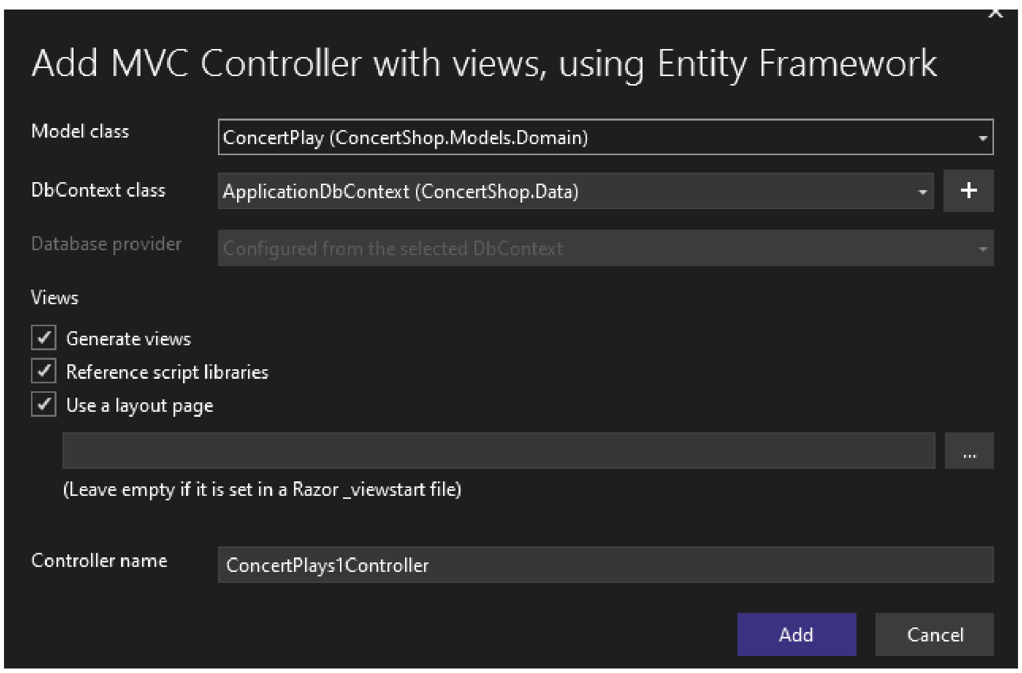


```
namespace ConcertShop.Data
{
    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }
        public DbSet<ConcertShop.Models.Domain.ConcertPlay> ConcertPlay { get;              set; }
        public DbSet<ConcertShop.Models.Domain.Ticket> Ticket { get; set; }

    }

}
```

-`Add Migration` & `Update Database`

Tip: If migration fails at any time and you are having difficulties fixing, find the Migrations folder and delete all migrations inside and try again, if it still fails than manually delete all the tables from SQL Server (VisualStudio VIew Tab-> SQL Server Object Explorer)
This makes sure you are creating fresh tables.

3. Create Controller For The New Classes We Made


Add MVC Controller with views, using Entity Framework

Model class: ConcertPlay (ConcertShop.Models.Domain)

DbContext class: ApplicationDbContext (ConcertShop.Data)

Database provider: Configured from the selected DbContext

Views

- [✓] Generate views
- [✓] Reference script libraries
- [✓] Use a layout page

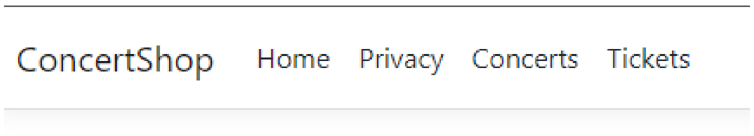
(Leave empty if it is set in a Razor _viewstart file)

Controller name: ConcertPlays1Controller

4. Add Buttons To Our Main Page - Views/Shared/Layouts -

```
<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="ConcertPlays"
 asp-action="Index">Concerts</a>
</li>

<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Tickets"
 asp-action="Index">Tickets</a>
</li>
```

All CRUD operations should work now.


ConcertShop
Home
Privacy
Concerts
Tickets