Sprint 3 Register and Authentication

Наследование: базовая модель и базовый контроллер

Создайте абстрактный класс `Base`, который будет служить целью базовой модели для всех моделей.

```
public abstract class Base
{
    public Guid Id { get; set; }
    public DateTime CreatedAt  { get; set; } = DateTime.UtcNow;
    public DateTime UpdatedAt { get; set;} = DateTime.UtcNow;
}
```

Теперь можно отнаследоваться от модели `Base`. Например, класс `User` будет выглядеть следующим образом:

```
public class User : Base
{
  // свойства
}
```

Entity Framework Core

Для реализации взаимодействия с реляционной базой данных мы будем использовать ORM - `Entity Framework Core`. Для этого надо установить пакеты относительно версии .net проектов следующим способом:

- пакет Microsoft.EntityFrameworkCore `dotnet add .\SportStore.API\ package Microsoft.EntityFrameworkCore -v 7.0.0`

для миграций

- `Microsoft.EntityFrameworkCore.Tools`
- `Microsoft.EntityFrameworkCore.Design`

провайдер для базы данных `PostgreSQL`

- `Npgsql.EntityFrameworkCore.PostgreSQL`

При установке пакетов надо соблюдать версионность относительно версии фреймворка .net В данном приложении применяется `net7.0`

Замечание: установить пакеты можно несколькими способами:

- dotnet cli
- установка графических пакетов
- Visual Studio
- через файл `csproj`

В результате добавления пакетов в `SportStore.API.csproj` будет выглядеть следующим образом:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net7.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="7.0.5" />
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="7.0.20" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design"
Version="7.0.20">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools"
Version="7.0.20">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.4.0" />
  </ItemGroup>

</Project>
```

Далее, создайте папку `Data` и добавьте класс `SportStoreContext`. Это класс будет конфигурировать отображение моделей данных на таблицы в базе данных.

```
public class SportStoreContext : DbContext
{
    public DbSet<User> Users { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {

optionsBuilder.UseNpgsql("Host=localhost;Port=5432;Database=SportStoreCourse;Usern
ame=;Password=");
```

```
    }
}
```

В параметрах подключения к базе данных поставьте свой `UserName` и `Password`

Миграции

Замечание. Перед работой с базой данных надо подготовить модели данных. Вместо типа `Guid` нам теперь нужно использовать тип `int` для свойства `Id`. Это требование EF. Внесите изменения в те места, где раньше использовался тип `Guid`: интерфейсы, репозитории, контроллеры, тесты. Команды `dotnet build` и `dotnet test` должны выполняться успешно.

В рабочей директории создайте первую миграцию.

```
dotnet ef migrations add Initial -s SportStore.API -p SportStore.API
```

Далее, выполните эту миграцию. То есть EF создаст реальные таблицы в базе данных PostgreSQL на основе классов моделей данных указанных в контексте базы данных `SportStoreContext`.

```
dotnet ef database update -s SportStore.API -p SportStore.API
```

Замечание: опция `-s` - это стартовый проект, опция `-p` - это текущий проект. Либо, можно зайти в проект `SportStore.API` явно и не прописывать данные параметры. Также при создании и выполнении миграции сборка решения должна происходить успешно.

Замечание: если база данных уже существует, то удалите базу данных `dotnet ef database drop -s SportStore.API -p SportStore.API`,а затем примените миграцию

Настройка хранения паролей

- добавьте в модель `User` новые свойства

```
    public string Login {get; set;}
    public required byte[] PasswordHash {get; set;}
    public required byte[] PasswordSalt {get; set;}
```

- теперь создайте новую миграцию по именем `AddLoginAndPasswordToUsers`
- примените миграцию

Создание репозитория для хранения пользователей в базе данных

- создайте в папке `Repositories` новый класс `UserRepository`,который будет реализовывать интерфейс `IUserRepository`, который мы определили в `sprint1`, но механизм хранения будет использовать базу данных `PostgreSQL`.

Обратите внимание на конструктор репозитория и зарегистрируйте в DI данный класс репозитория.

```
public class UserRepository : IUserRepository
{
    private readonly SportStoreContext _db;
    public UserRepository(SportStoreContext db)
    {
        _db = db;
    }

    public User CreateUser(User user)
    {
       try
       {
         _db.Add(user);
         _db.SaveChanges();
         return user;
       }
       catch(SqlTypeException ex)
       {
         throw new SqlTypeException($"Ошибка SQL: {ex.Message}");
       }
       catch (Exception ex)
       {
         throw new Exception($"Ошибка: {ex.Message}");
       }
    }

    // another methods interface
}
```

Задание: реализуйте методы интерфейса `IUserRepository`

Задание: после изменения модели `User` наши тесты теперь не проходят. Исправьте это, используя данный объект в классе `UserLocalRepositoryTests` для генерации пароля и его хэша:

```
public HMACSHA512 hmac = new HMACSHA512();
```

- для создания объекта пользователя:

```
var user = new User
{
    Name = "Test User",
    Login = "Login"
    PasswordHash = hmac.ComputeHash(Encoding.UTF8.GetBytes("Password")),
    PasswordSalt = hmac.Key
};
```

DTO

Создайте модель для передачи данных `UserDto` в папке `Dto`

```
public class UserDto
{
    public string Login { get; set; } = string.Empty;
    public string Password { get; set; } = string.Empty;
}
```

Замечание: Эквивалентом и краткой записью для класса со свойствами является `record`:

```
public record UserRecordDto {
  public required string Login {get; init;}
  public required string Password {get; init;}
};
```

Seed Data - генерация тестовых данных

- установите библиотеку `Bogus`
- создайте новый контроллер `SeedController` в котором реализуйте следующий метод для генерации пользователей:

```
    [HttpGet("generate")]
    public ActionResult SeedUsers(){

        using var hmac = new HMACSHA512();

        Faker<UserRecordDto> _faker = new Faker<UserRecordDto>("en")
            .RuleFor(u => u.Login, f => GenerateLogin(f).Trim())
            .RuleFor(u => u.Password, f => GeneratePassword(f).Trim().Replace("
","",""));

        string GenerateLogin(Faker faker)
        {
           return faker.Random.Word() + faker.Random.Number(3,5);
        }

        string GeneratePassword(Faker faker)
        {
           return faker.Random.Word() + faker.Random.Number(3, 5);
        }

        var users = _faker.Generate(100).Where(u => u.Login.Length > 4 &&
```

```
u.Login.Length <= 10);

        List<User> userToDb = new List<User>();

        try
        {
            foreach (var user in users)
            {
                var u = new User()
                {
                    Login = user.Login,
                    PasswordHash =
hmac.ComputeHash(Encoding.UTF8.GetBytes(user.Password)),
                    PasswordSalt = hmac.Key,
                };
                userToDb.Add(u);
            }
            _db.Users.AddRange(userToDb);
            _db.SaveChanges();
        }
        catch(Exception ex)
        {
           Console.WriteLine($"{ex.InnerException.Message}");
        }

        return Ok(userToDb);
    }
```

Регистрация реализации UserRepository

```
builder.Services.AddScoped<IUserRepository, UserRepository>();
```

EF Configuration (option)

Настройка каждого атрибута, связей, типов данных в конкретной базе данных

Reverse engineering (option)

Существует обратное проектирование - по готовой базе данных восстановить модели данных - скаффолдинг.

EF HasData (option)

В методе `OnModelCreating` контекста базы данных можно генерировать тестовые данные, а также применять пользовательские конфигурации.

```
        protected override void OnModelCreating(ModelBuilder builder){


builder.ApplyConfigurationsFromAssembly(typeof(SportStoreContext).Assembly);

            builder.Entity<User>().HasData(
                new User(){ Id = 1, Name = "user", ...}
            );

        }
```

SaveChanges (option)

Переопределение метода для обновление базовой модели

```
        public override Task<int> SaveChangesAsync(CancellationToken
cancellationToken){

            foreach(var entry in ChangeTracker.Entries<Base>())
            {
                switch(entry.State)
                {
                    case EntityState.Added:
                        entry.Entity.CreatedAt = DateTime.UtcNow;
                        break;
                    case EntityState.Modified:
                        entry.Entity.UpdatedAt = DateTime.UtcNow;
                        break;
                }
            }
            return base.SaveChangesAsync(cancellationToken);
        }
```

Задание 1: создайте базовый API контроллер.

Задание 2: можно создать реализацию репозитория пользователей, который будет работать с базой данных SQL Server. Для этого примените пакет `Microsoft.EntityFrameworkCore.SqlServer`.