

Sprint 1 Introduction

Общие настройки

- отключите расширения C# Dev Kit
- включите material icons

Проект API. Тип архитектуры: All-In

Реализовать базовую функциональность API

Использование dotnet CLI

- создайте папку SportStore и перейдите в нее в командной строке.
- посмотрите с помощью команды `dotnet new list` список доступных проектов и создайте проект `webapi` с именем `SportStore.API` командой:

```
dotnet new webapi -o SportStore.API
```

Замечание: в приложении будет использовать версия .net core 7.0, в версии 8.0 данная команда по умолчанию для работы с API использует `minimalAPI`, поэтому для включения контроллеров надо использовать флаг `--controllers`.

- добавьте файл решения, находясь в папке рабочей директории, командой `dotnet new sln`
- добавьте в решение проект API - `dotnet sln add SportStore.API`

Замечание: добавить все проекты в решение `dotnet sln add (ls -r **/*.csproj)`

- добавьте файл .gitignore (на уровне рабочей директории) - `dotnet new gitignore`

Замечание: Настройка Visual Code: settings => exclude: внесите шаблоны для `bin` и `obj`

- поместите в gitignore папки `bin` и `obj`:

```
**/bin  
**/obj
```

- откройте начальную архитектуру проекта в Visual Code командой `code .`
- добавьте файл .gitignore (на уровне рабочей директории) - `dotnet new gitignore`

Замечание: Настройка Visual Code: settings => exclude: внесите шаблоны для `bin` и `obj`

- поместите в gitignore папки `bin` и `obj`:

```
**/bin  
**/obj
```

- откройте начальную архитектуру проекта в **Visual Code** командой **code .**

Для проверки работоспособности приложения запустите API

dotnet run --project SportStore.API.

У вас по конечной точке `http://localhost:5290/weatherforecast` должен выводиться результат в формате json. **Примечание:** номер порта может быть другим. У вас по конечной точке `http://localhost:5290/weatherforecast` должен выводиться результат в формате json. **Примечание:** номер порта может быть другим.

- добавьте в решение файл **readme.md**
- commit: "Создание начального проекта API"
- от мастер создать ветку **git switch -c all** и перейти в нее **git switch all**. Далее работа будет вестись в этой ветке.
- commit: "Создание начального проекта API"
- от мастер создать ветку **git switch -c all** и перейти в нее **git switch all**. Далее работа будет вестись в этой ветке.

Разработка домена приложения. Модель пользователя

Создайте в проекте **SportStore.API** папку **Entities**, в которой создайте класс **User**

```
public class User{  
    public Guid Id {get; set;}  
    public string Name {get ;set;}  
}
```

Замечание: тип **Guid** будет пока использоваться для локальной разработки без использования базы данных.

Замечание: тип **Guid** будет пока использоваться для локальной разработки без использования базы данных.

Интерфейсы

Создайте папку **Interfaces** и поместите следующий класс Создайте папку **Interfaces** и поместите следующий класс

```
public interface IUserRepository
{
    User CreateUser(User user);
    List<User> GetUsers();
    User EditUser(User user, Guid id);
    bool DeleteUser(Guid id);
    User FindUserById(Guid id);
}
```

Реализация CRUD в UserRepository

Реализация CRUD в UserRepository

Создайте папку **Repositories** и поместите там следующий класс, который будет имплементировать (реализовывать) интерфейс **IUserRepository**. Создайте папку **Repositories** и поместите там следующий класс, который будет имплементировать (реализовывать) интерфейс **IUserRepository**.

```
public class UserLocalRepository : IUserRepository
public class UserLocalRepository : IUserRepository
{
    public IList<User> Users { get; set; } = new List<User>();
    public IList<User> Users { get; set; } = new List<User>();

    public User CreateUser(User user)
    {
        user.Id = Guid.NewGuid();
        Users.Add(user);
        return user;
        user.Id = Guid.NewGuid();
        Users.Add(user);
        return user;
    }

    public bool DeleteUser(Guid id)
    {
        var result = FindUserById(id);
        Users.Remove(result);
        return true;
    }

    public User EditUser(User user, Guid id)
    {
        var result = FindUserById(id);
        result.Name = user.Name;
        return result;
        var result = FindUserById(id);
        result.Name = user.Name;
        return result;
    }
}
```

```
public User FindUserById(Guid id)
{
    var result = Users.Where(u => u.Id == id).FirstOrDefault();

    if (result == null)
    {
        throw new Exception($"Нет пользователя с id = {id}");
    }
    if (result == null)
    {
        throw new Exception($"Нет пользователя с id = {id}");
    }

    return result;
    return result;
}

public List<User> GetUsers()
{
    return (List<User>)Users;
    return (List<User>)Users;
}
}
```

Примечание:

- очистите папку **Controllers** от файла WeatherForecastController и файл модели.
- в модели данных User будет предупреждение на свойство **Name**, которое можно убрать так: `public string Name { get; set; } = string.Empty;`, т.е указав значение по умолчанию.

Примечание:

- очистите папку **Controllers** от файла WeatherForecastController и файл модели.
- в модели данных User будет предупреждение на свойство **Name**, которое можно убрать так: `public string Name { get; set; } = string.Empty;`, т.е указав значение по умолчанию.

Unit Test

Для реализации unit-тестирования функциональности методов репозитория создадим проект:

```
dotnet new xunit -o SportStore.Tests
```

- добавьте проект с тестами в решение.
- добавьте ссылку в проект с тестами на проект API

```
dotnet add .\SportStore.Tests\ reference .\SportStore.API
```

- удалите файл **UnitTest1**
- создайте класс **UserLocalRepositoryTests** в тестовом проекте

- добавьте проект с тестами в решение.
- добавьте ссылку в проект с тестами на проект API

```
dotnet add .\SportStore.Tests\ reference .\SportStore.API
```

- удалите файл `UnitTest1`
- создайте класс `UserLocalRepositoryTests` в тестовом проекте

```
public class UserLocalRepositoryTests
public class UserLocalRepositoryTests
{
    private readonly UserLocalRepository _userLocalRepository;
    public UserLocalRepositoryTests()
    private readonly UserLocalRepository _userLocalRepository;
    public UserLocalRepositoryTests()
    {
        _userLocalRepository = new UserLocalRepository();
        _userLocalRepository = new UserLocalRepository();
    }

    [Fact]
    public void CreateUser_ShouldReturnNewUserWithGeneratedId()
    {
        // Arrange
        var newUser = new User { Name = "Test User" };
        // Act
        var createdUser = _userLocalRepository.CreateUser(newUser);
        var createdUser = _userLocalRepository.CreateUser(newUser);
        // Assert
        Assert.NotNull(createdUser);
        Assert.NotEqual(Guid.Empty, createdUser.Id);
        Assert.Equal(newUser.Name, createdUser.Name);
    }

    [Fact]
    public void DeleteUser_ShouldReturnTrueAndRemoveUser()
    {
        // Arrange
        var UserLocalRepository = new UserLocalRepository();
        var UserLocalRepository = new UserLocalRepository();
        var testUser = new User { Id = Guid.NewGuid(), Name = "Test User" };
        UserLocalRepository.Users.Add(testUser);
        UserLocalRepository.Users.Add(testUser);

        // Act
        bool result = UserLocalRepository.DeleteUser(testUser.Id);
        bool result = UserLocalRepository.DeleteUser(testUser.Id);

        // Assert
```

```
        Assert.True(result);
        Assert.Empty(UserLocalRepository.Users);
        Assert.Empty(UserLocalRepository.Users);
    }

    [Fact]
    public void EditUser_ShouldUpdateExistingUser()
    {
        // Arrange
        var UserLocalRepository = new UserLocalRepository();
        var UserLocalRepository = new UserLocalRepository();
        var originalUser = new User { Id = Guid.NewGuid(), Name = "Original User"
};

        UserLocalRepository.Users.Add(originalUser);
        UserLocalRepository.Users.Add(originalUser);

        // Act
        var editedUser = new User { Id = originalUser.Id, Name = "Edited User" };
        var result = UserLocalRepository.EditUser(editedUser, originalUser.Id);
        var result = UserLocalRepository.EditUser(editedUser, originalUser.Id);

        // Assert
        Assert.NotNull(result);
        Assert.Equal("Edited User", result.Name);
        Assert.Single(UserLocalRepository.Users);
        Assert.Single(UserLocalRepository.Users);
    }

    [Fact]
    public void FindUserById_ShouldReturnUserByValidId()
    {
        // Arrange
        var UserLocalRepository = new UserLocalRepository();
        var UserLocalRepository = new UserLocalRepository();
        var testUser = new User { Id = Guid.NewGuid(), Name = "Test User" };
        UserLocalRepository.Users.Add(testUser);
        UserLocalRepository.Users.Add(testUser);

        // Act
        var foundUser = UserLocalRepository.FindUserById(testUser.Id);
        var foundUser = UserLocalRepository.FindUserById(testUser.Id);

        // Assert
        Assert.NotNull(foundUser);
        Assert.Equal(testUser.Id, foundUser.Id);
        Assert.Equal(testUser.Name, foundUser.Name);
    }

    [Fact]
    public void FindUserById_ShouldThrowExceptionForInvalidId()
    {
        // Arrange
        var UserLocalRepository = new UserLocalRepository();
        var UserLocalRepository = new UserLocalRepository();
```

```

        // Act & Assert
        Assert.Throws<Exception>(() =>
        UserLocalRepository.FindUserById(Guid.NewGuid()));
        Assert.Throws<Exception>(() =>
        UserLocalRepository.FindUserById(Guid.NewGuid()));
    }

    [Fact]
    public void GetUsers_ShouldReturnAllUsers()
    {
        // Arrange
        var UserLocalRepository = new UserLocalRepository();
        var UserLocalRepository = new UserLocalRepository();
        var testUser1 = new User { Id = Guid.NewGuid(), Name = "User 1" };
        var testUser2 = new User { Id = Guid.NewGuid(), Name = "User 2" };
        UserLocalRepository.Users.Add(testUser1);
        UserLocalRepository.Users.Add(testUser2);
        UserLocalRepository.Users.Add(testUser1);
        UserLocalRepository.Users.Add(testUser2);

        // Act
        var users = UserLocalRepository.GetUsers();
        var users = UserLocalRepository.GetUsers();

        // Assert
        Assert.NotNull(users);
        Assert.Equal(2, users.Count);
        Assert.Contains(testUser1, users);
        Assert.Contains(testUser2, users);
    }

    [Fact]
    public void FindUserById_ShouldThrowExceptionForNonExistentId()
    {
        // Arrange
        var UserLocalRepository = new UserLocalRepository();
        var UserLocalRepository = new UserLocalRepository();

        // Act & Assert
        Assert.Throws<Exception>(() =>
        UserLocalRepository.FindUserById(Guid.NewGuid()));
        Assert.Throws<Exception>(() =>
        UserLocalRepository.FindUserById(Guid.NewGuid()));
    }
}

```

- запуск всех тестов `dotnet test`
- просмотр все доступных тестов `dotnet test --list-tests`
- запуск конкретного списка по фильтру `dotnet test --filter "FullyQualifiedName=SportStore.Tests.UserLocalRepositoryTests.CreateUser_ShouldReturnNewUserWithGeneratedId"`

- запуск конкретного списка по фильтру `dotnet test --filter "FullyQualifiedName=SportStore.Tests.UserLocalRepositoryTests.CreateUser_ShouldReturnNewUserWithGeneratedId"`

Создание UsersController для управления пользователями

```
[ApiController]
[Route("[controller]")]
public class UsersController : ControllerBase
public class UsersController : ControllerBase
{
    private readonly IUserRepository _repo;
    public UsersController(IUserRepository repo)
    public UsersController(IUserRepository repo)
    {
        _repo = repo;
    }

    [HttpPost]
    public ActionResult CreateUser(User user){

        return Ok(_repo.CreateUser(user));
        return Ok(_repo.CreateUser(user));
    }

    [HttpGet]
    public ActionResult GetUser(){
        return Ok(_repo.GetUsers());
    }

    [HttpPut]
    public ActionResult UpdateUser(User user){
        return Ok(_repo.EditUser(user, user.Id));
    }

    [HttpGet("{id}")]
    public ActionResult GetById(Guid id){
        return Ok(_repo.FindUserById(id));
    }

    [HttpDelete]
    public ActionResult DeleteUser(Guid id){
        return Ok(_repo.DeleteUser(id));
    }
}
```



```
}
```

- запустите API: `dotnet run --project SportStore.API`.
- запустите API: `dotnet run --project SportStore.API`.
- но сейчас вы получите ошибку

```
Unable to resolve service for type 'SportStore.API.Interfaces.IUserRepository'
while attempting to activate 'SportStore.API.Controllers.UsersController'.
```

Эта ошибка говорит о том, что контроллеру в конструктор требуется реализация интерфейса `IUserRepository`, которую мы будем получать из контейнера внедрения зависимостей (DI).

- но сейчас вы получите ошибку

```
Unable to resolve service for type 'SportStore.API.Interfaces.IUserRepository'
while attempting to activate 'SportStore.API.Controllers.UsersController'.
```

Эта ошибка говорит о том, что контроллеру в конструктор требуется реализация интерфейса `IUserRepository`, которую мы будем получать из контейнера внедрения зависимостей (DI).

DI

Контроллер `UserController` запрашивает в своем конструкторе Контроллер `UserController` запрашивает в своем конструкторе

```
private readonly IUserRepository _repo;
public UserController(IUserRepository repo)
public UserController(IUserRepository repo)
{
    _repo = repo;
}
```

реализацию интерфейса `IUserRepository`, который ему должен предоставить DI (Dependency Injection) - контейнер внедрения зависимости встроенный во фреймворк ASP Core. Для этого надо зарегистрировать сервис в коллекции сервисов в проекте API.

```
builder.Services.AddSingleton<IUserRepository, UserLocalRepository>();
builder.Services.AddSingleton<IUserRepository, UserLocalRepository>();
```

- запустите проект и проверьте все конечные точки по пути `http://localhost:[port]/swagger/index.html`
- запустите проект и проверьте все конечные точки по пути `http://localhost:[port]/swagger/index.html`

Validation

DataAnnotation

При отправке пост запросов надо проверять модель данных на соответствие валидности. Для этого применяются инструменты: встроенные средства проверки `DataAnnotation` и пакет `FluentValidation`.

Атрибут для проверки минимального длины имени

```
[MinLength(5,ErrorMessage = "Минимальное длина имени 5")]
public string Name {get ;set;} = string.Empty;
```

Для создания собственного атрибута валидации `DataAnnotation` создайте папку `Validations` и в ней создайте класс `UserValidator`

```
public class MaxLengthAttribute : ValidationAttribute
{
    private readonly int _maxLength;

    public MaxLengthAttribute(int maxLength) : base($"Name max {maxLength} ")
    {
        _maxLength = maxLength;
    }

    public override bool IsValid(object? value)
    {
        return ((String)value!).Length <= _maxLength;
    }
}
```

Таким образом модель будет выглядеть следующим образом:

```
public class User
{
    public Guid Id {get ;set;} = Guid.NewGuid();

    [MinLength(5,ErrorMessage = "Минимальное длина имени 5")]
    [SportStore.API.Validations.MaxLength(10)]
    public string Name {get ;set;} = string.Empty;
}
```

FluentValidation

Установите пакет **FluentValidation**:

```
dotnet add .\SportStore.API\ package FluentValidation
```

Установите пакет **FluentValidation**:

```
dotnet add .\SportStore.API\ package FluentValidation
```

Создайте в папке **Validations** новый класс. Создайте в папке **Validations** новый класс.

```
public class FluentValidator : AbstractValidator<User>
{
    public FluentValidator()
    {
        RuleFor(u => u.Name).Must(StartsWithCapitalLetter).WithMessage("Имя пользователя должно начинаться с заглавной буквы");
    }

    private bool StartsWithCapitalLetter(string username)
    {
        return char.IsUpper(username[0]);
    }
}
```

Для применения валидатора к конечной точки создания пользователя: Для применения валидатора к конечной точки создания пользователя:

```
var validator = new FluentValidator();
var result = validator.Validate(user);
if(!result.IsValid){
    throw new Exception($"{result.Errors.First().ErrorMessage}");
}
```

Задание: проверьте метод контроллера создания пользователя, чтобы имя пользователя начиналось с заглавной буквы.

Postman(Swagger,request.http) для тестирования API

- Способ 1 Протестируйте работу API на примере управления пользователями с помощью встроенного средства Swagger по адресу <http://localhost:5290/swagger>

- Способ 2. Postman
- Способ 3. Запросы .http

Создайте в корневой директории папку **requests** в которой создайте файл с расширением **http**. Например, **getusers.http** Создайте в корневой директории папку **requests** в которой создайте файл с расширением **http**. Например, **getusers.http**

```
GET http://localhost:5290/User
```

postuser.http

```
POST http://localhost:5290/User
Content-Type: application/json

{
  "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "name": "Newuser123213131"
}
```

Проверка запросов осуществляется с помощью VS Code.

Задание 1: у пользователя должна быть роль. Создайте модель для роли пользователя, интерфейс, репозиторий, контроллер, валидации, напишите unit-тесты для репозитории.

Задание 2: при запросе post на создание нового ресурса обычно принято отвечать кодом **201**.

Примените метод **Created** для возврата ответа типа **ActionResult** **Задание 1:** у пользователя должна быть роль. Создайте модель для роли пользователя, интерфейс, репозиторий, контроллер, валидации, напишите unit-тесты для репозитории.

Задание 2: при запросе post на создание нового ресурса обычно принято отвечать кодом **201**.

Примените метод **Created** для возврата ответа типа **ActionResult**