

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования  
Кафедра проектирования информационно-компьютерных систем

## **ОТЧЕТ**

### **ПО ИНДИВИДУАЛЬНОЙ ПРАКТИЧЕСКОЙ РАБОТЕ №1**

по дисциплине

«Современные технологии проектирования информационных систем»

специальности 1-40 05 01-10 «Информационные системы и технологии (в  
бизнес-менеджменте)»

Выполнил:  
студент группы №994351  
Богомаз Д. Л.

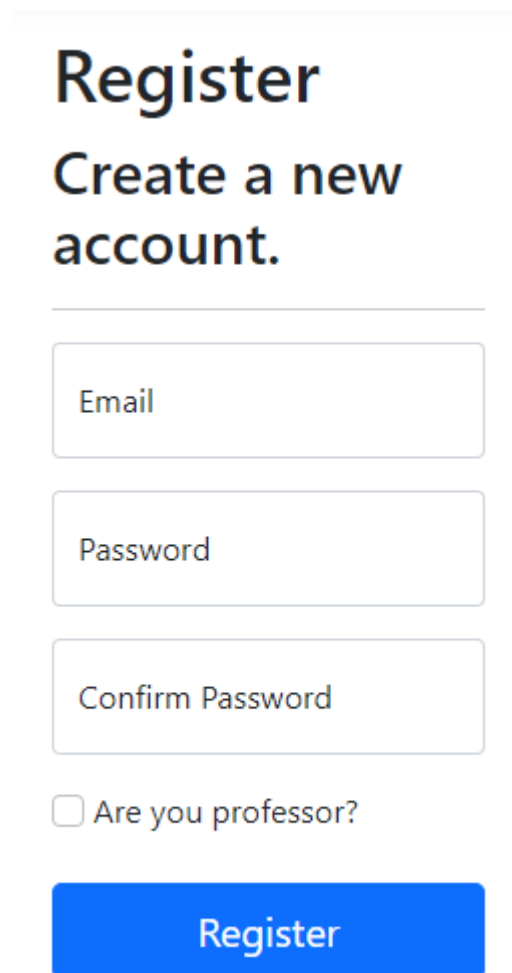
Проверила: Левченя Ж.Б.

Минск 2022

Цель: создать программные средства мониторинга и консультативной поддержки процесса изучения дисциплин вуза, реализовать паттерн проектирования (в данном случае singleton).

В данной работе выполняется вариант номер 3: Программные средства мониторинга и консультативной поддержки процесса изучения дисциплин вуза.

В ходе данной работы было создано приложение на dotnet 7, mvc тип, где в качестве frontend выбран razor pages. Базовая логика создана с помощью MVC паттерна проектирования, в качестве логина использован scaffold identity default с изменяем ролей для преподавателей, также были удалены неиспользованный функционал, как работа с почтой. Также на форме регистрации добавлен выбор режима регистрации:

A screenshot of a web registration form. At the top, the text "Register" is in a large, bold, dark blue font, followed by "Create a new account." in a slightly smaller, bold, dark blue font. Below this is a horizontal line. There are three input fields stacked vertically: "Email", "Password", and "Confirm Password". Each field has a light blue border and a light blue placeholder text. Below the input fields is a checkbox with the text "Are you professor?". At the bottom is a blue button with the text "Register" in white.

В качестве базы данных используется sql lite, как orm используется entity framework, automapper не используется, так как является излишним для подобного формата работы (mappings реализованы как статический класс).

Ссылка на проект: <https://github.com/bogomazdmitry/LmsCopy>

# Снимки экрана работы приложения

## Professor, your marks!

Create mark

Subject	Student name	Mark	Edit	Delete
Math	bogomaz.dima2013@gmail.comAA	9	<a href="#">Edit</a>	<a href="#">Delete</a>
Math	bogomaz.dima2013@gmail.comAA	5	<a href="#">Edit</a>	<a href="#">Delete</a>

## Subjects!

Create subject

Json

Generate

Name	Edit	Delete
Math	<a href="#">Edit</a>	<a href="#">Delete</a>
Sub-1	<a href="#">Edit</a>	<a href="#">Delete</a>
Sub-2	<a href="#">Edit</a>	<a href="#">Delete</a>
Sub-3	<a href="#">Edit</a>	<a href="#">Delete</a>
Sub-4	<a href="#">Edit</a>	<a href="#">Delete</a>
Sub-5	<a href="#">Edit</a>	<a href="#">Delete</a>

## Create subject

Name

Save

## Create mark

Subject name

Student name

Mark

Save

## Manage your account

### Change your account settings

Profile

Password

Personal data

#### Profile

Username

bogomaz.dima2013@gmail.comAA

Phone number

Save

## Log in

Use a local account to log in.

Email  
bogomaz.dima2013@gmail.comA

Password  
\*\*\*\*\*

☐ Remember me?

Log in

[Forgot your password?](#)

[Register as a new user](#)

Student, your marks!

No marks found.

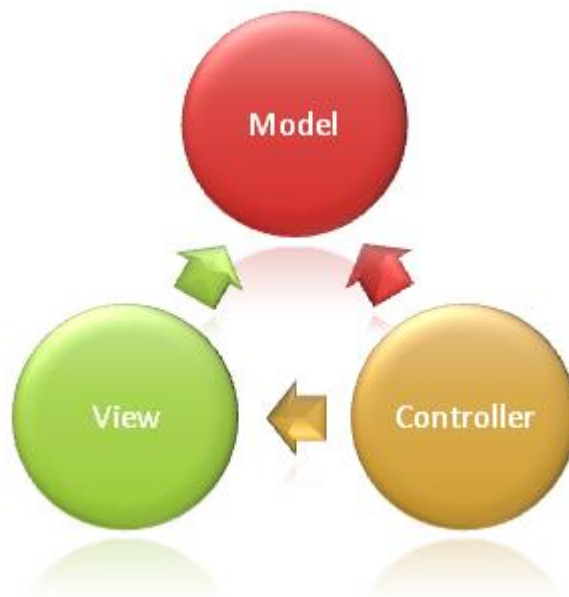
## Шаги реализации приложения

1. Скафолд стандартного identity
2. Исправления и настройка identity (создание ролей, задание поведения при попытке доступа на контроллеры, которые требуют авторизации, чистка скафолда, задание требований к паролю)
3. Создание моделей, изменения базы приложения с использованием миграций и ApplicationDbContext
4. Создание представлений на Razor для основных частей приложения (таблиц для просмотра и форм для изменения)
5. Создание контроллеров для каждого представления

## Теоретические сведения

Структура архитектуры MVC разделяет приложение на три основных группы компонентов: модели, представления и контроллеры. Это позволяет реализовать принципы разделения задач. Согласно этой структуре запросы пользователей направляются в контроллер, который отвечает за работу с моделью для выполнения действий пользователей и (или) получение результатов запросов. Контроллер выбирает представление для отображения пользователю со всеми необходимыми данными модели.

На следующей схеме показаны три основных компонента и существующие между ними связи.



ASP.NET Core поддерживает шаблон проектирования программного обеспечения внедрения зависимостей (DI), который представляет собой метод достижения инверсии управления (IoC) между классами и их зависимостями. Именно это и будет использовано для реализации паттерна Singleton. Такой тип зависимости создаст объект единожды, а дальше будет отдавать созданный контейнер.

### Настройка singleton

Файл Program.cs:

```
builder.Services.AddSingleton<SettingsService>();
```

Класс SettingsService:

```
namespace LmsCopy.Web.Services;
```

```
public class SettingsService
{
    public string SubjectReportName = "subjects";
}
```

Использование контейнера:

```
public class SubjectController : Controller
{
    private readonly ApplicationDbContext _context;

    private readonly SettingsService _settings;
```

```

        public SubjectController(ApplicationDbContext context, SettingsService
settings)
        {
            _context = context;
            _settings = settings;
        }

        [HttpGet]
        [Authorize(Roles = UserRole.Professor)]
        public IActionResult GenerateReport(String reportType)
        {
            var fileReport =
FileReportFactory.GetServiceIndexRequest(reportType,
_settings.SubjectReportName);

            var file =
fileReport.GenerateReport<Subject>(_context.Subjects.ToList());
            return File(file.Data, file.ContentType, file.FileName);
        }
    }
}

```

### ЛИСТИНГ кода

```

ApplicationDbContext:
using LmsCopy.Web.Entites;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace LmsCopy.Web.Data;

public class ApplicationDbContext : IdentityDbContext<User, UserRole, Guid>
{
    public DbSet<Subject> Subjects { get; set; }

    public DbSet<Mark> Marks { get; set; }

    public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
        : base(options)
    {

```

```

    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity<Mark>()
            .HasOne<Subject>(m => m.Subject)
            .WithMany();

        modelBuilder.Entity<Mark>()
            .HasOne<User>(m => m.Student)
            .WithMany(u => u.StudentMarks);

        modelBuilder.Entity<Mark>()
            .HasOne<User>(m => m.Professor)
            .WithMany(u => u.ProfessorMarks);
    }
}

    SubjectController:
using LmsCopy.Web.Data;
using LmsCopy.Web.Entites;
using LmsCopy.Web.Models;
using LmsCopy.Web.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Text;

namespace LmsCopy.Web.Controllers;

public class SubjectController : Controller
{
    private readonly ApplicationDbContext _context;

    private readonly SettingsService _settings;

    public SubjectController(ApplicationDbContext context, SettingsService
settings)
    {

```

```
    _context = context;
    _settings = settings;
}
```

```
[HttpGet]
[Authorize(Roles = UserRole.Professor)]
public IActionResult Create()
{
    return View();
}
```

```
[HttpPost]
[Authorize(Roles = UserRole.Professor)]
public IActionResult Create(Subject subject)
{
    if (ModelState.IsValid)
    {
        _context.Subjects.Add(subject);
        _context.SaveChanges();
        return RedirectToAction("Index");
    }
    return View("Create", subject);
}
```

```
[HttpGet]
[Authorize(Roles = UserRole.Professor)]
public IActionResult Edit(Guid id)
{
    var subject = _context.Subjects.FirstOrDefault(s => s.Id == id);
    return View(subject);
}
```

```
[HttpPost]
[Authorize(Roles = UserRole.Professor)]
public IActionResult Edit(Subject subject)
{
    if (ModelState.IsValid)
    {
        _context.Subjects.Update(subject);
    }
}
```



```

        _context.SaveChanges();
        return RedirectToAction("Index");
    }
    return View("Edit", subject);
}

```

```

[Authorize(Roles = UserRole.Professor)]
public IActionResult Delete(Guid id)
{
    if (ModelState.IsValid)
    {
        var subject = _context.Subjects.FirstOrDefault(s => s.Id == id);
        if (subject != null)
        {
            _context.Subjects.Remove(subject);
            _context.SaveChanges();
        }

        return RedirectToAction("Index");
    }
    return View("Index");
}

```

```

public IActionResult Index()
{
    var subjects = _context.Subjects.ToList();
    return View(subjects);
}

```

```

[HttpGet]
[Authorize(Roles = UserRole.Professor)]
public IActionResult GenerateReport(String reportType)
{
    var fileReport = FileReportFactory.GetServiceIndexRequest(reportType,
        _settings.SubjectReportName);

    var file = fileReport.GenerateReport<Subject>(_context.Subjects.ToList());
    return File(file.Data, file.ContentType, file.FileName);
}

```

}

Вывод: singleton является порождающим паттерном проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа. Также сам паттерн в то же время может представлять собой некоторый супер-класс, что в свою очередь является анти-паттерном. Поэтому не стоит использовать singleton, когда можно это избежать. Паттерн singleton был реализован на языке C#.