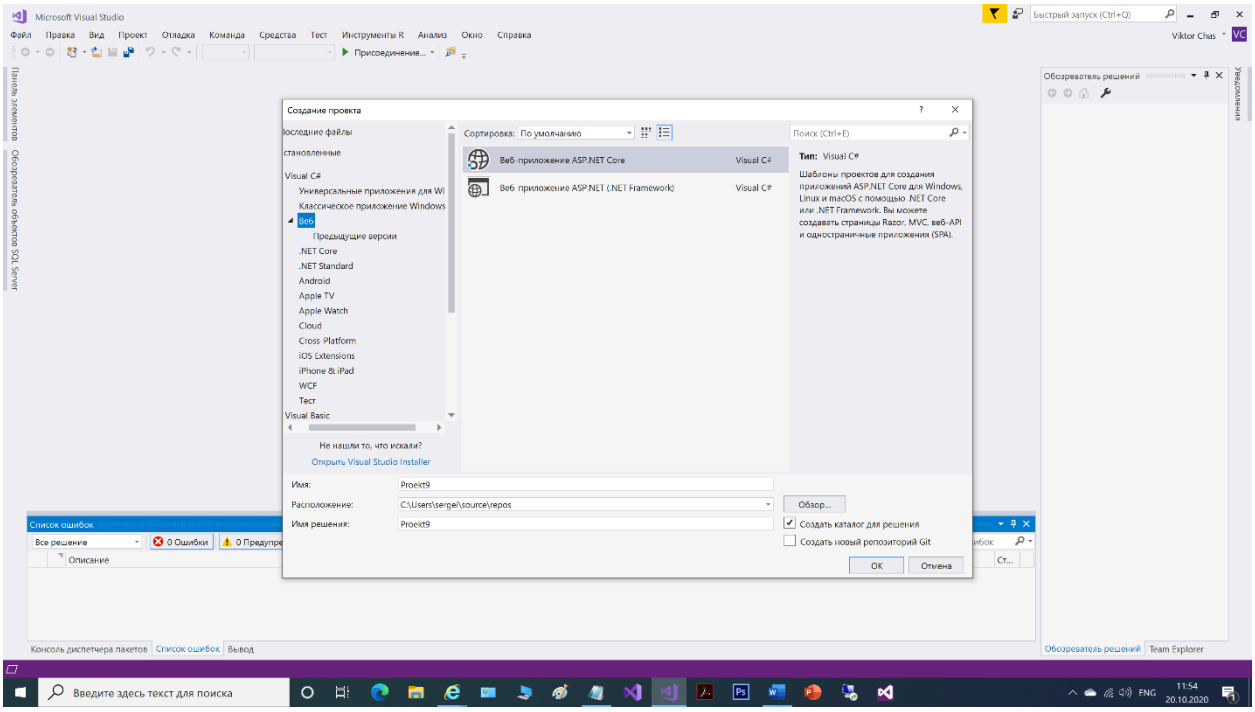
Лабораторная работа 2. Разработка технического проекта на создание ИС средствами технологии объектного проектирования информационных систем

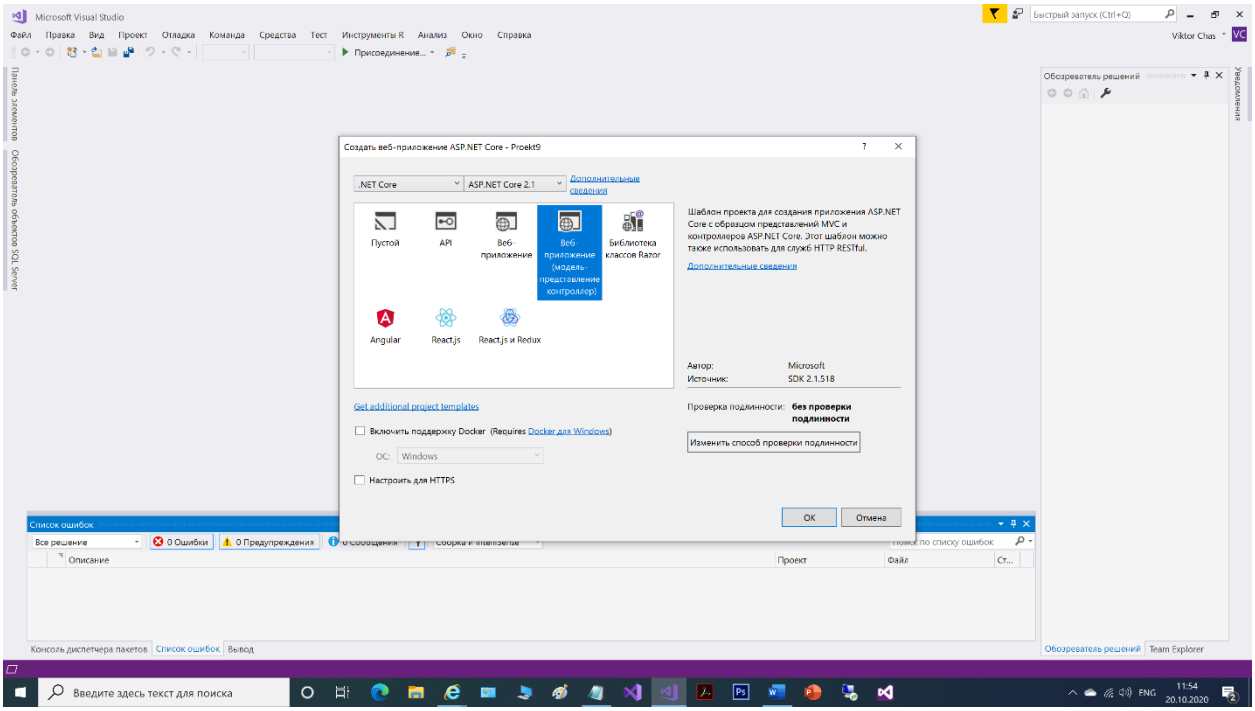
Часовских В.П., сайт http://vikchas.ru

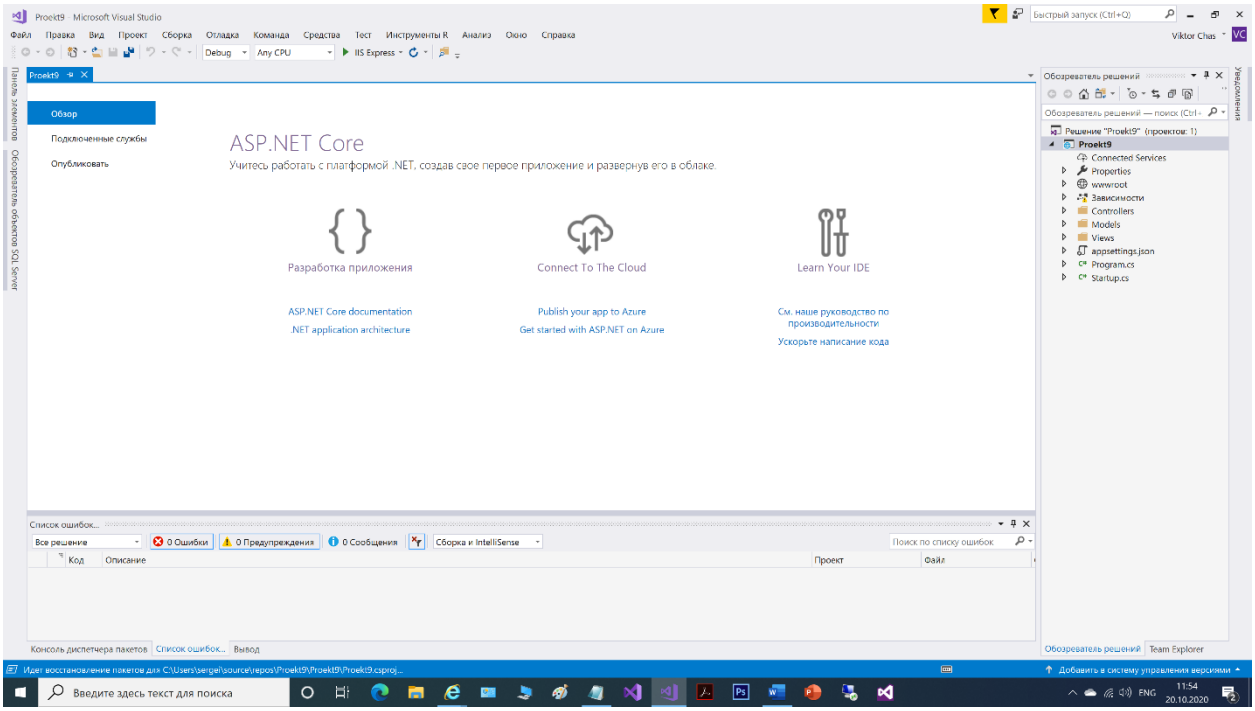
Разработка технического проекта на создание ИС основывается на технологии объектного проектирования ASP.NET Core MVC. Для начального освоения ASP.NET Core MVC предлагается реальный проект для подготовки и выполнения как показано в методической части. Предполагается, что студент знает как запустить Visual Studio 2017 Enterprise интегрированная комплексная платформа для групп любого размера с высокими требованиями к качеству и масштабу и полным набором инструментов и служб для разработки, **создания сложных корпоративных приложений** и управления ими.

Необходимо запустить Visual Studio 2017 Enterprise и последовательно выполнить последующие действия. После выполнения реального примера ИС можно приступить к разработке проекта.



Обратите внимание на «Настроить для HTTPS» - квадратик пуст.





Настройка стиля сайта

Выполните незначительную настройку меню, макета и домашней страницы сайта.

Откройте файл *Views/Shared/_Layout.cshtml* и внесите следующие изменения:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Proekt9</title>

    <environment include="Development">
        <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
        <link rel="stylesheet" href="~/css/site.css" />
    </environment>
    <environment exclude="Development">
        <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css"
              asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
              asp-fallback-test-class="sr-only" asp-fallback-test-property="position"
asp-fallback-test-value="absolute" />
        <link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />
    </environment>
</head>
<body>
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target=".navbar-collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a asp-area="" asp-controller="Home" asp-action="Index" class="navbar-
brand">Proekt9</a>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li><a asp-area="" asp-controller="Home" asp-action="Index">Домашняя
страница</a></li>
                    <li><a asp-area="" asp-controller="Students" asp-
action="Index">Студент</a></li>
                    <li><a asp-area="" asp-controller="Facultys" asp-
action="Index">Факультет</a></li>
                    <li><a asp-area="" asp-controller="Courses" asp-
action="Index">Курс</a></li>
                    <li><a asp-area="" asp-controller="Trainers" asp-
action="Index">Преподаватель</a></li>
                    <li><a asp-area="" asp-controller="Grades" asp-
action="Index">Оценка</a></li>


                </ul>
            </div>
        </div>
    </nav>

    <partial name="_CookieConsentPartial" />

    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; 2020 - Proekt9</p>
        </footer>
    </div>
```

```
    <environment include="Development">
        <script src="~/lib/jquery/dist/jquery.js"></script>
        <script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
        <script src="~/js/site.js" asp-append-version="true"></script>
    </environment>
    <environment exclude="Development">
        <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.3.1.min.js"
                asp-fallback-src="~/lib/jquery/dist/jquery.min.js"
                asp-fallback-test="window.jQuery"
                crossorigin="anonymous"
                integrity="sha384-
tsQFqpEReu7ZLhBV2VZlAu7zcOV+rXbYlF2cqB8txI/8aZajjp4Bqd+V6D5IgvKT">
        </script>
        <script
src="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"
                asp-fallback-src="~/lib/bootstrap/dist/js/bootstrap.min.js"
                asp-fallback-test="window.jQuery && window.jQuery.fn &&
window.jQuery.fn.modal"
                crossorigin="anonymous"
                integrity="sha384-
aJ21OjlMXNL5UyIl/XNwTMqvzeRMZH2w8c5cRVpzpU8Y5bApTppSuUkhZXN0VxHd">
        </script>
        <script src="~/js/site.min.js" asp-append-version="true"></script>
    </environment>

    @RenderSection("Scripts", required: false)
</body>
</html>
```

Замените содержимое файла *Views/Home/Index.cshtml* следующим кодом, который заменяет текст о ASP.NET и MVC описанием этого приложения:

```
@{
    ViewBag.Title = "Home Page";
}

 <h1 style="text-align:center; color:#796310">Курс  ASP.NET Core MVC  - создание базы
данных</h1>
```

```
Создадим в папке Models пять сущностей (таблиц будущей базы данных) – тип файла класс
 Student , Faculty , Course , Trainer , Grades

И заменим сгенерированный код следующим:

using System.Collections.Generic;
using System.ComponentModel;

namespace Proekt9.Models
{    public class Student
    {
        [DisplayName("Ключ")]
        public int StudentID { get; set; }
        [DisplayName("Фамилия")]
        public string FirstName { get; set; }
        [DisplayName("Имя")]
        public string LastName { get; set; }
        //      [DisplayName("Факультет")]
        //      public int FacultyID { get; set; }
        [DisplayName("Дата из")]
        public string DateFrom { get; set; }
        [DisplayName("Дата в")]
```

```
        public string DateTo { get; set; }

        //       public virtual Faculty Faculty { get; set; }

        public virtual ICollection<Student> Students { get; set; } // включает одну
ссылку

    }
}


using System.Collections.Generic;
using System.ComponentModel;

namespace Proekt9.Models
{
    public class Faculty
    {
        [DisplayName("Ключ")]
        public int FacultyID { get; set; }
        [DisplayName("Название факультета")]
        public string FacultyTitle { get; set; }

        public virtual ICollection<Faculty> Facultys { get; set; }  // Будет как
ссылачная таблица или классификатор
    }
}


using System.Collections.Generic;
using System.ComponentModel;

namespace Proekt9.Models
{    public class Course
    {
                [DisplayName("Ключ")]
        public int CourseID { get; set; }
        [DisplayName("Название курса")]
        public string CourseTitle { get; set; }
        [DisplayName("Трайнер")]
        public int TrainerID { get; set; }
        [DisplayName("Факультет")]
        public int FacultyID { get; set; }
        [DisplayName("Дата в")]
        public string DateTo { get; set; }

        public virtual Trainer Trainer { get; set; }
        public virtual Faculty Faculty { get; set; }

        public virtual ICollection<Course> Courses { get; set; } // включает две ссылки

    }
}



using System.Collections.Generic;
using System.ComponentModel;

namespace Proekt9.Models
{
   public class Trainer
    {
        [DisplayName("Ключ")]
        public int TrainerID { get; set; }
```

```
        [DisplayName("Фамилия")]
        public string FirstName { get; set; }
        [DisplayName("Имя")]
        public string LastName { get; set; }
        [DisplayName("Дата из")]
        public string DataFrom { get; set; }
        [DisplayName("Дата в")]
        public string DateTo { get; set; }

        public virtual ICollection<Trainer> Trainers { get; set; }  // Будет как
ссылачная таблица или классификатор
    }

}


using System.Collections.Generic;
using System.ComponentModel;

namespace Proekt9.Models
{    public class Grade
    {
        [DisplayName("Ключ")]
        public int GradeID { get; set; }
        [DisplayName("Студент")]
        public int StudentID { get; set; }
        [DisplayName("Курс")]
        public int CourseID { get; set; }
        [DisplayName("GradeValue")]
        public string GradeValue { get; set; }
        [DisplayName("Дата")]
        public string Date { get; set; }

        public virtual Course Course { get; set; }
        public virtual Student Student { get; set; }

        public virtual ICollection<Grade> Grades { get; set; } // включает две ссылки

    }
}
```

Создание контекста базы данных

Контекст базы данных — это основной класс, который координирует функциональные возможности Entity Framework для заданной модели данных. Этот класс создается путем наследования от класса `Microsoft.EntityFrameworkCore.DbContext`. В коде указываются сущности, которые включаются в модель данных. Также вы можете настроить реакцию платформы Entity Framework на некоторые события. В этом проекте соответствующий класс называется `EFDbContext`.

В папке проекта создайте папку *Data*.

В папке *Data* создайте новый файл класса с именем *EFDbContext.cs* и замените код шаблона следующим кодом:

```
using Proekt9.Models;
```

```
using Microsoft.EntityFrameworkCore;

namespace Proekt9.Data
{
    public class EFDbContext : DbContext
    {
        public EFDbContext(DbContextOptions<EFDbContext> options) : base(options)
        {
        }

        public DbSet<Course> Courses { get; set; }
        public DbSet<Faculty> Facultys { get; set; }
        public DbSet<Grade> Grades { get; set; }
        public DbSet<Student> Students { get; set; }
        public DbSet<Trainer> Trainers { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Course>().ToTable("Course");
            modelBuilder.Entity<Faculty>().ToTable("Faculty");
            modelBuilder.Entity<Student>().ToTable("Student");
            modelBuilder.Entity<Grade>().ToTable("Grade");
            modelBuilder.Entity<Trainer>().ToTable("Trainer");


        }

    }
}
```

Регистрация EFDbContext

ASP.NET Core по умолчанию реализует технологию внедрения зависимостей. С помощью внедрения зависимостей службы (например, контекст базы данных EF) регистрируются во время запуска приложения. Затем компоненты, которые используют эти службы (например, контроллеры MVC), обращаются к ним через параметры конструктора. Код конструктора контроллера, который получает экземпляр контекста, будет приведен позднее в этом учебнике.
Чтобы зарегистрировать `EFDbContext` как службу, откройте файл *Startup.cs* и добавьте выделенные строки в метод `ConfigureServices`.

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Proekt9.Data;

namespace Proekt9
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }
```

```
        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the
container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.Configure<CookiePolicyOptions>(options =>
            {
                // This lambda determines whether user consent for non-essential cookies
is needed for a given request.
                options.CheckConsentNeeded = context => true;
                options.MinimumSameSitePolicy = SameSiteMode.None;
            });

            services.AddDbContext<EFDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));


            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
        }

        // This method gets called by the runtime. Use this method to configure the HTTP
request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
            }

            app.UseStaticFiles();
            app.UseCookiePolicy();

            app.UseMvc(routes =>
            {
                routes.MapRoute(
                    name: "default",
                    template: "{controller=Home}/{action=Index}/{id?}");
            });
        }
    }
}
```

Откройте файл *appsettings.json* и добавьте строку подключения, как показано в следующем примере.

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=(localdb)\\mssqllocaldb;Database=Proekt91;Trusted_Connection=T
rue;MultipleActiveResultSets=true"
  },
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
```

```
      "Default": "Warning"
    }
  }
}
```

SQL Server Express LocalDB

Строка подключения указывает на базу данных SQL Server LocalDB. LocalDB — это упрощенная версия ядра СУБД SQL Server Express, предназначенная для разработки приложений и не ориентированная на использование в производственной среде. LocalDB запускается по запросу в пользовательском режиме, поэтому настройки не слишком сложны.

Инициализация базы данных с тестовыми данными

Платформа Entity Framework создает пустую базу данных. В этом разделе вы напишете метод, который вызывается после создания базы данных и заполняет ее тестовыми данными.
Здесь будет использоваться метод `EnsureCreated` для автоматического создания базы данных. .
В папке *Data* создайте новый файл класса с именем *DbInitializer.cs* и замените код шаблона следующим кодом, который обеспечивает создание базы данных и загрузку в нее тестовых данных в одну таблицу.

```
namespace Proekt9.Data
{
    public static class DbInitializer
    {
        public static void Initialize(EFDbContext context)
        {
            context.Database.EnsureCreated();

            // Look for any students.
            if (context.Students.Any())
            {
                return;   // DB has been seeded
            }

            var students = new Student[]
            {
            new Student{FirstName="Popov",LastName="Alexander",DateFrom = "11",DateTo = "22"},
            new Student{FirstName="Carson",LastName="Pavel",DateFrom = "111",DateTo = "222"},
            new Student{FirstName="Serov",LastName="Oleg",DateFrom = "1111",DateTo = "2222"}
             };
            foreach (Student s in students)
            {
                context.Students.Add(s);
            }
            context.SaveChanges();


        }
    }
}
```

В файле *Program.cs* измените метод `Main`, чтобы реализовать следующее поведение при запуске приложения:

- Получение экземпляра контекста базы данных из контейнера внедрения зависимостей.
- Вызов метода инициализации с передачей ему контекста.

- Высвобождение контекста после завершения работы метода заполнения.

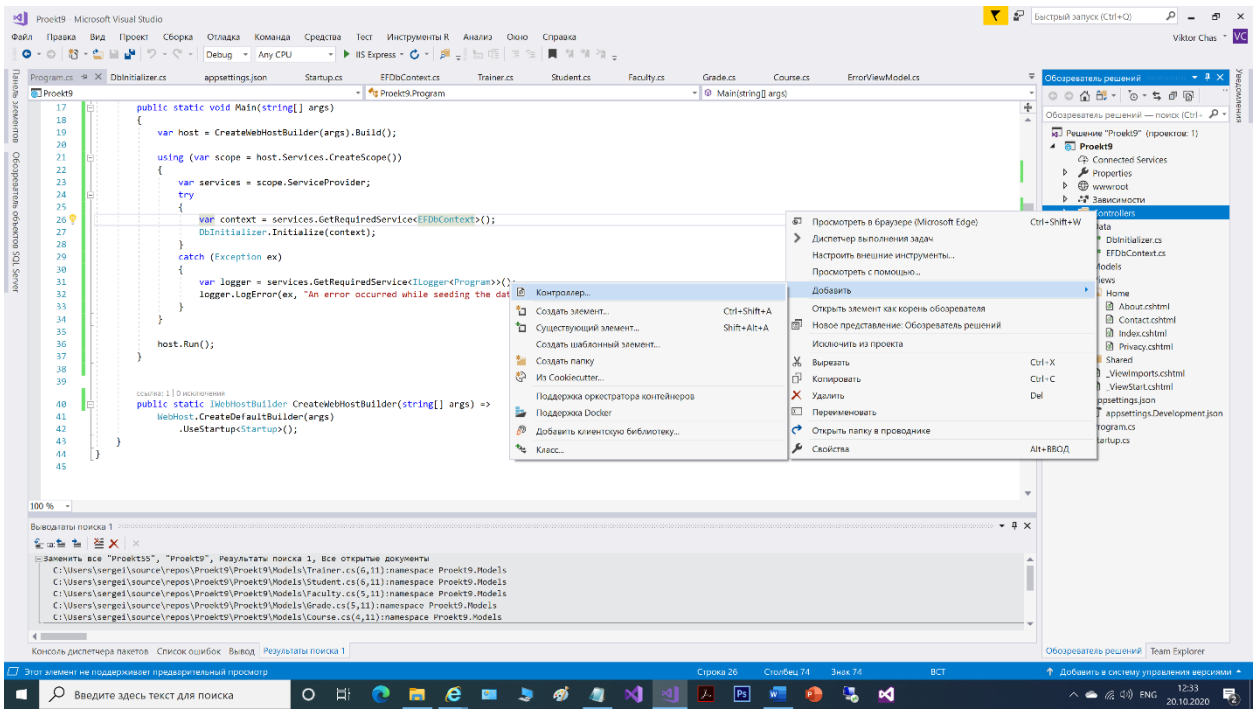
```
public static void Main(string[] args)
{
     var host = CreateWebHostBuilder(args).Build();

    using (var scope = host.Services.CreateScope())
    {
        var services = scope.ServiceProvider;
        try
        {
            var context = services.GetRequiredService<EFDbContext>();
            DbInitializer.Initialize(context);
        }
        catch (Exception ex)
        {
            var logger = services.GetRequiredService<ILogger<Program>>();
            logger.LogError(ex, "An error occurred while seeding the database.");
        }
    }

    host.Run();
}
```

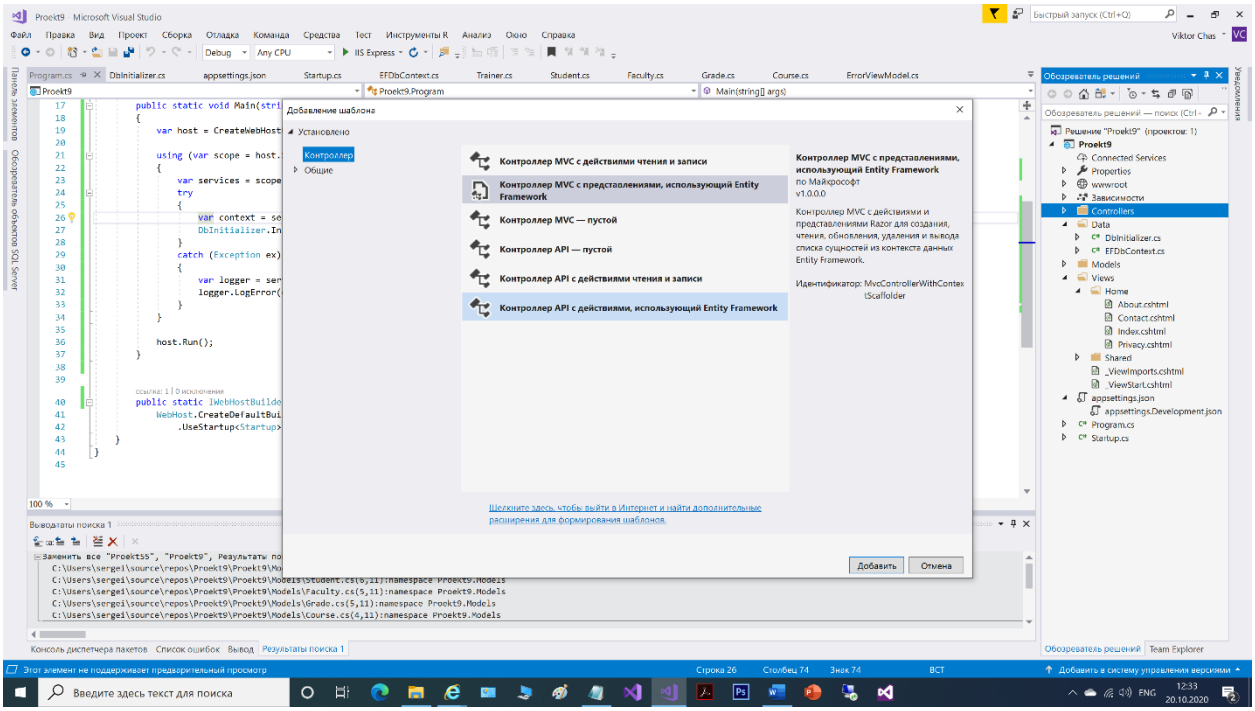
Теперь при первом запуске приложения будет создана и заполнена тестовыми данными необходимая для работы база данных. При любом изменении модели данных вы можете удалить базу, обновить метод заполнения и начать работу с новой базой данных аналогичным способом.

Создание контроллера и представлений

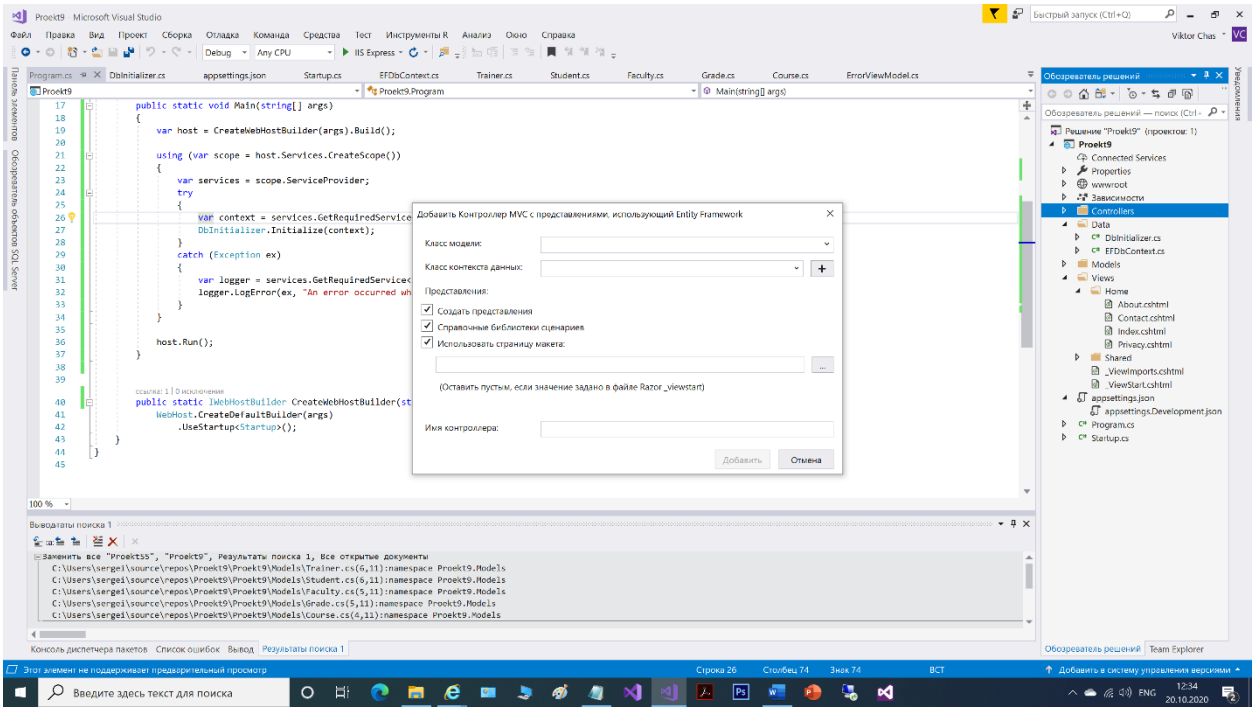
Далее вы будете использовать механизм шаблонов Visual Studio для добавления контроллера и представлений MVC, которые будут использовать платформу EF для запроса данных и их сохранения.
Автоматическое создание методов и представлений операций CRUD (создание, чтение, обновление и удаление) называется формированием шаблонов. Формирование шаблонов отличается от создания кода тем, что шаблонный код является отправной точкой и может изменяться в соответствии с потребностями, тогда как сформированный код обычно не изменяется. В тех случаях, когда требуется настроить созданный код в соответствии с внесенными изменениями, вы можете использовать разделяемые классы или повторно создать код.

- Щелкните правой кнопкой мыши папку **Контроллеры** в **обозревателе решений** и выберите **Добавить > Создать шаблонный элемент**.

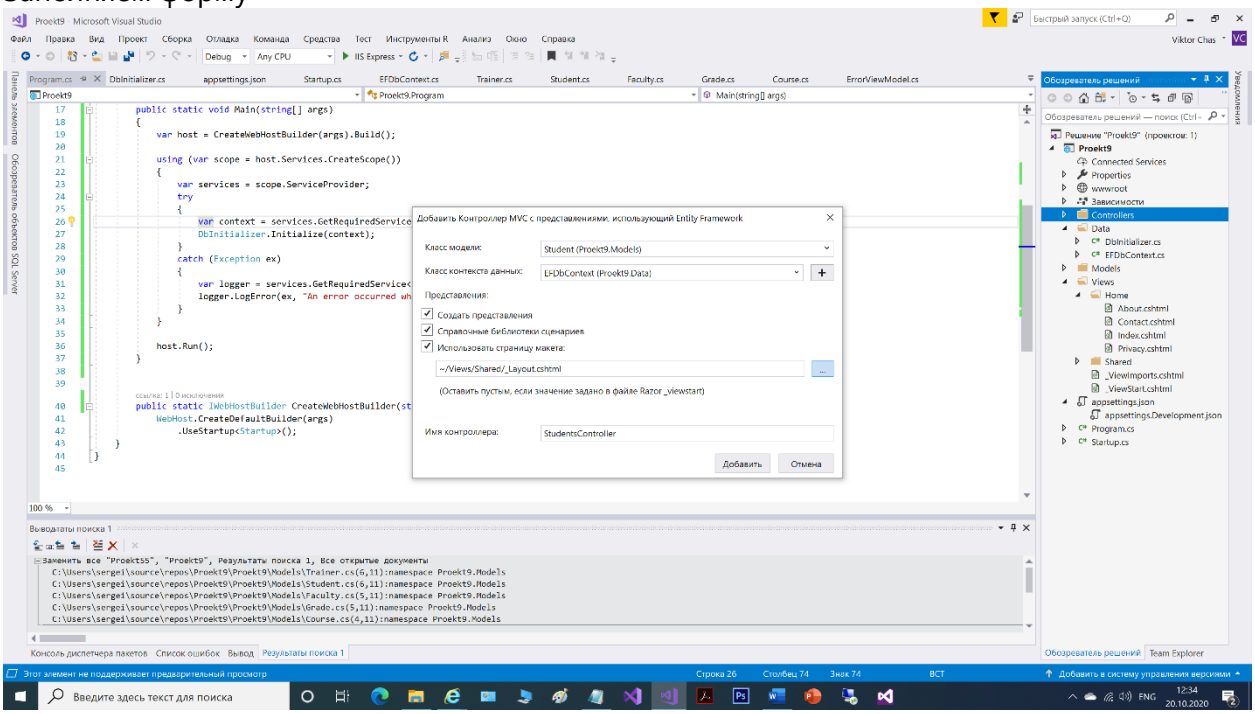




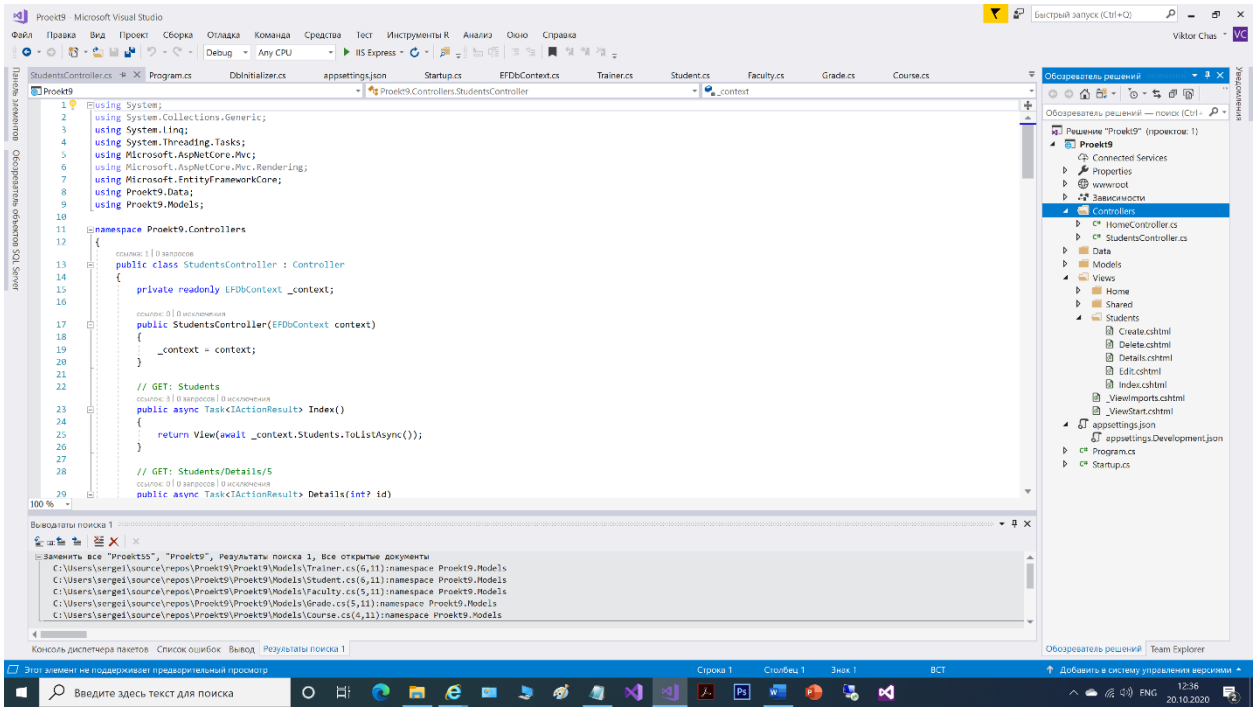
- В диалоговом окне **Добавление шаблона**:
  - Выберите **Контроллер MVC с представлениями, использующий Entity Framework**.
  - Нажмите кнопку **Добавить**. Откроется диалоговое окно **добавления контроллера MVC с представлениями с использованием Entity Framework**.



Заполняем форму



При нажатии кнопки **Добавить** подсистема формирования шаблонов Visual Studio создает файл *StudentsController.cs* и набор представлений (файлы с расширением *.cshtml*), которые будут работать с контроллером.


Proekt9 - Microsoft Visual Studio
Файл Правка Вид Проект Сборка Отладка Команда Средства Тест Инструменты R Анализ Окно Справка
Viktor Chas
Debug
Any CPU
IIS Express
StudentsController.cs
Program.cs
DbInitializer.cs
appsettings.json
Startup.cs
EFDbContext.cs
Trainer.cs
Student.cs
Faculty.cs
Grade.cs
Course.cs
Proekt9
Proekt9.Controllers.StudentsController
_context
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Proekt9.Data;
using Proekt9.Models;
namespace Proekt9.Controllers
{
ссылка: 1 | 0 запросов
public class StudentsController : Controller
{
private readonly EFDbContext _context;
ссылок: 0 | 0 исключения
public StudentsController(EFDbContext context)
{
_context = context;
}
// GET: Students
ссылок: 0 | 0 запросов | 0 исключения
public async Task<IActionResult> Index()
{
return View(await _context.Students.ToListAsync());
}
// GET: Students/Details/5
ссылок: 0 | 0 запросов | 0 исключения
public async Task<IActionResult> Details(int? id)
100 %
Вывод
Результаты поиска 1
Заменить все "Proekt55", "Proekt9", Результаты поиска 1, Все открытые документы
C:\Users\sergei\source\repos\Proekt9\Proekt9\Models\Trainer.cs(6,11):namespace Proekt9.Models
C:\Users\sergei\source\repos\Proekt9\Proekt9\Models\Student.cs(6,11):namespace Proekt9.Models
C:\Users\sergei\source\repos\Proekt9\Proekt9\Models\Faculty.cs(5,11):namespace Proekt9.Models
C:\Users\sergei\source\repos\Proekt9\Proekt9\Models\Grade.cs(5,11):namespace Proekt9.Models
C:\Users\sergei\source\repos\Proekt9\Proekt9\Models\Course.cs(4,11):namespace Proekt9.Models
Консоль диспетчера пакетов
Список ошибок
Вывод
Результаты поиска 1
Обозреватель решений
Решение "Proekt9" (проектов: 1)
Proekt9
Connected Services
Properties
wwwroot
Зависимости
Controllers
HomeController.cs
StudentsController.cs
Data
Models
Views
Home
Shared
Students
Create.cshtml
Delete.cshtml
Details.cshtml
Edit.cshtml
Index.cshtml
_ViewImports.cshtml
_ViewStart.cshtml
appsettings.json
appsettings.Development.json
Program.cs
Startup.cs
Team Explorer
Этот элемент не поддерживает предварительный просмотр
Строка 1
Столбец 1
Знак 1
ВСТ
Добавить в систему управления версиями
Введите здесь текст для поиска
ENG
12:36
20.10.2020