Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Пермский национальный исследовательский**

**политехнический университет‎»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

направление подготовки: 09.03.04 Программная инженерия

**О Т Ч Е Т**

**по учебной практике**

Выполнил студент гр. РИС-19-1б

Хасанов А. Н.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Проверил:

ст. преподаватель Кузнецов Д. Б.

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

**РЕФЕРАТ**

Пояснительная записка 33 с., 6 рис., 5 листингов, 5 источников, 5 приложений

РАЗРАБОТКА САЙТА, WEB-ТЕХНОЛОГИИ, ASP.NET CORE, MVC, ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ

Объектом исследования является процесс работы на производстве.

Целью исследования является разработка и создание web-приложения на платформе ASP.NET Core.

В ходе работы был проведен анализ предметной области, анализ предприятия, установлены задачи проекта, изучены актуальные паттерны проектирования информационных систем, а также Entity Framework.

Для определения метода реализации проекта рассмотрен процесс создания web-приложений.

В результате проведенного исследования разработано web-приложение.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ](#_g9azn5ob3ls) ………………………………………………………………………..4

[1 Анализ предметной области](#_wi5iigthrul7) ……………………………………………………5

[1.1 Место прохождение практики](#_4ug5r4yx74j9) …………………………………………….5

[1.2 Постановка задачи](#_giths9vw8s3e) ………………………………………………………....6

[1.3 Инструменты разработки](#_va0exihovmzn) ………………………………………………….7

[1.4 Подключение БД](#_1hacrlqcmle3) ………………………………………………………….10

[1.5 Ограничение информации для разных пользователей](#_kv70utmwc2yd) ………………....11

[2 Технология реализации](#_65a7q4vt3slq) ………………………………………………………..13

[2.1 Работа с базой данных](#_o3pha5gxyho9) …………………………………………………...13

[2.2 Реализация контроллеров](#_euunkt2ik0tp) ………………………………………………..15

[2.3 Авторизация и аутентификация пользователей](#_sng2j9kgjw37) ………………………...16

[ЗАКЛЮЧЕНИЕ](#_eqt2bn4fjrj6) …………………………………………………………………..19

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_sey275s3nush) …………………………….20

[ПРИЛОЖЕНИЕ А. Классы для работы с дахой данных](#_w1p8tuf7i53l) ……………………...21

[ПРИЛОЖЕНИЕ Б. Класс Startup](#_vqdoqxoade3g) ……………………………………………….26

[ПРИЛОЖЕНИЕ В. Набор классов для данных преподавателе](#_lvjozaqcbo4n).……………....[27](#_lvjozaqcbo4n)

[ПРИЛОЖЕНИЕ Г. Авторизация и аутентификация](#_8p917dx8kn7y) …………………………...30

[ПРИЛОЖЕНИЕ Д. Код иерархии классов паттерна декоратор](#_8trfsudy2al5) ……………....33

# **ВВЕДЕНИЕ**

В данном отчете описан процесс подготовки к участию в проекте компании в рамках информационно-образовательной системы (название проекта не разглашается). Для этого необходимо изучить документацию, отработать предпочитаемые компанией паттерны проектирования информационных систем, такие как внедрение зависимостей, паттерн декоратор и другие. Итогом данного этапа будет web-приложение на базе ASP.NET Core.

ASP.NET (Active Server Pages для .NET) — платформа разработки веб-приложений, в состав которой входят: веб-сервисы, программная инфраструктура, модель программирования, от компании Майкрософт. ASP.NET входит в состав платформы .NET Framework и является развитием более старой технологии Microsoft ASP.[1]

Цель данной учебной практики является реализация web-приложения с помощью платформы ASP.NET Core в рамках подготовки к участию в процессе доработки информационно-программной системы .

Для выполнения заданной цели решим поэтапно следующие задачи:

1. Провести анализ предметной области.
2. Разработать web-приложение.

# 

# **1 А**нализ предметной области

## **1.1 Место прохождение практики**

ООО “ДФС” (Диджитал Фьюче Системс) — российская компания, занимающаяся разработкой и интеграцией бизнес решений. Компания специализируется на:

* проектировании и разработке систем управления, систем поддержки принятия решений, информационно-аналитических и учетных систем;
* внедрении готовых бизнес-приложений и их техническая поддержка;
* управлении данными и глубокой аналитике;
* разработке специализированного программного обеспечения и уникальных заказных решений;
* аутсорсинг.

ДФС решает задачи государственной и корпоративной сфер, подробно указанные в таблице 1. [2]

*Таблица 1 — Задачи решаемые компанией*

| Государственная сфера | Корпоративная сфера |
| --- | --- |
| * Комплексная автоматизация деятельности государственных учреждений * Проектное управление * Управление закупками * Создание информационно-аналитических систем * Создание систем сбора и консолидации отчетности * Создание ситуационных центров | * Мониторинг реализации стратегии и KPI * Оптимизация производственной деятельности * Бизнес-планирование и бюджетирование * Управление инвестиционными программами и проектами * Создание систем сбора и консолидации отчетности * Создание электронного офиса руководителя * Создание информационно-аналитических порталов |
|
|
|
|
|

## 

Стек, применяющийся в проектах ООО «ДФС» представлен в таблице 2.

Таблица 2 —Перечень используемых технологий компании

| DBMS development | Backend web development | Frontend web development |
| --- | --- | --- |
| * PostgreSQL * Oracle Database * Microsoft SQL Server | * Java * C# * .NET Framework | * Angular * HTML * JS * CSS |

Разработка приложения будет вестись на языке C#, т.к. необходимо подготовиться к участию в работе над проектом, основная часть которого будет написана именно на этом языке программирования.

## **1.2 Постановка задачи**

В ходе подготовки необходимо создать демонстрационный проект web-приложения, позволяющего манипулировать данными о курсовых работах студентов вуза. Данная тема выбрана в связи с уже имеющейся на локальном сервере базой данных(БД), логическая модель которой приведена на рисунке 1.

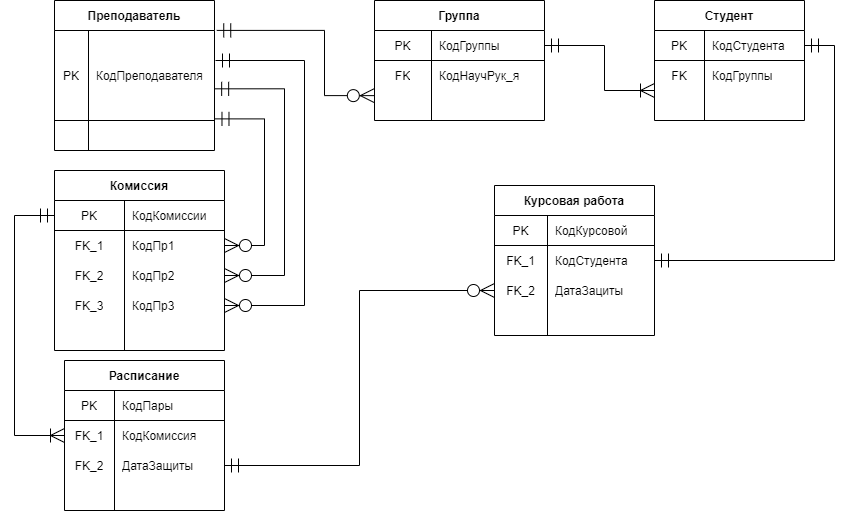


Рисунок 1 — Логическая модель базы данных

Далее приведено краткое описание сущностей БД:

1. **Студент.** Выполняет курсовую работу под наблюдением преподавателя, является частью какой-либо учебной группы.
2. **Преподаватель.** Руководит процессом выполнения курсовой работы студентом путем отметок о прогрессе.
3. **Группа**. Необходима для назначения руководителей (преподавателей) и курсов для больших групп студентов
4. **Курсовая работа**. Сущность содержащая всю информацию о курсовой работе(тему, описание, прогресс выполнения, оценку).
5. **Комиссия**. Сущность, обеспечивающая формирование 3 преподавателей в комиссию.
6. **Расписание**. Хранение доступных дат защиты. В одну дату может быть только одна комиссия.

База данных реализована средствами СУБД PostgreSQL. Она не содержит никаких данных, т.к. является демонстрационной.

Приложения должно позволять просматривать, изменять, удалять и добавлять данные в каждую таблицу базы БД.

Также приложение должно реализовывать авторизацию и аутентификацию пользователей и открывать доступ к разному функционалу в зависимости от роли (студент, преподаватель, администратор) вошедшего в систему.

Главным приоритетом является бэкэнд-разработка, поэтому интерфейс приложения может быть схематичным, простым. По этой же причине о нем не будет подробной информации в данном отчете.

## 1.3 Инструменты разработки

Первым этапом создания приложения будет создания проекта на платформе ASP.NET Core MVC с помощью среды разработки Visual Studio 2019 Preview.

Структура архитектуры MVC (model-view-controller) разделяет приложение на три основных группы компонентов: модели, представлении 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(рисунок 2). Контроллер выбирает представление для отображения пользователю со всеми необходимыми данными модели.[3]

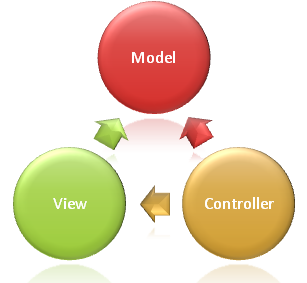


Рисунок 2 — Архитектура MVC

Данная архитектура приложения позволяет отделить бизнес логику, работу с БД и интерфейсом друг от друга. Также она предоставляет необходимые на будущий этапах инструменты. Данное решение позволяет эффективно использовать паттерн внедрения зависимостей (будет описано далее).

После создания проекта ASP.NET Core MVC в нем автоматически создается класс Startup.

Класс Startup является входной точкой в приложение ASP.NET Core. Этот класс производит конфигурацию приложения, настраивает сервисы, которые приложение будет использовать, устанавливает компоненты для обработки запроса к серверу. В ходе создания приложения именно в этом классе будет подключаться различный функционал приложения(например, авторизация).

Для корректной работы приложения необходимо подключить маршрутизацию, указав конечную точку адреса по умолчанию. Для этого и нужен класс Startup(подробно описано в технологии реализации). Все адреса будут состоять из контроллера и действия.

Контроллер - центральное звено в архитектуре ASP.NET Core MVC. При получении запроса система маршрутизации выбирает для обработки запроса нужный контроллер и передает ему данные запроса. Контроллер обрабатывает эти данные и посылает обратно результат обработки. По умолчанию создается контроллер HomeContoller.

В ASP.NET Core MVC контроллер представляет обычный класс на языке C#, который наследуется от абстрактного базового класса Microsoft.AspNetCore.Mvc.Controller.

Каждый контроллер имеет методы, которые можно назвать действиями. Действия контроллера - это публичные методы, которые могут сопоставляться с запросами. Например, стандартный контроллер HomeController содержит метод Index - он имеет модификатор public и поэтому может использоваться для обработки запроса.

Кроме контроллера по умолчанию, необходимо создать свои для работы с каждой моделью и прописать методы для манипуляции данными.

Такая архитектура приложения предусматривает использование паттерна Dependency Injection (DI) (англ. внедрение зависимостей)  *—* стиль настройки объекта, при котором поля объекта задаются внешней сущностью. Другими словами, объекты настраиваются внешними объектами. DI — это альтернатива самонастройке объектов.

Обычно данный паттерн реализуется через интерфейсы (или абстрактные классы) или сервисы и реализующие их классы. Объекты не содержат зависимостей сами по себе, а лишь хранят ссылку на интерфейс, что позволяет передать в конструктор любой из наследуемых классов явно или неявно. В этом и суть паттерна внедрения зависимостей. Далее останется лишь передать сервис в Startup, и приложения будет автоматически отправлять его во все конструкторы, принимающие этот сервис в качестве параметра. Применение данного паттерна позволит в будущем при необходимости заменить бизнес логику. Практическое применение будет описано далее в тексте отчета.

## 1.4 Подключение БД

Для реализации функционала приложения в первую очередь необходимо подключиться к БД. В данном случае не обойтись стандартными инструментами языка C#, так как база создана с помощью PostgreSQL, а не MS SQL Server.

Сделать это позволит Entity Framework — решение для работы с базами данных, которое используется в программировании на языках семейства.NET. Оно позволяет взаимодействовать с СУБД с помощью сущностей (entity), а не таблиц. Также код с использованием EF пишется гораздо быстрее.

Например, работая с базами данных напрямую, разработчик должен беспокоиться о подключении, подготовке SQL и параметров, отправке запросов и транзакций. На Entity Framework всё это делается автоматически — программист же работает непосредственно с сущностями и только говорит EF, что нужно сохранить изменения.[4]

EF будет служить прослойкой между БД и частью приложения, отвечающей за работу с данными(показано на рисунке 3), позволяя сконцентрироваться на написании кода без лишнего обращения к базе через СУБД.

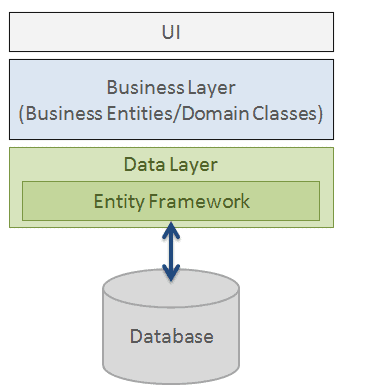


Рисунок 3 — Entity Framework. Связь БД и приложения

Также подключение данного фреймворка позволит отработать навыки использования стандартной C# библиотеки Linq (Language-Integrated Query), что ликвидирует необходимость отвлекаться на проектирование sql-запросов.

## 1.5 Ограничение информации для разных пользователей

Для решения задачи ограничения данных будет применен паттерн декоратор, позволяющий расширять функционал объекта, не изменяя при этом его внутренней логики.

Обычно данная структура реализуется через иерархию интерфейсов или абстрактных классов, устроенную как на рисунке 4.

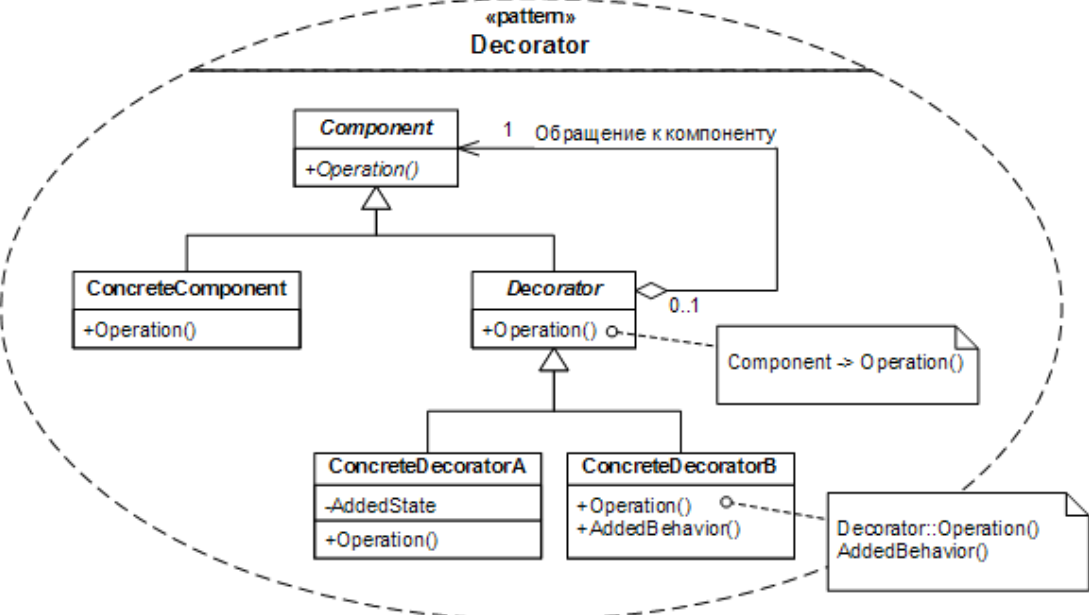


Рисунок 4 — Иерархия паттерна декоратор

Из рисунка наглядно понятно, что паттерн может как расширить уже имеющийся функционал, так и добавить новый(AddedBehavior в классе ConcreteDecoratorB). Также видно, что класс можно обернуть множество раз (если это необходимо), т.к. каждая обертка содержит ссылку на суперкласс.

Создаем класс для извлечения данных из БД, затем классы-декораторы для пользователя администратора, студента, преподавателя.

**Выводы по разделу:** чтобы реализовать web-приложение выбрана архитектура ASP.NET MVC с встроенным паттерном внедрения зависимостей. Для подключения к базе данных выбран NuGet пакет Npgsql.EntityFrameworkCore.PostgreSQL, а для автоматизации процесса переноса БД в приложение Microsoft.EntityFrameworkCore.Tools. Для ограничения информации, доступной пользователям, будет использован паттерн декоратор.

# 

# **2** Технология реализации

## **2.1 Работа с базой данных**

Для работы с базой данных были подключены NuGet пакеты:

* Npgsql.EntityFrameworkCore.PostgreSQL,
* Microsoft.EntityFrameworkCore.Tools,

последний из которых необходим для автоматизации процесса переноса БД в проект с помощью использование специальной команды scaffold-dbcontext. Текст команды приведен в листинге 1.

Листинг 1 — команда переноса моделей БД в проект.

| Scaffold-DbContext "Host=127.0.0.1;Database=coursework;Username=postgres;Password=postgres" Npgsql.EntityFrameworkCore.PostgreSQL -ContextDir Services -Context CourseworkContext -OutputDir Models -Tables student, educator, commission, timetable, term\_paper, study\_group |
| --- |

После ключевого слова в команде последовательно перечислены: строка подключения, содержащая адрес сервера, название БД, имя пользователя, пароль; используемый пакет Nuget; папка Services в которую необходимо поместить класс Coursework Context, основной класс, через который будет производиться обращение к данным; папка Models, в которую необходимо поместить классы-модели, являющиеся представлениями сущностей базы данных.

Результат выполнения данной команды в консоли менеджера пакетов представлен на рисунке 5.

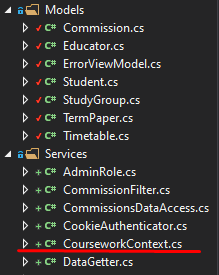
****

Рисунок 5 — Результат выполнения команды scaffold-dbcontext

Также необходимо предусмотрительно создать модели для передачи в представления(интерфейс приложения). Они ничем не будут отличаться от моделей, сгенерированных командой, но это только в нашем случае, так как нам не всегда нужно передавать все данные пользователям.

Для работы с данными необходимо добавить в класс Startup указание, что в качестве источника данных будет использоваться класс CourseworkContext. Это показано на листинге 2.

Листинг 2 — Внедрение зависимости Coursework

services.AddDbContext<CourseworkContext>

(options => options.UseNpgsql(Configuration.GetConnectionString("DefaultConnection")));

Данный код реализует паттерн внедрения зависимостей, т.к. теперь достаточно будет прописать его в конструкторе, и он не будет передаваться вручную, за нас это сделает программа.

Код классов CourseworkContext, StudentView и Student представлен в приложении А, класса Startup в приложении Б.

## 2.2 Реализация контроллеров

На данном этапе нас интересует подключение маршрутизации, которое реализуется как показано в листинге 3.

Листинг 3 — Подключение маршрутизации

app.UseRouting(); //использование маршрутизации

app.UseEndpoints(endpoints =>{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}");

//точка по умолчанию

});

Через класс Startup мы указываем программе, что мы будем использовать маршрутизацию и задаем адрес по умолчанию.

После необходимо создать контроллеры для работы с каждой моделью.

Для примера, создаем контроллер, отвечающий за манипуляцию данными о преподавателях. Назовем его EducatorsController.

Следующим шагом будет прописать действия для добавления, изменения, удаления, просмотра преподавателей. Действия как результат возвращают представление в которое передается модель данных для дальнейшего генерирования HTML разметки и отправки ее браузеру. Также действия могут ничего не возвращать, если нам не нужно отображение данных, например, просто удалить какие-то записи из БД. Вся логика была вынесена из самих методов в отдельный класс EducatorDataAccess для возможности ее замены в будущем.

Реализация данного контроллера и вспомогательного класса EducatorDataAccess представлены в приложении В.

## 2.3 Авторизация и аутентификация пользователей

Подключение авторизации также происходит в классе Startup за это отвечает код представленный в листинге 4.

Листинг 4 — Подключение авторизации и аутентификации

services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme).AddCookie(options =>

{options.LoginPath = new

Microsoft.AspNetCore.Http.PathString

("/Account/LoginStudent");});

...

app.UseAuthentication();

app.UseAuthorization();

Первая часть кода означает, что для хранения данных о пользователе используются файлы Cookie ( небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя). Далее указывается, что каждый запрос на сервер должен сопровождаться авторизацией и аутентификацией пользователя.

Далее необходимо написать контроллер AccountController для авторизации пользователей, добавить в него методы авторизации студентов и преподавателей.

Для сохранения данных в cookie были написаны интерфейс IAuthenticator и реализующий его класс CookieAuthenticator. Данное решение тоже реализует паттерн внедрения зависимостей. В классе Startup следующая строка - services.AddTransient<IAuthenticator, CookieAuthenticator>() - указывает, что во всех конструкторах, где в качестве параметра указан интерфейс IAuthenticator, автоматически будет передан класс CookieAuthenticator. Далее в любой момент будет возможность написать новый класс и передавать уже его.

Контроллер и все интерфейсы/классы связанные с авторизацией пользователей подробно представлены в приложение Г.

Если приложением могут пользоваться разные пользователи, то и уровень доступа к данным у них тоже разные. Так например, администратор видит в список всех существующих в БД студентов, преподаватель - только ту часть, курсовые которых проверяет, студент вообще не имеет доступа к данной информации.

В подобных ситуаций было решено применить паттерн декоратор. Была реализована иерархия классов упрощенная схема которой приведена на рисунке 6. Полный код приведен в приложении Д.

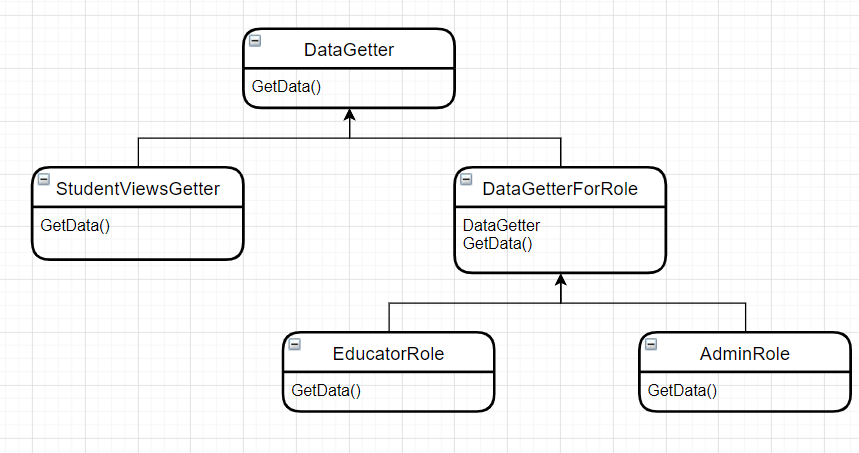


Рисунок 6 — Упрощенная схема реализации паттерна декоратор

Использование данной архитектуры продемонстрировано в листинге 5.

Листинг 5 — Использование декораторов

\_studentGetter = new StudentViewsGetter();

if (userRole==RolesData.Admin) \_studentGetter = new AdminRole(\_studentGetter, \_studentsDataAccess);

if (userRole==RolesData.Educator) \_studentGetter = new EducatorRole(\_studentGetter, \_studentsDataAccess);

return \_studentGetter;

В зависимости от роли пользователя объект \_student Getter оборачивается в соответствующий декоратор. Это позволяет ограничивать данные.

Данный этап был последним в разработке проекта. Весь задуманный функционал был реализован. База данных подключена с помощью EF. Через класс Startup подключена маршрутизация, авторизация, аутентификация, а также реализован паттерн внедрения зависимостей. Для разделения информации для разных пользователей был применен паттерн декоратор.

# 

# ЗАКЛЮЧЕНИЕ

В ходе производственной практики был проведен анализ предметной области, реализован демонстрационный проект web-приложения В ходе разработки были изучены: платформа ASP.NET Core, архитектура MVC, паттерны внедрения зависимостей и декоратор, Entity Framework работы с базой данных. После оценки проекта был получен к разработке информационно-образовательной системы. Цель работы успешно выполнена.

# 

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Adam Freeman ASP.NET MVC 5 с примерами на C# 5.0 для профессионалов, 5-е издание = Pro ASP.NET MVC 5. — М.: «Вильямс», 2014. — 736 с.*
2. *Digital Future Systems [Электронный ресурс]. — Режим доступа:* [*https://dfsystems.ru/*](https://dfsystems.ru/) *(дата обращения: 09.07.2021)*
3. *Общие сведения об ASP.NET Core MVC [Электронный ресурс]. —* [*https://docs.microsoft.com/ru-ru/aspnet/core/mvc/overview?view=aspnetcore-5.0*](https://docs.microsoft.com/ru-ru/aspnet/core/mvc/overview?view=aspnetcore-5.0)*(дата обращения 10.07.21)*
4. *Entity Framework: как быстрее написать код для работы с базой данных [Электронный ресурс]. —* [*https://skillbox.ru/media/code/entity\_framework/*](https://skillbox.ru/media/code/entity_framework/) *(дата обращения 10.07.21)*
5. *Inversion of Control Containers and the Dependency Injection Pattern — [Электронный ресурс].* [*https://www.martinfowler.com/articles/injection.html*](https://www.martinfowler.com/articles/injection.html) *(дата обращения 10.07.21)*

# 

# ПРИЛОЖЕНИЕ А. Классы для работы с дахой данных

Листинг А.1 — Код класса CourseworkContext

public partial class CourseworkContext : DbContext

{

public CourseworkContext(){}

public CourseworkContext(DbContextOptions<CourseworkContext> options)

: base(options) {}

public virtual DbSet<Commission> Commissions { get; set; }

public virtual DbSet<Educator> Educators { get; set; }

public virtual DbSet<Student> Students { get; set; }

public virtual DbSet<StudyGroup> StudyGroups { get; set; }

public virtual DbSet<TermPaper> TermPapers { get; set; }

public virtual DbSet<Timetable> Timetables { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder){}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.HasPostgresEnum(null, "availability", new[] { "не сдано", "сдано", "отклонено", "принято" })

.HasPostgresEnum(null, "work\_status", new[] { "не готово к защите", "готово к защите", "не защищено", "защищено(3)", "защищено(4)", "защищено(5)", "неявка" })

.HasAnnotation("Relational:Collation", "Russian\_Russia.1251");

modelBuilder.Entity<Commission>(entity =>

{

entity.ToTable("commission");

entity.Property(e => e.Id)

.HasColumnName("id")

.UseIdentityAlwaysColumn();

entity.Property(e => e.EducatorIdFirst).HasColumnName("educator\_id\_first");

entity.Property(e => e.EducatorIdSecond).HasColumnName("educator\_id\_second");

entity.Property(e => e.EducatorIdThird).HasColumnName("educator\_id\_third");

entity.HasOne(d => d.EducatorIdFirstNavigation)

.WithMany(p => p.CommissionEducatorIdFirstNavigations)

.HasForeignKey(d => d.EducatorIdFirst)

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("fk1");

entity.HasOne(d => d.EducatorIdSecondNavigation)

.WithMany(p => p.CommissionEducatorIdSecondNavigations)

.HasForeignKey(d => d.EducatorIdSecond)

.HasConstraintName("fk2");

entity.HasOne(d => d.EducatorIdThirdNavigation)

.WithMany(p => p.CommissionEducatorIdThirdNavigations)

.HasForeignKey(d => d.EducatorIdThird)

.HasConstraintName("fk3");

});

modelBuilder.Entity<Educator>(entity =>

{

entity.ToTable("educator");

entity.Property(e => e.Id)

.HasColumnName("id")

.UseIdentityAlwaysColumn();

entity.Property(e => e.CallNumber)

.IsRequired()

.HasMaxLength(11)

.HasColumnName("call\_number");

entity.Property(e => e.Name)

.IsRequired()

.HasMaxLength(50)

.HasColumnName("name");

entity.Property(e => e.Password)

.IsRequired()

.HasMaxLength(8)

.HasColumnName("password");

entity.Property(e => e.Patronymic)

.IsRequired()

.HasMaxLength(50)

.HasColumnName("patronymic");

entity.Property(e => e.Surname)

.IsRequired()

.HasMaxLength(50)

.HasColumnName("surname");

});

modelBuilder.Entity<Student>(entity =>

{

entity.ToTable("student");

entity.Property(e => e.Id)

.HasColumnName("id")

.UseIdentityAlwaysColumn();

entity.Property(e => e.Name)

.IsRequired()

.HasColumnType("character varying")

.HasColumnName("name");

entity.Property(e => e.Patronymic)

.HasColumnType("character varying")

.HasColumnName("patronymic");

entity.Property(e => e.StudyGroupName)

.IsRequired()

.HasMaxLength(10)

.HasColumnName("study\_group\_name");

entity.Property(e => e.Surname)

.IsRequired()

.HasColumnType("character varying")

.HasColumnName("surname");

entity.HasOne(d => d.StudyGroupNameNavigation)

.WithMany(p => p.Students)

.HasForeignKey(d => d.StudyGroupName)

.HasConstraintName("fk1");

});

modelBuilder.Entity<StudyGroup>(entity =>

{

entity.HasKey(e => e.Name)

.HasName("study\_group\_pkey");

entity.ToTable("study\_group");

entity.Property(e => e.Name)

.HasMaxLength(10)

.HasColumnName("name");

entity.Property(e => e.EducatorId).HasColumnName("educator\_id");

entity.HasOne(d => d.Educator)

.WithMany(p => p.StudyGroups)

.HasForeignKey(d => d.EducatorId)

.OnDelete(DeleteBehavior.SetNull)

.HasConstraintName("fk1");

});

modelBuilder.Entity<TermPaper>(entity =>

{

entity.ToTable("term\_paper");

entity.Property(e => e.Id)

.HasColumnName("id")

.UseIdentityAlwaysColumn();

entity.Property(e => e.Description)

.IsRequired()

.HasColumnName("description")

.HasDefaultValueSql("'не указано'::text");

entity.Property(e => e.PresentasionDate)

.HasColumnType("date")

.HasColumnName("presentasion\_date");

entity.Property(e => e.StudentId).HasColumnName("student\_id");

entity.Property(e => e.Subject)

.IsRequired()

.HasColumnName("subject")

.HasDefaultValueSql("'не указано'::text");

entity.HasOne(d => d.PresentasionDateNavigation)

.WithMany(p => p.TermPapers)

.HasForeignKey(d => d.PresentasionDate)

.OnDelete(DeleteBehavior.SetNull)

.HasConstraintName("fk2");

entity.HasOne(d => d.Student)

.WithMany(p => p.TermPapers)

.HasForeignKey(d => d.StudentId)

.HasConstraintName("fk1");

});

modelBuilder.Entity<Timetable>(entity =>

{

entity.HasKey(e => e.PresentionDate)

.HasName("timetable\_pkey");

entity.ToTable("timetable");

entity.Property(e => e.PresentionDate)

.HasColumnType("date")

.HasColumnName("presention\_date");

entity.Property(e => e.CommissionId).HasColumnName("commission\_id");

entity.HasOne(d => d.Commission)

.WithMany(p => p.Timetables)

.HasForeignKey(d => d.CommissionId)

.OnDelete(DeleteBehavior.SetNull)

.HasConstraintName("fk1");

});

OnModelCreatingPartial(modelBuilder);

}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}

Листинг А.2 — Код класса StudentView

public class StudentView

{

public int Id { get; set; }

public string Name { get; set; }

public string Surname { get; set; }

public string Patronymic { get; set; }

public string StudyGroupName { get; set; }

}

Листинг А.3 — Код класса Student

public partial class Student

{

public Student()

{

TermPapers = new HashSet<TermPaper>();

}

public int Id { get; set; }

public string Name { get; set; }

public string Surname { get; set; }

public string Patronymic { get; set; }

public string StudyGroupName { get; set; }

public virtual StudyGroup StudyGroupNameNavigation { get; set; }

public virtual ICollection<TermPaper> TermPapers { get; set; }

}

# 

# ПРИЛОЖЕНИЕ Б. **Класс Startup**

Листинг Б.1 — Код класса Startup

| public class Startup  {  public Startup(IConfiguration configuration)  { Configuration = configuration;}  public IConfiguration Configuration { get; }  public void ConfigureServices(IServiceCollection services)  {  services.AddControllersWithViews();  services.AddDbContext<CourseworkContext>(options => options.UseNpgsql(Configuration.GetConnectionString("DefaultConnection"))); services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme).AddCookie(options => {options.LoginPath = new Microsoft.AspNetCore.Http.PathString("/Account/LoginStudent");  });  services.AddTransient<StudentsDataAccess>();  services.AddTransient<EducatorsDataAccess>();  services.AddTransient<CommissionsDataAccess>();  services.AddTransient<TimetableDataAccess>();  services.AddTransient<TermPapersDataAccess>();  services.AddTransient<StudyGroupsDataAccess>();  services.AddTransient<IAuthenticator, CookieAuthenticator>();  services.AddTransient<ITermPaperDataFilter, TermPaperFilter>();  services.AddTransient<IStudentDataFilter, StudentFilter>();  services.AddTransient<ICommissionDataFilter, CommissionFilter>();}  public void Configure(IApplicationBuilder app, IWebHostEnvironment env){  if (env.IsDevelopment()) {  app.UseDeveloperExceptionPage();  }  else{  app.UseExceptionHandler("/Home/Error");  app.UseHsts();  }  app.UseHttpsRedirection();  app.UseStaticFiles();  app.UseRouting();  app.UseAuthentication();  app.UseAuthorization();  app.UseEndpoints(endpoints =>{  endpoints.MapControllerRoute(  name: "default",  pattern: "{controller=Home}/{action=Index}");  });}  } |
| --- |

# ПРИЛОЖЕНИЕ В. Набор классов дляданных преподавателей

Листинг В.1 — Код класса EducatorController

[Authorize(Roles = "admin")]

public class EducatorController : Controller

{

private readonly EducatorsDataAccess \_educatorDataAccess;

public EducatorController(EducatorsDataAccess dataAccess){

\_educatorDataAccess = dataAccess;}

public IActionResult ShowData(){

return View(\_educatorDataAccess.GetAll());

}

[Route("Educator/Update/{id:int}")]

[HttpGet]

public IActionResult Update(int id){

EducatorView educator = \_educatorDataAccess.GetById(id);

if (educator != null) return View(educator);

return NotFound();

}

[HttpPost]

public IActionResult Update(EducatorView educator){

if (\_educatorDataAccess.Update(educator))

return RedirectToAction("ShowData");

return View(educator);

}

public IActionResult Insert(){

return View();

}

[HttpPost]

public IActionResult Insert(EducatorView educator)

{

\_educatorDataAccess.Add(educator);

return RedirectToAction("ShowData");

}

[Route("Educator/Delete/{id:int}")]

[HttpGet]

public IActionResult Delete(int id)

{

if (\_educatorDataAccess.Delete(id))

return RedirectToAction("ShowData");

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

Листинг В.2 — Код класса EducatorDataAccess

public class EducatorsDataAccess

{

private readonly CourseworkContext \_db;

public EducatorsDataAccess(CourseworkContext context)

{

\_db = context;

}

public List<EducatorView> GetAll()

{

List<EducatorView> resultEducators = new List<EducatorView>();

IEnumerable<Educator> educators = \_db.Educators.ToList();

foreach (var educator in educators)

resultEducators.Add(new EducatorView{

Id = educator.Id,

Name = educator.Name,

Surname = educator.Surname,

Patronymic = educator.Patronymic,

CallNumber = educator.CallNumber,

Password = educator.Password

});

return resultEducators;

}

public EducatorView GetById(int id)

{

var educator = \_db.Educators.SingleOrDefault(ed => ed.Id == id);

return new EducatorView{

Id = educator.Id,

Name = educator.Name,

Surname = educator.Surname,

Patronymic = educator.Patronymic,

CallNumber = educator.CallNumber,

Password = educator.Password

};

}

public void Add(EducatorView educator)

{

\_db.Educators.Add(

new Educator

{

Name = educator.Name,

Surname = educator.Surname,

Password = educator.Password,

Patronymic = educator.Patronymic,

CallNumber = educator.CallNumber

});

\_db.SaveChanges();

}

public bool Update(EducatorView UpdatedEducator)

{

Educator newEducator = \_db.Educators.SingleOrDefault(educator => educator.Id == UpdatedEducator.Id);

if (newEducator != null)

{

newEducator.Name = UpdatedEducator.Name;

newEducator.Surname = UpdatedEducator.Surname;

newEducator.Password = UpdatedEducator.Password;

newEducator.Patronymic = UpdatedEducator.Patronymic;

newEducator.CallNumber = UpdatedEducator.CallNumber;

\_db.Entry(newEducator).State = EntityState.Modified;

\_db.SaveChanges();

return true;

}

return false;

}

public bool Delete(int id)

{

Educator educatorToDelete = \_db.Educators.SingleOrDefault(educator => educator.Id == id);

if (educatorToDelete != null)

{

\_db.Educators.Remove(educatorToDelete);

\_db.SaveChanges();

return true;

}

return false;

}

public bool CheckLoginData(int id, string password)

{

return \_db.Educators.Any(ed => ed.Id == id && ed.Password == password);

}

}

# ПРИЛОЖЕНИЕ Г. **Авторизация и аутентификация**

Листинг Г.1 — Код класса AccountController

public class AccountController : Controller

{

private readonly IAuthenticator \_authenticator;

private readonly EducatorsDataAccess \_edecatorDataAccess;

private readonly StudentsDataAccess \_studentDataAccess;

public AccountController(IAuthenticator authenticator, EducatorsDataAccess edDataAccess, StudentsDataAccess stDataAccess)

{

\_authenticator = authenticator;

\_edecatorDataAccess = edDataAccess;

\_studentDataAccess = stDataAccess;

}

[HttpGet]

public IActionResult LoginEducator()

{

if (User.Identity.IsAuthenticated)

return RedirectToAction("Index", "Home");

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> LoginEducator(ViewLoginEducator model)

{

if (ModelState.IsValid)

{

if (\_edecatorDataAccess.CheckLoginData(model.Id, model.Password))

{

await \_authenticator.Authenticate(model.Id, RolesData.Educator, this.HttpContext);

return RedirectToAction("ShowData", "Student");

}

if(model.Id == 12345678 && model.Password == "12345678")

{

await \_authenticator.Authenticate(model.Id, RolesData.Admin, this.HttpContext);

return RedirectToAction("ShowData", "Student");

}

ModelState.AddModelError("", "Некорректные логин и(или) пароль");

}

return View(model);

}

[HttpGet]

public IActionResult LoginStudent()

{

if (User.Identity.IsAuthenticated)

return RedirectToAction("Index", "Home");

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> LoginStudent(ViewLoginStudent model)

{

if (ModelState.IsValid)

{

if (\_studentDataAccess.CheckLoginData(model.Id))

{

await \_authenticator.Authenticate(model.Id, RolesData.Student, this.HttpContext);

return RedirectToAction("ShowData", "TermPaper");

}

ModelState.AddModelError("", "Некорректные логин и(или) пароль");

}

return View(model);

}

public async Task<IActionResult> Logout()

{

await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

return RedirectToAction("Index", "Home");

}

}

Листинг Г.2 — Код интерфейса IAuthenticator

public interface IAuthenticator

{

public Task Authenticate(int loginId, string role, HttpContext context);

}

Листинг Г.3 — Код класса CookieAuthenticator

public class CookieAuthenticator : IAuthenticator

{

public async Task Authenticate(int loginId, string role, HttpContext context )

{

var claims = new List<Claim>

{

new Claim(ClaimsIdentity.DefaultNameClaimType, loginId.ToString()),

new Claim(ClaimsIdentity.DefaultRoleClaimType, role)

};

ClaimsIdentity identity = new ClaimsIdentity(claims, "ApplicationCookie", ClaimsIdentity.DefaultNameClaimType, ClaimsIdentity.DefaultRoleClaimType);

await context.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(identity));

}

}

# ПРИЛОЖЕНИЕ Д. **Код иерархии классов паттерна декоратор**

Листинг Д.1 — Код класса DateGetter

public abstract class DataGetter

{

public abstract List<StudentView> GetData(string userName);

}

Листинг Д.2 — Код класса StudentViewsGetter

public class StudentViewsGetter : DataGetter

{

public override List<StudentView> GetData(string name)

{

return new List<StudentView>();

}

}

Листинг Д.3 — Код класса DataGetterForRole

public abstract class DataGetterForRole : DataGetter

{

protected DataGetter dataSource;

protected readonly StudentsDataAccess \_studentsDataAccess;

public DataGetterForRole(DataGetter source, StudentsDataAccess dataAccess)

{

dataSource = source;

\_studentsDataAccess = dataAccess;

}

}

Листинг Д.4 — Код класса EducatorRole

public class EducatorRole : DataGetterForRole

{

public EducatorRole(DataGetter dataSource, StudentsDataAccess dataAccess)

: base(dataSource, dataAccess)

{ }

public override List<StudentView> GetData(string userName)

{

return dataSource.GetData(userName)

.Union(\_studentsDataAccess.GetAllForEducator(userName)).ToList();

}

}

Листинг Д.5 — Код класса AdminRole

public class AdminRole : DataGetterForRole

{

public AdminRole(DataGetter dataSource, StudentsDataAccess dataAccess)

: base(dataSource, dataAccess)

{ }

public override List<StudentView> GetData(string userName)

{

Debug.WriteLine("Вы просматриваете студентов. Пользователь Админ :)");

return dataSource.GetData(userName).Union(\_studentsDataAccess.GetAll()).ToList();

}

}