**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«**КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА ИНФОРМАЦИОННЫХ СИСТЕМ

Направление: 09.03.02 Информационные системы и технологии

Профиль: Информационные системы в образовании

КУРСОВАЯ РАБОТА

**Создание веб-приложения для оптимизации администрирования студенческих концертов в КСК "УНИКС"**

Студент 3 курса

Группы 09-063

"\_\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.     \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Р.В. Красноперов)

Научный руководитель

канд. пед. наук, доцент

"\_\_\_"\_\_\_\_\_\_\_\_\_ 2023 г. \_\_\_\_\_\_\_\_\_\_\_\_ (Ч.Б. Миннегалиева)

Заведующий кафедрой

канд. физ.-мат. наук, доцент

"\_\_\_"\_\_\_\_\_\_\_\_\_ 2023 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Ф.М. Гафаров)

Казань, 2023 год

Содержание

[Введение 3](#_Toc135245406)

[Теоретический обзор 5](#_Toc135245407)

[Разработка приложения 9](#_Toc135245408)

[Создание проекта 9](#_Toc135245409)

[Дизайн проекта 9](#_Toc135245410)

[Работа с базой данных 10](#_Toc135245411)

[Создание Razor-страниц 14](#_Toc135245412)

[Главная страница 14](#_Toc135245413)

[Расписание залов 16](#_Toc135245414)

[Страница коллективов 17](#_Toc135245415)

[Страница обратной связи 18](#_Toc135245416)

[Регистрация пользователя 19](#_Toc135245417)

[Страница входа 20](#_Toc135245418)

[Страница профиля 22](#_Toc135245419)

[Страница профиля администратора 23](#_Toc135245420)

[Заключение 26](#_Toc135245421)

[Список литературы 27](#_Toc135245422)

[ПРИЛОЖЕНИЕ 1 29](#_Toc135245423)

[ПРИЛОЖЕНИЕ 2 30](#_Toc135245424)

Введение

В настоящее время организация студенческих концертов и мероприятий является важной составляющей академической среды и способом поддержки талантливых студентов. Но ведь перед участием в самом мероприятии сперва о нем нужно узнать. В данный момент информационная система студклуба в рамках КСК Уникс работает только посредствам пересылки информации и сообщений среди чатов руководителей. Те в свою очередь доносят информацию до своих коллективов. Чтобы избежать этой волокиты и утери информации из-за человеческого фактора, я решил разработать веб-приложение, которое решит все текущие неудобства. Сайт будет предоставлять актуальную информацию о последних новостях и предстоящих мероприятиях, а также предлагать удобные инструменты для взаимодействия между студентами и администрацией.

Целью данной работы является создание функционального и привлекательного информационного сайта для проведения студенческих концертов, который будет учитывать все потребностям студентов и давать возможность легко и просто взаимодействовать участникам мероприятий и администрацией. Для достижения этой цели я ставлю перед собой следующие задачи:

1. Изучение основ ASP.NET Razor Pages: провести обзор технологии ASP.NET Razor Pages, которая позволяет разрабатывать динамические веб-страницы с помощью языка программирования C#.
2. Разработка пользовательского интерфейса: отрисовка и последующая верстка привлекательного и интуитивно понятного пользовательского интерфейса для сайта, учитывая современные тенденции веб-дизайна и обеспечивая удобную навигацию и взаимодействие с функциональностью сайта.
3. Реализация основных функций: отображение последних новостей и мероприятий, расписание залов, список студенческих коллективов, а также функциональность обратной связи с администрацией.
4. Разработка системы авторизации и аутентификации: будет реализована система авторизации и аутентификации пользователей, чтобы обеспечить безопасный доступ к личным данным и предоставить различные роли и права доступа для пользователей и администраторов.

Я уверен, что успешная разработка информационного сайта для проведения студенческих концертов будет способствовать улучшению коммуникации и организации мероприятий, повышению интереса студентов к участию в них и облегчению работы администрации.

Теоретический обзор

**ASP.NET Razor Pages** является мощным фреймворком для разработки веб-приложений, основанных на платформе ASP.NET. Он предлагает удобный и эффективный способ создания динамических веб-страниц с использованием языка программирования C#. Razor Pages предлагает подход Model-View-Controller (MVC), который позволяет разделить логику, представление и данные приложения, упрощая разработку и обеспечивая лучшую организацию кода.

Razor Pages предлагает удобные шаблоны для создания страниц и компонентов. Он предоставляет интегрированные инструменты для работы с моделями данных, формами, маршрутизацией и обработкой событий. Кроме того, Razor Pages предлагает простой синтаксис, который упрощает чтение и понимание кода, и предоставляет богатые возможности для создания динамического и интерактивного пользовательского интерфейса.

**Аутентификация и авторизация** являются важными аспектами разработки веб-приложений, особенно когда речь идет о защите конфиденциальных данных и обеспечении безопасного доступа к функциональности приложения. Аутентификация подтверждает личность пользователя, а авторизация определяет его права и разрешения для доступа к определенным ресурсам или функциям.

ASP.NET Razor Pages предоставляет интегрированные инструменты для реализации аутентификации и авторизации. Это включает встроенные методы и классы для управления пользователями, регистрации, входа в систему и управления правами доступа. Разработчики могут использовать атрибуты авторизации, фильтры авторизации и другие инструменты для определения доступа к различным частям приложения на основе ролей и разрешений.

**Работа с базой данных** является важной частью разработки веб-приложений, поскольку они часто требуют хранения и извлечения данных. В ASP.NET Razor Pages разработчики могут использовать различные подходы для работы с базами данных, включая Entity Framework.

Entity Framework является ORM (Object-Relational Mapping) фреймворком, который предоставляет удобные средства для взаимодействия с базой данных. С его помощью можно создавать модели данных, выполнять операции CRUD (создание, чтение, обновление, удаление), осуществлять запросы и обеспечивать связи между таблицами.

Entity Framework поддерживает различные базы данных, включая SQL Server, MySQL, PostgreSQL и другие. Он предоставляет мощные средства миграции базы данных, что облегчает развертывание и обновление схемы базы данных.

**Хороший веб-дизайн** и умелая верстка играют ключевую роль в создании привлекательного и функционального пользовательского интерфейса. При разработке информационного сайта для проведения студенческих концертов необходимо учесть следующие аспекты веб-дизайна, а также использование HTML, CSS и JavaScript:

1. HTML (HyperText Markup Language) является основным языком разметки, используемым для создания структуры и содержимого веб-страниц. Верстка сайта должна быть четкой и семантически правильной, с использованием соответствующих HTML-элементов для размещения различных компонентов, таких как заголовки, абзацы, списки и таблицы.
2. CSS (Cascading Style Sheets) используется для задания внешнего вида и стилизации веб-страниц. С помощью CSS можно управлять цветами, шрифтами, размерами и расположением элементов, создавая единый и привлекательный дизайн. Также возможно использование CSS-фреймворков, таких как Bootstrap, для более быстрой и удобной разработки.
3. JavaScript является языком программирования, который позволяет добавлять интерактивность и динамичность на веб-страницы. С помощью JavaScript можно создавать анимации, выполнять валидацию форм, обрабатывать события и обеспечивать взаимодействие с пользователем без перезагрузки страницы.

Принципы хорошего веб-дизайна также следует учитывать при разработке информационного сайта для проведения студенческих концертов:

1. Простота и понятность: Сайт должен быть интуитивно понятным и легким в использовании для пользователя. Чистый и минималистичный дизайн поможет сосредоточиться на главной информации и функциональности.
2. Консистентность: Дизайн элементов должен быть последовательным и однородным по всему сайту, чтобы создать единое визуальное впечатление и обеспечить простоту навигации.
3. Цветовая гармония: Выбор цветовой схемы должен быть гармоничным и соответствовать целям и атмосфере сайта. Цвета могут использоваться для поддержки визуальной идентификации и передачи настроения.
4. Отзывчивый дизайн: Сайт должен быть адаптивным и отзывчивым, чтобы корректно отображаться на различных устройствах и экранах. Это важно для обеспечения удобства использования и доступности сайта для всех пользователей.
5. Визуальные акценты: Использование визуальных элементов, таких как изображения, иконки, шрифты и графика, может помочь выделить важную информацию и привлечь внимание пользователя.

При разработке дизайна и верстки информационного сайта для проведения студенческих концертов следует стремиться к привлекательному, функциональному и пользовательски ориентированному интерфейсу. Использование правильной структуры HTML, эффективного CSS и интерактивных возможностей JavaScript, соблюдение принципов хорошего веб-дизайна и адаптивный дизайн помогут создать удобный и привлекательный пользовательский опыт на сайте.

Разработка приложения

Создание проекта

Первым делом нужно будет создать сам проект. Работа произведена в среде программирования JetBrains Rider, имеющей понятный UI, поэтому с этим справиться даже новичок в этой области.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Дизайн проекта

Проект создан - начало положено. Но что именно будет в нем содержаться? Какая именно информация уже понятно, но что насчет UI приложения? Как оно будет выглядеть? Тут на помощь нам приходит такой инструмент для дизайна как Figma. Всегда в нынешних реалиях frontend разработчикам предоставляется макет сайта, чтобы у них было конкретное ТЗ. Мы же в данной работе являемся full-stack разработчиками, так что всю-всю работу над проектом предстоит сделать единолично.

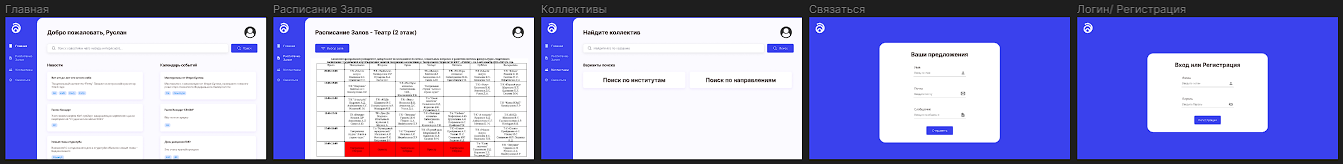
Заходим на веб ресурс figma.com и начинаем работу над нашим дизайном. Все удобство в том, что в общем доступе есть сделанные другими пользователями макеты. Ищем ту базу, которая нам понравится. Затем беремся редактировать ее под свои нужды. После редактуры главная страница сайта выглядит вот так:

Изображение выглядит как текст, программное обеспечение, Шрифт, число

Автоматически созданное описание

Слева у нас расположен общий Layout сайта. Он будет отображаться на всех главных страницах.

Создаем макеты остальных страниц по схожему сценарию. Теперь будет намного проще делать продукт, ведь не надо будет задумываться куда какой элемент ставить – все и так уже перед глазами. Общий проект figma выглядит вот так:



Работа с базой данных

Теперь мы имеем представление как именно будет выглядеть наше веб-приложение. Но что насчет информации, которая будет отображаться на страницах? Она же наверняка должна где-то храниться. Да, так оно и есть – будут использоваться базы данных. Мы можем вручную прописывать запросы добавления удаления каких-то записей, но по мне это будет слишком сложно, да и sql код очень тяжело отладить в проекте, так как он просто не будет показывать никаких ошибок. Поэтому, очень удобно и правильно будет использовать уже готовое решение, а именно Entity Framework Core (далее EF Core). Установим его в наш проект и подключим его в качестве сервиса в главном файле нашей программы – Program.cs:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

В программу EF Core мы подключили. Теперь озаботимся самими данными, которые будут храниться в нашей базе данных.

Для всех сущностей нашего веб-сайта от новостей, до пользователя нужно создать отдельный класс – модель.

Вот список всех моих моделей в проекте, а также пример описания одной из моделей:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Очень внимательные читатели могут заметить, что в нашей модели прописано поле типа List. Да, полностью соглашусь, что в одном поле базы данных не может храниться целый список, но на то мы и подключили EF Core, чтобы упрощать нам всю работу с хранилищем информации.

Вообще, создаваемая связь в базе данных может быть трех типов: **one2one, one2many, many2many**. Последнее является самым сложным, но и самым интересным. Мы просто объявляем в обоих связанных между собой моделях список объектов, состоящих из друг друга, а EF Core уже создаст дополнительную третью таблицу, где будет хранить связь между этими двумя сущностями по их ключам. Выглядит это примерно вот так: (таблица новостей, дополнительная таблица для связи, таблица тегов)

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Хорошо, все типы моделей мы создали, даже связали между собой. Теперь нужно создать класс контроллер всей нашей работы EF Core. В нем мы прописываем все нужны методы. Например, в конструкторе я прописал проверку наличия самой базы данных, так же прописал базовые записи при создании таблицы с помощью метода OnModelCreating, ну и под конец объявил сами DbSet’ы – наши будущие таблицы в базе данных:

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

На этом работа с базами данных закончена, мы создали модели данных, создали класс для работы с ними, ну и подключили все это к нашему главному файлу Program.cs. Теперь перейдем к самому интересному – backend и frontend разработке

Создание Razor-страниц

Как я говорил раньше, у нас есть некий Layout сайта – это некая подложка, где мы будем располагать основной контент наших страниц. Так же наш Layout файл содержит в себе sidebar для навигации по сайту. В плане кода там ничего сложного нет, поэтому пойдем дальше.

Главная страница

Первым делом займемся разработкой нашей главной страницы – страницы новостей и мероприятий.

В принципе, страница Razor имеет в себе два главных метода – Onget и Onpost. Первый отвечает за получаемые данные во время загрузки страницы, второй за отправляемые данные со страницы.

Изображение выглядит как снимок экрана, текст, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

В методе OnGet мы получаем:

1. Cписок новостей, список возможных тегов из нашей базы данных
2. Картинку профиля нашего пользователя и его имя из файлов Cookie (О файлах Cookie, откуда они берутся и что они делают будет немного попозже)

Посмотрим на front часть нашей страницы – как видите, в ней мы так же можем использовать С# код для отображения всего списка записей:

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Вот так выглядит полностью готовая главная страница нашего сайта:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Расписание залов

Следующим шагом была реализована страница расписания коллективов. Отличие от предыдущей страницы заключается в том, что тут мной было принято решения отображать расписание выбранного зала с помощью языка JavaScript (далее JS). Это сделано для того, чтобы пользователь не ждал подгрузки страницы при выборе нового зала, а вся новая информация появлялась сразу при нажатии на нужный элемент. JS как бы подгружает все нужные ему объекты во время компиляции программы, поэтому никакой задержки тут не будет. Ниже представлен JS код:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Сама страница расписания выглядит вот так:

Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описание

Страница коллективов

Следящей страницей в проекте является страница коллективов – тут пользователь может увидеть все зарегистрированные администратором коллективы. Реализация кодом тут похожа на реализацию главной страницы, поэтому ничего концептуально нового мы тут не увидим. Вот так выглядит страница коллективов:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

Страница обратной связи

Первая страница с выключенным Layout – страница обратной связи, тут все посетители сайта могут оставить свое сообщение администратору, может быть некие пожелания, может предложения об улучшении сайта, может какая-то новая инициатива – все зависит от пользователя. С помощью уже знакомого метода OnPost мы отправляем в базу данных сообщение пользователя:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Сама страница обратной связи выглядит вот так:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Регистрация пользователя

С основным функционалом сайта мы познакомились, но что если наш пользователь хочет зарегистрироваться в нашей системе для дальнейшего ею пользования? Тут на помощь нам приходит страница регистрации. На ней пользователь может выбрать свое отображаемое имя, логин, пароль, свои коллективы и даже выбрать собственный аватар, который генерируется рандомно в JS с помощью API от Dicebear, а затем сохраняется в базе данных в качестве ссылки.

Также, пароль пользователя, перед отправкой в базу данных хешируется для последующей безопасности данных пользователя. Код хеширования выглядит вот так:

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Следует упомянуть, что при выборе коллективов был реализован сторонний multiselect-dropdown, который и выглядит намного лучше, и функционально он более понятный.

Ниже приведен код генерации аватара пользователя (весь код multiselect-dropdown невозможно прикрепить из-за его масштабности – 260 строк):

Изображение выглядит как снимок экрана, текст, Шрифт

Автоматически созданное описание

Сама страница регистрации выглядит вот так:

Изображение выглядит как текст, снимок экрана, Операционная система, программное обеспечение

Автоматически созданное описание

Отлично, теперь мы зарегистрированы в системе. Осталось только войти и пользоваться всеми удобствами сервиса

Страница входа

Визуально страница входа пользователя в систему выглядит очень просто, однако серверная реализация в разы сложнее чем на других страницах. Все дело в том, что именно тут мы будем создавать файлы Cookie, которые потом будут храниться в оперативной памяти нашего браузера для доступа к его личной страничке пользователя. Так же файлы Cookie будут использоваться на главной странице для приветствия пользователя и для отображения его аватара на кнопке перехода в профиль.

Ниже приведен код создания этих файлов куки и добавления их в HTTPContetx:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Следует вспомнить, что на странице регистрации мы хешировали пароль, и теперь в базе данных он выглядит примерно вот так:



Пришло время его расхешировать. Для этого мы введенный пользователем пароль при попытке входа опять прохешируем и сравним с паролем того аккаунта, логин которого введет наш пользователь:

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

В этом участке кода вы можете увидеть незнакомый вам код – ModelState.AddModalError. Это **валидация** нашей модели.

Именно тут наша валидация на странице входа покажет сообщение о неправильном логине и пароле только при таковом событии. Например, и на странице обратной связи была реализована валидация модели лишь для проверки, чтобы пользователь не отправил пустое сообщение. Очень важная вещь, если мы хотим уведомить пользователя о том, что он пытается сделать что-то не то.

Ниже представлена страница входа пользователя с нашей валидацией:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Страница профиля

Отлично, мы вошли в систему после пройденной регистрации. Теперь главная страница для нас выглядит вот так:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Как видите, и система нас приветствует, и кнопка профиля теперь отображает наше имя и иконку профиля. Все благодаря вышеупомянутым файлам Cookie. Давайте перейдем в страницу нашего профиля.

Страница профиля обычного пользователя содержит в себе аватар, информацию о его логине, отображаемом имени и дате регистрации и самое главное – информацию о его коллективах:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

Сверху реализована кнопка выхода пользователя из системы. После ее нажатия из оперативной памяти браузера будут удалены все Cookies, отвечающие за аутентификацию пользователя. Код выхода отображен ниже:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Страница профиля администратора

Если мы будем входить в систему под логином администратора, то получим чутка другую страницу профиля. Во-первых, у администратора вместо коллективов будут обращаться те самые обращения пользователей из вкладки обратной связи, во-вторых, у администратора появиться дополнительные кнопки:

Изображение выглядит как текст, Человеческое лицо, снимок экрана, очки

Автоматически созданное описание

Все эти кнопки переадресуют нас на отдельную страницу для добавления нового контента. Может появиться логичный вопрос: что если обычный, но хитрый пользователь как-то узнает адрес этих страниц? Неужели он сможет на ровне с администратором добавлять новые записи на наш сайт?

Ответ – нет конечно, все это предусмотрено на этапе авторизации. Во время этого процесса мы создаем Policy – некий договор, который проверят роль нашего пользователя. Если у нашего пользователя роль админа, то он сможет перейти на эти страницы, если же у него роль обычного пользователя, то доступ к этим страницам для него будет запрещен.

Код Авторизации, а затем доступ к ограниченным страницам прописан в файле Program.cs и прикреплен ниже:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Сами же страницы добавления новых записей выглядят вот так:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Заключение

В процессе выполнения курсовой работы было разработано веб-приложение, для информирования студентов о предстоящих событиях. Для достижения результата были выполнены следующие цели:

1. Была изучена технология asp net Razor Pages
2. Был разработан авторский дизайн интерфейса приложения
3. Реализован основной функционал сайта
4. Разработана система авторизации и аутентификации пользователя на сайте

В процессе разработки использовалась сама система разработки asp net Razor Pages, для создания макета был использован инструмент Figma, для работы с базами данных был использован Entity Framework Core, для верстки были использованы HTML, CSS, JS, для создания картинок профиля была использована DiceBear Api, для серверной части приложения был использован язык C#.

Список литературы

1. Введение в Razor Pages [Электронный ресурс]. – Режим Доступа: <https://learn.microsoft.com/ru-ru/aspnet/core/razor-pages/?view=aspnetcore-7.0&tabs=visual-studio>, свободный (Дата обращения 17.05.23)
2. W3 School [Электронный ресурс]. – Режим Доступа: https://www.w3schools.com, свободный (Дата обращения 17.05.23)
3. Создание веб-приложения с помощью ASP.NET Core Razor Pages: пошаговое руководство [Электронный ресурс]. – Режим Доступа: https://www.youtube.com/watch?v=8LXCxHzEIhc&t=1719s, свободный (Дата обращения 17.05.23)
4. [Many-to-many relationship and multi select in asp.net core](https://stackoverflow.com/questions/50169617/many-to-many-relationship-and-multi-select-in-asp-net-core) [Электронный ресурс]. – Режим Доступа: https://stackoverflow.com/questions/50169617/many-to-many-relationship-and-multi-select-in-asp-net-core, свободный (Дата обращения 17.05.23)
5. Configure Many-to-Many Relationships in Entity Framework Core[Электронный ресурс]. –https://www.entityframeworktutorial.net/efcore/configure-many-to-many-relationship-in-ef-core.aspx, свободный (Дата обращения 17.05.23)
6. [multiselect-dropdown](https://github.com/admirhodzic/multiselect-dropdown) [Электронный ресурс]. – Режим Доступа: https://github.com/admirhodzic/multiselect-dropdown, свободный (Дата обращения 17.05.23)
7. Policy-based authorization in ASP.NET Core[Электронный ресурс]. – Режим Доступа:https://learn.microsoft.com/enus/aspnet/core/security/authorization/policies?view=aspnetcore-7.0, свободный (Дата обращения 17.05.23)
8. Handler Methods in Razor Pages [Электронный ресурс]. – Режим Доступа: https://www.learnrazorpages.com/razor-pages/handler-methods, свободный (Дата обращения 17.05.23)
9. [ASP.NET Identity's default Password Hasher](https://stackoverflow.com/questions/20621950/asp-net-identitys-default-password-hasher-how-does-it-work-and-is-it-secure) [Электронный ресурс]. – Режим Доступа: https://stackoverflow.com/questions/20621950/asp-net-identitys-default-password-hasher-how-does-it-work-and-is-it-secure, свободный (Дата обращения 17.05.23)

ПРИЛОЖЕНИЕ 1

**Код основного шаблона html (Layout)**

<!DOCTYPE html>  
<html lang="ru">  
<head>  
 <meta charset="utf-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>@ViewData["Title"]</title>  
 <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />  
 <**link** rel="stylesheet" **href**="~/css/site.css" **asp-append-version**="true" />  
 <**link** rel="stylesheet" **href**="~/l4Razor.styles.css" **asp-append-version**="true" />  
  
 <**link** rel="stylesheet" **href**="~/css/header.css" **asp-append-version**="true">  
 <**link** rel="stylesheet" **href**="~/css/style.css" **asp-append-version**="true">  
 <**link** rel="stylesheet" **href**="~/css/sidebar.css" **asp-append-version**="true">  
 <**link** rel="stylesheet" **href**="~/css/news.css" **asp-append-version**="true">  
   
 <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css" rel="stylesheet"/>  
   
 <link rel="preconnect" href="https://fonts.googleapis.com">  
 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>  
 <link href="https://fonts.googleapis.com/css2?family=Inter&display=swap" rel="stylesheet">  
   
 <link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=Material+Symbols+Outlined:opsz,wght,FILL,GRAD@20..48,100..700,0..1,-50..200" />  
  
</head>  
<body>  
 <div class="Sidebar">  
  
 <div class="Logo"><**a asp-page** = "/Index"><img src="css/images/Vector.png" alt="Website Logo"></**a**></div>  
  
 <ul class = "SidebarContent">  
  
 <li class="SidebarText SidebarTextSelected "><**a asp-page**="/Index"><span class="material-symbols-outlined">inventory\_2</span>Главная</**a**></li>  
 <li class="SidebarText SidebarTextNotSelected "><**a asp-page**="/Schedule"><span class="material-symbols-outlined">description</span>Расписание Залов</**a**></li>  
 <li class="SidebarText SidebarTextNotSelected "><**a asp-page**="/Collectives"><span class="material-symbols-outlined">group</span>Коллективы</**a**></li>  
 <li class="SidebarText SidebarTextNotSelected "><**a asp-page**="/ContactUs"><span class="material-symbols-outlined">cloud</span>Связаться</**a**></li>  
  
 </ul>  
   
 </div>  
  
 <div class="MainContent">  
 @RenderBody()  
 </div>  
  
</body>  
</html>

ПРИЛОЖЕНИЕ 2

**Таблица стилей news.css**

.NewsAndCalendar{  
 display: flex;  
 flex-direction: row;  
 justify-content: space-between;  
 height: 100%;  
 width: 100%;  
}  
  
.NewsBlock{  
 display: flex;  
 flex-direction: column;  
 width: 48%;  
 gap: 24px;  
}  
  
.NewsTitle{  
 padding-top: 24px;  
  
 font-family: 'Inter';  
 font-style: normal;  
 font-weight: 700;  
 font-size: 22px;  
 line-height: 130%;  
  
 display: flex;  
 align-items: center;  
  
 color: #403937;  
}  
  
.NewsCard{  
 display: flex;  
 flex-direction: column;  
 justify-content: center;  
  
 background: #FFFFFF;  
 box-shadow: 0px 4px 16px #EAE2FD;  
 border-radius: 8px;  
 padding: 24px;  
 gap:15px;  
}  
  
.NewsCard:*last-child*{  
 margin-bottom: 100px;  
}  
  
.NewsCardTitle{  
 font-family: 'Inter';  
 font-style: normal;  
 font-weight: 700;  
 font-size: 16px;  
 line-height: 130%;  
  
 display: flex;  
 align-items: center;  
  
 color: #403937;  
}  
  
.NewsCardTitleAndData{  
 display: flex;  
 flex-direction: row;  
 justify-content: space-between;  
}  
  
.NewsCardData{  
 font-family: 'Inter';  
 font-style: normal;  
 font-weight: 500;  
 font-size: 15px;  
 line-height: 130%;  
 font-style: italic;  
 color: #a9a9a9;  
}  
  
.NewsCardDescription{  
 font-family: 'Inter';  
 font-style: normal;  
 font-weight: 500;  
 font-size: 15px;  
 line-height: 130%;  
  
 display: flex;  
 align-items: center;  
}  
  
.NewsCardTags{  
 display: flex;  
 flex-direction: row;  
}  
  
.NewsCardTag{  
 margin-right: min(8px, 2%);  
  
 padding: 4px 8px;  
  
 background: #D6E2FF;  
 border-radius: 8px;  
  
 font-family: 'Inter';  
 font-style: normal;  
 font-weight: 500;  
 font-size: 12px;  
 line-height: 130%;  
  
 display: flex;  
 align-items: center;  
  
 color: #3A97ED;