**ГОУ СПО «Тираспольский техникум информатики и права»**

|  |  |  |
| --- | --- | --- |
| **РАССМОТРЕНО**  на заседании ЦМК дисциплин  профессионального цикла  Протокол № \_\_\_\_ от «\_\_»\_\_\_\_\_\_\_\_\_20\_\_\_ г.  Председатель ЦМК \_\_\_\_\_\_ Белоус Е.Г. |  | **СОГЛАСОВАНО**  Заместитель руководителя  по учебной работе  \_\_\_\_\_\_\_\_\_\_\_ Гончарук Л.И. «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_\_\_ г. |

**Индивидуальное задание**

**на практику \_\_\_ПП.01.01 ПРОИЗВОДСТВЕННАЯ ПРАКТИКА\_\_\_\_\_** индекс практики, вид практики

по профессиональному модулю (междисциплинарному курсу)

**\_\_\_\_ПМ.01.Разработка модулей программного обеспечения для компьютерных систем\_\_\_**

\_\_\_\_\_\_\_МДК 01.01. Разработка программных модулей\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_МДК 01.02. Поддержка и тестирование программных модулей\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ф.И.О. студента                           Афтентьев Эдуард                                                               ,

Учебная группа \_\_\_\_\_\_\_315\_\_\_\_\_\_\_ курс \_\_\_\_\_III\_\_\_\_\_\_\_\_

Специальность  \_\_\_**2.09.02.07 Информационные системы и программирование**

код и наименование специальности

Объем 72 часов с «12» декабря  2025 г. по «26» декабря 2025 г.

|  |  |  |
| --- | --- | --- |
| № задания | Содержание задания | ОК, ПК |
| 2 | Выполнить задания по проектированию, доработке сайта на основе шаблона MVC «Платформа для записи на курсы и отслеживания прогресса обучения»  Разработать меню пользователя, в котором будут ссылки на корзину/архив и профиль пользователя. Добавить на страницу с каталогом поиск. Добавить панель администратора.  Меню должно открываться по нажатию на иконку с картинкой пользователя вверху шапки сайта. Добавить возможность редактирования профиля пользователя.  Для корзины/архива необходимо реализовать функционал добавления элементов из каталога и впоследствии возможность удаление элемента из корзины/каталога. | ПК 1.1-  ПК 1.5 |
| 1. Разработать Frontend с использованием CSS и JS:меню пользователя, вкладку с корзиной/архивом, профиль пользователя, панель администратора.  2. Доработать базу данных.  3. Доработать проект с моделями  4. Доработать проект с сервисами, реализующими бизнес-логику приложения.  5. Реализовать контролеры для новых страниц. |

Руководитель практики Т.С.Подсекина.

# Календарно-тематический план

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№**  **п/п** | **Дата** | **Описание выполненной работы** | **Количество часов** | **Оценка и подпись руководителя практики** |
|  | 31.10.2025 | Роль практического обучения при формировании компетенций обучающихся. Инструктаж по технике безопасности и правилам поведения на учебной практике. | 2 |  |
|  | 31.10.2025 | Создание проекта на основе шаблона MVC. | 2 |  |
|  | 31.10.2025 | Оформление отчетной документации. | 2 |  |
|  | 01.11.2025 | Использование Git для контроля версий. | 2 |  |
|  | 01.11.2025 | Использование Git для контроля версий. | 2 |  |
|  | 01.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 03.11.2025 | Определение структуры MVC-приложения, взаимодействие моделей, контроллеров и представлений. | 2 |  |
|  | 03.11.2025 | Определение структуры MVC-приложения, взаимодействие моделей, контроллеров и представлений. | 2 |  |
|  | 03.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 04.11.2025 | Frontend разработка основная страница: написание шапки сайта, секции «О нас». | 2 |  |
|  | 04.11.2025 | Frontend разработка основная страница: написание шапки сайта, секции «О нас». | 2 |  |
|  | 04.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 05.11.2025 | Frontend разработка основная страница: написание секций «Услуги», «Контакты». | 2 |  |
|  | 05.11.2025 | Frontend разработка основная страница: написание секций «Услуги», «Контакты». | 2 |  |
|  | 05.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 06.11.2025 | Frontend разработка основная страница: написание форм входа/регистрации. | 2 |  |
|  | 06.11.2025 | Frontend разработка основная страница: написание форм входа/регистрации. | 2 |  |
|  | 06.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 07.11.2025 | Frontend разработка основная страница: написание секции «Написать сообщение» и подвала сайта. | 2 |  |
|  | 07.11.2025 | Frontend разработка основная страница: написание секции «Написать сообщение» и подвала сайта. | 2 |  |
|  | 07.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 08.11.2025 | Использование fetch-запросов для обновления данных без перезагрузки страницы. | 2 |  |
|  | 08.11.2025 | Использование fetch-запросов для обновления данных без перезагрузки страницы. | 2 |  |
|  | 08.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 10.11.2025 | Создание адаптивного интерфейса с использованием медиазапросов CSS. | 2 |  |
|  | 10.11.2025 | Создание адаптивного интерфейса с использованием медиазапросов CSS. | 2 |  |
|  | 10.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 11.11.2025 | Знакомство с PostgreSQL. | 2 |  |
|  | 11.11.2025 | Знакомство с PostgreSQL. | 2 |  |
|  | 11.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 12.11.2025 | Создание базы данных в СУБД PostgreSQL. | 2 |  |
|  | 12.11.2025 | Создание базы данных в СУБД PostgreSQL. | 2 |  |
|  | 12.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 13.11.2025 | Подключение PostgreSQL через appsettings.json. Определение сущностей и их атрибутов в C# для базы данных с использованием Entity Framework. | 2 |  |
|  | 13.11.2025 | Подключение PostgreSQL через appsettings.json. Определение сущностей и их атрибутов в C# для базы данных с использованием Entity Framework. | 2 |  |
|  | 13.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 14.11.2025 | Разработка базовых операций CRUD для всех сущностей. Добавление моделей. | 2 |  |
|  | 14.11.2025 | Разработка базовых операций CRUD для всех сущностей. Добавление моделей. | 2 |  |
|  | 14.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 15.11.2025 | Разработка сервисов для авторизации и аутентификации. | 2 |  |
|  | 15.11.2025 | Разработка сервисов для авторизации и аутентификации. | 2 |  |
|  | 15.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 17.11.2025 | Доработка сервисов для авторизации и аутентификации | 2 |  |
|  | 17.11.2025 | Доработка сервисов для авторизации и аутентификации | 2 |  |
|  | 17.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 18.11.2025 | Подготовка проекта к использованию ClaimsIdentity | 2 |  |
|  | 18.11.2025 | Подготовка проекта к использованию ClaimsIdentity | 2 |  |
|  | 18.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 19.11.2025 | Подключение HttpContext.U  ser и ClaimsIdentity | 2 |  |
|  | 19.11.2025 | Подключение HttpContext.User и ClaimsIdentity | 2 |  |
|  | 19.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 20.11.2025 | Использование C#, FluentValidation и JavaScript для валидации вводимых данных при авторизации и регистрации. | 2 |  |
|  | 20.11.2025 | Использование C#, FluentValidation и JavaScript для валидации вводимых данных при авторизации и регистрации. | 2 |  |
|  | 20.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 21.11.2025 | Использование C#, FluentValidation и JavaScript для валидации вводимых данных для других объектов. | 2 |  |
|  | 21.11.2025 | Использование C#, FluentValidation и JavaScript для валидации вводимых данных для других объектов. | 2 |  |
|  | 21.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 22.11.2025 | Использование Google аккаунта в качестве сервера отправки сообщений. | 2 |  |
|  | 22.11.2025 | Использование Google аккаунта в качестве сервера отправки сообщений | 2 |  |
|  | 22.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 24.11.2025 | Реализация проверки подлинности пользователя при регистрации. | 2 |  |
|  | 24.11.2025 | Реализация проверки подлинности пользователя при регистрации. | 2 |  |
|  | 24.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 25.11.2025 | Настройка Google Cloud для авторизации. | 2 |  |
|  | 25.11.2025 | Настройка Google Cloud для авторизации. | 2 |  |
|  | 25.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 26.11.2025 | Реализация авторизации Google. | 2 |  |
|  | 26.11.2025 | Реализация авторизации Google. | 2 |  |
|  | 26.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 27.11.2025 | Доработка авторизации Google. | 2 |  |
|  | 27.11.2025 | Доработка авторизации Google. | 2 |  |
|  | 27.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 28.11.2025 | Frontend разработка страницы со странами/каталог. | 2 |  |
|  | 28.11.2025 | Frontend разработка страницы со странами/каталог. | 2 |  |
|  | 28.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 29.11.2025 | Frontend доработка страницы со странами/каталог. | 2 |  |
|  | 29.11.2025 | Frontend доработка страницы со странами/каталог. | 2 |  |
|  | 29.11.2025 | Оформление отчетной документации. | 2 |  |
|  | 01.12.2025 | Frontend разработка страницы с элементами каталога: секция отображения. | 2 |  |
|  | 01.12.2025 | Frontend разработка страницы с элементами каталога: секция отображения. | 2 |  |
|  | 01.12.2025 | Оформление отчетной документации. | 2 |  |
|  | 02.12.2025 | Frontend доработка страницы с элементами каталога: секция отображения. | 2 |  |
|  | 02.12.2025 | Frontend доработка страницы с элементами каталога: секция отображения. | 2 |  |
|  | 02.12.2025 | Оформление отчетной документации. | 2 |  |
|  | 03.12.2025 | Frontend разработка страницы с элементами: реализация фильтров и сортировки данных. | 2 |  |
|  | 03.12.2025 | Frontend разработка страницы с элементами: реализация фильтров и сортировки данных. | 2 |  |
|  | 03.12.2025 | Оформление отчетной документации. | 2 |  |
|  | 04.12.2025 | Frontend доработка страницы с элементами: реализация фильтров и сортировки данных. | 2 |  |
|  | 04.12.2025 | Frontend доработка страницы с элементами: реализация фильтров и сортировки данных. | 2 |  |
|  | 04.12.2025 | Оформление отчетной документации. | 2 |  |
|  | 05.12.2025 | Разработка сервисов для реализации фильтров на странице. | 2 |  |
|  | 05.12.2025 | Разработка сервисов для реализации фильтров на странице. | 2 |  |
|  | 05.12.2025 | Оформление отчетной документации. | 2 |  |
|  | 06.12.2025 | Доработка сервисов для реализации фильтров на странице. | 2 |  |
|  | 06.12.2025 | Доработка сервисов для реализации фильтров на странице. | 2 |  |
|  | 06.12.2025 | Оформление отчетной документации. | 2 |  |
|  | 08.12.2025 | Frontend разработка страница с конкретным элементом | 2 |  |
|  | 08.12.2025 | Frontend разработка страница с конкретным элементом | 2 |  |
|  | 08.12.2025 | Оформление отчетной документации. | 2 |  |
|  | 09.12.2025 | Frontend доработка страница с конкретным элементом | 2 |  |
|  | 09.12.2025 | Frontend доработка страница с конкретным элементом | 2 |  |
|  | 09.12.2025 | Оформление отчетной документации. | 2 |  |
|  | 10.12.2025 | Оформление индивидуального проекта к защите работы. | 2 |  |
|  | 10.12.2025 | Оформление индивидуального проекта к защите работы. | 2 |  |
|  | 10.12.2025 | Оформление отчетной документации. | 2 |  |
|  | 11.12.2025 | Оформление презентации к защите работы. | 2 |  |
|  | 11.12.2025 | Оформление презентации к защите работы. | 2 |  |
|  | 11.12.2025 | Дифференцированный зачет. | 2 |  |

Содержание объемов выполненных работ подтверждаю

Руководитель практики

от техникума (предприятия) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_С.Г. Патеров\_\_/

подпись Ф.И.О.

# 31.10.2025

**Тема**: Роль практического обучения при формировании компетенций обучающихся. Инструктаж по технике безопасности и правилам поведения на учебной практике. Создание проекта на основе шаблона MVC. Оформление отчетной документации.

**Цель:** Изучить технику безопасности и создать начальный MVC-проект.

**Ход работы**:

**Главная роль** учебной практики — закрепить теоретические знания и научиться применять их на практике. Во время практики студент осваивает навыки работы за ПК, учится разрабатывать и программировать реальные проекты. Перед началом работы проводится инструктаж: нужно соблюдать правильную осанку, держать монитор на расстоянии 50–70 см от глаз, делать перерывы для отдыха и бережно обращаться с техникой.

Заходим в Visual Studio 2022 и выбрать пункт «Создание *проекта»* как показано на рисунке 1.

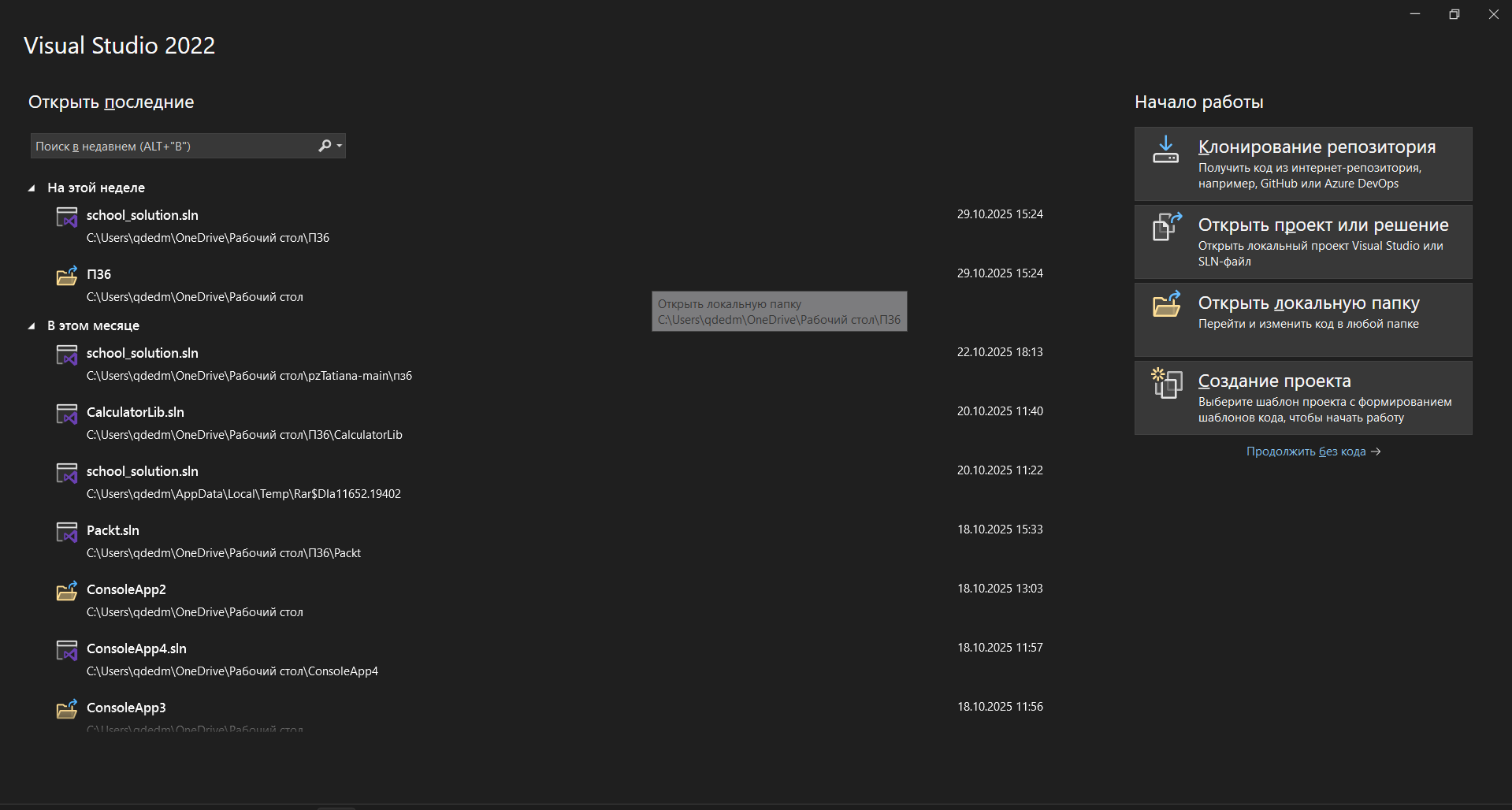


Рисунок 1 – Создание проекта

1. После выбираем необходимые параметры, показанные на рисунке 2 и 3 создаем проект с названием *«Aftentev».*

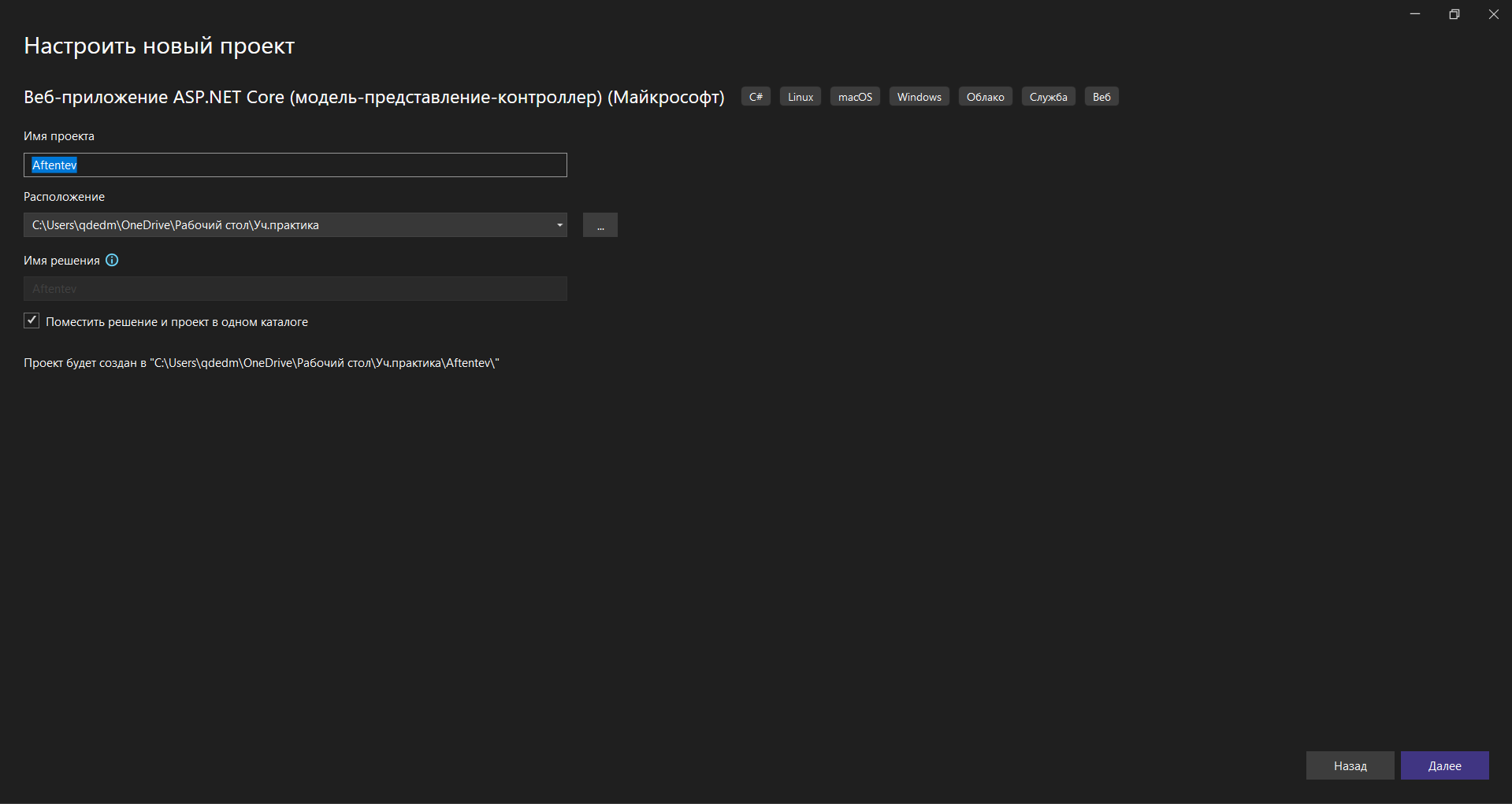


Рисунок 2

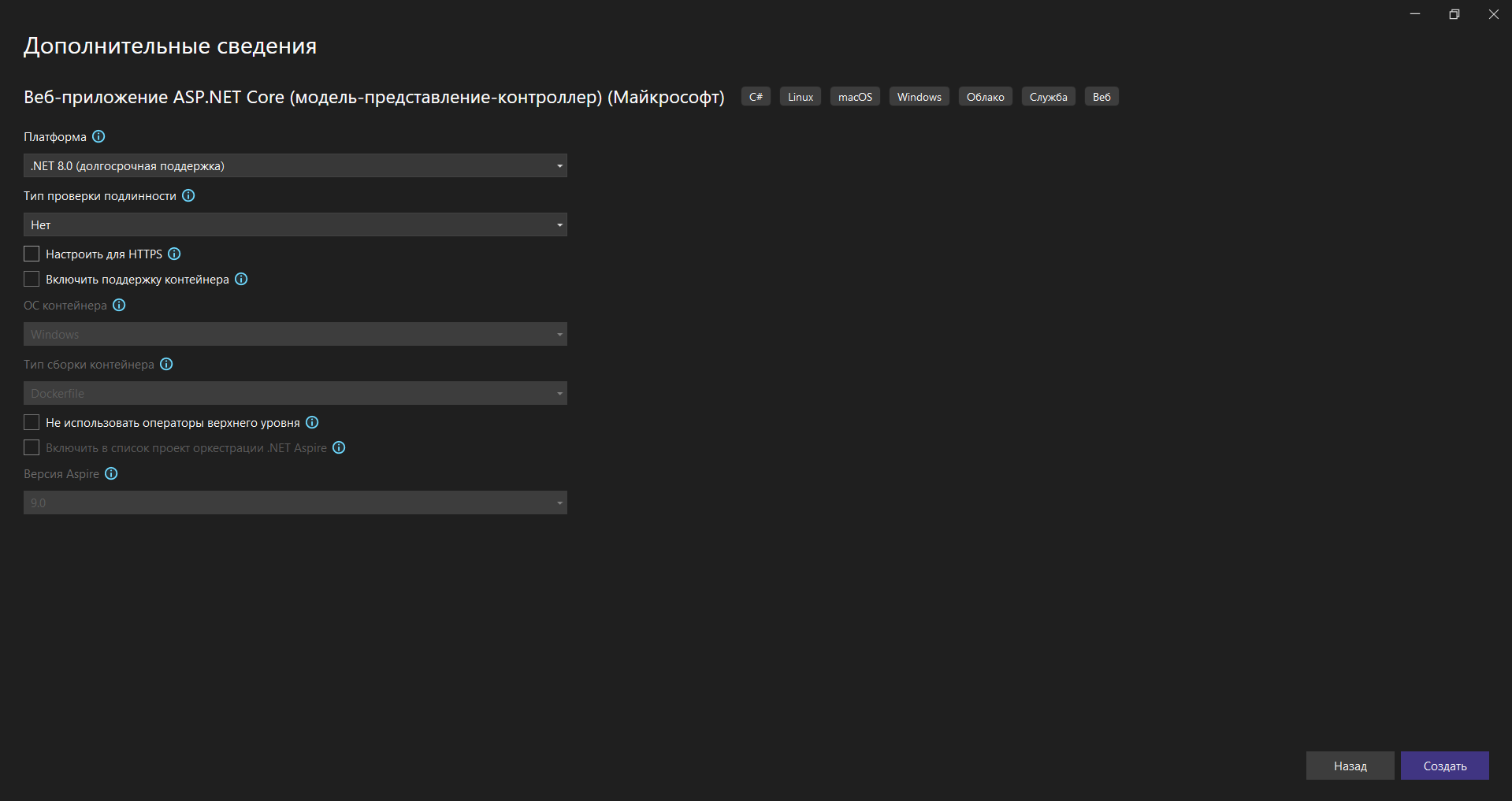


Рисунок 3 – Шаблоны

1. В обозревателе решений на рисунке 4 показана структура данного шаблона на начальном этапе разработки.

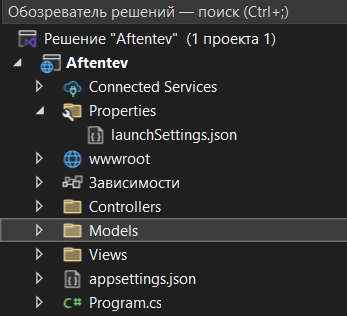


Рисунок 4 – Обозреватель решений

В данном проекте представлены основные папки, с которыми мы будем работать.

Папка *wwwroot*, показанная на рисунке 5 – содержит в себе папку для стилей проекта, папку, содержащую скрипты для динамического взаимодействия с проектом, и папку с дополнительными файлами, которые могут упростить работу если знать, как их использовать. В данном проекте мы будем использовать чистый *JavaScript.*

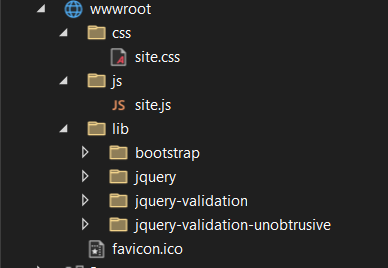
**

Рисунок 5 – папка *wwwroot*

Папка *Controllers*, показанная на рисунке 6 – обрабатывает входящие запросы. Во фреймворке это может заключаться в определении конкретных *URL*, на которые попадает пользователь при переходе по ссылке или при нажатии кнопки.



Рисунок 6 – папка *Controllers*

Папка *Models*, показанная на рисунке 7 – отвечает за данные, которые хранятся и обрабатываются на сервере.



Рисунок 7 – папка *Models*

Папка *Views* – это *HTML*-шаблон, который возвращает сервер после обработки запроса. В ней представлены следующие решения:

* *\_Layout.cshtml*

Файлы *layout* или мастер-страницы позволяют определить единый шаблон для страниц *Razor* и применяются для создания единообразного, унифицированного вида приложения. В этом файле рассмотрели такие теги как: <*head*> - предназначен для хранения служебной информации страницы. <*header*> - элемент, который нужен для отделения вводного содержимого или навигации от основного содержимого целой страницы или секционного блока. Тег <*footer*> - служит для создания футера сайта или же подвала сайта. Это нижняя часть сайта, в которой может находиться какая-нибудь контактная информация, нижнее меню, ссылки или же что-то в этом духе. Так же включает в себя различные методы, один из которых - *@RenderBody() -*  метод, который будет вставлять содержимое страниц *Razor*, которые используют данную мастер-страницу. В итоге мы сможем легко установить для всех страниц единообразный стиль оформления.

* *\_ViewImports.cshtml*

Строки в этом файле предназначены для глобального подключения пространств имен и дополнительных возможностей, которые применяются ко всем представлениям в вашем проекте *ASP.NET Core MVC*.

* *\_ViewStart.cshtml.*

Используется для указания файла макета (*layout*), который будет применен к текущему представлению.

**Вывод**: создали проект на основе *MVC* в *Rider,* ознакомившись со структурой: *wwwroot* (стили и скрипты), *Controllers* (обработка запросов), *Models* (данные) и *Views* (представления). Изучена мастер-страница *\_Layout.cshtml*, позволяющая задавать единый шаблон с общими элементами интерфейса, включая меню и стили, что упрощает управление страницами.

# 1.11.2025

**Тема:** Использование Git для контроля версий. Оформление отчетной документации.

**Цель:** Освоить работу с Git и оформить часть отчётной документации.

**Ход работы:**

1. Создание репозитория Git:

Сначала я создал удаленный репозиторий на сайте GitHub. Я зашел в свой профиль, выбрал раздел «Your repositories» и нажал кнопку «New». Я назвал проект «EduMaster», чтобы он был доступен онлайн.



Рисунок 8 – GitHub Desktop

1. Создание и клонирование Локальных копий:

Для работы с файлами на ПК я использовал два метода:

1. Создание локального репозитория: в программе GitHub Desktop я нажал «Add», указал имя репозитория, описание и локальный путь, после чего нажал «Create repository».

2. Клонирование существующего репозитория: Для скачивания файлов с GitHub я выбрал «File» > «Clone repository». Я выбрал нужный удаленный репозиторий, указал путь для сохранения и нажал «Clone».

1. Фиксация и Синхронизация Изменений

После добавления или изменения файлов в локальной папке я выполнил следующие действия:

1. Я зафиксировал изменения (сделал коммит), добавив сообщение и нажав «Commit to main».
2. Чтобы отправить изменения на удаленный сервер GitHub, я использовал кнопку «Publish repository» (или «Push origin» для последующих синхронизаций).

**Вывод:**

Я успешно освоил полный цикл работы с GitHub Desktop: создание удаленных и локальных репозиториев, клонирование проектов, фиксацию (коммиты) и публикацию (пуш) изменений.

# 3.11.2025

**Тема:** Определение структуры MVC-приложения, взаимодействие моделей, контроллеров и представлений. Применение принципов Чистой архитектуры для структурирования проекта на ASP.NET MVC (C#).

**Цель:** Разобраться в структуре MVC-приложения и описать её в отчёте.

**Ход работы:**

**Ключевые принципы и шаги:**

Правило зависимостей: Установлено, что зависимости в коде должны указывать исключительно внутрь (от внешних слоев к внутренним).

Определение структурных элементов: Изучены роли Сущностей (Entities), Сценариев (Use Cases) и Интерфейс-адаптеров.

Создание ядра Domain: Создан проект, содержащий Сущности (основные объекты) и Интерфейсы репозиториев (ITaskRepository), которые служат границами для внешних слоев.

Слой Application: Создан проект для реализации Сервисов приложения (бизнес-сценариев). Этот слой зависит только от Domain и использует его интерфейсы для выполнения операций.

Слой Infrastructure: Создан как Интерфейс-адаптер. Здесь реализован конкретный доступ к данным (например, SqlTaskRepository через Entity Framework Core), что позволяет изолировать Domain от технологий.

1. Слой Presentation (ASP.NET MVC):
2. Создан MVC-проект, который является самым внешним слоем.
3. Контроллеры принимают HTTP-запросы и используют сервисы из слоя Application.
4. Представления (Views) отвечают за вывод.
5. Модели (View Models) используются для передачи данных между Контроллером и Представлением.

Настройка DI (Dependency Injection): В ASP.NET настроена Инверсия Управления для связывания интерфейсов (ITaskRepository из Domain) с их реализациями (SqlTaskRepository из Infrastructure), что обеспечивает соблюдение Правила зависимостей.

**Вывод:**

В результате работы была успешно спроектирована и реализована многослойная структура MVC-приложения на C# с соблюдением принципов Чистой архитектуры. Основной результат — изоляция бизнес-логики (Domain) от внешних факторов (Infrastructure, Presentation), что обеспечивает высокую гибкость и тестируемость проекта.

# 4.11.2025

**Тема:**  Frontend разработка основная страница: написание шапки сайта, секции «О нас». Оформление отчетной документации.

**Цель:** Разработать шапку сайта и секцию «О нас» на главной странице.

**Ход работы:**

Разработан и оформлен хедер сайта, включающий логотип и навигационное меню. Благодаря фиксированному позиционированию он остаётся видимым при прокрутке, обеспечивая удобную навигацию. Стили на CSS с использованием Flexbox упрощают выравнивание элементов, а эффекты наведения делают интерфейс более интерактивным. (Рис. 9)

. 

Рисунок 9 – Шапка

**Вывод:**

Созданный хедер полностью соответствует требованиям современного веб-дизайна: он удобен, функционален и визуально привлекателен. Фиксированное позиционирование и продуманная структура упрощают навигацию, а стилизация и интерактивные эффекты повышают комфорт и качество взаимодействия пользователя с сайтом.

# 5.11.2025

**Тема:** Frontend разработка основная страница: написание секций «Услуги», «Контакты». Оформление отчетной документации.

**Цель:** Создать секции «Услуги» и «Контакты» на сайте.

**Ход работы:**

Шаг 1**.** Была разработана первая часть ознакомительной страницы, где пользователь может увидеть основные категории искусства, такие как интерьер, животные, природа и сказочные герои (рис. 10). Эти карточки служат быстрым переходом к соответствующим разделам сайта.

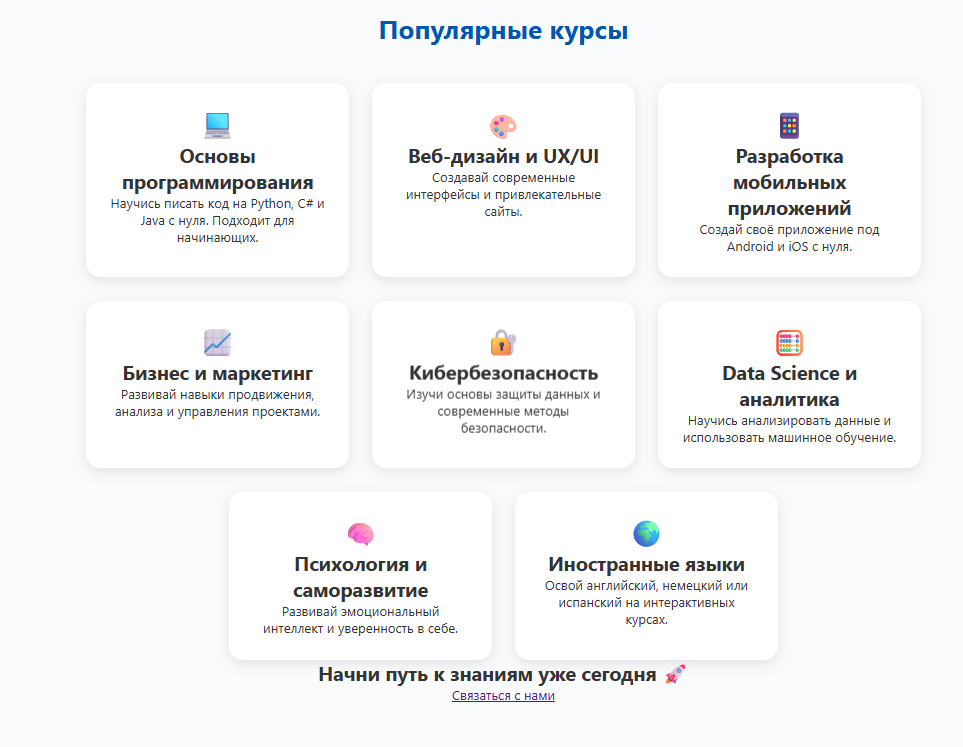


Рисунок 10 - карточки

Шаг 2**.** Для реализации карточек в CSS использовалась сеточная верстка (display: grid), что позволило разделить область на два столбца и гибко настроить размеры элементов. Такой подход обеспечивает аккуратное и равномерное размещение карточек на странице.

**Вывод:** Реализация карточек категорий улучшила структуру страницы и сделала взаимодействие с сайтом более удобным. Благодаря использованию CSS Grid карточки получили аккуратное визуальное оформление, а пользователь может легко выбирать интересующие его разделы, что повышает общую удобство и интерактивность интерфейса.

# 6.11.2025

**Тема:** Frontend разработка основная страница: написание форм входа/регистрации. Оформление отчетной документации.

**Цель:** Реализовать формы входа и регистрации.

**Ход работы:**

Шаг 1. После добавления карточек я реализовал навигацию по сайту в формате слайдов. Для этого были подключены необходимые JavaScript-скрипты, и началась разработка второго слайда.

Шаг 2**.** На втором слайде был создан шаблон для авторизации и регистрации пользователя. Также был оформлен подвал страницы, где размещены ссылки на социальные сети проекта и возможность обращения в техническую поддержку (рис. 11).

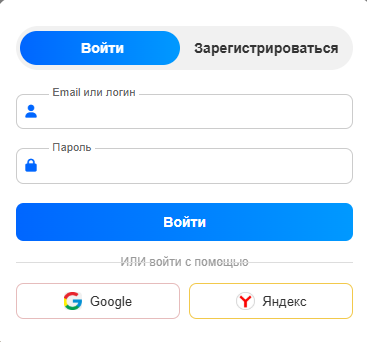


Рисунок 11 - Вход

**Вывод:** Сегодня была добавлена функциональность пролистывания страницы в виде слайдов, а также создан рабочий шаблон форм регистрации и авторизации. Эти элементы улучшают структуру сайта и открывают возможность дальнейшей реализации пользовательского взаимодействия.

# 7.11.2025

**Тема:** Frontend разработка основная страница: написание секции «Написать сообщение» и подвала сайта. Оформление отчетной документации.

**Цель:** Добавить секцию «Написать сообщение» и подвал сайта.

**Ход работы:**

**Шаг 1.** Выполнена стилизация панели для регистрации и авторизации пользователей. Интерфейс оформлен так, чтобы быть удобным, понятным и содержать все необходимые поля для ввода данных (рис. 12 – показ меню входа, рис. 13 – показ меню регистрации).

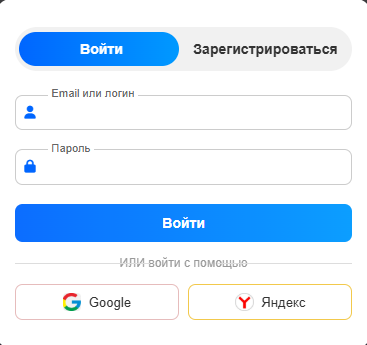


Рисунок 12 - показ меню входа

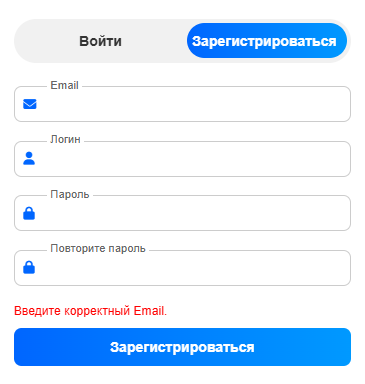


Рисунок 13 – показ меню регистрации

**Шаг 2.** Для переключения между формами регистрации и авторизации был добавлен JavaScript-скрипт. Благодаря ему при нажатии на кнопку «Зарегистрироваться» правая часть панели плавно смещается влево, открывая форму для создания аккаунта. Это повышает удобство взаимодействия и делает интерфейс более интерактивным.

**Вывод:** Панель авторизации и регистрации была улучшена и дополнена динамическим переключением между формами, что повысило удобство и функциональность пользовательского интерфейса.

# 8.11.2025

**Тема:** Использование fetch-запросов для обновления данных без перезагрузки страницы. Оформление отчетной документации.

**Цель:** Освоить работу с fetch-запросами для обновления данных без перезагрузки страницы.

**Ход работы:**

**Шаг 1.** Сегодня была изучена тема fetch-запросов, которые позволяют отправлять сетевые запросы к серверу и получать обновлённые данные по мере необходимости. Их базовый синтаксис выглядит следующим образом: let promise = fetch(url, [options]);

**Шаг 2.** Доработан подвал ознакомительной страницы, включающий ссылки на социальные сети, техническую поддержку и раздел часто задаваемых вопросов.

**Шаг 3.** Структура подвала организована таким образом, что каждый элемент занимает 33% ширины страницы и располагается по центру. Это обеспечивает аккуратную и симметричную компоновку.

**Шаг 4.** Начата разработка основной страницы, где будут представлены композиции разных пользователей. Создан хедер с кнопкой «Главная» для возврата в начало страницы и кнопкой «Создать» для публикации пользовательских работ. Структура кода продумана так, чтобы обеспечить удобное и быстрое перемещение по сайту.

**Вывод:** Сегодня был доработан подвал ознакомительной страницы и начата разработка основной страницы, включающая создание хедера и первичных элементов навигации.

# 10.11.2025

**Тема:** Создание адаптивного интерфейса с использованием медиа-запросов CSS. Оформление отчетной документации.

**Цель:** Создать адаптивный интерфейс сайта с помощью медиазапросов.

**Ход работы:**

Шаг 1. Сегодня были разработаны медиа-запросы для экранов шириной менее 1000 пикселей. В рамках адаптации интерфейса были выполнены следующие изменения:

1. уменьшен размер шрифтов для лучшей читаемости на компактных устройствах;
2. изменены размеры карточек и других элементов, чтобы они корректно отображались в ограниченном пространстве;
3. скорректировано расположение кнопок для удобства взаимодействия;
4. скрыта навигационная панель (navbar) для экономии места — в дальнейшем планируется реализовать «бургер-меню».

(рис. 14 — версия для экранов >1000 px; рис. 15 — адаптация для <1000 px)

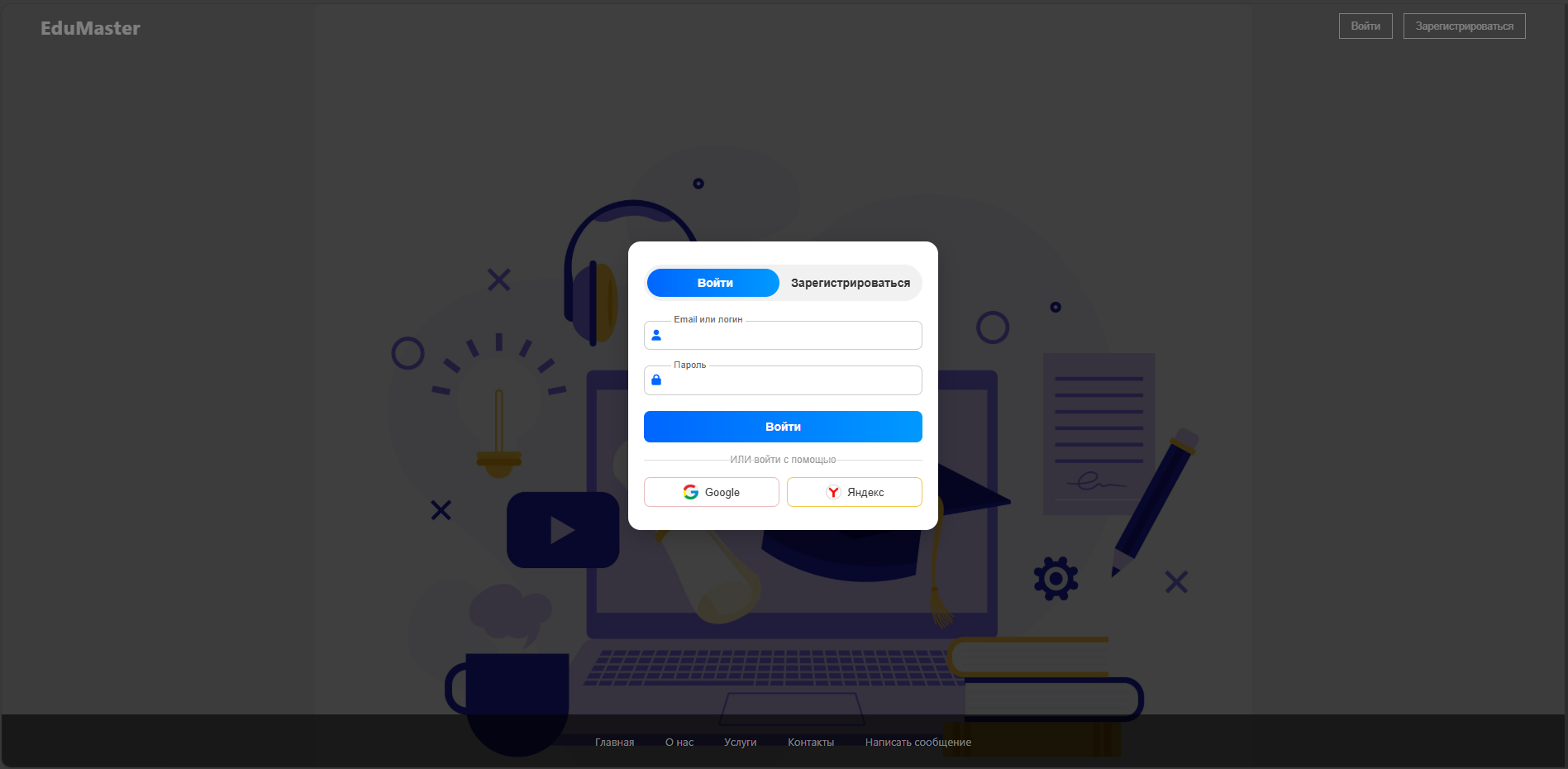


Рисунок 14 – более 1000 px

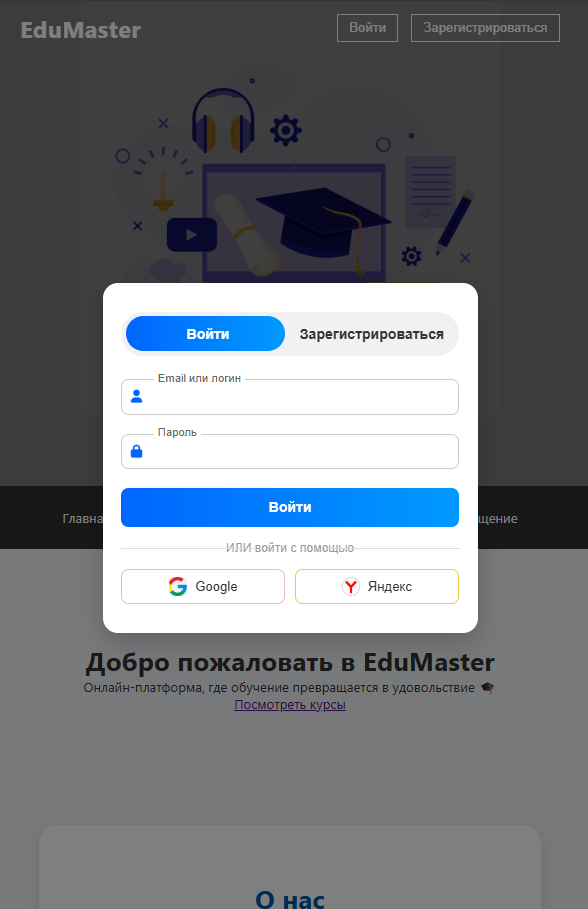


Рисунок 15 – менее 1000 px

**Вывод:** Начата адаптация интерфейса под устройства с меньшими экранами — телефоны, ноутбуки и планшеты. Реализованные медиа-запросы делают сайт более удобным и корректным на разных типах устройств.

# 11.11.2025

**Тема:** Знакомство с PostgreSQL. Оформление отчетной документации.

**Цель:** Познакомиться с основами PostgreSQL.

**Ход работы:**

Шаг 1. Сегодня была разработана система проверки корректности данных, вводимых пользователем в формах авторизации и регистрации. Если пользователь вводит неверную информацию, система выводит уведомление об ошибке (рис. 16).

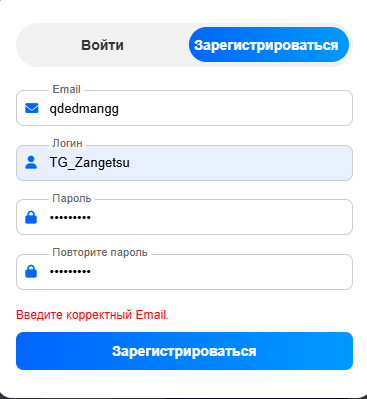


Рисунок 16 - ошибки

Шаг 2. В JavaScript-файле был реализован обработчик событий для регистрации и авторизации, а также функции, которые выводят пользователю сообщения о выявленных ошибках

Шаг 3. Сегодня я также познакомился с системой управления базами данных PostgreSQL. Это СУБД позволяет управлять базами данных: создавать, изменять, удалять записи, выполнять транзакции. Я узнал, что ранние версии PostgreSQL основаны на проекте POSTGRES, разработанном в Калифорнийском университете в Беркли.

Шаг 4. PostgreSQL широко используется бэкенд-разработчиками для взаимодействия с данными на серверной части сайта. СУБД предоставляет возможность:

1. выполнять запросы и просматривать информацию из базы;
2. редактировать, создавать и удалять записи;
3. контролировать обновления и версионность;
4. обеспечивать безопасность и предотвращать утечки;
5. настраивать доступ к данным.

Шаг 5. Преимущества PostgreSQL:

1. Объектно-реляционная модель, позволяющая связывать данные между собой и формировать сложные структуры.
2. Поддержка множества типов данных: целых, вещественных, строковых, булевых, геометрических, бинарных, а также XML, JSON, NoSQL, массивов и сложных композитных типов.
3. Многоязычность — поддержка SQL, C, Python, JavaScript и других языков.
4. Параллельный доступ (MVCC**)** — пользователи могут работать с БД одновременно без конфликтов.
5. Соответствие принципам ACID, что обеспечивает надежность и целостность данных.
6. Расширяемость — возможность добавлять собственные типы данных, индексы и функции.
7. Кроссплатформенность — работает на Linux, Windows, macOS и в облачных средах.
8. Открытый исходный код, стабильность и надежность.
9. Высокая производительность, подходящая для крупных проектов.

Шаг 6. Недостатки PostgreSQL:

1. сложность настройки из-за большого количества параметров;

2. повышенное потребление ресурсов;

3. отсутствие некоторых функций, которые присутствуют в других СУБД.

**Вывод:** Сегодня я реализовал систему проверки данных при авторизации и регистрации, а также познакомился с СУБД PostgreSQL, изучив её устройство, преимущества и недостатки.

# 12.11.2025

**Тема:** Создание базы данных в СУБД PostgreSQL. Оформление отчетной документации.

**Цель:** Создать базу данных в PostgreSQL и подготовить её структуру.

**Ход Работы:**

Первым этапом была создана база данных «EduMaster», предназначенная для хранения портфолио и личных данных пользователей.

Далее была разработана структура базы, состоящая из трех основных таблиц:

1. User- хранит учетные данные пользователей (имя и электронную почту);
2. CourseImage **-** содержит загруженные изображения;
3. Course **-** предназначена для хранения курсов
4. Category **–** предназначена для хранения категорий курсов

Итоговая структура созданной базы данных представлена ниже.

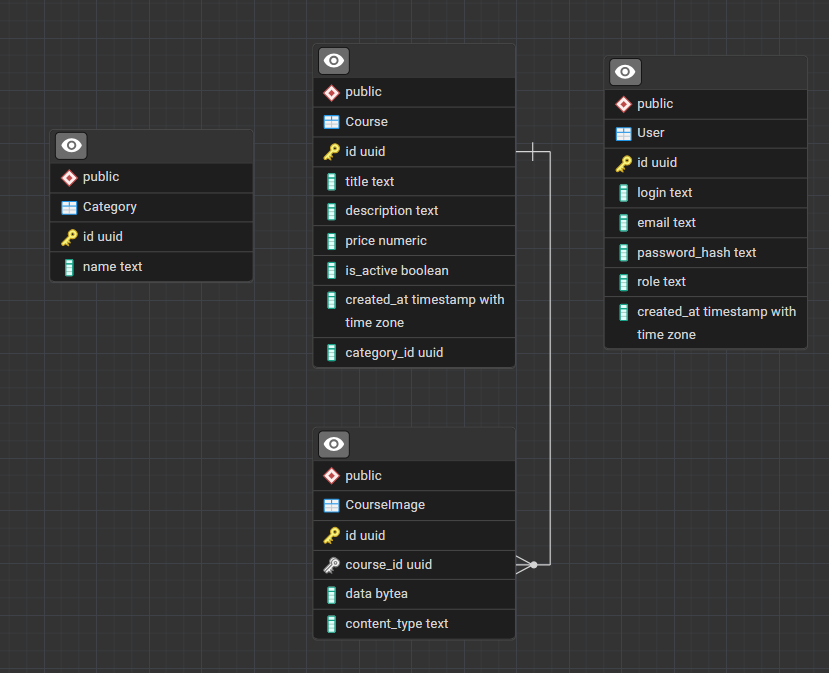


Рисунок 17. Структура базы данных EduMaster.

**Вывод:** В ходе работы была спроектирована и реализована база данных с необходимым набором таблиц для хранения контента и пользовательской информации.

# 13.11.2025

**Тема:** Подключение PostgreSQL через appsettings.json. Определение сущностей и их атрибутов в C# для базы данных с использованием Entity Framework. Оформление отчетной документации.

**Цель:** Подключить приложение к PostgreSQL и создать сущности в C# через Entity Framework.

**Ход работы:**

Шаг1**.** Настройка подключения и архитектуры

Работа началась с настройки соединения с базой данных: строка подключения была добавлена в файл конфигурации appsettings.json. Затем была сформирована структура проекта:

1. В проект DAL (Data Access Layer) добавлен класс ApplicationDbContext для взаимодействия с БД.
2. В проект Domain добавлена папка ModelsDb, в которой описаны классы-сущности, представляющие таблицы базы данных, а также необходимые перечисления (Enum).

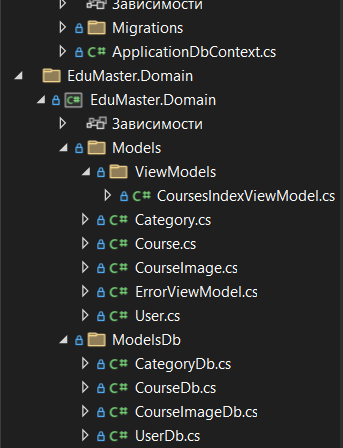


Рисунок 18. Структура проекта и класс контекста БД.

Шаг 2. Миграция и проверка

После описания всех моделей была выполнена миграция для проверки корректности связей и создания структуры в самой базе данных PostgreSQL.

**Вывод:** В ходе работы настроено подключение к PostgreSQL, реализована объектная модель данных (сущности и контекст) и успешно выполнена первичная миграция.

# 14.11.2025

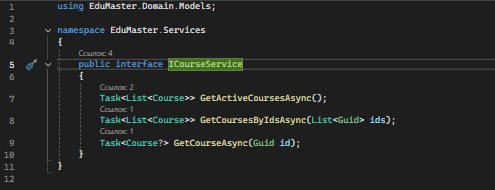
**Тема:** Разработка базовых операций CRUD для всех сущностей. Добавление моделей. Оформление отчетной документации.

**Цель:** Реализовать операции CRUD для всех сущностей проекта.

**Ход Работы:**

Шаг 1. Определение контрактов (Интерфейсы)

Для обеспечения гибкости архитектуры и возможности расширения функционала был создан слой абстракции. В папке Services (проект EduMaster.Services) я разработал интерфейс ICourseServices. В нем описаны сигнатуры методов для выполнения основных операций CRUD: получение списка курсов, создание нового курса, редактирование и удаление. Это позволяет отделить описание действий от их конкретной реализации.



Рискнок 19. Интерфейс ICourseServices

Шаг 2. Реализация сервисного слоя

На основе интерфейса был создан класс CourseService, который содержит фактическую логику работы с данными. Этот класс взаимодействует с базой данных через контекст ApplicationDbContext. Использование сервиса позволяет инкапсулировать бизнес-правила (например, валидацию данных перед сохранением) и разгрузить контроллеры от сложной логики.

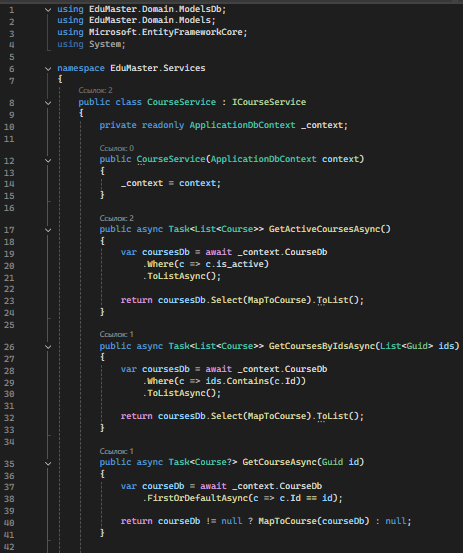


Рисунок 20. Класс CourseService

Шаг 3. Реализация методов управления данными

Внутри сервиса CourseService были запрограммированы конкретные методы. Для чтения данных используются запросы LINQ (например, для фильтрации курсов по категориям), для создания и обновления — методы Entity Framework (AddAsync, SaveChangesAsync). Это обеспечивает корректную трансляцию действий пользователя в SQL-запросы к базе данных PostgreSQL.

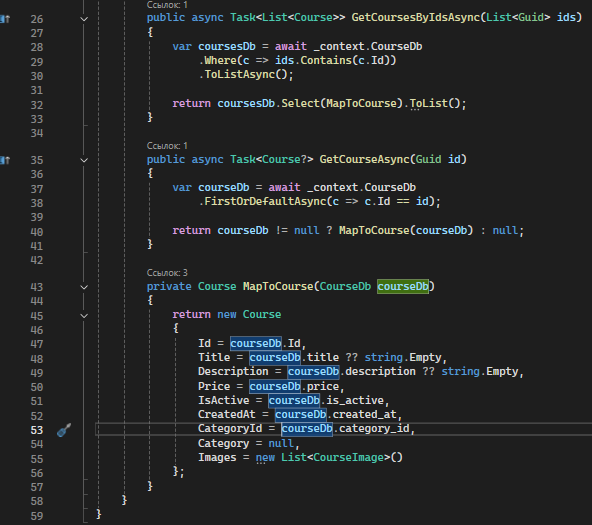


Рисунок 21. Пример метода Create/Edit в CourseService.

Шаг 4. Использование доменных моделей

для передачи данных между слоями приложения и базой данных используются классы-сущности из проекта EduMaster.Domain. Основной моделью для CRUD-операций является класс Course, содержащий такие свойства, как заголовок, описание, цена и ссылка на изображение. Эти модели обеспечивают строгую типизацию данных на всех этапах обработки.

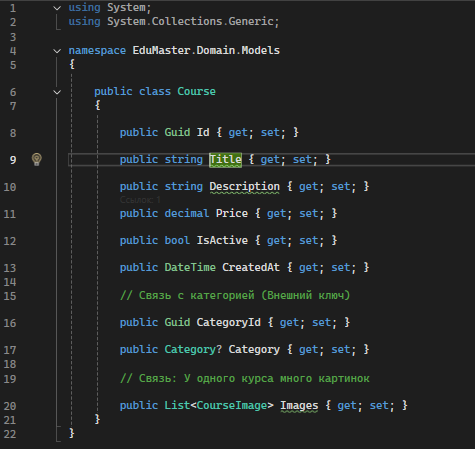


Рисунок 22. Модель Course

**Вывод:** в рамках выполнения работы была успешно реализована система CRUD на базе сервисной архитектуры. Связка интерфейса ICourseServices, реализации CourseService и доменной модели Course обеспечивает надежное управление образовательным контентом и позволяет легко масштабировать функционал приложения в будущем.

# 15.11.2025

**Тема:** Разработка сервисов для авторизации и аутентификации. Оформление отчетной документации.

**Цель:** Начать разработку сервисов авторизации и аутентификации.

**Ход работы:**

Шаг 1. Обеспечение безопасности (Хеширование)

Чтобы не хранить пароли пользователей в открытом виде, был создан класс PasswordHasher, реализующий интерфейс IPasswordHasher. Он отвечает за шифрование пароля перед записью в базу данных и сверку хешей при входе.

Шаг 2. Разработка сервиса авторизации

Для управления аккаунтами был создан интерфейс IAuthService и его реализация — класс AuthService. В конструкторе этого класса происходит внедрение зависимостей: контекста базы данных (ApplicationDbContext) для сохранения данных и хешера паролей (IPasswordHasher) для защиты.

Шаг 3. Реализация логики регистрации

В методе регистрации (RegisterAsync) прописана логика проверки: система ищет в базе данных пользователя с таким же Email. Если пользователь не найден, создается новый объект User, его пароль хешируется, и данные сохраняются в БД.

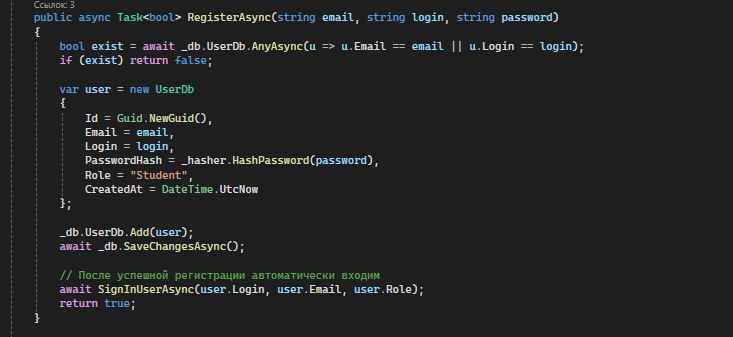


Рисунок 23. Реализация метода регистрации в AuthService.

**Вывод:** Сегодня была разработана и внедрена безопасная система регистрации и авторизации. Использование отдельного сервиса и хеширования паролей обеспечивает надежную защиту данных пользователей и правильную архитектуру приложения.

# 17.11.2025

**Тема:** Доработка сервисов для авторизации и аутентификации. Оформление отчетной документации.

**Цель:** Доработать механизмы авторизации и аутентификации.

**Ход Работы:**

Шаг 1. Настройка контроллеров

Для того чтобы контроллеры (например, CoursesController) могли взаимодействовать с бизнес-логикой, был применен механизм внедрения зависимостей. В конструкторы контроллеров добавлены параметры интерфейсов (ICourseServices, IAuthService), что позволяет автоматически получать экземпляры сервисов при обработке запросов, не создавая их вручную.

Шаг 2. Регистрация зависимостей в Program.cs

Роль инициализатора в проекте выполняет файл Program.cs. В нем я прописал регистрацию всех разработанных сервисов в контейнере внедрения зависимостей (DI). С помощью метода builder.Services.AddScoped интерфейсы были связаны с их реализациями (например, IPasswordHasher с PasswordHasher), что обеспечивает их доступность во всем приложении.

Шаг 3. Сборка и запуск приложения

После регистрации сервисов была настроена конфигурация HTTP-конвейера и выполнен вызов метода app.Run(). Это завершает процесс инициализации, делая приложение готовым к приему входящих подключений и обработке данных пользователей.

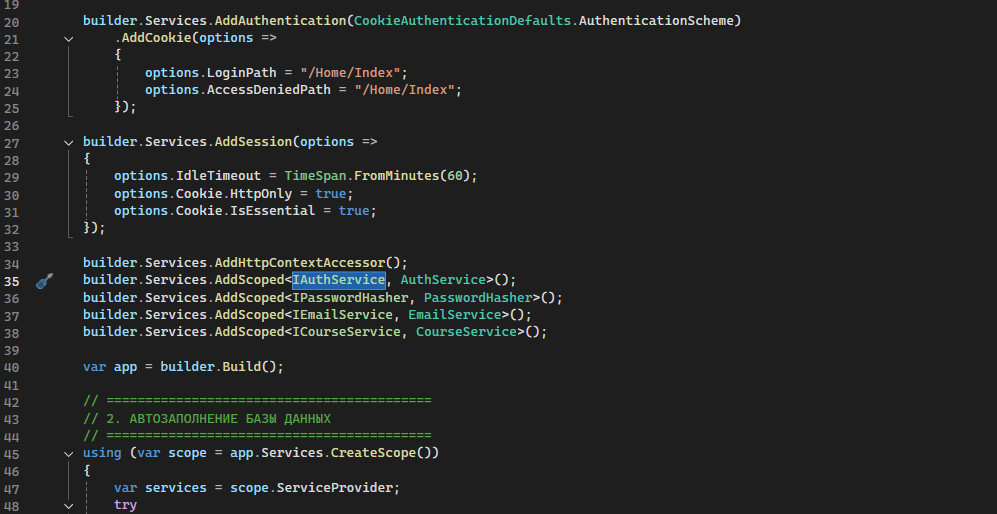


Рисунок 24. Регистрация сервисов и запуск в Program.cs.

**Вывод:** В результате работы была успешно настроена архитектура приложения. Использование встроенного контейнера зависимостей в Program.cs позволило связать все компоненты системы воедино, обеспечив стабильную работу авторизации и функционала курсов без лишнего усложнения кода.

# 18.11.2025

**Тема:** Подготовка проекта к использованию ClaimsIdentity. Оформление отчетной документации.

**Цель:** Подготовить проект к использованию ClaimsIdentity

**Ход работы:**

Шаг 1. Реализация хеширования

Для защиты пользовательских данных в папке Services был создан класс PasswordHasher. Он реализует интерфейс IPasswordHasher и отвечает за криптографическое преобразование паролей. Это гарантирует, что даже при доступе к БД злоумышленники не увидят реальные пароли пользователей.

Шаг 2. Интеграция хеширования в сервис авторизации

В классе AuthService была обновлена логика входа. Теперь при аутентификации введенный пользователем пароль не сравнивается напрямую со строкой в БД, а проходит через метод Verify хешера. Это обеспечивает безопасную проверку учетных данных.

Шаг 3. Внедрение ClaimsIdentity

Для управления сессией пользователя я изменил возвращаемый результат авторизации. Вместо простой модели пользователя теперь формируется объект ClaimsIdentity. Он наполняется необходимыми данными («заявлениями» или Claims), такими как имя и роль, что позволяет системе автоматически распознавать права доступа пользователя на протяжении всей сессии.

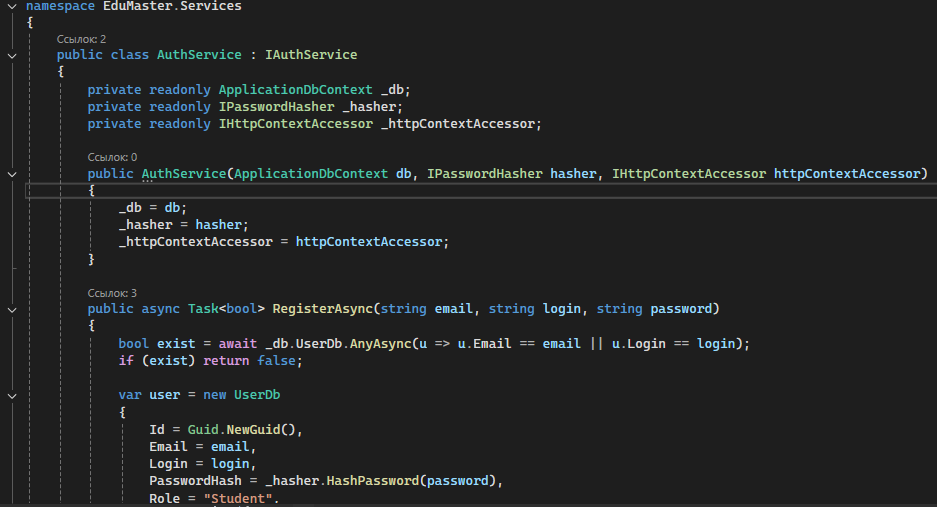


Рисунок 25 Проверка пароля и создание ClaimsIdentity в AuthService.

**Вывод:** В ходе работы была реализована надежная защита паролей с помощью хеширования, а также внедрен механизм ClaimsIdentity, который управляет аутентификацией и авторизацией в приложении.

# 19.11.2025

**Тема:** Подключение HttpContext.User и ClaimsIdentity для управления доступом пользователей и роли.

**Цель:** Подключить HttpContext.User и ClaimsIdentity для работы с пользователями.

**Ход работы:**

Шаг 1. Реализация функционала выхода (Logout)

Для управления жизненным циклом сессии пользователя был реализован метод выхода из аккаунта. В контроллере используется асинхронный вызов HttpContext.SignOutAsync, который принудительно завершает сессию и удаляет файлы cookie, отвечающие за аутентификацию. Это гарантирует, что после нажатия кнопки «Выйти» доступ к защищенным ресурсам прекращается немедленно.

Шаг 2. Интеграция с интерфейсом (HttpContext.User)

Чтобы интерфейс приложения реагировал на статус пользователя, в представлениях (Views) было настроено использование объекта HttpContext.User. С помощью свойства User.Identity.IsAuthenticated скрываются или отображаются кнопки «Вход/Регистрация» и «Выйти» в навигационной панели, обеспечивая корректное отображение меню для гостей и авторизованных пользователей.

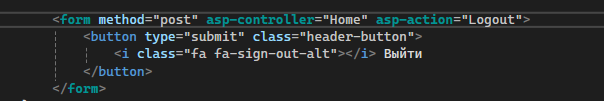


Рисунок 26. Реализация выхода и проверка статуса пользователя.

**Вывод:** В ходе работы была настроена безопасная система управления сессиями. Использование HttpContext позволяет корректно определять статус пользователя и безопасно завершать работу с приложением, очищая все данные аутентификации.

# 20.11.2025

**Тема**: Использование C#, FluentValidation и JavaScript для валидации вводимых данных при авторизации и регистрации. Оформление отчетной документации.

**Цель:** Настроить валидацию данных при авторизации и регистрации.

**Ход работы:**

Шаг 1. Использование Data Annotations

Для проверки корректности вводимых данных я использовал встроенный механизм ASP.NET Core — аннотации данных. В классах-моделях (например, для входа и регистрации) были добавлены атрибуты валидации, такие как [Required] (обязательное поле) и [EmailAddress]. Это позволяет системе автоматически проверять формат данных до начала обработки бизнес-логики.

Шаг 2. Клиентская валидация

Чтобы обеспечить мгновенный отклик интерфейса без перезагрузки страницы, я подключил клиентскую валидацию. На представлениях Login и Register был добавлен вызов частичного представления \_ValidationScriptsPartial.cshtml, который подключает библиотеки jQuery Validate.

Шаг 3. Проверка бизнес-логики в сервисе

Дополнительная валидация реализована на уровне сервиса AuthService. В методе регистрации перед созданием нового пользователя происходит обращение к базе данных для проверки уникальности Email. Если пользователь уже существует, сервис возвращает ошибку, предотвращая дублирование данных.

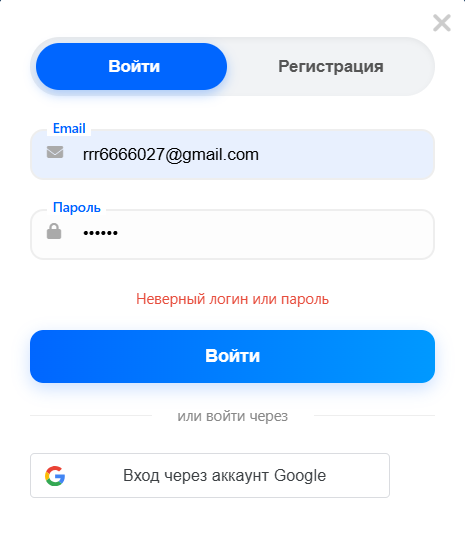


Рисунок 27. Пример валидации

**Вывод:** В ходе работы была реализована двухуровневая система валидации (клиентская и серверная), обеспечивающая надежность данных и удобство для пользователя при регистрации и авторизации.

# 21.11.2025

**Тема:** Использование C#, FluentValidation и JavaScript для валидации вводимых данных для других объектов. Оформление отчетной документации.

**Цель:** Добавить валидацию для других объектов системы.

**Ход работы:**

Шаг 1. Реализация адаптивного меню и профиля

Для обеспечения удобной навигации была доработана панель управления (Navbar) в файле \_Layout.cshtml. Я добавил логику проверки авторизации: если пользователь вошел в систему, вместо кнопок «Вход/Регистрация» отображается его Email и кнопка «Выйти». Для мобильных устройств меню автоматически сворачивается в кнопку-«гамбургер», экономя место на экране.

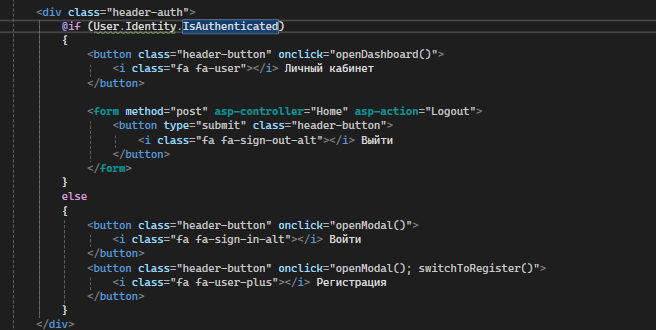


Рисунок 28. Код навигационной панели (Navbar) с условием авторизации.

Шаг 2. Оформление главной страницы

На главной странице (Index.cshtml) был реализован современный дизайн с использованием сетки Bootstrap. Размещены интерактивные информационные блоки и изображения, при взаимодействии с которыми (наведении или нажатии) пользователь получает визуальный отклик. Это делает интерфейс более живым и привлекательным для посетителей.

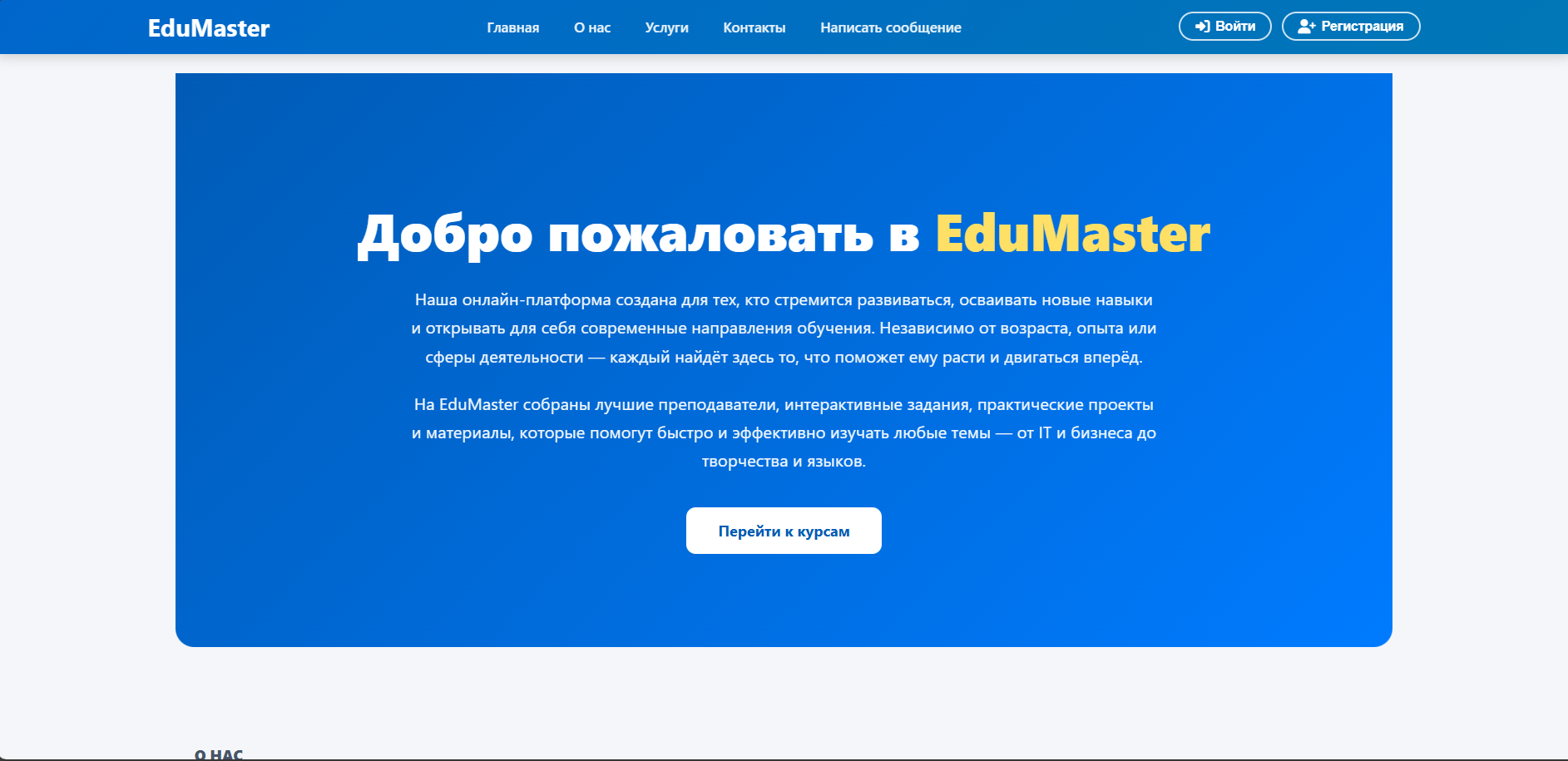


Рисунок 29. Обновленный дизайн главной страницы.

**Вывод:** Сегодня был завершен этап верстки ключевых элементов интерфейса. Реализовано адаптивное меню с доступом к профилю пользователя и обновлен визуальный стиль главной страницы, что значительно улучшило UX (пользовательский опыт).

# 22.11.2025

**Тема:** Использование Google аккаунта в качестве сервера отправки сообщений. Оформление отчетной документации.

**Цель:** Настроить отправку сообщений через Google-аккаунт.

**Ход работы:**

Шаг 1. Настройка конфигурации

Чтобы не хранить пароли в коде, параметры SMTP-сервера (хост, порт, логин и пароль приложения) были вынесены в файл конфигурации appsettings.json.

Шаг 2. Создание интерфейса

В слой сервисов добавлен интерфейс IEmailService. Он описывает контракт для отправки писем, что позволяет легко менять способ отправки без переписывания остального кода.

Шаг 3. Реализация отправки (SMTP)

Создан класс EmailService, реализующий этот интерфейс. В нем используется библиотека System.Net.Mail для подключения к почтовому серверу (Google/Yandex) и отправки сообщений в формате HTML.

Шаг 4. Внедрение в бизнес-логику

Новый сервис был внедрен в класс AuthService через конструктор. Это позволяет сервису аутентификации использовать возможности отправки почты.

Шаг 5. Обновление регистрации

Метод регистрации был доработан: теперь после создания пользователя генерируется код подтверждения, который сразу же отправляется на указанный Email через метод SendEmailAsync.

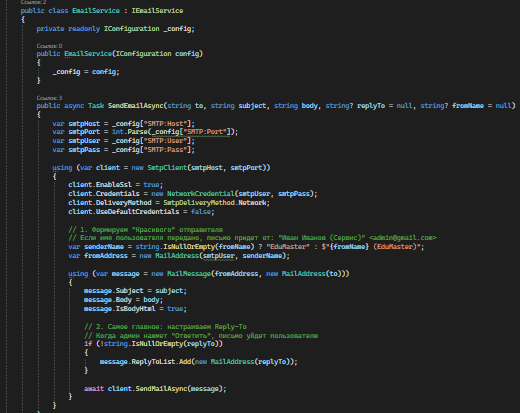


Рисунок 30. Реализация класса EmailService и отправка письма.

**Вывод:** Разработана система подтверждения почты. Благодаря разделению на шаги (конфигурация, интерфейс, реализация) функционал легко поддерживать, а аккаунты пользователей теперь защищены от фальшивых регистраций.

# 24.11.2025

**Тема:** Реализация проверки подлинности пользователя при регистрации. Оформление отчетной документации.

**Цель:** Реализовать проверку подлинности пользователя при регистрации.

**Ход работы:**

Шаг 1. Верстка окна подтверждения

В частичное представление \_LoginRegistrationPartial.cshtml был добавлен скрытый блок confirm\_\_email\_\_container. Он содержит форму для ввода цифрового кода. Стили настроены так, чтобы окно появлялось поверх основного контента (overlay), блокируя взаимодействие с остальной страницей до завершения проверки.

Шаг 2. Настройка JavaScript-логики

В файле login\_and\_registration\_script.js реализована функция управления окнами. При успешном ответе сервера на регистрацию скрипт скрывает форму регистрации и отображает поле ввода кода. Также добавлена функция confirmEmail, которая асинхронно (AJAX) отправляет введенный пользователем код на сервер без перезагрузки страницы.

Шаг 3. Обработка кода на сервере

В контроллере HomeController создан метод ConfirmEmail. Он принимает код от клиента, передает его в сервис авторизации для сверки и, в случае успеха, принудительно авторизует пользователя, перенаправляя его в личный кабинет.

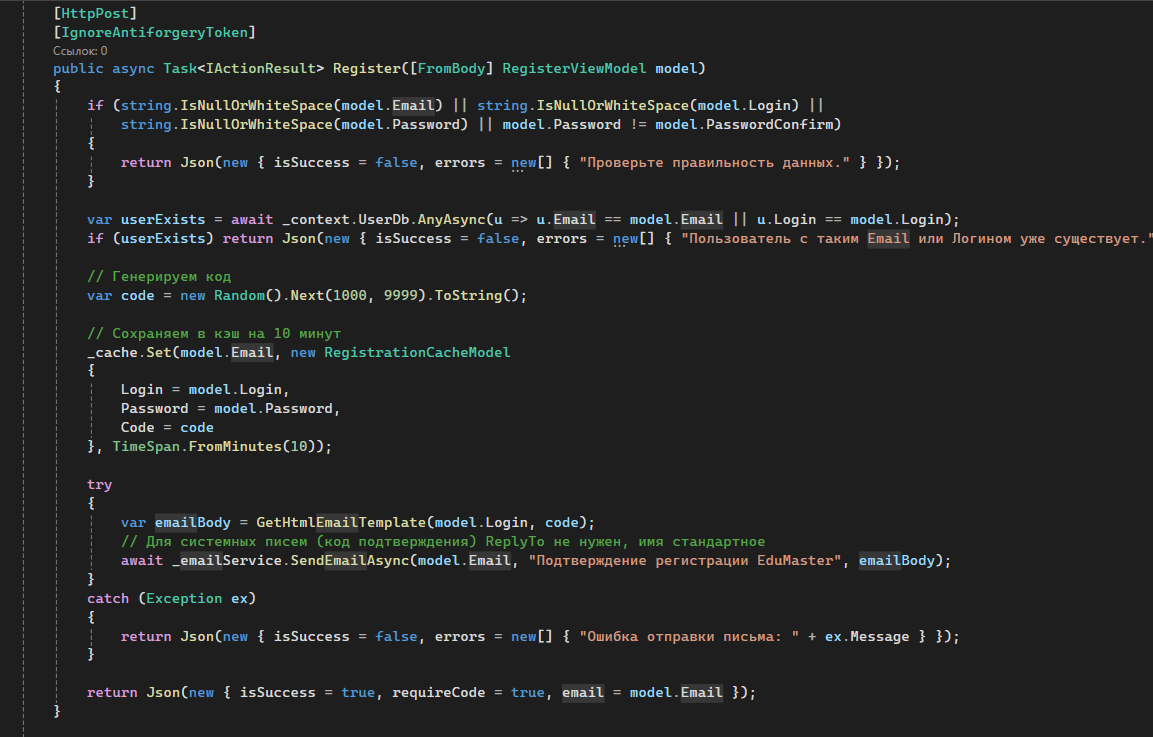


Рисунок 31. Метод подтверждения в контроллере Home.

**Вывод:** Реализован полный цикл подтверждения регистрации: от верстки всплывающего окна до серверной валидации кода. Связка AJAX-запросов и контроллера обеспечивает плавную и безопасную регистрацию пользователя.

# 25.11.2025

**Тема:** Настройка Google Cloud для авторизации. Реализация авторизации Google. Доработка авторизации Google. Оформление отчетной документации.

**Цель:** Реализовать и стабилизировать авторизацию через Google: от настройки Google Cloud до полной интеграции в проект и отладки работы

**Ход Работы:**

Шаг 1. Конфигурация провайдера идентификации (Google Cloud)

Разработка началась с настройки на стороне Google. В консоли Google Cloud Platform был создан проект и настроен экран доступа OAuth (Consent Screen), где указана информация о приложении. В разделе учетных данных (Credentials) был сгенерирован уникальный Client ID. Этот идентификатор необходим для того, чтобы Google мог распознавать запросы от нашего веб-приложения.

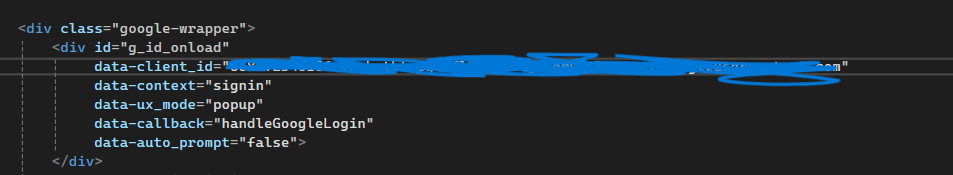


Рисунок 32. Настройка проекта и получение Client ID.

Шаг 2. Установка необходимых зависимостей

Для работы с токенами Google на сервере вместо стандартного Middleware была выбрана официальная библиотека от Google. В файл проекта EduMaster.csproj через NuGet был добавлен пакет Google.Apis.Auth. Эта библиотека предоставляет методы для криптографической проверки подписей JWT-токенов, что критически важно для безопасности.

Шаг 3. Реализация клиентской логики (Frontend)

На стороне интерфейса (View) была интегрирована кнопка "Войти через Google". Скрипт использует Google Identity Services API для получения ID-токена пользователя. При успешном входе Google возвращает зашифрованную строку (credential), которую JavaScript отправляет POST-запросом на серверный метод GoogleLogin.

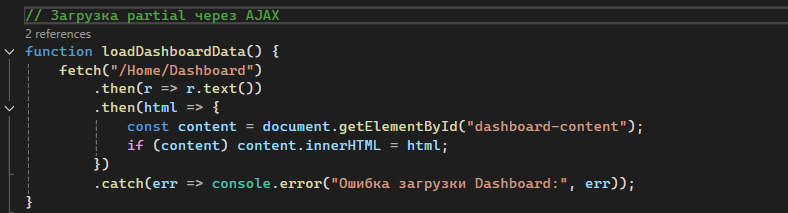


Рисунок 33. Кнопка входа и AJAX-запрос на сервер.

Шаг 4. Обработка запроса в контроллере

В контроллере HomeController был создан специальный асинхронный метод GoogleLogin, принимающий строковый параметр credential. Этот метод служит точкой входа для внешней авторизации и изолирует логику OAuth от стандартной формы входа по логину и паролю.

Шаг 5. Валидация токена безопасности

Ключевой этап безопасности реализован внутри метода контроллера. С помощью метода GoogleJsonWebSignature.ValidateAsync входящий токен проверяется на подлинность. Система сверяет цифровую подпись Google и поле Audience (наш Client ID), чтобы убедиться, что токен был выдан именно для нашего приложения, а не перехвачен злоумышленниками.

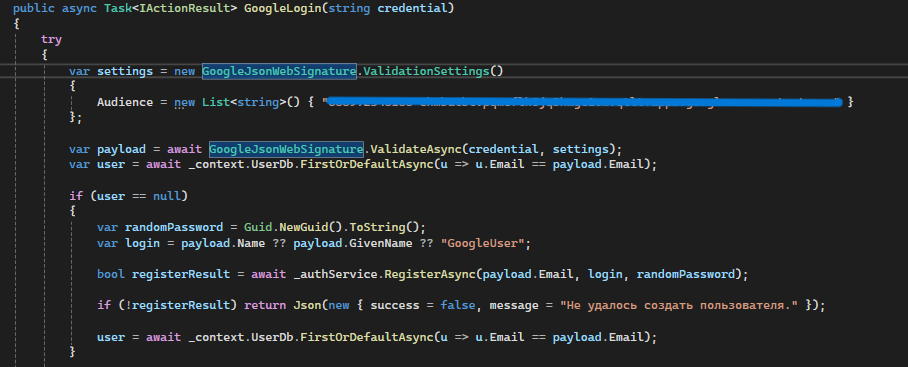


Рисунок 34. Код валидации токена в HomeController.

Шаг 6. Логика синхронизации пользователей

После успешной расшифровки токена (Payload) система получает Email и имя пользователя. Реализован алгоритм проверки:

1. Выполняется поиск пользователя в базе данных UserDb по Email.
2. Если пользователь не найден, вызывается метод \_authService.RegisterAsync. Система автоматически создает новый аккаунт, генерируя случайный пароль и сохраняя имя из профиля Google.

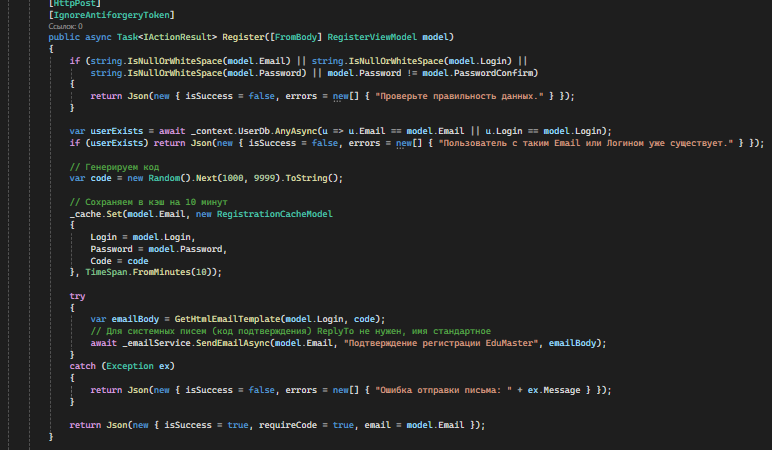


Рисунок 35. Алгоритм поиска и регистрации пользователя**.**

Шаг 7. Создание сессии авторизации

Завершающий этап — создание аутентификационной куки. Я вручную сформировал объект \_DashboardPartial с данными пользователя (Имя, Email, Роль) и вызвал метод HttpContext.SignInAsync. Это позволяет системе ASP.NET Core считать пользователя авторизованным и открывать ему доступ к защищенным разделам сайта (Личный кабинет).

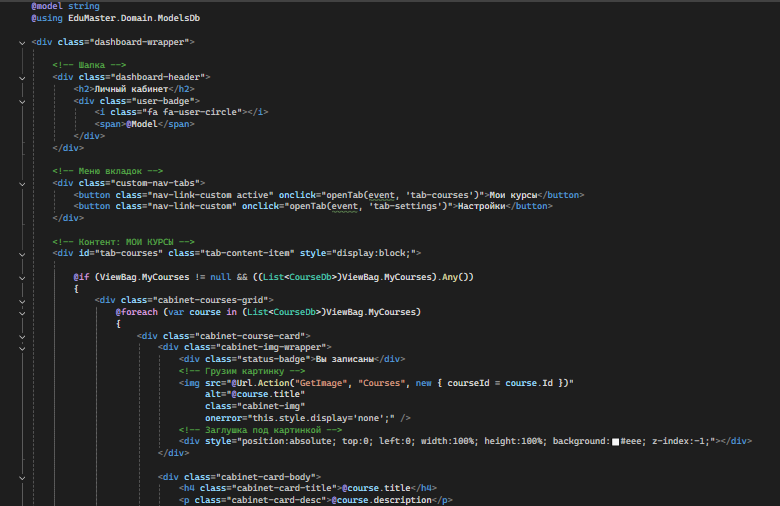


Рисунок 36. Создание \_DashboardPartial и вход в систему.

**Вывод:** В ходе работы была спроектирована и реализована безопасная система бесшовной авторизации. Использование библиотеки Google.Apis.Auth и ручная валидация токена на бэкенде позволили гибко интегрировать внешнюю авторизацию в существующую архитектуру приложения, обеспечив автоматическую регистрацию новых пользователей.

# 26.11.2025

**Тема:** Frontend разработка страницы с каталогом.

**Цель:** Разработать страницу каталога с каталогами и курсами.

**Ход работы:**

Шаг 1. Верстка приветственного экрана (Hero)

Была создана главная секция сайта. Я реализовал блок с заголовком, мотивирующим описанием платформы и кнопкой призыва к действию («Перейти к курсам»). Это первое, что видит пользователь при заходе на сайт.

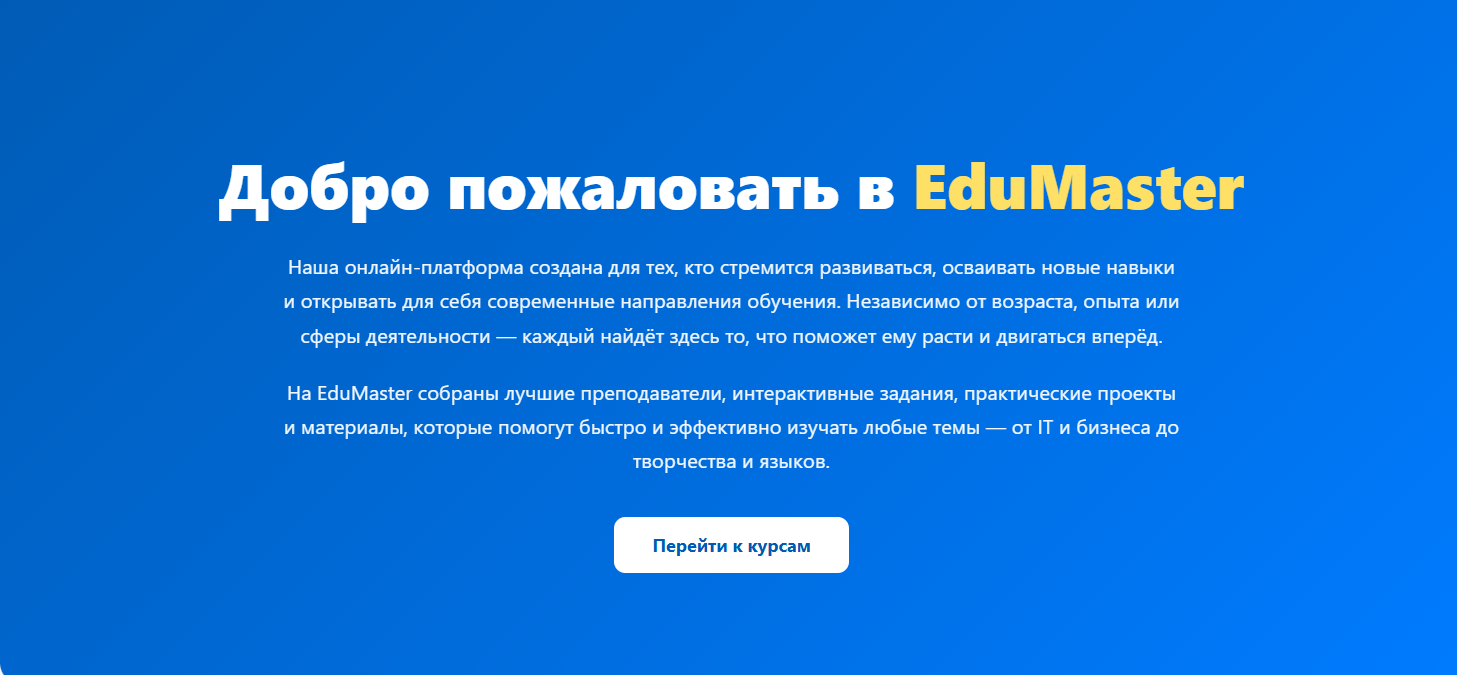


Рисунок 37. Приветственная секция (Hero).

Шаг 2. Информационные блоки

Сверстаны статические секции «О нас», «Отзывы» и «Партнеры». Для оформления использована CSS-верстка (Flexbox) и иконки, что позволяет структурировано подать информацию о преимуществах платформы и преподавателях.

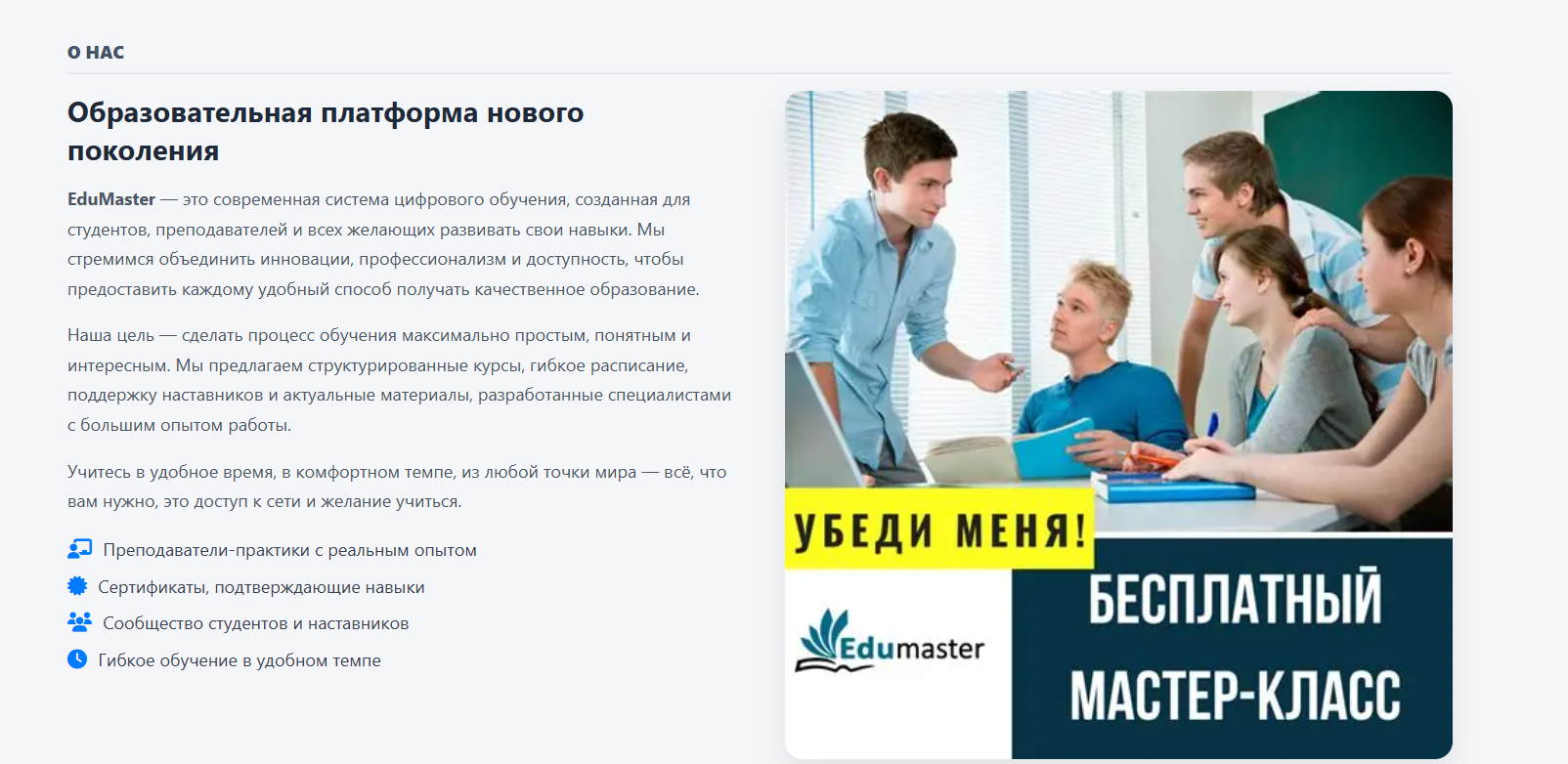


Рисунок 38. Секции "О нас"

Шаг 3. Реализация формы обратной связи

В нижней части страницы добавлена форма «Напишите нам». Я добавил логику: если пользователь авторизован, его имя и почта подставляются автоматически (через User.Identity), а поля блокируются от редактирования. Это упрощает отправку сообщения.

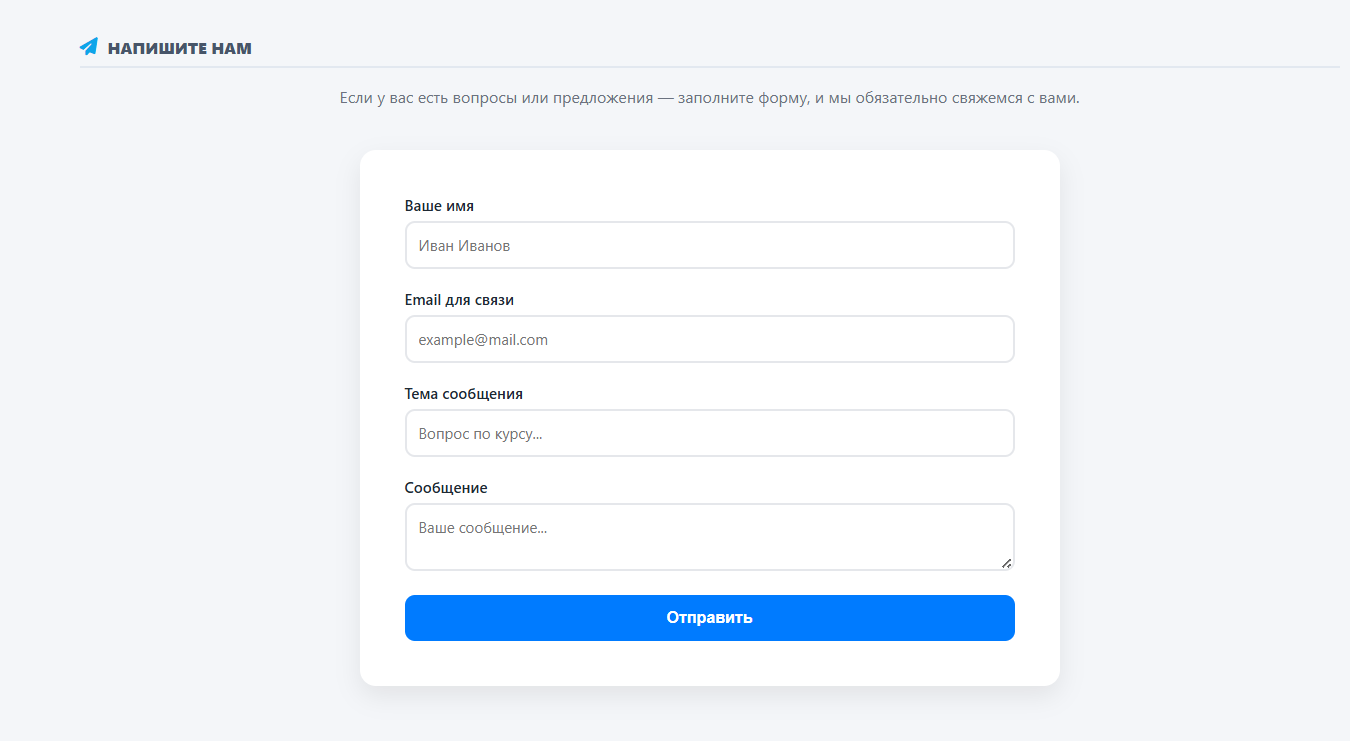


Рисунок 39. Умная форма обратной связи.

**Вывод:** Сегодня была полностью сверстана главная страница сайта в формате Landing Page. Реализована адаптивная структура блоков и настроена интерактивная форма связи с автозаполнением данных.

# 27.11.2025

**Тема:** Frontend доработка страницы с катологами.

**Цель:** Доработать интерфейс и функционал каталога курсов.

**Ход работы:**

Шаг 1. Реализация адаптивной сетки курсов

Сегодня была сверстана страница каталога (Catalog.cshtml). Для отображения списка курсов я использовал технологию **CSS Grid**. Свойство grid-template-columns: repeat(auto-fill, minmax(300px, 1fr)) позволяет карточкам автоматически перестраиваться в зависимости от ширины экрана, сохраняя ровную структуру без использования сложных JS-скриптов для выравнивания.

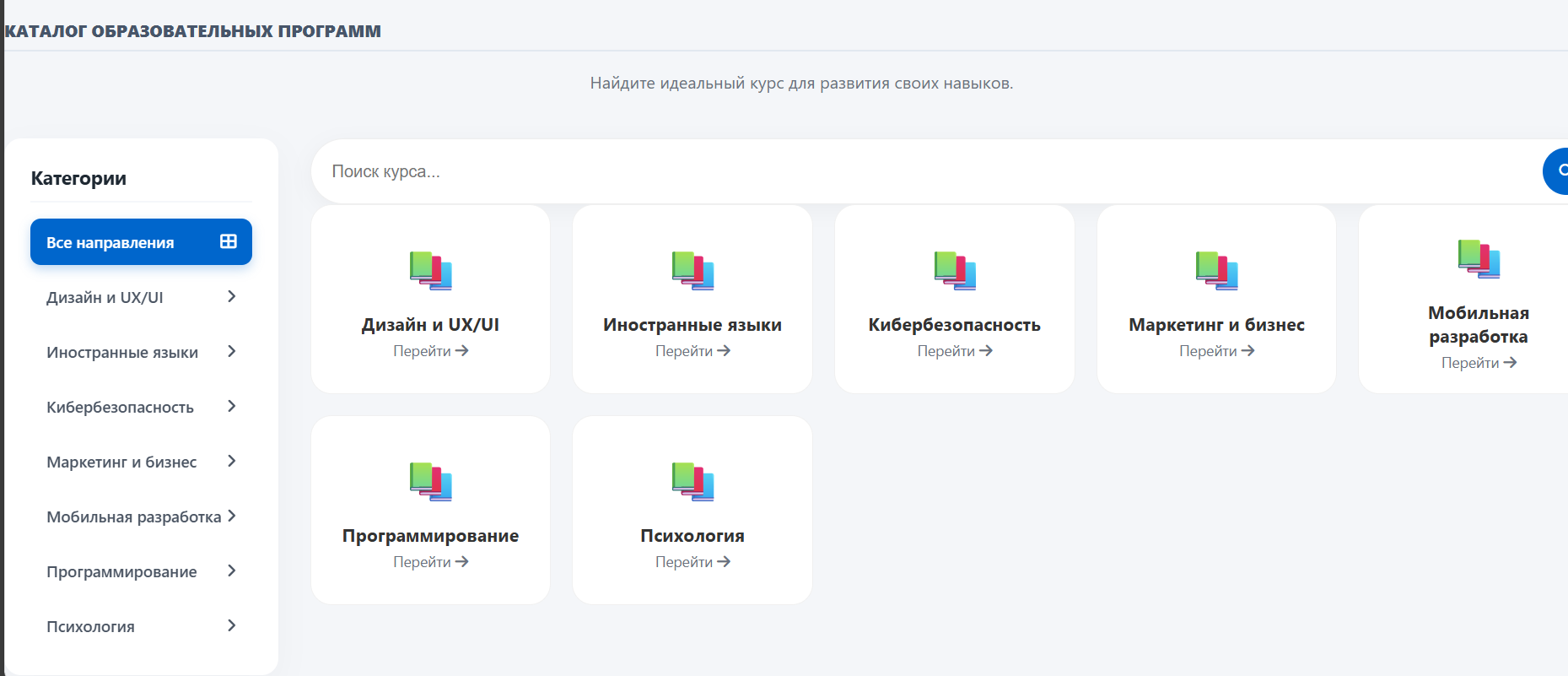


Рисунок. 40. Сетка курсов и сайдбар (Catalog.cshtml).

Шаг 2. Реализация интерактивного просмотра (JS)

Для просмотра подробной информации о курсе без перезагрузки страницы был написан JavaScript-код в файле courses\_script.js. Скрипт перехватывает нажатие на кнопку «Инфо», отправляет асинхронный запрос на сервер (Fetch API), получает данные (JSON) и динамически подставляет их в модальное окно Bootstrap.

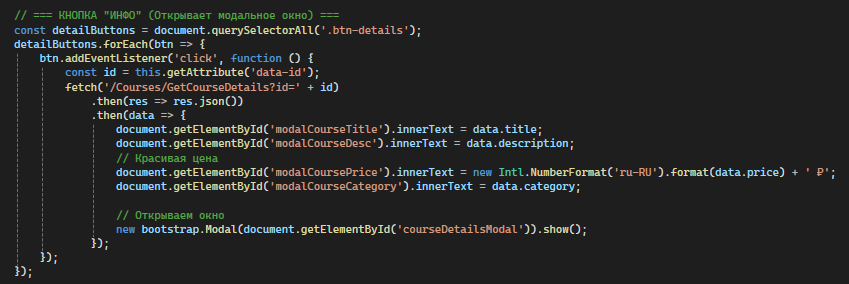


Рисунок 41. JavaScript для модального окна.

Шаг 3. Добавление анимаций и визуальных эффектов

В файле стилей courses.css были настроены анимации взаимодействия. При наведении курсора на карточку курса она плавно поднимается вверх (top: -10px), тень становится глубже, а изображение курса слегка увеличивается (scale(1.05)). Это делает интерфейс более отзывчивым и живым.

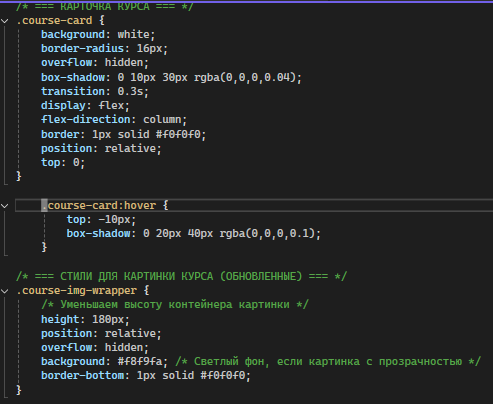


Рисунок 42. CSS-код анимации карточки.

**Вывод:** Сегодня была полностью доработана страница каталога: настроена адаптивная верстка, реализована логика модальных окон на JavaScript и добавлены микро-анимации для улучшения пользовательского опыта.

# 28.11.2025

**Тема:**Frontend доработка страницы со странами/каталог. Оформление отчетной документации.

**Цель:** Доработать интерфейс и функционал каталога курсов.

**Ход работы:**

Шаг 1. Реализация механизма загрузки изображений

Мной была завершена работа над серверной частью отображения медиа-контента. В отличие от хранения ссылок на файлы, в проекте реализовано хранение изображений курсов непосредственно в базе данных (PostgreSQL) в виде байтовых массивов. В контроллер CoursesController добавлен метод GetImage, который принимает ID курса, находит связанную запись в таблице CourseImageDb и возвращает файл клиенту. Это обеспечивает надежную привязку картинок к конкретным образовательным программам.

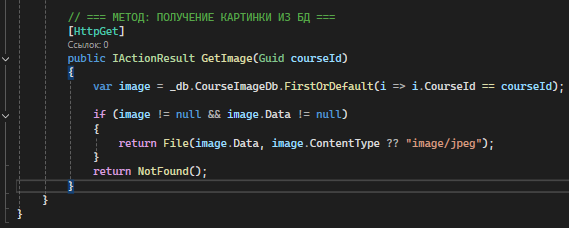


Рисунок 43. Метод GetImage в контроллере.

Шаг 2. Интеграция данных в представление каталога

Я доработал логику контроллера для страницы каталога. Метод Catalog теперь формирует специальную модель (CoursesIndexViewModel), в которую загружаются список категорий для фильтрации и список курсов, соответствующих поисковому запросу. На стороне интерфейса (Catalog.cshtml) настроен динамический вывод карточек: если изображение есть в базе, оно подгружается через созданный ранее метод, если нет — отображается стилизованная заглушка. Такой подход гарантирует, что каталог всегда выглядит опрятно, даже если контент заполнен частично.

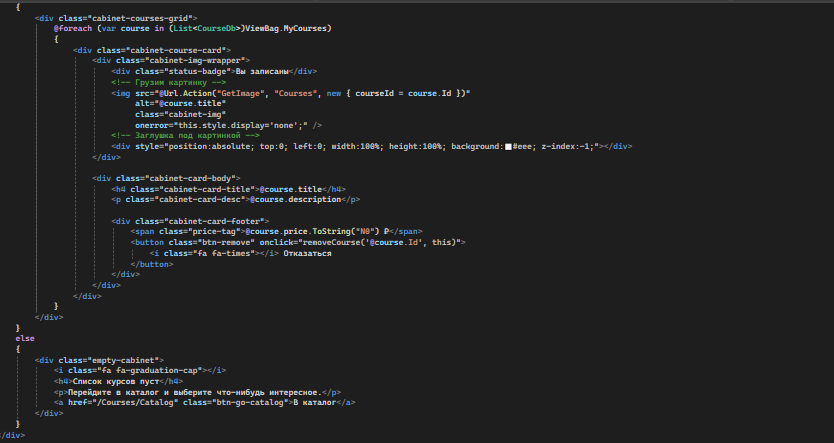


Рисунок 44. Формирование модели данных в методе Catalog.

**Вывод:** Сегодня была полностью налажена связка между базой данных и страницей каталога. Реализована передача сложных данных (курсы, категории, изображения) через ViewModel, что позволяет пользователю видеть актуальную информацию и пользоваться поиском по сайту.

# 29.11.2025

**Тема:** Frontend разработка страницы с элементами каталога: секция отображения.

**Цель:** Реализовать отображение элементов каталога.

**Ход работы:**

Шаг 1. Верстка Личного кабинета

Сегодня была сверстана страница Личного кабинета пользователя (Dashboard.cshtml). Интерфейс разделен на две основные зоны: боковая панель (сайдбар) с профилем и кнопкой выхода, и основная рабочая область. В основной области реализован вывод двух списков: «Мои курсы» (на которые пользователь записан) и «Доступные курсы» (общий каталог). Для пустых списков предусмотрены стилизованные заглушки («Вы пока не записаны...»).

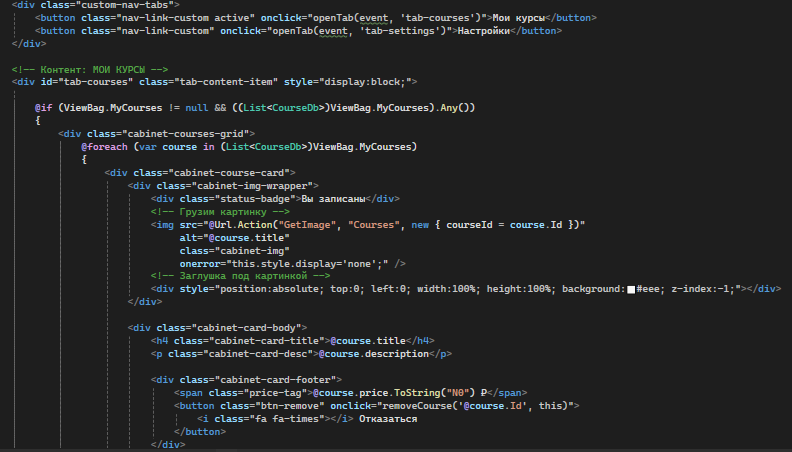


Рисунок 45. Верстка страницы \_DashboardPartial.cshtml**.**

Шаг 2. Реализация логики отображения курсов

Для наполнения страницы данными был использован механизм ViewBag. В контроллере HomeController (метод Dashboard) реализована логика: система определяет текущего пользователя, загружает список его подписок и передает их в представление. На странице (Dashboard.cshtml) данные выводятся через цикл foreach. Каждая карточка курса содержит название, описание и функциональную кнопку: «Отписаться» (для моих курсов) или «Записаться» (для доступных).

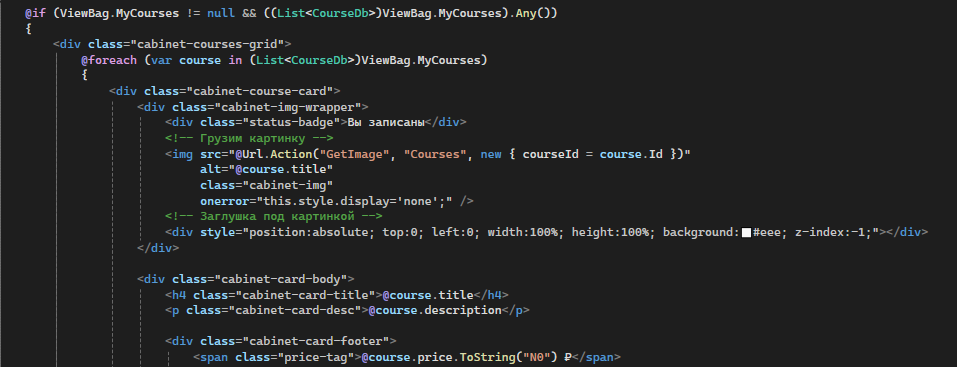


Рисунок 46. Цикл вывода курсов пользователя (\_DashboardPartial.cshtml**).**

**Вывод:** Сегодня был разработан функционал Личного кабинета. Пользователь теперь может видеть свой профиль, управлять подписками (записываться и отписываться) и отслеживать доступные образовательные программы в удобном интерфейсе.

# 1.12.2025

**Тема:** Frontend доработка страницы с элементами каталога: секция отображения.

**Цель:** Реализовать отображение элементов каталога.

**Ход работы:**

Шаг 1. Верстка сетки каталога

Сегодня была начата работа над представлением Catalog.cshtml. Основной задачей было создание адаптивной сетки для отображения списка курсов. В качестве основы использована модель CoursesIndexViewModel. Была реализована структура на базе div-контейнеров с применением CSS Grid (или Bootstrap Grid), что позволяет карточкам курсов автоматически перестраиваться в зависимости от ширины экрана (по 3 элемента в ряд на десктопе, по 1 на мобильных устройствах).

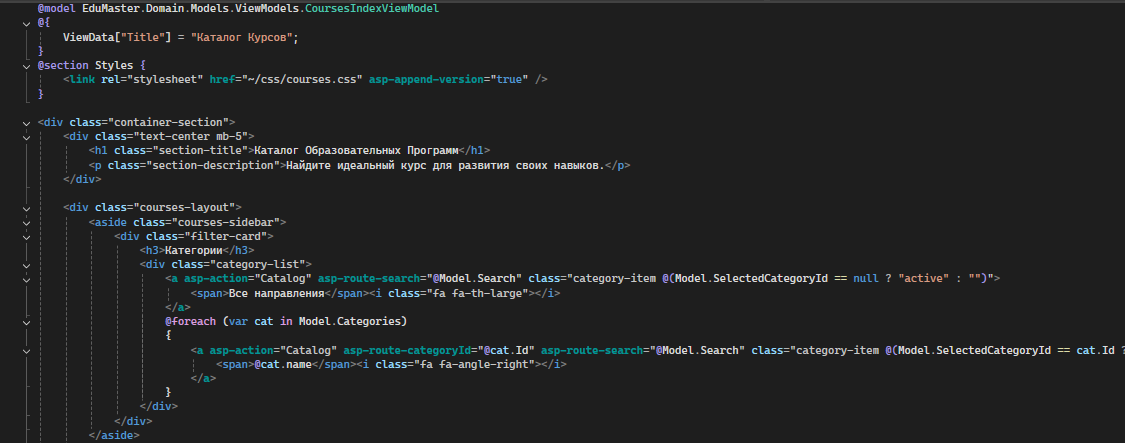


Рисунок 47. Сетка каталога курсов в файле Catalog.cshtml.

Шаг 2. Интеграция изображений

Для каждого курса в цикле foreach был реализован вывод изображения. Так как картинки хранятся в базе данных PostgreSQL в бинарном виде, прямой вывод через путь к файлу невозможен. В контроллере CoursesController был задействован метод GetImage, который возвращает файловый поток. В HTML-разметке это реализовано через конструкцию @Url.Action("GetImage", "Courses", new { courseId = item.Id }). Также добавлена заглушка (placeholder) на случай, если изображение отсутствует.

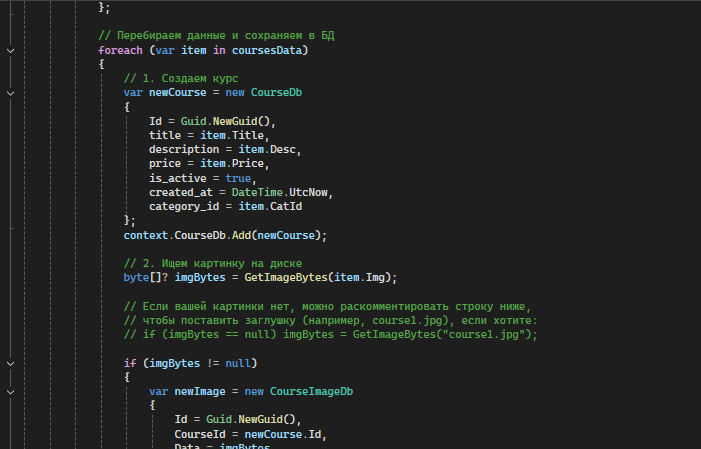


Рисунок 48. Код вывода изображения курса.

**Вывод:** Сегодня был сверстан каркас страницы каталога. Настроена динамическая генерация списка карточек на основе данных из БД и реализована корректная подгрузка изображений через метод контроллера.

# 2.12.2025

**Тема:** Frontend разработка страницы с элементами: реализация фильтров и сортировки

данных.

**Цель:** Улучшить отображение элементов каталога и их структуру.

**Ход работы:**

Шаг 1. Стилизация карточек товаров

В файл courses.css были добавлены стили для класса .course-card. Для улучшения визуального восприятия были настроены отступы, скругление углов (border-radius) и типографика заголовков. Реализован эффект наведения (:hover): при наведении курсора мыши карточка плавно увеличивается в масштабе и отбрасывает тень (box-shadow), создавая эффект "приподнятости" над страницей.

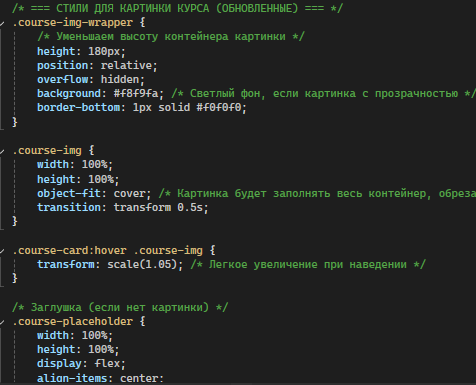


Рисунок 49. CSS-стили для анимации карточки.

Шаг 2. Добавление элементов взаимодействия

В нижнюю часть карточки была добавлена кнопка «Подробнее». Она сверстана в едином стиле с кнопками авторизации. Также добавлено отображение цены курса с форматированием валюты. Для текстового описания курса применено CSS-свойство, ограничивающее количество строк, чтобы карточки всегда имели одинаковую высоту, независимо от длины текста.

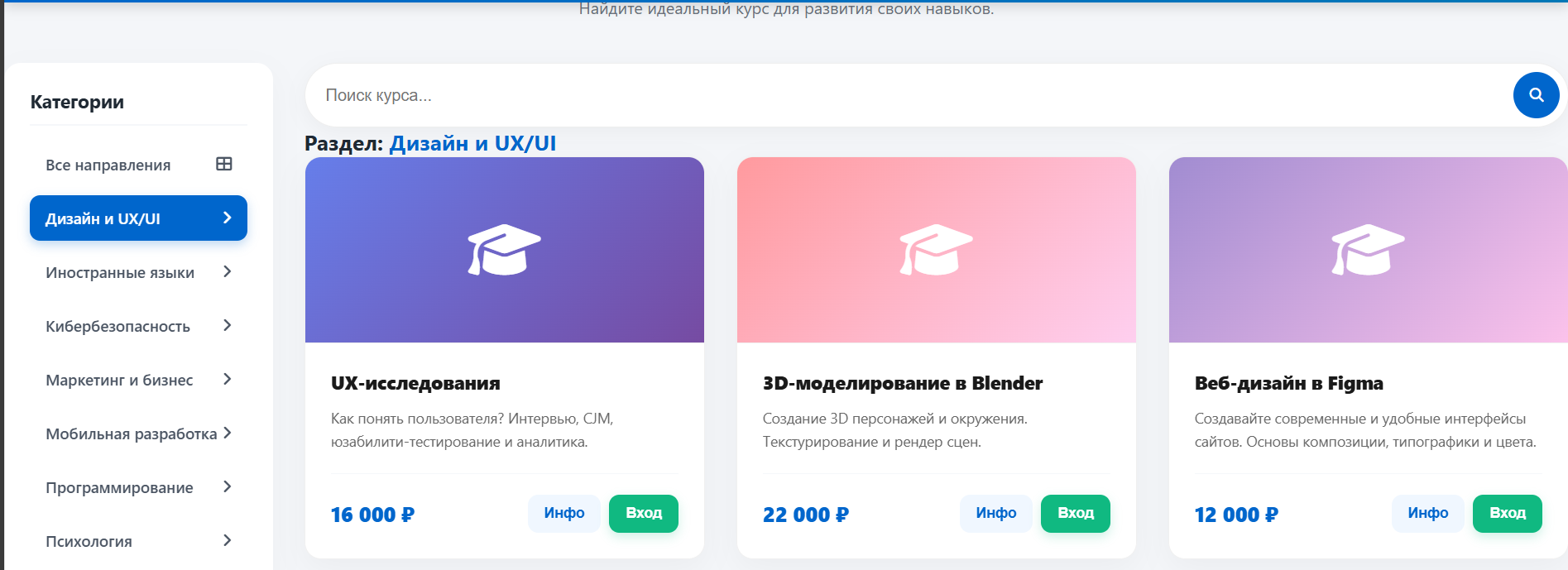


Рисунок 50. Итоговый вид карточки курса.

**Вывод:** Завершена визуальная часть каталога. Интерфейс стал более интерактивным и приятным для пользователя благодаря CSS-анимациям и аккуратной верстке.

# 3.12.2025

**Тема:** Frontend доработка страницы с элементами: реализация фильтров и сортировки данных.

**Цель:** Реализовать фильтры и сортировку данных.

**Ход работы:**

Шаг 1. Верстка боковой панели (Сайдбар)

Страница каталога была разделена на две колонки. В левой части (col-md-3) создан сайдбар для размещения инструментов фильтрации. В сайдбаре реализована HTML-форма с методом GET, содержащая поле ввода для текстового поиска (<input type="text" name="search">) и кнопку «Найти». Использование метода GET позволяет передавать параметры фильтрации прямо в адресной строке.

Шаг 2. Вывод категорий

Реализован динамический вывод списка категорий в сайдбаре. Данные берутся из свойства Model.Categories. Каждая категория оформлена как ссылка-фильтр. Добавлена логика проверки: если ID категории совпадает с текущим выбранным ID (Model.SelectedCategoryId), ссылке присваивается CSS-класс active, выделяющий её цветом.

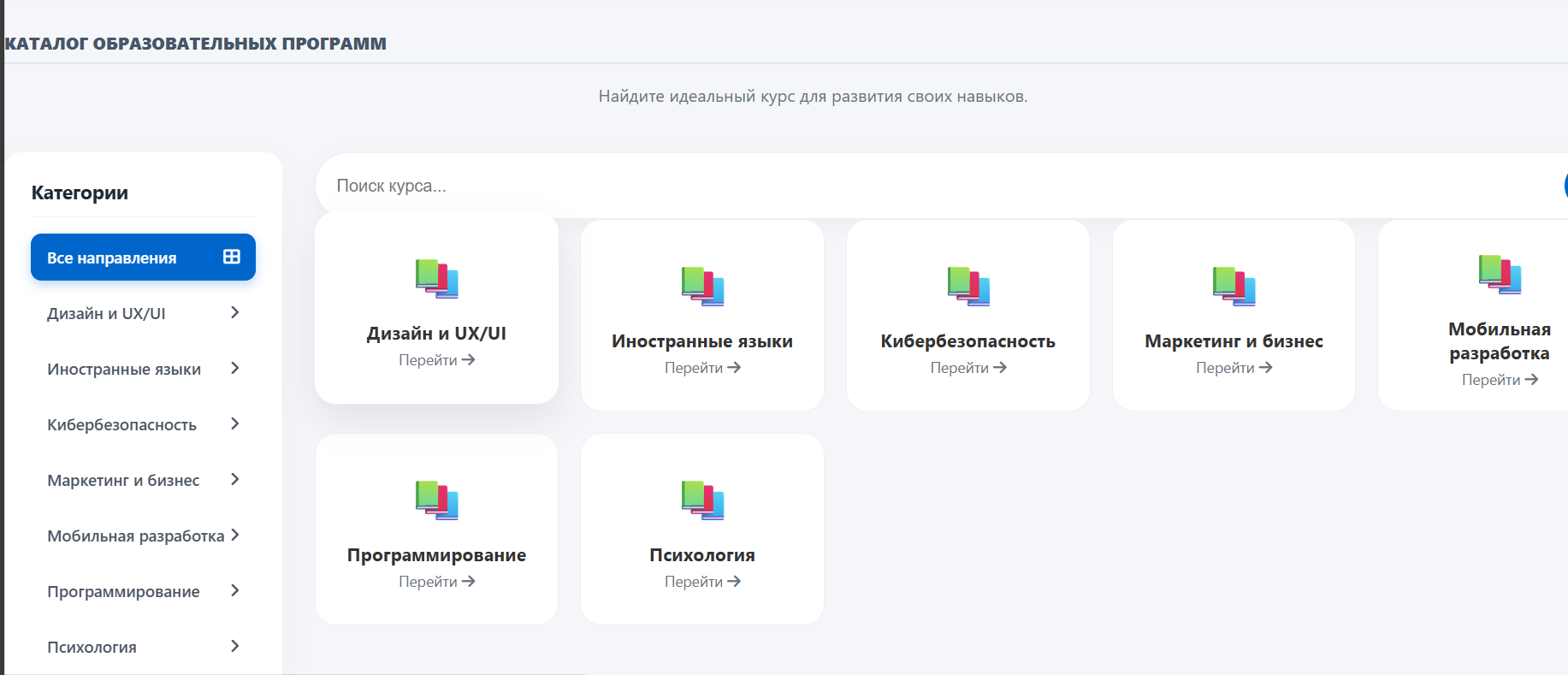


Рисунок 51. Список категорий с подсветкой активного элемента.

**Вывод:** Разработан пользовательский интерфейс для фильтрации. Пользователь получил удобные инструменты для поиска курсов по названию и навигации по категориям.

# 4.12.2025

**Тема:** Разработка сервисов для реализации фильтров на странице.

**Цель:** Доработать и улучшить фильтры и сортировку.

**Ход работы:**

Шаг 1. Сохранение состояния фильтров

Для улучшения UX (пользовательского опыта) была реализована логика сохранения введенных данных. В атрибут value поля поиска теперь подставляется значение @Model.Search из ViewModel. Это гарантирует, что после перезагрузки страницы введенный пользователем текст не исчезнет, и он сможет его отредактировать.

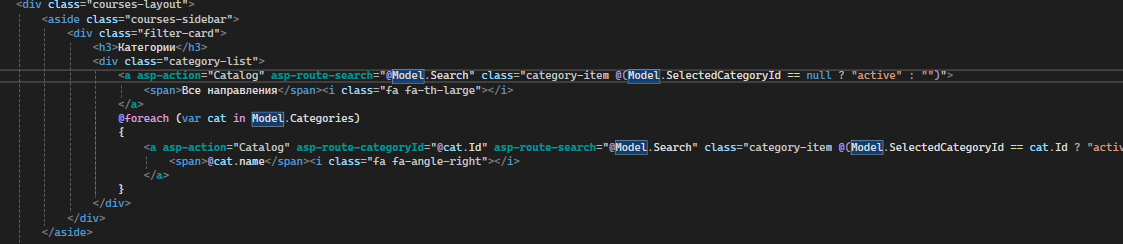


Рисунок 52. Код поля ввода с сохранением значения.

Шаг 2. Адаптивность и сброс

Добавлена кнопка «Сбросить фильтры», которая перенаправляет пользователя на чистый URL /Courses/Catalog, очищая все параметры запроса. С помощью медиа-запросов (@media) настроено поведение фильтров на мобильных устройствах: сайдбар перемещается в верхнюю часть страницы и сворачивается в компактный блок, чтобы не перекрывать основной контент.

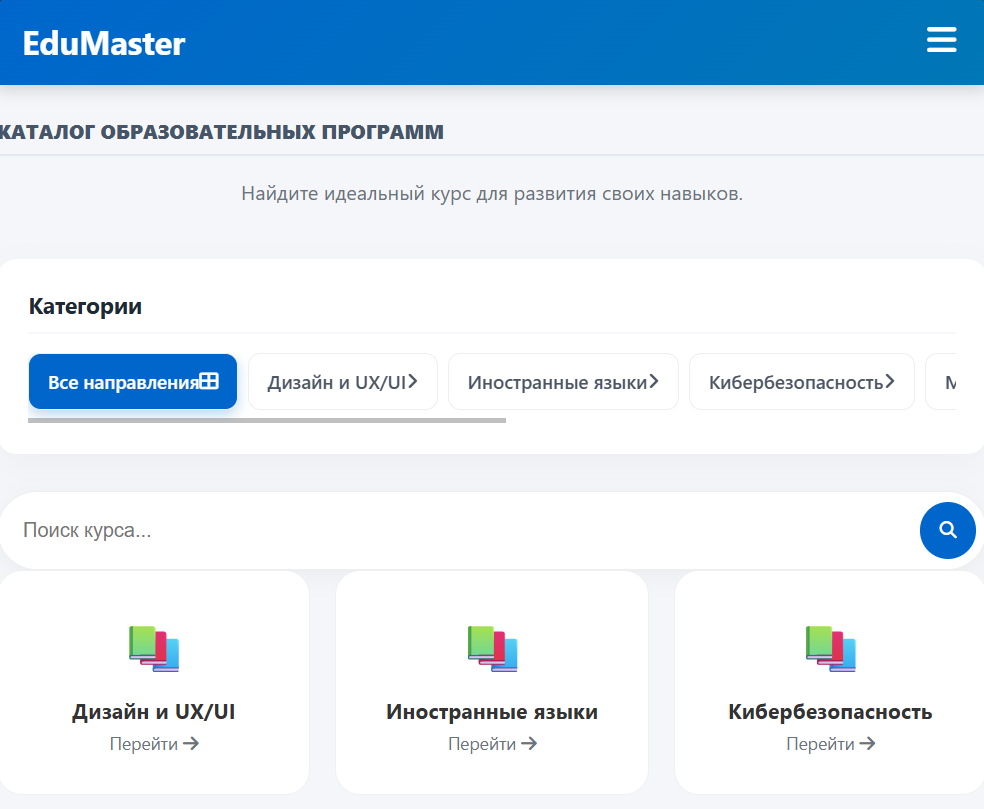


Рисунок 53. Отображение фильтров на мобильном устройстве.

**Вывод:** Завершена работа над интерфейсом фильтрации. Обеспечена корректная работа форм при перезагрузке страницы и адаптивность под разные разрешения экранов.

# 5.12.2025

**Тема:** Доработка сервисов для реализации фильтров на странице. Оформление отчетной документации.

**Цель:** Разработать backend-сервисы для фильтрации данных.

**Ход работы:**

Шаг 1. Модификация контроллера

Работа перешла на серверную часть (Backend). В метод Catalog контроллера CoursesController были добавлены входные параметры: string? search и Guid? categoryId. Инициализирован запрос к базе данных через интерфейс IQueryable (\_db.CourseDb.AsQueryable()), что позволяет динамически конструировать SQL-запрос.

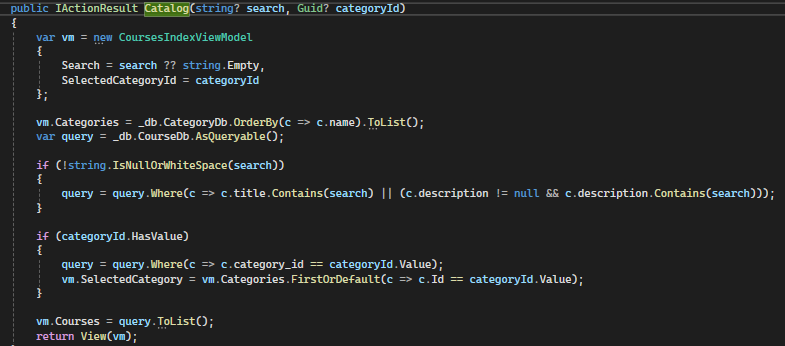


Рисунок 54. Сигнатура метода Catalog в контроллере.

Шаг 2. Логика текстового поиска

Реализована логика фильтрации по строке поиска. Добавлено условие: если параметр search не пустой, к запросу добавляется конструкция .Where, проверяющая вхождение подстроки в название (Title) или описание (Description) курса. Это позволяет искать курсы по ключевым словам.

**Вывод:** Реализован базовый механизм поиска на сервере. Использование отложенного выполнения запросов обеспечивает высокую производительность при фильтрации данных.

# 6.12.2025

**Тема:** Доработка сервисов для реализации фильтров на странице.

**Цель:** Улучшить работу сервисов фильтрации.

**Ход работы:**

Шаг 1. Фильтрация по категориям

В контроллер добавлена обработка параметра categoryId. Если пользователь выбрал категорию, к выборке добавляется строгое условие .Where(c => c.category\_id == categoryId). Также реализована подгрузка названия выбранной категории для отображения в заголовке страницы (например, «Курсы в разделе: Программирование»).

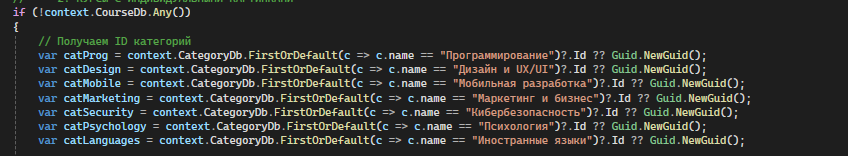


Рисунок 55. Код фильтрации по ID категории.

Шаг 2. Сортировка и сборка модели

Добавлена сортировка списка категорий по алфавиту (.OrderBy(c => c.name)), чтобы фильтры отображались в упорядоченном виде. В финале метода происходит выполнение запроса (.ToList()) и сборка объекта CoursesIndexViewModel, который передается в представление.

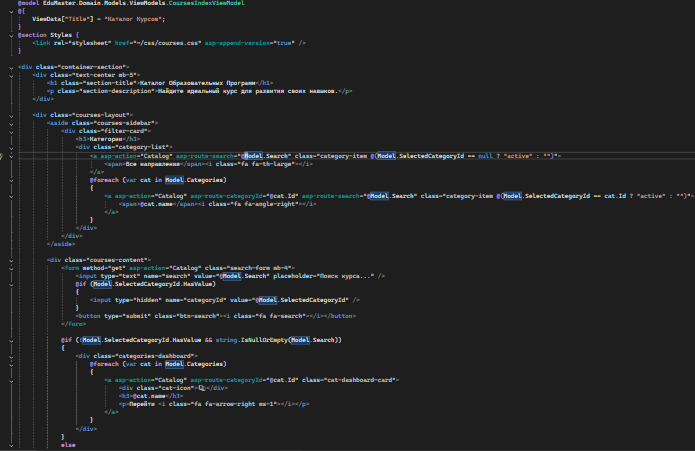


Рисунок 56. Итоговый код метода Catalog.

**Вывод:** Серверная логика фильтрации и сортировки полностью готова. Контроллер корректно обрабатывает комбинации поиска и выбора категорий, возвращая нужный набор данных.

# 8.12.2025

**Тема:** Frontend разработка страница с конкретным элементом

**Цель:** Создать страницу конкретного элемента каталога.

**Ход работы:**

Шаг 1. Создание API-метода

Было принято решение использовать модальное окно для просмотра деталей курса, чтобы не перезагружать страницу. В CoursesController создан новый метод GetCourseDetails(Guid id). Этот метод ищет курс в базе данных по ID и возвращает объект в формате JSON (return Json(...)), содержащий заголовок, описание, цену и категорию курса.

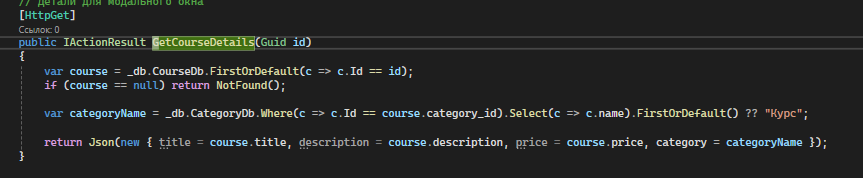


Рисунок 57. API-метод GetCourseDetails.

Шаг 2. Разметка модального окна

В файл \_Layout.cshtml (или отдельное частичное представление) добавлена скрытая HTML-разметка модального окна. Она включает в себя затемненный фон (overlay) и контейнер с пустыми элементами (заголовком, текстовым блоком), которые будут заполняться данными динамически.

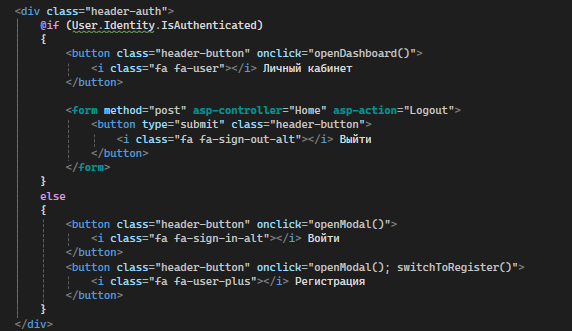


Рисунок 58. HTML-структура модального окна.

**Вывод:** Подготовлен Backend (API endpoint) для получения данных о курсе и Frontend-каркас модального окна.

# 9.12.2025

**Тема:** Frontend доработка страница с конкретным элементом

**Цель:** Доработать страницу конкретного элемента.

**Ход работы:**

Шаг 1. Написание скрипта AJAX

Создан JavaScript файл courses\_script.js. На кнопки «Подробнее» добавлен обработчик событий click. Внутри обработчика используется функция fetch для отправки асинхронного запроса на сервер по адресу /Courses/GetCourseDetails?id

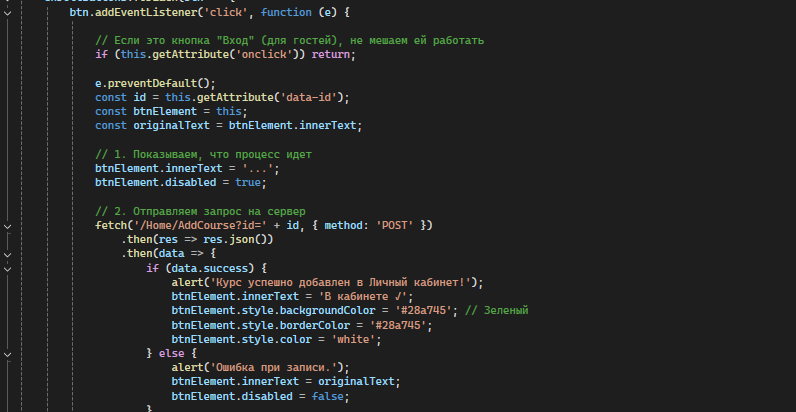


Рисунок 59. JS-код запроса данных о курсе.

Шаг 2. Отображение данных

Реализована логика обработки ответа от сервера. Полученный JSON-объект разбирается, и данные вставляются в соответствующие HTML-элементы модального окна (innerText). Также реализованы функции открытия (openModal) и закрытия окна (по крестику или клику вне области окна)

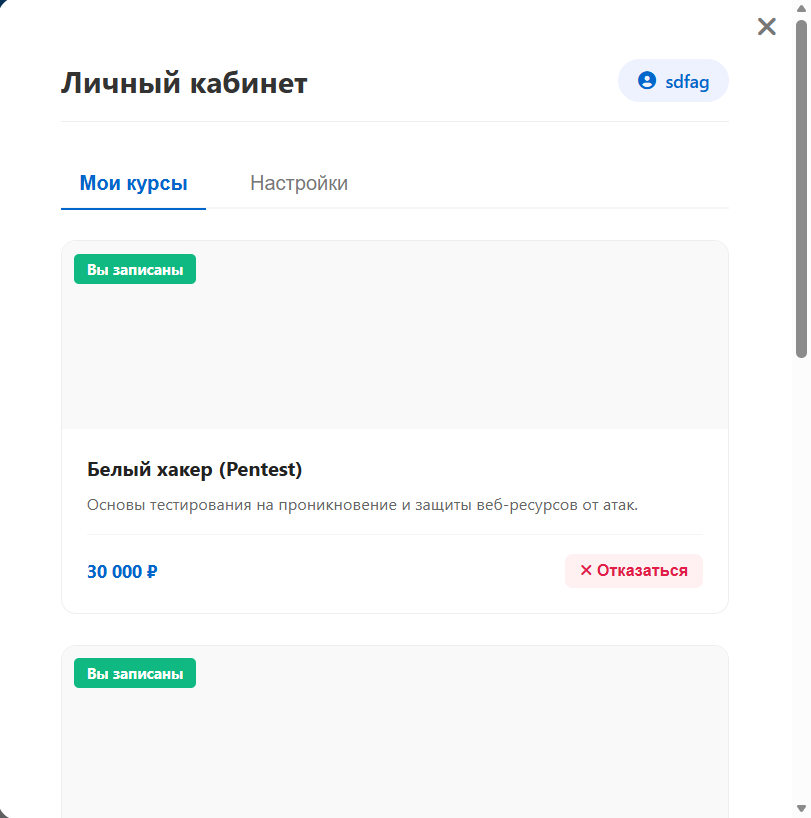


Рисунок 60. Работающее модальное окно с данными.

**Вывод:** Реализован просмотр деталей курса без перезагрузки страницы. Использование AJAX сделало интерфейс быстрым и отзывчивым.

# 10.12.2025

**Тема:** Оформление индивидуального проекта к защите работы. Оформление отчетной документации.

**Цель:** Оформить индивидуальный проект для защиты.

**Ход работы:**

Шаг 1. Тестирование исборка

Проект был переведен в конфигурацию Release. Проведено полное тестирование всех пользовательских сценариев: регистрация, вход, поиск курсов, просмотр деталей, выход из системы. Проверена работа валидации форм и обработки ошибок.

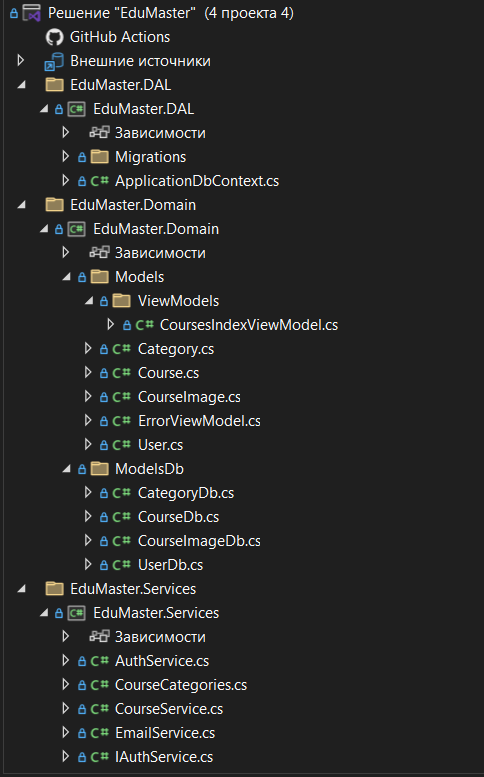


Рисунок 62. Структура проекта в обозревателе решений.

Шаг 2. Написание пояснительной записки

Составлена пояснительная записка. В ней описана архитектура приложения (MVC), структура базы данных PostgreSQL (таблицы Users, Courses, Categories), а также используемый технологический стек (ASP.NET Core 8, EF Core). В документ добавлены скриншоты интерфейса и ключевых фрагментов кода.

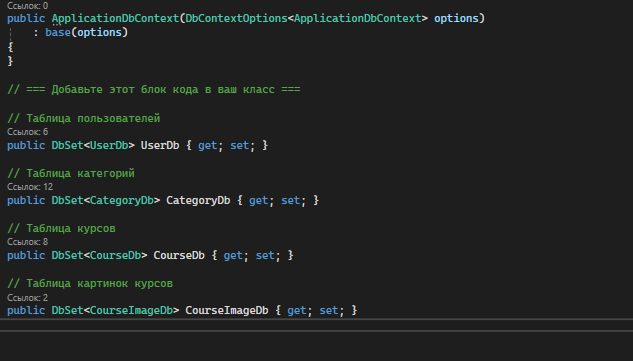


Рисунок 63. Фрагмент пояснительной записки.

**Вывод:** Программный продукт полностью отлажен и готов к демонстрации. Документация оформлена в соответствии с требованиями.

# 11.12.2025

**Тема:** Оформление презентации к защите работы. Дифференцированный зачет.

**Цель:** Создать презентацию и подготовиться к дифференцированному зачёту.

**Ход работы:**

Шаг 1. При составлении презентации первыми слайдами идет приветствие, а также краткое описание проекта EduMaster и для чего он нужен. В них говорится об актуальности онлайн-образования и преимуществах платформы.

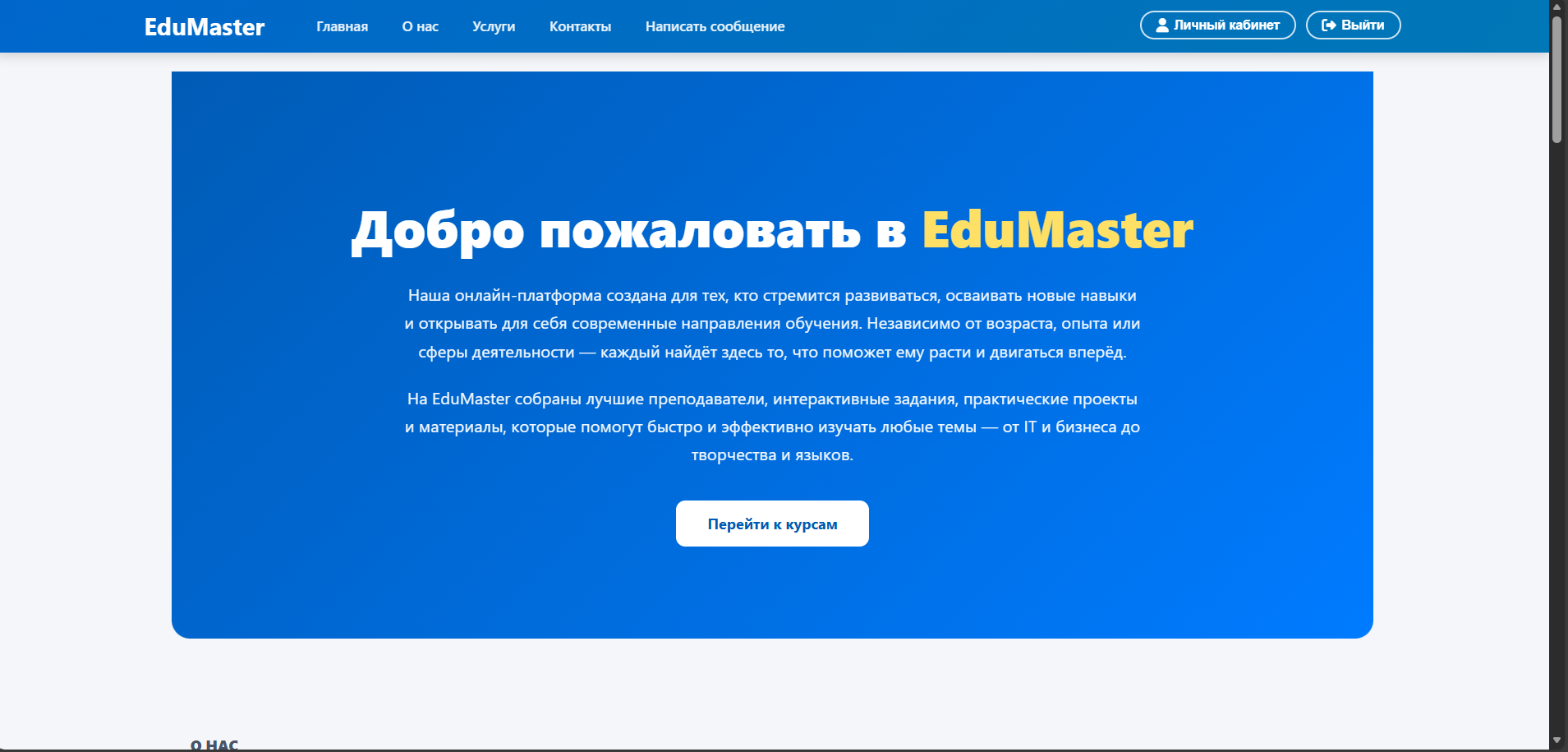


Рисунок 64. Главная страница

Шаг 2**.** Во второй части уже рассказывается что я использовал при создании сайта, а именно такой язык программирования как C# и JS, MVC шаблон, СУБД PostgreSQL, а также html с css.

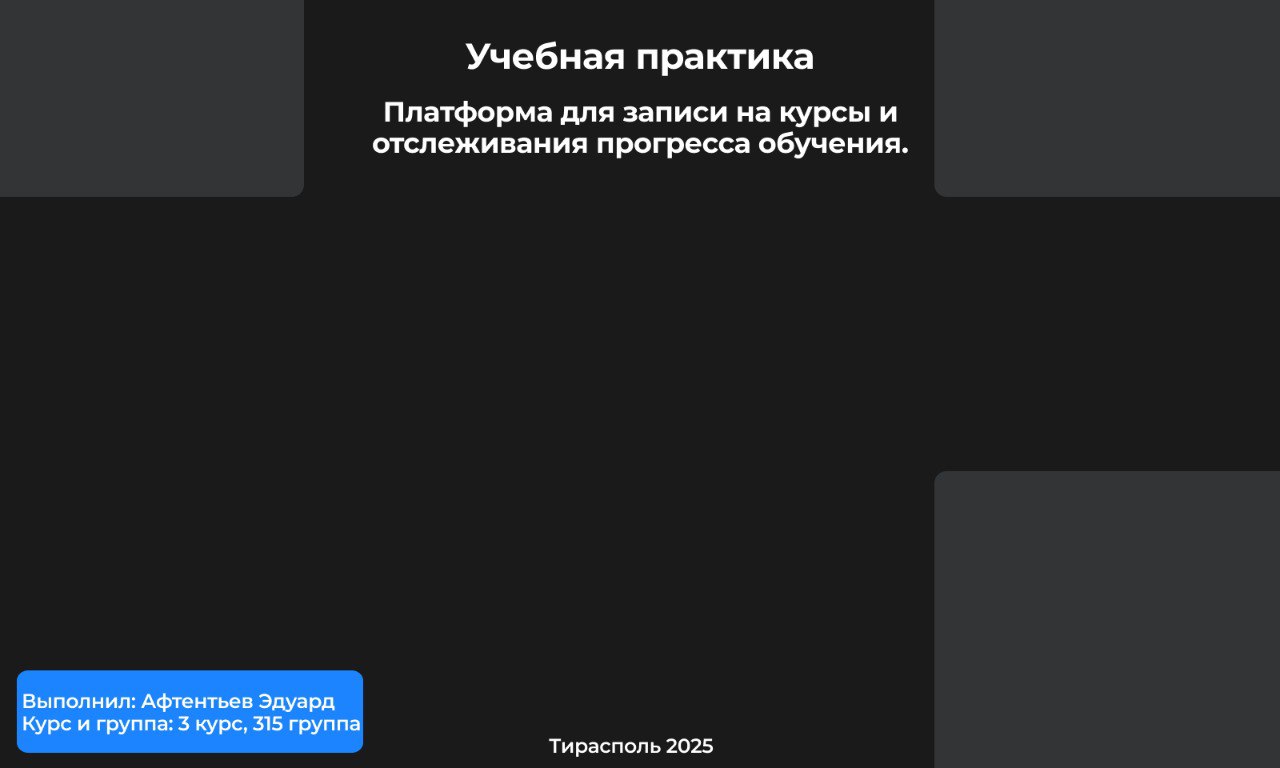


Рисунок 65. Первый слайд презентации

**Вывод:** Создание образовательной платформы — это важный и актуальный инструмент для развития. Оно позволяет демонстрировать курсы, находить новых студентов и укреплять бренд. Разработка сайта с использованием современных технологий, таких как C#, JavaScript, MVC, PostgreSQL, HTML и CSS, обеспечивает функциональность, удобство и эстетичный внешний вид, что делает платформу удобной и привлекательной как для преподавателей, так и для студентов.

# ПРИЛОЖЕНИЕ А

|  |  |
| --- | --- |
| AuthService.cs  namespace EduMaster.Services  {  public class AuthService : IAuthService  {  private readonly ApplicationDbContext \_db;  private readonly IPasswordHasher \_hasher;  private readonly IHttpContextAccessor \_httpContextAccessor;  public AuthService(ApplicationDbContext db, IPasswordHasher hasher, IHttpContextAccessor httpContextAccessor)  {  \_db = db;  \_hasher = hasher;  \_httpContextAccessor = httpContextAccessor;  }  public async Task<bool> RegisterAsync(string email, string login, string password)  {  bool exist = await \_db.UserDb.AnyAsync(u => u.Email == email || u.Login == login);  if (exist) return false;  var user = new UserDb  {  Id = Guid.NewGuid(),  Email = email,  Login = login,  PasswordHash = \_hasher.HashPassword(password),  Role = "Student",  CreatedAt = DateTime.UtcNow  };  \_db.UserDb.Add(user);  await \_db.SaveChangesAsync();  // После успешной регистрации автоматически входим  await SignInUserAsync(user.Login, user.Email, user.Role);  return true;  }  public async Task<bool> LoginAsync(string login, string password)  {  var user = await \_db.UserDb.FirstOrDefaultAsync(u => u.Login == login || u.Email == login);  if (user == null) return false;  if (!\_hasher.VerifyPassword(password, user.PasswordHash)) return false;  // Создаём Cookie-аутентификацию  await SignInUserAsync(user.Login, user.Email, user.Role);  return true;  }  private async Task SignInUserAsync(string login, string email, string role)  {  var httpContext = \_httpContextAccessor.HttpContext;  if (httpContext == null) return;  var claims = new List<Claim>  {  new Claim(ClaimTypes.Name, login ?? string.Empty),  new Claim(ClaimTypes.Email, email ?? string.Empty),  new Claim(ClaimTypes.Role, role ?? string.Empty)  };  var identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);  var principal = new ClaimsPrincipal(identity);  await httpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, principal);  }  }  } | Index.cshtml  @{  ViewBag.Title = "EduMaster — Главная";  }  <main>  <!-- HERO -->  <section id="home-section" class="page-section home-hero">  <div class="hero-content">  <h1>Добро пожаловать в <span class="highlight">EduMaster</span></h1>  <p class="hero-subtitle">  Наша онлайн-платформа создана для тех, кто стремится развиваться,  осваивать новые навыки и открывать для себя современные направления обучения.  Независимо от возраста, опыта или сферы деятельности — каждый найдёт здесь  то, что поможет ему расти и двигаться вперёд.  </p>  <p class="hero-subtitle">  На EduMaster собраны лучшие преподаватели, интерактивные задания,  практические проекты и материалы, которые помогут быстро и эффективно изучать  любые темы — от IT и бизнеса до творчества и языков.  </p>  <a href="#features-section" class="hero-btn">Перейти к курсам</a>  </div>  </section>  <!-- О НАС -->  <section id="about-section" class="page-section about">  <h1 class="section-title">О нас</h1>  <div class="about-grid">  <div class="about-text">  <h2>Образовательная платформа нового поколения</h2>  <p>  <strong>EduMaster</strong> — это современная система цифрового обучения,  созданная для студентов, преподавателей и всех желающих развивать свои навыки.  Мы стремимся объединить инновации, профессионализм и доступность,  чтобы предоставить каждому удобный способ получать качественное образование.  </p>  <p>  Наша цель — сделать процесс обучения максимально простым, понятным и интересным.  Мы предлагаем структурированные курсы, гибкое расписание, поддержку наставников  и актуальные материалы, разработанные специалистами с большим опытом работы.  </p>  <p>  Учитесь в удобное время, в комфортном темпе, из любой точки мира —  всё, что вам нужно, это доступ к сети и желание учиться.  </p>  <ul class="about-benefits">  <li><i class="fa-solid fa-chalkboard-user"></i> Преподаватели-практики с реальным опытом</li>  <li><i class="fa-solid fa-certificate"></i> Сертификаты, подтверждающие навыки</li>  <li><i class="fa-solid fa-users"></i> Сообщество студентов и наставников</li>  <li><i class="fa-solid fa-clock"></i> Гибкое обучение в удобном темпе</li>  </ul>  </div>  <div class="about-image">  <**img** src="~/img/command.jpg" alt="Наша команда" />  </div>  </div>  <!-- Статистика -->  <div class="about-extra">  <div class="stat">  <h3>10,000+</h3>  <p>Активных студентов со всего мира</p>  </div>  <div class="stat">  <h3>250+</h3>  <p>Опытных преподавателей и экспертов</p>  </div>  <div class="stat">  <h3>500+</h3>  <p>Курсов, программ и направлений</p>  </div>  </div>  </section>    <!-- ОТЗЫВЫ -->  <section id="reviews-section" class="page-section reviews">  <h1 class="section-title">Отзывы студентов</h1>  <div class="reviews-container">  <div class="review-card">  <p>  “EduMaster стал для меня отправной точкой в IT.  Поддержка наставников, качественные задания и удобная подача материала  буквально изменили мою карьеру.”  </p>  <h3>— Марина К.</h3>  </div> |