**МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ ПМР**

**ГОУ СПО «ТИРАСПОЛЬСКИЙ ТЕХНИКУМ ИНФОРМАТИКИ И ПРАВА»**

**ДНЕВНИК**

**ПРОХОЖДЕНИЯ ПРАКТИКИ**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**УП.01.01\_Учебная практика**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

вид практики

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

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

индекс, наименование профессионального модуля

Междисциплинарный курс \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

индекс, наименование междисциплинарного курса

Ф.И.О. студента(ки)\_\_\_\_\_\_\_Шеленков Кирилл Вячеславович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Специальность \_\_2.09.02.07 Информационные системы и программирование

код, наименование специальности

Курс 3 группа 313

Время прохождения практики: с «\_31\_» \_10\_ 20\_25\_ г. по «\_11\_» \_12\_ 20\_25\_ г.

База практики \_\_ГОУ СПО «Тираспольский техникум информатики и права»\_\_

**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 работы:** закрепить теоретические знания через выполнение практических заданий, сформировать профессиональные компетенции, освоить правила техники безопасности и поведения во время практики, а также научиться создавать проект на основе архитектурного шаблона MVC в среде Visual Studio.

**Ход работы**

Введение в практическое обучение

Целью моей учебной практики было не только закрепление теоретических знаний, но и получение практических навыков, которые необходимы для дальнейшего профессионального роста. Через выполнение реальных задач на практике я смог лучше понять, как работает теоретическая база, и какие навыки мне нужно развивать для успешной карьеры.

Формирование компетенций

Для меня основной задачей в ходе практики было развитие компетенций, которые включают в себя знания, умения и личностные качества, необходимые для работы в будущей профессии. В процессе работы я приобрел опыт, который связан с решением реальных профессиональных задач.

Инструктаж по технике безопасности

Перед началом практики мне был проведен обязательный инструктаж по технике безопасности. Это очень важный этап, так как помогает избежать травм и других неблагоприятных ситуаций. В ходе инструктажа мне объяснили правила поведения в случае чрезвычайных ситуаций и требования к безопасному выполнению работы.

Я ознакомился с основными правилами безопасности, такими как:

1. Обязанности студентов по соблюдению правил.
2. Использование защитных средств, если это необходимо.
3. Как действовать в экстренных ситуациях.

Правила поведения на учебной практике

Кроме соблюдения техники безопасности, на практике также важно было соблюдать нормы поведения. Я должен был быть дисциплинированным, ответственным и следовать указаниям руководителей практики. Это включало:

1. Соблюдение трудовой дисциплины и рабочего времени.
2. Взаимодействие с коллегами и преподавателями.
3. Исполнение инструкций и требований

4. Создание проекта на основе шаблона MVC

Что такое MVC?

Моя задача на практике заключалась в создании проекта с использованием шаблона архитектуры MVC (Model-View-Controller). Этот шаблон разделяет приложение на три компонента: модель, представление и контроллер. Такой подход позволяет эффективно управлять кодом и улучшать структуру проекта.

Шаги создания проекта

Для начала работы с проектом я открыл среду разработки Visual Studio. Это основное средство, которое я использую для создания и настройки проектов на платформе .NET

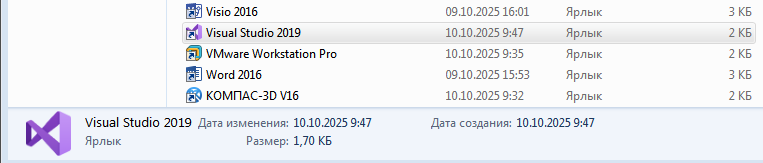


Рисунок 1 – Запуск Visual Studio

После открытия Visual Studio я выбрал опцию создания нового проекта. Это стандартный шаг для начала работы над любым новым приложением.

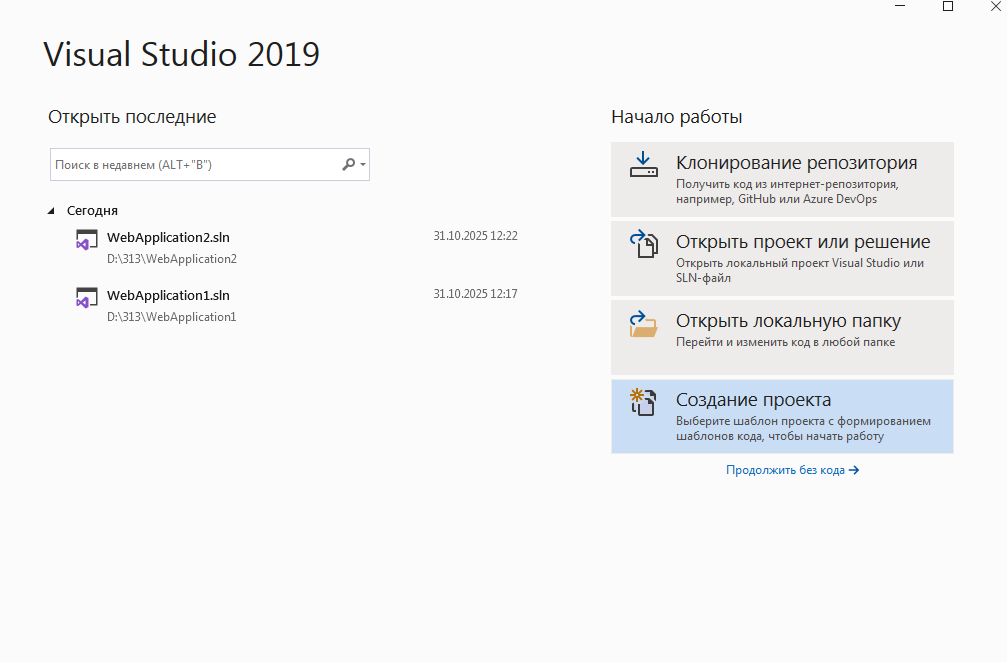


Рисунок 2 – создание проекта

На следующем этапе я выбрал шаблон проекта ASP.NET Core MVC. Это позволит создать приложение с архитектурой Model-View-Controller, которая разделяет логику приложения на три основные части.

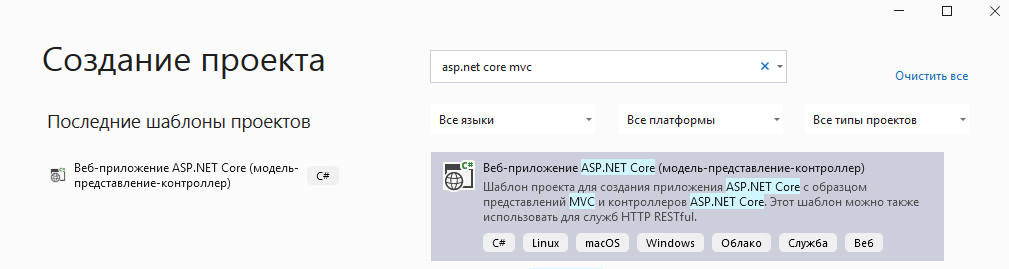


Рисунок 3 – выбор проекта

Затем я настроил параметры проекта, такие как имя, расположение, версия .NET и другие параметры. Это важно для правильной настройки окружения и конфигурации проекта.

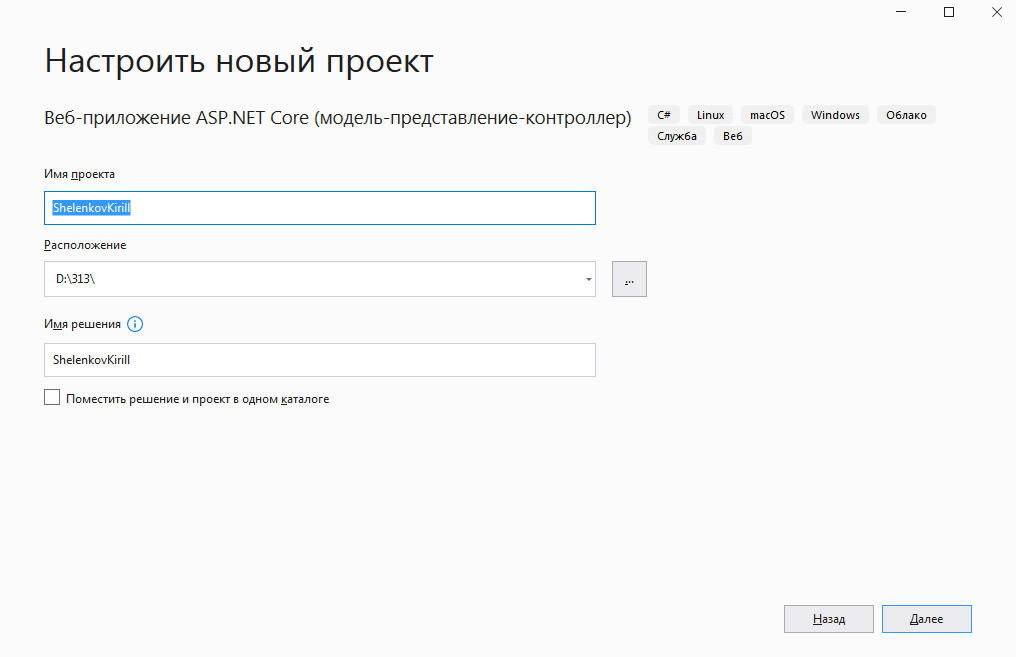


Рисунок 4 – настройка проекта

В этом разделе я уточнил параметры, такие как поддержка Docker, тестирование и другие настройки, которые могут понадобиться для специфических нужд проекта.

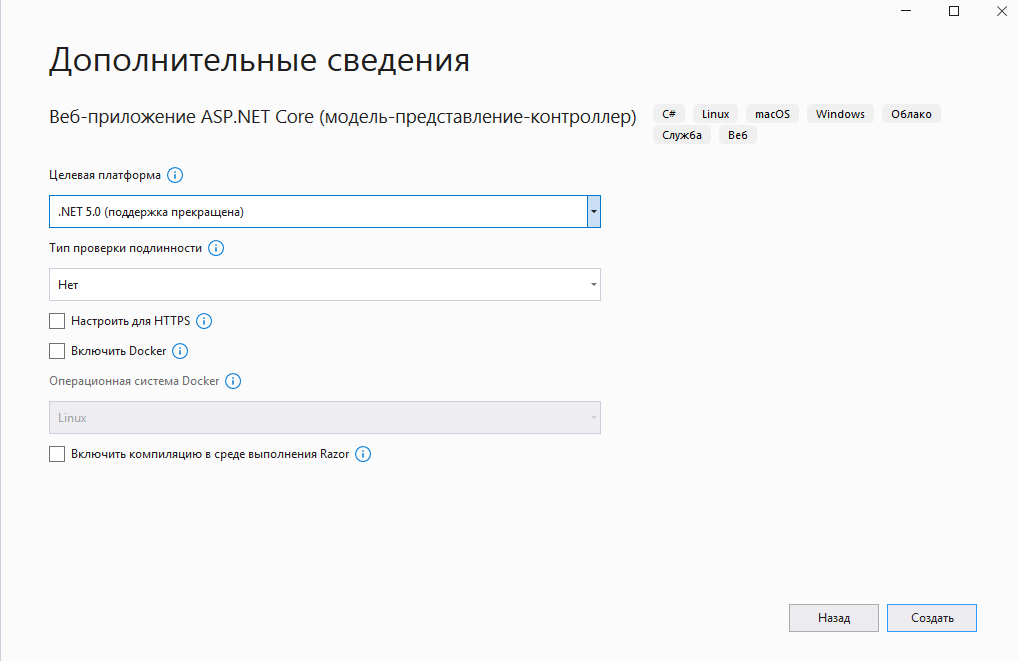


Рисунок 5 - **Дополнительные сведения**

После настройки проекта я открыл его в Visual Studio. На этом этапе я уже мог работать с проектом, используя все его компоненты и функционал.

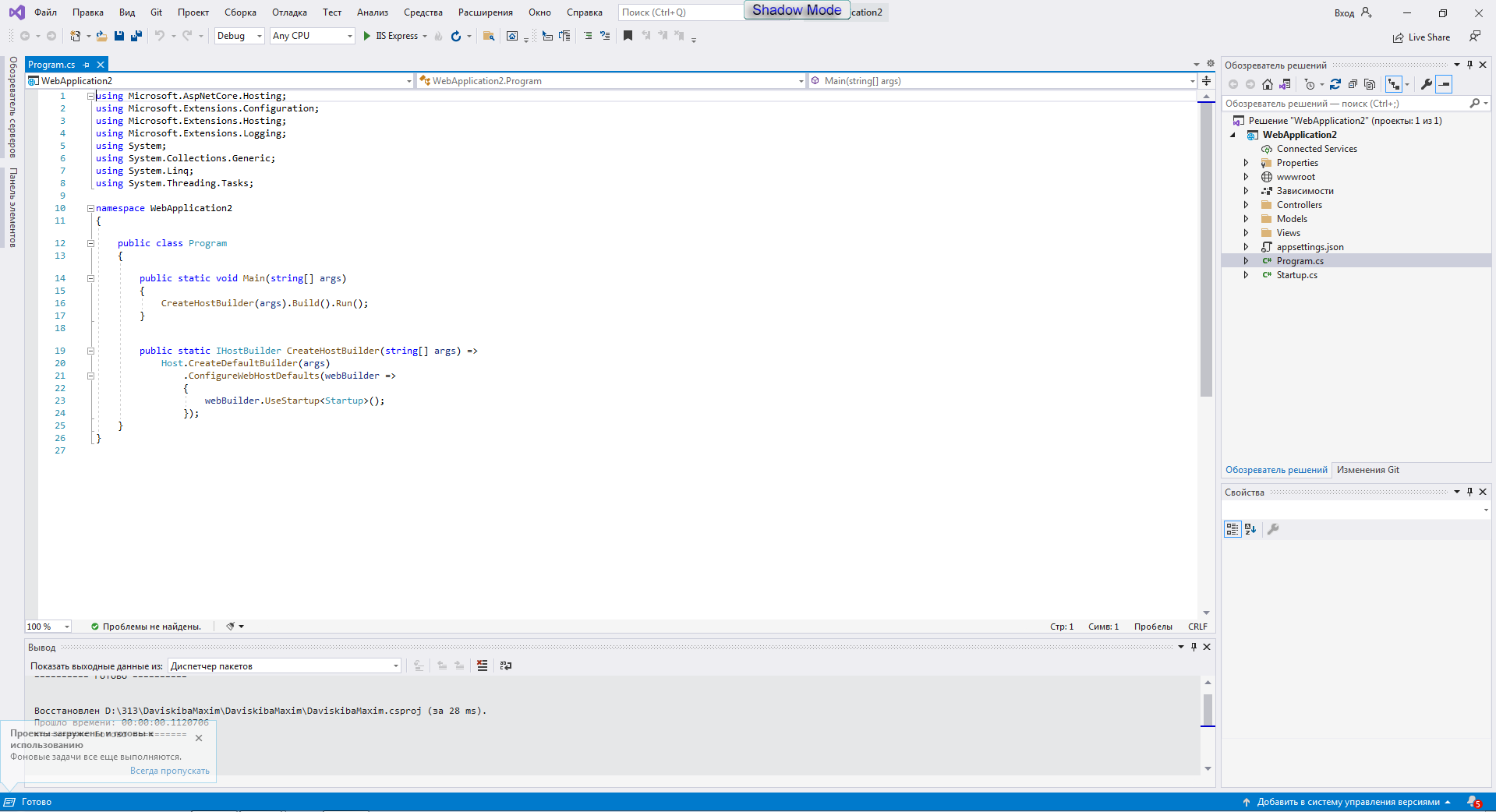


Рисунок 6 – созданный проект

Для проверки работоспособности проекта я запустил его в браузере. Это позволило мне убедиться, что проект работает корректно и отображает нужную информацию.

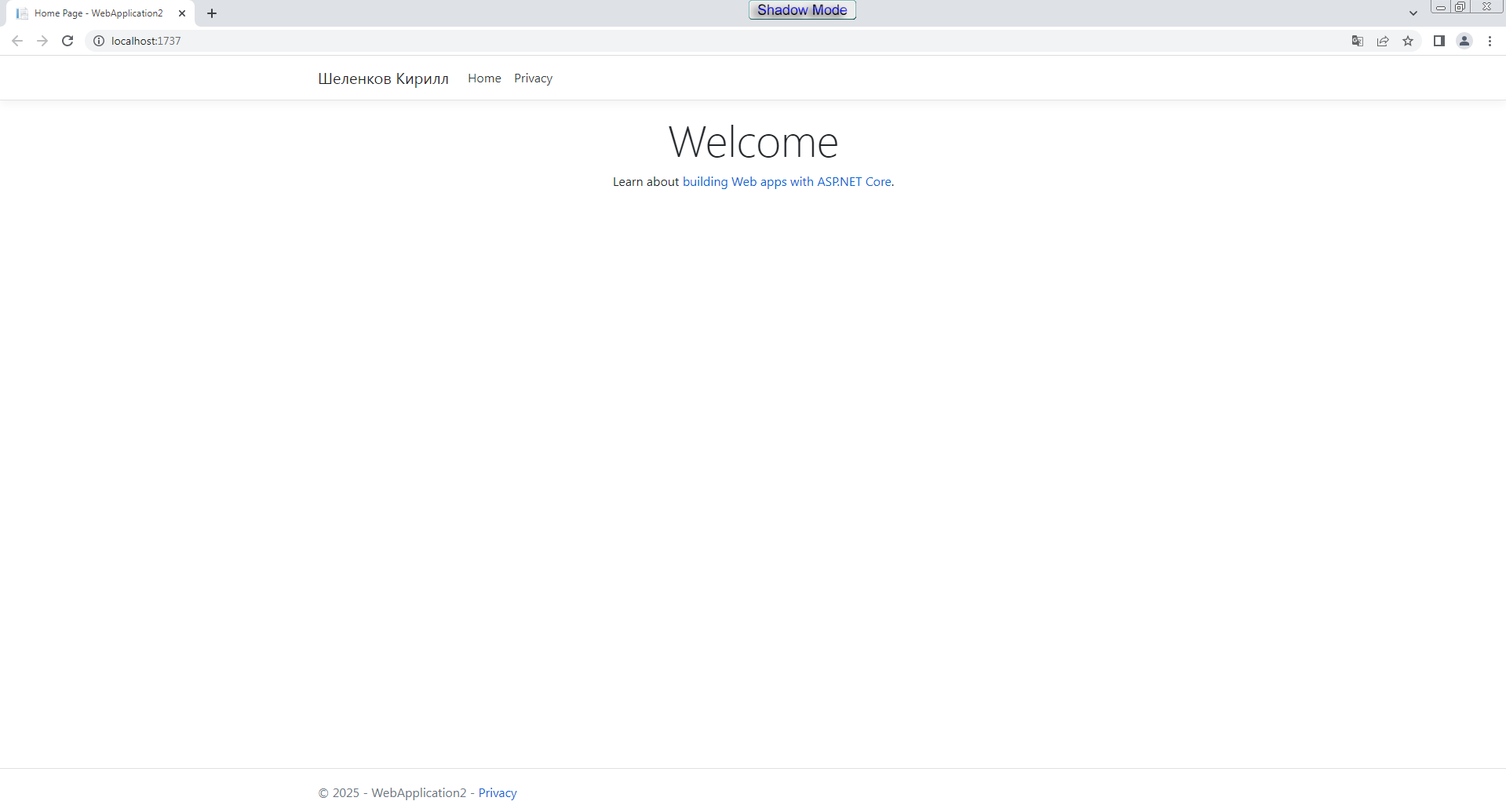


Рисунок 7 - **Скриншот из браузера запущенного проекта**

Вывод: в результате выполнения всех шагов я создал новый проект с использованием шаблона ASP.NET Core MVC. Каждый этап, начиная с установки Visual Studio и выбора шаблона, до настройки проекта и запуска его в браузере, позволил мне понять ключевые принципы работы с архитектурой MVC. Такой подход к разработке приложения значительно упрощает его поддержку и расширение. На примере этого проекта я убедился в эффективности разделения логики на модель, представление и контроллер, что способствует более чистой и структурированной кодовой базе.

**1.11.2025**

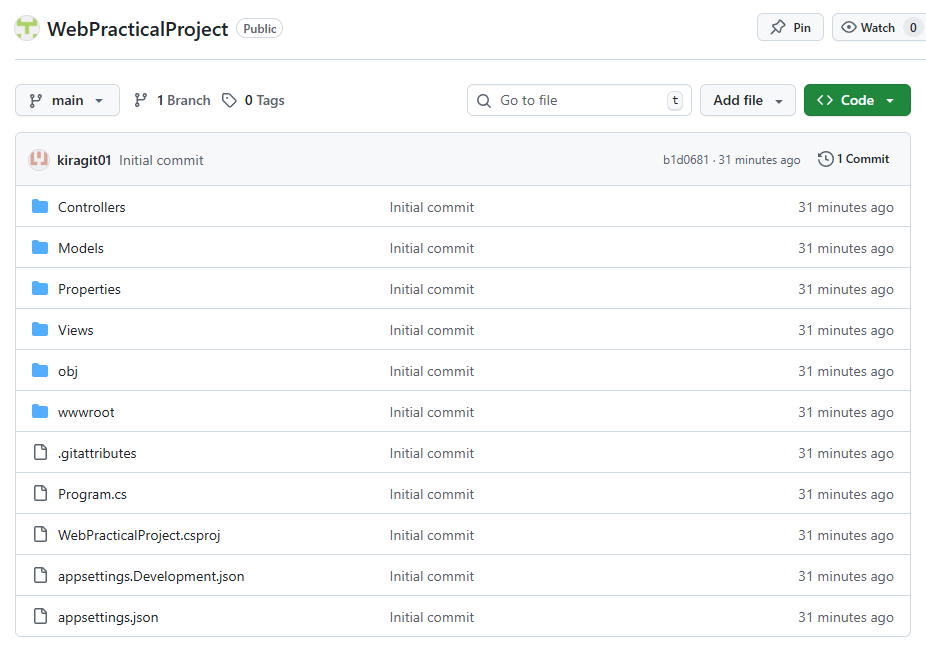
**Тема:** Использование Git для контроля версий. Оформление отчетной документации.

**Цель работы:** освоить принципы работы с системой контроля версий Git, научиться создавать и публиковать репозиторий проекта на GitHub, выполнять фиксацию изменений и вести историю разработки для обеспечения командной и безопасной работы над проектом.

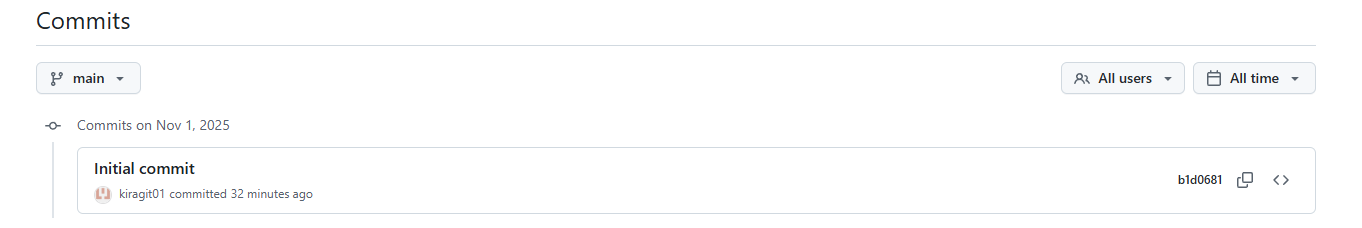
**Ход работы**

В современном процессе разработки программного обеспечения система контроля версий Git используется для отслеживания изменений в проекте, совместной работы и безопасного хранения кода. Благодаря Git можно вернуться к любой предыдущей версии проекта, анализировать историю изменений, а также синхронизировать работу нескольких разработчиков. Работа с Git через GitHub Desktop упрощает использование системы — позволяет управлять репозиториями через графический интерфейс без необходимости вводить команды в терминале.

Для работы с системой был создан новый репозиторий на GitHub, после чего в GitHub Desktop был выполнен его клон на локальный компьютер. В локальную папку репозитория добавлен проект WebPracticalProject, созданный на платформе ASP.NET MVC. После добавления всех исходных файлов проекта было выполнено первое сохранение изменений (commit) с кратким описанием, например: «Initial commit – added MVC project structure». Далее проект был опубликован на GitHub, что позволило хранить его в облачном репозитории и иметь возможность отслеживать дальнейшие изменения.

Рисунок 8 — Публикация проекта WebPracticalProject в GitHub

В процессе работы с Git можно фиксировать каждое изменение в коде, добавлять комментарии к коммитам, а также создавать отдельные ветки для разработки новых функций. Это обеспечивает прозрачность разработки и возможность безопасно тестировать новые решения, не нарушая стабильную версию проекта.

  
Рисунок 9 — История изменений и веток проекта в GitHub Desktop

**Вывод:** Использование Git и GitHub Desktop обеспечивает надежный контроль версий, сохраняет историю изменений проекта и облегчает коллективную работу. Интеграция с ASP.NET MVC-проектом позволяет безопасно хранить и обновлять код на всех этапах разработки, исключая риск потери данных.

**3.11.2025**

**Тема:** Определение структуры MVC-приложения, взаимодействие моделей, контроллеров и представлений. Оформление отчетной документации.

**Цель работы:** изучить принципы взаимодействия компонентов архитектуры MVC, определить структуру многослойного приложения, научиться различать назначение слоёв Domain, DAL и Service и понимать их взаимосвязь при построении MVC-проекта.

**Ход работы**

Структура MVC-приложения основана на разделении логики программы на три взаимосвязанные части: Model, View и Controller. Такой подход позволяет обеспечить гибкость, читаемость и масштабируемость кода, а также упростить тестирование и сопровождение проекта.

Архитектура проекта «WebPracticalProject» построена с применением принципов многослойного проектирования, где каждый уровень имеет чётко определённое назначение и взаимодействует только с теми слоями, которые находятся рядом. Это создаёт гибкую и логичную структуру, соответствующую принципам «чистой архитектуры» и паттерну «слойной модели» (Layered Architecture).

В основе лежат три ключевых проекта: Domain, DAL и Service, каждый из которых играет отдельную роль в формировании бизнес-логики приложения.

Слой Domain является центральным и определяет базовые структуры данных и общие типы, используемые во всём приложении. Он не зависит от других уровней, что делает его универсальной основой. В Domain могут находиться перечисления, контракты (интерфейсы), модели DTO и любые вспомогательные классы, которые не связаны с конкретной технологией хранения данных или реализации логики. Благодаря этому Domain остаётся устойчивым к изменениям инфраструктуры — например, при смене способа хранения данных или логики доступа к ним структура Domain не меняется.

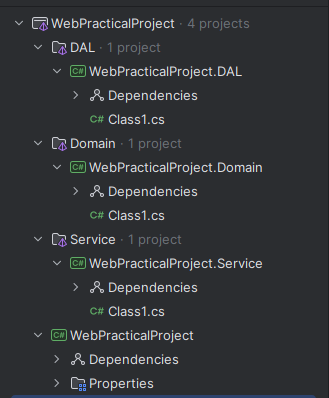
Слой DAL (Data Access Layer) реализует взаимодействие с источником данных. Он зависит от Domain, так как оперирует его типами и моделями. В данном проекте DAL отвечает за получение, сохранение и обновление данных, используя классы и интерфейсы, определённые в Domain. Таким образом, если в Domain описан, например, интерфейс репозитория, то DAL предоставляет его конкретную реализацию — например, обращающуюся к базе данных через ORM или другие механизмы.

Слой Service располагается выше DAL и также зависит от Domain, а через него — косвенно от DAL. Он реализует бизнес-логику приложения, объединяя операции разных источников данных и определяя последовательность действий, необходимых для выполнения бизнес-задачи. В этом уровне описываются сервисные классы, которые используют репозитории из DAL, применяют бизнес-правила, выполняют проверки и возвращают результат контроллерам или другим компонентам системы.

Такое построение обеспечивает строгую иерархию зависимостей:

* Domain не зависит ни от одного слоя;
* DAL зависит от Domain, так как использует его типы;
* Service зависит от Domain и использует DAL для работы с данными.

Визуально такая структура представлена на рисунке 10. В дереве проекта видно, что каждый уровень оформлен отдельным проектом внутри решения: «WebPracticalProject.Domain», «WebPracticalProject.DAL» и «WebPracticalProject.Service», а основной проект «WebPracticalProject» связывает их в единое MVC-приложение, выступая точкой входа и обеспечивая взаимодействие контроллеров с сервисами.

  
Рисунок 10 – Многоуровневая структура проекта «WebPracticalProject» и зависимость слоёв Domain, DAL и Service

Каждый слой имеет собственное назначение:

Благодаря подобной схеме обеспечивается высокая модульность и изоляция логики. Например, можно заменить реализацию DAL, не затрагивая бизнес-логику Service, либо расширить Domain новыми типами, не изменяя существующий код в других слоях.

**Вывод:** использование зависимости Service → DAL → Domain формирует устойчивую архитектуру, где каждый уровень решает строго определённые задачи. Такая структура упрощает сопровождение, тестирование и масштабирование проекта, делая код понятным и логически организованным.

**4.11.2025**

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

**Цель работы:** реализовать фиксированную навигацию и сделать содержательный блок «О нас» в соответствии с сеткой Bootstrap.

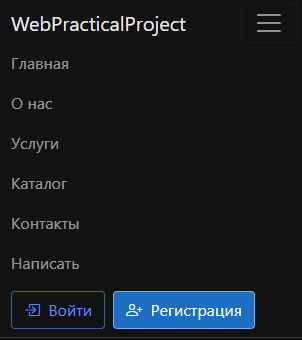
**Ход работы**

В «Views/Shared/\_Layout.cshtml» подключены Bootstrap и Bootstrap Icons, создана панель навигации «navbar navbar-expand-lg fixed-top» с идентификатором «mainNav». На теге body активирован ScrollSpy через «data-bs-spy="scroll" data-bs-target="#mainNav" data-bs-offset="80"», ссылки меню указывают на якоря главной страницы.

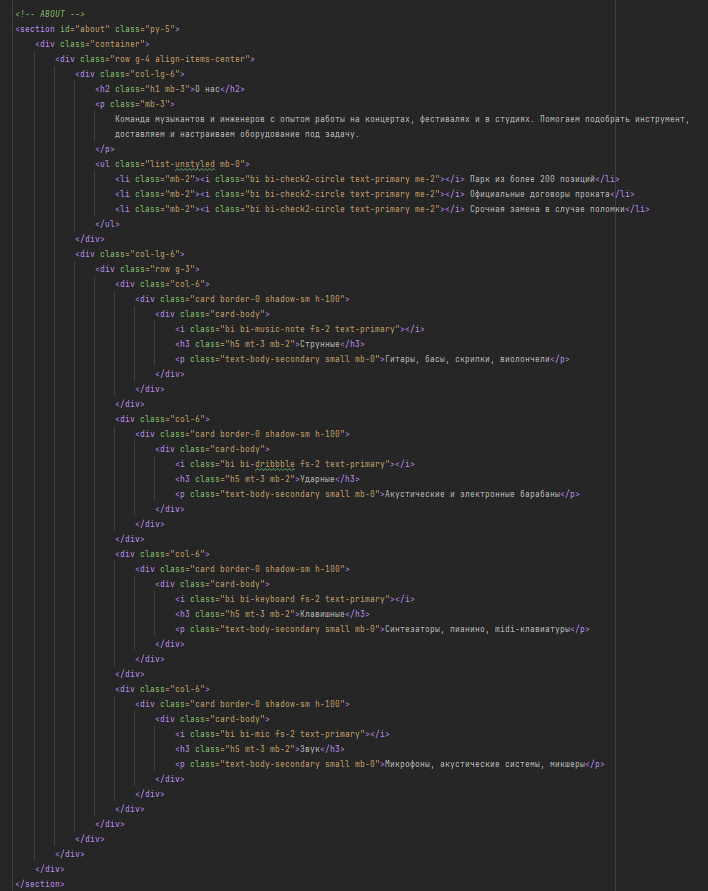


Рисунок 11 — фрагмент «\_Layout.cshtml» с разметкой navbar

Навигация фиксируется сверху, пункты меню подсвечиваются при прокрутке к соответствующим секциям. Кнопки справа инициируют модальные формы аутентификации без переключения страницы.

  
Рисунок 12 — шапка с раскрытием меню на мобильном устройстве.

Секция «О нас» размещена в «Index.cshtml» как двухколоночный блок: слева текст и перечень преимуществ, справа сетка из четырёх карточек. Комбинация «row g-4» и «col-lg-\*» обеспечивает адаптивное перестроение, карточки оформлены через «card border-0 shadow-sm», иконки — из Bootstrap Icons.

  
Рисунок 13 — фрагмент «Index.cshtml» секции «about»

На широких экранах блок остаётся двухколоночным, на узких перестраивается в одну колонку без горизонтальной прокрутки.

  
Рисунок 14 — браузер: отображение секции «О нас»

**Вывод:** реализована предсказуемая навигация по якорям и адаптивный информационный блок «О нас», пригодный к дальнейшей интеграции динамических данных.

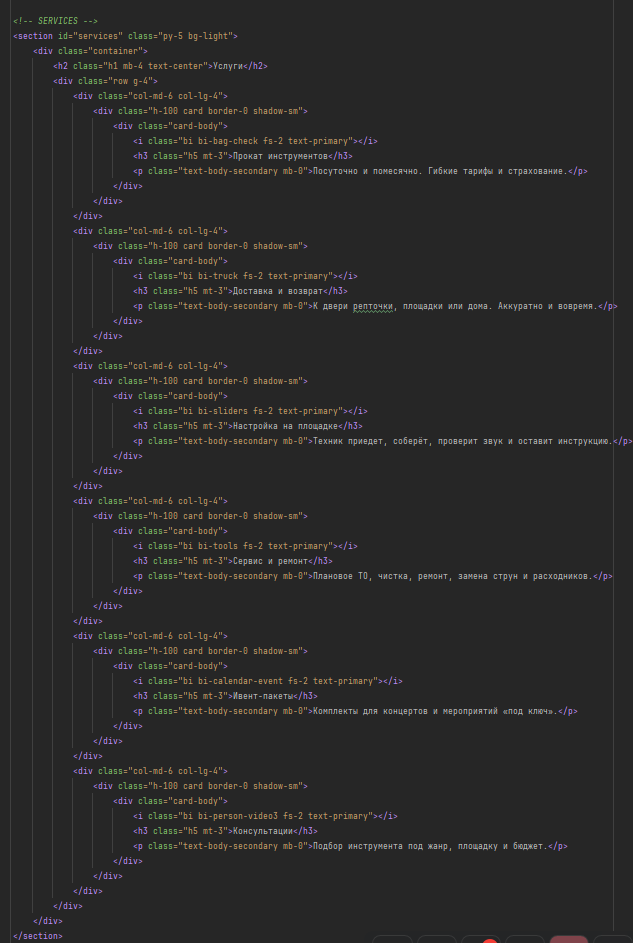
**5.11.2025**

**Тема:** Frontend разработка: секции «Услуги» и «Контакты». Оформление отчетной документации.

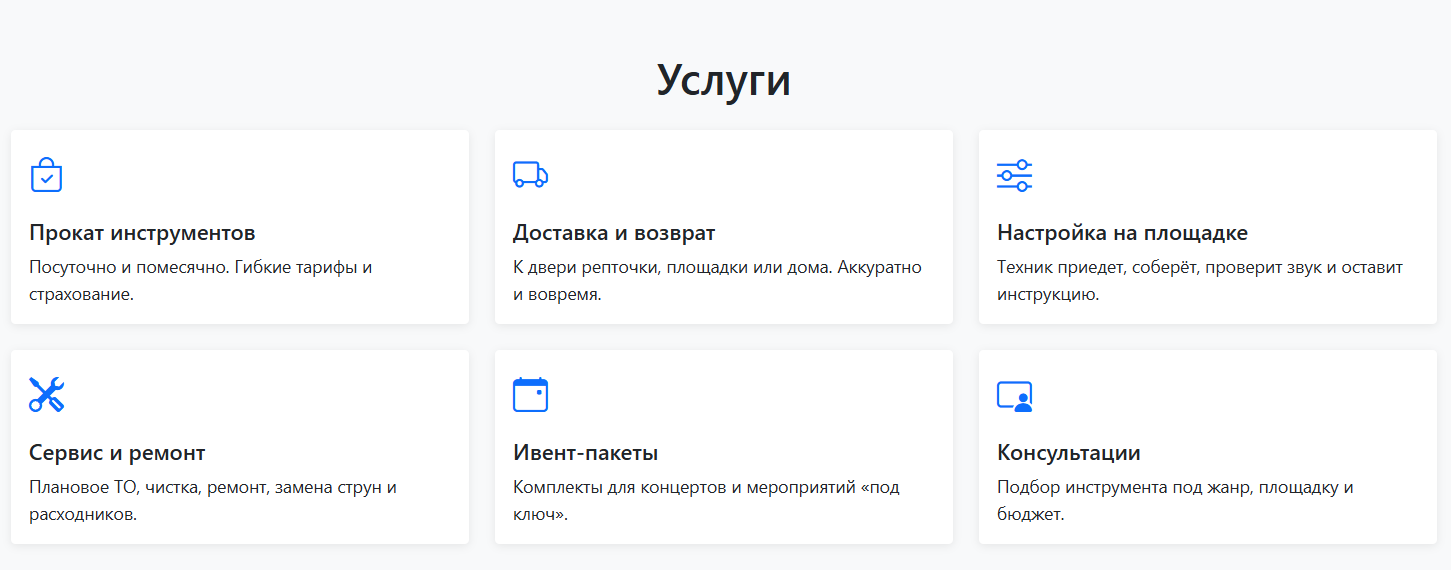
**Цель работы:** представить перечень сервисов в стандартизированном виде карточек и организовать контактный блок с кликабельными каналами связи и встроенной картой.

**Ход работы**

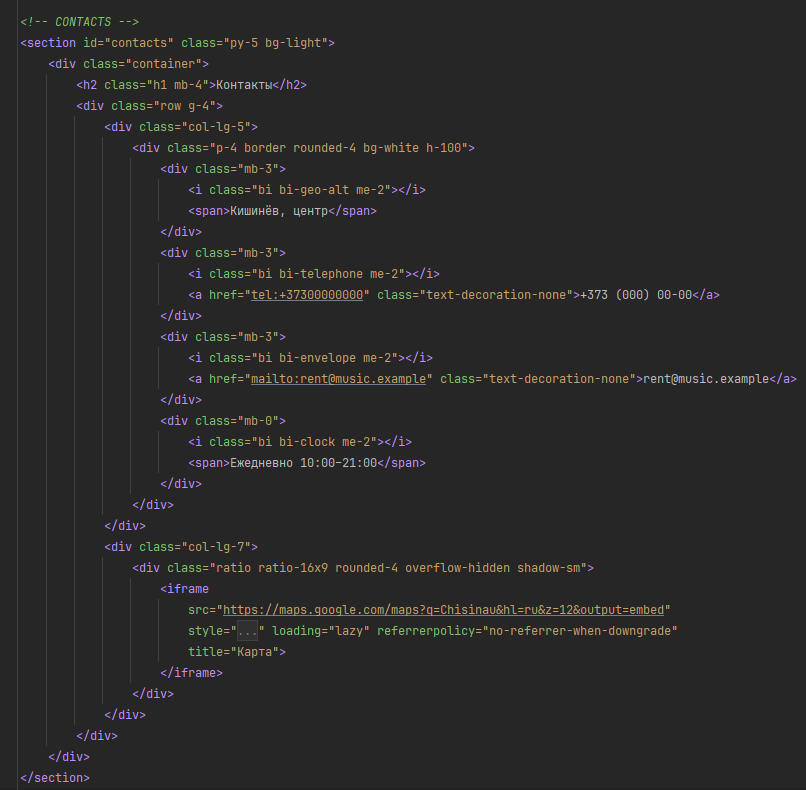
Секция «Услуги» сверстана набором карточек «card border-0 shadow-sm» с иконкой, заголовком и кратким описанием. Сетка «row g-4» перестраивается от трёх колонок на десктопе до одной на мобильных; размеры контролируются классами «col-md-6 col-lg-4».

  
Рисунок 15 — фрагмент «Index.cshtml» секции «services» с разметкой карточек услуг

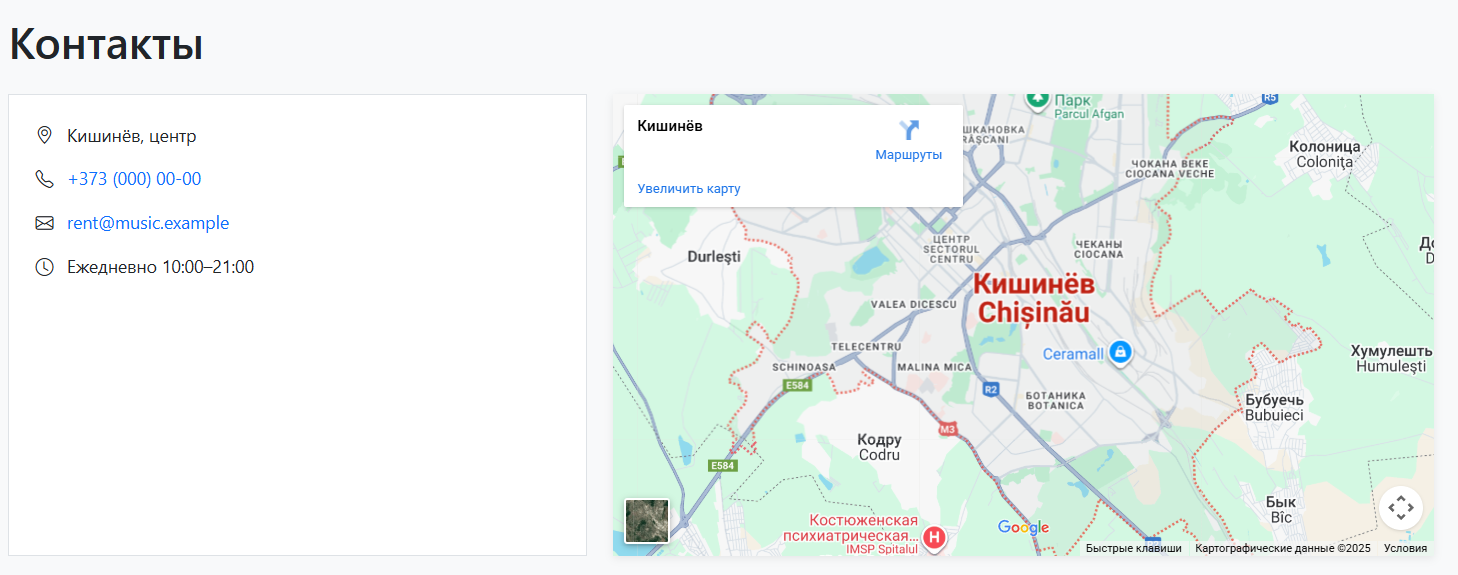
Карточки визуально однородны и читаемы, заголовки и иконки выровнены по вертикали, текст не приводит к скачкам высоты.

  
Рисунок 16 — сетка карточек «Услуги»

Секция «Контакты» содержит левую колонку с адресом, телефоном и почтой (ссылки формата «tel:» и «mailto:»), режимом работы; правая колонка — карта в iframe, обёрнутая в «ratio ratio-16x9» для стабильных пропорций и отсутствия сдвигов макета.

  
Рисунок 17 — фрагмент «Index.cshtml» секции «contacts» с колонкой контактных данных и картой

Ссылки позволяют инициировать звонок и письмо напрямую, карта центрирована на регионе и подстраивается под ширину контейнера.

  
Рисунок 18 — двухколоночный блок контактов с кликабельными ссылками и встроенной картой

**Вывод:** реализован перечень услуг и контактный блок обеспечивающий быстрый доступ к каналам связи и стабильное отображение карты.

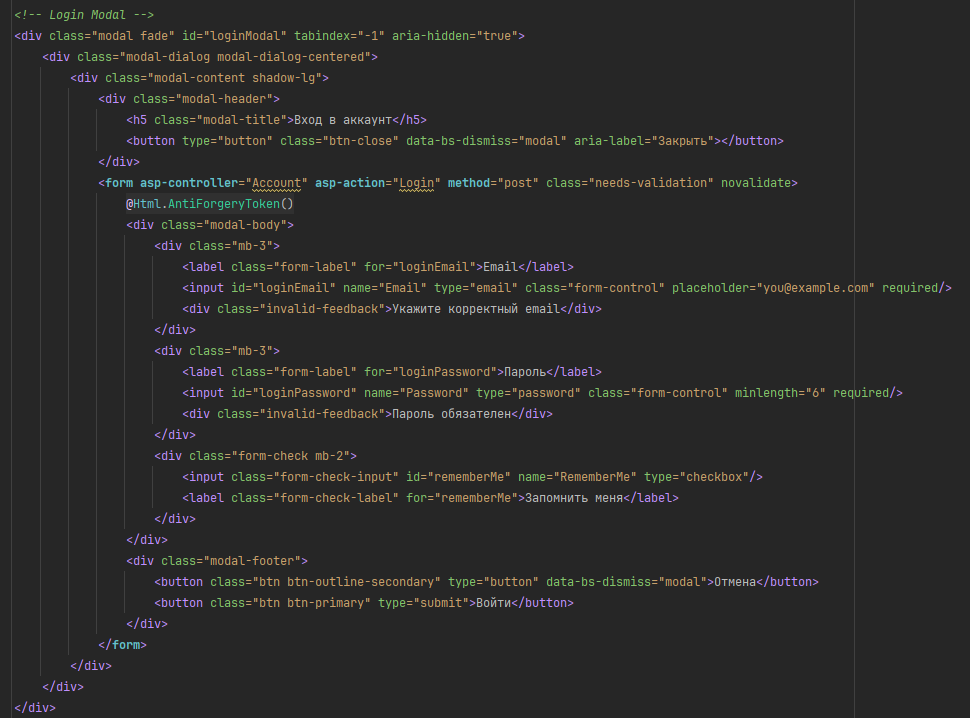
**6.11.2025**

**Тема:** Frontend разработка: формы входа и регистрации. Оформление отчетной документации.

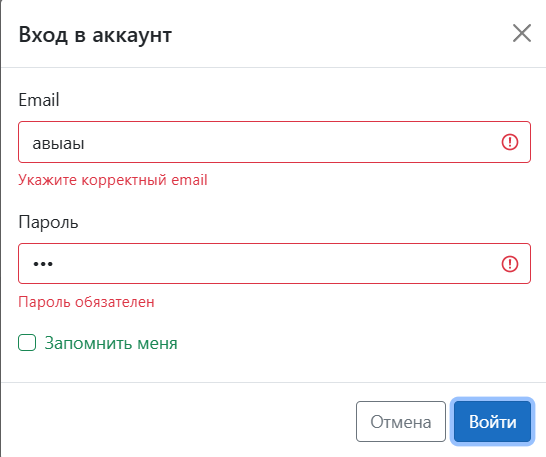
**Цель работы:** внедрить модальные формы аутентификации с клиентской валидацией и CSRF-защитой, не нарушая сценарий взаимодействия с главной страницей.

**Ход работы**

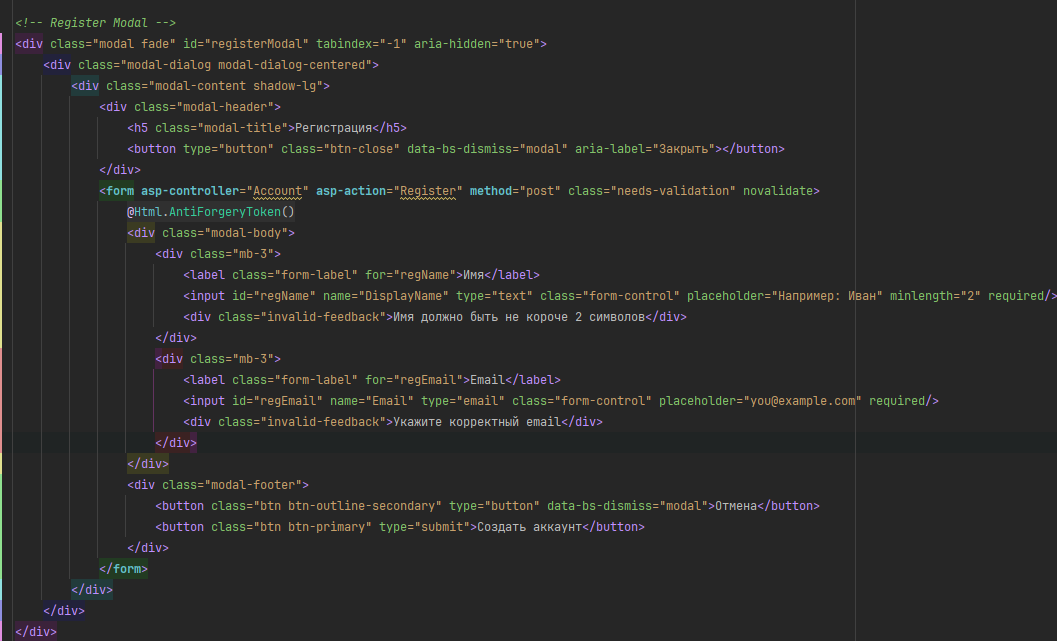
Форма входа размещена в «Layout.cshtml» в модальном окне Bootstrap. Поля «Email» и «Password», флаг «RememberMe», отправка «POST» на «Account/Login». Присутствует «@Html.AntiForgeryToken()», атрибуты «required» и «minlength». Общий скрипт активирует схему «needs-validation»/«was-validated», предотвращая отправку при невалидных значениях.

  
Рисунок 19 — фрагмент «\_Layout.cshtml» модального окна входа

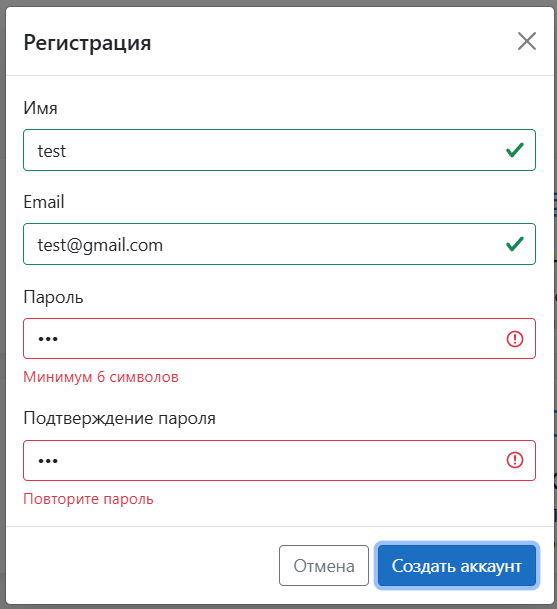
При открытии модального окна проверка выполняется нативными средствами браузера, сообщения выводятся через «invalid-feedback».

  
Рисунок 20 — модальное окно входа

Форма регистрации реализована в отдельной модальном окне с полями «DisplayName», «Email», «Password», «ConfirmPassword». Отправка «POST» на «Account/Register», присутствует «@Html.AntiForgeryToken()», валидация аналогична форме входа.

  
Рисунок 21 — фрагмент «\_Layout.cshtml» модального окна регистрации

Поля визуально унифицированы за счёт «form-control», кнопки используют иконки без потери доступности за счёт текстовых подписей.

  
Рисунок 22 — модальное окно регистрации, проверка длины и подтверждения пароля

**Вывод:** формы обеспечивают аутентификацию в модальном режиме с базовыми требованиями к безопасности и единым поведением валидации.

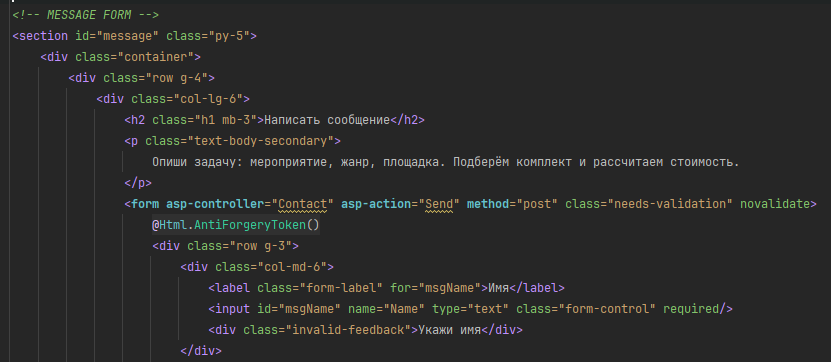
**7.11.2025**

**Тема:** Frontend разработка: секция «Написать сообщение» и подвал сайта. Оформление отчетной документации.

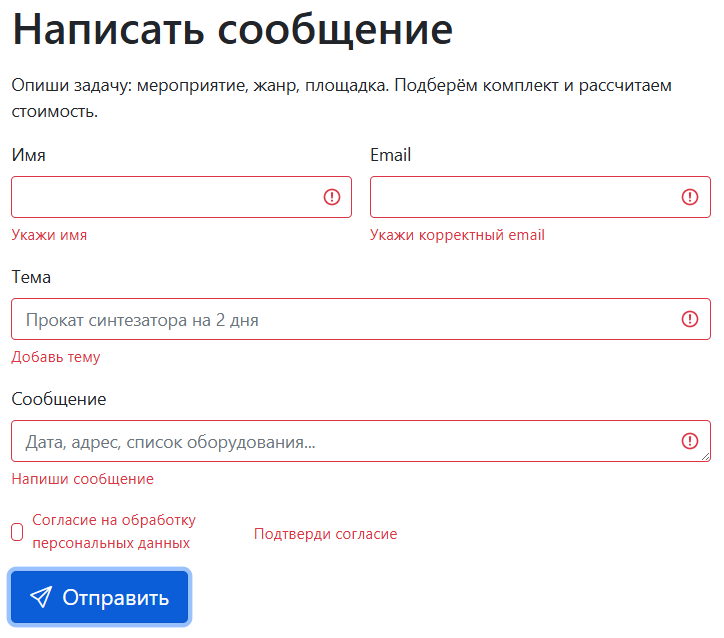
**Цель работы:** реализовать форму обратной связи с обязательными полями и согласием на обработку данных и оформить завершающий блок страницы с необходимыми ссылками.

**Ход работы**

Секция «Написать сообщение» находится в левой колонке «Index.cshtml». Поля «Name», «Email», «Subject», «Message», обязательный чекбокс «Agree». Отправка «POST» на «Contact/Send», присутствует «@Html.AntiForgeryToken()». Клиентская проверка выполнена через атрибуты и общий скрипт валидации.

  
Рисунок 23 — фрагмент «Index.cshtml» секции «message» с полями формы

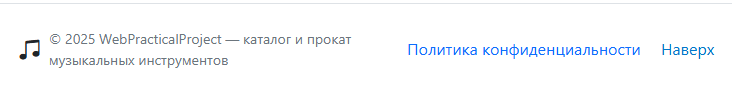
При заполнении форма показывает сообщения об ошибках для каждого поля, кнопка отправки помечена как основное действие.

  
Рисунок 24 — форма обратной связи с отображением валидационных подсказок

Подвал расположен в «\_Layout.cshtml». Разметка включает левый блок с идентификатором проекта и подписью, правый — ссылку на политику конфиденциальности через «asp-controller/asp-action» и якорь «Наверх». Блок отделён верхней границей и симметричными отступами.

  
Рисунок 25 — фрагмент «\_Layout.cshtml» подвала с ссылками и структурой колонок

Подвал сохраняет единый ритм отступов с основной частью страницы и корректно адаптируется на мобильных (перестройка колонок).

  
Рисунок 26 — отображение подвала

**Вывод:** форма обеспечивает сбор структурированных обращений с обязательным согласием, подвал завершает страницу и содержит служебные ссылки в соответствии с практикой веб-оформления.

**8.11.2025**

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

**Цель работы:** внедрить отправку запросов fetch и обработку ответов JSON/FormData без перезагрузки страницы.

**Ход работы**

Реализация выполнена на уровне представления, клиентских скриптов и контроллеров с возвратом JSON. В «Views/Shared/\_Layout.cshtml» подключены клиентские модули «~/js/fetch-helpers.js» и «~/js/app-interactions.js» перед «~/js/site.js», чтобы обеспечить доступность хелперов для всех страниц и модальных окон. Представление содержит формы входа и регистрации, а также форму «Написать сообщение» в «Views/Home/Index.cshtml», где отправка перехватывается на клиенте и уходит в контроллеры без перезагрузки страницы.

Клиентская часть построена вокруг двух узлов:

1. «wwwroot/js/fetch-helpers.js» — универсальные хелперы: получение антифрод-токена из скрытого поля формы, единая функция fetch с обработкой ошибок и автоматическим выбором формата тела запроса (FormData для форм с файлами и для «классической» MVC-схемы, JSON — при явной сериализации). Здесь же инициализируется простая система уведомлений на основе Bootstrap-alert для фиксации результата вызовов.
2. «wwwroot/js/app-interactions.js» — прикладная логика страницы: навешивание ajax-обработчика на форму «Написать сообщение» в «#message form» и на формы модальных окон авторизации и регистрации («#loginModal form», «#registerModal form»), а также динамическая загрузка блока каталога «Популярно сейчас» с GET-запросом к «/Catalog/Popular». При успешном ответе формы очищаются и снимается класс валидации; при ошибке показывается уведомление с текстом из JSON.

Связь представлений и действий контроллеров организована следующим образом. «ContactController.Send (POST)» принимает данные формы «Написать сообщение» и возвращает JSON с идентификатором созданной заявки; валидация обязательных полей выполняется на сервере независимо от фронтенда. «AccountController.Login (POST)» и «AccountController.Register (POST)» обрабатывают соответствующие модальные формы и возвращают компактные JSON-ответы, достаточные для обновления клиентского состояния. «CatalogController.Popular (GET)» отдаёт список карточек каталога; клиентский код в «app-interactions.js» формирует HTML-разметку карточек и вставляет её в «#catalog .row».

На уровне архитектуры контроллеры MVC остаются тонкими и подготовлены для подключения сервисного слоя. Предполагается, что в следующем шаге «Home/Contact/Account/Catalog» будут делегировать бизнес-логику в «Service», где далее обращения к данным будут маршрутизироваться в «DAL», а контракты/сущности определены в «Domain». В текущей теме сервисы и хранилище не задействованы; возвраты JSON имитируют целевые структуры ответов, чтобы зафиксировать стабильный фронтенд-контракт.

  
Рисунок 27 — подключение клиентских модулей в макете

Подключение скриптов в макете обеспечивает единообразную инициализацию ajax-поведения форм и динамических секций на всех страницах, включая модальные окна, которые рендерятся внизу макета и доступны после загрузки DOM.

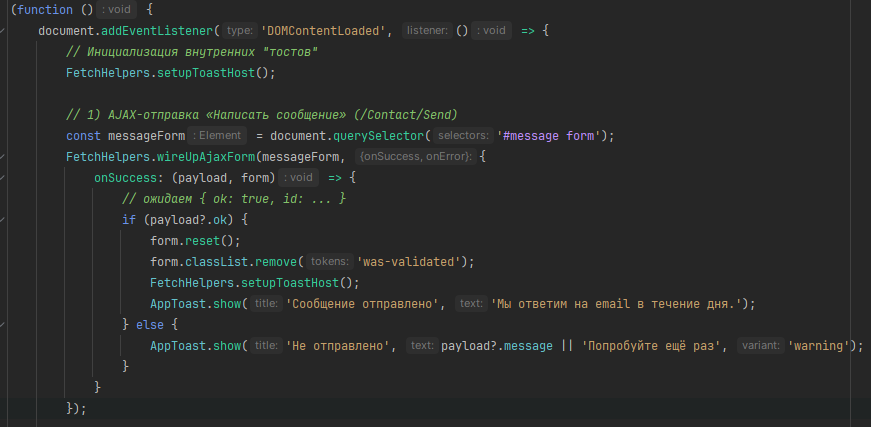
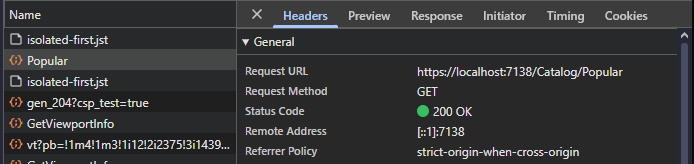
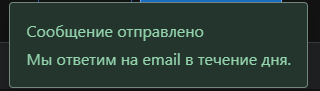


Рисунок 28 — навешивание ajax-обработчиков на формы

Динамическая секция каталога «Популярно сейчас» получает содержимое через «CatalogController.Popular (GET)». Клиентский код после получения JSON строит карточки на стороне браузера и вставляет их внутрь «#catalog .row», заменяя заглушку-лоадер.

  
Рисунок 29 — загрузка каталога через fetch без перезагрузки страницы

Для формы «Написать сообщение» используется отправка как FormData, что совместимо с «ValidateAntiForgeryToken», поскольку токен остаётся в скрытом поле формы и корректно обрабатывается сервером. Серверная проверка обязательных полей выполняется в действии контроллера, ответ возвращается в JSON-формате и обрабатывается на клиенте с показом уведомления.

  
Рисунок 30 — отправка формы через fetch и обработка JSON-ответа

На стороне контроллеров реализованы компактные действия: «ContactController.Send (POST)» валидирует «Email» и «Message» и возвращает JSON с признаком «ok» и идентификатором; «AccountController.Login (POST)» и «AccountController.Register (POST)» формируют минимальные JSON-ответы для обновления клиентского состояния; «CatalogController.Popular (GET)» возвращает коллекцию DTO с заголовком, ссылкой на изображение и минимальной ценой.

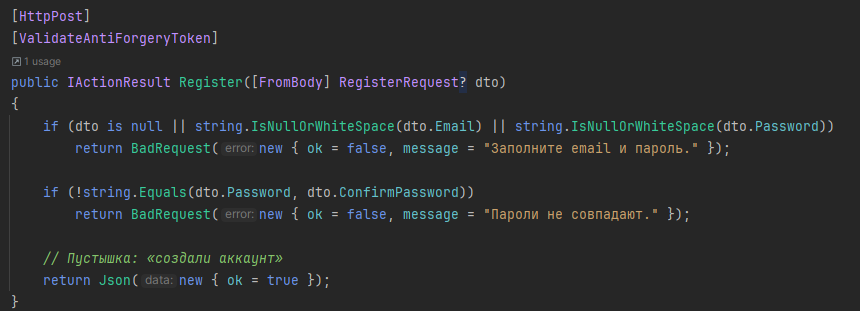


Рисунок 31 — действия контроллеров с JSON-ответами

**Вывод:** реализована отправка fetch-запросов и обработка JSON-ответов; формы и динамический каталог обновляются без перезагрузки страницы, а контроллеры возвращают согласованные структуры данных для дальнейшего подключения сервисного слоя.

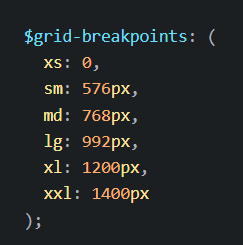
**10.11.2025**

**Тема:** Создание адаптивного интерфейса с использованием медиазапросов CSS. Оформление отчетной документации.

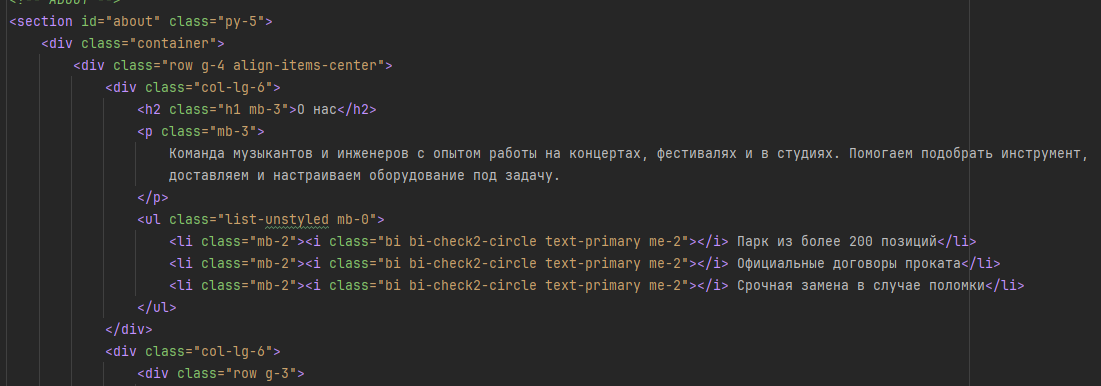
**Цель работы:** внедрить адаптивное поведение разметки с использованием медиазапросов Bootstrap и его responsive-утилит без добавления собственных @media-правил.

**Ход работы**

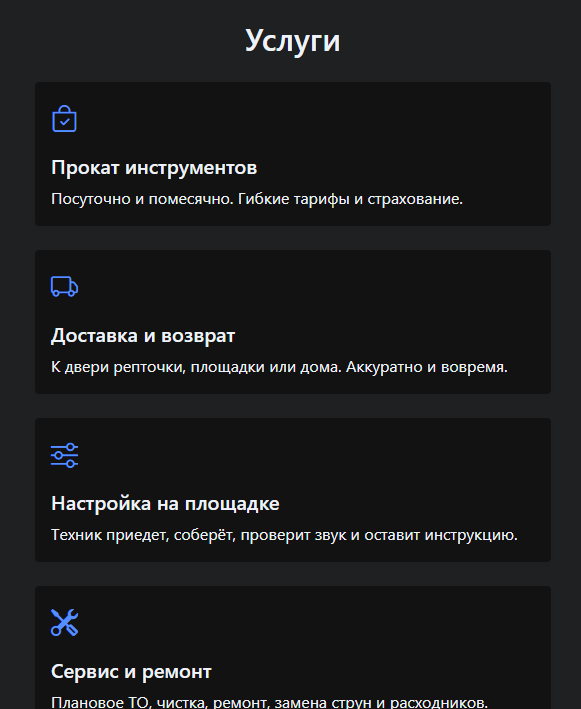
В «Views/Shared/\_Layout.cshtml» активирована адаптивность через метатег viewport и применены классы Bootstrap, которые переключают сетку и отступы по брейкпоинтам. Для шапки использован «navbar navbar-expand-lg fixed-top», что даёт свертывание меню в диапазонах меньше «lg» и фиксирование навигации по верхнему краю окна. В секции HERO применены «row align-items-center g-4» и колонки «col-lg-7» и «col-lg-5»: при ширине меньше «lg» колонки складываются в один столбец, на «lg» и выше — выстраиваются в два. Отступы «py-5 py-lg-6» увеличиваются с «lg». Блок изображения использует «ratio ratio-4x3» для сохранения пропорций на любых ширинах.

  
Рисунок 32 — Брейкпоинты Bootstrap

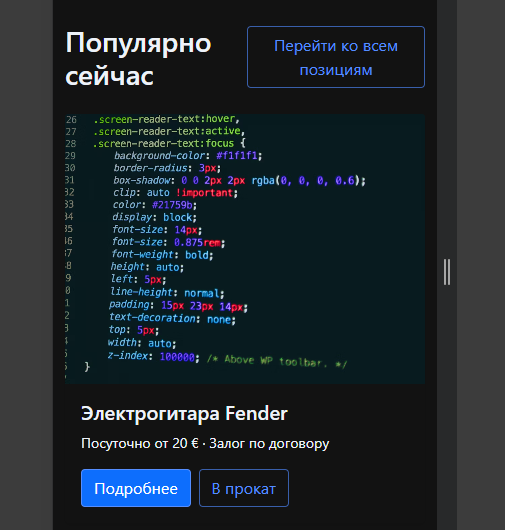
В секции «О нас» структура «row g-4 align-items-center» с колонками «col-lg-6» обеспечивает один столбец на мобильных и два столбца начиная с «lg». Карточная сетка внутри правой колонки построена на «row g-3» и «col-6», поэтому на телефонах выводится две карточки в ряд, а на очень узких экранах — автоматический перенос с сохранением двухколоночной сетки.

  
Рисунок 33 — Разметка секции «О нас» с брейкпоинтом «lg»

В секции «Услуги» для карточек заданы «col-md-6 col-lg-4». Это даёт 1 карточку в ряд на «xs/sm», 2 в ряд на «md», 3 в ряд на «lg» и выше. Отступы секции заданы «py-5», светлый фон «bg-light» стабильно работает на всех диапазонах. Такой подбор классов исключает необходимость собственных медиазапросов: перестройка сетки целиком управляется Bootstrap.

  
Рисунок 34 — Поведение сетки «Услуги» на «md»

В «Catalog preview» использованы «col-sm-6 col-lg-4»: одна карточка на «xs», две — на «sm/md», три — на «lg+». Внутри карточек «ratio ratio-4x3» удерживает стабильную высоту обложки, а «object-fit-cover» (на классе изображения) препятствует деформации при резайзе. Заголовок блока и кнопка «Перейти ко всем позициям» помещены в «d-flex justify-content-between align-items-center», оставаясь в одной строке на «md+» и переносясь на две строки в «sm-» при нехватке ширины.

  
Рисунок 35 — Поведение секции каталога на «xs»

**Вывод:** реализовано адаптивное поведение разделов страницы средствами Bootstrap: сетка и утилиты брейкпоинтов применены в «Views/Shared/\_Layout.cshtml» для HERO, «О нас», «Услуги», «Каталог», «Контакты», навигации и модальных окон; собственные @media-правила не требуются.

**11.11.2025**

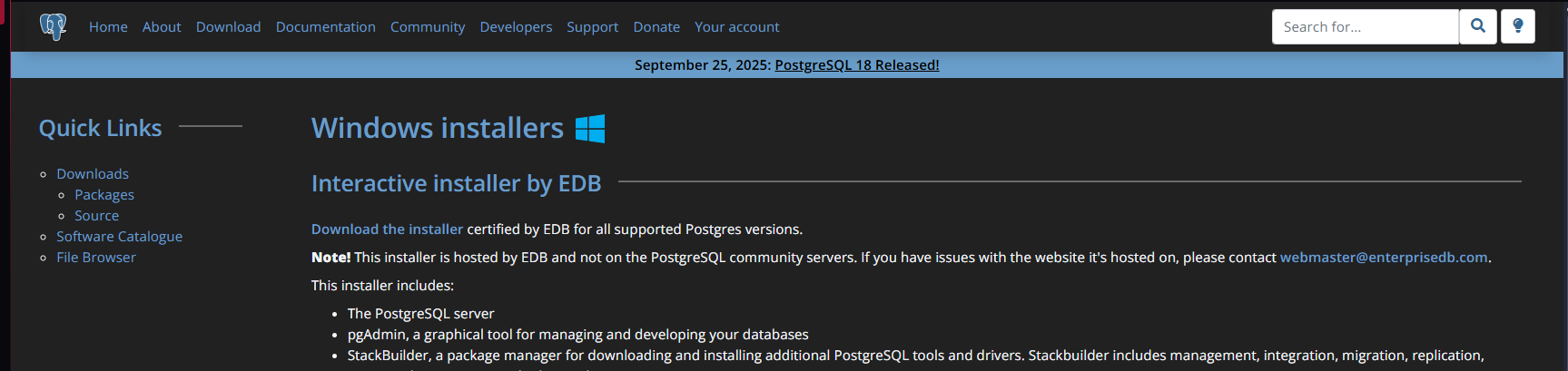
**Тема:** Знакомство с PostgreSQL. Оформление отчетной документации.

**Цель работы:** подготовить рабочую установку PostgreSQL и запустить pgAdmin до состояния открытой панели управления с подключением к локальному серверу.

**Ход работы**

Поэтапная установка и запуск PostgreSQL:

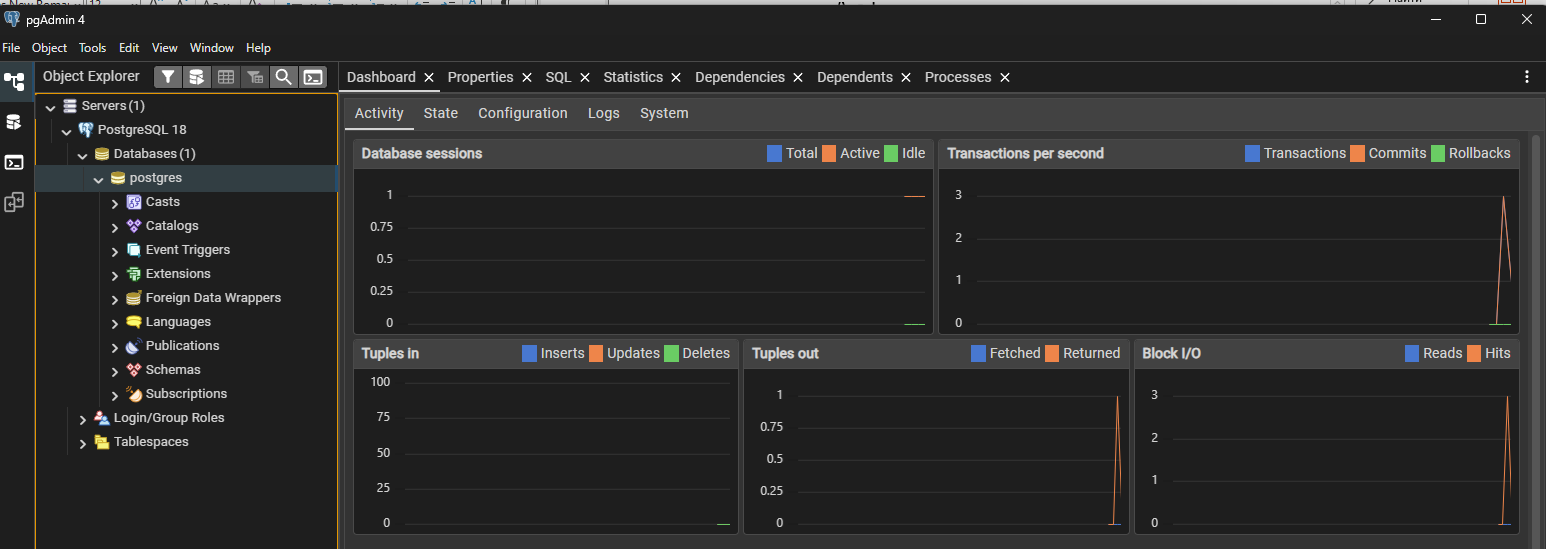
1. Выполнена загрузка дистрибутива с официального сайта <https://www.postgresql.org/download/>. Для Windows выбран установщик от EDB («Download the installer»), версия PostgreSQL актуальная на момент установки и разрядность, соответствующая ОС.
2. Запущен установщик, в мастере отмечены компоненты: «PostgreSQL Server», «pgAdmin 4», опционально «Stack Builder». Путь установки оставлен по умолчанию.
3. На шаге настройки кластера задан пароль суперпользователя «postgres», выбран порт «5432» и локаль по умолчанию. Мастер завершён, служба PostgreSQL запущена.
4. Открыт «pgAdmin 4» из меню системы. При первом запуске создан мастер-пароль pgAdmin (отличается от пароля «postgres»).
5. В левой панели «Browser» создано подключение к локальному серверу: «Add New Server…» далее вкладка «General» — имя произвольное, вкладка «Connection» — «Host name/address: localhost», «Port: 5432», «Maintenance database: postgres», «Username: postgres», «Password: <заданный ранее>» с сохранением пароля.
6. После сохранения подключение установлено, в дереве «Databases» отображается база «postgres» и системные схемы; панель pgAdmin готова к дальнейшей работе.

  
Рисунок 36 — Страница загрузки PostgreSQL

Официальный установщик содержит сервер и pgAdmin; при выборе версии важно сопоставить разрядность с разрядностью ОС.

  
Рисунок 37 — Выбор компонентов в установщике

После установки всех компонентов задается мастер-пароль. Далее мы добавляем локальное подключение к localhost. После сохранения дерево объектов отображает базу «postgres» и служебные разделы; панель готова к операциям с базами и схемами.

  
Рисунок 38 — Подключённый локальный сервер в pgAdmin

**Вывод:** выполнена установка PostgreSQL и pgAdmin с официального дистрибутива; создано и проверено подключение к локальному серверу в pgAdmin, панель управления открыта и готова к дальнейшей конфигурации.

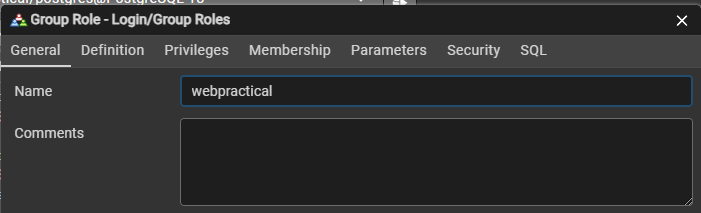
**12.11.2025**

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

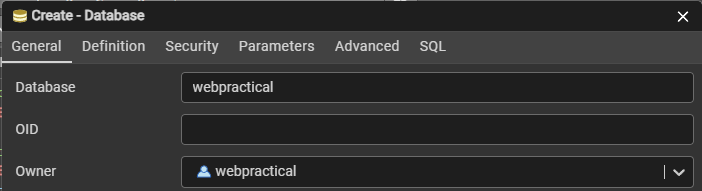
**Цель работы:** создать пользователя СУБД и базу данных PostgreSQL для проекта; инициализировать схему приложения и настройки поиска объектов, подготовив БД к выполнению скрипта таблиц.

**Ход работы**

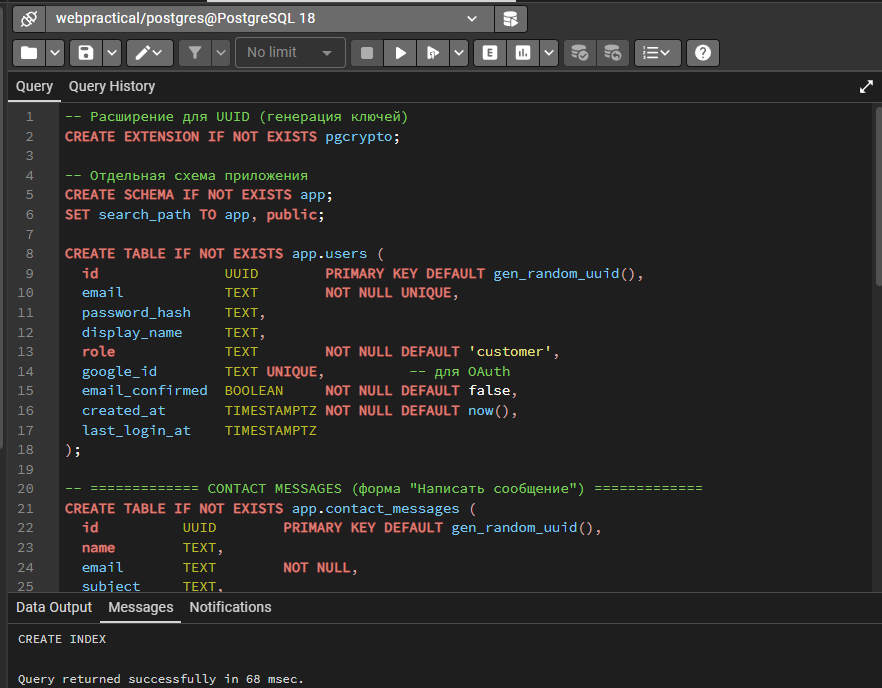
В pgAdmin 4 выполнено подключение к локальному серверу PostgreSQL. В «Login/Group Roles» создана роль «webpractical» с правом входа и паролем. Во вкладке привилегий дополнительные полномочия не выдавались, назначение роли — владелец прикладной базы.

  
Рисунок 39 — Создание роли webpractical

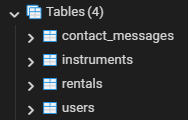
В «Databases» создана база «webpractical» с владельцем «webpractical». Параметры оставлены по умолчанию: кодировка UTF-8, шаблон template1. После сохранения проверено подключение к новой БД через панель свойств.

  
Рисунок 40 — Создание базы webpractical

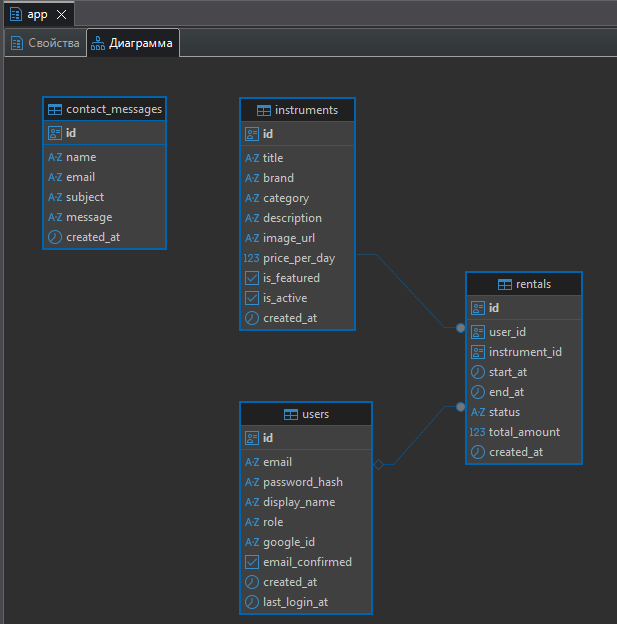
В Query Tool на базе «webpractical» выполнена инициализация: подключено расширение для генерации UUID, создана прикладная схема «app», установлен search\_path на «app, public». Настройка search\_path сохранена на уровне базы, чтобы обращаться к объектам без префикса схемы.

  
Рисунок 41 — Инициализация расширения, схемы и search\_path

В pgAdmin 4 в Query Tool выполнен скрипт с таблицами «users», «contact\_messages», «instruments», «rentals» и индексами. В «Object Explorer» проверено появление объектов в схеме «app»; связи внешних ключей отображаются в свойствах таблицы «rentals».

  
Рисунок 42 — Созданные таблицы и внешние ключи

В DBeaver построена диаграмма по схеме «app» для визуальной проверки связей: «rentals.user\_id → users.id» с политикой удаления «SET NULL» и «rentals.instrument\_id → instruments.id» с «CASCADE».

  
Рисунок 43 — Визуальная проверка связей в DBeaver

**Вывод:** создан пользователь СУБД и база «webpractical», инициализирована схема «app» и поисковый путь, выполнен скрипт таблиц и индексов; структура подтверждена в pgAdmin 4 и на ER-диаграмме в DBeaver.

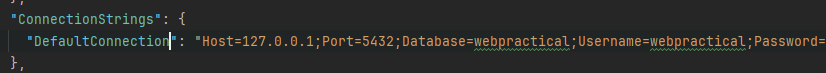
**13.11.2025**

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

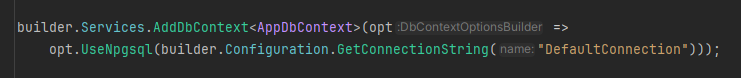
**Цель работы:** настроить подключение к PostgreSQL через «appsettings.json» и связать Entity Framework Core с существующей схемой БД, определив сущности и их атрибуты для точного соответствия таблицам.

**Ход работы**

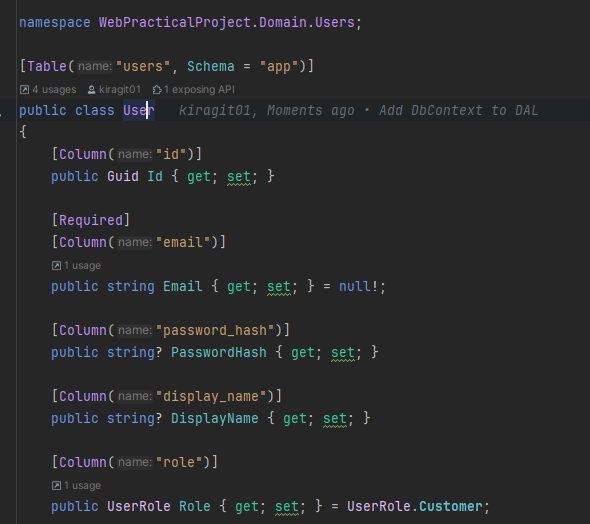
В «appsettings.json» добавлена строка подключения «ConnectionStrings:DefaultConnection». В «Program.cs» зарегистрирован «AppDbContext» с провайдером Npgsql по ключу «DefaultConnection». Подключение используется контроллерами напрямую через DAL.

  
Рисунок 44 — Строка подключения в appsettings.json

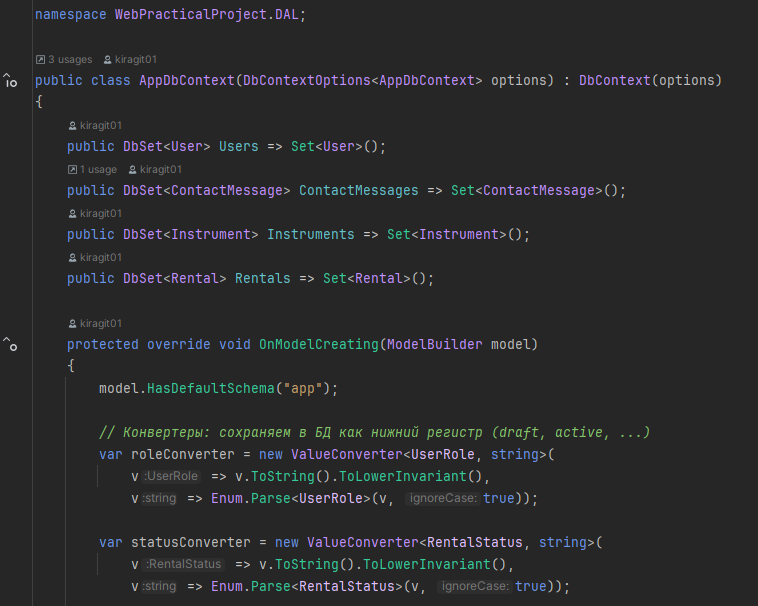
В «Program.cs» явно указан тип «AppDbContext» и вызов DefaultConnection. Регистрация добавлена до маршрутизации. Дополнительные параметры провайдера не задавались.

  
Рисунок 45 — Регистрация AppDbContext

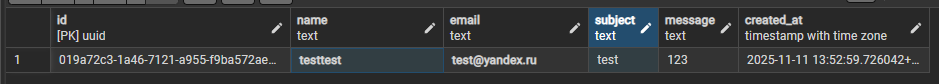
Сущности описаны атрибутами для явного сопоставления со схемой «app» и snake\_case-колонками. Для каждой модели добавлены: [Table] и [Column]. Это исключает несоответствие регистров имён («Id» ↔ «id», «CreatedAt» ↔ «created\_at») и ошибки вида «столбец "Id" не существует».

  
Рисунок 46 — Атрибуты сопоставления сущности User

В «DAL/AppDbContext.cs» в «OnModelCreating» заданы конвертеры enum→text: «UserRole» и «RentalStatus» сериализуются в нижний регистр для колонок «role» и «status». Там же оформлены внешние ключи «rentals.user\_id → users.id (SetNull)» и «rentals.instrument\_id → instruments.id (Cascade)» и проверка «ck\_rentals\_dates: end\_at > start\_at». Индексы повторяют добавленные в БД (категория/бренд/стоимость/признак «is\_featured»).

  
Рисунок 47 — Наши таблицы с объектами

Работоспособность проверена: при отправке формы «Написать сообщение» запись создаётся в «app.contact\_messages» с колонками «id, email, message, created\_at…». В «pgAdmin 4» таблица открыта в «View/Edit Data»; строка содержит значения из POST-запроса.

  
Рисунок 48 — Фиксация записи в таблице контактных сообщений

**Вывод:** настроено подключение к PostgreSQL через «appsettings.json» и «Program.cs»; сущности аннотированы для точного соответствия таблицам в схеме «app»; в контексте заданы конвертеры enum’ов и ограничения. DAL готов к использованию в контроллерах для операций чтения и записи.

**14.11.2025**

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

**Цель работы:** реализовать базовые операции CRUD для сущностей каталога, аренды, сообщений и пользователей.

**Ход работы**

В «Domain» определены сущности и перечисления: «Domain/Catalog/Instrument.cs», «Domain/Rentals/Rental.cs», «Domain/Contacts/ContactMessage.cs», «Domain/Users/User.cs», «Domain/Common/{UserRole,RentalStatus}.cs». В «DAL/AppDbContext.cs» настроены таблицы и индексы; для «rentals» добавлены ограничения целостности по внешним ключам и проверка периода. CRUD вынесен на уровень репозиториев и сервисов:



Рисунок 49 — Интерфейсы для использования репозитория

Контроллеры обращаются только к сервисам: «Controllers/CatalogController.cs» (каталог и детали), «Controllers/RentalsController.cs» (создание аренды и список «Мои аренды»), «Controllers/ContactController.cs» (создание сообщения), «Controllers/AdminController.cs» (просмотр/правка пользователей и сообщений). Представления: «Views/Catalog/{Index, Details}.cshtml», «Views/Rentals/{Create, My}.cshtml», «Views/Admin/{Users, Messages}.cshtml».

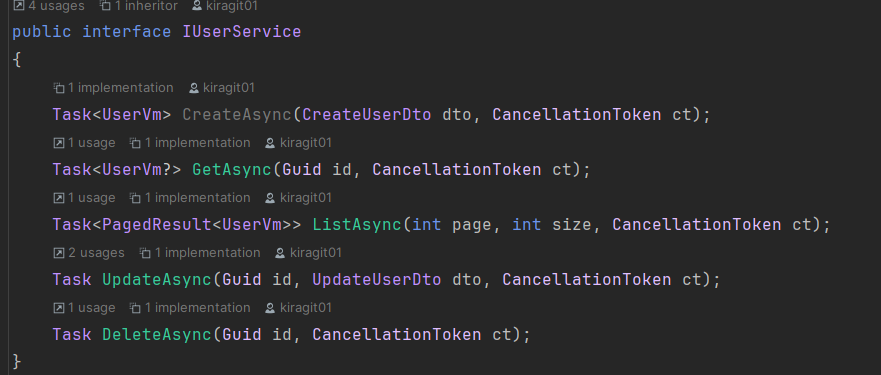


Рисунок 50 — CRUD в интерфейсе сервисов

В «Views/Catalog/Index.cshtml» выведена сетка карточек с пагинацией; «Views/Rentals/Create.cshtml» оформляет заказ аренды, «Views/Rentals/My.cshtml» показывает собственные заказы с форматированием дат, сумм и бейджами статуса. «AdminController» предоставляет страницы управления пользователями и входящими сообщениями.

  
Рисунок 51 — Отрисовка CRUD для инструментов на сайте

**Вывод:** реализованы сервисы и репозитории CRUD, подключены контроллеры и представления; операции создания, чтения, обновления и удаления доступны для всех сущностей.

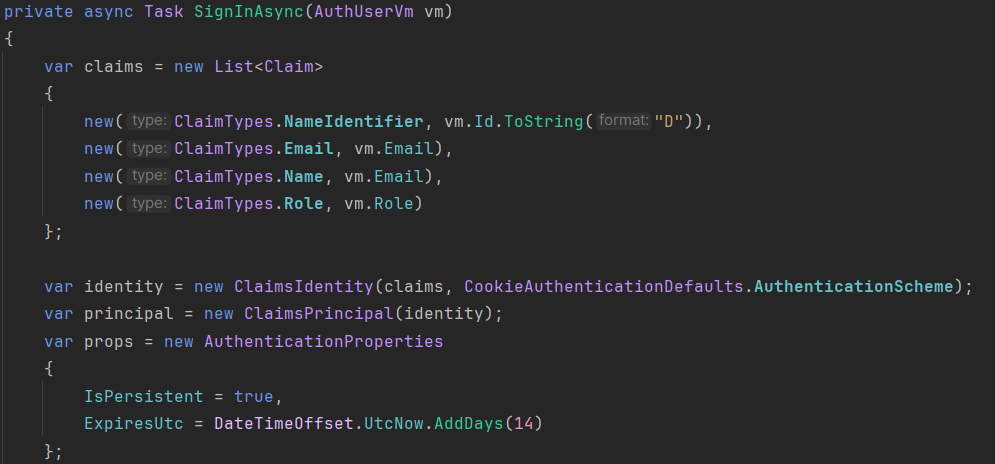
**15.11.2025**

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

**Цель работы:** внедрить сервисы регистрации и входа, аутентификацию через cookie и возврат сведений о пользователе.

**Ход работы**

Добавлен «Service/Services/AuthService.cs» с методами регистрации и входа: проверка уникальности email через «IUserRepository», хэширование пароля при создании, сверка хэша при входе, возврат «AuthUserVm». В «Controllers/AccountController.cs» реализованы «Register (POST)», «Login (POST)», «Logout (POST)», «Me (GET)»; при успешной аутентификации создаётся cookie. Конфигурация схемы cookie активирована в «Program.cs».

  
Рисунок 52 — Создание куки и возврат клеймов

Проверка выполнена через модальные формы во «Views/Shared/\_Layout.cshtml»: «Вход» отправляет данные на «AccountController.Login (POST)», при успехе выполняется вход и перезагрузка страницы. «Регистрация» вызывает «AccountController.Register (POST)» с авто-входом.

[ФОТО] Браузер: модальное окно входа с успешным ответом и закрытием  
Рисунок 4 — Выполнение входа через UI

**Вывод:** реализованы сервис и контроллеры для регистрации/входа, настроена куки-аутентификация и базовый эндпоинт «Me» для получения текущего пользователя.

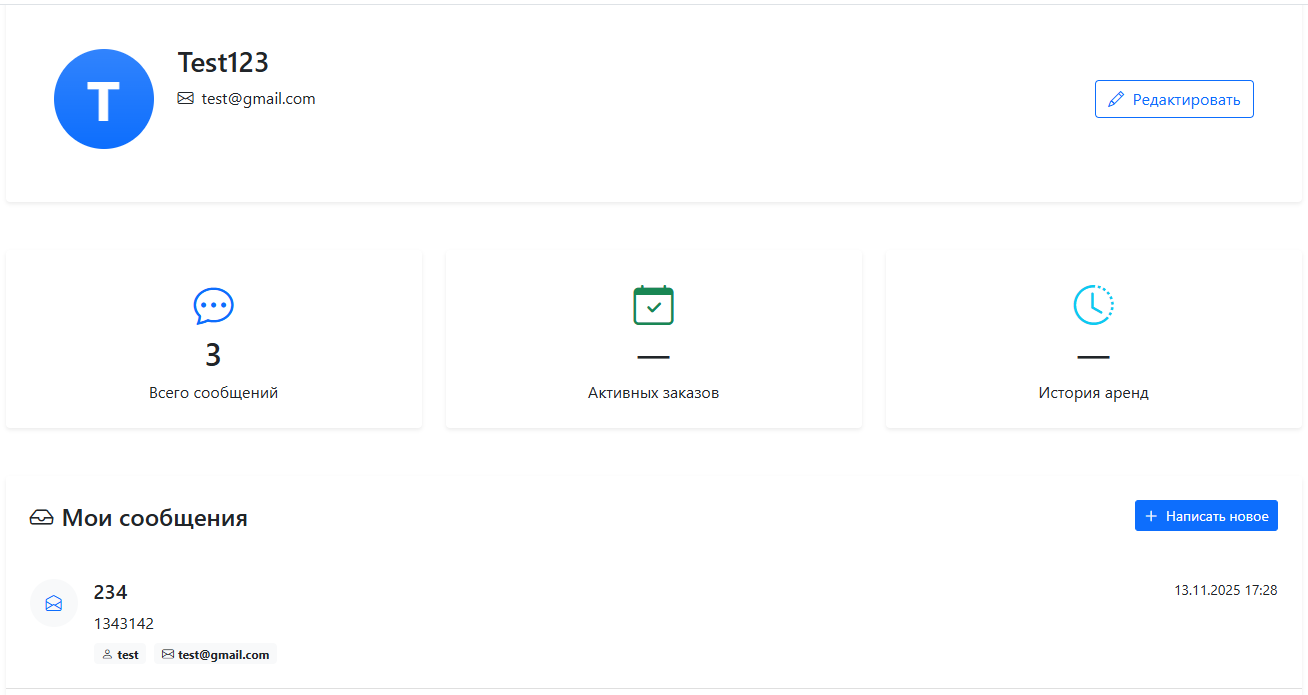
**17.11.2025**

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

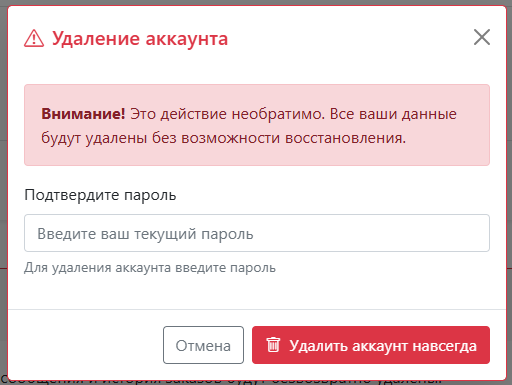
**Цель работы:** расширить сервисы аутентификации дополнительными проверками и операциями профиля.

**Ход работы**

В «AuthService» добавлены: проверка дубликата email с выдачей кода ошибки; валидация пароля при входе; операция удаления собственной учётной записи «DeleteSelfAsync» с повторной проверкой пароля. В «Controllers/AccountController.cs» добавлены действия профиля: «Profile (GET)» — сбор данных пользователя и его сообщений; «UpdateProfile (POST)» — изменение имени; «DeleteAccount (POST)» — удаление аккаунта и выход. В «Service/Services/UserService.cs» реализована частичная правка отображаемого имени; в «Service/Services/ContactService.cs» — выборка сообщений по email «ListMineAsync».

  
Рисунок 53 — Раздел профиля

После удаления учётной записи выполняется «SignOut» и редирект на «Home/Index». Для «RememberMe» в «SignInAsync» установлены «IsPersistent» и «ExpiresUtc»; этим же методом можно обновлять куку при изменении роли/почты.

  
Рисунок 54 — Удаление аккаунта и завершение сессии

**Вывод:** сервисы доработаны проверками, операциями профиля и удалением учётной записи; контроллеры подключены и выполняют выход/обновление cookie.

**18.11.2025**

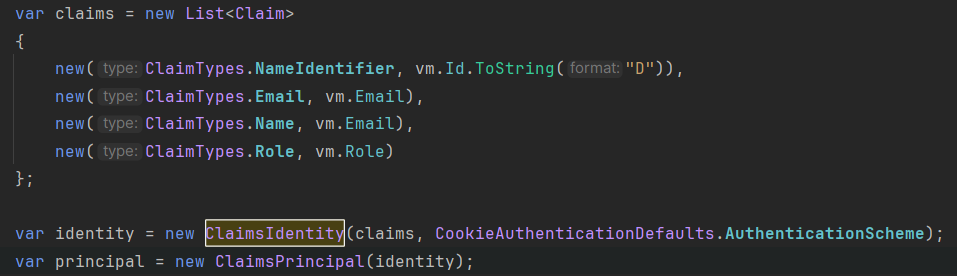
**Тема:** Подготовка проекта к использованию ClaimsIdentity. Оформление отчетной документации.

**Цель работы:** подготовить проект к использованию ClaimsIdentity: описать состав клеймов, места их формирования, хранение в cookie и использование для разграничения доступа по ролям.

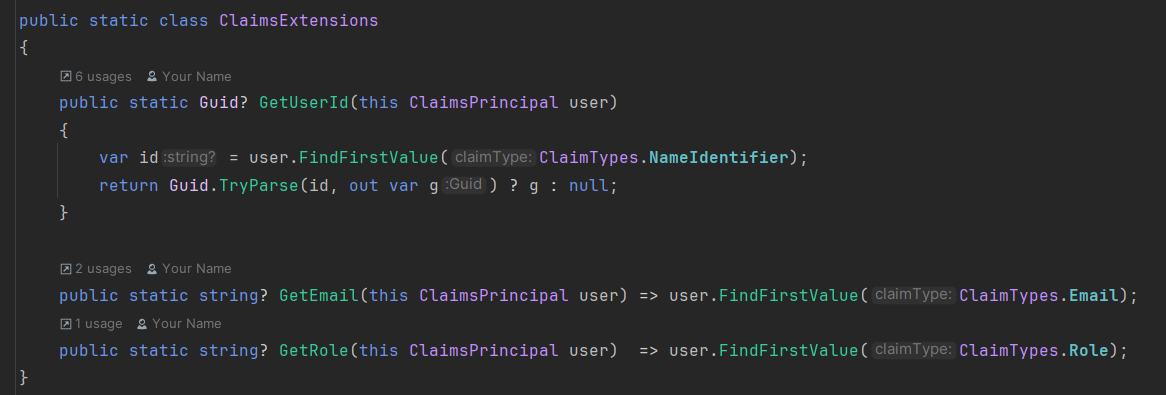
**Ход работы**

В «Program.cs» включена схема куки-аутентификации. При успешной регистрации и входе «AccountController.SignInAsync» формирует ClaimsIdentity с клеймами идентификатора, email и роли и создаёт аутентификационную cookie. Состав клеймов: «NameIdentifier» — строковый GUID пользователя; «Email» — почта; «Name» — отображаемое имя (в текущей версии совпадает с email); «Role» — «customer|manager|admin».

После регистрации «AccountController.Register (POST)» вызывает «AuthService.Register», а затем «SignInAsync», чтобы сразу создать куку. Вход выполняется «AccountController.Login (POST)» через «AuthService.Login» и «SignInAsync». Выход — «AccountController.Logout (POST)» с «SignOutAsync».

  
Рисунок 55 — Формирование ClaimsIdentity и запись cookie

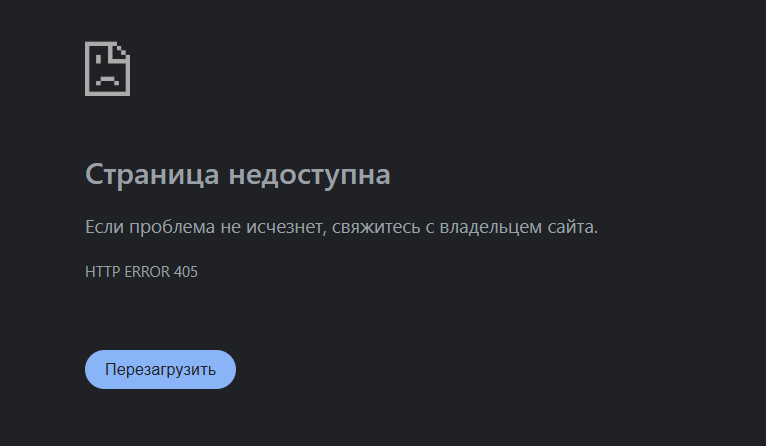
Для чтения клеймов и единообразного доступа добавлены расширения «Common/ClaimsExtensions.cs»: «GetUserId()», «GetEmail()», «GetRole()». Они используют «HttpContext.User.Claims» и возвращают значения в типах, удобных для контроллера и сервисов.

  
Рисунок 56 — Утилиты работы с клеймами

Ограничение доступа по ролям настроено атрибутами: «AdminController» помечен «[Authorize(Roles="admin")]», а просмотр входящих сообщений — «[Authorize(Roles="admin, manager")]». Эти атрибуты сравнивают требуемые роли с клеймом «Role» в созданной ClaimsIdentity.

  
Рисунок 57 — Разграничение доступа по ролям

Если статус или роль пользователя меняются, а значение продублировано в клейме, куку следует переиздать тем же «SignInAsync», чтобы «HttpContext.User» отражал актуальные данные на следующих запросах.

  
Рисунок 58 — Проверка политик доступа и ролей

**Вывод:** настроено формирование ClaimsIdentity и выпуск cookie при регистрации/входе; клеймы стандартизированы; доступ к административным разделам ограничен по ролям через атрибуты авторизации.

**19.11.2025**

**Тема:** Подключение HttpContext.User и ClaimsIdentity. Оформление отчетной документации.

**Цель работы:** подключить «HttpContext.User» в контроллерах и представлениях для получения текущего пользователя, использования клеймов и фильтрации данных без дополнительных запросов.

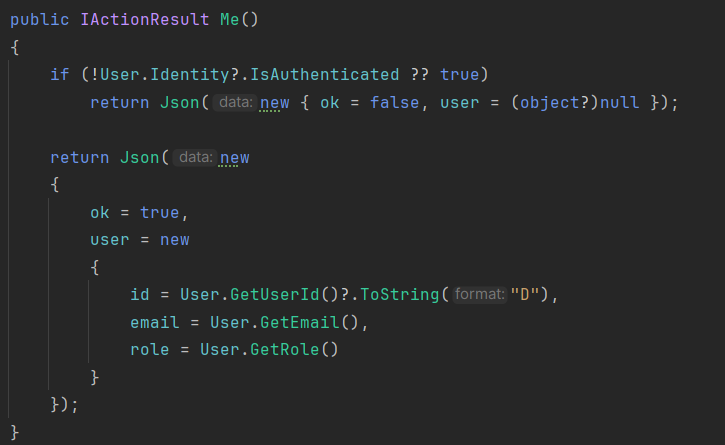
**Ход работы**

В контроллерах используется «HttpContext.User» (доступен как свойство «Controller.User»). В «RentalsController.Create (POST)» извлекается идентификатор текущего пользователя «User.GetUserId()», дата-время нормализуется к UTC и передаётся в «RentalService.CreateAsync». В «RentalsController.My (GET)» возвращается пагинированный список аренд только текущего пользователя — по его «userId».

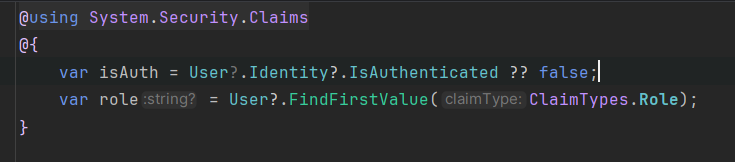


Рисунок 59 — Связка аренды с текущим пользователем

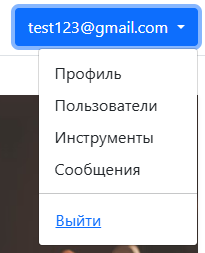
Страница профиля «AccountController.Profile (GET)» использует «User.GetEmail()» для выборки входящих сообщений пользователя «contacts.ListMineAsync(email, …)», а «User.GetUserId()» — для загрузки карточки пользователя. Таким образом, часть операций не требует промежуточных обращений к БД за идентификатором/почтой.

  
Рисунок 60 — Использование HttpContext.User в действиях аккаунта

В представлениях Razor «User.Identity.IsAuthenticated» управляет состоянием UI: в «Views/Shared/\_Layout.cshtml» при входе отображаются пункты «Профиль» и «Выход», при выходе — кнопки «Войти» и «Регистрация». На странице оформления аренды «Views/Rentals/Create.cshtml» формы создания доступны только аутентифицированному пользователю; для гостя выводится предупреждение с открытием модалок входа/регистрации.

  
Рисунок 61 — Подключение HttpContext.User в Razor-представлениях

Для административных страниц используется сочетание «[Authorize]» и чтение «User.IsInRole("admin")» там, где требуется тонкая проверка на уровне разметки (например, показать/скрыть кнопки «Удалить» на «Views/Admin/Users.cshtml»). Это опирается на клейм «Role», сформированный в момент входа.

  
Рисунок 62 — Практическое использование HttpContext.User в UI

**Вывод:** «HttpContext.User» подключён в контроллерах и Razor; клеймы читаются через расширения; фильтрация данных и условный интерфейс строятся на текущем пользователе без лишних запросов.

**20.11.2025**

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

**Цель работы:** внедрить валидацию данных авторизации и регистрации на стороне сервера с помощью C# и FluentValidation и на стороне браузера с использованием HTML5 и JavaScript.

**Ход работы**

Валидация данных для входа и регистрации реализуется на уровне контроллера «AccountController» (файл «Controllers/AccountController.cs») и отдельных классов-валидаторов в пространстве имен «WebPracticalProject.Validation». Для сериализации входных данных используются вложенные классы «LoginRequest» и «RegisterRequest» внутри «AccountController», которые принимаются методами «Login» и «Register» с атрибутом «[FromBody]», то есть ожидают JSON-запросы от фронтенда.

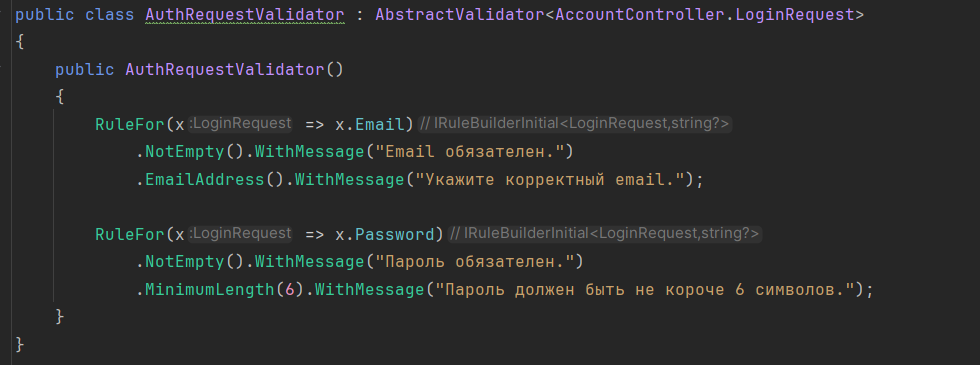
Серверная валидация строится на базе библиотеки FluentValidation. Для моделей «AccountController.LoginRequest» и «AccountController.RegisterRequest» созданы валидаторы «AuthRequestValidator» и «RegisterRequestValidator» в файле «WebPracticalProject.Validation/AuthRequestValidator.cs». В первом валидаторе проверяется наличие email и его формат, а также обязательность пароля и минимальная длина. Во втором валидаторе отдельно проверяется имя пользователя (обязательное поле и минимальная длина), корректность формата email, обязательность и длина пароля, а также совпадение пароля и подтверждения. Эти правила привязаны к конкретным свойствам моделей и при несоблюдении заполняют коллекцию ошибок модели.

Подключение FluentValidation к пайплайну ASP.NET MVC выполняется в конфигурации сервисов в «Program.cs» через вызовы «AddFluentValidationAutoValidation» и «AddFluentValidationClientsideAdapters». Дополнительно регистрация валидаторов выполняется вызовами «AddValidatorsFromAssemblyContaining()» и «AddValidatorsFromAssemblyContaining()», что позволяет автоматически находить и применять правила ко всем моделям в рамках сборки. Параметр «DisableDataAnnotationsValidation = true» отключает встроенную проверку атрибутов данных, оставляя единственным источником серверных ошибок FluentValidation.

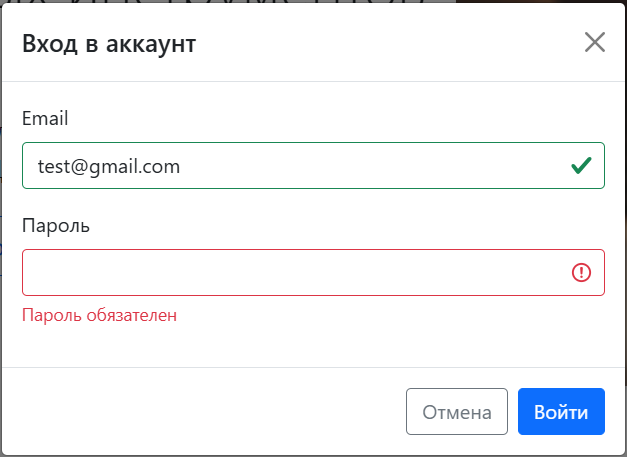
Методы «Register» и «Login» контроллера «AccountController» сначала проверяют, что объект dto не равен null, а затем анализируют «ModelState». При наличии ошибок формируется словарь с ключами, соответствующими названиям полей, и массивами текстов ошибок. Этот словарь возвращается в клиент в виде JSON-объекта формата «{ ok = false, errors = { ... } }» через «BadRequest». При успешной валидации создаются DTO для сервисного слоя («RegisterDto» и «LoginDto») и передаются в «IAuthService», после чего при успешной аутентификации вызывается приватный метод «SignInAsync», который формирует набор claim и создаёт cookie-аутентификацию через «HttpContext.SignInAsync». Клиенту при успехе возвращается JSON вида «{ ok = true, user = { ... } }», который далее используется фронтендом.

Клиентская валидация модальных окон для входа и регистрации реализуется в представлении макета «Views/Shared/\_Layout.cshtml». Внутри модальных окон «#loginModal» и «#registerModal» используются стандартные поля ввода с атрибутами «required», «type="email"», «minlength» и блоками «div» с классом «invalid-feedback» для отображения текстов ошибок браузерной валидации. В нижней части layout-файла подключён скрипт, который находит все формы с классом «.needs-validation» и при событии «submit» вызывает стандартный метод браузера «checkValidity». Если форма не проходит проверку, отправка предотвращается, добавляется класс «was-validated» и визуализируются сообщения «invalid-feedback» согласно разметке Bootstrap.

Асинхронная отправка форм входа и регистрации выполняется в скрипте «wwwroot/js/app-interactions.js». Там с помощью функции-хелпера «FetchHelpers.wireUpAjaxForm» формам в модальных окнах «#loginModal form» и «#registerModal form» подключается обработка AJAX-отправки. После ответа сервера колбэк «onSuccess» анализирует JSON: при «ok = true» отображается всплывающее сообщение о входе и модальное окно скрывается через «bootstrap.Modal.getInstance(...).hide()», затем выполняется перезагрузка страницы для подтягивания актуального состояния авторизованного пользователя. Если сервер возвращает «ok = false» или сообщение об ошибке, на фронтенде через «AppToast.show» отображается текст, полученный с сервера (например, сообщение о неверной паре логина и пароля или о занятости email).

  
Рисунок 63 — Валидаторы FluentValidation для авторизации

В браузере модальные окна «Вход в аккаунт» и «Регистрация» используют как встроенную HTML5-валидацию, так и последующую проверку на сервере. При неверно заполненном поле (например, слишком коротком пароле) перед отправкой формы пользователь видит подсветку полей и текст внутри блока «invalid-feedback». После успешной отправки JSON-запроса к методам «/Account/Login» или «/Account/Register» скрипт отображает всплывающие уведомления о входе или регистрации и закрывает модальные окна.

  
Рисунок 64 — Клиентская валидация модальной формы регистрации

**Вывод:** реализована серверная валидация авторизации и регистрации с использованием C# и FluentValidation, а также клиентская валидация модальных форм входа и регистрации на базе HTML5 и JavaScript с интеграцией AJAX-отправки.

**21.11.2025**

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

**Цель работы:** внедрить валидацию данных для пользовательских форм проката и обратной связи на стороне сервера с использованием C#, FluentValidation и механизмов обработки ошибок в JavaScript.

**Ход работы**

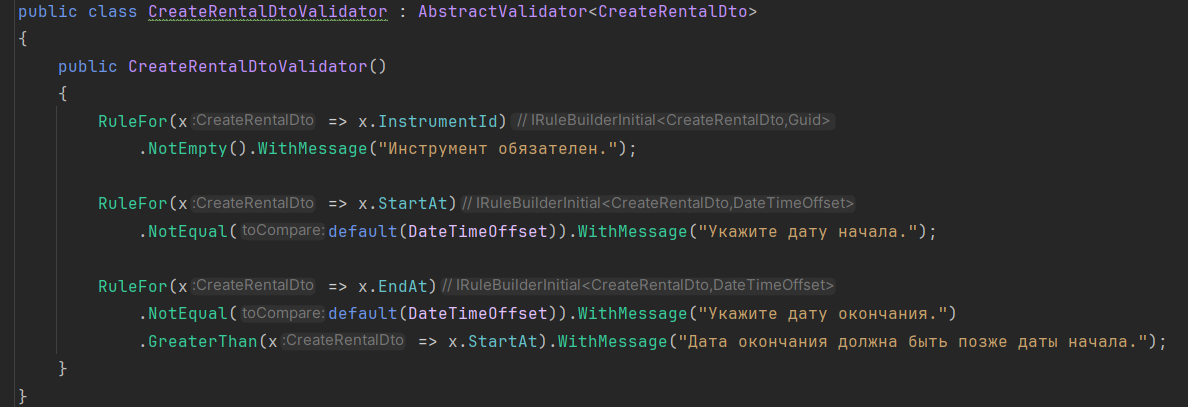
Для проверки данных, связанных с оформлением аренды, используются объекты, описанные в контроллере «RentalsController» (файл «Controllers/RentalsController.cs») и валидаторы в пространстве имен «WebPracticalProject.Validation». Для страницы оформления аренды заданы две отдельные модели: «CreateRentalForm» как входная модель для метода «Create (POST)» и «CreateRentalPageVm» как модель представления для «Create (GET)». Валидация данных для JSON-запроса реализована на уровне «CreateRentalForm», которую принимает метод «Create» с атрибутом «[FromBody]».

Класс «CreateRentalFormValidator» расположен в файле «WebPracticalProject.Validation/CreateRentalFormValidator.cs» и наследуется от «AbstractValidator<RentalsController.CreateRentalForm>». Внутри него определены правила: обязательное заполнение «InstrumentId», запрет значения по умолчанию для «StartAt» и «EndAt», а также ограничение «EndAt» строго больше времени начала аренды. Эти проверки предотвращают ситуации, когда пользователь отправляет пустой идентификатор инструмента или некорректный диапазон дат, ещё до передачи данных в сервисный слой.

Дополнительная валидация для слоя DTO реализуется в классе «CreateRentalDtoValidator» в файле «WebPracticalProject.Validation/CreateRentalDtoValidator.cs». Этот валидатор применяется к объектам «CreateRentalDto» из пространства имен «WebPracticalProject.Service.Dto» и дублирует базовые требования к идентификатору инструмента и диапазону дат. Таким образом, данные проверяются как на уровне входной модели контроллера, так и на уровне DTO, которые передаются в «IRentalService». Это позволяет гарантировать консистентность требований при использовании этих DTO из различных мест приложения.

Подключение валидаторов аренды к контейнеру выполняется в «Program.cs» через вызовы «AddValidatorsFromAssemblyContaining()» и «AddValidatorsFromAssemblyContaining()» вместе с уже описанной конфигурацией «AddFluentValidationAutoValidation» и клиентскими адаптерами. Благодаря этому, при обработке запроса к «RentalsController.Create» модель «CreateRentalForm» автоматически проверяется FluentValidation до выполнения тела метода. При наличии ошибок метод формирует JSON-ответ «BadRequest» с объектом «{ ok = false, errors = { ... } }», где ключи соответствуют названиям полей, а значения — массивам сообщений. При успешной валидации контроллер дополнительно проверяет существование инструмента через сервис «IInstrumentService», переводит даты в UTC, создаёт объект «CreateRentalDto» и вызывает «IRentalService.CreateAsync».

Фронтенд-часть обработки аренды реализована в файле «wwwroot/js/app-interactions.js». Там выбирается форма с идентификатором «#rentalForm», и к ней через «FetchHelpers.wireUpAjaxForm» подключается AJAX-отправка. После ответа сервера колбэк «onSuccess» анализирует JSON: при «ok = true» показывается всплывающее сообщение и выполняется перенаправление на страницу «/Rentals/My» или на URL из «payload.redirectUrl». Если сервер возвращает «ok = false», отображается уведомление с текстом из «payload.message», а ошибки валидации, связанные с полями, остаются в JSON-структуре «errors», доступной для дальнейшей обработки.

  
Рисунок 65 — Валидаторы FluentValidation для формы аренды и DTO аренды

Валидация для формы обратной связи реализована в контроллере «ContactController» (файл «Controllers/ContactController.cs»). Метод «Send» принимает объект «ContactForm» с атрибутом «[FromBody]», проверяет «ModelState» и при наличии ошибок формирует JSON с полем «ok = false» и словарём «errors». При успешной проверке данные транслируются в DTO «CreateContactDto» из пространства имен «WebPracticalProject.Service.Dto» и передаются в «IContactService.CreateAsync». Ответом клиенту служит JSON вида «{ ok = true, id = ... }». На фронтенде форма обратной связи находится внутри секции «#message», а в «wwwroot/js/app-interactions.js» для неё также вызывается «FetchHelpers.wireUpAjaxForm». При успешном ответе форма сбрасывается, класс валидации снимается и через «AppToast.show» отображается уведомление об отправке сообщения.

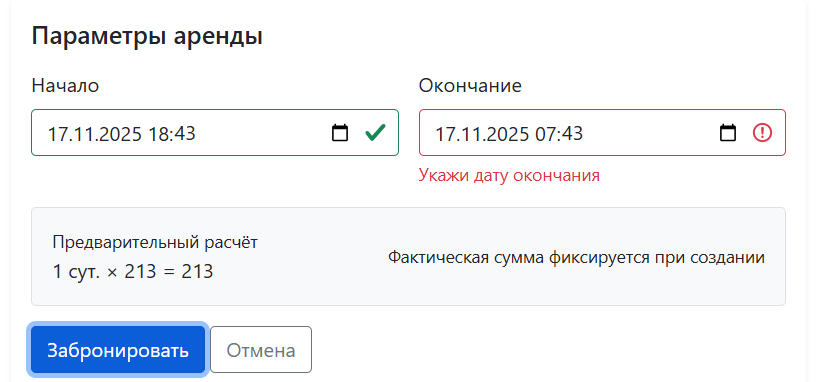


Рисунок 66 — Обработка ошибок валидации аренды на стороне клиента

**Вывод:** реализована многоуровневая валидация данных для оформления аренды и формы обратной связи с использованием FluentValidation на стороне сервера и обработкой JSON-ответов в JavaScript для отображения ошибок и статуса выполнения операций.

**22.11.2025**

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

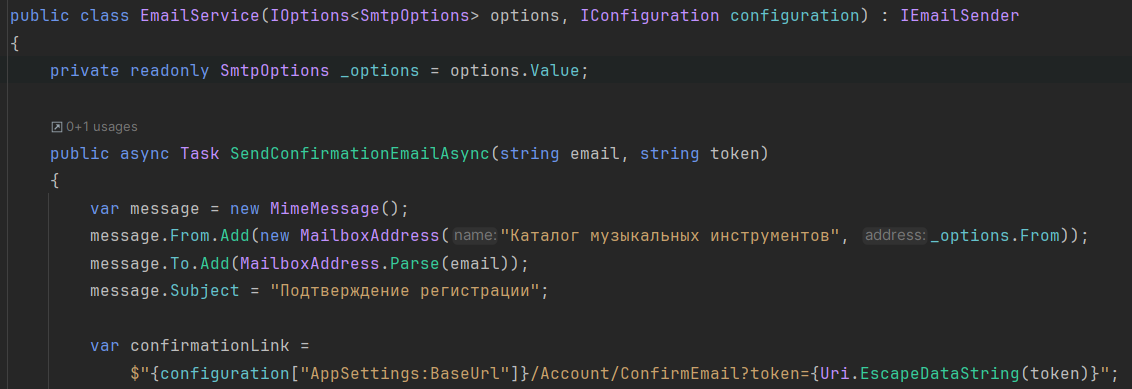
**Цель работы:** настроить отправку почтовых сообщений через SMTP-сервер Gmail и вынести параметры подключения в конфигурацию приложения.

**Ход работы**

В файле конфигурации «appsettings.json» добавлена секция «EmailSettings», содержащая хост, порт, учётные данные SMTP и адрес отправителя. Эти параметры далее считываются на уровне сервисного слоя и не захардкожены в коде.

  
Рисунок 67 — Конфигурация SMTP Gmail в appsettings.json

В слое Service создан сервис отправки почты «EmailService» («Service/Services/EmailService.cs»), реализующий интерфейс «IEmailSender» из «Service/Interfaces/IEmailSender.cs». Сервис получает «IConfiguration», формирует MIME-сообщение с HTML-телом и использует SMTP-клиент MailKit для подключения к «smtp.gmail.com» по порту 587 с режимом «StartTLS». Логика отправки вынесена в отдельный метод, который принимает email получателя и токен подтверждения и формирует ссылку для подтверждения с использованием базового URL из «AppSettings:BaseUrl».

  
Рисунок 68 — Сервис отправки писем через Gmail SMTP

В файле «Program.cs» сервис регистрируется в DI-контейнере как реализация интерфейса «IEmailSender». Таким образом, любой другой сервис (например, «AuthService») может вызывать отправку писем, не зная деталей SMTP-подключения. Конфигурация SMTP централизована, а логика отправки инкапсулирована в сервисе Service-слоя.

**Вывод:** реализована отправка почтовых сообщений через SMTP-сервер Gmail на основе настроек в «appsettings.json»; создан сервис «EmailService», доступный через интерфейс «IEmailSender» для использования в других частях приложения.

**24.11.2025**

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

**Цель работы:** реализовать подтверждение email пользователя при регистрации с использованием одноразового токена и записи состояния в базе данных.

**Ход работы**

В доменном слое создана сущность «EmailConfirmationToken» («Domain/Emails/EmailConfirmationToken.cs»), аннотированная атрибутом таблицы PostgreSQL «[Table("email\_confirmation\_tokens", Schema = "app")]». Для полей заданы явные имена столбцов («id», «user\_id», «token», «expires\_at», «used»), а навигационное свойство «User» связывает токен с конкретным пользователем из «Domain/Users/User».

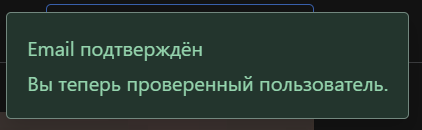


Рисунок 69 — Доменная сущность токена подтверждения email

В слое данных «AppDbContext» («DAL/AppDbContext.cs») добавлен DbSet «EmailConfirmationTokens» и конфигурация сущности в «OnModelCreating». Для таблицы «email\_confirmation\_tokens» настроены: генерация UUID через «gen\_random\_uuid()» по умолчанию для поля «id», тип столбца «timestamptz» для «expires\_at», флаг «used» с значением по умолчанию false, уникальный индекс по полю «token» для исключения дубликатов токенов и внешняя ссылка на «users(id)» с каскадным удалением. Дополнительно реализован репозиторий «EmailConfirmationRepository», предоставляющий методы добавления токена и поиска по значению «token» с фильтрацией по флагу «Used».

В сервисном слое переработан метод регистрации в «AuthService» («Service/Services/AuthService.cs»). После создания пользователя метод генерирует токен (URL-safe строка на основе Guid), сохраняет его через «IEmailConfirmationRepository» с указанием срока действия и флага «Used = false», а затем вызывает «IEmailSender.SendConfirmationEmailAsync», передавая email и токен. Таким образом, при успешной регистрации пользователю отправляется письмо с уникальной ссылкой вида «/Account/ConfirmEmail?token=...&email=...».

Контроллер «AccountController» в MVC-слое реализует действие «ConfirmEmail (GET)», принимающее параметры «token» и «email». Экшен вызывает «auth.ConfirmEmailAsync(token, email, ct)». В сервисе проверяется существование токена, его неиспользованное состояние, срок действия и соответствие email. При успешной проверке флаг «Used» переводится в true, а у пользователя в таблице «users» выставляется «email\_confirmed = true». После этого контроллер перенаправляет на главную страницу с параметром состояния подтверждения в query-строке.

  
Рисунок 70 — Подтверждение email через GET-запрос и редирект в интерфейсе

На стороне клиента (скрипт в «wwwroot/js/site.js») реализована обработка параметра «email\_confirmed» из query-строки: в зависимости от значения показывается toast-сообщение через «AppToast.show», после чего параметр удаляется из URL посредством «history.replaceState», предотвращая повторное отображение уведомления при обновлении страницы.

**Вывод:** реализована проверка подлинности пользователя при регистрации на основе токена подтверждения email, хранимого в таблице «email\_confirmation\_tokens», с обновлением флага «email\_confirmed» у пользователя и отображением результата в веб-интерфейсе.

**25.11.2025**

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

**Цель работы:** подготовить параметры OAuth-клиента Google для использования в авторизации и сохранить их в конфигурации приложения.

**Ход работы**

В консоли Google Cloud для проекта создан OAuth-клиент типа «Web application» с указанием разрешённого адреса обратного вызова «[https://localhost:7138/signin-google»](https://localhost:7138/signin-google%C2%BB). На стороне Google настроен экран доступа (OAuth consent screen), добавлены домен и список доступных API, после чего сгенерированы значения «Client ID» и «Client Secret» для текущего веб-приложения.

  
Рисунок 71 — Настройка OAuth-клиента Google в Google Cloud Console

В локальной конфигурации приложения создана секция «Authentication:Google» в «appsettings.json», содержащая значения «ClientId» и «ClientSecret», выданные Google Cloud. Эти параметры не используются напрямую во view или контроллерах, а считываются при настройке аутентификации в файле «Program.cs». Таким образом, доступ к ключам OAuth сосредоточен в конфигурации и может быть изменён без правки исходного кода.

**Вывод:** подготовлен OAuth-клиент Google в Google Cloud Console и параметры «ClientId»/«ClientSecret» сохранены в секции «Authentication:Google» файла «appsettings.json» для дальнейшего использования при настройке авторизации.

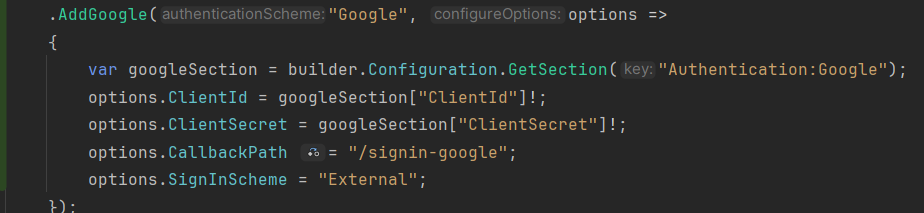
**26.11.2025**

**Тема:** Реализация авторизации Google. Оформление отчетной документации.

**Цель работы:** реализовать схему авторизации через Google OAuth и интегрировать её с существующей cookie-аутентификацией и пользовательской сущностью.

**Ход работы**

В файле «Program.cs» переработана конфигурация аутентификации: в «AddAuthentication» явно задана схема по умолчанию «CookieAuthenticationDefaults.AuthenticationScheme», для которой настроен cookie «wpp\_auth» (имя, политика SameSite, срок действия). К ней добавлена дополнительная cookie-схема «External», используемая для промежуточного хранения результата аутентификации через внешние провайдеры. Затем вызван «AddGoogle("Google", ...)», где из конфигурации «Authentication:Google» считываются «ClientId» и «ClientSecret», задан «CallbackPath = "/signin-google"» и «SignInScheme = "External"».

  
Рисунок 72 — Настройка схем аутентификации и провайдера Google в Program.cs

В MVC-слое добавлен контроллер «ExternalAuthController» с двумя действиями: «GoogleLogin» и «GoogleCallback». Действие «GoogleLogin» формирует «AuthenticationProperties» с «RedirectUri» на «GoogleCallback» и возвращает «Challenge(properties, "Google")», инициируя перенаправление пользователя на Google для входа. После успешного входа действие «GoogleCallback» выполняет «AuthenticateAsync("External")», извлекает из полученного «ClaimsPrincipal» значения email, имени пользователя и внешний идентификатор Google (claim «NameIdentifier»). Далее вызывается сервисный метод «SignInWithGoogleAsync» из «AuthService», который ищет пользователя по «google\_id», при отсутствии — по email, при необходимости создаёт новую запись в таблице «users» с флагом «email\_confirmed = true» и сохраняет привязку «google\_id». На основании доменной модели формируется собственный «ClaimsPrincipal» с идентификатором, email, отображаемым именем и ролью, который записывается в основную cookie-схему через «SignInAsync».

В пользовательском интерфейсе кнопка входа через Google добавлена в модальное окно входа во view «Views/Shared/\_Layout.cshtml» или отдельное представление логина. Кнопка ссылается на маршрут «/ExternalAuth/GoogleLogin?returnUrl=/», что позволяет вернуть пользователя на нужную страницу после завершения авторизации.

  
Рисунок 73 — Контроллер внешней авторизации и сервисный метод входа через Google

**Вывод:** настроена схема авторизации через Google OAuth, добавлен контроллер внешней авторизации и сервисный метод маппинга внешних данных на доменную модель, с записью аутентифицированного пользователя в основную cookie-схему.

**27.11.2025**

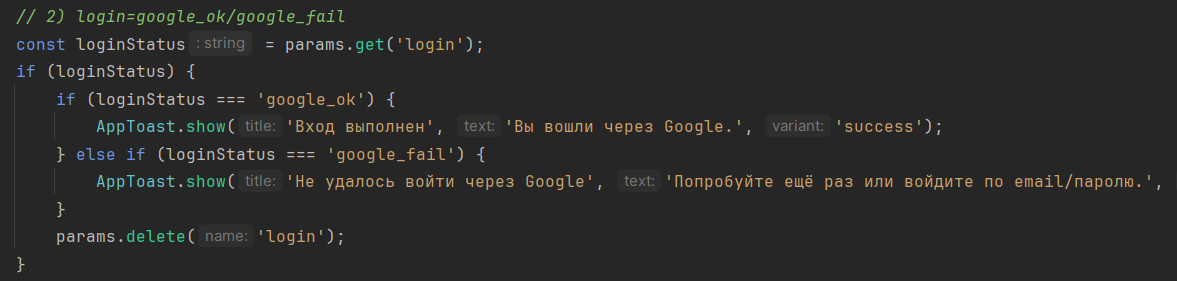
**Тема:** Доработка авторизации Google. Оформление отчетной документации.

**Цель работы:** доработать обработку результатов авторизации через Google с показом статуса входа и очисткой параметров из URL.

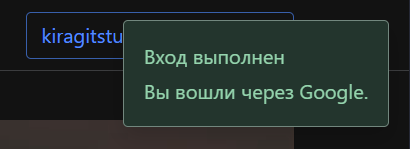
**Ход работы**

В действии «GoogleCallback» контроллера «ExternalAuthController» после успешного создания «ClaimsPrincipal» и записи пользователя в основную cookie-схему выполняется перенаправление на целевой URL с query-параметром «login=google\_ok», а при неуспехе — редирект с параметром «login=google\_fail». Таким образом, информация о результате входа передаётся на клиентскую часть через адресную строку без использования состояния на сервере.

На стороне клиента в общем скрипте («wwwroot/js/site.js») в обработчике «DOMContentLoaded» добавлена логика анализа текущего URL. С помощью интерфейса «URL» и «searchParams» из query-строки извлекается параметр «login». При значении «google\_ok» вызывается «AppToast.show» c коротким сообщением о выполненном входе через Google, при «google\_fail» — сообщение об ошибке авторизации. После обработки параметр «login» удаляется из «searchParams», и новая строка запроса записывается в историю браузера методом «history.replaceState», что визуально возвращает URL к обычному виду без технических параметров и предотвращает повторное появление toast-уведомлений при обновлении страницы.

  
Рисунок 74 — Логика отображения результатов авторизации Google и очистки URL в клиентском скрипте

В результате при завершении авторизации через Google пользователь перенаправляется на главную страницу или указанный returnUrl, где однократно отображается toast-уведомление о статусе входа. После этого адресная строка очищается от параметров «login=…», и при последующих переходах или обновлении страницы никаких дополнительных уведомлений не показывается.

  
Рисунок 75 — Отображение результата авторизации Google через toast и очищенный URL

**Вывод:** выполнена доработка авторизации через Google: контроллер возвращает результат через query-параметр, а клиентский скрипт отображает соответствующее toast-уведомление и очищает URL, обеспечивая однократное отображение статуса входа.

**28.11.2025**

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

**Цель работы:** ЦЕЛЬ

**Ход работы**

СОДЕРЖАНИЕ

**29.11.2025**

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

**Цель работы:** ЦЕЛЬ

**Ход работы**

СОДЕРЖАНИЕ

**1.12.2025**

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

**Цель работы:** ЦЕЛЬ

**Ход работы**

СОДЕРЖАНИЕ

**2.12.2025**

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

**Цель работы:** ЦЕЛЬ

**Ход работы**

СОДЕРЖАНИЕ

МАКЕТ:

**12.11.2025**

**Тема:** ТЕМА Оформление отчетной документации.

**Цель работы:** ЦЕЛЬ

**Ход работы**

СОДЕРЖАНИЕ