**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)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Специальность \_\_2.09.02.07 Информационные системы и программирование

код, наименование специальности

Курс 3 группа 315

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.

**31.10.25**

**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.

**Ход работы**:

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



Рисунок 1

* Выбираем шаблон «Веб-приложение *ASP.NET Core* (модель-представление-контроллер)» и нажимаем далеее

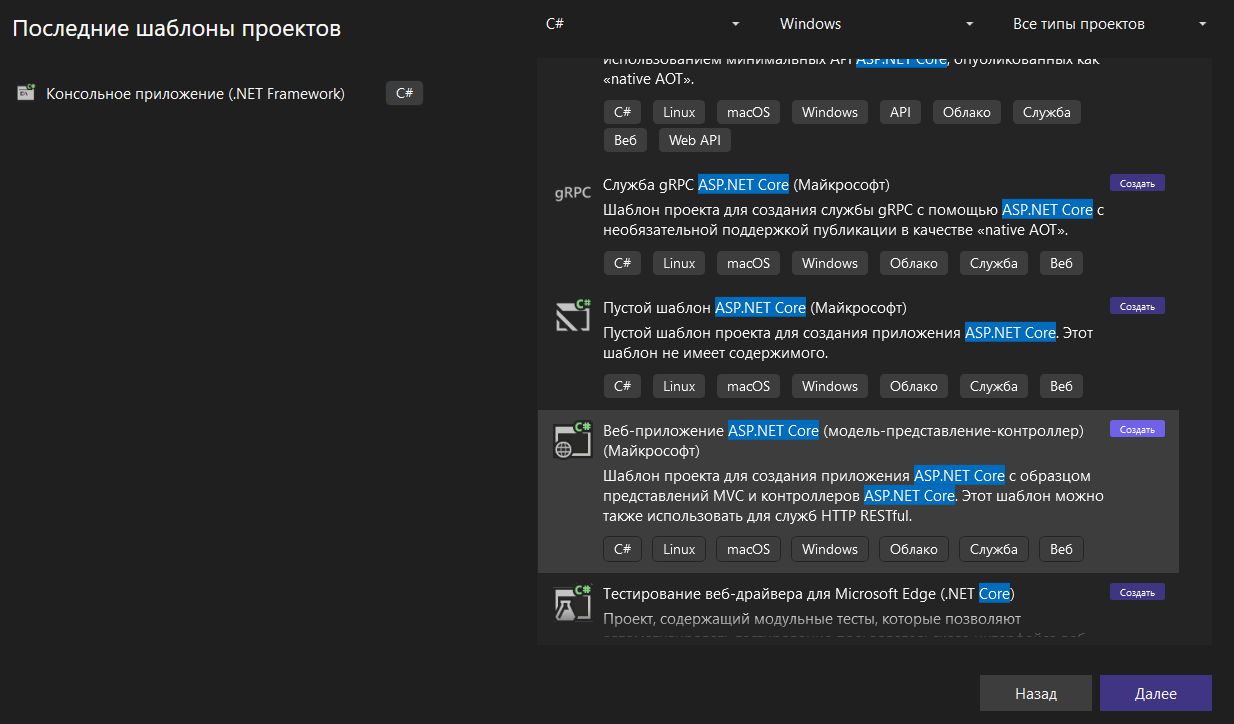


Рисунок 2

* Настраиваем новый проект. Называем его как хотим и помещаем в наш репозиторий.

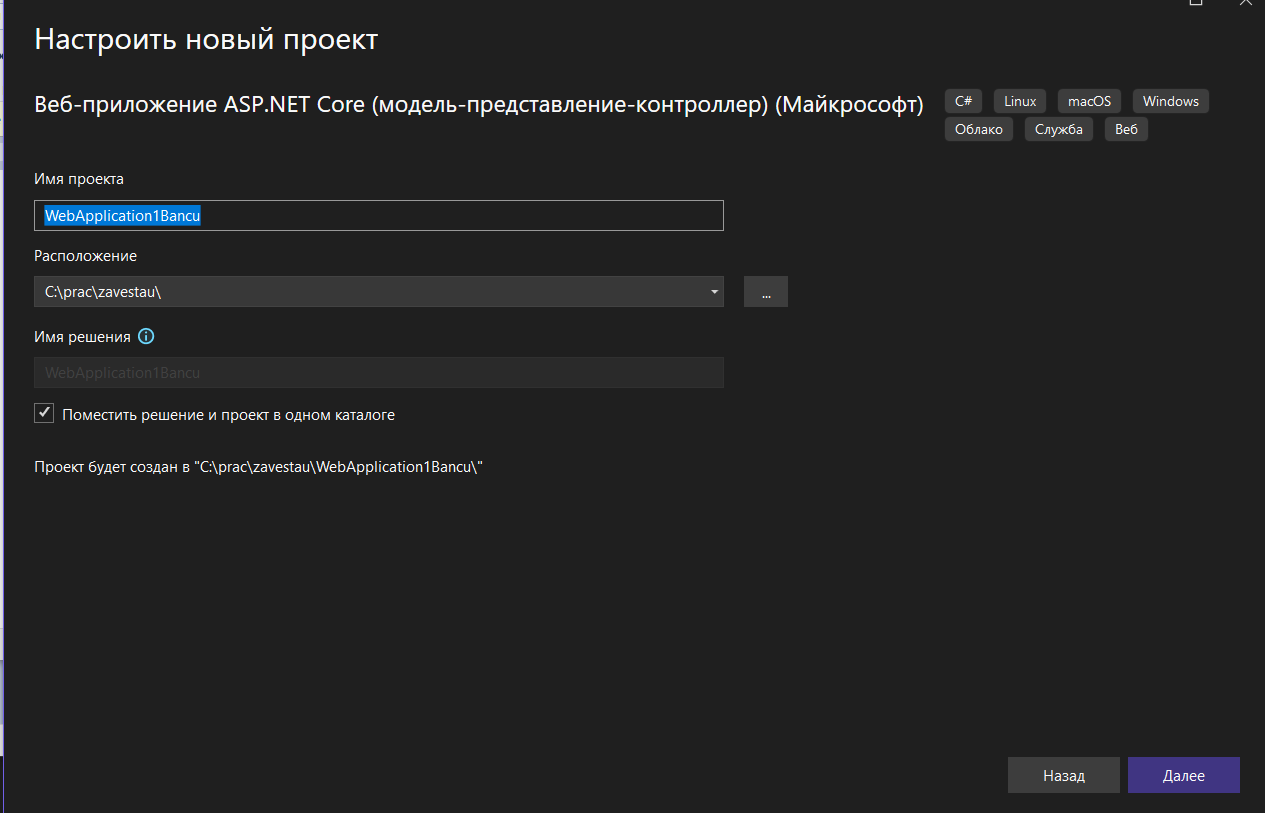


Рисунок 3

* В обозревателе решений показана структура данного шаблона на начальном этапе разработки.

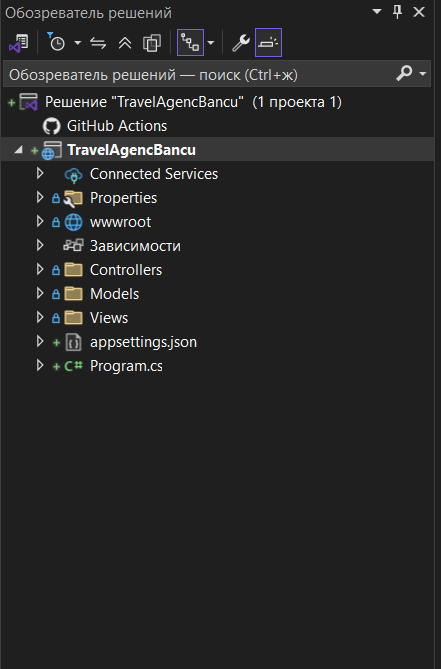


Рисунок 4

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.*

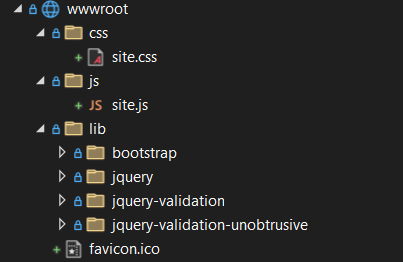


Рисунок 5

Папка *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.

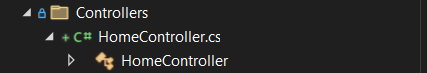


Рисунок 6

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

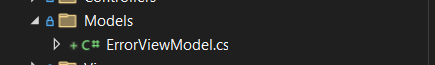


Рисунок 7

Папка Views, на рисунке 8 – представляет собой набор HTML-шаблонов, которые сервер возвращает пользователю после обработки запроса. В ней содержатся основные файлы, обеспечивающие отображение страниц приложения.

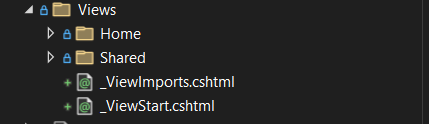


Рисунок 7

* *\_Layout.cshtml*

Файл \_Layout.cshtml служит мастер-страницей, задающей общий шаблон для Razor-страниц. Он используется для формирования единого и согласованного внешнего вида всего веб-приложения. В нём применяются такие теги, как <head>, предназначенный для хранения служебной информации страницы; <header>, который отделяет вводное содержимое или навигационное меню от основного блока данных; и <footer>, формирующий нижнюю часть сайта, где обычно размещаются контактные сведения, ссылки или дополнительное меню. Также файл содержит метод @RenderBody(), который отвечает за вставку контента из других Razor-страниц, использующих данный макет. Благодаря этому обеспечивается единообразие дизайна и структуры всех страниц приложения.

* *\_ViewImports.cshtml*

Файл \_ViewImports.cshtml предназначен для глобального подключения пространств имён и общих настроек, применяемых ко всем представлениям проекта ASP.NET Core MVC.

* *\_ViewStart.cshtml*

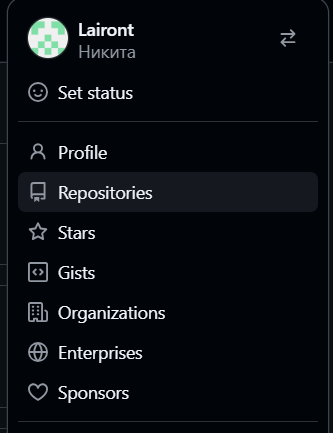
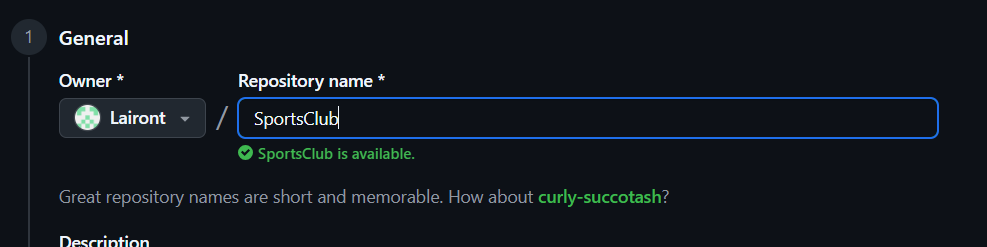
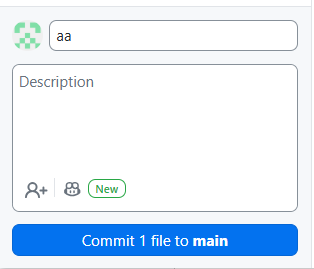
Файл \_ViewStart.cshtml определяет, какой макет (layout) будет использоваться для текущего представления.

**Вывод**: В ходе работы был создан проект на основе архитектуры MVC в среде Rider и изучена его структура: папка wwwroot содержит стили, скрипты и статические ресурсы; Controllers — отвечает за обработку пользовательских запросов; Models — управляет данными; а Views — обеспечивает визуальное отображение информации. Особое внимание уделено мастер-странице \_Layout.cshtml, которая позволяет задать единый шаблон интерфейса с общими элементами, такими как меню и оформление, что значительно упрощает сопровождение и изменение страниц приложения.

**01.11.2025**

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

**Ход работы:**

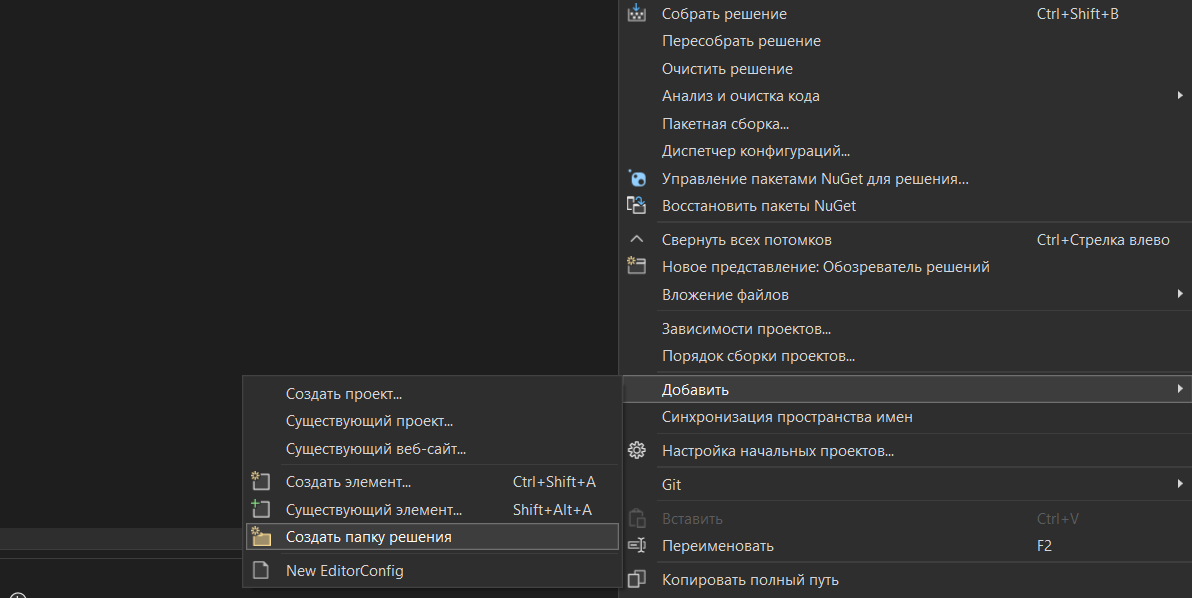
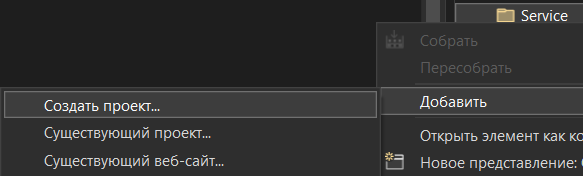
1. Создали личный аккаунт на GitHub — предложено перейти на сайт GitHub.com, выбрать пункт «Регистрация» и следовать инструкциям для создания аккаунта.  
     
   Рисунок 8 – созданный нами аккаунт
2. Создали удаленный репозиторий — после входа в меню пользователя и выбора раздела «Repositories» нажали на «New», указали название репозитория и сохранили его, как показано на рисунке 9.  
     
     
   Рисунок 9 – создание удаленного репозитория
3. Выполнили первый вход в GitHub Desktop — при запуске GitHub Desktop ввели логин и пароль для получения доступа ко всем репозиториям.
4. Создали локальный репозиторий — на главном экране GitHub Desktop выбрали пункт «Create a New Repository», указали название и папку для сохранения файлов.
5. Клонировали репозиторий — выбрали «Add -> Clone Repository...», указали нужный репозиторий, после чего ждем скачивание файлов.
6. При добавлении какого-либо файла в репозиторий, нажимаем на кнопку «Commit to main» как показано на рисунке 10.  
     
   Рисунок 10 – сохранение проекта
7. Опубликовали репозиторий на GitHub — после нажатия «Publish repository» проверили название и описание, затем нажали «Publish».
8. Просмотрели и отправили изменения — после внесения изменений в файлы создали коммит и нажали «Push origin», отправив изменения в удалённый репозиторий.
9. Проверили изменения на GitHub — после завершения работы изменения отобразились в удалённом репозитории

**Вывод:** В ходе работы с Git и GitHub Desktop выполнено создание и настройка аккаунта, организация локального и удалённого репозиториев, а также освоены основные операции: добавление файлов, создание коммитов и отправка изменений на GitHub.

**03.11.2025**

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

**Ход работы:**

1. Рассмотрено правило зависимостей – это правило которое гласит, что зависимости в исходном коде должны указывать исключительно внутрь. Внутренние компоненты не должны иметь знаний о внешних компонентах и не могут ссылаться на них. Это правило касается всех элементов, включая функции, классы и переменные.
2. Рассмотрены сущности – это объекты либо структуры данных и функций. Их смысл в том, что они не должны быть изменены от внешних факторов.
3. Рассмотрены сценарии – это такие данные, в которых если что-то изменить, то это не будет влиять на весь код, но если будут какие-либо изменения в работе приложения, то это повлияет на сценарии.
4. Рассмотрены интерфейс адаптеры – это конверторы, которые преобразуют какие-либо форматы на наиболее удобные форматы сущностей и сценариев.
5. Рассмотрены фреймворки и драйверы – внешний слой, содержащий детали реализации (например, работу с базами данных) и взаимодействующий с внутренними слоями через адаптеры.
6. Рассмотрено пересечение границ, осуществляемое с использованием инверсии зависимостей, что обеспечивает взаимодействие слоёв через интерфейсы и сохраняет целостность архитектуры.
7. Рассмотрена передача данных через границы, при которой применяются DTO и аналогичные структуры для изоляции внутреннего слоя от данных внешнего.
8. Выполнено изменение структуры проекта: для реализации архитектуры созданы отдельные проекты («DAL», «Domain», «Service»), пошаговое построение которых представлено на рисунке 11. Такая организация способствует разделению бизнес-логики, данных и сервисов в соответствии с принципами чистой архитектуры.Рисунок 11 – создание папок.Рисунок 12 – создание проекта в папке.*Data Access Layer (DAL)* - это слой доступа к данным, отвечающий за взаимодействие с базой данных или другими источниками данных.   
   DAL включает в себя методы для выполнения CRUD-операций (создание, чтение, обновление и удаление данных). С его помощью приложения могут получать данные из базы, преобразовывать их в объекты и передавать другим слоям без непосредственного обращения к базе данных  
   Domain - это слой, содержащий бизнес-логику и сущности (модели) приложения. Он представляет собой основу системы, состоящую из объектов и логики, отражающей реальные процессы и правила предметной области. В нем описаны модели данных и их связи, а также основные операции, связанные с ними. Domain независим от деталей реализации DAL или представлений, что позволяет сохранить чистую бизнес-логику.  
   Service - это слой, связывающий Domain и DAL, а также обеспечивающий дополнительную бизнес-логику и координирующий взаимодействие различных слоев.

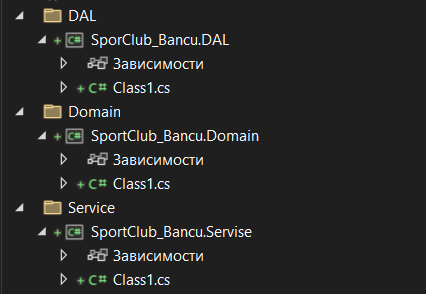


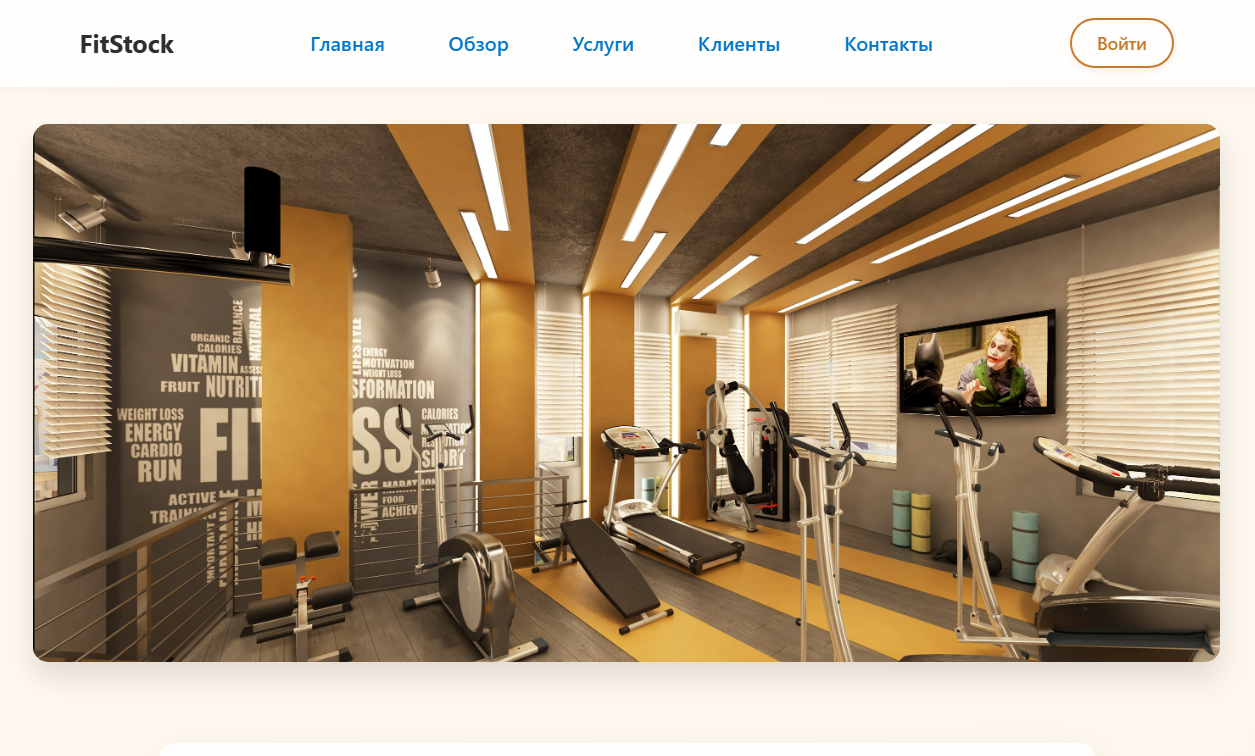
Рисунок 13 – Конечный результат. **Вывод:** применение принципов чистой архитектуры в проекте MVC обеспечивает чёткое разделение обязанностей между слоями и упорядочивает взаимодействие бизнес-логики, интерфейсных компонентов и внешних элементов, включая базы данных и пользовательские интерфейсы. Благодаря Правилу зависимостей внутренние слои сохраняют независимость от внешних, что упрощает модификацию интерфейсов и баз данных без изменения основной логики. Структурирование проекта по слоям — Сущности, Сценарии, Интерфейс-Адаптеры и Фреймворки — повышает гибкость и модульность системы, облегчая поддержку и масштабирование приложения.

**04.11.25**

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

**Ход работы:**

1. В рамках работы над основным дизайном сайта был разработан и стилизован хедер (шапка) страницы. Он содержит логотип и навигационное меню, оформленное с помощью градиента и фиксированного позиционирования, чтобы всегда оставаться в верхней части страницы при прокрутке. Структура хедера позволяет пользователям быстро переходить между разделами сайта, такими как «Услуги», «Клиенты» и «Главная».

Рисунок 14 – Хедер

*HTML* (от английского *HyperText Markup Language*) — это язык гипертекстовой разметки текста. Он нужен, чтобы размещать на веб-странице элементы: текст, картинки, таблицы и видео.

1. В CSS был добавлен набор стилей для оформления элементов хедера, включая настройки для логотипа, навигационного меню и кнопок. Фиксированное позиционирование (position: fixed) делает хедер доступным на всех уровнях прокрутки страницы. Это улучшает навигацию по сайту, позволяя пользователю иметь быстрый доступ к основным разделам без необходимости прокручивать страницу вверх. Эффекты при наведении, такие как изменение цвета фона и увеличение логотипа, делают интерфейс более интерактивным и привлекательным.

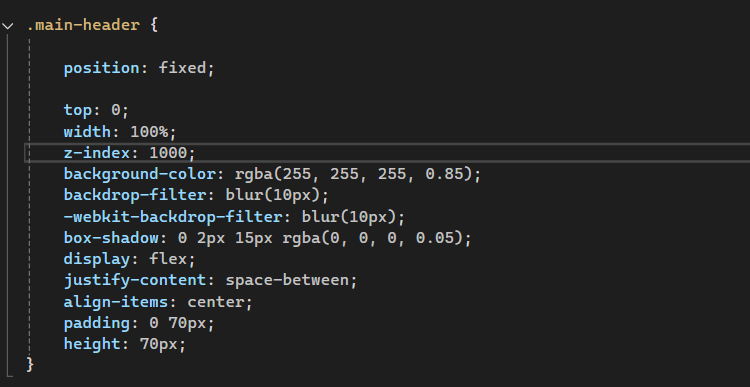


Рисунок 15 – Код хедера

**Вывод:** Разработка хедера сайта, включая структуру и оформление, была успешно выполнена и отвечает современным требованиям веб-дизайна. Примененные методы и стили обеспечивают не только привлекательный внешний вид, но и удобство для пользователей. Пользователь может легко ориентироваться в навигации, переходя между разделами сайта, а функциональность и интуитивность интерфейса способствуют повышению качества восприятия и взаимодействия с контентом. Удобство использования, улучшенная визуализация и интерактивные элементы помогают повысить общую удовлетворенность пользователей и привлекают внимание к ключевым разделам сайта

**05.11.2025**

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

**Ход работы:**

1. Изменяем файл *SiteIinfo,* добавляем карточки для меню «Услуги». Данные действия показаны на рисунке 23.

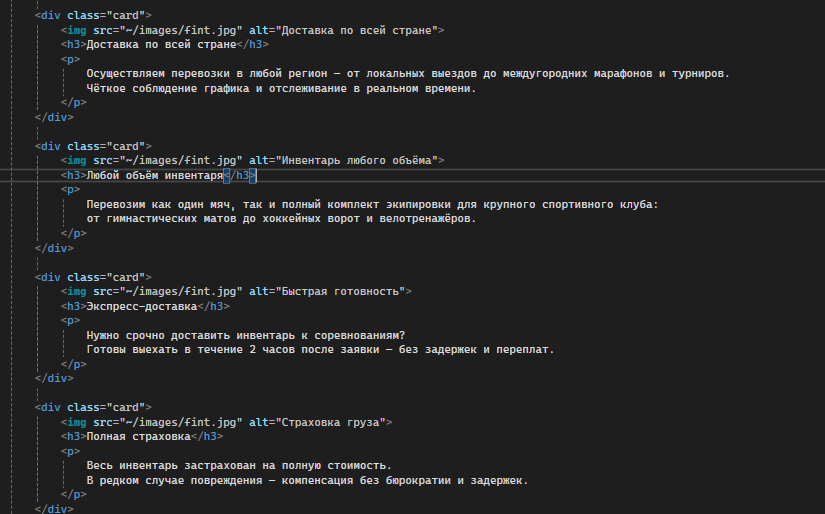


Рисунок 16 – разметка «Услуги»

1. На рисунке 17 показаны стили для «Услуг».



Рисунок 17 – «Услуги» стили для карточек

1. На рисунке 18 показан результат секции «Услуги».

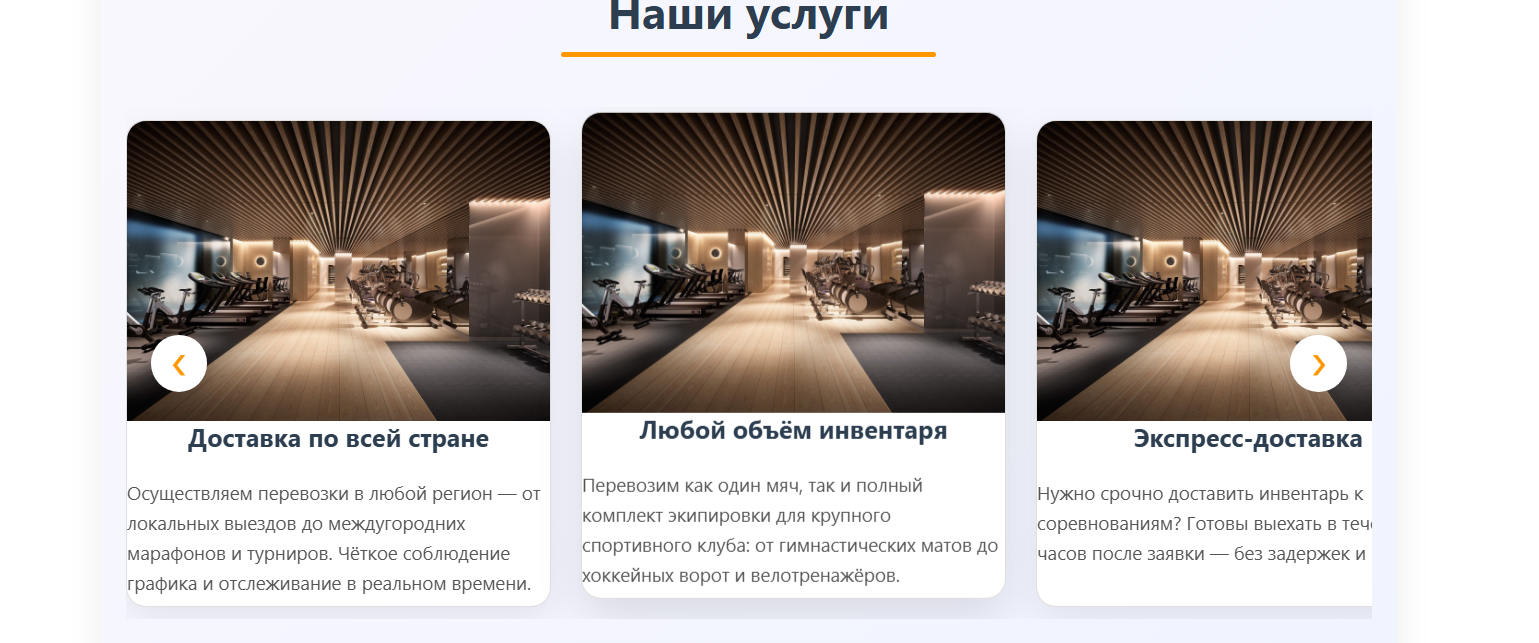


Рисунок 18 – результат секции «Услуги»

1. Для добавления Яндекс карты в наш проект, нам необходимо зайти на сайт, показан на рисунке 19.

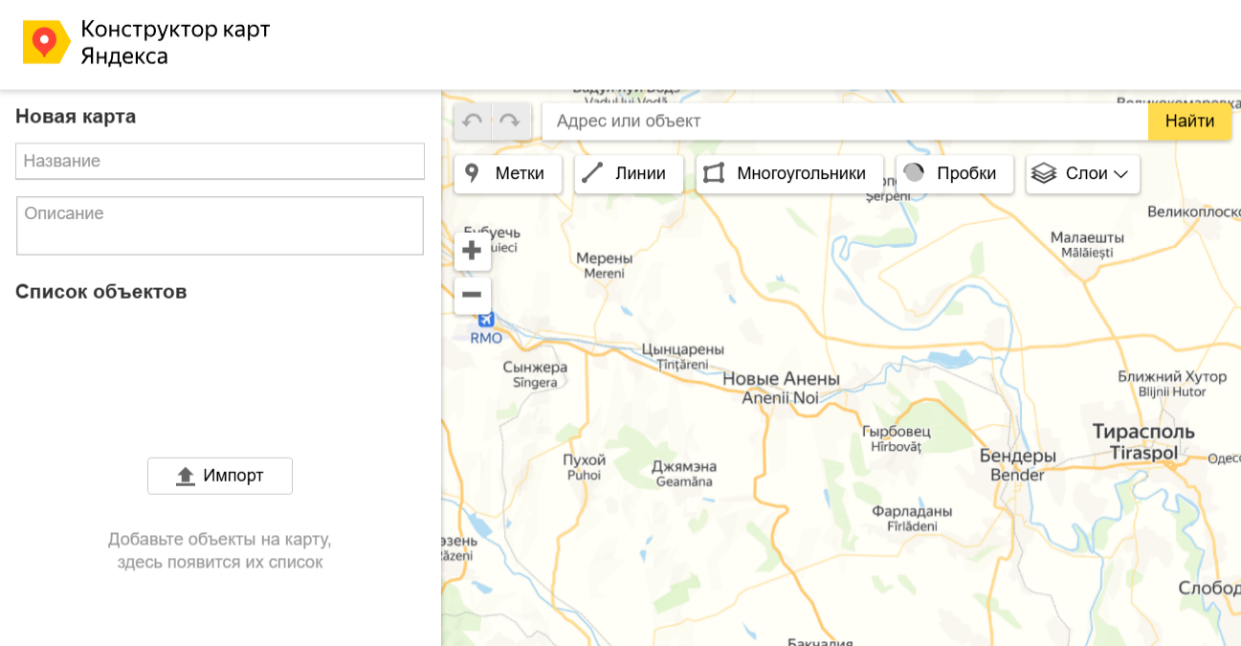


Рисунок 19 – сайт Яндекс карты

В данном сайте задам нужное местонахождение и получаем нужный нам скрипт для карты и выставляем нужные нам стили и подключаем наш скрипт.

1. После добавления нужного файла для секции «Контакты» и заполняем его.
2. Итоговый результат секции «Контакты» показан на рисунке 20.

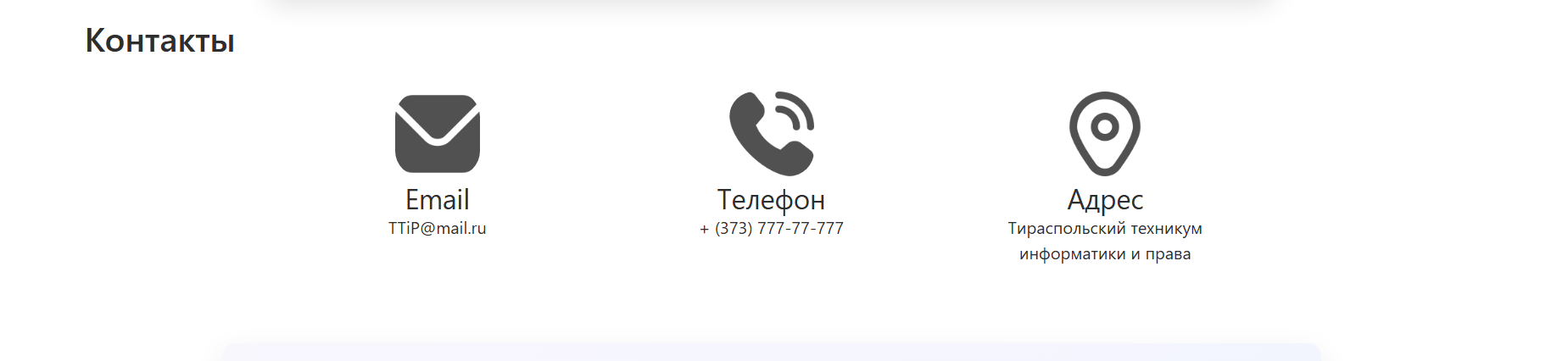


Рисунок 20 – результат секции «Контакты»

**Вывод**: в ходе выполнения работы были успешно выполнены изменения в секциях «Услуги» и «Контакты». Секция «Услуги» была оформлена с использованием карточек, улучшены стили и добавлены анимации.

**06.11.25**

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

**Ход работы:**

1. Создаем новый файл *\_LoginRegistrationPartial.cshtml* в папке *\Views\Shared*. В этом файле для начала пропишем разметку контейнера формы, а также пропишем в файле *\_Layout.cshtml  - @Html.Partial("\_LoginRegistrationPartial")*

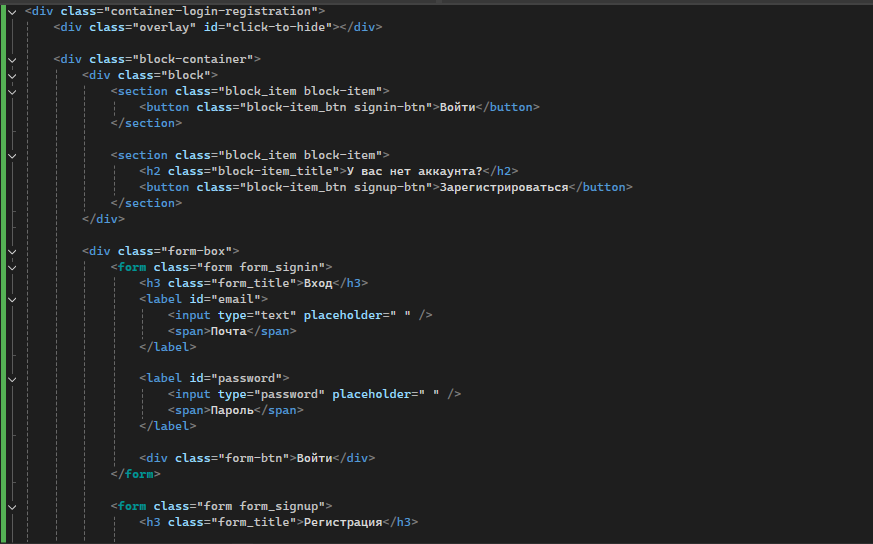


Рисунок 21– контейнер формы

1. Пропишем стили и скрипты для формы, как показано на рисунке 22 и 23.

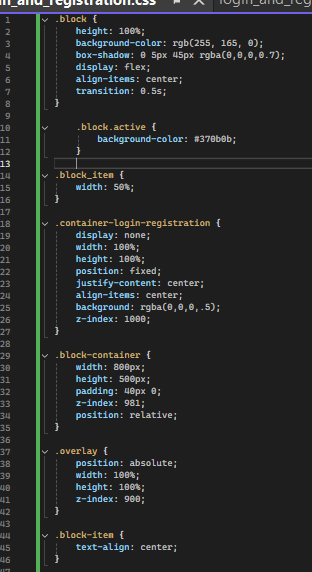


Рисунок 22 – стили для формы



Рисунок 23– скрипты для формы

1. В результате правильного подключения и составления скриптов и стилей, мы получим следующий результат, показанный на рисунке 24.

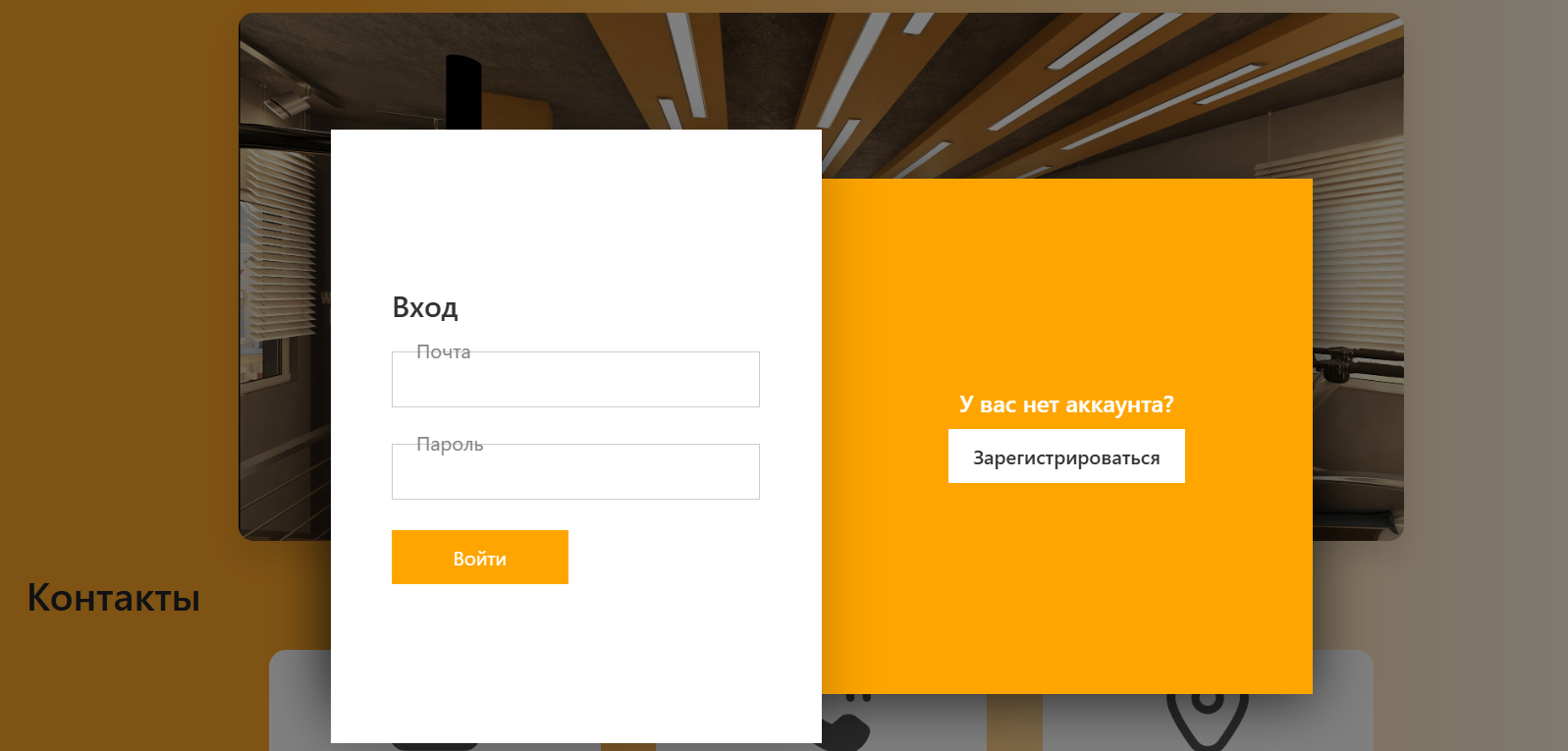


Рисунок 24 – итоговый результат

**Вывод:** в ходе работы была успешно реализована форма входа и регистрации. Были созданы соответствующие частичные представления *\_LoginRegistrationPartial.cshtml*, а также выполнена интеграция со страницей *\_Layout.cshtml*.

**07.11.25**

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

**Ход работы:**

1. В файле, где у нас расположены контакты, добавили новую панель, отвечающую за написание сообщения, данное действие показано на рисунке 25.

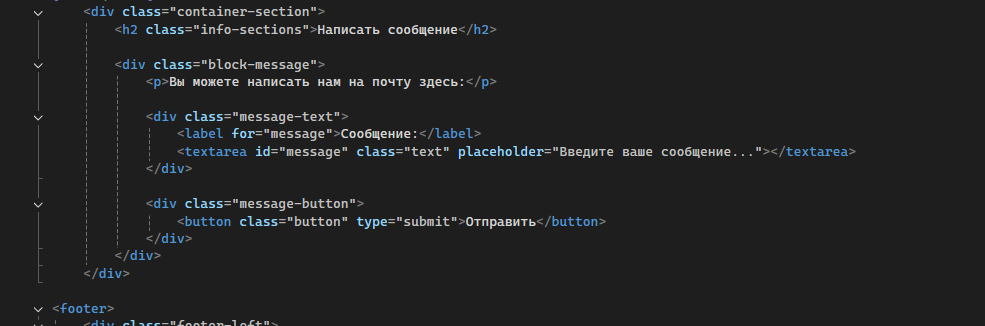


Рисунок 25 – Код «Написать сообщение» разметка

1. После написания секции, необходимо написать стили.
2. Напишем разметку для подвала сайта.
3. После проделанной работы, мы получим результат, показанный на рисунке 26.

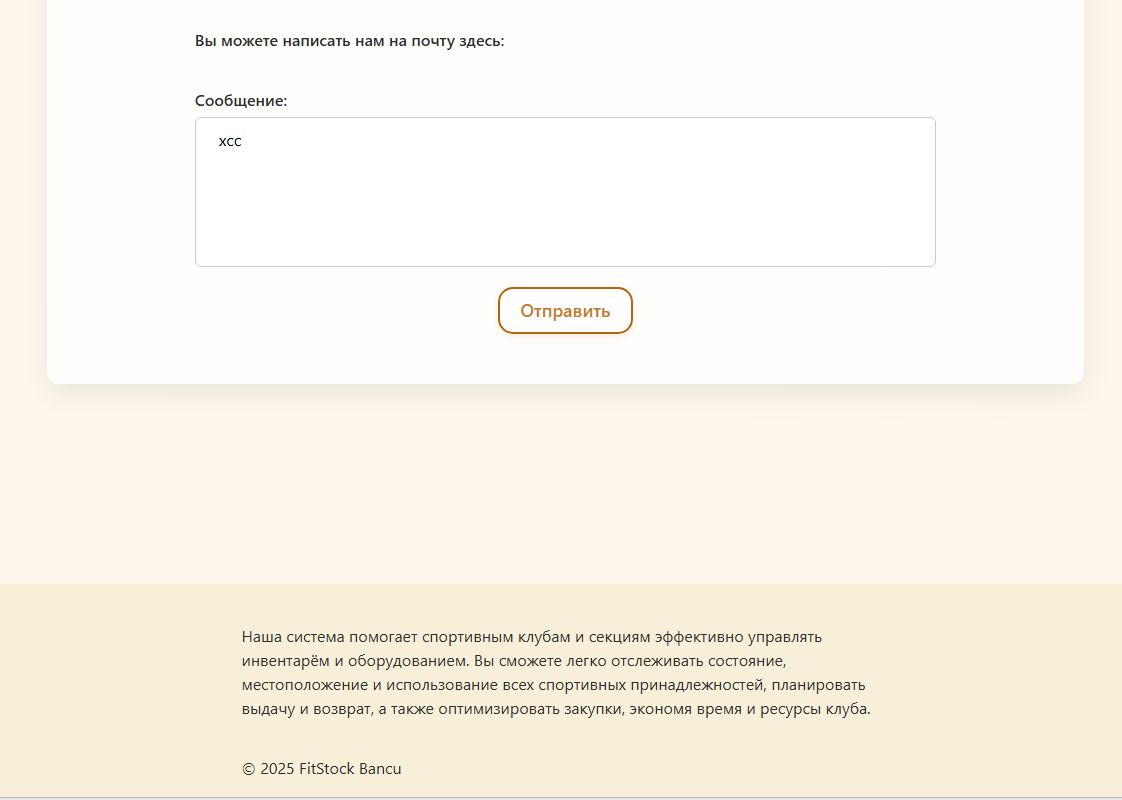


Рисунок 26 – Результат

**Вывод:** в ходе работы была успешно реализована панель «Написать сообщение». А также написание подвала сайта.

**08.11.25**

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

**Ход работы:**

*fetch* — это современный веб-API в JavaScript, предназначенный для отправки *HTTP*-запросов на сервер и получения данных от него. Он предоставляет более удобный и гибкий способ работы с асинхронными запросами по сравнению с устаревшим *XMLHttpRequest.*

1. Создаем объекты в решение *Domain.* Необходимо создать папки *ViewModels/LoginAndRegistration,* также создаем класс *LoginViewModel* который будет содержать некоторые свойства и атрибуты для проверки валидности.
2. Далее создаем класс *RegisterViewModel* аналогично предыдущему этот класс также содержит некоторые свойства и атрибуты, которые совпадают с прописанными ранее формами.
3. Чтобы наш mvc-проект получил доступ к другому проекту ему необходимо добавить ссылку. Правой кнопкой мыши нажимаем на проект и выбираем пункт *Добавить/Ссылка на проект*.
4. В контроллере добавляем методы для обработки запросов входа и регистрации. При получении запроса сервер проверяет валидность данных с помощью атрибутов и, если данные корректны, выполняет нужные операции. Данные методы показаны на рисунке 36.

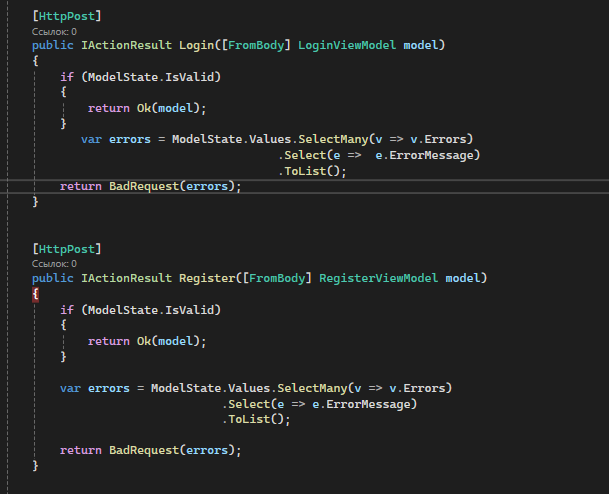


Рисунок 27– проверка валидности

1. Добавим новый скрипт для отправки данных на сервер при нажатии на кнопки формы входа и регистрации используя *fetch*, как показано на рисунке 37.
2. Добавим новый скрипт для обработки ошибок, как показано на рисунке 38.
3. Также добавим стили для ошибок, как показано на рисунке 39.

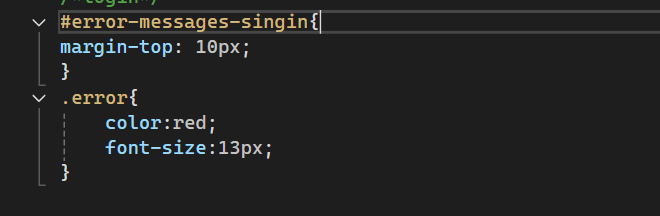


Рисунок 28 – Стили для отображения ошибок

**Вывод:** использовали *fetch* для асинхронных запросов в *JavaScript*, позволяющее эффективно отправлять данные на сервер и обрабатывать их без перезагрузки страницы. Также начали разрабатывать структуру *Domain.*

**9.11.25**

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

**Ход работы:**

1. Нужно создать файл *media\_style.css* и подключить его в *\_Layout.cshtml*.
2. Добавим стили для адаптивного интерфейса.
3. Создадим файл *chared\_script.js* и добавим следующий скрипт, отвечающий за добавления функции для открытия/закрытия бокового меню, как показано на рисунке 29.

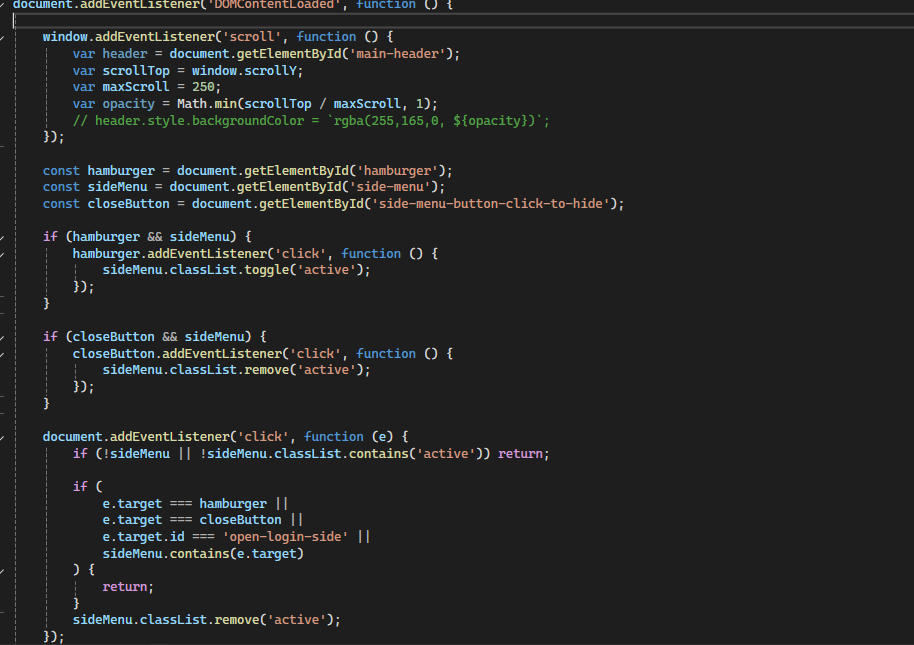


Рисунок 29 – функция появления/закрытия меню

**Вывод:** при создании адаптивного интерфейса использовали медиа-запросы, которые позволяют настроить веб-страницу под разные размеры экранов. Адаптивная вёрстка важна для того, чтобы страницы отображались корректно как на экранах смартфонов, так и на мониторах компьютеров.

**10.11.25**

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

**Ход работы:**

*PostgreSQ***L** — это свободная объектно-реляционная СУБД (система управления базами данных), распространяемая с открытым исходным кодом. Она завоевала популярность благодаря гибкости в работе с данными, стабильной поддержке транзакций и высокой надёжности. С её помощью удобно добавлять, обновлять и удалять записи, организовывать параллельный доступ из множества источников, обеспечивать отказоустойчивость и защищать информацию от потерь, а также гибко настраивать права доступа для разных пользователей.

*PostgreSQL* особенно востребована среди:

1. Backend-разработчиков, отвечающих за серверную логику приложений;
2. Администраторов БД, занимающихся их настройкой, мониторингом и оптимизацией;
3. DevOps-специалистов, поддерживающих стабильность и масштабируемость инфраструктуры.

Ключевые достоинства *PostgreSQL*:

1. Богатая типизация данных — от базовых (числа, строки) до продвинутых (JSON, массивы, пользовательские типы, поддержка NoSQL-подходов);
2. Возможность выполнения сложных, вложенных и ресурсоёмких запросов с гарантией целостности транзакций;
3. Поддержка различных операционных систем: Linux, Windows, macOS, BSD, а также развертывание в облаке;
4. Высокая производительность и масштабируемость — способна эффективно работать даже с очень крупными объёмами данных.

Ограничения и сложности:

1. Непростая первоначальная настройка — из-за широкой функциональности новичкам может потребоваться время на освоение;
2. Повышенные требования к ресурсам — в ряде сценариев потребляет больше оперативной памяти и CPU по сравнению с более лёгкими СУБД;
3. Некоторые специфические функции, доступные в коммерческих решениях, в *PostgreSQL* либо отсутствуют, либо реализованы иначе.
4. После завершения установки, мы увидим следующий интерфейс, как показано на рисунке 30.

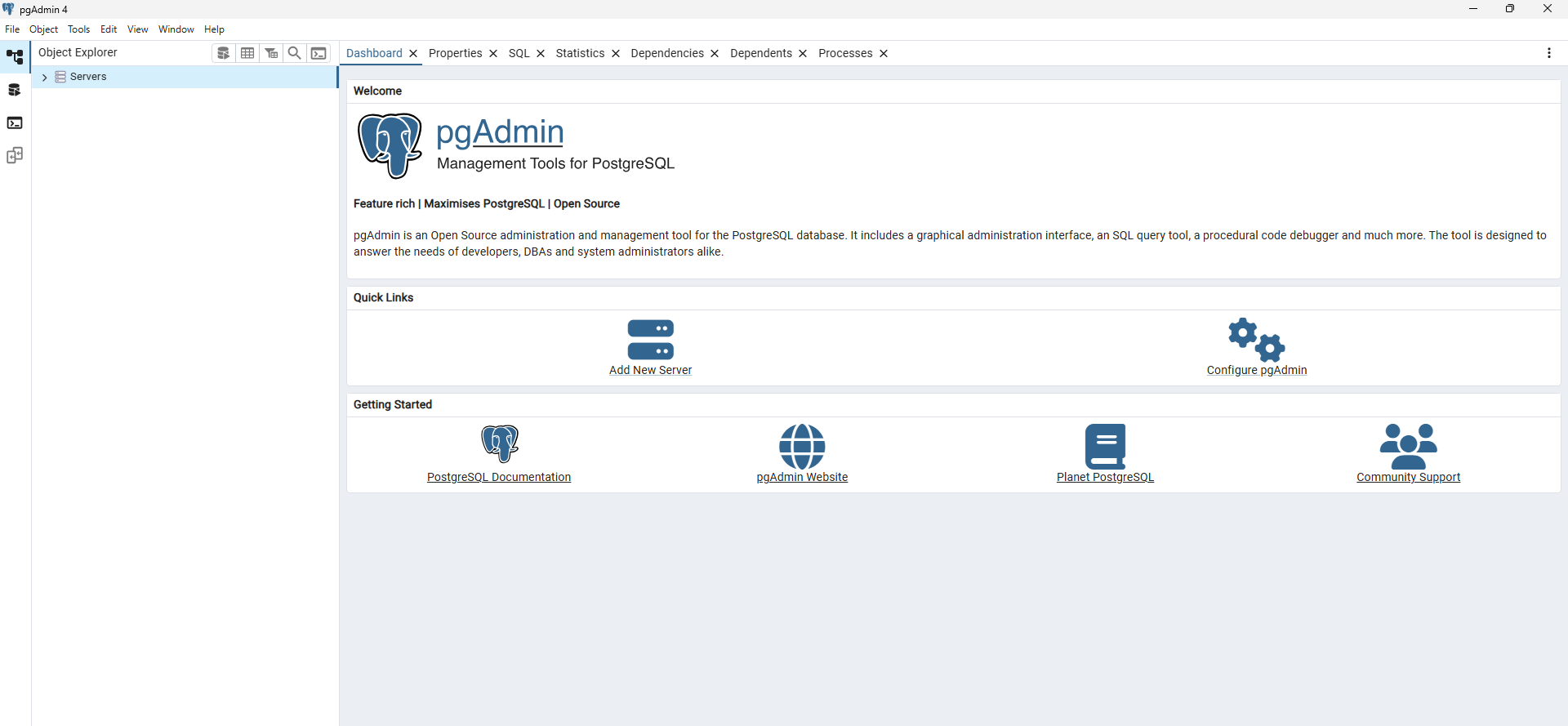


Рисунок 30 – интерфейс *PostgreSQL*

**Вывод:** познакомились с *PostgreSQL* — мощной объектно-реляционной СУБД, которая отличается высокой производительностью, гибкостью и широкой функциональностью. Благодаря множеству преимуществ, таких как поддержка сложных запросов, *PostgreSQL* стала популярным решением как для разработчиков, так и для *DevOps*-инженеров и администраторов баз данных.

**11.11.25**

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

**Ход работы:**

1. Для создания базы данных необходимо открыть *pgAdmin*
2. Необходимо создать новую базу данных, как показано на рисунке 31

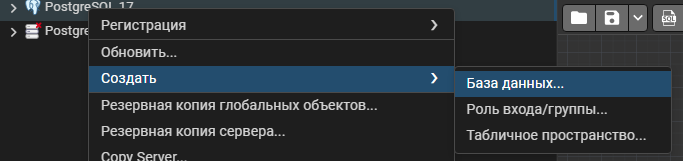


Рисунок 31 – создание базы данных

1. Далее находим пункт «*Схемы*» в созданной БД в нем нажимаем правой кнопкой мыши на «*Таблицы*» и выбираем *Создать*/*Таблица*…, как показано на рисунке.

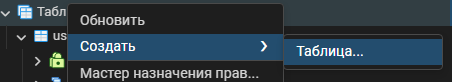


Рисунок 32 – создание таблицы

1. После создания всех таблиц, должны получиться таблицы как показано на рисунке 33.

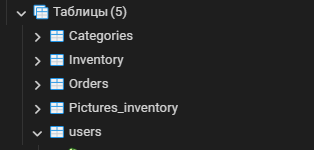


Рисунок 33 – таблицы

1. После заполнения таблиц, необходимо задать вторичные ключи для связи, как показано на рисунке 34.

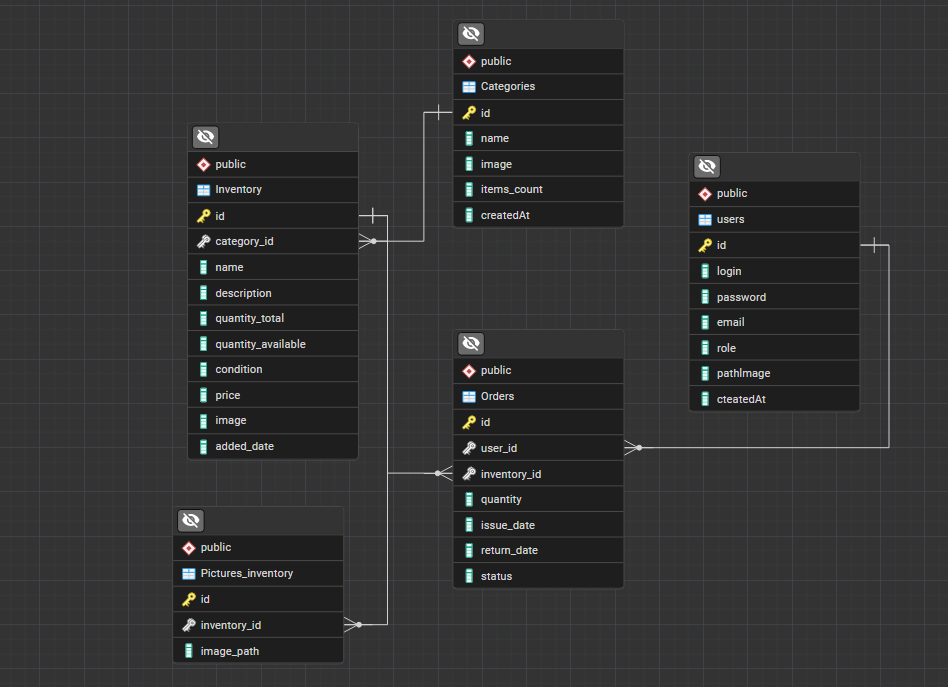


Рисунок 34 – Диаграмма БД

**Вывод:** научились создавать базу данных необходимую для проекта и пользоваться таблицами со столбцами. Познакомились с типами данных БД.

**12.11.25**

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

**Ход работы:**

1. Подключили пакет Npgsql.EntityFrameworkCore.PostgreSQL через *NuGet* пакет для работы с *PostgreSQL* в проекте *DAL*, как показано на рисунке 35.

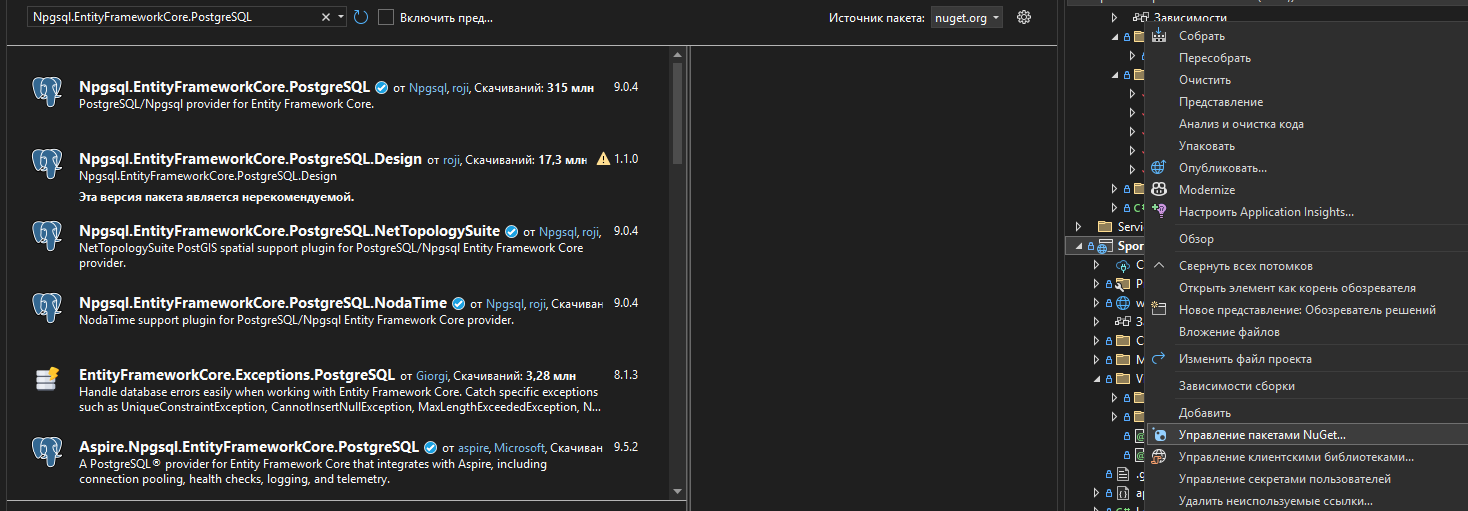


Рисунок 35 – подключение пакета *Npgsql.EntityFrameworkCore.PostgreSQL*

1. Добавили строку подключения в файл «*appsettings.json*», указав параметры: хост, имя базы данных, пользователя, пароль и порт.
2. Прописали конфигурацию подключения к базе данных в файле «*Program*.*cs*».
3. Разработали класс «*UserDb*.*cs*» с атрибутами, соответствующими столбцам таблицы пользователей, пример показан на рисунке 36.

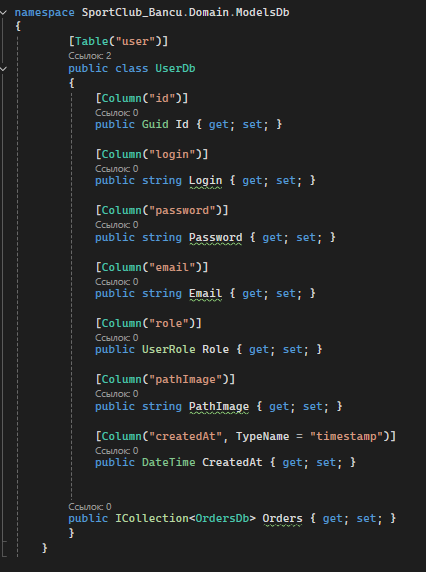


Рисунок 36 - класс «*UserDb*.*cs*»

1. Создали папку «*Enum*» и определили перечисления для ролей и статусов.
2. Создали классы для других таблиц базы данных, аналогичные «*User*.cs».
3. Добавили новый класс *ApplicationDbContext* в проект DAL для управления контекстом базы данных, как показано на рисунке 37.

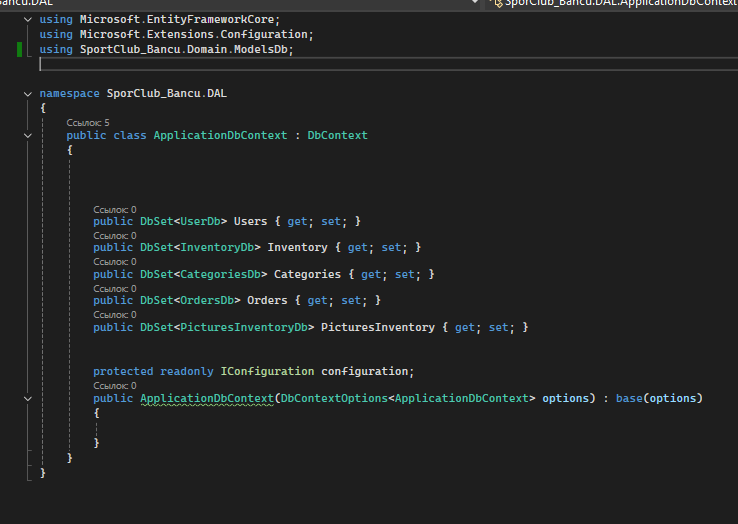


Рисунок 37 – добавление класса *ApplicationDbContext*

1. Создали миграцию с помощью команды в консоли диспетчера пакетов.
2. Применили изменения к базе данных с помощью команды «*Update*-*Database*».

**Вывод:** настроили подключение к базе данных *PostgreSQL* с использованием *Entity* *Framework* *Core* и добавлены модели данных для сущностей, таких как пользователи, лекарства, категории, заказы, записи и корзина. Определены все необходимые связи и проведена начальная миграция, чтобы обеспечить удобное взаимодействие с базой данных.

**13.11.25**

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

**Ход работы:**

*CRUD* (*Create, Read, Update, Delete*) — это базовая модель операций для управления данными в приложениях и системах.

Основные операции *CRUD*:

1. *Create* (Создание): Добавление новых записей, например, регистрация пользователя или добавление товара.
2. *Read* (Чтение): Извлечение данных, например, просмотр каталога товаров.
3. *Update (Обновление):*Изменение существующих данных, например, редактирование профиля.
4. *Delete* (Удаление): Удаление записей из базы, например, удаление аккаунта.
5. Создадим папку *Interfaces* в проекте *DAL* и в ней создадим класс *IBaseStorage<T>,* как показано на рисунке 38.

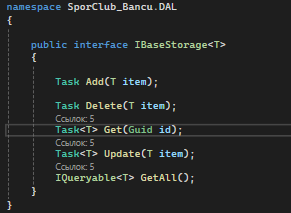


Рисунок 38 - *IBaseStorage<T>*

1. Необходимо создать классы для каждой сущности из БД, которые будут реализовывать этот интерфейс *IBaseStorage<T>.* Создайте папку и классы в соответствии с картинкой 39.

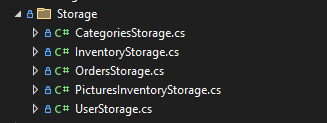


Рисунок 39 – папка с классами

1. Реализуем класс *UserStorage* и все остальные классы по этому примеру как показано на Рисунке 40.



Рисунок 40 – реализация класса *UserStorage*

1. А далее мы создадим еще одни модели. У нас уже есть пару моделей представлений несколько моделей БД теперь нужно создать модели, которые будут использоваться в сервисах. Модель должна соответствовать модели БД.

**Вывод**: сделали операции *CRUD* на наши модели, а также создали интерфейс *IBaseStorage<T>,* который используется для создания абстракции, которая описывает общее поведение без указания конкретной реализации. Это позволяет разработчику определять методы и их сигнатуры, не заботясь о деталях их реализации.

**14.11.24**

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

**Ход работы:**

1. Cоздадим папку *Response,* а в ней класс *BaseResponse*.
2. Далее добавим перечисление статусов ответов как показано на рисунке 41.

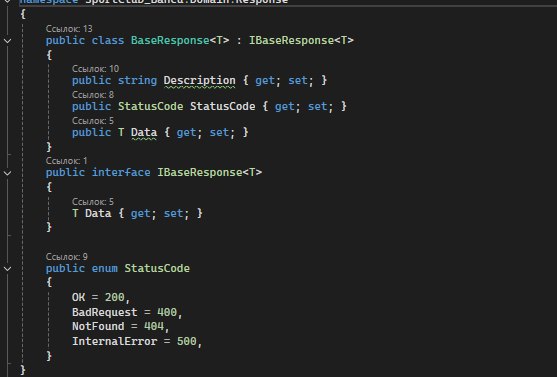


Рисунок 41 – перечисление статусов

1. Продолжим и реализуем сервис для работы с аккаунтом. Создадим в проекте Service папку с интерфейсами и реализациями этих интерфейсов, как показано на рисунке 43.

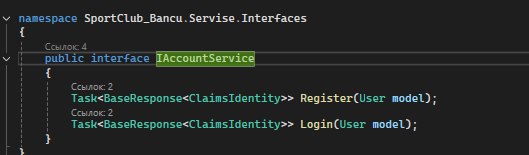


Рисунок 43 – интерфейс для работы с аккаунтом

1. После этого можно приступить к реализации этого интерфейса. Создадим класс *AccountService* в паке *Service.*  В сервисах нам нужно будет переводить из одной модели в другую нам в этом поможет библиотека *AutoMapper*. Зайдем в *Nuget* проекта *Service* найдем и закачаем нужный пакет. Теперь нужно создать класс, в котором мы объявим для каких моделей будет работать мапер, как показано на рисунке 62.

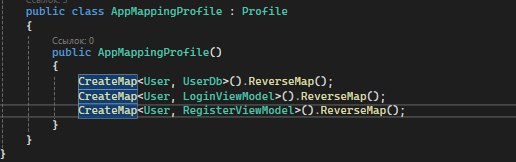


Рисунок 44 – класс мапера

**Вывод:** в процессе работы были разработаны сервисы для авторизации и аутентификации, обеспечивающие управление учетными записями пользователей. Реализована базовая архитектура.