**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Владимирский государственный университет**

**имени Александра Григорьевича и Николая Григорьевича Столетовых»**

**(ВлГУ)**

**Колледж инновационных технологий и предпринимательства**

кафедра физики и прикладной математики

**ОТЧЕТ**

**ПО УЧЕБНОЙ ПРАКТИКЕ**

**ПО МОДУЛЮ ПМ.01 «Разработка модулей программного обеспечения для компьютерных систем»**

#### студентки Казанкиной Татьяны Сергеевны

специальности 09.02.07 Информационные системы и программирование, квалификация Программист

группы ИПсп-122

Время прохождения практики

с «05» мая 2025 г.

по «25» мая 2025 г.

*Руководитель от ВлГУ*:

Преподаватель КИТП *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*М.Ю. Кабанова

Владимир, 2025

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ |
|  | Зав. кафедрой /К.С. Хорьков/ |
|  | «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г. |

З А Д А Н И Е

на *учебную практику по модулю ПМ.01 «Разработка модулей программного обеспечения для компьютерных систем»* студентки 3 курса группы **ИПсп-122**

***Казанкиной Татьяны Сергеевны***

Специальность: 09.02.07 Информационные системы и программирование (программист)

Предприятие: Кафедра ФиПМ ВлГУ г. Владимир

***Последовательность прохождение практики:***

1. получить задание;
2. пройти инструктаж по технике безопасности и охране труда;
3. выполнить все задания;
4. оформить отчет согласно требованиям.

***За время прохождения практики необходимо:***

**1)** Разработайте код программного продукта в соответствии с отраслевыми стандартами.

При открытии формы для редактирования все поля выбранного объекта должны быть подгружены в соответствующие поля из базы данных. После редактирования/добавления в БД данные в окне/форме системы должны быть обновлены.

Соблюдайте культуру кодирования.

1. *Название приложения*. Используйте соответствующие названия для ваших приложений и файлов. Так, например, наименование настольного приложения должно обязательно включать название компании- заказчика. Используйте пиктограмму компании в качестве пиктограмм всех форм.
2. *Файловая структура.* Файловая структура проекта должна отражать логику, заложенную в приложение. Например, все формы содержатся в одной директории, пользовательские визуальные компоненты – в другой, классы сущностей – в третьей.
3. *Структура проекта*. Каждая сущность должна быть представлена в программе как минимум одним отдельным классом. Классы должны быть небольшими, понятными и выполнять одну единственную функцию (Single responsibility principle). Для работы с разными сущностями используйте разные формы, где это уместно.
4. *Макет и технические характеристики*. Все компоненты системы должны иметь единый согласованный внешний вид, соответствующий руководству по стилю, а также следующим требованиям:

– разметка и дизайн (предпочтение отдается масштабируемой компоновке;

– должно присутствовать ограничение на минимальный размер окна;

– должна присутствовать возможность изменения размеров окна, где это необходимо;

– увеличение размеров окна должно увеличивать размер контентной части, например, таблицы с данными из БД);

– группировка элементов (в логические категории);

– использование соответствующих элементов управления (например, выпадающих списков для отображения подстановочных значений из базы данных);

– расположение и выравнивание элементов (метки, поля для ввода и т.д.);

– последовательный переход фокуса по элементам интерфейса (по нажатию клавиши TAB);

– общая компоновка логична, понятна и проста в использовании;

– последовательный пользовательский интерфейс, позволяющий перемещаться между существующими окнами в приложении (в том числе обратно, например, с помощью кнопки «Назад»);

– соответствующий заголовок на каждом окне приложения (не должно быть значений по умолчанию типа MainWindow, Form1 и т.п.).

1. *Руководство по стилю*. Все экранные формы пользовательского интерфейса должны иметь заголовок с логотипом (для форм – квадратный вариант, для отчетов – прямоугольный вариант) и название (за исключением простых диалогов).

Все визуальные компоненты должны быть выровнены, доступны, иметь соизмеримый масштаб и не оставлять много свободного пространства. Не допускайте орфографические и грамматические ошибки.

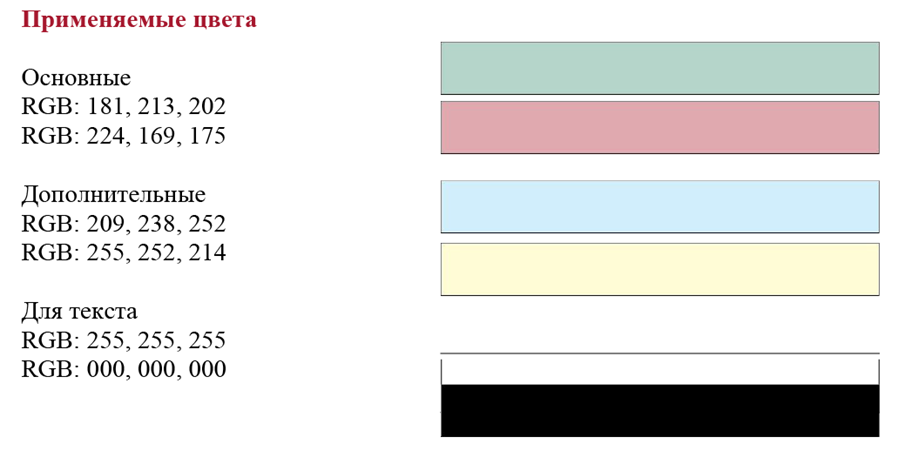
В таблицах рекомендуется применять дополнительные цвета для разного оформления четных и нечетных строк.

Для надписей используйте черный цвет, или белый (в том случае, если фон – темный); начертание: обычный, курсив или полужирный; размер: 11 – 18 пт.

Все элементы интерфейса должны быть логически сгруппированы вместе, чтобы система была более простой в использовании.

Во всей системе должны применяться одинаковые макетные решения.

К внешнему виду относится любой визуальный элемент, с которым контактирует конечный пользователь, включая экранные формы, отчеты, надписи и т.д.



1. *Обратная связь с пользователем*. Уведомляйте пользователя о совершаемых им ошибках или о запрещенных в рамках задания действиях, запрашивайте подтверждение перед удалением, предупреждайте о неотвратимых операциях, информируйте об отсутствии результатов поиска и т.п. Окна сообщений соответствующих типов (например, ошибка, предупреждение, информация) должны отображаться с соответствующим заголовком и пиктограммой. Текст сообщения должен быть полезным и информативным, содержать полную информацию о совершенных ошибках пользователя и порядок действий для их исправления. Также можно использовать визуальные подсказки для пользователя при вводе данных.
2. *Обработка ошибок*. Не позволяйте пользователю вводить некорректные значения в текстовые поля сущностей. Например, в случае несоответствия типа данных или размера поля введенному значению. Оповестите пользователя о совершенной им ошибке. При возникновении непредвиденной ошибки приложение не должно аварийно завершать работу.

Запрещено удаление сущностей, которое приведет к нарушению ограничений связей. При удалении любых сущностей необходимо либо спрашивать подтверждение пользователя, либо реализовать возможность отмены операции удаления.

1. *Оформление кода*. Идентификаторы переменных, методов и классов должны отражать суть и/или цель их использования, в том числе и наименования элементов управления (например, не должно быть значений по умолчанию типа Form1, button3). Идентификаторы должны соответствовать соглашению об именовании (Code Convention) и стилю CamelCase (для C# и Java) и snake\_case (для Python). Допустимо использование не более одной команды в строке. Максимальная длина строки 80 символов. Отступ составляет четыре пробела (без tab).

*Реализуйте переиспользуемые визуальные компоненты*. Не дублируйте логику – это отнимет много времени. Например, понадобится текстовое поле с валидацией целочисленного значения много более чем в одном месте. Другой пример, с каждой сущностью нужно осуществлять четыре одинаковых действия: чтение, обновление, создание и удаление. Где уместно используйте наследование и полиморфизм.

1. *Комментарии*. Используйте комментарии для пояснения неочевидных фрагментов кода. Запрещено комментирование кода. Хороший код воспринимается как обычный текст. Не используйте комментарии для пояснения очевидных действий. Комментарии должны присутствовать только в местах, которые требуют дополнительного пояснения. Используйте тип комментариев, который в дальнейшем позволит сгенерировать XML- документацию, с соответствующими тегами (например, param, return(s), summary и др.)
2. Создайте отчёт в формате **.xls**.

**2)** Определить наборы входных данных и выполнить функциональное тестирование модуля по определенному сценарию. Провести тестирование для проверки функциональности программы (хотя бы один тест на одну функцию). Использовать инструментальные средства для тестирования. Представить результаты тестирования в виде протокола тестирования, в соответствии со стандартами.

*Создайте форму для авторизации зарегистрированных пользователей*. Пользователи будут входить в систему, используя эту форму. Они будут вводить свой логин и пароль. При попытке входа введенные данные проверяются на совпадение в базе данных. Должно выдаваться сообщение об ошибке в случае неправильного ввода связки логин/пароль. После успешного входа пользователя в зависимости от их роли происходит перенаправление в соответствующие разделы. Когда пользователь нажимает на кнопку «Выход» в любом окне/форме системы, пользователи должны быть отключены от системы и совершен переход на *Форму авторизации*.

*Необходимо предусмотреть возможность регистрации*. При регистрации пароль должен отвечать следующим требованиям:

– Минимум 6 символов

– Минимум 1 прописная буква

– Минимум 1 цифра

– Минимум один символ из набора: ! @ # $ % ^.

**3)** Результаты работы предоставить в системе контроля версий.

**4)** Задание по стандартизации: оформить отчет по результатам практики согласно требованиям, к оформлению документации.

**5)** Задание по охране труда, технике безопасности и охране окружающей среды: изучить правила техники безопасности при работе за компьютером, вопросы охраны труда на месте прохождения практики.

Отчет по практике составить к **23.05.2025**

Задание выдала: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.Ю. Кабанова

Задание получила: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Т.С. Казанкина

05.05.2025

СОДЕРЖАНИЕ

[введение 7](#_Toc198795617)

[1 разработка приложения 8](#_Toc198795618)

[1.1 Описание предметной области 8](#_Toc198795619)

[2 подключение к базе данных 13](#_Toc198795620)

[3 разработка приложения 19](#_Toc198795621)

[3.1 Разработка пользовательского интерфейса 19](#_Toc198795622)

[3.2 Файловая структура 26](#_Toc198795623)

[3.3 Структура проекта. 27](#_Toc198795624)

[3.4 Оформление кода 29](#_Toc198795625)

[3.5 Создание отчета в формате .xls 34](#_Toc198795626)

[4 система контроля версий 38](#_Toc198795627)

[5 тестирование 43](#_Toc198795628)

[5.1 Тест-требования 43](#_Toc198795629)

[5.2 Тест-план 44](#_Toc198795630)

[5.3 Результаты тестирования 48](#_Toc198795631)

[заключение 60](#_Toc198795632)

[список использованных источников 61](#_Toc198795633)

[приложение А 62](#_Toc198795634)

[приложение Б 77](#_Toc198795635)

введение

Учебная практика УП.01.01 по модулю ПМ.01 «Разработка модулей программного обеспечения для компьютерных систем» проходит с 05 мая 2025 года по 25 мая 2025 года.

Целью практики является разработка кода программного продукта в соответствии с отраслевыми стандартами. Процесс должен включать в себя описание предметной области, подключение к базе данных (далее БД), разработку пользовательского интерфейса и структуры проекта, тестирование готового продукта и создание для него документации.

Приложение предусматривает возможность регистрации и авторизации и реализует интерфейс в соответствии с руководством по стилю.

1. разработка приложения

1.1 Описание предметной области

Темой учебной практики является разработка приложения для управления арендой недвижимости.

Для БД «Аренда недвижимости» был выявлен следующий набор сущностей с атрибутами:

* Пользователь: фамилия, имя, отчество, адрес электронной почты, номер телефона.
* Аренда недвижимости: начало аренды, конец аренды, стоимость за месяц, залог.
* Объект недвижимости: тип объекта, район, город, улица, дом, квартира, количество комнат, площадь, минимальный срок аренды, проживание с детьми, проживание с животными.
* Собственник: фамилия, имя, отчество, номер телефона, город, улица, дом, квартира.
* Агентство недвижимости: название, номер телефона, фамилия владельца, имя владельца, отчество владельца, город, улица, дом, квартира, комиссия.
* Риелтор: фамилия, имя, отчество, номер телефона, адрес электронной почты.

Между сущностями организованы связи типа «один-ко-многим» и так же связь «многие-ко-многим» через ассоциативные таблицы.

Для наглядного представления предметной области была создана ER-диаграмма в нотации Мартина. Диаграмма в нотации Мартина представлена на рисунке 1.

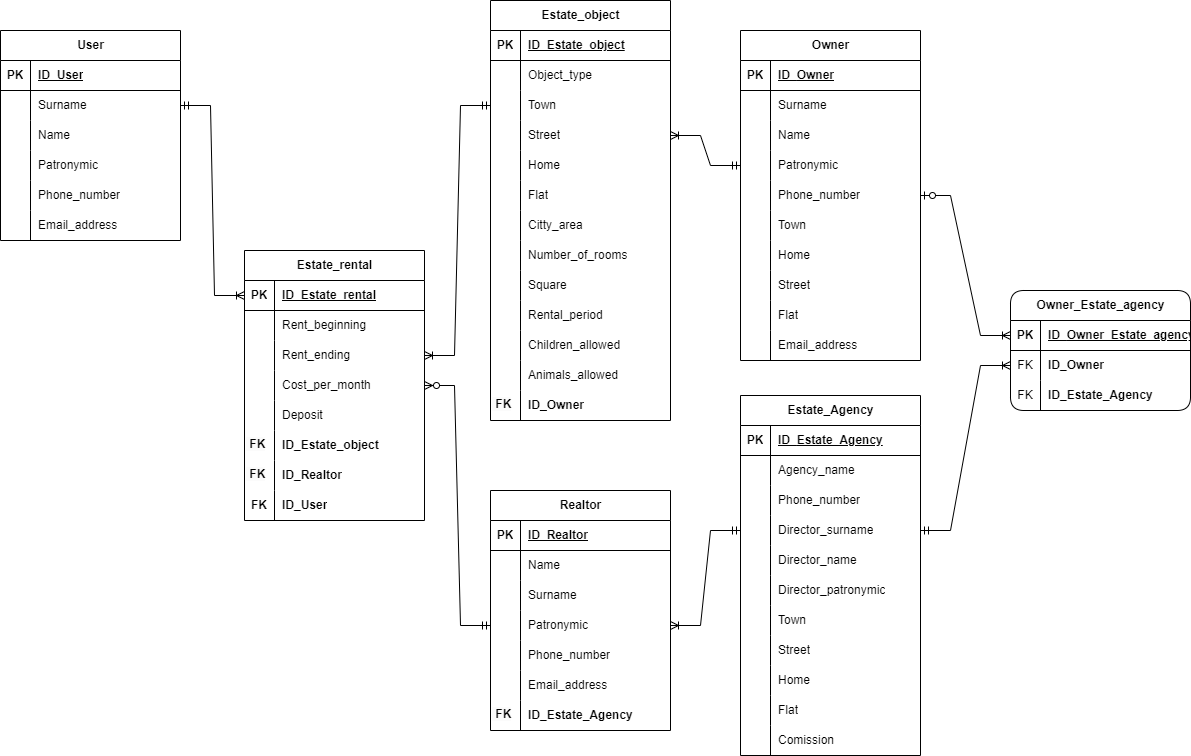


Рисунок 1

Словарь данных, отражающий структуру и содержимое базы данных представлен в таблице 1.

Таблица 1 – Словарь данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | | | | |
| Ключ | Поле | Тип | Обязательное (да/нет) | Примечание |
| PK | ID\_User | INT | Да | Первичный ключ таблицы User |
|  | Surname | NVARCHAR(30) | Да | Фамилия |
|  | Name | NVARCHAR(30) | Да | Имя |
|  | Patronymic | NVARCHAR(30) | Нет | Отчество |
|  | Phone\_number | CHAR(11) | Да | Номер телефона |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Email\_address | VARCHAR(30) | Да | Адрес электронной почты |
| Estate\_rental | | | | |
| PK | ID\_Estate\_rental | INT | Да | Первичный ключ таблицы Estate\_rental |
|  | Rent\_beginning | DATE | Да | Начало аренды |
|  | Rent\_ending | DATE | Нет | Конец аренды |
|  | Cost\_per\_month | DECIMAL(10,2) | Да | Плата за месяц |
|  | Deposit | DECIMAL(10,2) | Да | Залог |
| FK | ID\_Estate\_object | INT | Да | Внешний ключ на таблицу Estate\_object |
| FK | ID\_Realtor | INT | Нет | Внешний ключ на таблицу Realtor |
| FK | ID\_User | INT | Да | Внешний ключ на таблицу User |
| Realtor | | | | |
| PK | ID\_Realtor | INT | Да | Первичный ключ таблицы Realtor |
|  | Surname | NVARCHAR(30) | Да | Фамилия |
|  | Name | NVARCHAR(30) | Да | Имя |
|  | Patronymic | NVARCHAR(30) | Нет | Отчество |
|  | Phone\_number | CHAR(11) | Да | Номер телефона |
|  | Email-address | VARCHAR(30) | Да | Адрес электронной почты |
| FK | ID\_Estate\_agency | INT | Да | Внешний ключ на таблицу Estate\_agency |
| Estate\_agency | | | | |
| PK | ID\_Estate\_agency | INT | Да | Первичный ключ таблицы Estate\_agency |
|  | Agency\_name | NVARCHAR(30) | Да | Название агентства |
|  | Phone\_number | CHAR(11) | Да | Номер телефона |

Продолжение таблицы 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Director\_surname | | | NVARCHAR(30) | Да | | Фамилия директора |
|  | Director\_name | | | NVARCHAR(30) | Да | | Имя директора |
|  | Director\_patronymic | | | NVARCHAR(30) | Нет | | Отчество директора |
|  | Town | | | NVARCHAR(30) | Да | | Город |
|  | Street | | | NVARCHAR(30) | Да | | Улица |
|  | Home | | | NVARCHAR(30) | Да | | Дом |
|  | Flat | | | INT | Да | | Квартира |
|  | Comission | | | DECIMAL(2,1) | Да | | Комиссия |
| Owner | | | | | | | |
| PK | ID\_ Owner | | | INT | Да | | Первичный ключ таблицы Owner |
|  | Surname | | | NVARCHAR(30) | Да | | Фамилия |
|  | Name | | | NVARCHAR(30) | Да | | Имя |
|  | Patronymic | | | NVARCHAR(30) | Нет | | Отчество |
|  | Phone\_number | | | CHAR(11) | Да | | Номер телефона |
|  | Town | | | NVARCHAR(30) | Да | | Город |
|  | Street | | | NVARCHAR(30) | Да | | Улица |
|  | Home | | | NVARCHAR(30) | Да | | Дом |
|  | Flat | | | INT | Да | | Квартира |
|  | Email\_address | | | VARCHAR(30) | Да | | Адрес электронной почты |
| Estate\_object | | | | | | | |
| PK | | ID\_ Estate\_object | INT | | Да | Первичный ключ таблицы Estate\_object | |
|  | | Object\_type | NVARCHAR(20) | | Да | Тип объекта (дом, квартира, офим) | |
|  | | Town | NVARCHAR(30) | | Да | Город | |
|  | | Street | NVARCHAR(30) | | Да | Улица | |
|  | | Home | NVARCHAR(30) | | Да | Дом | |
|  | | Flat | INT | | Нет | Квартира | |
|  | | City\_area | NVARCHAR(30) | | Да | Район города | |
|  | | Number\_of\_rooms | INT | | Да | Количество комнат | |
|  | | Square | DECIMAL(10,2) | | Да | Площадь | |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Rental\_period | INT | Да | Минимальный срок аренды (в дянх) |
| FK | ID\_Owner | INT | Да | Внешний ключ на таблицу Owner |
|  | Children\_allowed | BIT | Да | Дети разрешены |
|  | Animals\_allowed | BIT | Да | Животные разрешены |
| Owner\_Estate\_agency | | | | |
| PK | ID\_ Owner\_Estate\_agency | INT | Да | Первичный ключ таблицы Owner\_Estate\_agency |
|  | ID\_Owner | INT | Да | Внешний ключ на таблицу Owner |
|  | ID\_Estate\_agency | INT | Нет | Внешний ключ на таблицу Estate\_agency |

Доступ к таблицам будет осуществляться в соответствии с ролями пользователей.

2 подключение к базе данных

Для подключения БД к проекту необходимо в верхнем меню перейти в раздел «Средства» и выбрать пункт «Подключение к базе данных». Выпадающий список «Средства» представлен на рисунке 2.

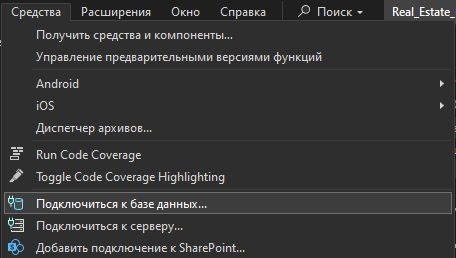


Рисунок 2 – Выпадающий список «Средства»

Откроется окно «Добавить подключение», в которое нужно ввести имя сервера и выбрать базу данных из списка. Окно «Добавить подключение» представлено на рисунке 3.

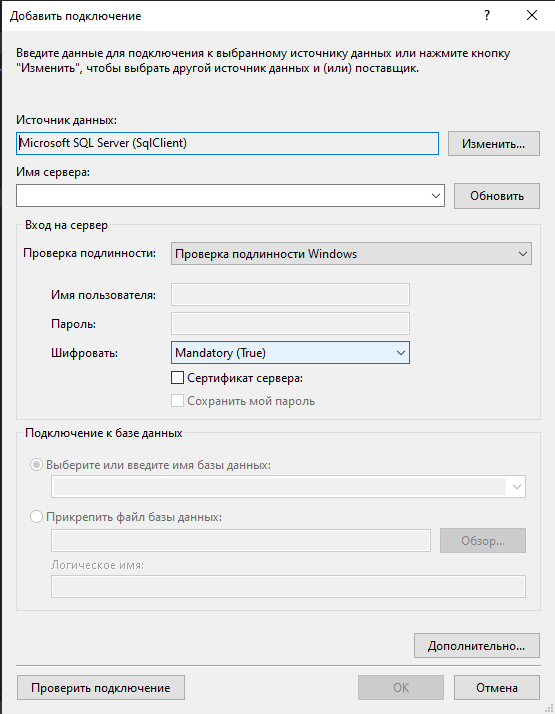


Рисунок 3 – Окно «Добавить подключение»

Для того, чтобы получить имя сервера, нужно открыть Microsoft SQL Server Manager Studio. При входе в приложение открывается окно «Соединение с сервером». Необходимо скопировать строчку «Имя сервера». Окно «Соединение с сервером» представлено на рисунке 4.

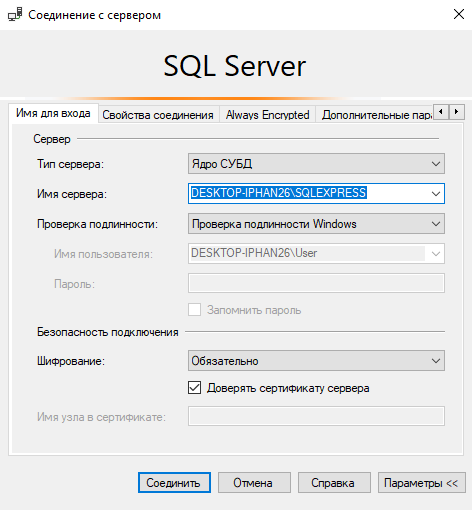


Рисунок 4 – Окно «Соединение с сервером»

Скопированную строчку нужно вставить в поле «Имя сервера» в окне «Добавить подключение» и выбрать БД из раскрывающегося списка. Окно «Добавить подключение» с выбранной БД представлено на рисунке 5.

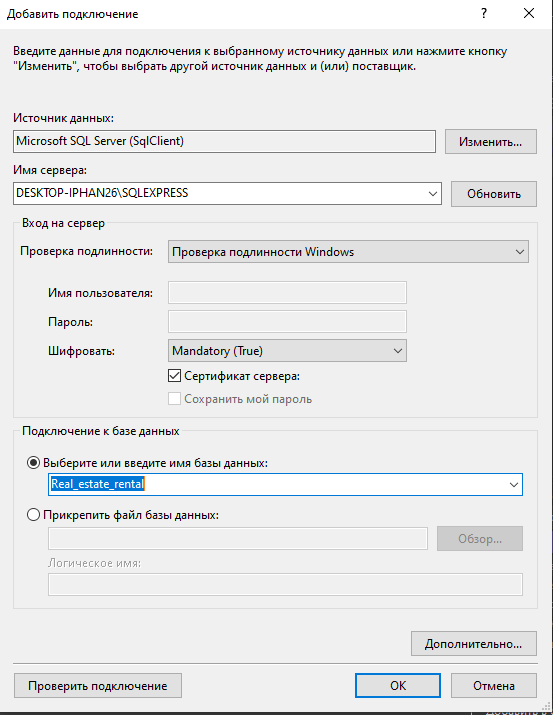


Рисунок 5 – Окно «Добавить подключение» с выбранной БД

Важно перейти в раздел «Дополнительно» и в открывшемся окне «Расширенные свойства» указать «Trust Server Certificate=true». Нажимаем «ОК». Окно «Расширенные свойства» представлено на рисунке 6.

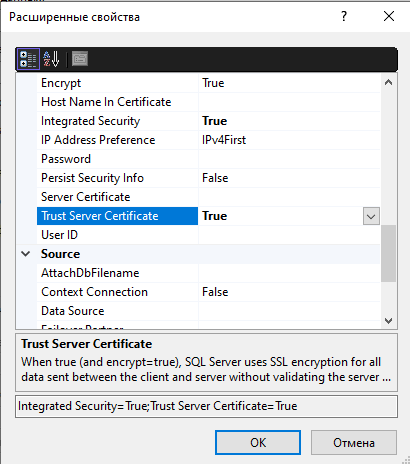


Рисунок 6 – Окно «Расширенные свойства»

Далее в окне «Добавить подключение нужно нажать кнопку «ОК». Выбранная БД отобразится в активных подключениях. Активные подключения представлены на рисунке 7.

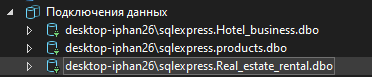


Рисунок 7 – Активные подключения

Для автоматического создания классов для каждой сущности нужно скопировать строку подключения и вставить в соответствующую команду. Для этого выбираем необходимое подключение, нажимаем правой кнопкой мыши и выбираем в раскрывшемся меню пункт «Свойства». Раскрывающееся меню представлено на рисунке 8.

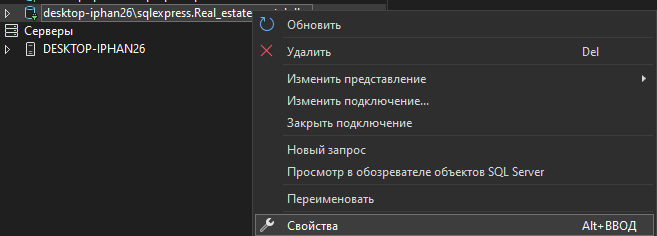


Рисунок 8 – Раскрывающееся меню

В окне «Свойства» находим пункт «Строка подключения» и копируем из него значение. Окно «Свойства» представлено на рисунке 9.

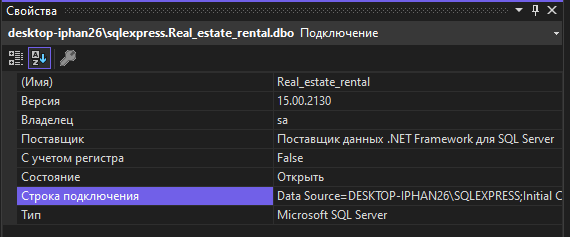


Рисунок 9 – Окно «Свойства»

Далее нужно установить необходимые для подключения пакеты. Команды для установки представлены в листинге 2.1.

Листинг 2.1 – Команды для установки пакетов

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Tools

Install-Package Microsoft.EntityFrameworkCore.Design

Команда для подключения к БД и автоматического создания классов моделей представлена в листинге 2.2.

Листинг 2.2 – Команда для подключения к БД

Scaffold-DbContext "Data Source=DESKTOP-IPHAN26\SQLEXPRESS;Initial Catalog=Real\_estate\_rental;Integrated Security=True;Trust Server Certificate=True" Microsoft.EntityFrameworkCore.SqlServer

Результат выполнения команды представлен на рисунке 10.

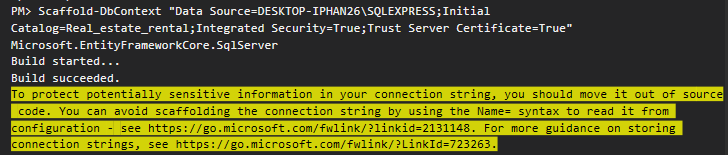


Рисунок 10 – Результат выполнения команды

Автоматически созданные классы и контекст подключения представлены на рисунке 11.

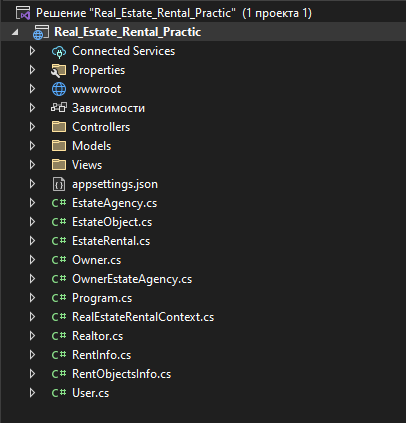


Рисунок 11 – Созданные классы и контекст подключения

Таким образом было осуществлено подключение к разработанной БД и автоматически созданы все необходимые для работы классы.

3 разработка приложения

3.1 Разработка пользовательского интерфейса

При запуске приложения для пользователя отображается страница «Вход».

Страница «Вход» представлена на рисунке 12.



Рисунок – Страница «Вход»

На странице «Вход» присутствуют поля для ввода электронной почты и пароля. При вводе правильной пары пароля и почты осуществляется вход и перенаправление на главную страницу «Недвижимость». Также для пользователя становится доступно боковое меню.

Верхнее меню состоит из пунктов «Войти» и «Регистрация». Если пользователь не был зарегистрирован, с помощью меню он переходит на страницу «Регистрация». Страница «Регистрация» представлена на рисунке 13.

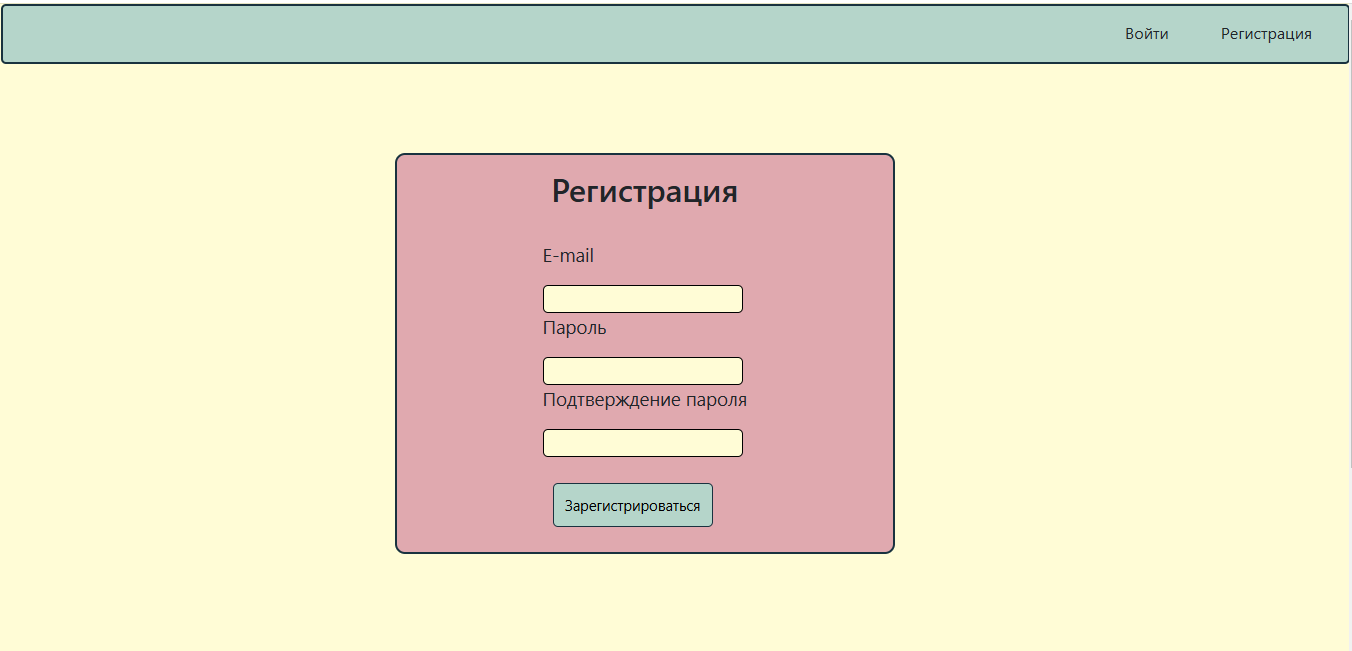


Рисунок – Страница «Регистрация»

На странице «Регистрация» присутствуют поля для ввода электронной почты, пароля и подтверждения пароля. При вводе валидных данных и соблюдения политики пароля происходит создание нового пользователя и автоматически происходит вход в его учетную запись. Происходит перенаправление на главную страницу «Недвижимость» и становится доступным боковое меню. После авторизации кнопки «Регистрация» и «Вход» в верхнем меню меняются на кнопку «Выйти», по нажатию на которую происходит выход из текущей учетной записи. Страница «Недвижимость» представлена на рисунке 14.



Рисунок – Страница «Недвижимость»

Для роли администратор боковое меню содержит пункты «Недвижимость», «Агентства», «Аренда», «Арендодатели», «Риелторы» и «Арендаторы». Также доступны все кнопки управления объектами на страницах.

Страница «Недвижимость» содержит кнопку «Добавить», по нажатию на которую открывается форма добавления нового объекта. Форма добавления представлена на рисунке 15.

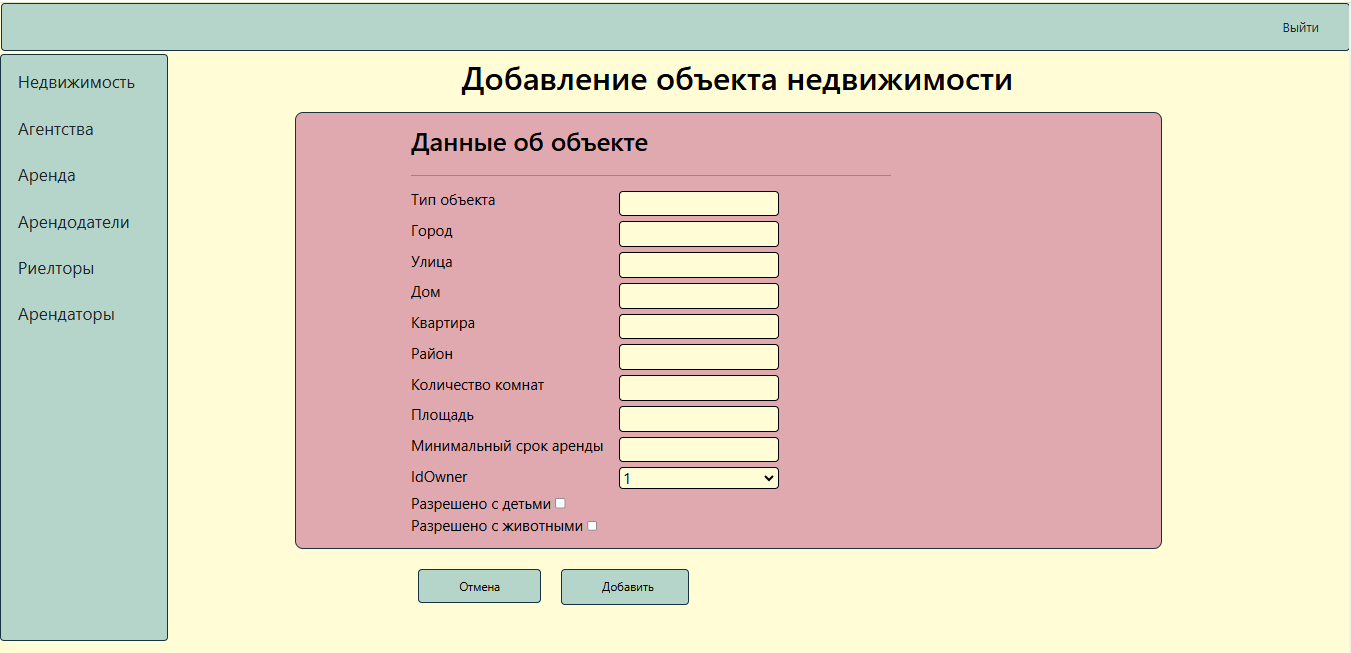


Рисунок – Форма добавления

Страница содержит поля для ввода типа объекта, города, улицы, дома, квартиры, района, количества комнат, площади и минимального срока аренды, а также позволяет выбрать IdOwner из выпадающего списка и отметить разрешение на заселение с животными и детьми с помощью чек боксов.

При нажатии на кнопку «Отмена» происходит возвращение на страницу «Недвижимость».

При вводе валидных данных в формы и нажатии на кнопку «Добавить», в таблицу добавляется новый элемент с указанными значениями и происходит возврат на страницу «Недвижимость», где в конце списка отобразится добавленный элемент.

По нажатию на карточку в списке недвижимости происходит переход на страницу изменения объекта недвижимости. Страница изменения объекта недвижимости представлена на рисунке 16.

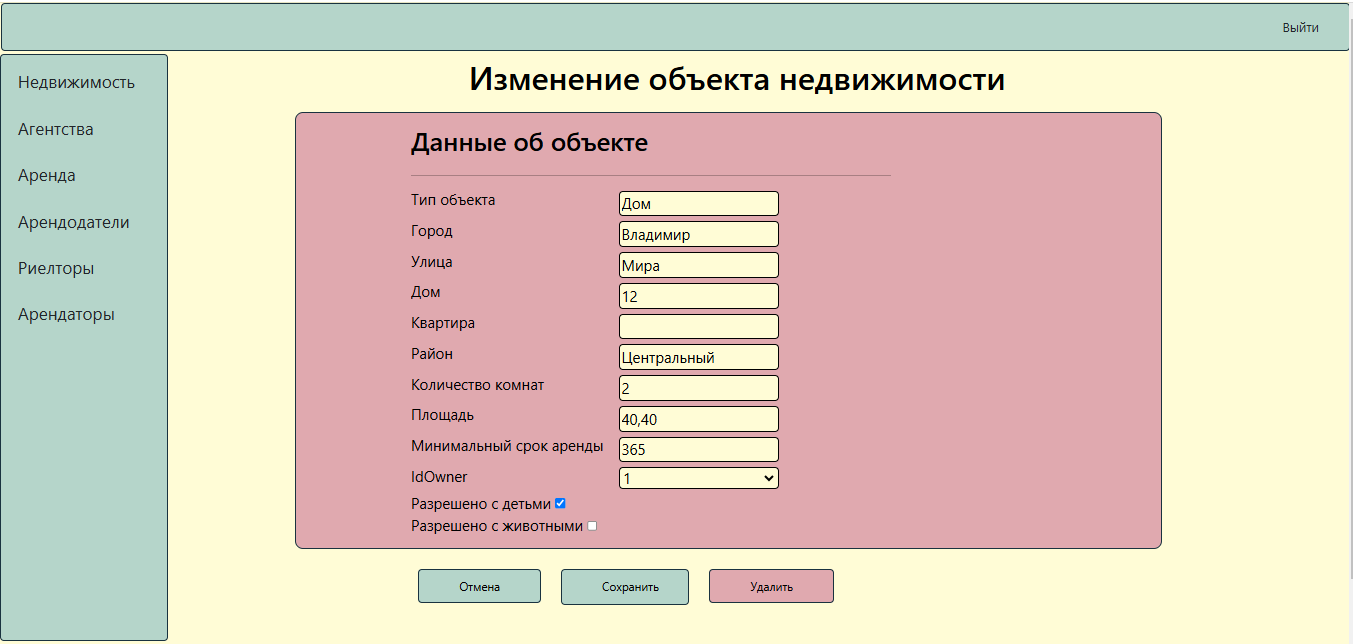


Рисунок – Страница изменения объекта недвижимости

На данной странице отображаются поля, аналогичные форме добавления. В полях ввода отображается вся текущая информация об объекте.

На странице имеется кнопка «Отмена», по нажатию на которую происходит возврат на страницу «Недвижимость» и кнопка «Сохранить», при нажатии на которую после ввода валидных данных происходит изменение данных о текущем объекте и возврат на страницу «Недвижимость».

Кнопка «Удалить» перенаправляет пользователя на страницу подтверждения удаления. Страница подтверждения удаления представлена на рисунке 17.

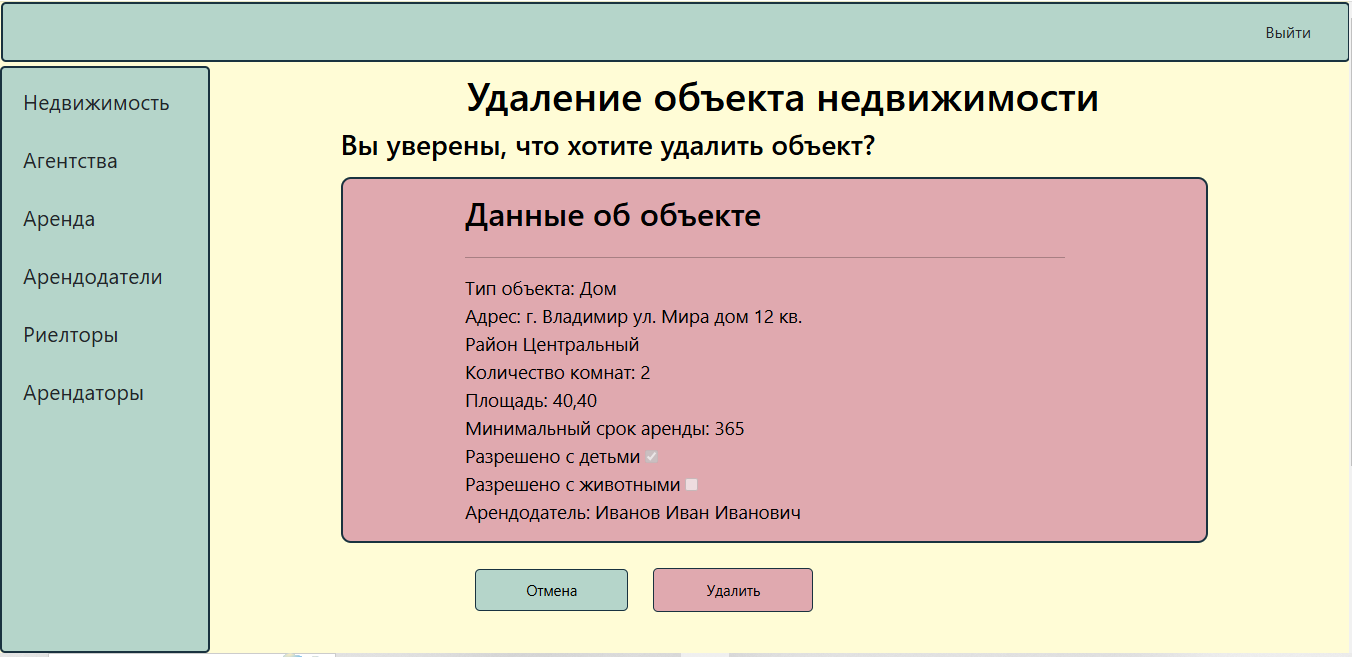


Рисунок – Страница подтверждения удаления

Данная страница содержит информацию об объекте, вопрос о подтверждении удаления, кнопку «Отмена», по нажатию на которую происходит возврат на страницу изменения объекта и кнопку «Удалить», по нажатию на которую происходит удаление данного объекта из базы данных и возврат на страницу «Недвижимость».

Остальные страницы имеют аналогичный интерфейс и набор кнопок управления. Страницы «Агентства» и «Арендодатели» также имеют кнопку «Экспорт». Отображение кнопки «Экспорт» на странице «Арендодатели» представлено на рисунке 18.

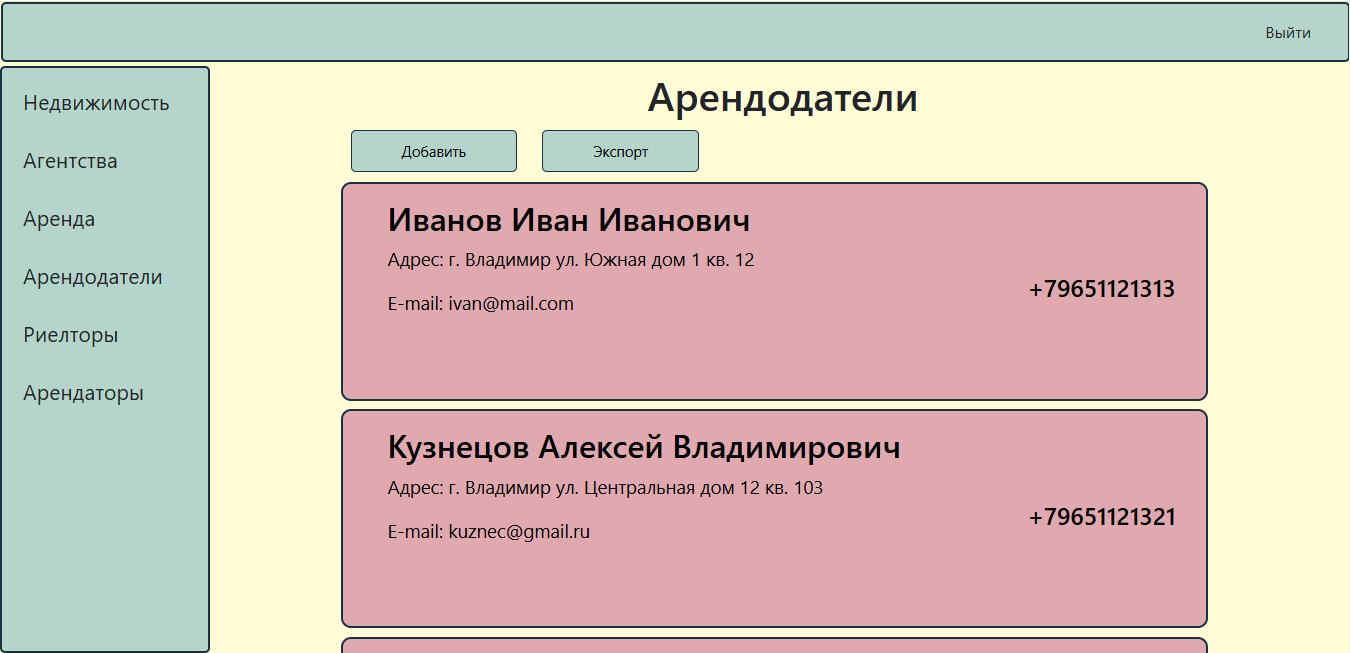


Рисунок – Отображение кнопки «Экспорт»

При нажатии на кнопку «Экспорт» генерируется отчет в формате Excel. После завершения генерации отчета файл отображается в загрузках браузера, свидетельствуя о завершении операции.

Для разных ролей боковое меню имеет разное количество доступных пунктов. Боковое меню и главная страница для роли пользователь представлены на рисунке 19.



Рисунок – Боковое меню и главная страница для роли пользователь

Пользователь имеет только три пункта меню – «Недвижимость», «Агентства» и «Риелторы». Кнопки управления полностью отсутствуют. При нажатии на карточку объекта переход на страницу редактирования не осуществляется.

На странице «Агентства» доступна кнопка «Экспорт», по нажатию на которую выполняются все ранее описанные действия для этой кнопки. Страница «Агентства» для роли «Пользователь» представлена на рисунке 20.

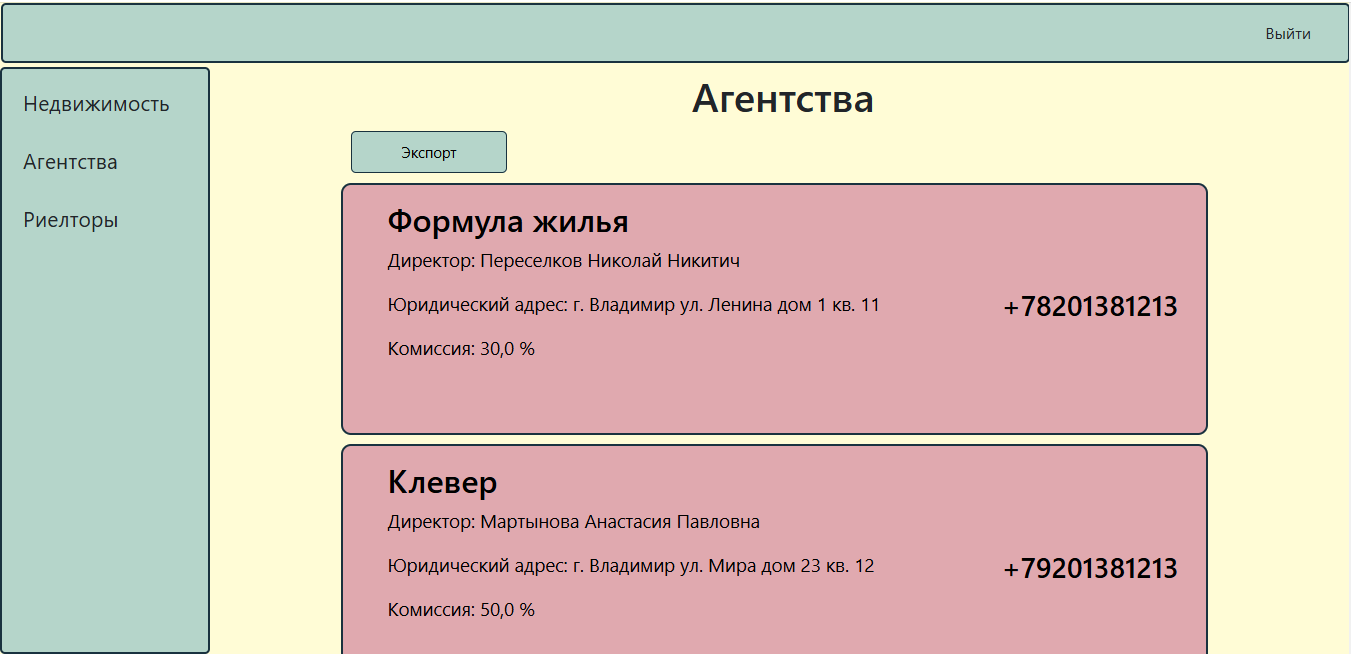


Рисунок – Страница «Агентства» для роли пользователь

Боковое меню для роли менеджер агентств представлено на рисунке 21.

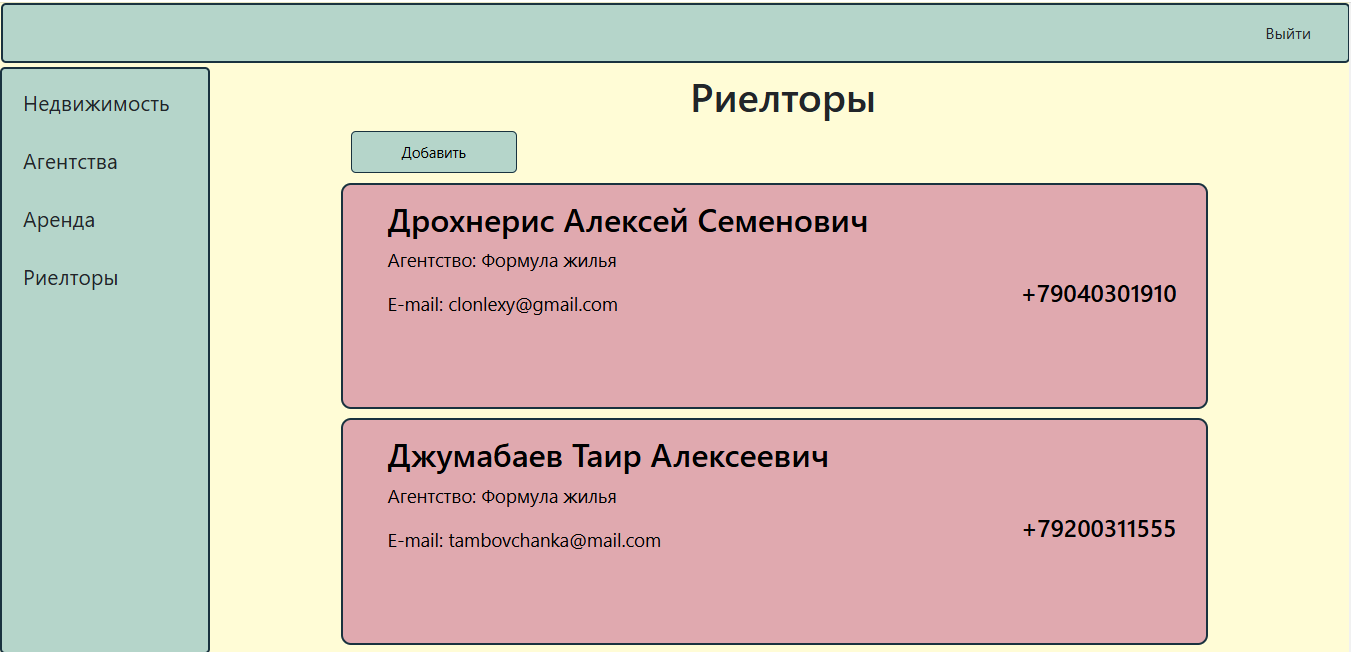


Рисунок – Боковое меню для роли менеджер агентств

Менеджер агентств имеет доступ к пунктам меню «Недвижимость», «Агентства», «Аренда» и «Риелторы». На страницах «Агентства» и «Риелторы» присутствуют все ранее описанные кнопки управления объектами. Страницы «Недвижимость» и «Аренда» доступны только для просмотра.

3.2 Файловая структура

Файловая структура проекта представлена на рисунке 22.

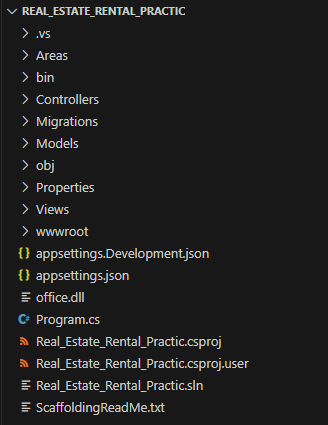


Рисунок – Файловая структура проекта

При создании пользовательского приложения был использован паттерн проектирования MVC(Models-Views-Controllers), в основе которого лежит принцип разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер.

Проект содержит следующие папки:

1. Areas. Содержит папку Identity, которая хранит в себе инструменты для управления системой авторизации и регистрации пользователей. Identity содержит следующие папки:
   1. Data. Содержит контекст подключения к базе данных и класс пользователя Identity.
   2. Pages. Содержит страницы подключения необходимых зависимостей и папку Account, в которой находятся страницы и логика для осуществления авторизации и регистрации.
2. Controllers. Содержит классы контроллеров, которые обрабатывают запросы от пользователей и возвращают соответствующие представления.
3. Migrations. Используется для хранения миграций базы данных, что позволяет управлять изменениями в ее структуре.
4. Models. Содержит классы моделей, которые представляют данные и бизнес-логику приложения. Models содержит следующие папки:
   1. Data. Содержит контекст подключения к БД.
   2. DbModels. Содержит классы сущностей БД.
5. Views. Содержит представления, которые предоставляют пользователю графический интерфейс и служат для отображения данных.

3.3 Структура проекта

В разработанной программе каждая сущность представлена одним отдельным классом. Классы сущностей представлены на рисунке 23.

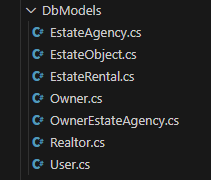


Рисунок – Классы сущностей

Для каждого класса был реализован контроллер, который управляет набором страниц для просмотра, редактирования, удаления и добавления записей в таблицу. Контроллеры для каждой сущности представлены на рисунке 24.

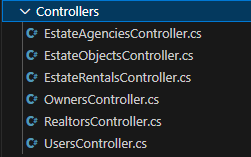


Рисунок – Контроллеры для каждой сущности

Для каждого контроллера были созданы отдельные папки, содержащие представления для соответствующих сущностей. Папки, содержащие представления для каждой сущности представлены на рисунке 25.

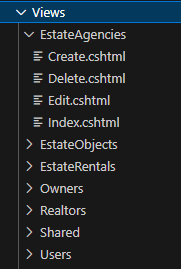


Рисунок – Представления для каждой сущности

Таким образом был реализован принцип единственной ответственности, а также разделена логика и пользовательский интерфейс, что значительно упрощает процесс разработки, позволяя безопасно изменять независимые друг от друга части проекта, улучшая читаемость и поддержку кода.

3.4 Оформление кода

Все вышеперечисленные классы, представления и методы имеют названия, соответствующие функциям, которые они выполняют, а также соглашению об именовании и стилю CamelCase. Комментирование кода отсутствует, за исключением файла Program.cs, который отвечает за настройку используемых сервисов. Комментирование кода в файле Program.cs представлено на рисунке 26.

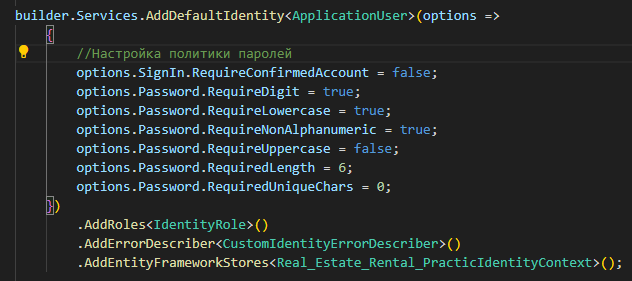


Рисунок – Комментирование кода

Во избежание дублирования логики, ко всем повторяющимся графическим элементам страниц – кнопкам, карточкам, заголовкам и формам были подключены единые стили. Использование одинаковых классов стилей на разных страницах представлено на рисунках 27-30.

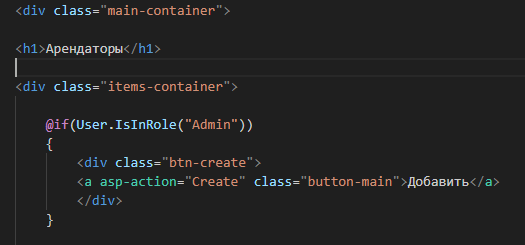


Рисунок – Подключение стилей к странице «Арендаторы»

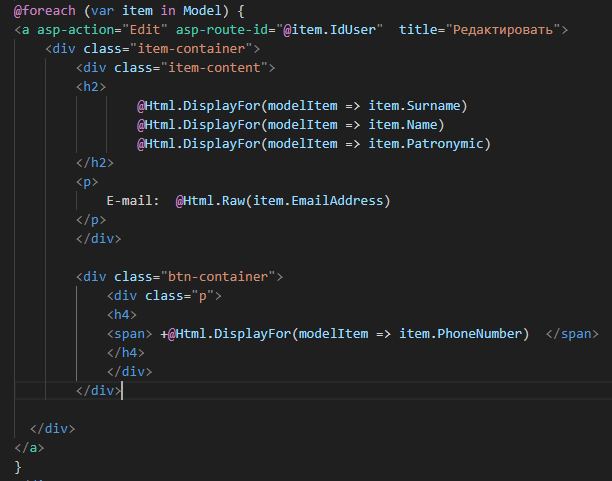


Рисунок – Подключение стилей к странице «Арендаторы»

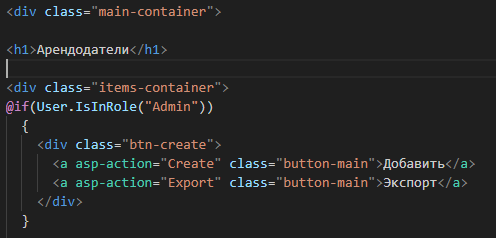


Рисунок – Подключение стилей к странице «Арендодатели»



Рисунок – Подключение стилей к странице «Арендодатели»

На данных скриншотах видно, что каждая страница имеет одинаковые классы для контейнеров и кнопок, что обеспечивает общее единство стиля и упрощение процесса разработки.

Отображение повторяющихся объектов представлено на рисунках 31-32.

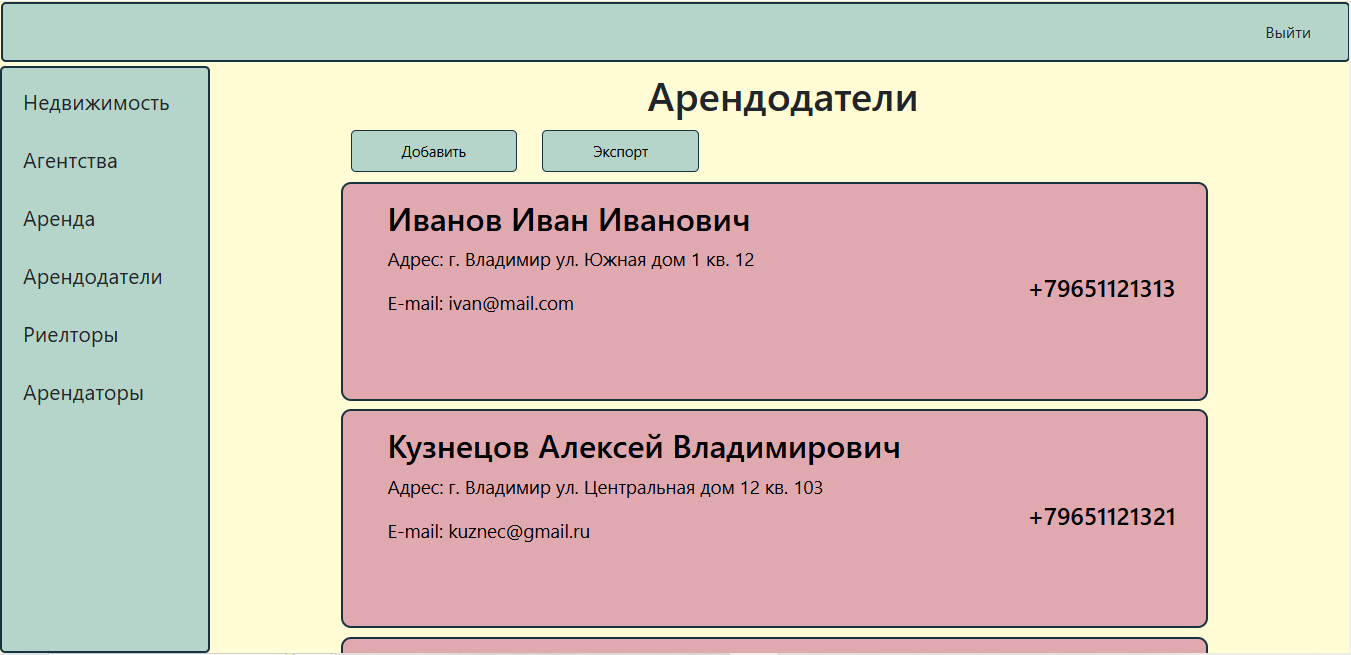


Рисунок – Страница «Арендодатели»

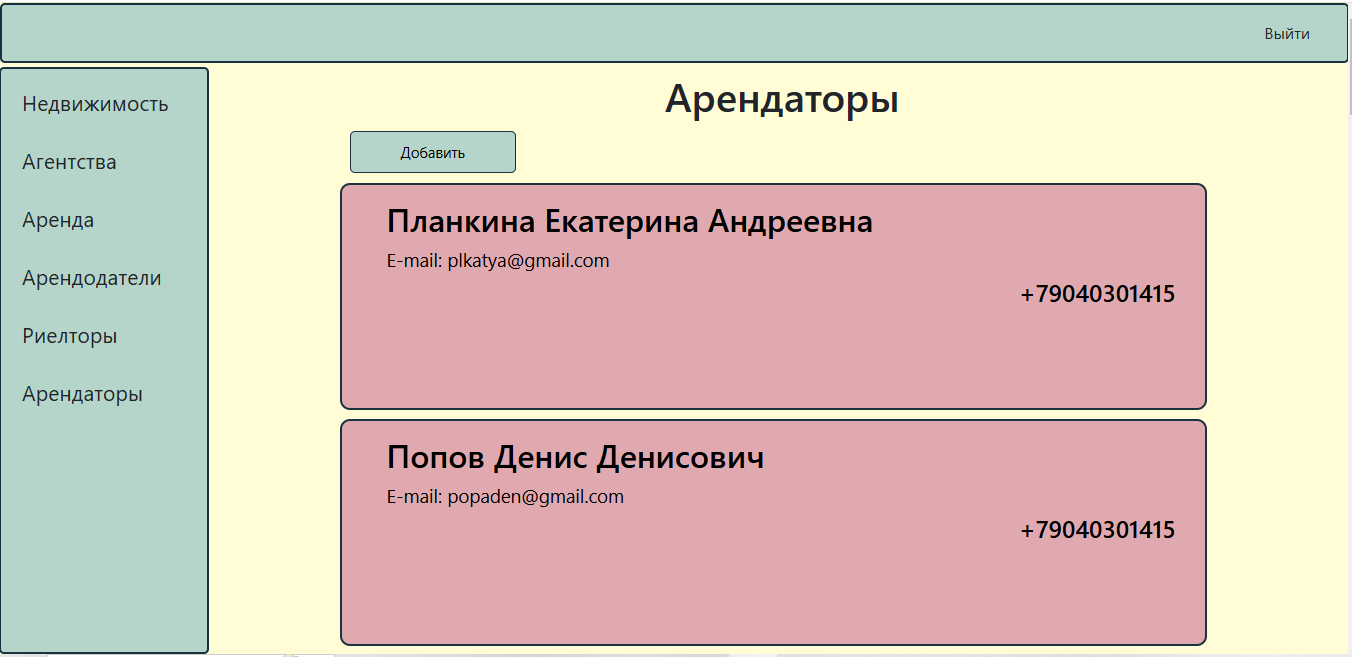


Рисунок – Страница «Арендаторы»

В приложении используются единые классы для форм и отображения ошибок. Подключение стилей для форм и валидационных сообщений представлено на рисунках 33-34.

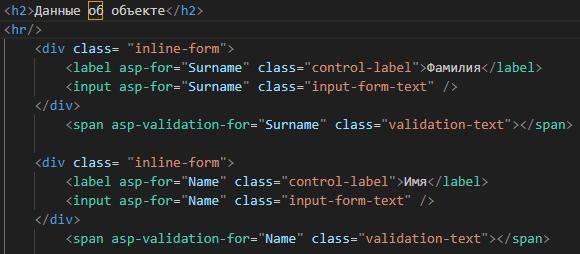


Рисунок – Подключение стилей для форм и валидационных сообщений

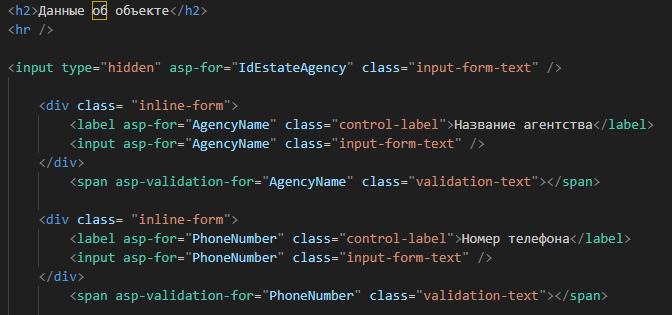


Рисунок – Подключение стилей для форм и валидационных сообщений

Отображение одинаковых форм и валидационных сообщений представлено на рисунках 35-36.

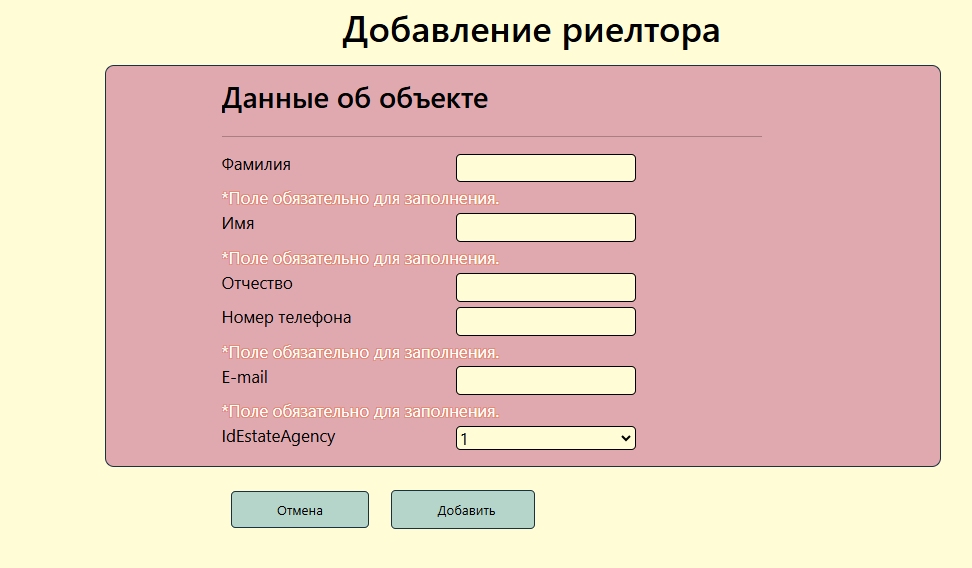


Рисунок – Страница «Добавление риелтора»

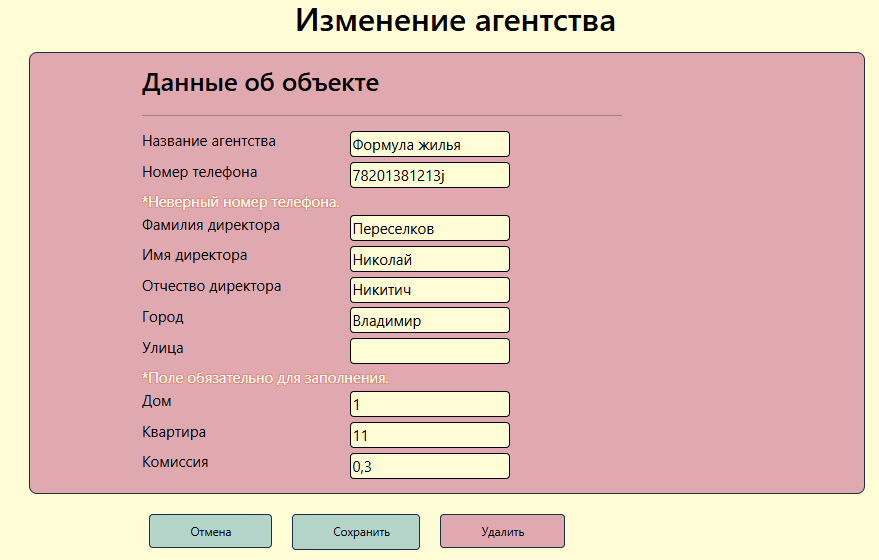


Рисунок – Страница «Изменение агентства»

Кроме того, присутствуют визуальные компоненты, которые отображаются на всех страницах приложения – верхнее и боковое меню. Они были реализованы единожды в папке Shared на странице \_Layout, которая используется для повторяющихся элементов интерфейса.

Таким образом были соблюдены все требования к оформлению кода. Код классов контроллеров представлен в приложении А.

3.5 Создание отчета в формате .xls

В ходе работы для таблиц БД «EstateAgencies» и «Owners» были созданы функции для генерации отчетов в формате .xlsx.

Доступ к данным функциям осуществляется по нажатию на кнопку «Экспорт» в главном представлении класса. Завершение загрузки отчетов Excel представлено на рисунках 37-38.

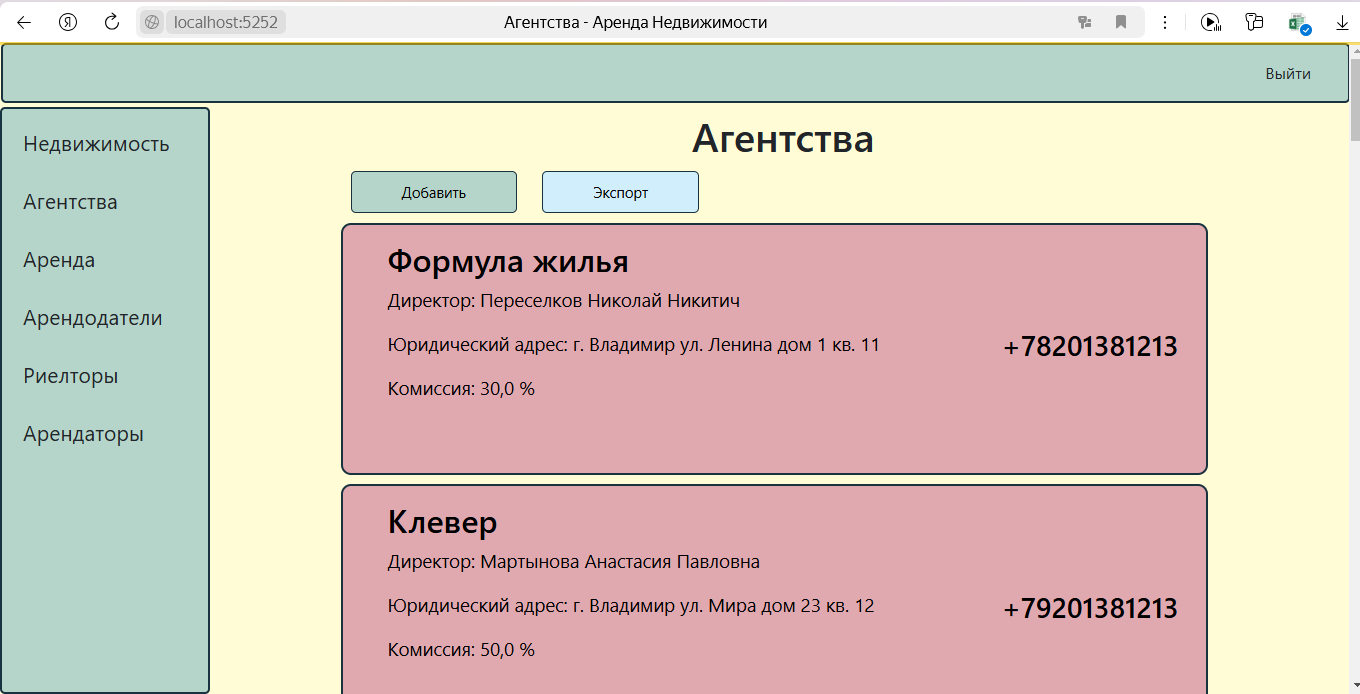


Рисунок – Завершение загрузки отчета Excel

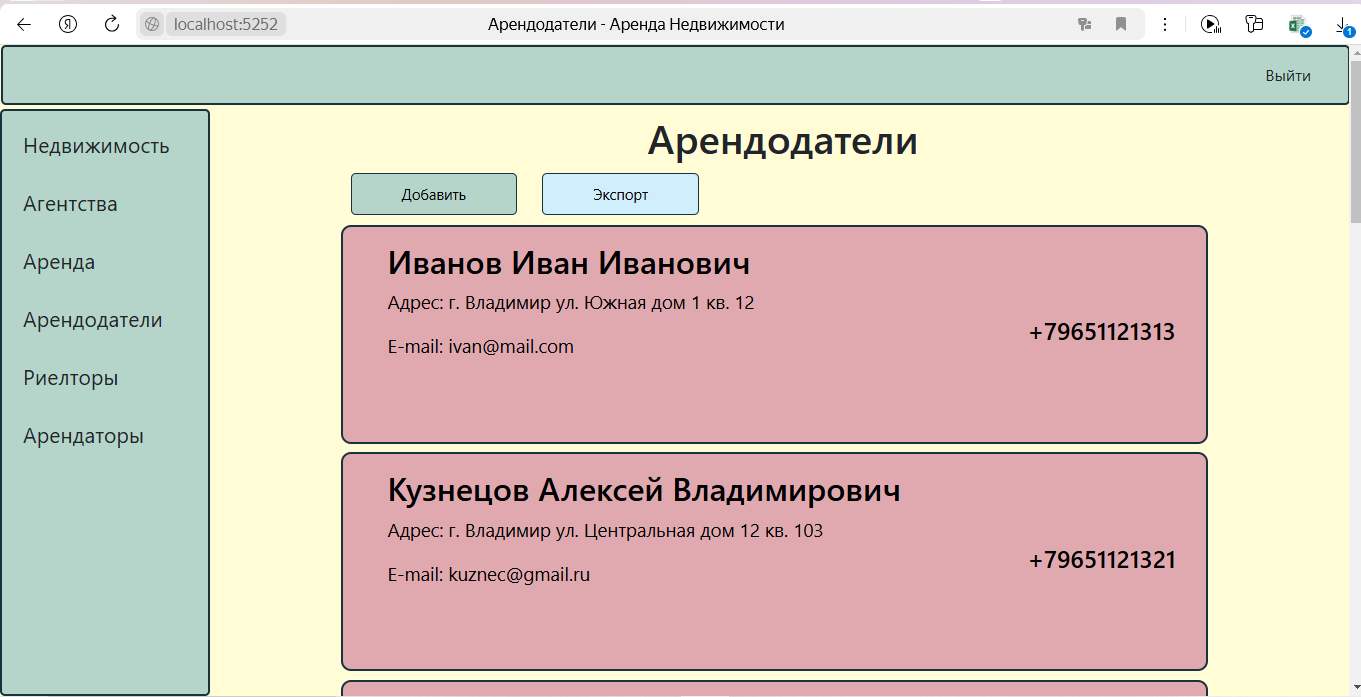


Рисунок – Завершение загрузки отчета Excel

Загруженные файлы «Agencies.xlsx» и «Owners.xlsx» представлены на рисунке 39.

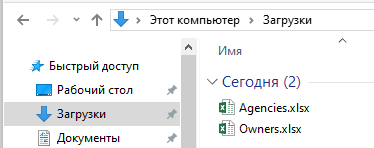


Рисунок – Загруженные файлы «Agencies.xlsx» и «Owners.xlsx»

Содержимое файла «Agencies.xlsx» представлено на рисунке 40.

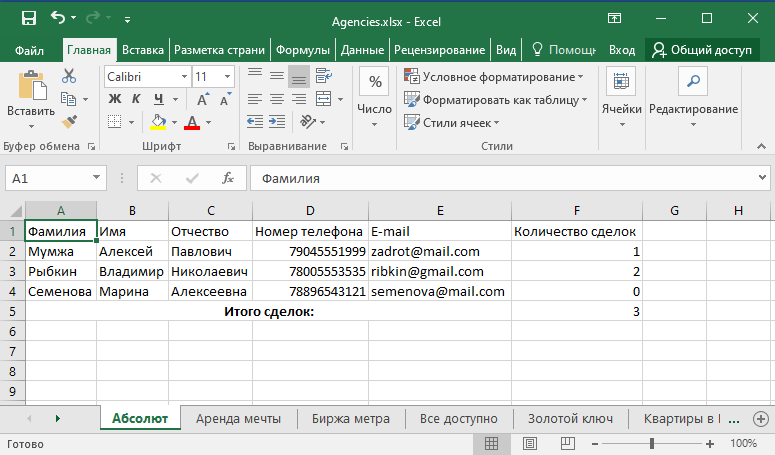


Рисунок – Содержимое файла «Agencies.xlsx»

Отчет содержит в себе страницы с названиями агентств, на каждой из которых представлен список принадлежащих им риелторов, а также количество заключенных ими сделок и общее количество заключенных сделок для каждого агентства.

Содержимое файла «Owners.xlsx» представлено на рисунке 41.

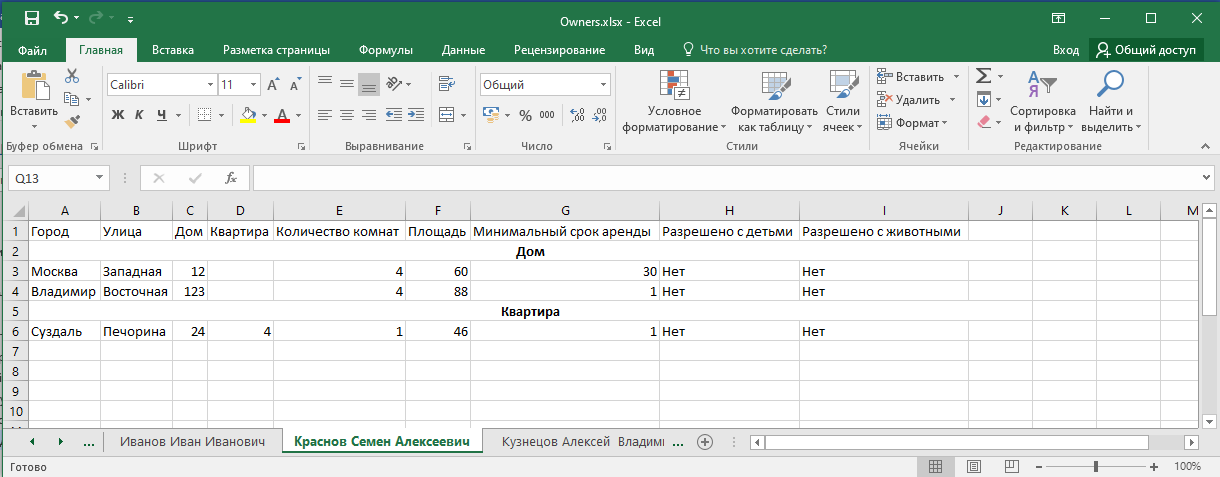


Рисунок – Содержимое файла «Owners.xlsx»

Отчет содержит в себе страницы с фамилией, именем и отчеством арендодателей, на каждой из которых представлен список объектов, которые сдавались ими в аренду, сгруппированные по типу объекта.

Таким образом были созданы отчеты в формате xlsx, которые предоставляют пользователем обобщенные данные, объединенные из нескольких таблиц БД в удобной для них форме.

Код функций для создания отчетов представлен в приложении Б.

4 система контроля версий

Система контроля версий — это инструмент, который сохраняет историю изменений в файлах проекта, позволяя разработчикам отслеживать изменения и управлять версиями кода. Для подключения системы контроля версий к разработанному программному продукту необходимо зарегистрироваться на платформе GitHub и создать новый репозиторий. Создание нового репозитория представлено на рисунке 42.

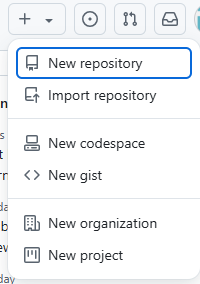


Рисунок – Создание нового репозитория

В открывшемся окне необходимо ввести название репозитория, а также выбрать другие необходимые настройки. Для данного проекта добавлено описание, остальные настройки выставлены по умолчанию. Окно создания репозитория представлено на рисунке 43.

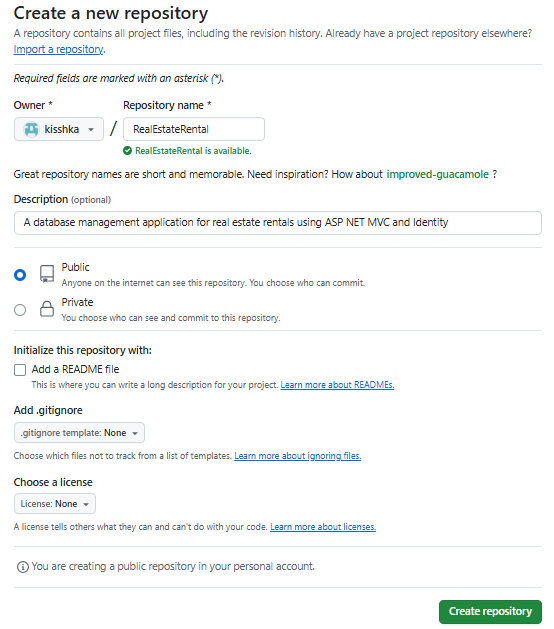


Рисунок – Окно создания репозитория

После создания репозитория, GitHub предоставляет инструкции для загрузки проекта в репозиторий с помощью консоли. Инструкции для загрузки проекта в репозиторий представлены на рисунке 44.



Рисунок – Инструкции для загрузки проекта в репозиторий

Команды запускаются в консоли dot net разработанного приложения. Команда git init используется для создания локального репозитория в указанной директории. Результат выполнения команды представлен на рисунке 45.

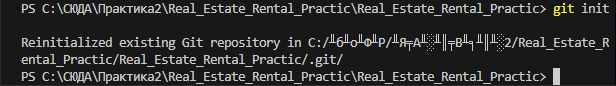


Рисунок – Результат выполнения команды git init

Команда git add добавляет все существующие в данной директории файлы и папки в созданный репозиторий. Результат выполнения команды представлен на рисунке 46.

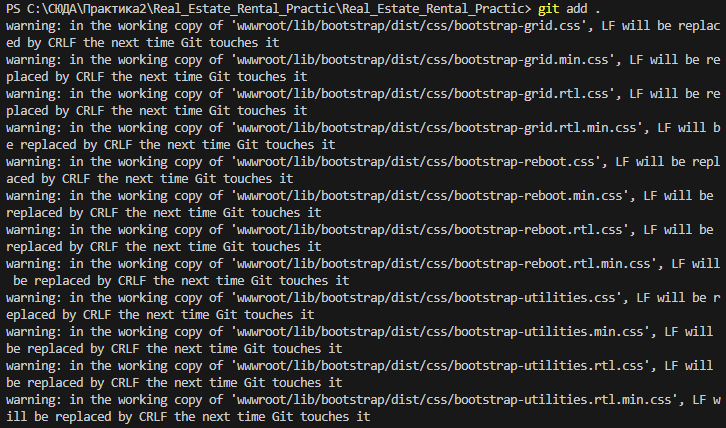


Рисунок – Результат выполнения команды git add

С помощью команды git commit создается сохранение текущего состояния проекта. Создание таких точек позволяет отслеживать все произошедшие изменения и вернуться к предыдущей версии проекта. Результат выполнения команды представлен на рисунке 47.

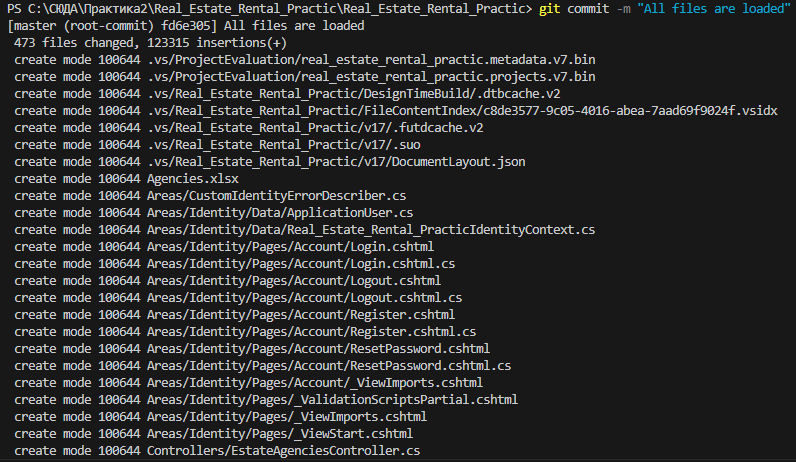


Рисунок – Результат выполнения команды git commit

Команда git branch –M main создает локальную ветку с именем main. Команда git remote add origin устанавливает связь с удаленным репозиторием, созданным ранее. Команда git push отправляет изменения из локальной ветки main в удаленный репозиторий с именем origin. Результат команд представлен на рисунке 48.

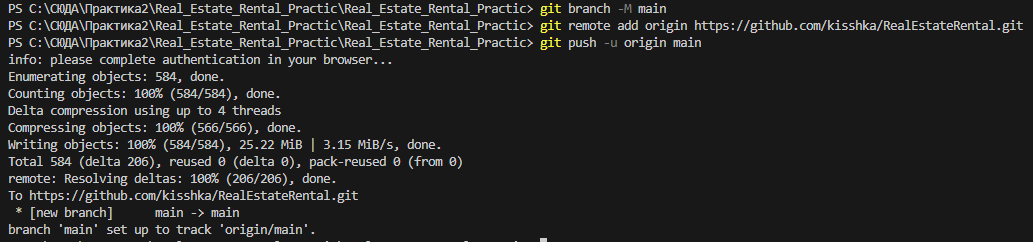


Рисунок – Результат выполнения команд

Все папки текущего проекта были добавлены в удаленный репозиторий GitHub и после обновления страницы отображаются на странице. Репозиторий с загруженным проектом представлен на рисунке 49.

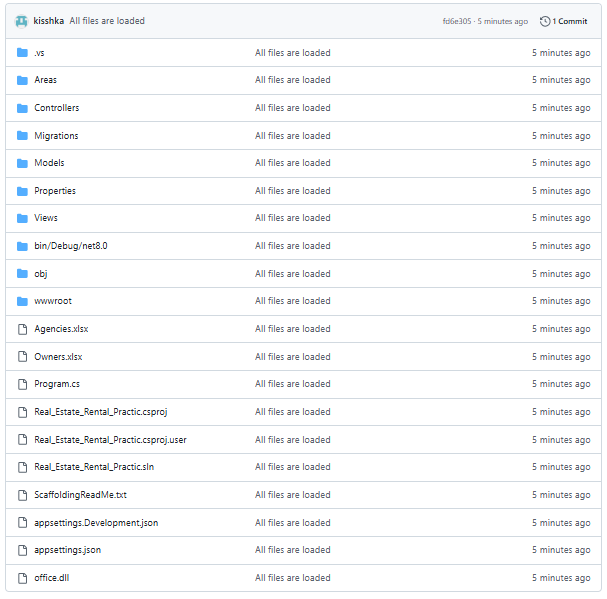


Рисунок – Репозиторий с загруженным проектом

Таким образом к проекту была подключена система контроля версий. Ссылка на репозиторий с добавленным проектом представлена в приложении А.

5 тестирование

5.1 Тест-требования

Тест-требование 1. Проверка корректной работы политики паролей и возможность осуществления операции регистрации.

Проверяет возможность для пользователя зарегистрироваться на сайте при введении логина и пароля, соответствующих установленной политике и появление сообщений об ошибке в ином случае.

Тест-требование 2. Проверка возможности входа в учетную запись.

Проверяет возможность для пользователя осуществить вход в учетную запись при введении существующей пары логина и пароля и наличие сообщения об ошибках при введении несуществующей пары логина и пароля.

Тест-требование 3. Проверка корректного добавления записи в список риелторов.

Проверяет возможность для пользователя с соответствующем уровнем доступа и при введении корректных данных добавить новую запись в список риелторов, а также появление сообщений об ошибках при введении некорректных данных.

Тест-требование 4. Проверка корректного изменения записи в списке агентств.

Проверяет возможность для пользователя с соответствующем уровнем доступа и при введении корректных данных изменить существующую запись в списке агентств, а также появление сообщений об ошибках при введении некорректных данных.

Тест-требование 5. Проверка появления страницы с предупреждением при попытке удалить запись из списка недвижимости.

Проверяет перенаправление на страницу подтверждения удаления при попытке удалить запись из списка недвижимости.

Тест-требование 6. Проверка корректного удаления записи из списка недвижимости.

Проверяет удаление записи из списка недвижимости при подтверждении операции удаления.

Тест-требование 7. Проверка корректной работы выхода из учетной записи.

Проверяет завершение сессии текущего пользователя и перенаправление на страницу входа при нажатии на соответствующую кнопку.

5.2 Тест-план

Ручное тестирование.

Тестовый пример: тестовый пример 1.

Тест-требование: тест-требование 1.

Входные данные:

* Некорректные данные: логин – test@gmail.com, пароль – 123456, подтверждение пароля – 123456.
* Корректные данные: логин – test@gmail.com, пароль – test@gmail.com123, подтверждение пароля – test@gmail.com123.

Описание теста: в данном тесте проверяется работа политики паролей и появление сообщений об ошибке при введении пароля, несоответствующего политике, а также возможность для пользователя зарегистрироваться при введении корректных данных.

Шаги по выполнению теста:

* Перейти на страницу «Регистрация»
* Ввести пароль, несоответствующий установленной политике.
* Проверить наличие сообщений об ошибке.
* Ввести пароль, соответствующий установленной политике
* Убедиться, что был создан новый пользователь и осуществлен вход в новую учетную запись.

Тестовый пример: тестовый пример 2.

Тест-требование: тест-требование 2.

Входные данные:

* Некорректные данные: логин – test@gmail.com, пароль – 123456.
* Корректные данные: логин – test@gmail.com, пароль – test@gmail.com123.

Описание теста: в данном тесте проверяется возможность войти в систему для пользователя, который ввел существующую пару логина и пароля и появление сообщений об ошибке при вводе несуществующей пары.

Шаги по выполнению теста:

* Перейти на страницу «Вход».
* Ввести несуществующую пару логина и пароля.
* Ввести существующую пару логина и пароля
* Проверить вход в учетную запись.
* Наблюдать сообщения об ошибке

Тестовый пример: тестовый пример 3.

Тест-требование: тест-требование 3.

Входные данные:

* Некорректные данные: фамилия – пустое поле, имя – Иван, отчество – Иванович, номер телефона – 19209209292, E-mail – ivan@gmail.com, IdEstateAgecy – 1.
* Корректные данные: фамилия – Иванов, имя – Иван, отчество – Иванович, номер телефона – 79209209292, E-mail – ivan@gmail.com, IdEstateAgecy – 1.

Описание теста: проверяет корректное добавление записи в список риелторов при заполнении формы корректными данными и появление сообщений об ошибке при заполнении некорректными данными.

Шаги по выполнению теста:

* Открыть страницу «Риелторы» в роли администратора.
* Нажать кнопку «Добавить».
* Ввести некорректные данные в форму
* Нажать кнопку «Добавить».
* Наблюдать сообщения об ошибках
* Ввести корректные данные в форму
* Убедиться в добавлении новой записи в список риелторов.

Тестовый пример: тестовый пример 4.

Тест-требование: тест-требование 4.

Входные данные:

* Некорректные данные: фамилия директора – пустое поле, номер директора – 9999.
* Корректные данные: название агентства – Новое жилье, фамилия директора – Куранов, комиссия – 0,5.

Описание теста: проверяет изменение существующей записи в списке агентств при заполнении формы корректными данными и появление сообщений об ошибке при заполнении некорректными данными.

Шаги по выполнению теста:

* Открыть страницу «Агентства» в роли администратора.
* Нажать на карточку агентства.
* Ввести некорректные данные в форму.
* Наблюдать сообщения об ошибке
* Ввести корректные данные в форму.
* Убедиться, что существующая запись была изменена.

Тестовый пример: тестовый пример 5.

Тест-требование: тест-требование 5.

Входные данные: нет.

Описание теста: проверяет невозможность удаления записи без подтверждения от пользователя.

Шаги по выполнению теста:

* Открыть страницу «Недвижимость» в роли администратора
* Нажать на карточку объекта недвижимости.
* Нажать кнопку «Удалить».
* Убедиться, что пользователь был перенаправлен на страницу подтверждения удаления.

Тестовый пример: тестовый пример 6.

Тест-требование: тест-требование 6.

Входные данные: нет.

Описание теста: проверяет, что объект недвижимости будет удален из БД после подтверждения операции удаления.

Шаги по выполнению теста:

* Открыть страницу «Недвижимость» в роли администратора
* Нажать на карточку объекта недвижимости.
* Нажать кнопку «Удалить».
* Подтвердить операцию удаления.
* Убедиться, что карточка была удалена из списка объектов недвижимости

Тестовый пример: тестовый пример 7

Тест-требование: тест-требование 7

Входные данные: Нет

Описание теста: проверяет корректную работу операции выхода из учетной записи.

Шаги по выполнению теста:

* Войти в учетную запись с любой ролью.
* Нажать кнопку «Выйти».
* Убедиться, что был произведен выход из учетной записи.

5.3 Результаты тестирования

Ручное тестирование.

Тест-требование 1. Проверка корректной работы политики паролей и регистрации в системе.

Был осуществлен переход на страницу «Регистрация» и введены данные, несоответствующие установленной политике пароля. Заполнение формы регистрации представлено на рисунке 50.

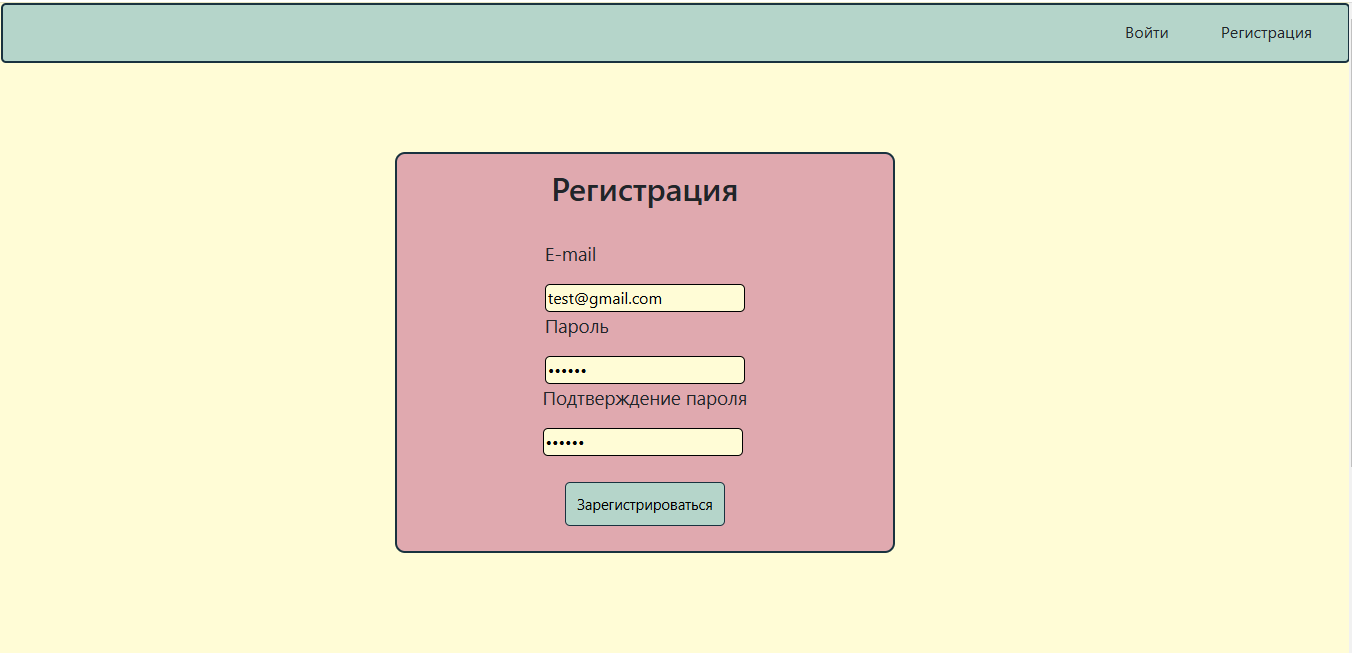


Рисунок – Заполнение формы регистрации

После нажатия на кнопку «Зарегистрироваться» можно наблюдать валидационные сообщения, свидетельствующие о несоблюдении политики паролей. Валидационные сообщения представлены на рисунке 51.

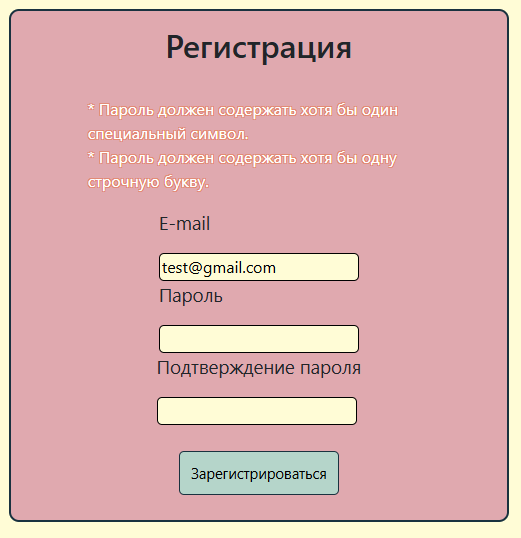


Рисунок – Валидационные сообщения

Далее в форму были введены данные, соответствующие установленной политике пароля. Заполнение формы регистрации представлено на рисунке 52.

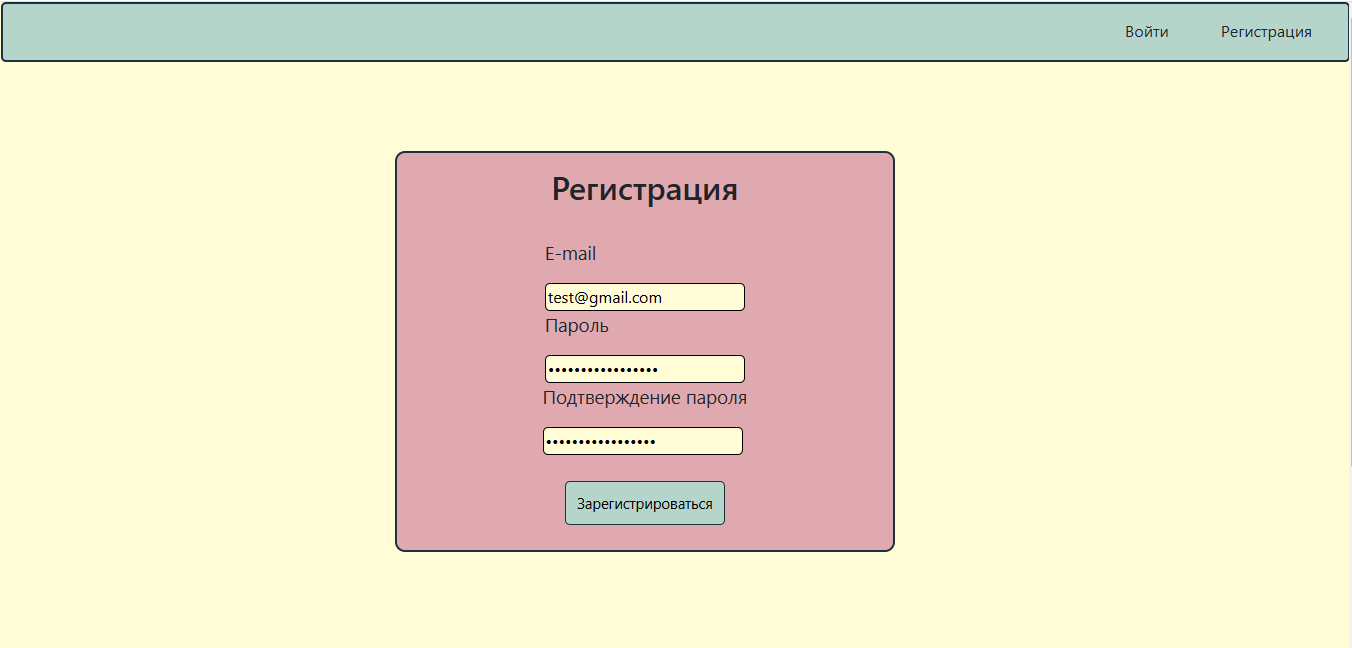


Рисунок – Заполнение формы регистрации

После нажатия на кнопку «Регистрация» осуществляется вход в приложение в роли пользователь. Главная страница сайта для пользователя представлена на рисунке 53.

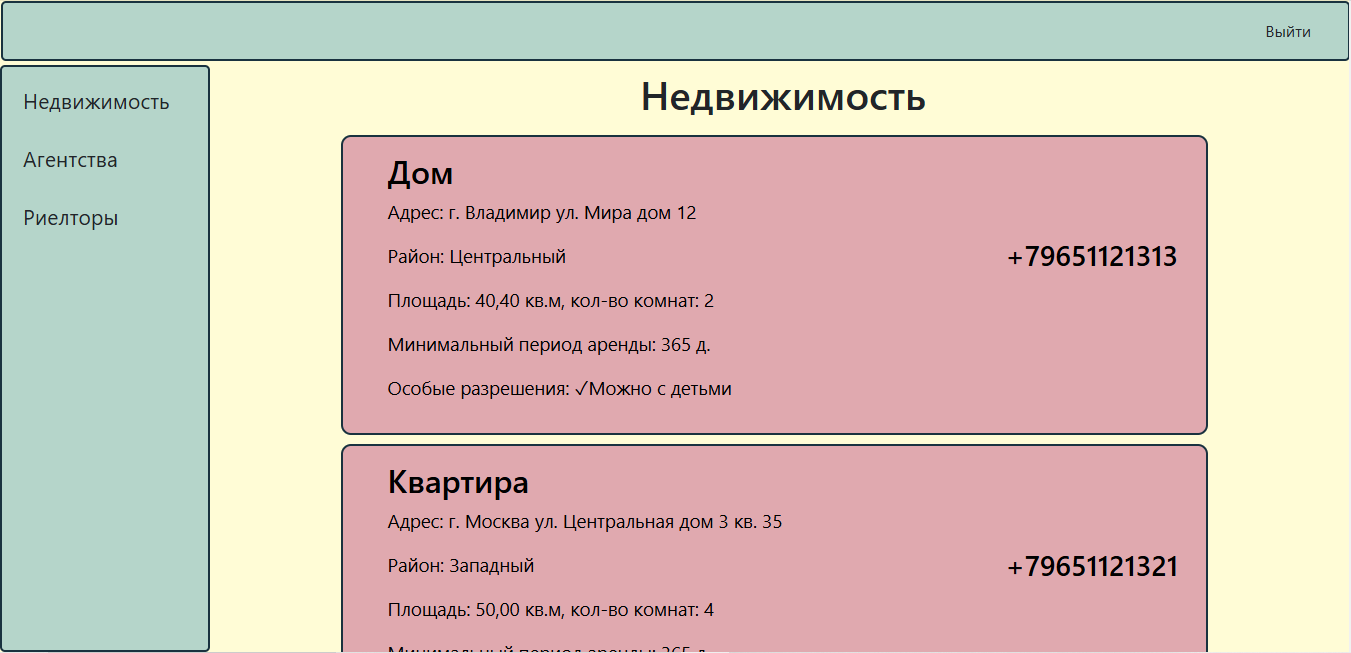


Рисунок – Главная страница сайта для пользователя

В таблицу AspNetUsers был добавлен новый пользователь. Таблица AspNetUsers представлена на рисунке 54.

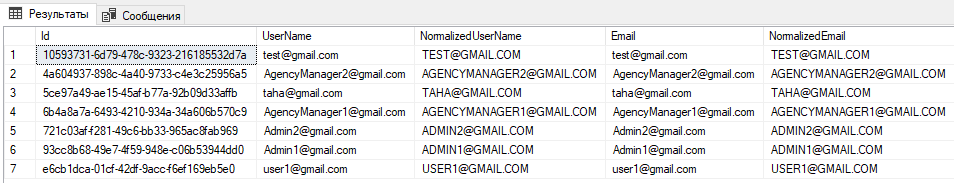


Рисунок – Таблица AspNetUsers

Результаты теста на проверку корректной работы политики паролей и осуществления регистрации соответствует ожиданиям.

Тест-требование 2. Проверка возможности входа в учетную запись.

При открытии приложения осуществляется переход на страницу «Вход». В форму были введены несуществующие логин и пароль. После нажатия на кнопку «Войти» пользователь получает сообщение об ошибке. Сообщение об ошибке представлено на рисунке 55.



Рисунок – Сообщение об ошибке

Далее в форму были введены логин и пароль пользователя, зарегистрированного в тестовом примере 1. Заполнение формы входа представлено на рисунке 56.

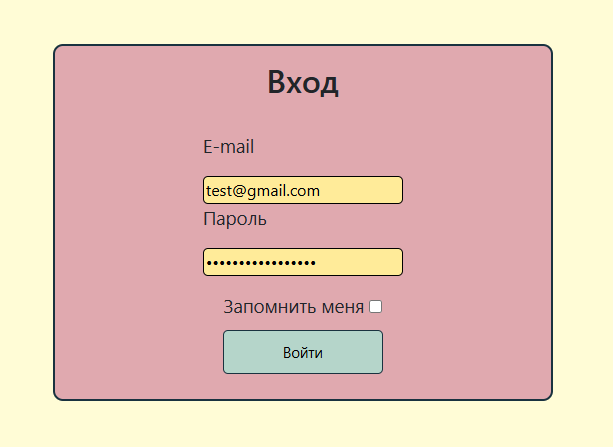


Рисунок – Заполнение формы входа

После нажатия на кнопку «Войти» производится вход в учетную запись, и пользователь перенаправляется на главную страницу. Переход на главную страницу представлен на рисунке 57.



Рисунок – Переход на главную страницу

Результаты теста на проверку осуществления корректного входа в учетную запись соответствует ожиданиям.

Тест-требование 3. Проверка корректного добавления записи в список риелторов.

Был произведен вход в учетную запись пользователя в роли администратора и осуществлен переход на страницу «Риелтор». После нажатия на кнопку «Добавить», в форму были введены некорректные данные для создания нового риелтора. После нажатия на кнопку «Добавить» можно наблюдать сообщения об ошибках. Сообщения об ошибке представлены на рисунке 58.

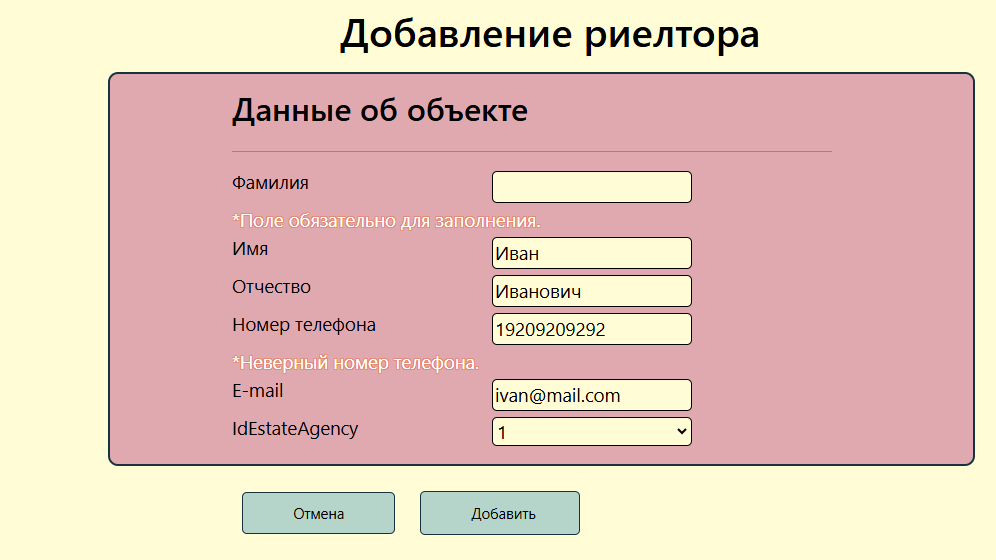


Рисунок – Сообщения об ошибке

Далее данные в форме были изменены на корректные. Заполнение формы корректными данными представлено на рисунке 59.

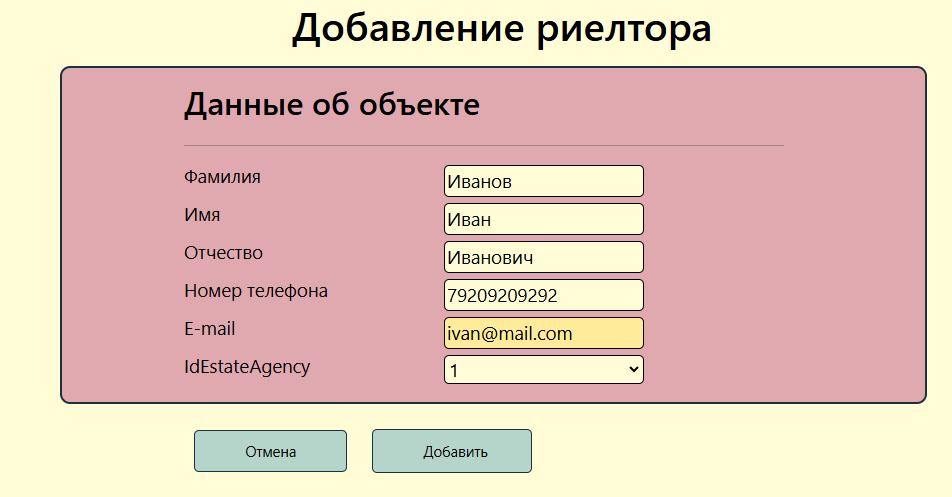


Рисунок – Заполнение формы корректными данными

После нажатия на кнопку «Добавить» новая запись отображается в списке риелторов. Отображение нового риелтора представлено на рисунке 60.

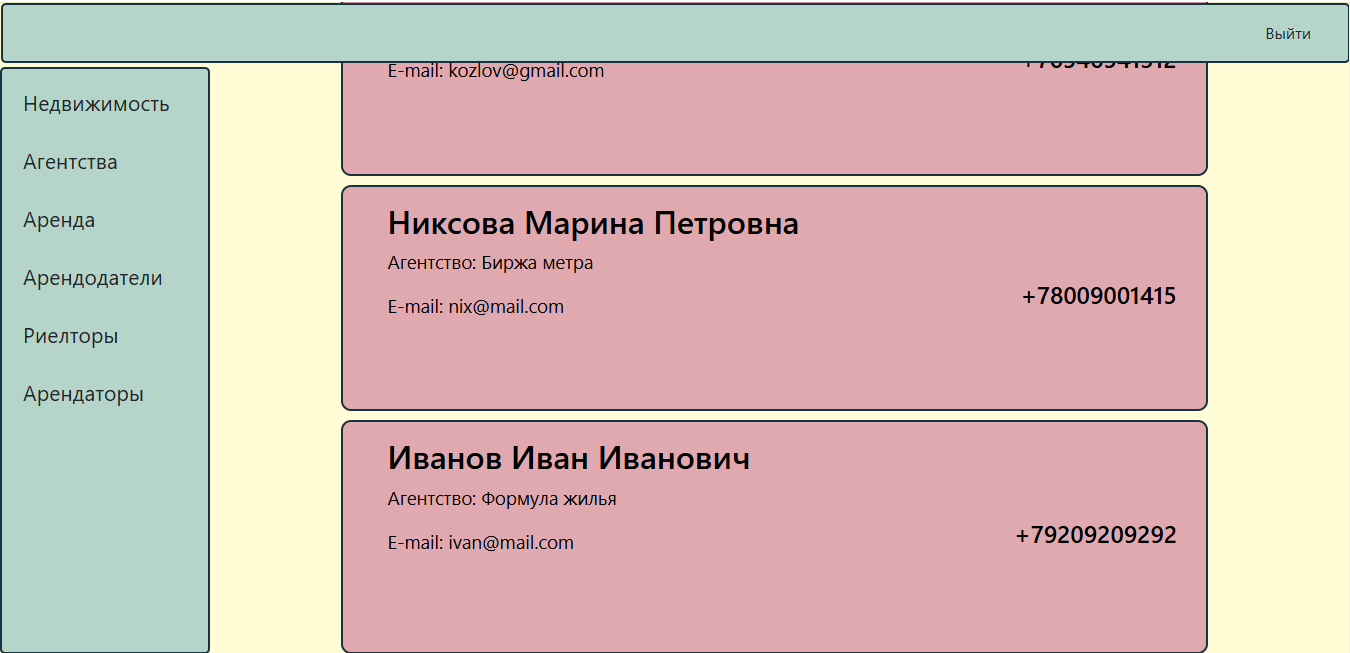


Рисунок – Отображение нового риелтора

Результаты теста на проверку корректного добавления записи в список риелторов соответствуют ожиданиям.

Тест-требование 4. Проверка корректного изменения записи в списке агентств.

После входа в учетную запись с ролью администратор и перехода на страницу «Агентства» выбирается карточка для изменения. В данном случае это будет самая первая отображаемая в списке карточка. Карточка агентства до изменения представлена на рисунке 61.



Рисунок – Карточка агентства до изменения

По нажатию на карточку агентства, была открыта и заполнена некорректными данными форма редактирования. После нажатия на кнопку «Сохранить» можно наблюдать сообщения об ошибке. Сообщения об ошибке представлены на рисунке 62.

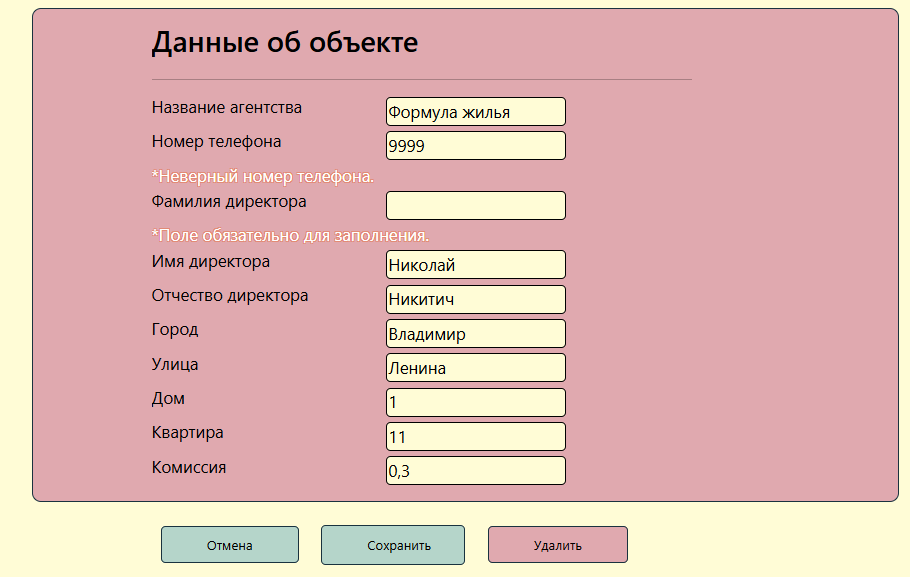


Рисунок – Сообщения об ошибке

Далее форма была заполнена корректными данными, отличными от изначальных. Заполнение формы корректными данными представлено на рисунке 63.

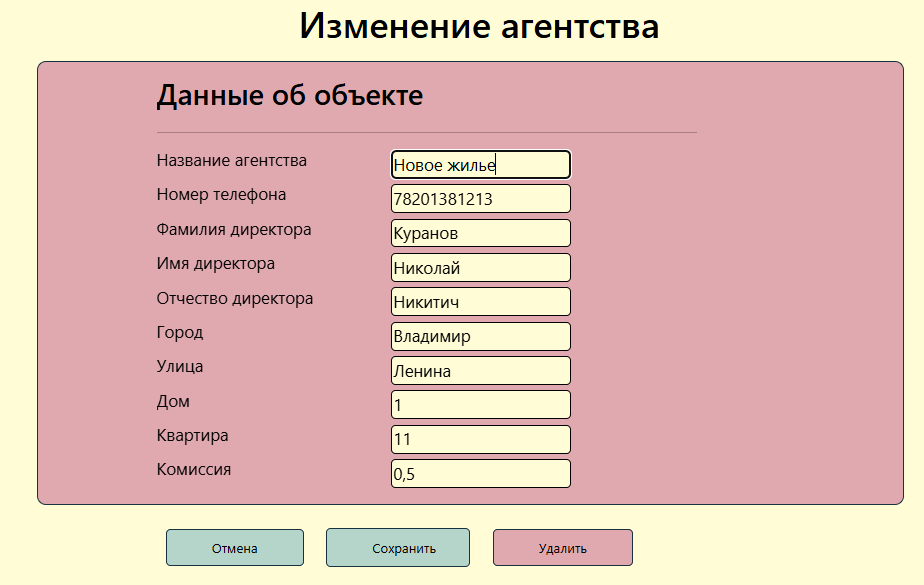


Рисунок – Заполнение формы корректными данными

После нажатия на кнопку «Сохранить», можно наблюдать измененную карточку в списке агентств. Измененная карточка в списке агентств представлена на рисунке 64.

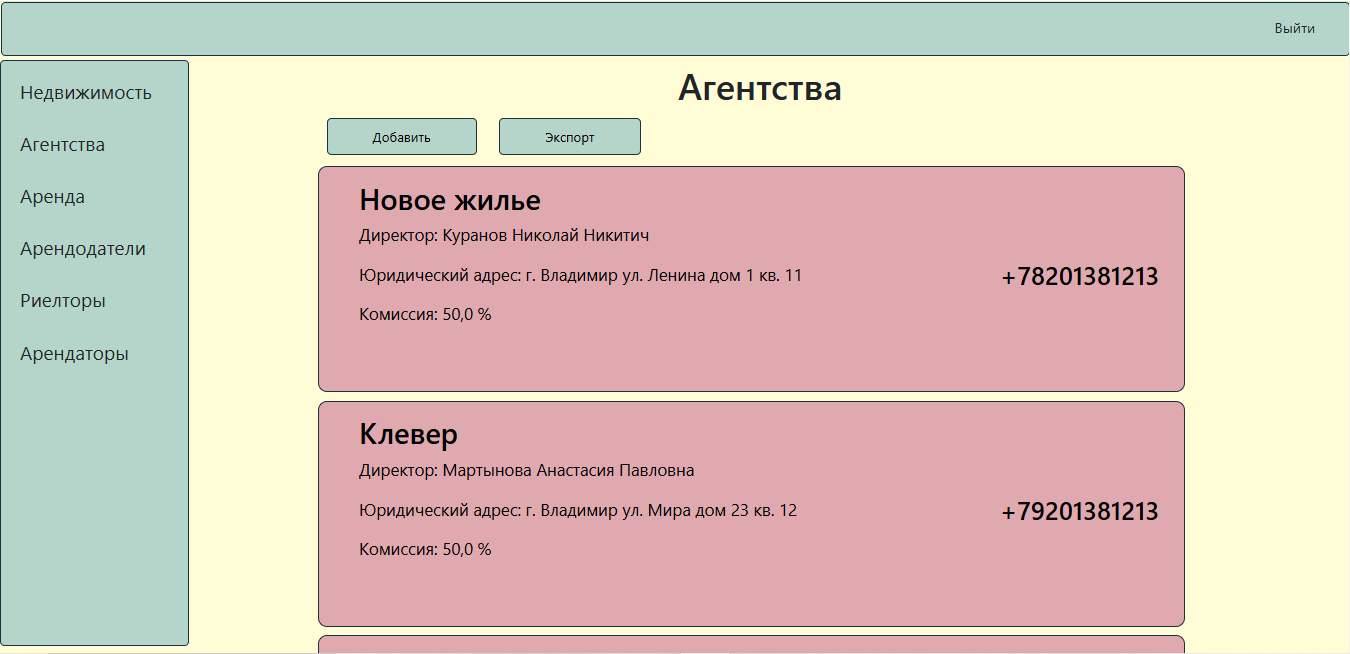


Рисунок – Измененная карточка в списке агентств

Результаты теста на проверку корректного изменения записи в списке агентств соответствуют ожиданиям.

Тест-требование 5. Проверка появления страницы с предупреждением при попытке удалить запись из списка недвижимости.

После входа в учетную запись с ролью администратор, был осуществлен переход на страницу недвижимость. При нажатии на карточку товара и перехода в форму редактирования, пользователю доступна кнопка удалить. При нажатии на кнопку «Удалить», появляется страница с предупреждением. Страница с предупреждением представлена на рисунке 65.

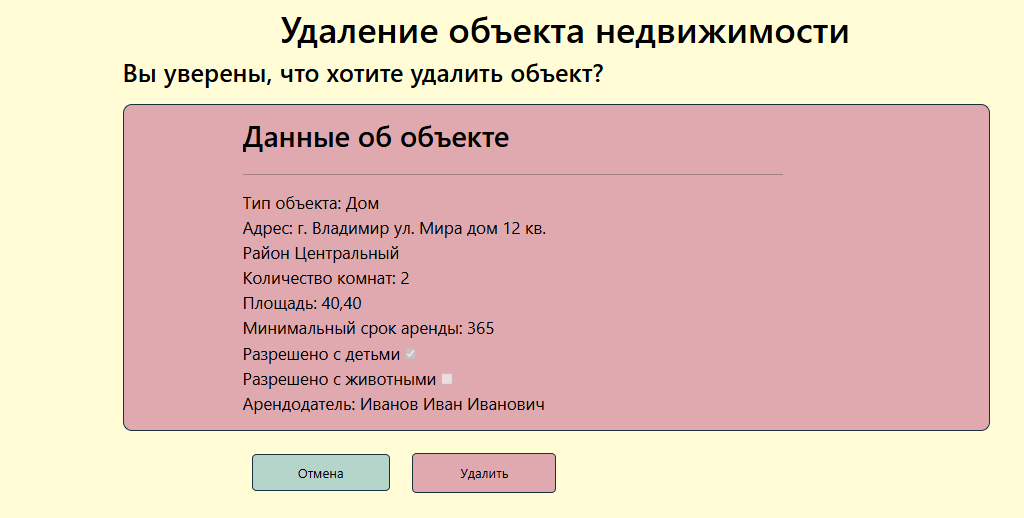


Рисунок – Страница с предупреждением

Результаты теста на проверку появления страницы для подтверждения удаления соответствуют ожиданиям.

Тест-требование 6. Проверка корректного удаления записи из списка недвижимости.

Для удаления выбирается самая первая карточка в списке объектов недвижимости. Список объектов недвижимости до удаления представлен на рисунке 66.



Рисунок – Список объектов недвижимости до удаления

В данном тесте для данной карточки повторяются все действия из тестового сценария 5.

В окне подтверждения удаления, после нажатия на кнопку «Удалить», данная карточка больше не отображается в списке объектов недвижимости. Список объектов недвижимости после удаления представлен на рисунке 67.



Рисунок – Список объектов недвижимости после удаления

Результаты теста на проверку удаления объекта недвижимости из списка при подтверждении операции удаления соответствуют ожиданиям.

Тест-требование 7. Проверка корректной работы выхода из учетной записи.

Был осуществлен вход в учетную запись test@gmail.com. Интерфейс для пользователя test@gmail.com представлен на рисунке 68.



Рисунок – Интерфейс для пользователя test@gmail.com

После нажатия на кнопку «Выйти» осуществляется переход на страницу «Вход» и текущая сессия пользователя завершается. Выход из учетной записи представлен на рисунке 69.

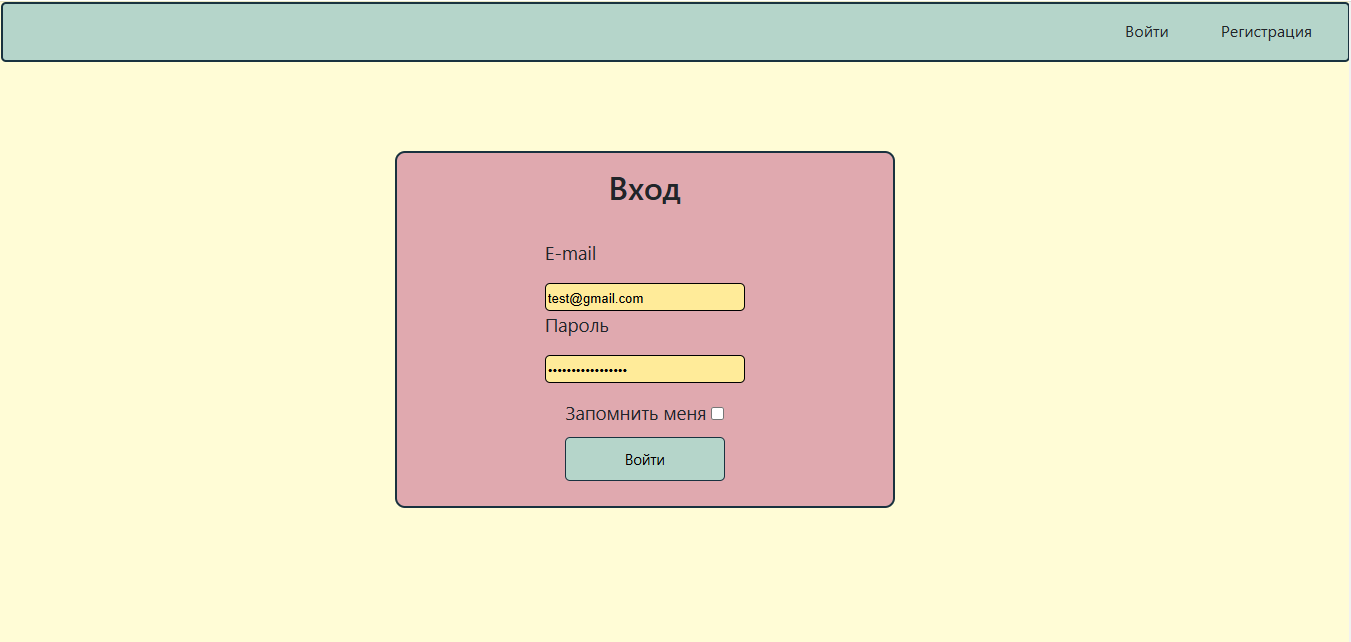


Рисунок – Выход из учетной записи

Результаты теста на проверку корректной работы выхода из учетной записи соответствует ожиданиям.

заключение

В процессе прохождения учебной практики УП.01.01 по модулю ПМ.01 «Разработка модулей программного обеспечения для компьютерных систем» была рассмотрена предметная область аренды недвижимости.

Для создания приложения для управления арендами в соответствии с установленным заданием, были выполнены такие задачи, как подключение к базе данных, создание и управление ролями и пользователями и генерация отчетов в формате .xls.

Для готового приложения были описаны файловая структура и структура проекта, оформление кода в соответствии с требованиями. Приложение было загружено на GitHub и подключено к системе контроля версий, что позволило отслеживать все изменения в проекте.

Для всех функций программного продукта были проведены и описаны ручные тесты.

список использованных источников

1. Безопасность баз данных: учебник для СПО / М. А. Полтавцева. — Санкт-Петербург: Лань, 2024. — 356 с.
2. Microsoft Corporation. Документация по ASP.NET Core MVC [Электронный ресурс] – режим доступа: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview) (дата обращения: 07.05.2025)
3. ASP.NET Core в действии / пер. с англ. Д.А. Беликова — М.: ДМК Пресс, 2024. – 1046 с
4. C# 8-и .NET Core. Разработка и оптимизация. — СПб: Питер, 2021. – 816с.
5. METANIT.COM [Электронный ресурс]: – Режим доступа: https://metanit.com/sharp/mvc5/ (дата обращения: 07.05.2025);

приложение А

Ссылка на репозиторий проекта

https://github.com/kisshka/RealEstateRental.git

Листинг класса EstateAgenciesController

[Authorize]

public class EstateAgenciesController : Controller

{

private readonly RealEstateRentalContext \_context;

public EstateAgenciesController(RealEstateRentalContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> Index()

{

return View(await \_context.EstateAgencies.ToListAsync());

}

public IActionResult Export()

{

var allAgencies = \_context.EstateAgencies.OrderBy(p => p.AgencyName).ToList();

var application = new Excel.Application();

application.Visible = false;

application.SheetsInNewWorkbook = allAgencies.Count;

Excel.Workbook workbook = application.Workbooks.Add(Type.Missing);

for (int i = 0; i < allAgencies.Count(); i++)

{

Excel.Worksheet worksheet = (Worksheet)application.Worksheets.Item[i + 1];

worksheet.Name = allAgencies[i].AgencyName;

worksheet.Cells[1, 1] = "Фамилия";

worksheet.Cells[1, 2] = "Имя";

worksheet.Cells[1, 3] = "Отчество";

worksheet.Cells[1, 4] = "Номер телефона";

worksheet.Cells[1, 5] = "E-mail";

worksheet.Cells[1, 6] = "Количество сделок";

var realtors = \_context.Realtors

.Where(p => p.IdEstateAgency == allAgencies[i].IdEstateAgency)

.OrderBy(p => p.Surname)

.ToList();

int rowIndex = 2;

int totalDeals = 0;

foreach (var realtor in realtors)

{

int dealsCount = \_context.EstateRentals.Count(e => e.IdRealtor == realtor.IdRealtor);

totalDeals += dealsCount;

worksheet.Cells[rowIndex, 1] = realtor.Surname;

worksheet.Cells[rowIndex, 2] = realtor.Name;

worksheet.Cells[rowIndex, 3] = realtor.Patronymic;

worksheet.Cells[rowIndex, 4] = realtor.PhoneNumber.ToString();

worksheet.Cells[rowIndex, 5] = realtor.EmailAddress;

worksheet.Cells[rowIndex, 6] = dealsCount;

rowIndex++;

}

Excel.Range range = worksheet.Range[worksheet.Cells[rowIndex, 1], worksheet.Cells[rowIndex, 5]];

range.Merge();

range.HorizontalAlignment = Excel.XlHAlign.xlHAlignCenter;

range.Font.Bold = true;

range.Value = "Итого сделок:";

worksheet.Cells[rowIndex, 6] = totalDeals;

}

string filePath = Path.Combine(Directory.GetCurrentDirectory(), "Agencies.xlsx");

if (System.IO.File.Exists(filePath))

{

System.IO.File.Delete(filePath);

}

workbook.SaveAs(filePath);

workbook.Close();

application.Quit();

// Отправка файла пользователю

var fileStream = new FileStream(filePath, FileMode.Open, FileAccess.Read);

return File(fileStream, "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet", "Agencies.xlsx");

}

public IActionResult ExportSuccess() {

return View("~/Views/Shared/ExportSuccess.cshtml");

}

[HttpGet]

public IActionResult Create()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Create([Bind("IdEstateAgency,AgencyName,PhoneNumber,DirectorSurname,DirectorName,DirectorPatronymic,Town,Street,Home,Flat,Comission")] EstateAgency estateAgency)

{

if (ModelState.IsValid)

{

\_context.Add(estateAgency);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(estateAgency);

}

[Authorize(Roles = "Admin,AgencyManager")]

[HttpGet]

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var estateAgency = await \_context.EstateAgencies.FindAsync(id);

if (estateAgency == null)

{

return NotFound();

}

return View(estateAgency);

}

[HttpPost]

public async Task<IActionResult> Edit(int id, [Bind("IdEstateAgency,AgencyName,PhoneNumber,DirectorSurname,DirectorName,DirectorPatronymic,Town,Street,Home,Flat,Comission")] EstateAgency estateAgency)

{

if (id != estateAgency.IdEstateAgency)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(estateAgency);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!EstateAgencyExists(estateAgency.IdEstateAgency))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(estateAgency);

}

[HttpGet]

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var estateAgency = await \_context.EstateAgencies

.FirstOrDefaultAsync(m => m.IdEstateAgency == id);

if (estateAgency == null)

{

return NotFound();

}

return View(estateAgency);

}

[HttpPost, ActionName("Delete")]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var estateAgency = await \_context.EstateAgencies.FindAsync(id);

if (estateAgency != null)

{

\_context.EstateAgencies.Remove(estateAgency);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool EstateAgencyExists(int id)

{

return \_context.EstateAgencies.Any(e => e.IdEstateAgency == id);

}

}

Листинг класса EstateObjectsController

[Authorize]

public class EstateObjectsController : Controller

{

private readonly RealEstateRentalContext \_context;

public EstateObjectsController(RealEstateRentalContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> Index()

{

var realEstateRentalContext = \_context.EstateObjects.Include(e => e.IdOwnerNavigation);

return View(await realEstateRentalContext.ToListAsync());

}

[HttpGet]

public IActionResult Create()

{

ViewData["IdOwner"] = new SelectList(\_context.Owners, "IdOwner", "IdOwner");

return View();

}

[HttpPost]

public async Task<IActionResult> Create([Bind("IdEstateObject,ObjectType,Town,Street,Home,Flat,CittyArea,NumberOfRooms,Square,RentalPeriod,IdOwner,ChildrenAllowed,AnimalsAllowed")] EstateObject estateObject)

{

if(ModelState.IsValid)

{

\_context.Add(estateObject);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["IdOwner"] = new SelectList(\_context.Owners, "IdOwner", "IdOwner", estateObject.IdOwner);

return View(estateObject);

}

[Authorize(Roles = "Admin")]

[HttpGet]

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var estateObject = await \_context.EstateObjects.FindAsync(id);

if (estateObject == null)

{

return NotFound();

}

ViewData["IdOwner"] = new SelectList(\_context.Owners, "IdOwner", "IdOwner", estateObject.IdOwner);

return View(estateObject);

}

[HttpPost]

public async Task<IActionResult> Edit(int id, [Bind("IdEstateObject,ObjectType,Town,Street,Home,Flat,CittyArea,NumberOfRooms,Square,RentalPeriod,IdOwner,ChildrenAllowed,AnimalsAllowed")] EstateObject estateObject)

{

if (id != estateObject.IdEstateObject)

{

return NotFound();

}

if (ModelState.IsValid)

{

\_context.Update(estateObject);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ModelState.AddModelError(nameof(EstateObject.Square), "\*Неверное значение.");

ViewData["IdOwner"] = new SelectList(\_context.Owners, "IdOwner", "IdOwner", estateObject.IdOwner);

return View(estateObject);

}

[HttpGet]

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var estateObject = await \_context.EstateObjects

.Include(e => e.IdOwnerNavigation)

.FirstOrDefaultAsync(m => m.IdEstateObject == id);

if (estateObject == null)

{

return NotFound();

}

return View(estateObject);

}

[HttpPost, ActionName("Delete")]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var estateObject = await \_context.EstateObjects.FindAsync(id);

if (estateObject != null)

{

\_context.EstateObjects.Remove(estateObject);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool EstateObjectExists(int id)

{

return \_context.EstateObjects.Any(e => e.IdEstateObject == id);

}

}

Листинг класса EstateRentalsController

[Authorize]

public class EstateRentalsController : Controller

{

private readonly RealEstateRentalContext \_context;

public EstateRentalsController(RealEstateRentalContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> Index()

{

var realEstateRentalContext = \_context.EstateRentals.Include(e => e.IdEstateObjectNavigation).Include(e => e.IdRealtorNavigation).Include(e => e.IdUserNavigation);

return View(await realEstateRentalContext.ToListAsync());

}

[HttpGet]

public IActionResult Create()

{

ViewData["IdEstateObject"] = new SelectList(\_context.EstateObjects, "IdEstateObject", "IdEstateObject");

ViewData["IdRealtor"] = new SelectList(\_context.Realtors, "IdRealtor", "IdRealtor");

ViewData["IdUser"] = new SelectList(\_context.Users, "IdUser", "IdUser");

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("IdEstateRental,RentBeginning,RentEnding,CostPerMonth,Deposit,IdEstateObject,IdRealtor,IdUser")] EstateRental estateRental)

{

if (ModelState.IsValid)

{

\_context.Add(estateRental);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["IdEstateObject"] = new SelectList(\_context.EstateObjects, "IdEstateObject", "IdEstateObject", estateRental.IdEstateObject);

ViewData["IdRealtor"] = new SelectList(\_context.Realtors, "IdRealtor", "IdRealtor", estateRental.IdRealtor);

ViewData["IdUser"] = new SelectList(\_context.Users, "IdUser", "IdUser", estateRental.IdUser);

return View(estateRental);

}

[Authorize(Roles = "Admin")]

[HttpGet]

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var estateRental = await \_context.EstateRentals.FindAsync(id);

if (estateRental == null)

{

return NotFound();

}

ViewData["IdEstateObject"] = new SelectList(\_context.EstateObjects, "IdEstateObject", "IdEstateObject", estateRental.IdEstateObject);

ViewData["IdRealtor"] = new SelectList(\_context.Realtors, "IdRealtor", "IdRealtor", estateRental.IdRealtor);

ViewData["IdUser"] = new SelectList(\_context.Users, "IdUser", "IdUser", estateRental.IdUser);

return View(estateRental);

}

[Authorize(Roles = "Admin")]

[HttpPost]

public async Task<IActionResult> Edit(int id, [Bind("IdEstateRental,RentBeginning,RentEnding,CostPerMonth,Deposit,IdEstateObject,IdRealtor,IdUser")] EstateRental estateRental)

{

if (id != estateRental.IdEstateRental)

{

return NotFound();

}

if (estateRental.RentEnding.HasValue && estateRental.RentBeginning > estateRental.RentEnding.Value)

{

ModelState.AddModelError(nameof(EstateRental.RentBeginning), "Дата начала аренды должна быть раньше даты окончания.");

}

if (ModelState.IsValid)

{

\_context.Update(estateRental);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["IdEstateObject"] = new SelectList(\_context.EstateObjects, "IdEstateObject", "IdEstateObject", estateRental.IdEstateObject);

ViewData["IdRealtor"] = new SelectList(\_context.Realtors, "IdRealtor", "IdRealtor", estateRental.IdRealtor);

ViewData["IdUser"] = new SelectList(\_context.Users, "IdUser", "IdUser", estateRental.IdUser);

return View(estateRental);

}

[HttpGet]

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var estateRental = await \_context.EstateRentals

.Include(e => e.IdEstateObjectNavigation)

.Include(e => e.IdRealtorNavigation)

.Include(e => e.IdUserNavigation)

.FirstOrDefaultAsync(m => m.IdEstateRental == id);

if (estateRental == null)

{

return NotFound();

}

return View(estateRental);

}

[HttpPost, ActionName("Delete")]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var estateRental = await \_context.EstateRentals.FindAsync(id);

if (estateRental != null)

{

\_context.EstateRentals.Remove(estateRental);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool EstateRentalExists(int id)

{

return \_context.EstateRentals.Any(e => e.IdEstateRental == id);

}

}

Листинг класса OwnersController

[Authorize]

public class OwnersController : Controller

{

private readonly RealEstateRentalContext \_context;

public OwnersController(RealEstateRentalContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> Index()

{

return View(await \_context.Owners.ToListAsync());

}

public IActionResult Export()

{

var allOwners = \_context.Owners.OrderBy(p => p.Surname).ToList();

var application = new Excel.Application();

application.Visible = false;

application.SheetsInNewWorkbook = allOwners.Count;

Excel.Workbook workbook = application.Workbooks.Add(Type.Missing);

for (int i = 0; i < allOwners.Count(); i++)

{

Excel.Worksheet worksheet = (Excel.Worksheet)workbook.Worksheets.Item[i + 1];

string FIO = allOwners[i].Surname + " " + allOwners[i].Name + " " + allOwners[i].Patronymic;

worksheet.Name = FIO;

worksheet.Cells[1, 1] = "Город";

worksheet.Cells[1, 2] = "Улица";

worksheet.Cells[1, 3] = "Дом";

worksheet.Cells[1, 4] = "Квартира";

worksheet.Cells[1, 5] = "Количество комнат";

worksheet.Cells[1, 6] = "Площадь";

worksheet.Cells[1, 7] = "Минимальный срок аренды";

worksheet.Cells[1, 8] = "Разрешено с детьми";

worksheet.Cells[1, 9] = "Разрешено с животными";

var estateObjects = \_context.EstateObjects

.Where(p => p.IdOwner == allOwners[i].IdOwner)

.OrderBy(p => p.ObjectType)

.GroupBy(p => p.ObjectType)

.ToList();

int rowIndex = 2;

foreach (var group in estateObjects)

{

Excel.Range range = worksheet.Range[worksheet.Cells[rowIndex, 1], worksheet.Cells[rowIndex, 10]];

range.Merge();

range.HorizontalAlignment = Excel.XlHAlign.xlHAlignCenter;

range.Font.Bold = true;

range.Value = group.Key;

rowIndex++;

foreach (var estateObject in group)

{

worksheet.Cells[rowIndex, 1] = estateObject.Town;

worksheet.Cells[rowIndex, 2] = estateObject.Street;

worksheet.Cells[rowIndex, 3] = estateObject.Home;

worksheet.Cells[rowIndex, 4] = estateObject.Flat;

worksheet.Cells[rowIndex, 5] = estateObject.NumberOfRooms;

worksheet.Cells[rowIndex, 6] = estateObject.Square;

worksheet.Cells[rowIndex, 7] = estateObject.RentalPeriod;

worksheet.Cells[rowIndex, 8] = estateObject.ChildrenAllowed ? "Да" : "Нет";

worksheet.Cells[rowIndex, 9] = estateObject.AnimalsAllowed ? "Да" : "Нет";

rowIndex++;

}

}

}

string filePath = Path.Combine(Directory.GetCurrentDirectory(), "Owners.xlsx");

if (System.IO.File.Exists(filePath))

{

System.IO.File.Delete(filePath);

}

workbook.SaveAs(filePath);

workbook.Close();

application.Quit();

var fileStream = new FileStream(filePath, FileMode.Open, FileAccess.Read);

return File(fileStream, "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet", "Owners.xlsx");

}

public IActionResult ExportSuccess() {

return View("~/Views/Shared/ExportSuccess.cshtml");

}

[HttpGet]

public IActionResult Create()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Create([Bind("IdOwner,Surname,Name,Patronymic,PhoneNumber,Town,Street,Home,Flat,EmailAddress")] Owner owner)

{

if (ModelState.IsValid)

{

\_context.Add(owner);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(owner);

}

[HttpGet]

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var owner = await \_context.Owners.FindAsync(id);

if (owner == null)

{

return NotFound();

}

return View(owner);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("IdOwner,Surname,Name,Patronymic,PhoneNumber,Town,Street,Home,Flat,EmailAddress")] Owner owner)

{

if (id != owner.IdOwner)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(owner);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!OwnerExists(owner.IdOwner))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(owner);

}

[HttpGet]

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var owner = await \_context.Owners

.FirstOrDefaultAsync(m => m.IdOwner == id);

if (owner == null)

{

return NotFound();

}

return View(owner);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var owner = await \_context.Owners.FindAsync(id);

if (owner != null)

{

\_context.Owners.Remove(owner);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool OwnerExists(int id)

{

return \_context.Owners.Any(e => e.IdOwner == id);

}

}

Листинг класса RealtorsController

[Authorize]

public class RealtorsController : Controller

{

private readonly RealEstateRentalContext \_context;

public RealtorsController(RealEstateRentalContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> Index()

{

var realEstateRentalContext = \_context.Realtors.Include(r => r.IdEstateAgencyNavigation);

return View(await realEstateRentalContext.ToListAsync());

}

[HttpGet]

public IActionResult Create()

{

ViewData["IdEstateAgency"] = new SelectList(\_context.EstateAgencies, "IdEstateAgency", "IdEstateAgency");

return View();

}

[HttpPost]

public async Task<IActionResult> Create([Bind("IdRealtor,Surname,Name,Patronymic,PhoneNumber,EmailAddress,IdEstateAgency")] Realtor realtor)

{

if (ModelState.IsValid)

{

\_context.Add(realtor);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["IdEstateAgency"] = new SelectList(\_context.EstateAgencies, "IdEstateAgency", "IdEstateAgency", realtor.IdEstateAgency);

return View(realtor);

}

[Authorize(Roles = "Admin,AgencyManager")]

[HttpGet]

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var realtor = await \_context.Realtors.FindAsync(id);

if (realtor == null)

{

return NotFound();

}

ViewData["IdEstateAgency"] = new SelectList(\_context.EstateAgencies, "IdEstateAgency", "IdEstateAgency", realtor.IdEstateAgency);

return View(realtor);

}

[HttpPost]

public async Task<IActionResult> Edit(int id, [Bind("IdRealtor,Surname,Name,Patronymic,PhoneNumber,EmailAddress,IdEstateAgency")] Realtor realtor)

{

if (id != realtor.IdRealtor)

{

return NotFound();

}

ViewData["IdEstateAgency"] = new SelectList(\_context.EstateAgencies, "IdEstateAgency", "IdEstateAgency", realtor.IdEstateAgency);

if (ModelState.IsValid)

{

\_context.Update(realtor);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(realtor);

}

[HttpGet]

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var realtor = await \_context.Realtors

.Include(r => r.IdEstateAgencyNavigation)

.FirstOrDefaultAsync(m => m.IdRealtor == id);

if (realtor == null)

{

return NotFound();

}

return View(realtor);

}

[HttpPost, ActionName("Delete")]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var realtor = await \_context.Realtors.FindAsync(id);

if (realtor != null)

{

\_context.Realtors.Remove(realtor);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool RealtorExists(int id)

{

return \_context.Realtors.Any(e => e.IdRealtor == id);

}

}

Листинг класса UsersController

[Authorize]

public class UsersController : Controller

{

private readonly RealEstateRentalContext \_context;

public UsersController(RealEstateRentalContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> Index()

{

return View(await \_context.Users.ToListAsync());

}

[HttpGet]

public IActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("IdUser,Surname,Name,Patronymic,PhoneNumber,EmailAddress")] User user)

{

if (ModelState.IsValid)

{

\_context.Add(user);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(user);

}

[Authorize(Roles = "Admin")]

[HttpGet]

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var user = await \_context.Users.FindAsync(id);

if (user == null)

{

return NotFound();

}

return View(user);

}

[HttpPost]

public async Task<IActionResult> Edit(int id, [Bind("IdUser,Surname,Name,Patronymic,PhoneNumber,EmailAddress")] User user)

{

if (id != user.IdUser)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(user);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!UserExists(user.IdUser))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(user);

}

[HttpGet]

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var user = await \_context.Users

.FirstOrDefaultAsync(m => m.IdUser == id);

if (user == null)

{

return NotFound();

}

return View(user);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var user = await \_context.Users.FindAsync(id);

if (user != null)

{

\_context.Users.Remove(user);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool UserExists(int id)

{

return \_context.Users.Any(e => e.IdUser == id);

}

}

приложение Б

Листинг функции Export для таблицы EstateAgencies

public IActionResult Export()

{

var allAgencies = \_context.EstateAgencies.OrderBy(p => p.AgencyName).ToList();

var application = new Excel.Application();

application.Visible = false;

application.SheetsInNewWorkbook = allAgencies.Count;

Excel.Workbook workbook = application.Workbooks.Add(Type.Missing);

for (int i = 0; i < allAgencies.Count(); i++)

{

Excel.Worksheet worksheet = (Worksheet)application.Worksheets.Item[i + 1];

worksheet.Name = allAgencies[i].AgencyName;

worksheet.Cells[1, 1] = "Фамилия";

worksheet.Cells[1, 2] = "Имя";

worksheet.Cells[1, 3] = "Отчество";

worksheet.Cells[1, 4] = "Номер телефона";

worksheet.Cells[1, 5] = "E-mail";

worksheet.Cells[1, 6] = "Количество сделок";

var realtors = \_context.Realtors

.Where(p => p.IdEstateAgency == allAgencies[i].IdEstateAgency)

.OrderBy(p => p.Surname)

.ToList();

int rowIndex = 2;

int totalDeals = 0;

foreach (var realtor in realtors)

{

int dealsCount = \_context.EstateRentals.Count(e => e.IdRealtor == realtor.IdRealtor);

totalDeals += dealsCount;

worksheet.Cells[rowIndex, 1] = realtor.Surname;

worksheet.Cells[rowIndex, 2] = realtor.Name;

worksheet.Cells[rowIndex, 3] = realtor.Patronymic;

worksheet.Cells[rowIndex, 4] = realtor.PhoneNumber.ToString();

worksheet.Cells[rowIndex, 5] = realtor.EmailAddress;

worksheet.Cells[rowIndex, 6] = dealsCount;

rowIndex++;

}

Excel.Range range = worksheet.Range[worksheet.Cells[rowIndex, 1], worksheet.Cells[rowIndex, 5]];

range.Merge();

range.HorizontalAlignment = Excel.XlHAlign.xlHAlignCenter;

range.Font.Bold = true;

range.Value = "Итого сделок:";

worksheet.Cells[rowIndex, 6] = totalDeals;

}

string filePath = Path.Combine(Directory.GetCurrentDirectory(), "Agencies.xlsx");

if (System.IO.File.Exists(filePath))

{

System.IO.File.Delete(filePath);

}

workbook.SaveAs(filePath);

workbook.Close();

application.Quit();

// Отправка файла пользователю

var fileStream = new FileStream(filePath, FileMode.Open, FileAccess.Read);

return File(fileStream, "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet", "Agencies.xlsx");

}

Листинг функции Export для таблицы Owners

public IActionResult Export()

{

var allOwners = \_context.Owners.OrderBy(p => p.Surname).ToList();

var application = new Excel.Application();

application.Visible = false;

application.SheetsInNewWorkbook = allOwners.Count;

Excel.Workbook workbook = application.Workbooks.Add(Type.Missing);

for (int i = 0; i < allOwners.Count(); i++)

{

Excel.Worksheet worksheet = (Excel.Worksheet)workbook.Worksheets.Item[i + 1];

string FIO = allOwners[i].Surname + " " + allOwners[i].Name + " " + allOwners[i].Patronymic;

worksheet.Name = FIO;

worksheet.Cells[1, 1] = "Город";

worksheet.Cells[1, 2] = "Улица";

worksheet.Cells[1, 3] = "Дом";

worksheet.Cells[1, 4] = "Квартира";

worksheet.Cells[1, 5] = "Количество комнат";

worksheet.Cells[1, 6] = "Площадь";

worksheet.Cells[1, 7] = "Минимальный срок аренды";

worksheet.Cells[1, 8] = "Разрешено с детьми";

worksheet.Cells[1, 9] = "Разрешено с животными";

var estateObjects = \_context.EstateObjects

.Where(p => p.IdOwner == allOwners[i].IdOwner)

.OrderBy(p => p.ObjectType)

.GroupBy(p => p.ObjectType)

.ToList();

int rowIndex = 2;

foreach (var group in estateObjects)

{

Excel.Range range = worksheet.Range[worksheet.Cells[rowIndex, 1], worksheet.Cells[rowIndex, 10]];

range.Merge();

range.HorizontalAlignment = Excel.XlHAlign.xlHAlignCenter;

range.Font.Bold = true;

range.Value = group.Key;

rowIndex++;

foreach (var estateObject in group)

{

worksheet.Cells[rowIndex, 1] = estateObject.Town;

worksheet.Cells[rowIndex, 2] = estateObject.Street;

worksheet.Cells[rowIndex, 3] = estateObject.Home;

worksheet.Cells[rowIndex, 4] = estateObject.Flat;

worksheet.Cells[rowIndex, 5] = estateObject.NumberOfRooms;

worksheet.Cells[rowIndex, 6] = estateObject.Square;

worksheet.Cells[rowIndex, 7] = estateObject.RentalPeriod;

worksheet.Cells[rowIndex, 8] = estateObject.ChildrenAllowed ? "Да" : "Нет";

worksheet.Cells[rowIndex, 9] = estateObject.AnimalsAllowed ? "Да" : "Нет";

rowIndex++;

}

}

}

string filePath = Path.Combine(Directory.GetCurrentDirectory(), "Owners.xlsx");

if (System.IO.File.Exists(filePath))

{

System.IO.File.Delete(filePath);

}

workbook.SaveAs(filePath);

workbook.Close();

application.Quit();

var fileStream = new FileStream(filePath, FileMode.Open, FileAccess.Read);

return File(fileStream, "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet", "Owners.xlsx");

}