Частное учреждение образования

«Колледж бизнеса и права»

«Разработка веб ‒ приложения для автоматизации работы АВТОСАЛОНА «West Motors»

Пояснительная записка

к курсовому проекту по дисциплине

«Конструирование программ и языки программирования»

КП Т.318021.401

Руководитель проекта (А.В.Кривошеина)

Учащийся (П.В.Стадуб)

2025

Изм.

Лист

№ докум.

Подпись

Датаа

Лист

3

КП Т.318021.401

Разраб.

Стадуб П. В.

Провер.

Кривошеина А.В

Реценз.

Н. Контр.

Утверд.

*Разработка веб-приложения для автоматизации работы автосалона «WEST MOTORS»*

Лит.

Листов

КБиП

у

СОДЕРЖАНИЕ

[Введение 4](#_Toc200978271)

[1 Описание задачи 6](#_Toc200978272)

[1.1 Анализ предметной области 6](#_Toc200978273)

[1.2 Постановка задачи 7](#_Toc200978274)

[2 Проектирование веб-приложения 9](#_Toc200978275)

[2.1 Проектирование модели 9](#_Toc200978276)

[2.2 Требования к веб-приложению 10](#_Toc200978277)

[2.3 Структура веб-приложения 11](#_Toc200978278)

[2.4 Проектирование макета веб-приложения 12](#_Toc200978279)

[2.5 Программно-технические средства, необходимые для разработки приложения 13](#_Toc200978280)

[2.6 Защита и сохранность данных 15](#_Toc200978281)

[2.7 Организация и ведение информационной базы (модели) 15](#_Toc200978282)

[3 Реализация веб-приложения 22](#_Toc200978283)

[3.1 Описание разделов веб-приложения 22](#_Toc200978284)

[3.2 Разработка административной части приложения 23](#_Toc200978285)

[3.3 Разработка клиентской части приложения 24](#_Toc200978286)

[3.4 Описание используемых функций и процедур 25](#_Toc200978287)

[3.5 Функциональное тестирование 28](#_Toc200978288)

[4 Применение 30](#_Toc200978289)

[4.1 Назначение веб-приложения 30](#_Toc200978290)

[4.2 Программно-аппаратное обеспечение сервера и клиента 30](#_Toc200978291)

[Заключение 32](#_Toc200978292)

[Список информационных источников 33](#_Toc200978293)

[Приложения А Ментальная карта 34](#_Toc200978294)

[Приложение Б Логическая и физическая структура 35](#_Toc200978295)

[Приложение В Макеты страниц веб – приложения 37](#_Toc200978296)

[Приложение Г Код модулей веб – приложения 46](#_Toc200978297)

[Приложение Д Результаты работы веб – приложения 67](#_Toc200978298)

# Введение

Целью проекта на тему «Разработка веб-приложения по автоматизации работы автосалона «West Motors»» является создание эффективной, надёжной и удобной информационной системы, предназначенной для автоматизации ключевых процессов, связанных с продажей и учётом автомобилей в рамках сети автосалонов. Проект направлен на упрощение взаимодействия с клиентами, повышение качества обслуживания, сокращение времени обработки сделок и централизованное управление ассортиментом автомобилей и заключенными сделками через единое веб-приложение.

Разрабатываемое решение обеспечит пользователям интуитивно понятный интерфейс, включающий просмотр и фильтрацию ассортимента автомобилей, добавление новых транспортных средств, управление информацией о клиентах, создание и оформление сделок купли-продажи, а также доступ к личному кабинету с историей продаж и управлением профилем. Для внутреннего использования приложение поддерживает экспорт договоров купли-продажи в формат Word, что упрощает документооборот и анализ данных.

Актуальность проекта обусловлена необходимостью повышения эффективности и прозрачности работы торговых точек, оптимизации бизнес-процессов и внедрения цифровых технологий в деятельность сети автосалонов. В условиях растущей конкуренции и цифровизации автомобильного рынка автоматизация автосалона «West Motors» позволит значительно улучшить клиентский сервис и оперативность обработки заказов.

Решение поставленных задач подробно рассмотрено в пояснительной записке, которая состоит из четырёх основных разделов и описывает этапы анализа, проектирования, реализации и применения веб-приложения.

В первом разделе «Описание задачи» описываются цели автоматизации, анализируется предметная область (процессы продажи автомобилей, учет клиентов и машин), выделяются основные проблемы текущих процессов (например, ручной документооборот, сложности с отслеживанием статуса сделок), формулируются функциональные и нефункциональные требования к системе, а также рассматриваются существующие аналоги и их недостатки.

Во втором разделе «Проектирование системы» представлена архитектура веб-приложения, структура базы данных, логика взаимодействия между компонентами, схемы пользовательского интерфейса и особенности обеспечения безопасности данных. Описываются сущности (например, Автомобили, Клиенты, Сделки, Пользователи), связи между таблицами и схемы обработки пользовательских сценариев.

В третьем разделе «Описание и реализация программного средства» рассматриваются технологии и инструменты разработки (например, ASP.NET Core MVC, SQL Server, HTML/CSS, JavaScript). Подробно описываются модули: регистрация и авторизация, управление автомобилями, управление клиентами, создание и оформление сделок, история продаж, а также функциональность экспорта договоров купли-продажи и отчётов.

Четвёртый раздел «Применение» посвящён эксплуатации веб-приложения как для клиентов, так и для сотрудников компании. Рассматриваются сценарии использования, особенности адаптивного интерфейса, поддержка работы на различных устройствах, защита персональных данных и организация дальнейшего сопровождения системы.

В заключении анализируются результаты работы, степень выполнения поставленных задач, эффективность внедрения системы и направления для возможного расширения функциональности.

Список использованных источников включает нормативные документы, справочную литературу и интернет-ресурсы, использованные при проектировании и разработке системы.

В Приложении А представлена ментальная карта.

В Приложении Б представлена логическая и физическая структура.

В Приложении В макеты страниц веб-приложения.

В Приложении Г код модулей веб-приложений.

В Приложении Д результаты работы веб-приложения.

Графическая часть включает диаграмму классов, диаграмму развертывания, диаграмму прецедентов, структуру базы данных и архитектурную схему веб-приложения «West Motors».

# Описание задачи

## Анализ предметной области

Темой проекта является разработка веб-приложения «West Motors», предназначенного для автоматизации процессов в автосалоне, реализующем автомобили. Предметной областью выступает сфера автомобильного бизнеса, в частности — организация онлайн-продаж и учёта автомобилей с возможностью администрирования, управления сделками и взаимодействия с клиентами.

Основными объектами предметной области являются:

* автомобили с техническими характеристиками и изображениями;
* фильтры для удобного поиска;
* учётные записи пользователей (менеджеры, администраторы);
* сделки с возможностью отслеживания;
* интерфейс администраторов для управления контентом сайта.

Бизнес-процессы, реализуемые в системе, охватывают полный цикл продажи автомобиля:

пользователь (менеджер) проходит регистрацию или авторизацию, просматривает каталог автомобилей, применяет фильтры по характеристикам (марка, модель, цена), оформляет сделку с указанием клиента, автомобиля и способа оплаты. После оформления сделка сохраняется в системе и доступна для просмотра в истории продаж. Администратор получает доступ к управлению автомобилями, сделками, пользователями и другими справочниками через панель администратора.

Необходимость компьютерной обработки обусловлена большим объёмом данных, ежедневной динамикой сделок и потребностью в быстрой, безопасной и централизованной обработке информации. Веб-приложение «West Motors» автоматизирует хранение и обработку сделок, взаимодействие с пользователями и аналитическую отчетность. Безопасность обеспечивается через систему авторизации, разграничение ролей (менеджер/админ) и защищённое хранение данных в базе.

Ключевые задачи, подлежащие реализации:

* проектирование и реализация базы данных для хранения информации об автомобилях, сделках, клиентах, пользователях;
* реализация системы регистрации и авторизации с защитой пользовательских данных;
* создание API-контроллеров для работы с автомобилями, клиентами, сделками, авторизацией и админ-панелью;
* разработка адаптивного фронтенда на подходящей технологии (HTML, CSS) с поддержкой базовых пользовательских сценариев;
* реализация функции экспорта договоров купли-продажи в Word-документы для ведения отчётности;

Целевую аудиторию составляют:

* менеджеры по продажам автосалона;
* администраторы и контент-менеджеры сайта;

На современном рынке существует множество CRM-систем и инструментов для автосалонов, однако они не всегда обеспечивают полную интеграцию с внутренними процессами (складской учет, документооборот, отчетность) или имеют избыточный функционал. Разрабатываемая система «West Motors» направлена на устранение этих ограничений за счёт адаптивной архитектуры, расширенного функционала и модульного API. Она предоставляет пользователям современный интерфейс, интеграцию с бэк-офисом и возможности дальнейшего масштабирования сети.

## Постановка задачи

Основная задача разработки веб-приложения West Motors заключается в создании современного и удобного инструмента для автоматизации ключевых бизнес-процессов, связанных с продажей автомобилей, управлением сделками и взаимодействием с клиентами в рамках автосалона.

С помощью разрабатываемого программного обеспечения решаются следующие задачи в рамках предметной области:

* автоматизация отображения и фильтрации автомобилей. Приложение предоставляет пользователям удобный каталог транспортных средств с возможностью фильтрации по пробегу, цене и ключевым словам. Это позволяет ускорить поиск нужного автомобиля и улучшает пользовательский опыт;
* формирование и управление сделками. Менеджеры могут создавать новые сделки, редактировать их содержимое, выбирать количество единиц (например, при покупке нескольких автомобилей для корпоративных клиентов), а также оформлять сделки с указанием способа оплаты и условий получения автомобиля;
* администрирование через панель администратора. Для руководителей и администраторов предусмотрен интерфейс для управления автомобилями, сделками, клиентами и пользователями, а также возможность выгрузки отчётов по продажам в формате Word и генерации договоров купли-продажи;
* безопасность и защита данных. Использование авторизации, разграничения ролей и безопасного хранения информации в базе данных позволяет защитить персональные данные пользователей и обеспечить соответствие современным стандартам безопасности;
* адаптивный и интуитивно понятный интерфейс. Благодаря выбранной технологии фронтенда, CSS-модулям и базовой адаптивной вёрстке интерфейс корректно отображается на различных устройствах, обеспечивая удобную навигацию как с компьютеров, так и с мобильных устройств;

На основе анализа бизнес-процессов в сфере продажи автомобилей были выделены следующие функции, которые подлежат автоматизации в рамках веб-приложения:

* каталогизация и фильтрация автомобилей по характеристикам;
* создание и оформление сделки купли-продажи с возможностью выбора способа оплаты;
* регистрация и авторизация пользователей;
* управление клиентской базой данных;
* хранение истории сделок и генерация договоров купли-продажи (в формате Word);
* управление ассортиментом автомобилей, сделками, клиентами и пользователями через админ-панель;
* интеграция с системами учёта и потенциально с CRM-системами;
* обеспечение безопасности пользовательских данных через авторизацию и разграничение ролей.

На рынке присутствуют такие аналоги, как CRM-системы для автобизнеса (AutoCRM, Сармакс), специализированные онлайн-площадки для продажи автомобилей (Kufar Авто, Av.by, Drom.ru). Однако большинство из них не предусматривает полную интеграцию с внутренней системой управления конкретного автосалона, не всегда предоставляет гибкий экспорт всех необходимых отчётов в Word и часто ограничено в гибкости интерфейса и возможности персонализированного взаимодействия с внутренними пользователями. Разработка собственного веб-приложения West Motors позволяет учесть все особенности внутренней работы сети автосалонов, реализовать необходимые функции в единой системе и обеспечить удобство как для сотрудников, так и для руководства.

# Проектирование веб-приложения

## Проектирование модели

Диаграмма прецедентов веб-приложения «West Motors» отражает структуру взаимодействия пользователей с системой, включая администратора и менеджера. Она демонстрирует все основные действия, доступные каждому типу пользователя, а также зависимости между ними.

Менеджер начинает взаимодействие с системой через регистрацию и авторизацию, после чего получает доступ к ключевому функционалу: просмотру каталога автомобилей, поиску и фильтрации, оформлению сделки и добавлению клиента к сделке. Добавление автомобиля в сделку встроено в процесс оформления сделки и связано через механизм include. Также к оформлению сделки может быть подключён выбор способа оплаты, реализованный через extend.

Администратор имеет расширенные полномочия: управление сделками, каталогом автомобилей и менеджерами. Эти действия позволяют администрировать систему, контролировать содержимое и поведение пользователей.

Диаграмма прецедентов представлена на листе 1 графической части.

Диаграмма классов отображает внутреннюю структуру приложения «West Motors» и показывает основные сущности, их свойства и связи между ними. Она необходима для построения базы данных, а также для описания логики работы системы.

Класс Пользователь содержит поля: Id, UserName, Email, PasswordHash, FullName, Position, ContactInfo, SoldCarsCount, ClientId. Один Пользователь (являющийся менеджером или администратором) может оформлять множество сделок и быть механиком в записях сервиса. Также реализованы методы: register(), login(), logout(), updateProfile().

Класс Клиент содержит поля: Id, FullName, PassportData, ContactInfo, Address, Preferences. Один Клиент (покупатель) может иметь множество сделок и заявок. Методы: addClient(), updateClient(), getClientDetails().

Класс Автомобиль включает: Id, Brand, Model, ManufactureYear, Mileage, VIN, Price, Condition, ArrivalDate, PhotoUrl. Методы: updateStatus(), getDetails(). Связан со сделками, записями автосервиса и заявками.

Класс ЗаписьАвтоСервиса: Id, CarId, Car, WorkType, RepairCost, MechanicId, Mechanic, ServiceDate. Методы: calculateRepairCost(), updateStatus(). Один автомобиль может иметь множество записей автосервиса.

Класс Сделка: Id, CarId, Car, SellerId, Seller, BuyerId, Buyer, DealDate, FinalCost, PaymentMethod. Методы: calculateTotal(), updateStatus(), cancelDeal(). Один Менеджер может иметь множество сделок.

Диаграмма классов представлена на листе 2 графической части.

Диаграмма развёртывания веб-приложения «West Motors» иллюстрирует архитектуру системы и распределение компонентов между клиентской, серверной и облачной частями. Она отражает потоки данных, направления обмена информацией и назначение каждого узла.

На стороне клиентского узла расположен веб-браузер с адаптивным интерфейсом, реализованным с помощью HTML и CSS. Через протокол HTTPS осуществляется обмен с сервером.

Серверный узел включает в себя веб-сервер, на котором размещено серверное приложение на ASP.NET Core. Логика сервера взаимодействует с базой данных через SQL-запросы.

База данных развернута на SQL Server и хранит информацию о пользователях, товарах, заказах, корзинах и категориях.

Для размещения проекта используется облачный узел — Vercel или Azure, куда осуществляется деплой серверного и клиентского кода.

Диаграмма развёртывания представлена на листе 3 графической части.

## Требования к веб-приложению

При разработке веб-приложения для автосалона «West Motors» необходимо учитывать широкий спектр требований, касающихся стилистического, графического, функционального и технического аспектов. Крайне важно обеспечить единообразие визуального оформления, соответствующего фирменному стилю, и интуитивную структуру пользовательского интерфейса. Интерфейс должен быть минималистичным, удобным и способствующим быстрому восприятию информации.

Графический дизайн веб-приложения должен отличаться визуальной чистотой, современностью и доступностью. Применение качественных изображений, иконок и акцентных элементов повышает визуальную привлекательность и уровень доверия со стороны пользователей. Элементы управления (каталог, фильтры, кнопки, меню) должны быть легко различимы, а логика взаимодействия — интуитивно понятной. Все функциональные блоки должны быть визуально разграничены. Цветовая палитра выдержана в темных (черный, серый) и акцентных (красный) тонах, что подчёркивает надёжный и современный стиль автомобильного бизнеса.

Шрифт, применяемый в интерфейсе — Montserrat для основного текста и Playfair Display для заголовков, как современные, геометричные и легко читаемые. Рекомендуется ограничить палитру до двух-трёх начертаний. Цвет текста должен чётко выделяться на фоне, заголовки должны быть визуально доминирующими, а важные элементы — выделяться цветом или иконками.

Веб-приложение должно корректно работать во всех современных браузерах (Chrome, Firefox, Safari, Edge) и адаптироваться под устройства разного типа: ПК, планшеты, смартфоны. Это обеспечит максимальное удобство и доступность для всех категорий пользователей вне зависимости от типа устройства.

Контент веб-приложения должен быть актуальным, структурированным и легко воспринимаемым. В каталоге автомобилей информация должна представляться кратко и чётко: марка, модель, изображение, цена, VIN-номер. Карточка автомобиля включает расширенные характеристики и историю. Система управления контентом должна позволять легко добавлять, редактировать и удалять автомобили, а также управлять текстовыми разделами сайта (профиль, политика конфиденциальности, о компании).

Также важно обеспечить надёжную защиту персональных данных, включая безопасную регистрацию, авторизацию и разграничение прав доступа.

Все ключевые функции должны быть доступны на каждой странице, а информация — отображаться логично и удобно для восприятия. Обязательна реализация функций поиска и фильтрации автомобилей, а также управление сделками, заявками и историей взаимодействия с клиентами.

Для разработки веб-приложения «West Motors» определены следующие требования к визуальному оформлению, обеспечивающие единообразие и привлекательность интерфейса:

Шрифт — Montserrat:

* «Montserrat Regular» — для основного текста и большинства элементов;
* «Montserrat Medium» — для навигации и подписей;
* «Montserrat SemiBold/Bold» — используется для заголовков и акцентов.

Рекомендуемые размеры шрифтов для интерфейса:

* заголовки 1 уровня 24px
* заголовки 2 уровня 20px
* заголовки 3 уровня 18px
* основной текст: 16px
* подписи, технический текст: 14px

Цветовая палитра интерфейса:

* основной текст чёрный #000000, белый #FFFFFF
* основной фон светло-серый #F5F5F5
* хедер/меню тёмный фиолетово-серый #343a40
* фон карточек и блоков — белый #FFFFFF
* акценты и кнопки: красный #FF6B6B

Такая система визуального оформления позволяет поддерживать строгую и понятную структуру интерфейса, а также способствует созданию единообразного, эстетически привлекательного облика веб-приложения West Motors.

## Структура веб-приложения

Структура веб-приложения представляет собой упорядоченную схему, показывающую, как различные компоненты и функциональные элементы взаимодействуют между собой, а также каким образом они организованы. Она определяет способ отображения информации пользователям и взаимодействие пользователей с приложением. Структура веб-приложения (ментальная карта) представлена на приложении А на рисунке А.1

## Проектирование макета веб-приложения

Веб-сайт был спроектирован в онлайн-сервисе Figma, который предоставляет мощные инструменты для прототипирования и проектирования интерфейсов. С помощью Figma были созданы интерактивные элементы, включая кнопки, формы, меню и векторные изображения, что позволило реализовать современный и удобный дизайн.

Сайт использует шрифты Montserrat и Playfair Display, которые обеспечивают отличную читаемость и поддерживают современный стиль дизайна. Шрифт Montserrat применяется для большинства текстов, включая описания и подписи к кнопкам, а Playfair Display используется для заголовков.

Дизайн сайта имеет в основном темные (черный, серый) цвета с яркими акцентными элементами красного цвета. Он был спроектирован таким образом, чтобы удовлетворить интересы пользователей. Это улучшает качество пользовательского опыта.

Веб-сайт был спроектирован в онлайн-сервисе Figma, который предоставляет мощные инструменты для прототипирования и проектирования интерфейсов. С помощью Figma были созданы интерактивные элементы, включая кнопки, формы, меню и векторные изображения, что позволило реализовать современный и удобный дизайн.

Сайт использует шрифты Montserrat и Playfair Display, которые обеспечивают отличную читаемость и поддерживают современный стиль дизайна. Шрифт Montserrat применяется для большинства текстов, включая описания и подписи к кнопкам, а Playfair Display используется для заголовков.

Дизайн сайта имеет в основном темные (черный, серый) цвета с яркими акцентными элементами красного цвета. Он был спроектирован таким образом, чтобы удовлетворить интересы пользователей. Это улучшает качество пользовательского опыта.

Главная страница сайта имеет размер 1920 пикселей по ширине и 4899 по высоте.

Раздел «Каталог автомобилей» имеет размер 1920 пикселей по ширине и 2940 по высоте.

Раздел «Карточка автомобиля» имеет размер 1920 пикселей по ширине и 2660 по высоте.

На приложении В на рисунке В.1 представлен макет главной страницы, на которой размещена основная информация о веб-приложении.

На приложении В на рисунке В.2 представлен макет страницы «Каталог автомобилей», содержащей фильтры и карточки автомобилей.

На приложении В на рисунке В.3 представлен макет страницы «Карточка автомобиля», содержащей описание, характеристики и отзывы.

## Программно-технические средства, необходимые для разработки приложения

Для реализации веб-приложения West Motors App используется современный стек технологий, включающий языки программирования, фреймворки, библиотеки и инструменты разработки, которые обеспечивают высокую производительность, надёжность и адаптивность системы. Приложение ориентировано на клиент-серверную архитектуру с поддержкой сетевого взаимодействия, регистрации пользователей, отображения автомобилей и обработки сделок и заявок в режиме онлайн.

С помощью выбранных технологий решаются следующие задачи в рамках реализации проекта:

* построение серверной логики приложения. Для разработки backend-части используется язык программирования C# с фреймворком ASP.NET Core MVC. Он выбран благодаря своей высокой надёжности, масштабируемости, встроенной поддержке безопасной авторизации и удобной интеграции с базами данных. ASP.NET Core позволяет строить REST API, что идеально подходит для взаимодействия с клиентской частью;
* создание динамичного интерфейса. Для frontend-части применяются языки разметки и дизайна HTML и CSS, а также язык программирования JavaScript. HTML задаёт структуру контента, CSS (в частности, CSS-модули) позволяют применять адаптивные стили, задавать цветовые схемы, шрифты и элементы дизайна, а JavaScript позволяет создавать интерактивные элементы и динамическое поведение интерфейса. Такой выбор обеспечивает кроссбраузерность и высокую производительность при работе с изменяемыми данными;
* взаимодействие между клиентом и сервером. Для отправки HTTP-запросов и получения данных используется библиотека Axios. Она проста в использовании и обеспечивает эффективную работу с API: получение списка товаров, регистрация, оформление заказа и т.д.;
* работа с базой данных. Для хранения данных о пользователях, товарах, заказах и избранном используется SQL Server, один из самых стабильных и производительных реляционных СУБД. С ним работает ORM-фреймворк Entity Framework Core, упрощающий доступ к данным и сокращающий количество SQL-запросов за счёт модели сущностей. Такой подход позволяет работать с БД на уровне C#-объектов;
* организация среды разработки. Для клиентской и сервисной части применяется Visual Studio — лёгкий, но функциональный редактор, который идеально подходит для ASP.NET-проектов. Инструмент поддерживает Git, расширения и плагины;
* тестирование API и отладка. Для проверки и документирования API-запросов используются Unit-тесты. Это позволяет автоматически тестировать маршруты, аутентификацию и отправку данных, обеспечивая уверенность в корректной работе backend-сервиса.

Приложение West Motors реализует полноценную сетевую архитектуру. Пользователи (менеджеры и администраторы) взаимодействуют с серверной частью через интернет с помощью HTTPS, получая актуальные данные из базы, отправляя запросы и обновляя информацию в реальном времени. Такая архитектура позволяет использовать приложение как на ПК, так и на мобильных устройствах, обеспечивая круглосуточный доступ к функционалу: каталогу автомобилей, оформлению сделок, профилю, управлению клиентами и заявками, записям сервиса.

Выбор всех вышеуказанных технологий обусловлен их стабильностью, широким сообществом, большим количеством готовых решений, а также возможностью быстрой разработки, масштабирования и лёгкой поддержки проекта в будущем. Благодаря такому подходу обеспечивается высокая производительность, безопасность и современный пользовательский опыт веб-приложения West Motors.

## Защита и сохранность данных

Веб-приложение West Motors разработано с соблюдением современных требований безопасности, направленных на защиту пользовательских данных и предотвращение несанкционированного доступа.

Для обеспечения конфиденциальности реализована система регистрации и аутентификации, основанная на использовании уникальных логинов и паролей. Пользовательский профиль (менеджера или администратора) связан с конкретной сессией и идентифицируется в базе данных, что исключает возможность доступа к персональным данным со стороны третьих лиц.

Данные о пользователях, клиентах, автомобилях, сделках, заявках и записях автосервиса сохраняются в базе данных SQL Server, доступ к которой осуществляется через защищённые серверные запросы с использованием API. Вся передача данных между клиентом и сервером осуществляется по защищённому протоколу HTTPS, что предотвращает перехват или подмену информации.

Для взаимодействия с приложением пользователь предоставляет только минимально необходимые разрешения:

* доступ к интернету — используется для отправки запросов к API при регистрации, авторизации, управлении автомобилями, клиентами, сделками, заявками и записями автосервиса;
* сохранение токенов аутентификации — применяется для обеспечения постоянной сессии и упрощения повторного входа;

Серверная часть приложения написана на ASP.NET Core с реализацией механизма авторизации, включающего шифрование паролей и использование токенов безопасности. Это обеспечивает надёжную защиту пользовательских данных, а также позволяет эффективно разграничивать доступ к различным ресурсам и функциям приложения в соответствии с ролями (Администратор, Менеджер). Реализованная система безопасности делает взаимодействие с приложением безопасным и устойчивым к несанкционированным действиям.

Разграничение прав пользователей по ролям, шифрование данных при хранении и передаче, а также строгая структура API обеспечивают высокий уровень защиты пользовательской информации. Вся архитектура приложения спроектирована с учётом принципов «privacy by design» и «minimum access», что минимизирует риски утечек и потерь данны.

## Организация и ведение информационной базы (модели)

Диаграмма «сущность-связь» представляет логическую модель базы данных веб-приложения по автоматизации работы автосалона «West Motors» и отражает основные сущности, их атрибуты и связи между ними. Она используется на этапе проектирования системы для определения структуры хранения данных и логики их взаимодействия.

Сущность «Пользователь» (ApplicationUser) содержит атрибуты Id, UserName, Email, PasswordHash, FullName, Position, ContactInfo, SoldCarsCount, ClientId. Пользователь является сотрудником (менеджером или администратором) и используется для идентификации при входе в систему, оформления сделок и выполнения сервисных работ. Связь с другими таблицами позволяет отслеживать активность и историю сделок, а также выполненных сервисных работ конкретного пользователя.

Сущность «Клиент» содержит атрибуты Id, FullName, PassportData, ContactInfo, Address, Preferences. Один Клиент может быть покупателем во множестве сделок и создавать множество заявок.

Сущность «Автомобиль» (Car) включает атрибуты Id, Brand, Model, ManufactureYear, Mileage, VIN, Price, Condition, ArrivalDate, PhotoUrl. Это основная сущность, отображающая все автомобили, доступные в каталоге автосалона. Каждый автомобиль может фигурировать во множестве сделок, заявок и записей автосервиса.

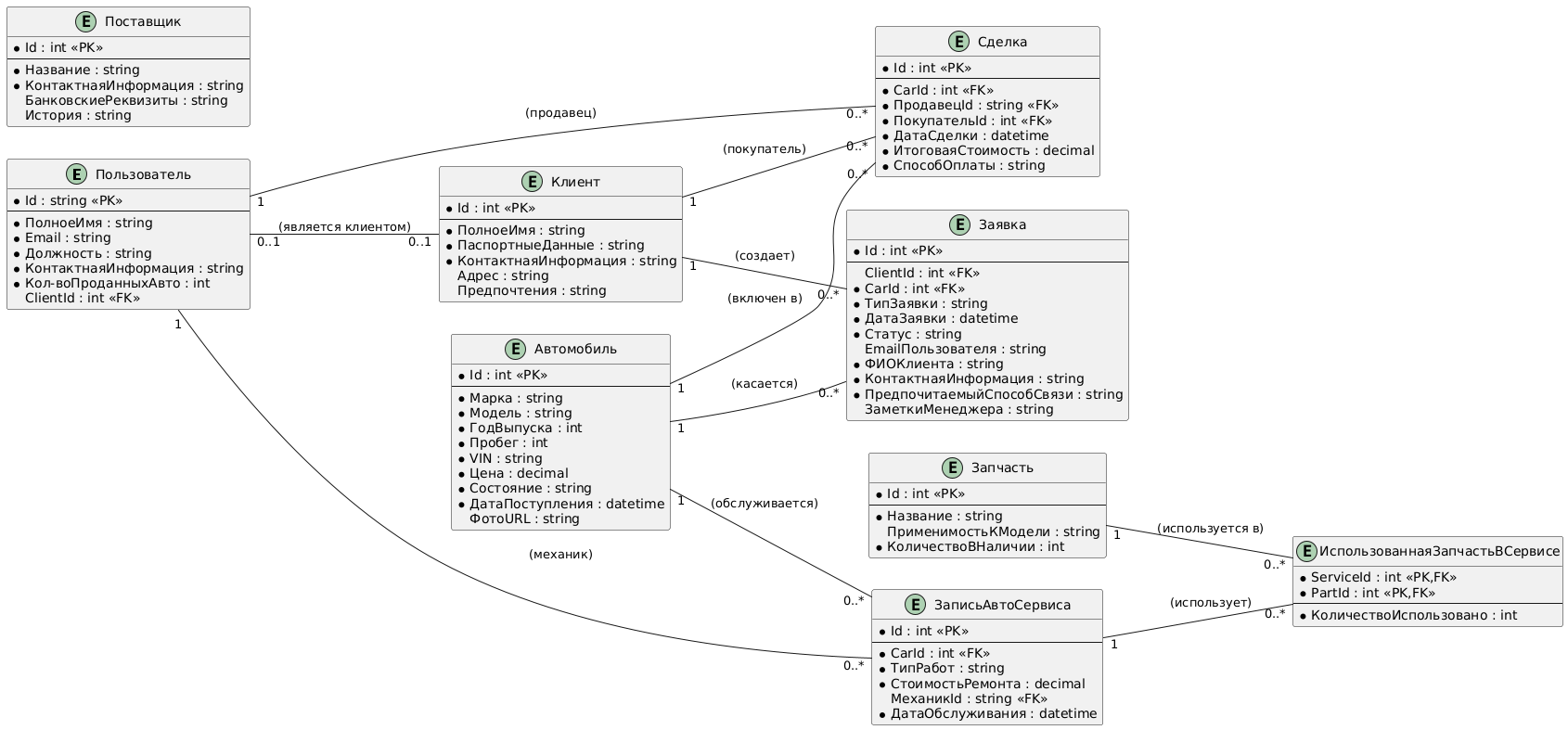
Сущность «ЗаписьАвтоСервиса» (AutoServiceEntry) хранит информацию об обслуживании автомобилей: Id, CarId, Car (навигационное свойство к Автомобилю), WorkType, RepairCost, MechanicId, Mechanic (навигационное свойство к Пользователю), ServiceDate. Одна запись автосервиса может использовать множество запчастей.

Сущность «Заявка» (ApplicationRequest) содержит информацию о запросах клиентов: Id, ClientId, Client (навигационное свойство к Клиенту), CarId, Car (навигационное свойство к Автомобилю), RequestType, RequestDate, Status, UserEmail, ClientFullName, ContactInfo, PreferredContactMethod, ManagerNotes.

Сущность «Сделка» (Deal) хранит информацию об оформленных продажах автомобилей: Id, CarId, Car (навигационное свойство к Автомобилю), SellerId, Seller (навигационное свойство к Пользователю), BuyerId, Buyer (навигационное свойство к Клиенту), DealDate, FinalCost, PaymentMethod.

Связи между сущностями построены таким образом, чтобы обеспечить логическую целостность данных и правильную маршрутизацию между автомобилями, клиентами, пользователями, сделками, заявками, записями автосервиса, запчастями и поставщиками. Один Пользователь (менеджер) может оформлять множество Сделок и быть Механиком во множестве ЗаписейАвтоСервиса. Один Клиент может быть Покупателем во множестве Сделок и создавать множество Заявок. Одна ЗаписьАвтоСервиса может использовать множество Запчастей через ИспользованнаяЗапчастьВСервисе.

Диаграмма «Сущность-связь» представлена на Рисунке 1.



1. – Диаграмма «Сущность-связь»

Приведённые таблицы базы данных были спроектированы на основании ER-диаграммы и отражают физическую реализацию логической структуры, разработанной в рамках предметной области веб-приложения «WestMotors» — системы автоматизации сети магазинов техники. Каждая таблица соответствует отдельной сущности, представленной на диаграмме, а связи между таблицами реализованы с помощью внешних ключей, обеспечивающих целостность и согласованность данных.

Такая структура позволяет эффективно реализовать все заявленные функции: управление пользователями и заказами, обработку корзины, отображение каталога, работу с отчетами, а также хранение характеристик и изображений товаров. Кроме того, база данных обеспечивает надёжную основу для расширения функциональности приложения в будущем.

Для реализации этих функций была разработана следующая структура таблиц:

Таблица «Пользователи» хранит информацию о зарегистрированных пользователях системы.

Таблица 1 – Структура таблицы «AspNetUsers»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| Id | nvarchar | 900 | Уникальный идентификатор пользователя (PK) |
| FullName | nvarchar | 200 | Полное имя пользователя |
| Position | nvarchar | 100 | Должность в компании |
| ContactInfo | nvarchar | 400 | Контактная информация |
| SoldCarsCount | int | 4 | Количество проданных автомобилей (для менеджеров) |
| UserName | nvarchar | 512 | Логин пользователя |
| NormalizedUserName | nvarchar | 512 | Нормализованный логин для поиска |
| Email | nvarchar | 512 | Электронная почта |
| NormalizedEmail | nvarchar | 512 | Нормализованный email для поиска |
| EmailConfirmed | bit | 1 | Флаг подтверждения email |
| PasswordHash | nvarchar | MAX | Хэш пароля |
| SecurityStamp | nvarchar | MAX | Метка безопасности |
| ConcurrencyStamp | nvarchar | MAX | Метка контроля версий |
| PhoneNumber | nvarchar | MAX | Номер телефона |
| PhoneNumberConfirmed | bit | 1 | Флаг подтверждения телефона |
| TwoFactorEnabled | bit | 1 | Включена ли двухфакторная аутентификация |
| LockoutEnd | datetimeoffset | 10 | Дата окончания блокировки |
| LockoutEnabled | bit | 1 | Возможность блокировки аккаунта |
| AccessFailedCount | int | 4 | Количество неудачных попыток входа |

Таблица «Роли» Содержит роли пользователей системы.

Таблица 2 – Структура таблицы «AspNetRoles»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| Id | nvarchar | 900 | Уникальный идентификатор роли (PK) |
| Name | nvarchar | 512 | Название роли |
| NormalizedName | nvarchar | 512 | Нормализованное название для поиска |
| ConcurrencyStamp | nvarchar | MAX | Метка контроля версий |

Таблица «Связи пользователей и ролей» хранит информацию о назначенных пользователям ролях в системе.

Таблица 3 – Структура таблицы «AspNetUserRoles»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| UserId | nvarchar | 900 | Идентификатор пользователя (PK, FK) |
| RoleId | nvarchar | 900 | Идентификатор роли (PK, FK) |

Таблица «Утверждения ролей» содержит дополнительные права и разрешения для ролей.

Таблица 4 – Структура таблицы «AspNetRoleClaims»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| Id | int | 4 | Уникальный идентификатор (PK) |
| RoleId | nvarchar | 900 | Идентификатор роли (FK) |
| ClaimType | nvarchar | MAX | Тип разрешения |
| ClaimValue | nvarchar | MAX | Значение разрешения |

Таблица «Утверждения пользователей» содержит дополнительные права и разрешения для конкретных пользователей.

Таблица 5 – Структура таблицы «AspNetUserClaims»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| Id | int | 4 | Уникальный идентификатор (PK) |
| UserId | nvarchar | 900 | Идентификатор пользователя (FK) |
| ClaimType | nvarchar | MAX | Тип разрешения |
| ClaimValue | nvarchar | MAX | Значение разрешения |

Таблица «Клиенты» содержит информацию о клиентах автосалона.

Таблица 6 – Структура таблицы «Clients»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| Id | int | 4 | Уникальный идентификатор (PK) |
| FullName | nvarchar | 200 | Полное имя клиента |
| PassportData | nvarchar | 400 | Паспортные данные |
| ContactInfo | nvarchar | 400 | Контактная информация |
| Preferences | nvarchar | MAX | Предпочтения клиента |

Таблица «Автомобили» содержит информацию об автомобилях в автосалоне.

Таблица 7 – Структура таблицы «Cars»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| Id | int | 4 | Уникальный идентификатор (PK) |
| Brand | nvarchar | 100 | Марка автомобиля |
| Model | nvarchar | 100 | Модель автомобиля |
| ManufactureYear | int | 4 | Год выпуска |
| Mileage | int | 4 | Пробег (км) |
| VIN | nvarchar | 34 | VIN-номер |
| Price | decimal | 9 | Цена автомобиля |
| Condition | nvarchar | 100 | Состояние (новый, б/у) |
| ArrivalDate | datetime2 | 8 | Дата поступления |
| PhotoUrl | nvarchar | 1000 | Ссылка на фото |

Таблица «Сделки» фиксирует информацию о совершенных сделках купли-продажи.

Таблица 8 – Структура таблицы «Deals»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| Id | int | 4 | Уникальный идентификатор (PK) |
| CarId | int | 4 | Идентификатор автомобиля (FK) |
| SellerId | nvarchar | 900 | Идентификатор продавца (FK) |
| BuyerId | int | 4 | Идентификатор покупателя (FK) |
| DealDate | datetime2 | 8 | Дата совершения сделки |
| FinalCost | decimal | 9 | Финальная стоимость |
| PaymentMethod | nvarchar | 100 | Способ оплаты |

Таблица «Журнал миграций» одержит историю примененных миграций базы данных.

Таблица 9 – Структура таблицы «\_\_EFMigrationsHistory»

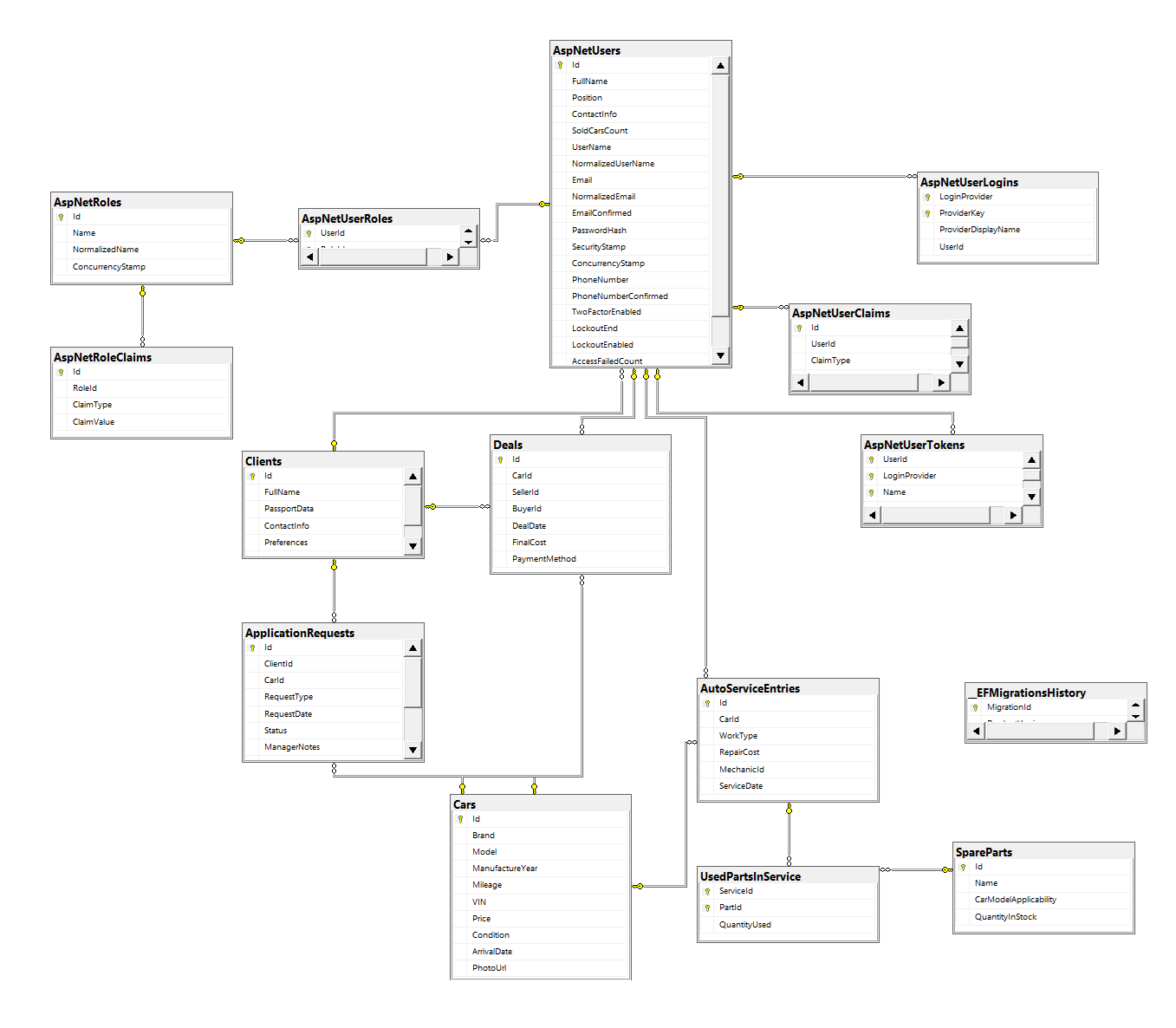
|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| MigrationId | nvarchar | 300 | Уникальный идентификатор миграции (PK) |
| ProductVersion | nvarchar | 64 | Версия Entity Framework, использованная для миграции |

Таблица «Записи автосервиса» содержит информацию о выполненных работах по техническому обслуживанию и ремонту автомобилей.

Таблица 10 – Структура таблицы «AutoServiceEntries»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер | Описание поля |
| Id | int | 4 | Уникальный идентификатор записи (PK) |
| CarId | int | 4 | Идентификатор автомобиля (FK) |
| WorkType | nvarchar | 200 | Тип выполненных работ (ТО, ремонт и т.д.) |
| RepairCost | decimal | 9 | Стоимость выполненных работ |
| MechanicId | nvarchar | 900 | Идентификатор механика (FK) |
| ServiceDate | datetime2 | 8 | Дата выполнения работ |

Диаграмма базы данных представлена на рисунке 2



1. – Диаграмма базы данных

# Реализация веб-приложения

## Описание разделов веб-приложения

Главная страница отображает приветствие и базовую информацию о автосалоне. Основные элементы: баннер с изображением салона, кнопка перехода в каталог, кнопки авторизации для новых пользователей.

Каталог автомобилей реализован в виде карточек с фильтрацией. Фильтры позволяют сортировать по марке, году выпуска, пробегу и цене. Каждая карточка содержит фото автомобиля, основные характеристики и цену. Администраторы могут добавлять, редактировать и удалять автомобили через соответствующие кнопки.

Раздел сделок представляет таблицу с полной информацией о заключенных договорах. Отображаются данные покупателя, сведения об автомобиле (марка, модель, VIN), дата сделки и окончательная стоимость. Доступны функции создания новых сделок, их редактирования и удаления.

В разделе заявок отображается таблица с текущими обращениями клиентов. Для каждой заявки показаны контакты клиента, интересующий автомобиль, тип запроса, дата обращения и текущий статус. Все пользователи могут просматривать детали заявок, администраторы - дополнительно удалять их.

## Разработка административной части приложения

Административная часть системы включает несколько функциональных модулей для управления автосалоном. Модуль авторизации обеспечивает доступ только для сотрудников с разграничением прав по ролям: администратор, менеджер, консультант.

Модуль управления автомобилями позволяет добавлять новые автомобили в каталог, редактировать существующие записи (марка, модель, год выпуска, пробег, цена, состояние), загружать фотографии, управлять статусами (в наличии/продано/на обслуживании). Для каждого автомобиля ведется полная техническая карточка.

Модуль обработки заявок предоставляет инструменты для работы с клиентскими обращениями. Менеджеры могут просматривать новые заявки, назначать ответственных, добавлять комментарии. Реализована система уведомлений о новых заявках.

Модуль управления сделками фиксирует все заключенные договоры купли-продажи. Вносится информация о покупателе, автомобиле, условиях сделки (цена, дата, способ оплаты), ответственных сотрудниках. Формируются электронные копии документов.

Логическая и Физическая структуры административной части представлены в приложении Б рисуноки Б.1 и Б.2.

## Разработка клиентской части приложения

Клиентская часть приложения WestMotors реализована как веб-интерфейс для пользователей. Основные функции включают авторизацию через логин и пароль, регистрацию новых клиентов и восстановление доступа. Пользователи могут просматривать каталог автомобилей с фильтрацией по марке, году выпуска, цене и пробегу. Реализована система сортировки по различным параметрам. Для каждого автомобиля доступна детальная карточка с фотографиями и характеристиками.

Функционал оформления заявок позволяет создавать запросы на конкретные автомобили, записываться на тест-драйв.

Технически клиентская часть построена на HTML, CSS и JavaScript с использованием С# ASP.NET Core MVC. Интерфейс адаптирован для мобильных устройств и компьютеров. Взаимодействие с сервером осуществляется через REST API. Система сохраняет историю поисковых запросов для удобства пользователей. Все формы имеют валидацию вводимых данных.

Логическая и Физическая структуры клиентской части представлены на приложение Б на рисунке Б.3 и Б.4.

## Описание используемых функций и процедур

Для реализации графического интерфейса веб-приложения «West Motors» использовались технологии C#, ASP.NET Core MVC, HTML, CSS, JS и SQL Server. Такой стек обеспечивает современный внешний вид, кроссбраузерную совместимость и надёжное хранение данных. Архитектура построена по принципу клиент–серверного взаимодействия с использованием REST API, что позволяет эффективно масштабировать проект и разделять логику между фронтендом и бэкендом.

Функция в программировании — это отдельный блок кода, выполняющий конкретную задачу и вызываемый из разных частей программы. Использование функций помогает структурировать код, делает его более понятным, удобным для повторного использования и сопровождения. Функции могут принимать входные параметры и возвращать результат своей работы.

Кроссплатформенность и адаптивность обеспечиваются за счёт использования React: все компоненты интерфейса автоматически адаптируются под размер экрана устройства. Для стилизации применяются кастомные CSS-решения и адаптивная вёрстка, что делает сайт удобным для использования как на ПК, так и на мобильных устройствах.

Локальная база данных реализована с использованием SQL Server и ORM-библиотеки Entity Framework Core, что обеспечивает надёжную и эффективную работу с таблицами пользователей, товаров, заказов, корзины, изображений, категорий и отзывов.

Таким образом, веб-приложение «West Motors» представляет собой полноценное веб-приложение, предлагающее пользователю удобную навигацию, авторизацию, возможность оформления заявок и удобого управления приложением со стороны администратора.

Код функции регистрации пользователя представлен ниже:

public async Task<IActionResult> OnPostAsync(string? returnUrl = null)

{

returnUrl ??= Url.Content("~/");

ExternalLogins = (await \_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

if (ModelState.IsValid)

{

var user = CreateUser();

await \_userStore.SetUserNameAsync(user, Input.Email, CancellationToken.None);

await \_emailStore.SetEmailAsync(user, Input.Email, CancellationToken.None);

var client = await \_dbContext.Clients

.FirstOrDefaultAsync(c => c.FullName == Input.ClientFullName &&

c.PassportData == Input.ClientPassportData);

if (client == null)

{

client = new Client

{

FullName = Input.ClientFullName,

PassportData = Input.ClientPassportData,

ContactInfo = Input.ClientContactInfo,

Preferences = Input.ClientPreferences

};

\_dbContext.Clients.Add(client);

await \_dbContext.SaveChangesAsync();

}

user.ClientId = client.Id;

var result = await \_userManager.CreateAsync(user, Input.Password);

if (result.Succeeded)

{

\_logger.LogInformation("Пользователь создал новую учетную запись с паролем.");

if (!await \_roleManager.RoleExistsAsync("Клиент"))

{

await \_roleManager.CreateAsync(new IdentityRole("Клиент"));

}

if (!await \_roleManager.RoleExistsAsync("Администратор"))

{

await \_roleManager.CreateAsync(new IdentityRole("Администратор"));

}

if (!await \_roleManager.RoleExistsAsync("Менеджер"))

{

await \_roleManager.CreateAsync(new IdentityRole("Менеджер"));

}

await \_userManager.AddToRoleAsync(user, "Клиент");

var userId = await \_userManager.GetUserIdAsync(user);

var code = await \_userManager.GenerateEmailConfirmationTokenAsync(user);

code = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));

var callbackUrl = Url.Page(

"/Account/ConfirmEmail",

pageHandler: null,

values: new { area = "Identity", userId = userId, code = code, returnUrl = returnUrl },

protocol: Request.Scheme);

await \_emailSender.SendEmailAsync(Input.Email, "Подтвердите свой email",

$"Пожалуйста, подтвердите свою учетную запись, <a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>нажав здесь</a>.");

if (\_userManager.Options.SignIn.RequireConfirmedAccount)

{

return RedirectToPage("RegisterConfirmation", new { email = Input.Email, returnUrl = returnUrl });

}

else

{

await \_signInManager.SignInAsync(user, isPersistent: false);

return LocalRedirect(returnUrl);

}

}

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

return Page();

}

Код функции авторизации пользователя:

public async Task<IActionResult> OnPostAsync(string returnUrl = null)

{

returnUrl ??= Url.Content("~/");

ExternalLogins = (await \_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

if (ModelState.IsValid)

{

var result = await \_signInManager.PasswordSignInAsync(Input.Email, Input.Password, Input.RememberMe, lockoutOnFailure: false);

if (result.Succeeded)

{

\_logger.LogInformation("User logged in.");

return LocalRedirect(returnUrl);

}

if (result.RequiresTwoFactor)

{

return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl, RememberMe = Input.RememberMe });

}

if (result.IsLockedOut)

{

\_logger.LogWarning("User account locked out.");

return RedirectToPage("./Lockout");

}

else

{

ModelState.AddModelError(string.Empty, "Invalid login attempt.");

return Page();

}

}

return Page();

}

Код функции оставления заявки:

[Authorize(Roles = "Клиент")]

public async Task<IActionResult> CreateForCar(int? carId)

{

if (carId == null)

{

TempData["ErrorMessage"] = "Автомобиль не выбран для заявки.";

return RedirectToAction("Index", "Cars");

}

var car = await \_context.Cars.FindAsync(carId);

if (car == null)

{

TempData["ErrorMessage"] = "Выбранный автомобиль не найден в базе данных.";

return NotFound();

}

var currentUser = await \_userManager.GetUserAsync(User);

if (currentUser == null)

{

\_logger.LogError("Ошибка: \_userManager.GetUserAsync(User) вернул null для залогиненного пользователя {UserName}", User.Identity?.Name);

TempData["ErrorMessage"] = "Произошла ошибка при определении данных пользователя. Пожалуйста, попробуйте войти снова.";

return RedirectToPage("/Account/Login", new { area = "Identity" });

}

currentUser = await \_context.Users

.Include(u => u.Client)

.FirstOrDefaultAsync(u => u.Id == currentUser.Id);

var applicationRequest = new ApplicationRequest

{

CarId = car.Id,

ClientId = currentUser?.ClientId,

RequestDate = DateTime.UtcNow,

Status = "Новая",

UserEmail = currentUser?.Email,

ClientFullName = currentUser?.Client?.FullName ?? currentUser?.FullName ?? currentUser?.Email ?? "Неизвестно"

};

await PopulateViewBagsForCreateForCar(applicationRequest, currentUser);

return View(applicationRequest);

}

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Клиент")]

public async Task<IActionResult> CreateForCar([Bind("CarId,RequestType,ClientFullName,ContactInfo,PreferredContactMethod")] ApplicationRequest applicationRequest)

{

var currentUser = await \_userManager.GetUserAsync(User);

if (currentUser == null)

{

ModelState.AddModelError(string.Empty, "Ошибка: Пользователь не найден. Пожалуйста, попробуйте войти снова.");

await PopulateViewBagsForCreateForCar(applicationRequest, null);

return View(applicationRequest);

}

currentUser = await \_context.Users

.Include(u => u.Client)

.FirstOrDefaultAsync(u => u.Id == currentUser.Id);

applicationRequest.ClientId = currentUser?.ClientId;

applicationRequest.UserEmail = currentUser?.Email;

applicationRequest.RequestDate = DateTime.UtcNow;

applicationRequest.Status = "Новая";

if (applicationRequest.PreferredContactMethod == "Другой")

{

if (string.IsNullOrWhiteSpace(applicationRequest.ContactInfo))

{

ModelState.AddModelError("ContactInfo", "Пожалуйста, введите контактные данные, если выбран 'Другой' способ связи.");

}

}

else

{

if (applicationRequest.ContactInfo != applicationRequest.PreferredContactMethod)

{

if (string.IsNullOrWhiteSpace(applicationRequest.ContactInfo))

{

applicationRequest.ContactInfo = applicationRequest.PreferredContactMethod;

}

else

{

applicationRequest.ContactInfo = applicationRequest.PreferredContactMethod;

}

}

}

if (string.IsNullOrWhiteSpace(applicationRequest.ClientFullName))

{

ModelState.AddModelError("ClientFullName", "ФИО клиента обязательно.");

}

ModelState.Remove(nameof(applicationRequest.Car));

if (ModelState.IsValid)

{

try

{

\_context.Add(applicationRequest);

await \_context.SaveChangesAsync();

TempData["SuccessMessage"] = "Ваша заявка успешно отправлена!";

return RedirectToAction("MyApplications");

}

catch (DbUpdateException dbEx)

{

\_logger.LogError(dbEx, "Ошибка базы данных при сохранении заявки: {Message}", dbEx.Message);

var innerException = dbEx.InnerException;

while (innerException != null)

{

\_logger.LogError("Inner Exception: {Message}", innerException.Message);

innerException = innerException.InnerException;

}

ModelState.AddModelError("", "Произошла ошибка при сохранении заявки в базу данных. Пожалуйста, проверьте вводимые данные или обратитесь в поддержку.");

if (applicationRequest.CarId > 0 && !await \_context.Cars.AnyAsync(c => c.Id == applicationRequest.CarId))

{

ModelState.AddModelError("CarId", "Выбранный автомобиль не существует.");

}

}

catch (Exception ex)

{

\_logger.LogError(ex, "Неизвестная ошибка при сохранении заявки: {Message}", ex.Message);

ModelState.AddModelError("", "Произошла непредвиденная ошибка при отправке заявки. Пожалуйста, попробуйте еще раз.");

}

}

await PopulateViewBagsForCreateForCar(applicationRequest, currentUser);

return View(applicationRequest);

}

private async Task PopulateViewBagsForCreateForCar(ApplicationRequest model, ApplicationUser? currentUser)

{

if (model.CarId > 0)

{

ViewBag.Car = await \_context.Cars.FindAsync(model.CarId);

}

else

{

ViewBag.Car = null;

}

List<SelectListItem> contactOptions = new List<SelectListItem>();

if (currentUser != null && !string.IsNullOrEmpty(currentUser.Email))

{

contactOptions.Add(new SelectListItem { Value = currentUser.Email, Text = $"Email: {currentUser.Email}" });

}

if (currentUser?.Client != null && !string.IsNullOrEmpty(currentUser.Client.ContactInfo) && currentUser.Client.ContactInfo != currentUser?.Email)

{

contactOptions.Add(new SelectListItem { Value = currentUser.Client.ContactInfo, Text = $"Телефон/Другой: {currentUser.Client.ContactInfo}" });

}

contactOptions.Add(new SelectListItem { Value = "Другой", Text = "Другой (ввести вручную)" });

ViewBag.ContactOptions = contactOptions;

if (string.IsNullOrEmpty(model.PreferredContactMethod) && contactOptions.Any())

{

model.PreferredContactMethod = contactOptions.First().Value;

if (model.PreferredContactMethod != "Другой")

{

model.ContactInfo = model.PreferredContactMethod;

}

}

else if (model.PreferredContactMethod != "Другой" && string.IsNullOrEmpty(model.ContactInfo))

{

model.ContactInfo = model.PreferredContactMethod;

}

}

## Функциональное тестирование

Функциональное тестирование представляет собой процесс проверки программного обеспечения на соответствие заявленным требованиям, при котором оценивается корректность выполнения системой всех предусмотренных функций. Основная задача данного вида тестирования — удостовериться, что программный продукт надёжно выполняет заданные действия как при корректных, так и при ошибочных вводах.

Проверка будет осуществляться на основе заранее подготовленных тест-кейсов, каждый из которых описывает определённый сценарий использования, включает входные данные, ожидаемый результат и фактический результат выполнения. Тест-кейсы охватывают основные модули приложения: регистрацию, вход в систему, просмотр каталога рецептов, добавление рецептов в избранное, автоматический перерасчёт ингредиентов, формирование списка покупок и добавление пользовательских рецептов. Отдельное внимание будет уделено обработке некорректных данных, таких как пустые поля, неправильный формат электронной почты, неверные учетные данные и т. д.

Результаты функционального тестирования будут документированы и дополнены скриншотами пользовательского интерфейса, подтверждающими корректность работы функций или наличие выявленных ошибок.

Таблица 11 – Тест - кейсы

|  |  |  |  |
| --- | --- | --- | --- |
| Модуль / Функция | Шаги выполнения | Ожидаемый результат | Фактический результат |
| 1 | 2 | 3 | 4 |
| Регистрация | 1. Открыть сайт 2. Нажать "Регистрация" 3. Заполнить обязательные поля (email, пароль) 4. Подтвердить регистрацию | Создание нового аккаунта, переход на главную страницу | Аккаунт создан, данные сохранены. Результат представлен на приложении Д на рисунке Д.1 |
| Авторизация | 1. Ввести email и пароль 2. Нажать "Войти" | Успешный вход в систему | Вход в систему совершен. Результат представлен на приложении Д на рисунке Д.2 |
| Оставление заявки | 1. Выбрать услугу/товар 2. Нажать "Оставить заявку" 3. Заполнить контактные данные 4. Подтвердить отправку | Заявка отправлена менеджеру | Заявка появилась в админской панели.  Результат представлен на приложении Д на рисунке Д.3 |

# Применение

## Назначение веб-приложения

Данное веб-приложение разработано для автоматизации бизнес-процессов автосалона «West Motors». Система предоставляет комплексные решения для управления автомобильным ассортиментом, обработки заявок клиентов, организации тест-драйвов и ведения клиентской базы. Приложение оптимизирует работу персонала и повышает качество обслуживания покупателей.

Для развертывания системы необходимо выполнить установку согласно руководству разработчика. После размещения на сервере требуется настроить рабочее окружение: установить необходимые зависимости, подключить базу данных SQL Server и выполнить первоначальную миграцию данных. По завершении установки приложение готово к запуску с использованием стандартных команд для верификации его работоспособности.

Веб-приложение направлено на цифровизацию продаж автомобилей и сопутствующих услуг в автосалоне «West Motors». Клиенты получают возможность просматривать каталог доступных автомобилей, бронировать тест-драйвы, оформлять онлайн-заявки на покупку и кредитование. Персонал салона имеет доступ к специализированной панели для управления автомобилями, обработки заявок, ведения клиентов и формирования отчетности.

Ограничения области применения включают:

* оформление заявок доступны только авторизованным пользователям;
* административные функции управления автомобилями и заявками доступны исключительно сотрудникам с соответствующими правами доступа;
* система ориентирована на автомобильный бизнес и поддерживает специфичные для отрасли процессы.

## Программно-аппаратное обеспечение сервера и клиента

Разрабатываемое веб-приложение для автосалона «West Motors» функционирует на сервере под управлением операционной системы Windows 11.

Минимальные системные требования для сервера:

* процессор с четырьмя и более физическими ядрами;
* оперативная память объёмом не менее 4 ГБ;
* свободное место на жёстком диске от 4 ГБ.

Для запуска и корректной работы программного обеспечения необходимо установить следующие компоненты:

* веб-сервер Apache HTTP Server — сервер с открытым исходным кодом, обрабатывающий HTTP-запросы от клиентов и предоставляющий доступ к статическому и динамическому контенту (HTML, CSS, JavaScript). Используется для хостинга клиентской части и API;
* система управления базами данных Microsoft SQL Server — реляционная СУБД, применяемая для хранения информации о товарах, пользователях, заказах, отзывах и других элементах бизнес-логики маркетплейса;
* платформа разработки .NET SDK (ASP.NET Core) — набор средств разработки для реализации серверной логики приложения на языке C# с архитектурой MVC. Поддерживает построение REST API и безопасную работу с базой данных;

Клиентская часть веб-приложения создаётся с использованием следующих инструментов:

* ведактор кода Visual Studio — основной инструмент для разработки клиентской части, поддерживающий С#, HTML, CSS и JavaScript а также интеграцию с системами контроля версий;
* веб-браузер (Google Chrome, Firefox, Microsoft Edge и др.) — необходим для запуска клиентской части, предоставляющей доступ к веб-интерфейсу приложения.

Развёртывание и запуск приложения:

Развёртывание базы данных осуществляется с помощью SQL Server Management Studio (SSMS), где создаётся база данных WestMotorsAppDb со всеми необходимыми таблицами.  
Серверная и клиентская части запускаются через Visual Studio.

После запуска веб-приложение становится доступным локально: API функционирует на http://localhost:5085, клиентская часть — на http://localhost:3000. Клиент взаимодействует с сервером через HTTP-запросы, обрабатываемые ASP.NET Core и SQL Server.

# Заключение

В рамках выполнения проекта на тему «Разработка веб-приложения для автосалона West Motors» было создано полнофункциональное решение, предназначенное для автоматизации продаж автомобилей и сопутствующих услуг.

Для достижения целей проектирования были выполнены следующие задачи:

* разработка комплексного веб-решения для удобного взаимодействия клиентов с автомобильным каталогом и эффективной работы персонала салона;
* проектирование пользовательских сценариев и ролей (покупатель, администратор);
* создание интерфейсов с адаптивной вёрсткой для поддержки мобильных и десктопных устройств;
* реализация серверной части с использованием ASP.NET Core и архитектуры REST API;
* подключение базы данных SQL Server и построение схемы данных с использованием Entity Framework Core;
* описание структуры приложения через диаграммы: прецедентов, классов и развёртывания;
* разработка административной панели для управления товарами, заказами и пользователями;
* проведение тестирования основных функций: регистрация, поиск, оформление заказа, управление каталогом.

Приложение обладает удобным, современным и адаптивным интерфейсом. Пользователь может быстро зарегистрироваться, подобрать нужный товар с помощью фильтрации, оставить заявку на тест-драйв или покупку, выбрав подходящий способ связи и оплаты. Благодаря использованию компонентов интерфейс динамично реагирует на действия пользователя и сохраняет целостность на всех типах устройств.

Ключевыми преимуществами проекта являются модульная архитектура, масштабируемость, безопасность и простота обслуживания. В ходе разработки были закреплены практические навыки по построению клиент-серверных приложений, проектированию базы данных, реализации бизнес-логики и созданию адаптивного фронтенда.

Приложение WestMotors готово к использованию в реальной среде и может применяться в качестве платформы для электронной коммерции в сфере продажи техники и аксессуаров.

# Список информационных источников

1. Багласова, Т.Г. Методические указания по выполнению курсового проекта для учащихся по специальности 2-40 01 01 «Программное обеспечение технологий» / Т.Г. Багласова. – Минск : КБП, 2017. – 30 c.
2. Багласова, Т.Г. Методические указания по оформлению курсовых и дипломных проектов / Т.Г. Багласова, К.О. Якимович. – Минск : КБП, 2013. – 29 c.
3. Бондарь, А.Г. Microsoft SQL Server 2012 / А.Г. Бондарь. – СПб. : БХВ-Петербург, 2013. – 608 с.
4. Тепляков, С. Паттерны проектирования на платформе .NET / С.Тепляков. – СПб. : Питер, 2015. – 320 с
5. Руководство по программированию на C# [Электронный ресурс]. – Microsoft, 2020. – Режим доступа : http://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/. – Дата доступа : 25.04.2020.
6. HTML и CSS. Полный справочник [Электронный ресурс] / J. Duckett. — Режим доступа: <https://developer.mozilla.org> — Дата доступа: 13.06.2025.
7. xUnit.net: документация [Электронный ресурс]. – Режим доступа: https://xunit.net/. – Дата доступа: 13.05.2025.
8. Visual Studio: официальная документация [Электронный ресурс]. – Microsoft, 2024. – Режим доступа: https://learn.microsoft.com/ru-ru/visualstudio/?view=vs-2022. – Дата доступа: 13.05.2025.
9. ASP.NET Core: документация [Электронный ресурс]. – Microsoft, 2024. – Режим доступа: https://learn.microsoft.com/aspnet/core. – Дата доступа: 13.05.2025.
10. Entity Framework Core: официальная документация [Электронный ресурс]. – Microsoft, 2024. – Режим доступа: <https://learn.microsoft.com/ef/core> – Дата доступа: 13.05.2025.

Приложения А  
Ментальная карта

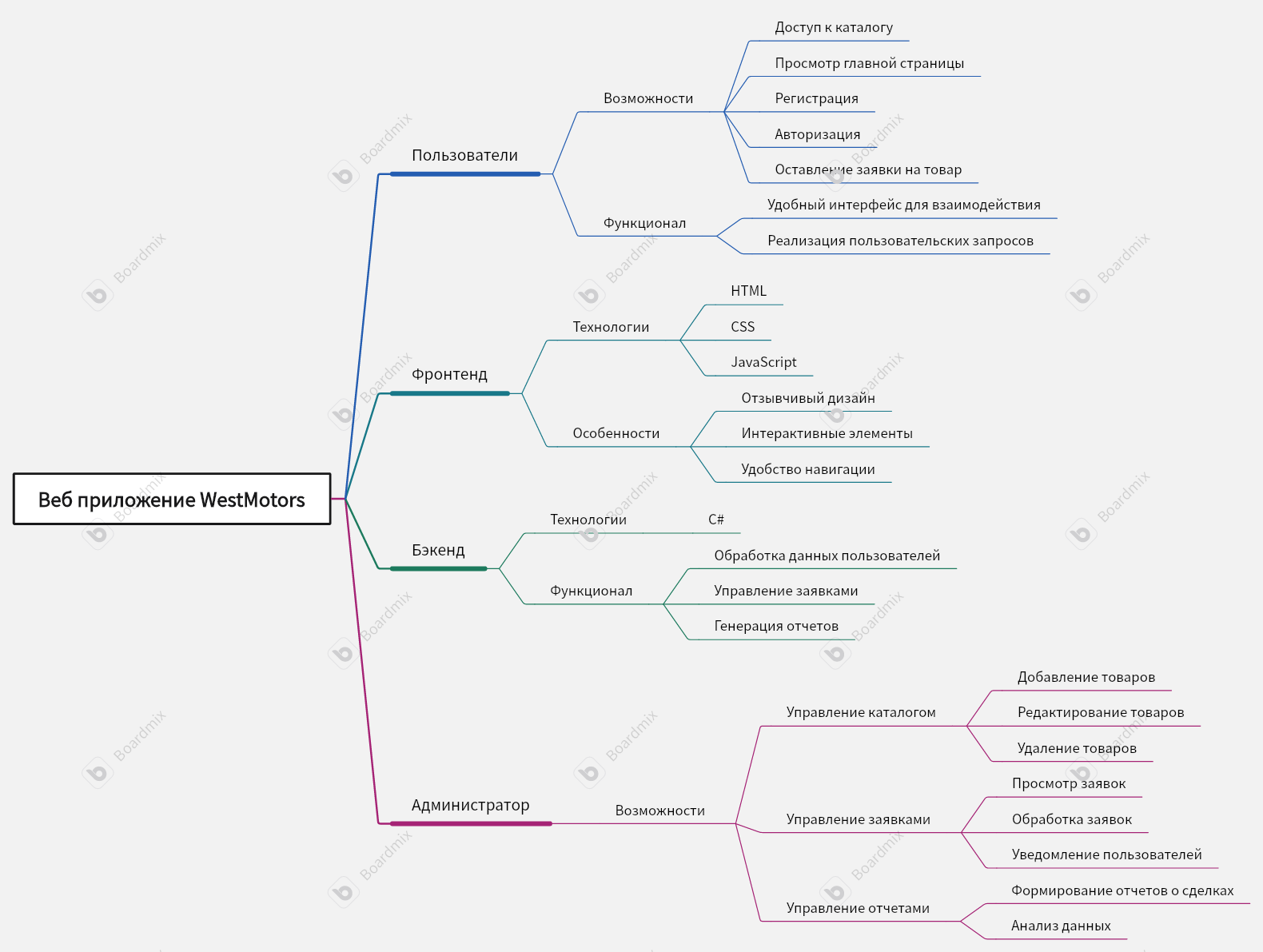
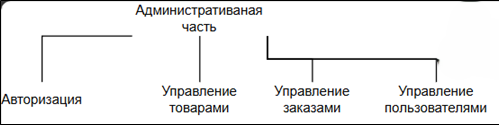
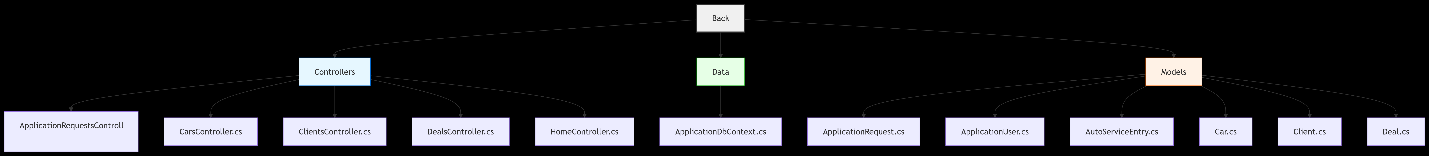


Рисунок А.1 – Ментальная карта

Приложение Б  
Логическая и физическая структура



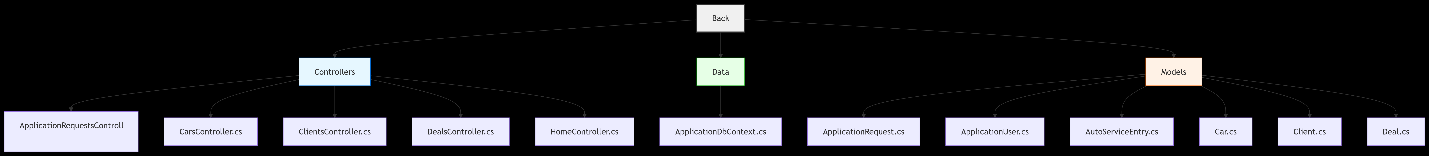
1. Логическая структура административной части веб -приложения



1. Физическая структура административной части веб – приложения

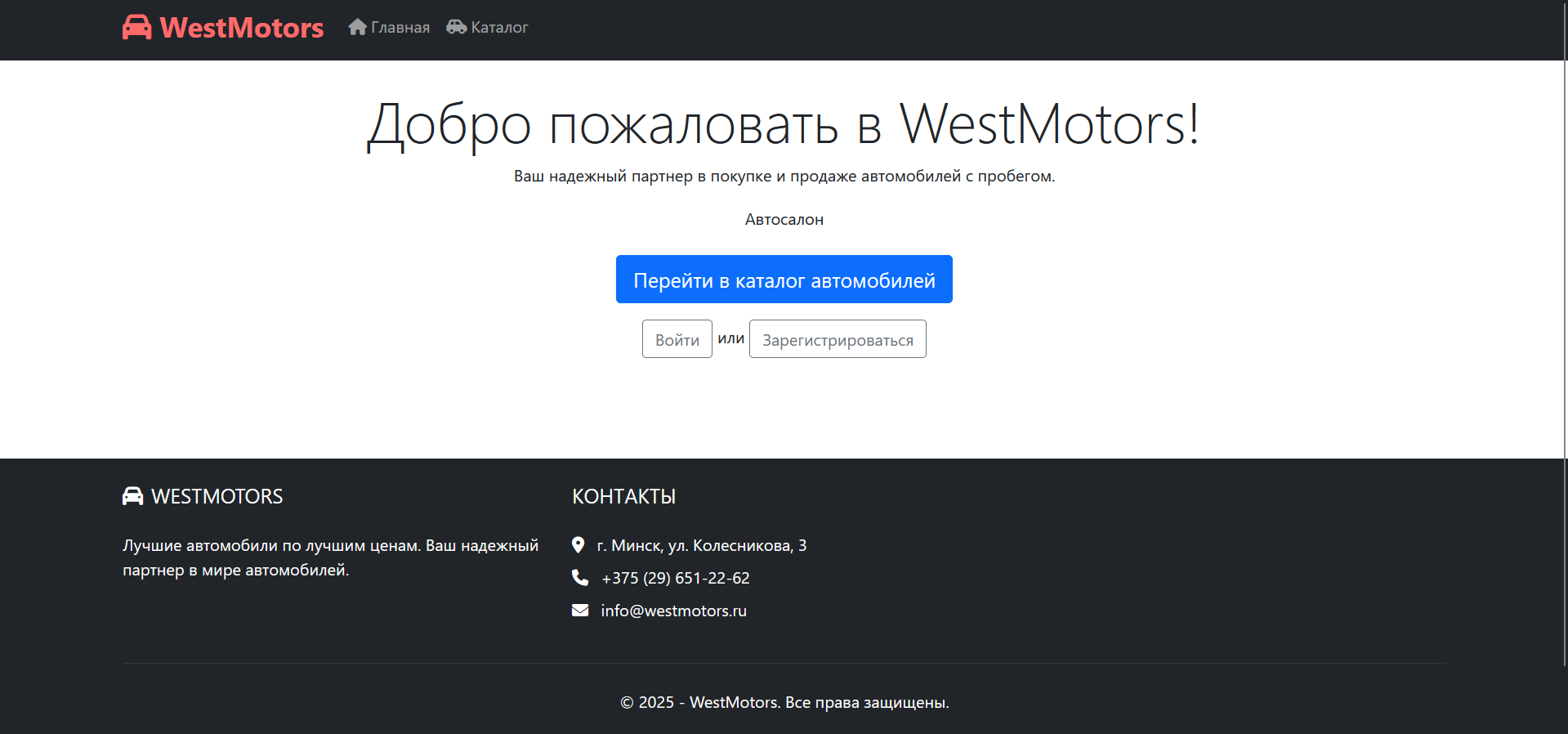


1. Логическая структура клиентской части веб – приложения

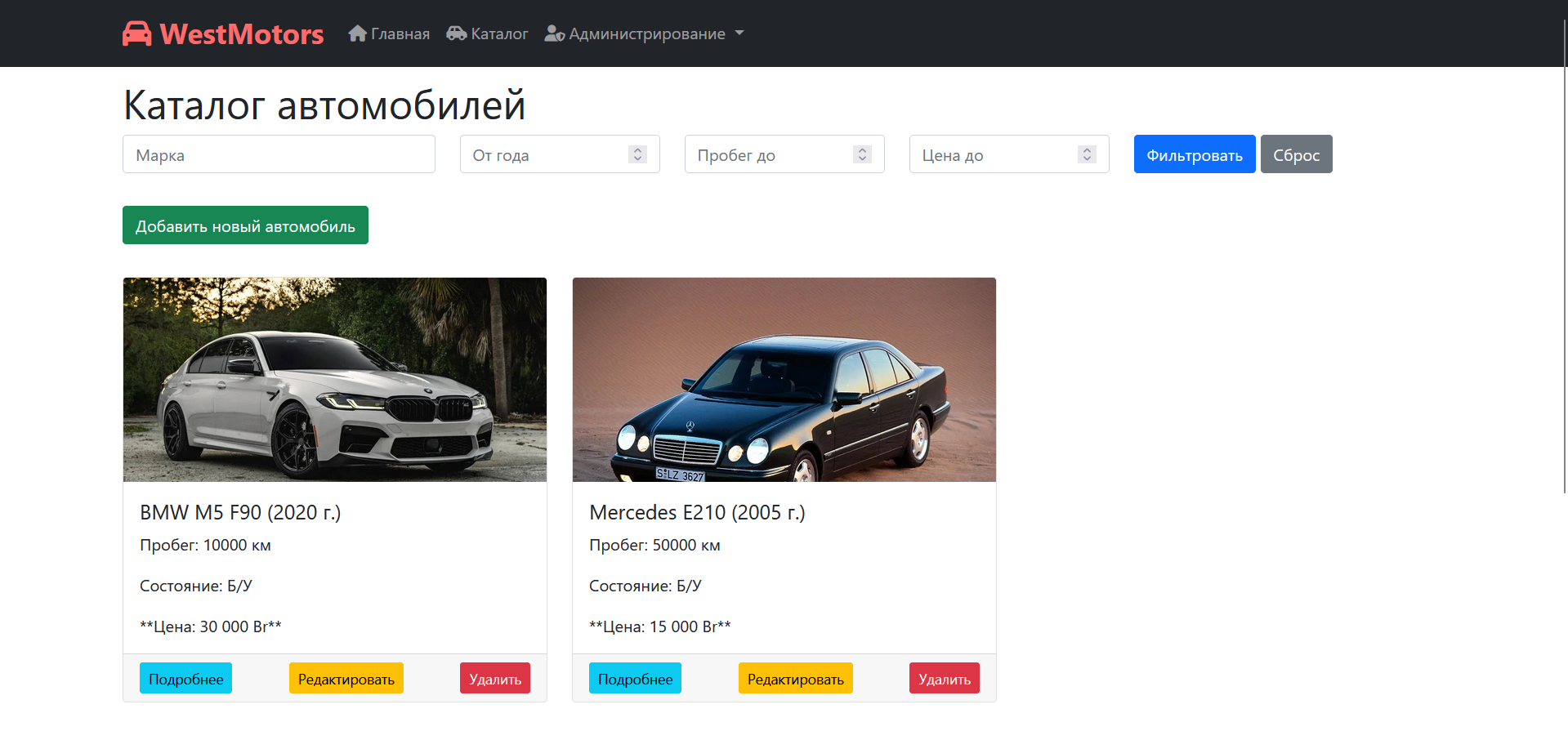


1. Физическая структура клиентской части веб – приложения

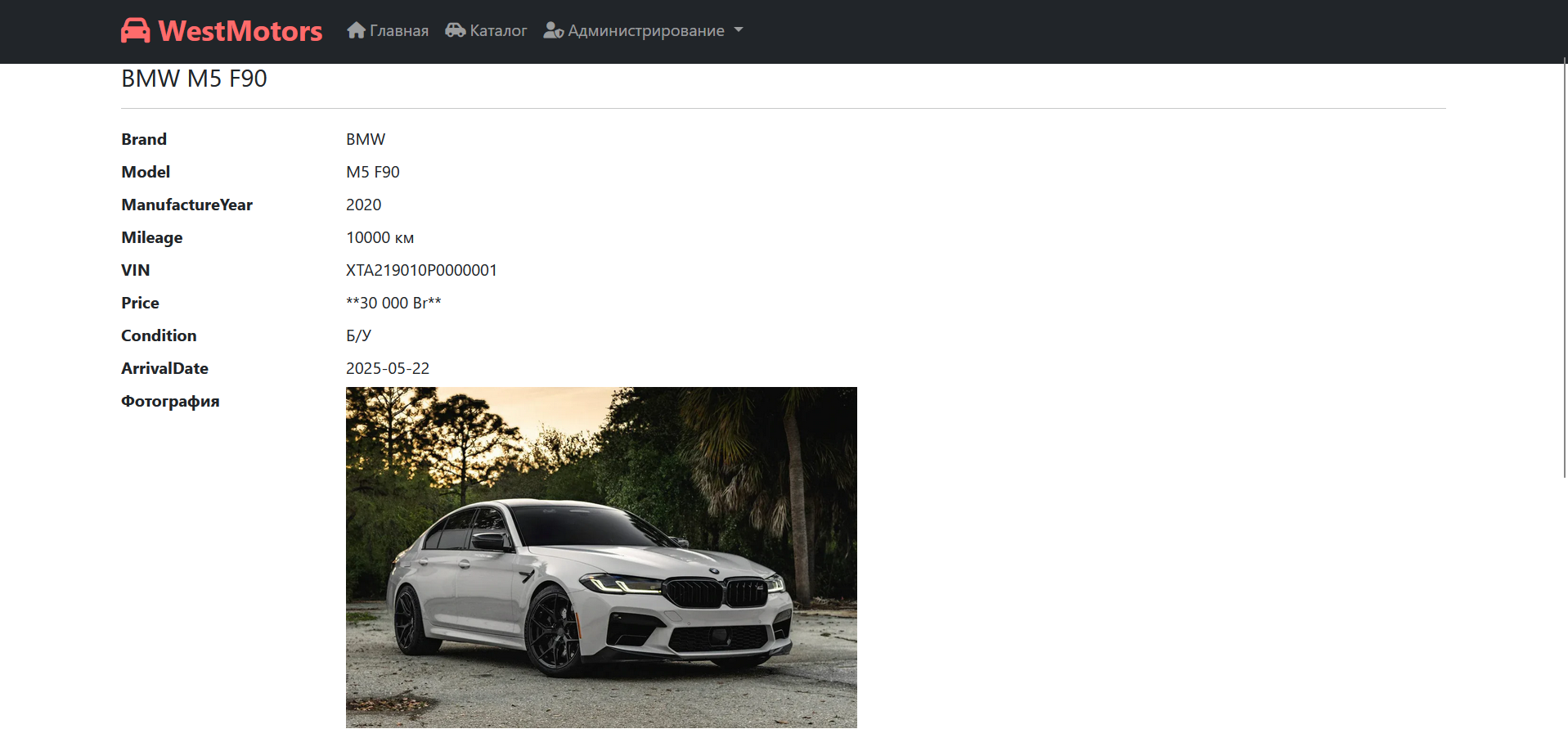
Приложение В  
Макеты страниц веб – приложения



1. Макет страницы «Главная



1. Макет страницы «Каталог товаров»



1. Макет страницы «Карточка автомобиля»

Приложение Г  
Код модулей веб – приложения

ApplicationRequest.cs

using System;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace WestMotorsApp.Models

{

public class ApplicationRequest

{

public int Id { get; set; }

// Изменено: ClientId сделан nullable (int?), т.к. не каждый пользователь обязательно является клиентом

[Display(Name = "Клиент")]

public int? ClientId { get; set; }

public Client? Client { get; set; } // Также nullable навигационное свойство

[Required(ErrorMessage = "Автомобиль обязателен")]

[Display(Name = "Автомобиль")]

public int CarId { get; set; }

public Car Car { get; set; } = default!; // Инициализация, чтобы избежать NullReferenceException

[Required(ErrorMessage = "Тип заявки обязателен")]

[StringLength(100)] // Увеличил длину

[Display(Name = "Тип заявки")]

public string RequestType { get; set; } = string.Empty; // Инициализация

[Required(ErrorMessage = "Дата заявки обязательна")]

[Display(Name = "Дата заявки")]

public DateTime RequestDate { get; set; } = DateTime.UtcNow; // Использовать UtcNow по умолчанию

[Required(ErrorMessage = "Статус заявки обязателен")]

[StringLength(50)]

[Display(Name = "Статус")]

public string Status { get; set; } = "Новая"; // Статус по умолчанию

// Дополнительные поля для связи с пользователем и его данными

[StringLength(256)]

[Display(Name = "Email пользователя")]

public string? UserEmail { get; set; } // Может быть null, если заявка от анонима или менеджера

[Required(ErrorMessage = "ФИО клиента обязательно")]

[StringLength(100)]

[Display(Name = "ФИО клиента")]

public string ClientFullName { get; set; } = string.Empty;

[Required(ErrorMessage = "Контактные данные обязательны")]

[StringLength(200)]

[Display(Name = "Контактная информация")]

public string ContactInfo { get; set; } = string.Empty;

[Required(ErrorMessage = "Способ связи обязателен")]

[StringLength(50)]

[Display(Name = "Предпочитаемый способ связи")]

public string PreferredContactMethod { get; set; } = string.Empty;

[StringLength(500)] // Добавляем ограничение длины

public string? ManagerNotes { get; set; } // Заметки менеджера (могут быть null)

}

}

ApplicationDbContext.cs

using System;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace WestMotorsApp.Models

{

public class ApplicationRequest

{

public int Id { get; set; }

// Изменено: ClientId сделан nullable (int?), т.к. не каждый пользователь обязательно является клиентом

[Display(Name = "Клиент")]

public int? ClientId { get; set; }

public Client? Client { get; set; } // Также nullable навигационное свойство

[Required(ErrorMessage = "Автомобиль обязателен")]

[Display(Name = "Автомобиль")]

public int CarId { get; set; }

public Car Car { get; set; } = default!; // Инициализация, чтобы избежать NullReferenceException

[Required(ErrorMessage = "Тип заявки обязателен")]

[StringLength(100)] // Увеличил длину

[Display(Name = "Тип заявки")]

public string RequestType { get; set; } = string.Empty; // Инициализация

[Required(ErrorMessage = "Дата заявки обязательна")]

[Display(Name = "Дата заявки")]

public DateTime RequestDate { get; set; } = DateTime.UtcNow; // Использовать UtcNow по умолчанию

[Required(ErrorMessage = "Статус заявки обязателен")]

[StringLength(50)]

[Display(Name = "Статус")]

public string Status { get; set; } = "Новая"; // Статус по умолчанию

// Дополнительные поля для связи с пользователем и его данными

[StringLength(256)]

[Display(Name = "Email пользователя")]

public string? UserEmail { get; set; } // Может быть null, если заявка от анонима или менеджера

[Required(ErrorMessage = "ФИО клиента обязательно")]

[StringLength(100)]

[Display(Name = "ФИО клиента")]

public string ClientFullName { get; set; } = string.Empty;

[Required(ErrorMessage = "Контактные данные обязательны")]

[StringLength(200)]

[Display(Name = "Контактная информация")]

public string ContactInfo { get; set; } = string.Empty;

[Required(ErrorMessage = "Способ связи обязателен")]

[StringLength(50)]

[Display(Name = "Предпочитаемый способ связи")]

public string PreferredContactMethod { get; set; } = string.Empty;

[StringLength(500)] // Добавляем ограничение длины

public string? ManagerNotes { get; set; } // Заметки менеджера (могут быть null)

}

}

Car.cs

using System;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Collections.Generic;

using Microsoft.AspNetCore.Http; // Для IFormFile

namespace WestMotorsApp.Models

{

public class Car

{

public int Id { get; set; }

[Required(ErrorMessage = "Марка обязательна")]

[StringLength(50)]

public string Brand { get; set; }

[Required(ErrorMessage = "Модель обязательна")]

[StringLength(50)]

public string Model { get; set; }

[Required(ErrorMessage = "Год выпуска обязателен")]

[Range(1900, 2100, ErrorMessage = "Некорректный год выпуска")]

public int ManufactureYear { get; set; }

[Required(ErrorMessage = "Пробег обязателен")]

[Range(0, 1000000, ErrorMessage = "Некорректный пробег")]

public int Mileage { get; set; }

[Required(ErrorMessage = "VIN-номер обязателен")]

[StringLength(17, MinimumLength = 17, ErrorMessage = "VIN-номер должен содержать 17 символов")]

public string VIN { get; set; }

[Required(ErrorMessage = "Цена обязательна")]

[Column(TypeName = "decimal(18, 2)")]

public decimal Price { get; set; }

[Required(ErrorMessage = "Состояние обязательно")]

[StringLength(50)]

public string Condition { get; set; } // Например: "Новый", "Б/У", "После ремонта"

[Required(ErrorMessage = "Дата поступления обязательна")]

[DataType(DataType.Date)]

[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]

public DateTime ArrivalDate { get; set; } = DateTime.Now; // По умолчанию текущая дата

[StringLength(500)]

public string? PhotoUrl { get; set; } // Путь к фотографии автомобиля

[NotMapped] // Это свойство не будет отображаться в базе данных

[Display(Name = "Загрузить фото")]

public IFormFile? PhotoFile { get; set; } // Для загрузки файла через форму

// Навигационные свойства

public ICollection<Deal>? Deals { get; set; }

public ICollection<AutoServiceEntry>? AutoServiceEntries { get; set; }

public ICollection<ApplicationRequest>? ApplicationRequests { get; set; }

}

}

Client.cs

using System.ComponentModel.DataAnnotations;

using System.Collections.Generic;

namespace WestMotorsApp.Models

{

public class Client

{

public int Id { get; set; }

[Required(ErrorMessage = "ФИО клиента обязательно")]

[StringLength(100)]

public string FullName { get; set; } = string.Empty;

[Required(ErrorMessage = "Паспортные данные обязательны")]

[StringLength(200)]

public string PassportData { get; set; } = string.Empty;

[Required(ErrorMessage = "Контактные данные обязательны")]

[StringLength(200)]

public string ContactInfo { get; set; } = string.Empty;

[Display(Name = "Адрес")]

[StringLength(500)]

public string? Address { get; set; }

public string? Preferences { get; set; }

public ICollection<Deal>? Deals { get; set; }

public ICollection<ApplicationRequest>? ApplicationRequests { get; set; }

}

}

ApplicationUser.cs

using System.ComponentModel.DataAnnotations;

using System.Collections.Generic;

namespace WestMotorsApp.Models

{

public class Client

{

public int Id { get; set; }

[Required(ErrorMessage = "ФИО клиента обязательно")]

[StringLength(100)]

public string FullName { get; set; } = string.Empty;

[Required(ErrorMessage = "Паспортные данные обязательны")]

[StringLength(200)]

public string PassportData { get; set; } = string.Empty;

[Required(ErrorMessage = "Контактные данные обязательны")]

[StringLength(200)]

public string ContactInfo { get; set; } = string.Empty;

[Display(Name = "Адрес")]

[StringLength(500)]

public string? Address { get; set; }

public string? Preferences { get; set; }

public ICollection<Deal>? Deals { get; set; }

public ICollection<ApplicationRequest>? ApplicationRequests { get; set; }

}

}

AutoServiceEntry.cs

using System.ComponentModel.DataAnnotations;

using System.Collections.Generic;

namespace WestMotorsApp.Models

{

public class Client

{

public int Id { get; set; }

[Required(ErrorMessage = "ФИО клиента обязательно")]

[StringLength(100)]

public string FullName { get; set; } = string.Empty;

[Required(ErrorMessage = "Паспортные данные обязательны")]

[StringLength(200)]

public string PassportData { get; set; } = string.Empty;

[Required(ErrorMessage = "Контактные данные обязательны")]

[StringLength(200)]

public string ContactInfo { get; set; } = string.Empty;

[Display(Name = "Адрес")]

[StringLength(500)]

public string? Address { get; set; }

public string? Preferences { get; set; }

public ICollection<Deal>? Deals { get; set; }

public ICollection<ApplicationRequest>? ApplicationRequests { get; set; }

}

}

Deal.cs

using System;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace WestMotorsApp.Models

{

public class Deal

{

public int Id { get; set; }

[Required(ErrorMessage = "Автомобиль обязателен")]

[Display(Name = "Автомобиль")]

public int CarId { get; set; }

public Car Car { get; set; }

[Display(Name = "Продавец")]

public string SellerId { get; set; }

[ForeignKey("SellerId")]

public ApplicationUser Seller { get; set; }

[Required(ErrorMessage = "Покупатель обязателен")]

[Display(Name = "Покупатель")]

public int BuyerId { get; set; }

public Client Buyer { get; set; }

[Required(ErrorMessage = "Дата сделки обязательна")]

[Display(Name = "Дата сделки")]

[DataType(DataType.Date)]

public DateTime DealDate { get; set; } = DateTime.Now;

[Required(ErrorMessage = "Итоговая стоимость обязательна")]

[Column(TypeName = "decimal(18, 2)")]

[Display(Name = "Итоговая стоимость")]

public decimal FinalCost { get; set; }

[Required(ErrorMessage = "Способ оплаты обязателен")]

[StringLength(50)]

[Display(Name = "Способ оплаты")]

public string PaymentMethod { get; set; }

}

}

ErrorViewModel.cs

namespace WestMotors.Models

{

public class ErrorViewModel

{

public string? RequestId { get; set; }

public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

}

}

SeedData.cs

using Microsoft.AspNetCore.Identity;

using WestMotorsApp.Models; // Убедитесь, что namespace соответствует вашему проекту

namespace WestMotorsApp.Services

{

public static class SeedData

{

public static async Task Initialize(

UserManager<ApplicationUser> userManager,

RoleManager<IdentityRole> roleManager)

{

// Создаем роли, если их нет

string[] roleNames = { "Администратор", "Менеджер", "Клиент" };

foreach (var roleName in roleNames)

{

if (!await roleManager.RoleExistsAsync(roleName))

{

await roleManager.CreateAsync(new IdentityRole(roleName));

}

}

// Создаем пользователя-администратора, если его нет

string adminEmail = "admin@westmotors.com";

string adminPassword = "AdminPassword123!";

if (await userManager.FindByEmailAsync(adminEmail) == null)

{

var adminUser = new ApplicationUser

{

UserName = adminEmail,

Email = adminEmail,

EmailConfirmed = true,

FullName = "Главный Администратор",

Position = "Администратор",

ContactInfo = "+1234567890"

};

var result = await userManager.CreateAsync(adminUser, adminPassword);

if (result.Succeeded)

{

await userManager.AddToRoleAsync(adminUser, "Администратор");

}

}

}

}

}

CarsController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using WestMotorsApp.Data;

using WestMotorsApp.Models;

using Microsoft.AspNetCore.Authorization; // Для авторизации

using System.IO; // Для работы с файлами

namespace WestMotorsApp.Controllers

{

public class CarsController : Controller

{

private readonly ApplicationDbContext \_context;

private readonly IWebHostEnvironment \_hostEnvironment; // Для доступа к путям файлов

public CarsController(ApplicationDbContext context, IWebHostEnvironment hostEnvironment)

{

\_context = context;

\_hostEnvironment = hostEnvironment;

}

// GET: Cars - Главный каталог для всех пользователей

public async Task<IActionResult> Index(string brand, int? minYear, int? maxMileage, decimal? maxPrice)

{

var cars = from c in \_context.Cars

select c;

if (!string.IsNullOrEmpty(brand))

{

cars = cars.Where(c => c.Brand.Contains(brand));

}

if (minYear.HasValue)

{

cars = cars.Where(c => c.ManufactureYear >= minYear.Value);

}

if (maxMileage.HasValue)

{

cars = cars.Where(c => c.Mileage <= maxMileage.Value);

}

if (maxPrice.HasValue)

{

cars = cars.Where(c => c.Price <= maxPrice.Value);

}

return View(await cars.ToListAsync());

}

// GET: Cars/Details/5 - Подробности об автомобиле

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

{

if (id == null)

{

return NotFound();

}

var car = await \_context.Cars

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

if (car == null)

{

return NotFound();

}

return View(car);

}

// GET: Cars/Create - Добавление нового автомобиля (Только для Администратора)

[Authorize(Roles = "Администратор")]

public IActionResult Create()

{

return View();

}

// POST: Cars/Create

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Администратор")]

public async Task<IActionResult> Create([Bind("Id,Brand,Model,ManufactureYear,Mileage,VIN,Price,Condition,ArrivalDate,PhotoFile")] Car car)

{

if (ModelState.IsValid)

{

if (car.PhotoFile != null)

{

// Сохранение файла

string wwwRootPath = \_hostEnvironment.WebRootPath;

string fileName = Guid.NewGuid().ToString() + Path.GetExtension(car.PhotoFile.FileName);

string path = Path.Combine(wwwRootPath, "uploads", fileName);

using (var fileStream = new FileStream(path, FileMode.Create))

{

await car.PhotoFile.CopyToAsync(fileStream);

}

car.PhotoUrl = "/uploads/" + fileName; // Сохраняем относительный путь в БД

}

\_context.Add(car);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(car);

}

// GET: Cars/Edit/5 - Редактирование автомобиля (Только для Администратора)

[Authorize(Roles = "Администратор")]

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

{

if (id == null)

{

return NotFound();

}

var car = await \_context.Cars.FindAsync(id);

if (car == null)

{

return NotFound();

}

return View(car);

}

// POST: Cars/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Администратор")]

public async Task<IActionResult> Edit(int id, [Bind("Id,Brand,Model,ManufactureYear,Mileage,VIN,Price,Condition,ArrivalDate,PhotoUrl,PhotoFile")] Car car)

{

if (id != car.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

if (car.PhotoFile != null)

{

// Удалить старое фото, если есть

if (!string.IsNullOrEmpty(car.PhotoUrl))

{

string oldImagePath = Path.Combine(\_hostEnvironment.WebRootPath, car.PhotoUrl.TrimStart('/'));

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

{

System.IO.File.Delete(oldImagePath);

}

}

// Сохранить новое фото

string wwwRootPath = \_hostEnvironment.WebRootPath;

string fileName = Guid.NewGuid().ToString() + Path.GetExtension(car.PhotoFile.FileName);

string path = Path.Combine(wwwRootPath, "uploads", fileName);

using (var fileStream = new FileStream(path, FileMode.Create))

{

await car.PhotoFile.CopyToAsync(fileStream);

}

car.PhotoUrl = "/uploads/" + fileName;

}

else if (string.IsNullOrEmpty(car.PhotoUrl)) // Если файл не загружен и PhotoUrl очищен

{

// Если старое фото было и его удалили

var existingCar = await \_context.Cars.AsNoTracking().FirstOrDefaultAsync(c => c.Id == id);

if (existingCar != null && !string.IsNullOrEmpty(existingCar.PhotoUrl))

{

string oldImagePath = Path.Combine(\_hostEnvironment.WebRootPath, existingCar.PhotoUrl.TrimStart('/'));

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

{

System.IO.File.Delete(oldImagePath);

}

}

}

\_context.Update(car);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!CarExists(car.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(car);

}

// GET: Cars/Delete/5 - Удаление автомобиля (Только для Администратора)

[Authorize(Roles = "Администратор")]

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

{

if (id == null)

{

return NotFound();

}

var car = await \_context.Cars

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

if (car == null)

{

return NotFound();

}

return View(car);

}

// POST: Cars/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Администратор")]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var car = await \_context.Cars.FindAsync(id);

if (car != null)

{

// Удаление файла фотографии

if (!string.IsNullOrEmpty(car.PhotoUrl))

{

string imagePath = Path.Combine(\_hostEnvironment.WebRootPath, car.PhotoUrl.TrimStart('/'));

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

{

System.IO.File.Delete(imagePath);

}

}

\_context.Cars.Remove(car);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool CarExists(int id)

{

return \_context.Cars.Any(e => e.Id == id);

}

}

}

ApplicationRequestsController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using WestMotorsApp.Data;

using WestMotorsApp.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.Extensions.Logging;

namespace WestMotorsApp.Controllers

{

public class ApplicationRequestsController : Controller

{

private readonly ApplicationDbContext \_context;

private readonly UserManager<ApplicationUser> \_userManager;

private readonly ILogger<ApplicationRequestsController> \_logger;

public ApplicationRequestsController(ApplicationDbContext context, UserManager<ApplicationUser> userManager, ILogger<ApplicationRequestsController> logger)

{

\_context = context;

\_userManager = userManager;

\_logger = logger;

}

[Authorize(Roles = "Менеджер,Администратор")]

public async Task<IActionResult> Index()

{

var applicationDbContext = \_context.ApplicationRequests

.Include(a => a.Car)

.Include(a => a.Client);

return View(await applicationDbContext.ToListAsync());

}

[Authorize(Roles = "Менеджер,Администратор")]

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

{

if (id == null)

{

return NotFound();

}

var applicationRequest = await \_context.ApplicationRequests

.Include(a => a.Car)

.Include(a => a.Client)

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

if (applicationRequest == null)

{

return NotFound();

}

return View(applicationRequest);

}

[Authorize(Roles = "Клиент")]

public async Task<IActionResult> CreateForCar(int? carId)

{

if (carId == null)

{

TempData["ErrorMessage"] = "Автомобиль не выбран для заявки.";

return RedirectToAction("Index", "Cars");

}

var car = await \_context.Cars.FindAsync(carId);

if (car == null)

{

TempData["ErrorMessage"] = "Выбранный автомобиль не найден в базе данных.";

return NotFound();

}

var currentUser = await \_userManager.GetUserAsync(User);

if (currentUser == null)

{

\_logger.LogError("Ошибка: \_userManager.GetUserAsync(User) вернул null для залогиненного пользователя {UserName}", User.Identity?.Name);

TempData["ErrorMessage"] = "Произошла ошибка при определении данных пользователя. Пожалуйста, попробуйте войти снова.";

return RedirectToPage("/Account/Login", new { area = "Identity" });

}

currentUser = await \_context.Users

.Include(u => u.Client)

.FirstOrDefaultAsync(u => u.Id == currentUser.Id);

var applicationRequest = new ApplicationRequest

{

CarId = car.Id,

ClientId = currentUser?.ClientId,

RequestDate = DateTime.UtcNow,

Status = "Новая",

UserEmail = currentUser?.Email,

ClientFullName = currentUser?.Client?.FullName ?? currentUser?.FullName ?? currentUser?.Email ?? "Неизвестно"

};

await PopulateViewBagsForCreateForCar(applicationRequest, currentUser);

return View(applicationRequest);

}

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Клиент")]

public async Task<IActionResult> CreateForCar([Bind("CarId,RequestType,ClientFullName,ContactInfo,PreferredContactMethod")] ApplicationRequest applicationRequest)

{

var currentUser = await \_userManager.GetUserAsync(User);

if (currentUser == null)

{

ModelState.AddModelError(string.Empty, "Ошибка: Пользователь не найден. Пожалуйста, попробуйте войти снова.");

await PopulateViewBagsForCreateForCar(applicationRequest, null);

return View(applicationRequest);

}

currentUser = await \_context.Users

.Include(u => u.Client)

.FirstOrDefaultAsync(u => u.Id == currentUser.Id);

applicationRequest.ClientId = currentUser?.ClientId;

applicationRequest.UserEmail = currentUser?.Email;

applicationRequest.RequestDate = DateTime.UtcNow;

applicationRequest.Status = "Новая";

if (applicationRequest.PreferredContactMethod == "Другой")

{

if (string.IsNullOrWhiteSpace(applicationRequest.ContactInfo))

{

ModelState.AddModelError("ContactInfo", "Пожалуйста, введите контактные данные, если выбран 'Другой' способ связи.");

}

}

else

{

if (applicationRequest.ContactInfo != applicationRequest.PreferredContactMethod)

{

if (string.IsNullOrWhiteSpace(applicationRequest.ContactInfo))

{

applicationRequest.ContactInfo = applicationRequest.PreferredContactMethod;

}

else

{

applicationRequest.ContactInfo = applicationRequest.PreferredContactMethod;

}

}

}

if (string.IsNullOrWhiteSpace(applicationRequest.ClientFullName))

{

ModelState.AddModelError("ClientFullName", "ФИО клиента обязательно.");

}

ModelState.Remove(nameof(applicationRequest.Car));

if (ModelState.IsValid)

{

try

{

\_context.Add(applicationRequest);

await \_context.SaveChangesAsync();

TempData["SuccessMessage"] = "Ваша заявка успешно отправлена!";

return RedirectToAction("MyApplications");

}

catch (DbUpdateException dbEx)

{

\_logger.LogError(dbEx, "Ошибка базы данных при сохранении заявки: {Message}", dbEx.Message);

var innerException = dbEx.InnerException;

while (innerException != null)

{

\_logger.LogError("Inner Exception: {Message}", innerException.Message);

innerException = innerException.InnerException;

}

ModelState.AddModelError("", "Произошла ошибка при сохранении заявки в базу данных. Пожалуйста, проверьте вводимые данные или обратитесь в поддержку.");

if (applicationRequest.CarId > 0 && !await \_context.Cars.AnyAsync(c => c.Id == applicationRequest.CarId))

{

ModelState.AddModelError("CarId", "Выбранный автомобиль не существует.");

}

}

catch (Exception ex)

{

\_logger.LogError(ex, "Неизвестная ошибка при сохранении заявки: {Message}", ex.Message);

ModelState.AddModelError("", "Произошла непредвиденная ошибка при отправке заявки. Пожалуйста, попробуйте еще раз.");

}

}

await PopulateViewBagsForCreateForCar(applicationRequest, currentUser);

return View(applicationRequest);

}

private async Task PopulateViewBagsForCreateForCar(ApplicationRequest model, ApplicationUser? currentUser)

{

if (model.CarId > 0)

{

ViewBag.Car = await \_context.Cars.FindAsync(model.CarId);

}

else

{

ViewBag.Car = null;

}

List<SelectListItem> contactOptions = new List<SelectListItem>();

if (currentUser != null && !string.IsNullOrEmpty(currentUser.Email))

{

contactOptions.Add(new SelectListItem { Value = currentUser.Email, Text = $"Email: {currentUser.Email}" });

}

if (currentUser?.Client != null && !string.IsNullOrEmpty(currentUser.Client.ContactInfo) && currentUser.Client.ContactInfo != currentUser?.Email)

{

contactOptions.Add(new SelectListItem { Value = currentUser.Client.ContactInfo, Text = $"Телефон/Другой: {currentUser.Client.ContactInfo}" });

}

contactOptions.Add(new SelectListItem { Value = "Другой", Text = "Другой (ввести вручную)" });

ViewBag.ContactOptions = contactOptions;

if (string.IsNullOrEmpty(model.PreferredContactMethod) && contactOptions.Any())

{

model.PreferredContactMethod = contactOptions.First().Value;

if (model.PreferredContactMethod != "Другой")

{

model.ContactInfo = model.PreferredContactMethod;

}

}

else if (model.PreferredContactMethod != "Другой" && string.IsNullOrEmpty(model.ContactInfo))

{

model.ContactInfo = model.PreferredContactMethod;

}

}

[Authorize(Roles = "Клиент")]

public async Task<IActionResult> MyApplications()

{

var currentUser = await \_userManager.GetUserAsync(User);

if (currentUser == null)

{

TempData["ErrorMessage"] = "Пользователь не найден. Пожалуйста, войдите в систему.";

return RedirectToPage("/Account/Login", new { area = "Identity" });

}

if (!currentUser.ClientId.HasValue)

{

currentUser = await \_context.Users.Include(u => u.Client).FirstOrDefaultAsync(u => u.Id == currentUser.Id);

}

var clientApplications = await \_context.ApplicationRequests

.Include(ar => ar.Car)

.Include(ar => ar.Client)

.Where(ar => ar.UserEmail == currentUser.Email || (ar.ClientId.HasValue && ar.ClientId == currentUser.ClientId))

.OrderByDescending(a => a.RequestDate)

.ToListAsync();

return View(clientApplications);

}

[Authorize(Roles = "Менеджер,Администратор")]

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

{

if (id == null)

{

return NotFound();

}

var applicationRequest = await \_context.ApplicationRequests.FindAsync(id);

if (applicationRequest == null)

{

return NotFound();

}

ViewData["CarId"] = new SelectList(\_context.Cars, "Id", "Brand", applicationRequest.CarId);

ViewData["ClientId"] = new SelectList(\_context.Clients, "Id", "FullName", applicationRequest.ClientId);

return View(applicationRequest);

}

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Менеджер,Администратор")]

public async Task<IActionResult> Edit(int id, [Bind("Id,ClientId,CarId,RequestType,RequestDate,Status,ManagerNotes")] ApplicationRequest applicationRequest)

{

if (id != applicationRequest.Id)

{

return NotFound();

}

var existingRequest = await \_context.ApplicationRequests.AsNoTracking().FirstOrDefaultAsync(r => r.Id == id);

if (existingRequest == null)

{

return NotFound();

}

existingRequest.RequestType = applicationRequest.RequestType;

existingRequest.RequestDate = applicationRequest.RequestDate;

existingRequest.Status = applicationRequest.Status;

existingRequest.ManagerNotes = applicationRequest.ManagerNotes;

existingRequest.CarId = applicationRequest.CarId;

existingRequest.ClientId = applicationRequest.ClientId;

if (ModelState.IsValid)

{

try

{

\_context.Update(existingRequest);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ApplicationRequestExists(applicationRequest.Id))

{

return NotFound();

}

else

{

throw;

}

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка при редактировании заявки: {Message}", ex.Message);

ModelState.AddModelError("", "Произошла ошибка при сохранении изменений. Пожалуйста, попробуйте еще раз.");

}

return RedirectToAction(nameof(Index));

}

ViewData["CarId"] = new SelectList(\_context.Cars, "Id", "Brand", applicationRequest.CarId);

ViewData["ClientId"] = new SelectList(\_context.Clients, "Id", "FullName", applicationRequest.ClientId);

return View(applicationRequest);

}

[Authorize(Roles = "Администратор")]

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

{

if (id == null)

{

return NotFound();

}

var applicationRequest = await \_context.ApplicationRequests

.Include(a => a.Car)

.Include(a => a.Client)

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

if (applicationRequest == null)

{

return NotFound();

}

return View(applicationRequest);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Администратор")]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var applicationRequest = await \_context.ApplicationRequests.FindAsync(id);

if (applicationRequest != null)

{

\_context.ApplicationRequests.Remove(applicationRequest);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool ApplicationRequestExists(int id)

{

return \_context.ApplicationRequests.Any(e => e.Id == id);

}

}

}

ClientsController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using WestMotorsApp.Data;

using WestMotorsApp.Models;

using Microsoft.AspNetCore.Authorization;

namespace WestMotorsApp.Controllers

{

[Authorize(Roles = "Менеджер,Администратор")] // Только менеджеры и администраторы

public class ClientsController : Controller

{

private readonly ApplicationDbContext \_context;

public ClientsController(ApplicationDbContext context)

{

\_context = context;

}

// GET: Clients

public async Task<IActionResult> Index()

{

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

}

// GET: Clients/Details/5

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

{

if (id == null)

{

return NotFound();

}

var client = await \_context.Clients

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

if (client == null)

{

return NotFound();

}

return View(client);

}

// GET: Clients/Create

public IActionResult Create()

{

return View();

}

// POST: Clients/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,FullName,PassportData,ContactInfo,Preferences")] Client client)

{

if (ModelState.IsValid)

{

\_context.Add(client);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(client);

}

// GET: Clients/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var client = await \_context.Clients.FindAsync(id);

if (client == null)

{

return NotFound();

}

return View(client);

}

// POST: Clients/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,FullName,PassportData,ContactInfo,Preferences")] Client client)

{

if (id != client.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(client);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ClientExists(client.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(client);

}

// GET: Clients/Delete/5

[Authorize(Roles = "Администратор")] // Удалять клиентов может только администратор

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

{

if (id == null)

{

return NotFound();

}

var client = await \_context.Clients

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

if (client == null)

{

return NotFound();

}

return View(client);

}

// POST: Clients/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Администратор")]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var client = await \_context.Clients.FindAsync(id);

if (client != null)

{

\_context.Clients.Remove(client);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool ClientExists(int id)

{

return \_context.Clients.Any(e => e.Id == id);

}

}

}

DealsController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using WestMotorsApp.Data;

using WestMotorsApp.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using System.IO;

using Microsoft.AspNetCore.Hosting;

using NPOI.XWPF.UserModel;

using NPOI.OpenXmlFormats.Wordprocessing;

namespace WestMotorsApp.Controllers

{

[Authorize(Roles = "Менеджер,Администратор")]

public class DealsController : Controller

{

private readonly ApplicationDbContext \_context;

private readonly UserManager<ApplicationUser> \_userManager;

private readonly IWebHostEnvironment \_webHostEnvironment;

public DealsController(ApplicationDbContext context, UserManager<ApplicationUser> userManager, IWebHostEnvironment webHostEnvironment)

{

\_context = context;

\_userManager = userManager;

\_webHostEnvironment = webHostEnvironment;

}

public async Task<IActionResult> Index()

{

var applicationDbContext = \_context.Deals

.Include(d => d.Buyer)

.Include(d => d.Car)

.Include(d => d.Seller);

return View(await applicationDbContext.ToListAsync());

}

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

{

if (id == null)

{

return NotFound();

}

var deal = await \_context.Deals

.Include(d => d.Buyer)

.Include(d => d.Car)

.Include(d => d.Seller)

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

if (deal == null)

{

return NotFound();

}

return View(deal);

}

public async Task<IActionResult> Create()

{

ViewData["BuyerId"] = new SelectList(\_context.Clients, "Id", "FullName");

ViewData["CarId"] = new SelectList(\_context.Cars.Where(c => !c.Deals.Any()), "Id", "VIN");

var currentUser = await \_userManager.GetUserAsync(User);

ViewData["SellerId"] = new SelectList(new List<ApplicationUser> { currentUser }, "Id", "FullName", currentUser.Id);

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,CarId,SellerId,BuyerId,DealDate,FinalCost,PaymentMethod")] Deal deal)

{

ModelState.Remove("Car");

ModelState.Remove("Buyer");

ModelState.Remove("Seller");

if (ModelState.IsValid)

{

var currentUser = await \_userManager.GetUserAsync(User);

if (currentUser == null)

{

TempData["ErrorMessage"] = "Не удалось определить продавца. Пожалуйста, войдите снова.";

ViewData["BuyerId"] = new SelectList(\_context.Clients, "Id", "FullName", deal.BuyerId);

ViewData["CarId"] = new SelectList(\_context.Cars.Where(c => !c.Deals.Any()), "Id", "VIN", deal.CarId);

ViewData["SellerId"] = new SelectList(new List<ApplicationUser> { currentUser }, "Id", "FullName", currentUser.Id);

return View(deal);

}

deal.SellerId = currentUser.Id;

var existingDealForCar = await \_context.Deals.AnyAsync(d => d.CarId == deal.CarId);

if (existingDealForCar)

{

ModelState.AddModelError("CarId", "Этот автомобиль уже продан.");

ViewData["BuyerId"] = new SelectList(\_context.Clients, "Id", "FullName", deal.BuyerId);

ViewData["CarId"] = new SelectList(\_context.Cars.Where(c => !c.Deals.Any()), "Id", "VIN", deal.CarId);

ViewData["SellerId"] = new SelectList(new List<ApplicationUser> { currentUser }, "Id", "FullName", currentUser.Id);

return View(deal);

}

\_context.Add(deal);

await \_context.SaveChangesAsync();

currentUser.SoldCarsCount++;

await \_userManager.UpdateAsync(currentUser);

TempData["SuccessMessage"] = "Сделка успешно создана!";

return RedirectToAction(nameof(Index));

}

ViewData["BuyerId"] = new SelectList(\_context.Clients, "Id", "FullName", deal.BuyerId);

ViewData["CarId"] = new SelectList(\_context.Cars.Where(c => !c.Deals.Any()), "Id", "VIN", deal.CarId);

var currentUserForView = await \_userManager.GetUserAsync(User);

ViewData["SellerId"] = new SelectList(new List<ApplicationUser> { currentUserForView }, "Id", "FullName", currentUserForView.Id);

TempData["ErrorMessage"] = "Не удалось создать сделку. Пожалуйста, проверьте введённые данные.";

return View(deal);

}

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

{

if (id == null)

{

return NotFound();

}

var deal = await \_context.Deals.FindAsync(id);

if (deal == null)

{

return NotFound();

}

ViewData["BuyerId"] = new SelectList(\_context.Clients, "Id", "FullName", deal.BuyerId);

ViewData["CarId"] = new SelectList(\_context.Cars, "Id", "VIN", deal.CarId);

ViewData["SellerId"] = new SelectList(\_context.Users, "Id", "FullName", deal.SellerId);

return View(deal);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,CarId,SellerId,BuyerId,DealDate,FinalCost,PaymentMethod")] Deal deal)

{

if (id != deal.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

var originalCarId = (await \_context.Deals.AsNoTracking().FirstOrDefaultAsync(d => d.Id == id))?.CarId;

if (originalCarId != deal.CarId && await \_context.Deals.AnyAsync(d => d.CarId == deal.CarId))

{

ModelState.AddModelError("CarId", "Этот автомобиль уже продан в другой сделке.");

ViewData["BuyerId"] = new SelectList(\_context.Clients, "Id", "FullName", deal.BuyerId);

ViewData["CarId"] = new SelectList(\_context.Cars, "Id", "VIN", deal.CarId);

ViewData["SellerId"] = new SelectList(\_context.Users, "Id", "FullName", deal.SellerId);

return View(deal);

}

\_context.Update(deal);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!DealExists(deal.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["BuyerId"] = new SelectList(\_context.Clients, "Id", "FullName", deal.BuyerId);

ViewData["CarId"] = new SelectList(\_context.Cars, "Id", "VIN", deal.CarId);

ViewData["SellerId"] = new SelectList(\_context.Users, "Id", "FullName", deal.SellerId);

return View(deal);

}

[Authorize(Roles = "Администратор")]

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

{

if (id == null)

{

return NotFound();

}

var deal = await \_context.Deals

.Include(d => d.Buyer)

.Include(d => d.Car)

.Include(d => d.Seller)

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

if (deal == null)

{

return NotFound();

}

return View(deal);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Администратор")]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var deal = await \_context.Deals.FindAsync(id);

if (deal != null)

{

\_context.Deals.Remove(deal);

var seller = await \_userManager.FindByIdAsync(deal.SellerId);

if (seller != null)

{

seller.SoldCarsCount--;

await \_userManager.UpdateAsync(seller);

}

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool DealExists(int id)

{

return \_context.Deals.Any(e => e.Id == id);

}

// Метод для генерации договора купли-продажи в Word

public async Task<IActionResult> GenerateContract(int id)

{

var deal = await \_context.Deals

.Include(d => d.Buyer)

.Include(d => d.Car)

.Include(d => d.Seller)

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

if (deal == null)

{

return NotFound();

}

XWPFDocument document = new XWPFDocument();

// Заголовок договора

XWPFParagraph titleParagraph = document.CreateParagraph();

titleParagraph.Alignment = ParagraphAlignment.CENTER;

XWPFRun titleRun = titleParagraph.CreateRun();

titleRun.SetText("ДОГОВОР КУПЛИ-ПРОДАЖИ АВТОМОБИЛЯ №" + deal.Id);

titleRun.FontSize = 16;

titleRun.IsBold = true;

document.CreateParagraph(); // Пустая строка

// Место и дата

XWPFParagraph headerParagraph = document.CreateParagraph();

headerParagraph.Alignment = ParagraphAlignment.RIGHT;

XWPFRun headerRun = headerParagraph.CreateRun();

headerRun.SetText($"г. [Город], {deal.DealDate:«dd» MMMM yyyy г.}");

document.CreateParagraph(); // Пустая строка

// Стороны договора

XWPFParagraph partiesParagraph = document.CreateParagraph();

XWPFRun partiesRun = partiesParagraph.CreateRun();

partiesRun.SetText($"Мы, нижеподписавшиеся, {deal.Seller.FullName}, именуемый в дальнейшем «Продавец», с одной стороны, и {deal.Buyer.FullName}, именуемый в дальнейшем «Покупатель», с другой стороны, заключили настоящий Договор о нижеследующем:");

document.CreateParagraph(); // Пустая строка

// Предмет договора

XWPFParagraph subjectTitle = document.CreateParagraph();

XWPFRun subjectTitleRun = subjectTitle.CreateRun();

subjectTitleRun.IsBold = true;

subjectTitleRun.SetText("1. ПРЕДМЕТ ДОГОВОРА");

document.CreateParagraph().CreateRun().SetText($"1.1. Продавец обязуется передать в собственность Покупателю, а Покупатель обязуется принять и оплатить автомобиль марки {deal.Car.Brand}, модель {deal.Car.Model}, VIN {deal.Car.VIN}, год выпуска {deal.Car.ManufactureYear}, пробег {deal.Car.Mileage} км, в состоянии «{deal.Car.Condition}» (далее – «Автомобиль»).");

document.CreateParagraph().CreateRun().SetText("1.2. Автомобиль принадлежит Продавцу на праве собственности.");

document.CreateParagraph();

// Цена и порядок расчетов

XWPFParagraph priceTitle = document.CreateParagraph();

XWPFRun priceTitleRun = priceTitle.CreateRun();

priceTitleRun.IsBold = true;

priceTitleRun.SetText("2. ЦЕНА И ПОРЯДОК РАСЧЕТОВ");

document.CreateParagraph().CreateRun().SetText($"2.1. Стоимость Автомобиля по настоящему Договору составляет {deal.FinalCost:C}.");

document.CreateParagraph().CreateRun().SetText($"2.2. Оплата производится Покупателем Продавцу в размере {deal.FinalCost:C} путем {deal.PaymentMethod}.");

document.CreateParagraph();

// Права и обязанности сторон

XWPFParagraph rightsTitle = document.CreateParagraph();

XWPFRun rightsTitleRun = rightsTitle.CreateRun();

rightsTitleRun.IsBold = true;

rightsTitleRun.SetText("3. ПРАВА И ОБЯЗАННОСТИ СТОРОН");

document.CreateParagraph().CreateRun().SetText("3.1. Продавец обязуется передать Автомобиль Покупателю в срок и на условиях, предусмотренных настоящим Договором.");

document.CreateParagraph().CreateRun().SetText("3.2. Покупатель обязуется принять и оплатить Автомобиль в соответствии с условиями настоящего Договора.");

document.CreateParagraph();

// Заключительные положения

XWPFParagraph finalTitle = document.CreateParagraph();

XWPFRun finalTitleRun = finalTitle.CreateRun();

finalTitleRun.IsBold = true;

finalTitleRun.SetText("4. ЗАКЛЮЧИТЕЛЬНЫЕ ПОЛОЖЕНИЯ");

document.CreateParagraph().CreateRun().SetText("4.1. Настоящий Договор вступает в силу с момента его подписания Сторонами.");

document.CreateParagraph().CreateRun().SetText("4.2. Все изменения и дополнения к настоящему Договору действительны лишь в том случае, если они совершены в письменной форме и подписаны уполномоченными представителями Сторон.");

document.CreateParagraph();

// Реквизиты сторон

XWPFParagraph requisitesTitle = document.CreateParagraph();

XWPFRun requisitesTitleRun = requisitesTitle.CreateRun();

requisitesTitleRun.IsBold = true;

requisitesTitleRun.SetText("РЕКВИЗИТЫ СТОРОН:");

XWPFTable table = document.CreateTable(1, 2); // 1 строка, 2 столбца для начала

table.SetColumnWidth(0, 4500); // Ширина для первой колонки

table.SetColumnWidth(1, 4500); // Ширина для второй колонки

// Продавец

XWPFParagraph p1 = table.GetRow(0).GetCell(0).AddParagraph();

XWPFRun r1 = p1.CreateRun();

r1.IsBold = true;

r1.SetText("Продавец:");

table.GetRow(0).GetCell(0).AddParagraph().CreateRun().SetText($"ФИО: {deal.Seller.FullName}");

table.GetRow(0).GetCell(0).AddParagraph().CreateRun().SetText($"Должность: {deal.Seller.Position}");

table.GetRow(0).GetCell(0).AddParagraph().CreateRun().SetText($"Email: {deal.Seller.Email}");

table.GetRow(0).GetCell(0).AddParagraph().CreateRun().SetText("Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

// Покупатель

XWPFParagraph p2 = table.GetRow(0).GetCell(1).AddParagraph();

XWPFRun r2 = p2.CreateRun();

r2.IsBold = true;

r2.SetText("Покупатель:");

table.GetRow(0).GetCell(1).AddParagraph().CreateRun().SetText($"ФИО: {deal.Buyer.FullName}");

table.GetRow(0).GetCell(1).AddParagraph().CreateRun().SetText($"Контакт: {deal.Buyer.ContactInfo}");

table.GetRow(0).GetCell(1).AddParagraph().CreateRun().SetText($"Паспорт: {deal.Buyer.PassportData}");

table.GetRow(0).GetCell(1).AddParagraph().CreateRun().SetText("Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

byte[] fileBytes;

using (MemoryStream outputStream = new MemoryStream())

{

document.Write(outputStream);

fileBytes = outputStream.ToArray();

}

return File(fileBytes, "application/vnd.openxmlformats-officedocument.wordprocessingml.document", $"Договор\_Купли-Продажи\_{deal.Id}.docx");

}

// Метод для генерации HTML-предварительного просмотра договора

public async Task<IActionResult> GetContractHtmlPreview(int id)

{

var deal = await \_context.Deals

.Include(d => d.Buyer)

.Include(d => d.Car)

.Include(d => d.Seller)

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

if (deal == null)

{

return NotFound();

}

// Формируем HTML-строку для предварительного просмотра

string htmlContent = $@"

<!DOCTYPE html>

<html>

<head>

<title>Предварительный просмотр договора</title>

<style>

body {{ font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif; margin: 20px; color: #333; font-size: 14px; }}

h2 {{ text-align: center; color: #0056b3; margin-bottom: 20px; }}

h3 {{ color: #0056b3; border-bottom: 1px solid #eee; padding-bottom: 5px; margin-top: 20px; }}

p {{ margin-bottom: 10px; line-height: 1.6; }}

.section-title {{ font-weight: bold; margin-top: 15px; margin-bottom: 5px; }}

.details-table {{ width: 100%; border-collapse: collapse; margin-top: 15px; }}

.details-table td {{ padding: 8px; border: 1px solid #ddd; vertical-align: top; }}

.details-table .label {{ font-weight: bold; width: 30%; }}

.signature-block {{ margin-top: 40px; display: flex; justify-content: space-around; }}

.signature-item {{ flex: 1; text-align: center; margin: 0 10px; }}

.signature-line {{ border-bottom: 1px solid #000; width: 80%; margin: 20px auto 5px auto; }}

</style>

</head>

<body>

<h2>ДОГОВОР КУПЛИ-ПРОДАЖИ АВТОМОБИЛЯ №{deal.Id}</h2>

<p style='text-align: right;'>г. [Город], {deal.DealDate:«dd» MMMM yyyy г.}</p>

<p>Мы, нижеподписавшиеся, <strong>{deal.Seller.FullName}</strong>, именуемый в дальнейшем «Продавец», с одной стороны, и <strong>{deal.Buyer.FullName}</strong>, именуемый в дальнейшем «Покупатель», с другой стороны, заключили настоящий Договор о нижеследующем:</p>

<h3>1. ПРЕДМЕТ ДОГОВОРА</h3>

<p>1.1. Продавец обязуется передать в собственность Покупателю, а Покупатель обязуется принять и оплатить автомобиль марки <strong>{deal.Car.Brand}</strong>, модель <strong>{deal.Car.Model}</strong>, VIN <strong>{deal.Car.VIN}</strong>, год выпуска <strong>{deal.Car.ManufactureYear}</strong>, пробег <strong>{deal.Car.Mileage} км</strong>, в состоянии «{deal.Car.Condition}» (далее – «Автомобиль»).</p>

<p>1.2. Автомобиль принадлежит Продавцу на праве собственности.</p>

<h3>2. ЦЕНА И ПОРЯДОК РАСЧЕТОВ</h3>

<p>2.1. Стоимость Автомобиля по настоящему Договору составляет <strong>{deal.FinalCost:C}</strong>.</p>

<p>2.2. Оплата производится Покупателем Продавцу в размере <strong>{deal.FinalCost:C}</strong> путем {deal.PaymentMethod}.</p>

<h3>3. ПРАВА И ОБЯЗАННОСТИ СТОРОН</h3>

<p>3.1. Продавец обязуется передать Автомобиль Покупателю в срок и на условиях, предусмотренных настоящим Договором.</p>

<p>3.2. Покупатель обязуется принять и оплатить Автомобиль в соответствии с условиями настоящего Договора.</p>

<h3>4. ЗАКЛЮЧИТЕЛЬНЫЕ ПОЛОЖЕНИЯ</h3>

<p>4.1. Настоящий Договор вступает в силу с момента его подписания Сторонами.</p>

<p>4.2. Все изменения и дополнения к настоящему Договору действительны лишь в том случае, если они совершены в письменной форме и подписаны уполномоченными представителями Сторон.</p>

<h3>РЕКВИЗИТЫ СТОРОН:</h3>

<table class='details-table'>

<tr>

<td style='width: 50%;'>

<p><strong>Продавец:</strong></p>

<p>ФИО: {deal.Seller.FullName}</p>

<p>Должность: {deal.Seller.Position}</p>

<p>Email: {deal.Seller.Email}</p>

<p style='margin-top: 30px;'>Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_</p>

</td>

<td style='width: 50%;'>

<p><strong>Покупатель:</strong></p>

<p>ФИО: {deal.Buyer.FullName}</p>

<p>Контакт: {deal.Buyer.ContactInfo}</p>

<p>Паспорт: {deal.Buyer.PassportData}</p>

<p style='margin-top: 30px;'>Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_</p>

</td>

</tr>

</table>

</body>

</html>";

return Content(htmlContent, "text/html");

}

private PictureType GetNPOIPictureType(string imagePath)

{

string extension = Path.GetExtension(imagePath)?.ToLowerInvariant();

switch (extension)

{

case ".jpg":

case ".jpeg":

return PictureType.JPEG;

case ".png":

return PictureType.PNG;

case ".gif":

return PictureType.GIF;

case ".bmp":

return PictureType.BMP;

case ".tiff":

return PictureType.TIFF;

default:

return PictureType.PNG;

}

}

}

}

HomeController.cs

using System.Diagnostics;

using Microsoft.AspNetCore.Mvc;

using WestMotors.Models;

namespace WestMotors.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

public IActionResult Index()

{

return View();

}

public IActionResult Privacy()

{

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

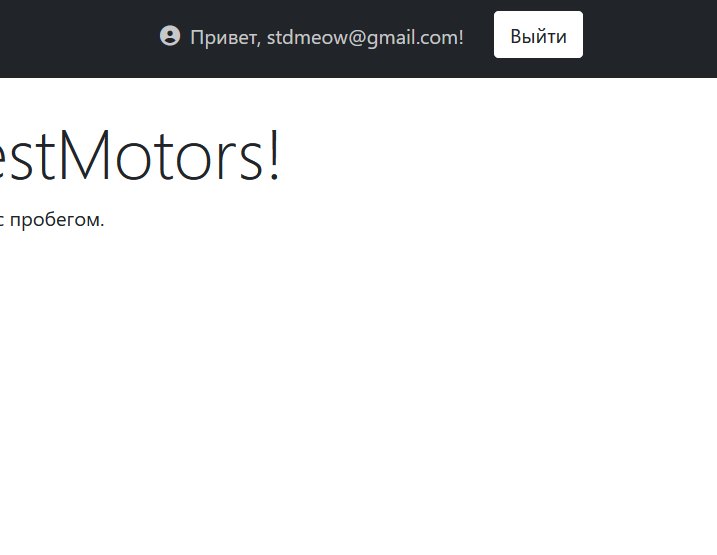
return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

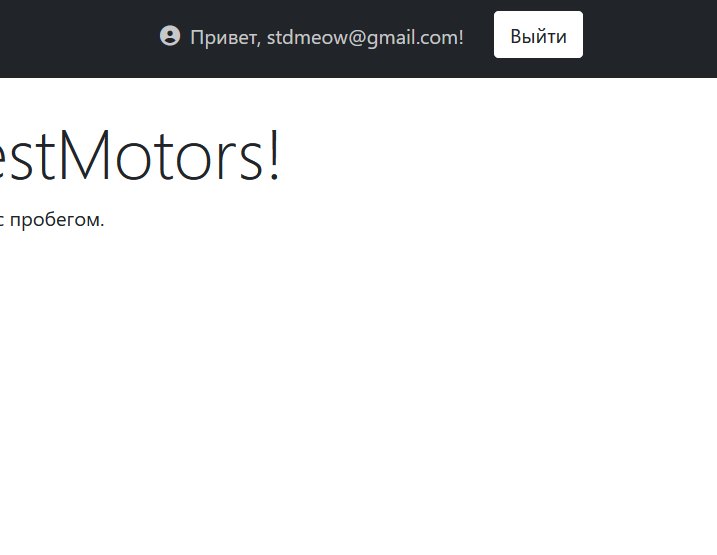
}

}

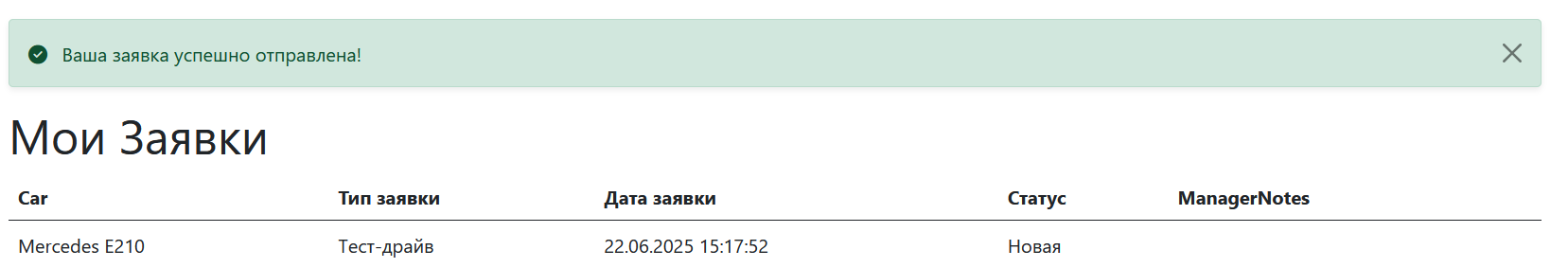
Приложение Д  
Результаты работы веб – приложения



1. ‒ Результат тестирование «Регистрация»



1. ‒ Результат тестирование «Авторизация»

****

1. ‒ Результат тестирование «Оставление заявки»

КП Т.318021.401 ГЧ

КБП

У

Т. Контр.

Разраб. Разраб.

Стадуб П.В.

Провер. Провер.

Кривошеина А.В.

Реценз.

Утверд.

Н. Контр.

Диаграмма прецедентов

№ докум.

Подпись

Дата

Изм.

Лист

Масса

Лит.

Масштаб

Лист 1

Листов 3

Инв.№подл.

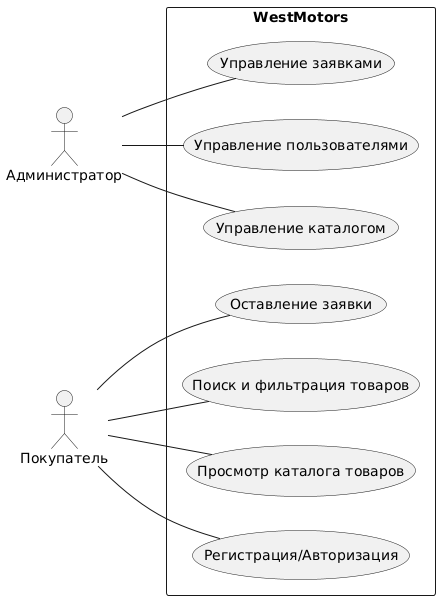
Подп. и дата

Взам.инв.№

Инв.№дубл.

Подп. и дата

КП Т.318021.401 ГЧ Т.ХХХХХХ.401 ГЧ



*Разработка веб-приложения для автоматизации работы автосалона «WEST MOTORS»*

*Разработка веб-приложения по автоматизации работы сети магазинов техники «COOLZY»*

КП Т.318021.401 ГЧ

КБП

У

Т. Контр.

Разраб. Разраб.

Стадуб П.В.

Провер. Провер.

Кривошеина А.В.

Реценз.

Утверд.

Н. Контр.

№ докум.

Подпись

Дата

Изм.

Лист

Масса

Лит.

Масштаб

Лист 2

Листов 3

Инв.№подл.

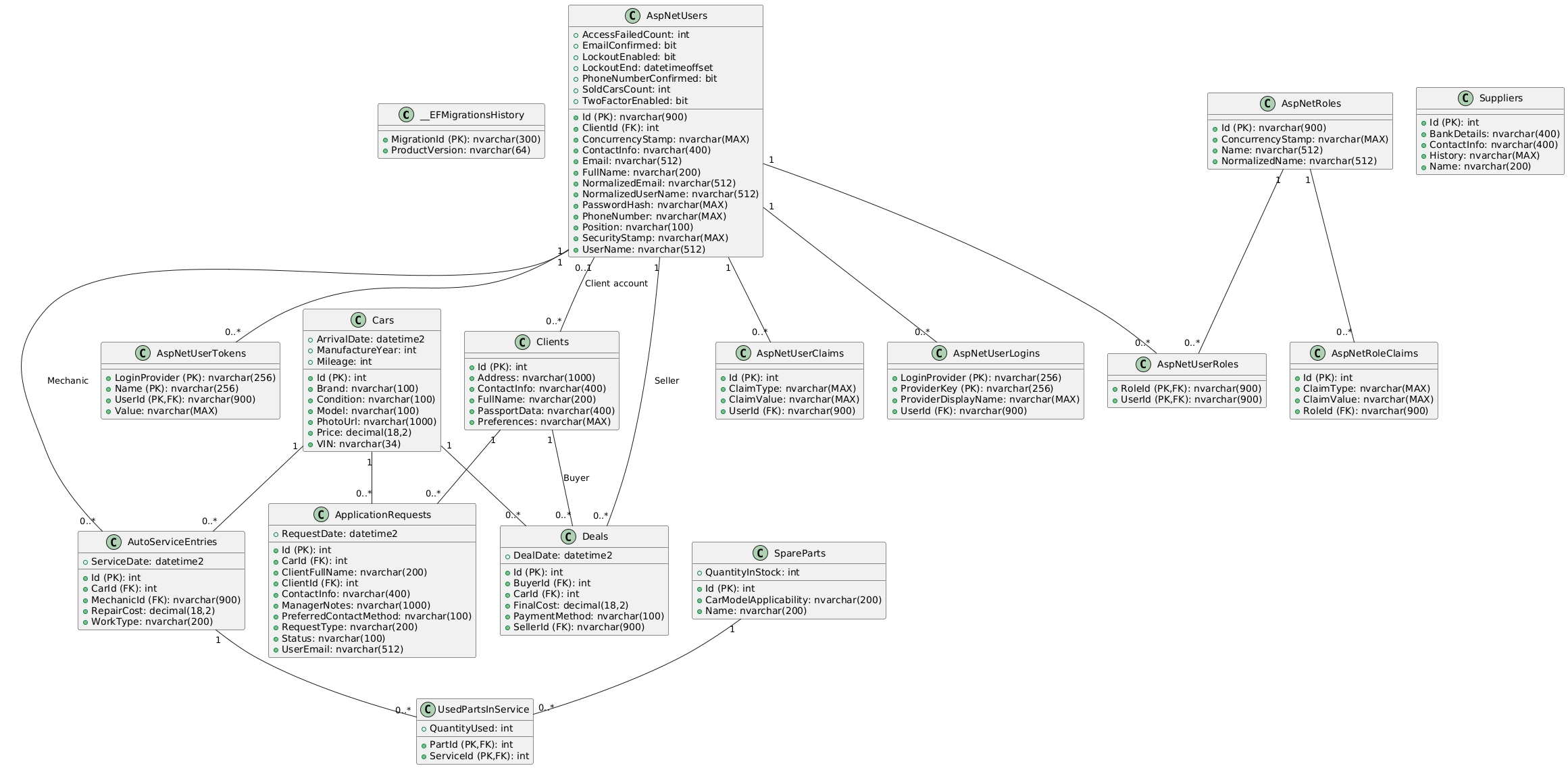
Подп. и дата

Взам.инв.№

Инв.№дубл.

Подп. и дата

КП Т.31021.401 ГЧ Т.ХХХХХХ.401 ГЧ



*Разработка веб-приложения для автоматизации работы автосалона «WEST MOTORS»*

Диаграмма классов

КП Т.318021.401 ГЧ

КБП

У

Т. Контр.

Разраб. Разраб.

Стадуб П.В.

Провер. Провер.

Кривошеина А.В.

Реценз.

Утверд.

Н. Контр.

№ докум.

Подпись

Дата

Изм.

Лист

Масса

Лит.

Масштаб

Лист 3

Листов 3

Инв.№подл.

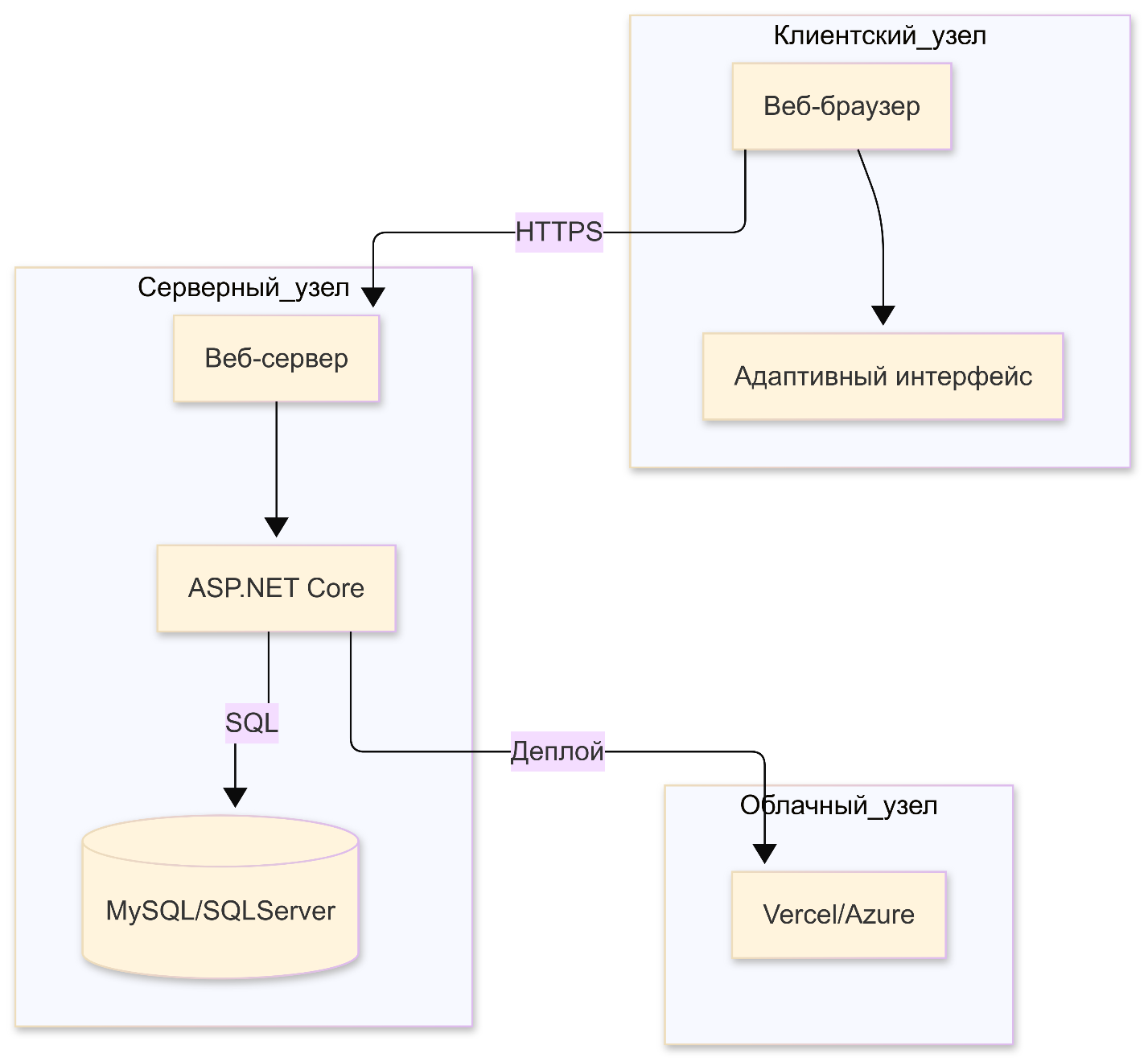
Подп. и дата

Взам.инв.№

Инв.№дубл.

Подп. и дата

КП Т.318021.401 ГЧ Т.ХХХХХХ.401 ГЧ



*Разработка веб-приложения для автоматизации работы автосалона «WEST MOTORS»*

Диаграмма развертывания

**Удостоверяющий лист**

электронного документа – курсовой проект

Тема КПРазработка веб-приложения для автоматизации работы автосалона «WEST MOTORS»

Обозначение КП Т.318021.401

Разработчик Стадуб П.В. Руководитель Кривошеина А.В.

(Ф.И.О.) (Ф.И.О.)

Подписи лиц, ответственных за разработку электронного документа

|  |  |  |
| --- | --- | --- |
| Состав электронного документа | Разработчик | Руководитель |
| Пояснительная записка (на бумажном носителе формата А4), ПЗ\_Стадуб\_т–318.docх |  |  |
| ГЧ, Диаграммы.docx |  |  |
| Папка с проектом «kursach» |  |  |
| Файл базы данных «WestMotorsAppDb» |  |  |
| Тип носителя: USB – флешка |  |  |

**Этикетка**

для курсового проекта

**Курсовой проект**

Тема «Разработка веб-приложения для автоматизации работы автосалона «WEST MOTORS»

КП Т.318021.401

Разработан 22.06.2025

Разработчик: Стадуб П.В.

Руководитель: Кривошеина А.В.

Технические средства: Персональный компьютер с процессором Intel Core i7, 16 ГБ оперативной памяти, жёсткий диск объёмом 1 ТБ, монитор с разрешением 1920×1080, клавиатура, мышь.

Программные средства: Операционная система Windows 11, среда разработки Microsoft Visual Studio 2022, платформа .NET 8 с использованием ASP.NET Core MVC, система управления базами данных Microsoft SQL Server, браузер Firefox

**Состав документа:**

Пояснительная записка – ПЗ\_Стадуб\_т-318.docх

Графическая часть –Диаграммы.docx

Папка с проектом – kursach

Файл базы данных – WestMotorsAppDb