|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное АВТОНОМНОЕ образовательное учреждение высшего образования  «Национальный исследовательский ядерный университет «МИФИ» |
| **Димитровградский инженерно-технологический институт –**  филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский ядерный университет «МИФИ»  **(ДИТИ НИЯУ МИФИ)** |

Факультет информационно-технологический и социально-экономических наук

Кафедра Информационных технологий

Направление подготовки 09.03.01 Информатика и вычислительная техника

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ**

Разработка модуля авторизации для серверной инфраструктуры маркетплейса автопроизводителя

Студент

Группы ВТ-41 Д.Д. Кузин

(подпись)

Руководитель

ассистент М.С. Сутягина

(подпись)

Заведующий кафедрой

доцент, к.п.н. О.А. Ракова

(подпись)

Димитровград, 2025

|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное АВТОНОМНОЕ образовательное учреждение высшего образования  «Национальный исследовательский ядерный университет «МИФИ» |
| **Димитровградский инженерно-технологический институт –**  филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский ядерный университет «МИФИ»  **(ДИТИ НИЯУ МИФИ)** |

**ФАКУЛЬТЕТ** Информационно-технологический и социально-экономических наук

**КАФЕДРА** Информационных технологий

**НАПРАВЛЕНИЕ ПОДГОТОВКИ** 09.03.01 **ГРУППА** ВТ-41

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

1. Фамилия, имя, отчество Кузин Данила Денисович

2. Тема «Разработка модуля авторизации для серверной инфраструктуры маркетплейса автопроизводителя»

3. Срок сдачи студентом готовой работы 18 апреля 2025 г.

4. Руководитель курсовой работы Сутягина Мария Сергеевна

Дата выдачи задания «\_3\_»\_марта\_\_\_2025 г.

Руководитель курсовой работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Задание принял к исполнению 3 марта 2025 г. .

(дата и подпись студента)

1. Исходные данные к работе:

*требования к уровню разработки программного продукта:*

*а) разработка базы данных на языке PostgreSQL;*

*а) современный пользовательский интерфейс;*

*б) язык программирования C#.*

1. Содержание работы:
   1. *теоретическая часть:*

* исследование предметной области;
* описание логики программы;
* проектирование элементов интерфейса.
  1. *практическая часть:*
* *реализация интерфейса;*
* *реализация логического компонента программного продукта.*

1. Основная литература по теме работы:
2. *Стасышин, В.М. Разработка информационных систем и баз данных: учебное пособие для СПО [Текст] // В.М. Стасышин. – Саратов: Профобразование, 2020. – 100 с.*
3. *Умрихин Е. Д. Разработка веб-приложений с помощью ASP.Net Core MVC [Текст] / Е.Д. Умрихин – Санкт-Петербург: БХВ-Петербург, 2023. – 416 с.*
4. Отчётный материал работы:
5. *пояснительная записка;*
6. *CD/DVD-диск с пояcнительной запиской и файлами проекта приложения.*

РЕФЕРАТ

Кузин Д.Д. «РАЗРАБОТКА МОДУЛЯ АВТОРИЗАЦИИ ДЛЯ СЕРВЕРНОЙ ИНФРАСТРУКТУРЫ МАРКЕТПЛЕЙСА АВТОПРОИЗВОДИТЕЛЯ»: КУРСОВОЙ ПРОЕКТ / ДИТИ НИЯУ МИФИ № КП.09.03.01.74.2025 – Димитровград, 2025. – стр. 27, рис. 7, библ. назв. 12.

Ключевые слова: База данных, АВТОРИЗАЦИЯ, АУТЕНТИФИКАЦИЯ, МАРКЕТПЛЕЙС, POSTGRESQL.

Объект исследования – процесс авторизации в маркетплейсе автопроизводителя

Цель работы – разработать модуль авторизации для серверной инфраструктуры маркетплейса автопроизводителя

Выводы: в ходе выполнения курсового проекта был разработан модуль авторизации для серверной части маркетплейса автопроизводителя. На основе теоретического исследования была выбрана технология авторизации через cookie в ASP.NET Core как наиболее безопасная и подходящая для данной архитектуры. Реализованы регистрация, вход и выход пользователей с поддержкой ролей, хешированием паролей и использованием базы данных PostgreSQL. Разработка велась на языке C# в среде Visual Studio с использованием подхода Code First и Entity Framework Core.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc195457775)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 7](#_Toc195457776)

[1.1 Основные понятия и принципы аутентификации и авторизации 7](#_Toc195457777)

[1.2 Методы и протоколы аутентификации 9](#_Toc195457778)

[1.3 Сценарий пользовательского взаимодействия с модулем авторизации 10](#_Toc195457779)

[1.4 Исследование и обоснование выбора технологий и инструментов разработки 11](#_Toc195457780)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 13](#_Toc195457781)

[2.1 Используемые технологии и инструменты разработки 13](#_Toc195457782)

[2.2 Проектирование и реализация структуры базы данных 14](#_Toc195457783)

[2.3 Разработка серверной логики модуля авторизации 18](#_Toc195457784)

[ЗАКЛЮЧЕНИЕ 24](#_Toc195457785)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 25](#_Toc195457786)

ВВЕДЕНИЕ

В современном цифровом мире информационные технологии играют ключевую роль в обеспечении безопасности и эффективного функционирования бизнес-процессов. Для крупных компаний, таких как автопроизводители, надежная серверная инфраструктура является основой стабильной работы всех сервисов, включая маркетплейсы, которые обеспечивают взаимодействие с клиентами, дилерами и партнерами. Одним из критически важных компонентов такой инфраструктуры является система авторизации пользователей, позволяющая управлять доступом к различным сервисам и защищать конфиденциальные данные.

В курсовом проекте будет рассмотрен процесс разработки модуля авторизации пользователей для серверной инфраструктуры маркетплейса автопроизводителя. Будут проанализированы основные требования к системе авторизации, изучены современные методы аутентификации и защиты данных, а также реализованы механизмы управления правами пользователей.

Актуальность темы: разработка безопасного и масштабируемого модуля авторизации позволит повысить уровень защиты пользовательских данных и предотвратить несанкционированный доступ к информационным ресурсам маркетплейса. Это особенно важно в условиях роста киберугроз и увеличения объемов персональных данных, обрабатываемых в цифровых экосистемах.

Цель проекта: создание надежной и гибкой системы авторизации, обеспечивающей безопасный доступ пользователей к ресурсам маркетплейса, а также управление их ролями и правами.

Задачи проекта:

* анализ предметной области и изучение существующих решений в области авторизации пользователей;
* разработка требований к модулю авторизации с учетом специфики маркетплейса автопроизводителя;
* выбор технологий и инструментов для реализации системы аутентификации и управления доступом;
* проектирование архитектуры модуля, разработка логической и физической моделей данных;
* реализация механизма аутентификации пользователей с поддержкой многофакторной авторизации;
* обеспечение безопасности хранения и передачи данных, внедрение современных криптографических методов;
* тестирование и оптимизация модуля, интеграция с существующей серверной инфраструктурой маркетплейса.

# 1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## Основные понятия и принципы аутентификации и авторизации

Аутентификация — это процедура проверки подлинности пользователя, то есть подтверждение, что он действительно является тем, за кого себя выдает. Чаще всего это осуществляется с помощью пароля, пин-кода, биометрических данных или других методов [1].

Авторизация — это процесс предоставления пользователю или группе пользователей определенных разрешений, прав доступа и привилегий в компьютерной системе [2].

В сфере информационной безопасности аутентификация и авторизация являются ключевыми процессами, обеспечивающими контроль доступа к системам и данным. Существует несколько видов аутентификации и авторизации, применяемых в зависимости от требований безопасности и специфики системы. Далее рассмотрены некоторые виды аутентификации:

* по количеству факторов два основных вида аутентификации: Однофакторная аутентификация: использует один метод проверки, например, ввод пароля;
* многофакторная аутентификация (MFA): предполагает использование нескольких независимых факторов для подтверждения личности, таких как знание: то, что знает пользователь (пароль, PIN-код);владение: то, что есть у пользователя (смартфон, смарт-карта);присутствие: биометрические данные пользователя (отпечаток пальца, распознавание лица) [3].
  + - 1. По методам реализации:
* парольная аутентификация является самым распространенным методом, при котором пользователь вводит статический или одноразовый пароль [4];
* биометрическая аутентификация: использует уникальные физиологические характеристики пользователя, такие как отпечатки пальцев или голос;
* аутентификация с использованием токенов: предполагает применение физических или программных устройств, генерирующих одноразовые коды доступа;
* аутентификация по сертификатам: основана на использовании цифровых сертификатов для подтверждения личности пользователя [5].

Виды авторизации:

* + - 1. По моделям управления доступом:
* ролевая (RBAC): пользователям назначаются определенные роли, каждая из которых имеет набор разрешений;
* мандатная (MAC): доступ к ресурсам определяется на основе установленных меток безопасности и правил, заданных администратором;
* дискреционная (DAC): владелец ресурса самостоятельно определяет, кто и какие действия может выполнять с его ресурсами.
  + - 1. По способу предоставления доступа:
* онлайн-авторизация: требует постоянного соединения с сервером для проверки прав доступа;
* офлайн-авторизация: позволяет пользователю получать доступ к ресурсам без постоянного подключения к сети, основываясь на ранее полученных данных.

1. По количеству проверок существует одноступенчатая авторизация.

Предполагает единовременную проверку прав доступа; многоступенчатая авторизация: требует прохождения нескольких этапов проверки перед предоставлением доступа [6].

## Методы и протоколы аутентификации

Аутентификация — это процесс проверки подлинности пользователя, системы или устройства, направленный на подтверждение их идентичности перед предоставлением доступа к ресурсам. Существует множество методов и протоколов аутентификации, каждый из которых имеет свои особенности и области применения.

Существуют следующие методы аутентификации:

* + - парольная аутентификация. Пользователь предоставляет секретный код (пароль), известный только ему и системе. Несмотря на свою распространенность, этот метод подвержен рискам, таким как подбор пароля и фишинг;
    - аутентификация с использованием одноразовых паролей (OTP). В этой системе для каждого входа генерируется уникальный пароль, который действует ограниченное время. Алгоритмы, такие как HOTP (HMAC-Based One-Time Password), используют счетчик событий для генерации паролей [7];
    - многофакторная аутентификация (MFA). Сочетание двух или более факторов: что-то, что знает пользователь (пароль), что-то, что он имеет (смарт-карта или мобильное устройство), и что-то, что он есть (биометрические данные). Это повышает уровень безопасности системы;
* биометрическая аутентификация. Данный вид аутентификации использует уникальные физиологические или поведенческие характеристики пользователя, такие как отпечатки пальцев, радужка глаза, лицо или голос, для подтверждения его личности.

Популярными протоколами авторизации являются следующие:

* basic authentication: простой протокол, при котором имя пользователя и пароль передаются в заголовке HTTP-запроса в кодировке Base64;
* digest authentication: предлагает более высокий уровень безопасности по сравнению с basic authentication. Сервер генерирует уникальное значение (nonce), которое используется клиентом для создания хэшированного пароля, защищая учетные данные от перехвата. Однако digest authentication также имеет свои уязвимости, такие как возможность атак «человек посередине»;
* kerberos: протокол аутентификации, использующий билеты для подтверждения личности. Он работает по принципу «единого входа» (SSO), позволяя пользователю аутентифицироваться один раз и получать доступ к различным сервисам без повторного ввода учетных данных. Kerberos широко используется в корпоративных сетях;
* oAuth 2.0: протокол авторизации, который позволяет сторонним приложениям получать ограниченный доступ к ресурсам пользователя без необходимости передачи его учетных данных. OAuth 2.0 часто используется для интеграции с социальными сетями и другими сервисами;
* openID connect: надстройка над OAuth 2.0, добавляющая слой аутентификации. Она позволяет клиентским приложениям подтверждать личность пользователя на основе аутентификации, выполненной авторизационным сервером, и получать базовую информацию о профиле пользователя;
* SAML (Security Assertion Markup Language): стандартный протокол для обмена данными об аутентификации и авторизации между различными доменами безопасности. Часто используется в корпоративных приложениях для реализации единого входа [8].

## Сценарий пользовательского взаимодействия с модулем авторизации

Сценарий пользовательского взаимодействия с модулем авторизации описывает последовательность действий, которые пользователь выполняет для доступа к защищённым ресурсам системы. Эти сценарии помогают понять и спроектировать удобный и безопасный процесс аутентификации и авторизации. Далее рассмотрены основные этапы и лучшие практики разработки таких сценариев:

* + - 1. Регистрация пользователя: инициация: пользователь посещает страницу регистрации на платформе. Следующим этапов происходит вводданных: пользователь предоставляет необходимую информацию, такую как имя, адрес электронной почты и пароль. Последний этап – подтверждение: система отправляет письмо с подтверждением на указанный адрес электронной почты. Пользователь должен перейти по ссылке в письме для активации учётной записи.
      2. Авторизация пользователя происходит по следующему принципу: инициация, в рамках которой пользователь посещает страницу входа в систему. Далее происходит вводучётныхданных, а именно пользователь вводит свой логин (или адрес электронной почты) и пароль. Последним этапом является аутентификация, где система проверяет введённые данные. В случае успеха: пользователь получает доступ к личному кабинету или защищённым разделам сайта. В случае неудачи – отображается сообщение об ошибке с предложением повторить попытку.
      3. Выход из системы (Logout): инициация: пользователь нажимает кнопку "Выход" в личном кабинете. Завершениесеанса: система завершает текущий сеанс и перенаправляет пользователя на страницу входа или главную страницу сайта [9].

## Исследование и обоснование выбора технологий и инструментов разработки

При разработке системы авторизации для веб-приложений важно тщательно подойти к выбору технологий и инструментов, чтобы обеспечить безопасность, удобство и масштабируемость системы. Существует несколько подходов и технологий для реализации аутентификации и авторизации:

1. Cookie-based аутентификация использует файлы cookie для хранения информации о сессии пользователя, позволяя серверу идентифицировать пользователя при последующих запросах. Основные преимущества – удобство пользователь не требует повторного ввода учетных данных при каждом посещении сайта. Производительность так же является большим преимуществом, так как снижается нагрузка на сервер за счет хранения информации о сессии на стороне клиента.
2. OAuth 2.0 — это открытый стандарт для делегирования доступа, позволяющий сторонним приложениям получать ограниченный доступ к ресурсам пользователя без передачи его пароля. Вместо этого OAuth предоставляет приложению токен доступа после того, как пользователь аутентифицируется через поставщика идентификации. Преимуществами можно назвать тот факт, что пользователь может авторизоваться в приложении, используя уже существующие учетные данные от других сервисов. Сторонние приложения получают доступ к ресурсам без необходимости хранения пароля пользователя. Недостатками являются сложность настройки, так как требуется правильная интеграция с поставщиками идентификации. Зависимость от сторонних сервисов так же является одной из проблем, ведь в случае компрометации поставщика идентификации может возникнуть дополнительный риск.
3. JSON Web Tokens (JWT) — это компактные, безопасные для URL токены, которые содержат утверждения о пользователе и часто используются для управления сессиями в современных веб-приложениях. JWT может хранить информацию о пользователе, такую как идентификатор, роль и привилегии, и подписан для обеспечения целостности данных. Преимуществамиявляютсябессессионнаяаутентификация: сервер не хранит информацию о сессии, что снижает нагрузку на сервер. Легко интегрируется с одностраничными приложениями (SPA) и мобильными приложениями. Срединедостатковможновыделить безопасность: кража токена может привести к несанкционированному доступу, если токен не защищен должным образом. Такжетокены могут становиться слишком большими, если в них хранится много данных.

Таким образом, на основе проведённого исследования и анализа различных методов авторизации, можно сделать вывод, что для текущего проекта наибольшую эффективность и безопасность обеспечит использование cookie-based авторизации. Этот подход сочетает в себе простоту реализации, удобство для пользователя и высокую производительность, что делает его оптимальным выбором для данного случая.

# 2 ПРАКТИЧЕСКАЯ ЧАСТЬ

* 1. Используемые технологии и инструменты разработки

В рамках проекта по разработке модуля авторизации для серверной инфраструктуры маркетплейса автопроизводителя применены современные технологии и инструменты, обеспечивающие безопасность, масштабируемость и удобство поддержки.

Язык программирования C# – основной язык проекта, используемый для написания всей бизнес-логики, моделей, сервисов и контроллеров.

Среди преимуществ: статическая типизация и высокая безопасность типов, богатая стандартная библиотека, отлично интегрируется с ASP.NET Core, поддерживает объектно-ориентированное, асинхронное и функциональное программирование, хорошо документирован и активно развивается.

Фреймворк ASP.NET Core – базовый фреймворк для построения веб-приложения и REST API.

Плюсы фреймворка ASP.NET Core: высокая производительность, кроссплатформенность, поддержка Middleware и Dependency Injection, простота интеграции с различными форматами аутентификации (в т.ч. Cookie, JWT, OAuth).

Фреймворк Entity Framework Core – ORM (Object-Relational Mapping) для взаимодействия с базой данных.

Среди преимуществ фреймворка Entity Framework Core: поддержка миграций; поддержка подходов code First / DB First; простота работы с таблицами через модели; автоматическое связывание сущностей через навигационные свойства.

Среда разработки: Visual Studio – Интегрированная среда разработки (IDE), используемая для написания, отладки и тестирования проекта.

Преимущества данной среды разработки: поддержка ASP.NET Core «из коробки»; удобные инструменты для отладки и профилирования; поддержка NuGet — системы управления пакетами; интеграция с Git; визуальное создание миграций и работа с EF Core; возможность разворачивания проекта прямо из IDE.

* 1. Проектирование и реализация структуры базы данных

При разработке систем авторизации с использованием Entity Framework (EF) в ASP.NET Core существуют два основных подхода к проектированию и взаимодействию с базой данных: Database First и Code First.

Database First предполагает, что разработка начинается с уже существующей базы данных. Схема базы данных создается заранее, и затем на ее основе генерируются классы моделей и контекст данных в приложении.

Преимуществами данного подхода можно назвать интеграцию с существующими базами данных. Если база данных уже разработана и используется, этот подход позволяет быстро создать соответствующие модели в приложении. Визуальное проектирование: Возможность использовать инструменты для визуального проектирования схемы базы данных, что облегчает понимание структуры данных.

Среди недостатков подхода Database First: ограниченная гибкость – изменения в модели данных требуют внесения изменений непосредственно в базу данных, после чего необходимо повторно генерировать или обновлять модели в коде; зависимость от базы данных: бизнес-логика может быть тесно связана с конкретной схемой базы данных, что затрудняет переносимость и тестирование [10].

Code First ориентирован на первичное создание моделей данных в коде. Разработчик определяет классы сущностей и их конфигурацию, после чего EF генерирует соответствующую схему базы данных.

Преимущества данного подхода: гибкость в разработке: модели данных определяются в коде, что упрощает их изменение и поддержку; миграции: EF предоставляет механизм миграций, позволяющий автоматически обновлять схему базы данных в соответствии с изменениями моделей, что облегчает управление версиями базы данных; независимость от конкретной СУБД: Модели могут быть легко адаптированы для работы с различными базами данных.

Недостатки подхода Code First: потенциальная сложность начальной настройки: требуется тщательная настройка моделей и понимание механизмов EF для корректной генерации схемы базы данных; возможные ограничения в сложных сценариях: для очень сложных схем базы данных может потребоваться дополнительная конфигурация или отказ от некоторых автоматизированных возможностей EF [11].

На основании проведенного исследования был выбран подход Code First. При работе с удалёнными базами данных PostgreSQL важно выбрать подходящий инструмент для администрирования и управления. Существует несколько популярных решений, каждое из которых обладает своими особенностями и преимуществами.

DBeaver — это универсальный графический клиент для работы с различными СУБД, включая PostgreSQL.

Преимущества DBeaver:

* кроссплатформенность: Работает на Windows, macOS и Linux;
* поддержка множества СУБД: Совместим более чем с 80 различными базами данных;
* визуальный интерфейс: Удобный графический интерфейс с поддержкой визуального построителя запросов, что облегчает работу с данными;
* импорт и экспорт данных: Поддерживает различные форматы, включая CSV, HTML, XML, JSON, XLS, XLSX;
* расширяемость: Возможность установки дополнительных плагинов для расширения функциональности.

pgAdmin — это бесплатный и открытый графический пользовательский интерфейс для администрирования PostgreSQL.

Преимущества pgAdmin:

* специализация на PostgreSQL: Разработан специально для работы с PostgreSQL, что обеспечивает глубокую интеграцию с особенностями этой СУБД;
* веб-интерфейс: Позволяет управлять базой данных через браузер, что удобно для удалённого администрирования;
* мощные инструменты разработки и отладки: Включает в себя редактор запросов, инструменты для мониторинга производительности и управления пользователями.

DataGrip — это коммерческий инструмент от компании JetBrains для работы с базами данных.

Преимущества DataGrip:

* интеллектуальный редактор SQL: Поддерживает автодополнение, рефакторинг и анализ кода;
* поддержка множества СУБД: Помимо PostgreSQL, совместим с MySQL, Oracle, SQL Server и другими;
* интеграция с системой контроля версий: Упрощает совместную работу над проектами [12].

На основании проведенного исследования, в качестве СУБД для удаленного доступа к базе данных выбрана DBeaver. Структура базы данных:

* + - 1. Таблица «Users» (Пользователи):
* Id (long) — уникальный идентификатор пользователя;
* Username (string) — логин пользователя (необходим для аутентификации);
* PasswordHash (string) — хэш пароля (для безопасности);
* Role (Role) — связь с ролью пользователя. Один пользователь может иметь одну роль;
* FirstName (string) — имя пользователя;
* LastName (string) — фамилия пользователя;
* Avatar (string) — путь к изображению аватара (если используется);
* Phone (string) — номер телефона (валидируется с помощью атрибута [Phone]);
* Email (string) — email пользователя (валидируется с помощью атрибута [EmailAddress]);
* Address (string) — домашний адрес пользователя;
* Created (DateTime) — дата и время регистрации пользователя;
* Updated (DateTime) — дата последнего обновления информации о пользователе;
* IsActive (bool) — флаг активности пользователя (по умолчанию активен).

1. Таблица «Roles» (Роли):

* Id (long) — уникальный идентификатор роли;
* Name (string) — имя роли (например, "Admin", "User", "Manager");
* IsActive (bool) — флаг активности роли (по умолчанию активна).

Для отправки моделей в базу данных с помощью Entity Framework Core в ASP.NET Core, создается контекст базы данных, который будет содержать DbSet для каждой сущности, пример приведен ниже.

public class ApplicationDbContext : DbContext

{

public DbSet<User> Users { get; set; }

public DbSet<Role> Roles { get; set; }

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options): base(options) {}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{ base.OnModelCreating(modelBuilder);

}}

В файле «appsettings.json» необходимо указать строку подключения к базе данных.

{"ConnectionStrings": {

"DefaultConnection": "Host=localhost;Port=5432;Database=mydb;Username=myuser;Password=mypassword"

}

}

Затем в файле «Program.cs» зарегистрировать контекст базы данных.

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ApplicationDbContext>(options =>

options.UseNpgsql(Configuration.GetConnectionString("DefaultConnection")));

}

Для создания миграции используется команда «dotnet ef migrations add (название миграции)», а для того, чтобы применить миграции команда «dotnet ef database update».

На рисунке 2.1 изображена вся структура таблиц в базе данных, что доказывает успешность отправленных моделей.

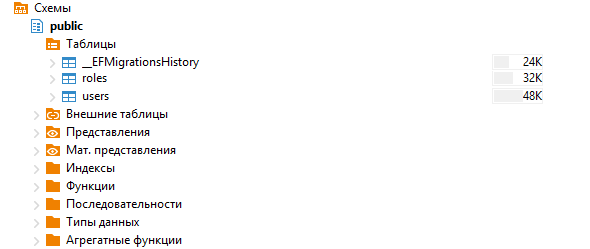


Рисунок 2.1 – Список таблиц в базе данных

* 1. Разработка серверной логики модуля авторизации

В начале разработки определяется контракт для работы с пользовательскими данными в контексте аутентификации. Данным контрактом является «IAuthRepository».

public interface IAuthRepository

{

Task<User> GetByUsernameAsync(string username);

Task<User> CreateAsync(User user);

}

В примере ниже приведена реализация интерфейса «IAuthRepository» в классе «AuthRepository». В рамках класса «AuthRepository» реализовано два метода: «GetByUsernameAsync», в который передается переменная строкового типа и если в базе данных есть запись с таким Username, то возвращается данная запись. Второй же метод «CreateAsync» принимает модель пользователя, которая подготовлена к созданию и происходит создание и сохранение данного пользователя в базу данных.

public class AuthRepository: IAuthRepository

{

private readonly AppDbContext \_context;

public AuthRepository(AppDbContext context) {\_context = context;}

public async Task<User> GetByUsernameAsync(string username)

{

return await \_context.Users.Include(u => u.Role).FirstOrDefaultAsync(a => a.Username == username);

}

public async Task<User> CreateAsync(User auth)

{

\_context.Users.Add(auth);

await \_context.SaveChangesAsync();

return auth;

}

}

Класс «AuthService» реализует бизнес-логику аутентификации и регистрации пользователей в системе. Он работает через абстракции «IAuthRepository» и «IRoleRepository», то есть не взаимодействует напрямую с базой данных, а использует репозитории, что соответствует принципам чистой архитектуры. Метод «AuthenticateAsync» получает на входе логин и пароля для их последующей проверки. Метод «RegisterAsync» создаёт пользователя в системе с хешированным паролем и назначенной ролью. Метод «HashPassword» хэширует пароль, что позволяет защитить хранимые пароли от компрометации.

public class AuthService

{

private readonly IAuthRepository \_authRepository;

private readonly IRoleRepository \_roleRepository;

public AuthService(IAuthRepository authRepository, IRoleRepository roleRepository){

\_authRepository = authRepository;

\_roleRepository = roleRepository;

}

public async Task<User> AuthenticateAsync(string username, string password)

{

var auth = await \_authRepository.GetByUsernameAsync(username);

if (auth == null || auth.PasswordHash != HashPassword(password)){ return null; }

return auth;

}

public async Task<User> RegisterAsync(string firstname, string lastname, string username, string password, string email)

{

var existingUser = await \_authRepository.GetByUsernameAsync(username);

if (existingUser != null)

{

throw new Exception("Пользователь с таким именем уже существует");

}

var auth = new User

{

FirstName = firstname,

LastName = lastname,

Username = username,

PasswordHash = HashPassword(password),

Email = email,

Role = await \_roleRepository.GetByNameAsync("USER"),

Created = DateTime.UtcNow,

IsActive = true,

};

return await \_authRepository.CreateAsync(auth);

}

private string HashPassword(string password)

{

using var sha256 = SHA256.Create();

return Convert.ToBase64String(sha256.ComputeHash(Encoding.UTF8.GetBytes(password)));

}

}

ApiAuthController – API-контроллер для авторизации пользователей. Он предоставляет три HTTP-метода: регистрацию, вход и выход. Работает через сервис AuthService, который реализует бизнес-логику. Метод «Register» принимает JSON-объект с данными из «RegisterViewModel» (имя, фамилия, логин, пароль, email) и передает их сервису AuthService для последующей регистрации пользователя.

public async Task<IActionResult> Register([FromBody] RegisterViewModel request){

try{

var user = await \_authService.RegisterAsync(request.Firstname, request.Lastname, request.Username, request.Password, request.Email);

return Ok(new { user.Id, user.Username, user.Email });

}

catch (Exception ex)

{

return BadRequest(new { message = ex.Message });

}

}

Метод «Login» принимает JSON-объект с Username и Password, проверяет учетные данные через «AuthenticateAsync» и, если пользователь найден, создаёт набор клеймов (Claims) и сохраняет их в cookie через «HttpContext.SignInAsync»:

public async Task<IActionResult> Login([FromBody] LoginViewModel request)

{

var user = await \_authService.AuthenticateAsync(request.Username, request.Password);

if (user == null)

return Unauthorized(new { message = "Неверные учетные данные" });

var claims = new List<Claim>

{

new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),

new Claim(ClaimTypes.Name, user.Username),

new Claim(ClaimTypes.Email, user.Email),

new Claim(ClaimTypes.Role, user.Role.Name) // Добавляем роль в куки

};

var claimsIdentity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);

var claimsPrincipal = new ClaimsPrincipal(claimsIdentity);

await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, claimsPrincipal);

return Ok(new { user.Id, user.Username, user.Email });

}

Метод «Logout» удаляет аутентификационный cookie, тем самым разлогинивает пользователя.

public async Task<IActionResult> Logout()

{

await Task.Run(() => Response.Cookies.Delete(".AspNetCore.Cookies")); // Асинхронно удаляем куку

return Ok(new { message = "Вы успешно вышли из системы" });

}

При запуске сайта отображается титульная форма (рисунок 2.2), на которой есть кнопки для входа и регистрации.

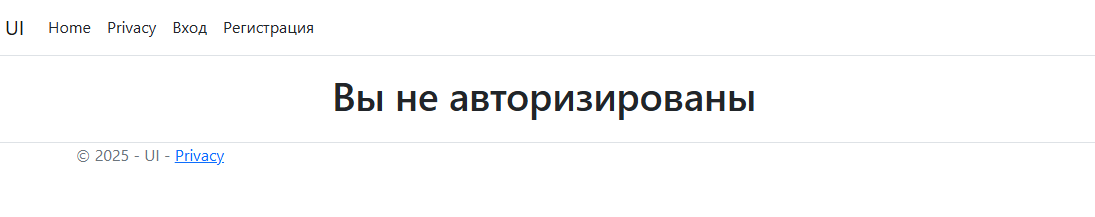


Рисунок 2.2 – Титульная форма

На рисунках 2.3-2.4 приведен пример регистрации пользователя и ответ программы на успешную регистрацию.

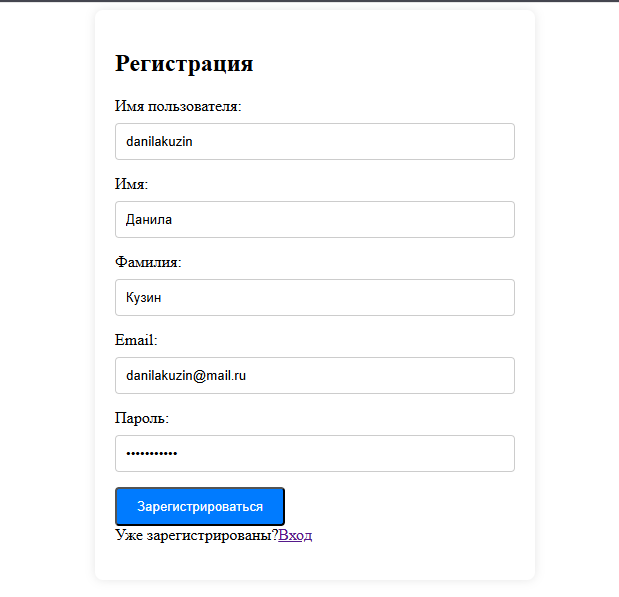


Рисунок 2.3 – Форма регистрации пользователя

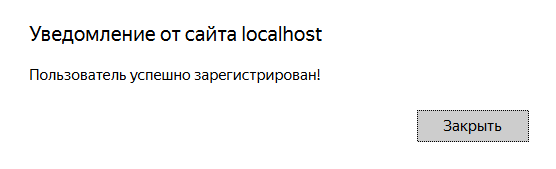


Рисунок 2.4 – Уведомление об успешной регистрации пользователя

На рисунке 2.5 продемонстрирован список пользователей после регистрации, где можно убедиться в наличии созданного пользователя, что об успешности работы методов, связанных с регистрацией пользователя

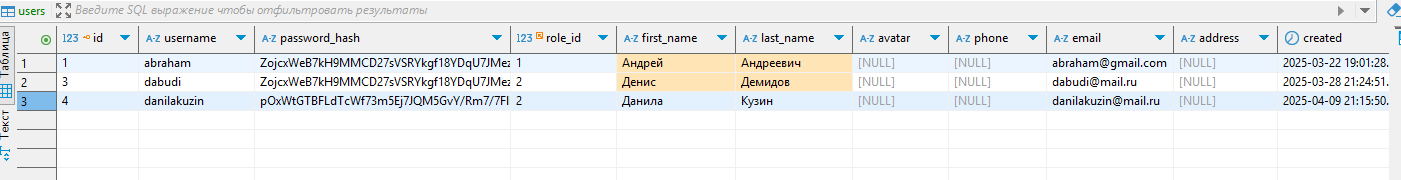


Рисунок 2.5 – Список пользователей в базе данных

По нажатию на кнопку входа отображается форма для ввода логина и пароля, показанная на рисунке 2.6.

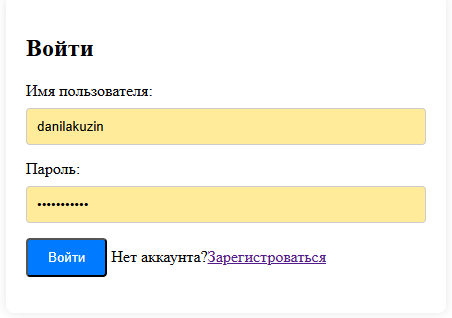


Рисунок 2.6 – Форма авторизации пользователя

После успешного введения данных происходит отображение титульной формы, на которой будет отображаться приветствие пользователя. Результат продемонстрирован ниже на рисунке 2.7.

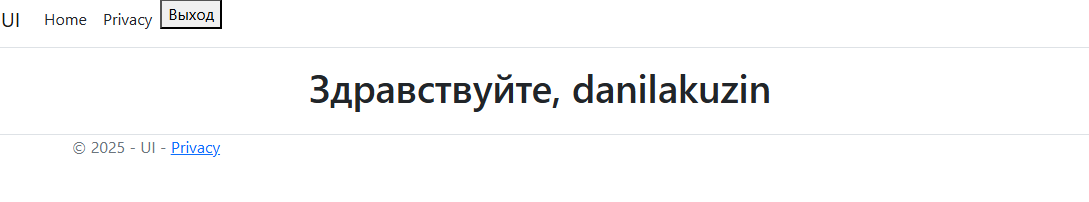


Рисунок 2.7 – Титульная форма для авторизованного пользователя

# ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта была успешно решена задача разработки модуля авторизации, предназначенного для использования в серверной части маркетплейса автопроизводителя. Основной целью модуля стало обеспечение надежной идентификации пользователей системы с возможностью разграничения доступа на основе ролей и прав.

На теоретическом этапе работы были изучены ключевые понятия, касающиеся аутентификации и авторизации, а также проведен обзор существующих методов хранения и проверки учетных данных. Были рассмотрены различные технологии передачи данных авторизации, включая cookie, JWT и OAuth, а также подходы к проектированию баз данных, такие как DB First и Code First. В результате анализа было принято решение реализовать авторизацию с использованием технологии cookie на базе фреймворка «ASP.NET Core». Этот подход был выбран по причине его удобной интеграции в используемую платформу, высокой надежности, встроенной поддержки механизма Claims и общей технологической целесообразности для данного проекта.

В результате работы был создан полностью функционирующий, расширяемый и безопасный модуль авторизации, обеспечивающий стабильную работу, соответствует современным требованиям и готов к внедрению в серверную инфраструктуру маркетплейса.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Что такое аутентификация [Электронный ресурс] // Решения для бизнеса: технологии и сервисы для компании от экосистемы СберБанка // Режим доступа: <https://developers.sber.ru/help/sber-id/what-is-authentication> (дата обращения 17.03.2025)
2. Авторизация [Электронный ресурс] // Энциклопедия «Касперского» // Режим доступа: <https://encyclopedia.kaspersky.ru/glossary/authorization/> (дата обращения 17.03.2025)
3. Аутентификация – что это такое и зачем она нужна [Электронный ресурс] // Солар // Режим доступа: <https://rt-solar.ru/products/solar_inrights/blog/4518/> (дата обращения 18.03.2025)
4. Обзор способов и протоколов аутентификации в веб-приложениях [Электронный ресурс] // Хабр // Режим доступа: https://habr.com/ru/companies/dataart/articles/262817/ (дата обращения 19.03.2025)
5. Аутентификация в Интернете [Электронный ресурс] // Википедия // Режим доступа: <https://ru.wikipedia.org/wiki/Аутентификация_в_Интернете> (дата обращения 19.03.2025)
6. Авторизация. Определение и методы [Электронный ресурс] // eKassir // Режим доступа: <https://ekassir.com/blog/avtorizacziya-opredelenie-i-metody/> (дата обращения 20.03.2025)
7. HOTP [Электронный ресурс] // Википедия // Режим доступа: <https://ru.wikipedia.org/wiki/HOTP> (дата обращения 21.03.2025)
8. Протоколы аутентификации: как они работают [Электронный ресурс] // Skypro Wiki // Режим доступа: <https://sky.pro/wiki/sql/protokoly-autentifikacii-kak-oni-rabotayut/> (дата обращения 28.03.2025)
9. Пользовательский модуль для двухфакторной аутентификации [Электронный ресурс] // ELMA365 Справка // Режим доступа: <https://elma365.com/ru/help/platform/two-factor-integration.html> (дата обращения 29.03.2025)
10. «Истина в последней инстанции» или зачем нужен Database First Design [Электронный ресурс] // Хабр // Режим доступа: <https://habr.com/ru/articles/413597/> (дата обращения 29.03.2025)
11. Разработка баз данных с Code First [Электронный ресурс] // Хабр // Режим доступа: <https://habr.com/ru/articles/234827/> (дата обращения 30.03.2025)
12. 8 лучших GUI клиентов PostgreSQL в 2021 году [Электронный ресурс] // Библиотека программиста // Режим доступа: <https://proglib.io/p/8-luchshih-gui-klientov-postgresql-v-2021-godu-2021-09-20> (дата обращения 05.03.2025)