

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ИНФОРМАЦИОННАЯ КЛИЕНТ-СЕРВЕРНАЯ
СИСТЕМА ОРГАНИЗАЦИИ И ПРОВЕДЕНИЯ
ЛЮБИТЕЛЬСКИХ АВТОТРАНСПОРТНЫХ
МЕРОПРИЯТИЙ

Студент

44 28.05.2026 Беляев Даниил Ильич
(подпись, дата)

Руководитель

ф.и.о. 28.05.2026 Токарев Андрей Николаевич
(подпись, дата)

Нормоконтролёр

ф.и.о. 28.05.2026 Бычков Андрей Станиславович
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой

ф.и.о.
(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

15 июня 2026 г. № 32)

Секретарь ГЭК

ф.и.о.
(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2026

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«07» 04 2026 г.

ЗАДАНИЕ НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Беляев Даниил Ильич гр. 22ВВП1 факультета вычислительной техники
2. Тема работы: Информационная клиент-серверная система организации и проведения любительских автотранспортных мероприятий

Тема утверждена приказом ПГУ № 347/о от "07" апреля 2026 г.

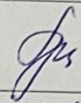
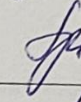
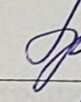
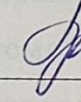
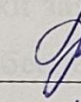
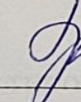
3. Руководитель бакалаврской работы к.т.н., доцент каф. ВТ ПГУ Токарев А.Н.
4. Задание на работу (назначение разработки, исходные данные и т.п.)

В рамках выпускной квалификационной работы необходимо разработать клиент-серверное веб-приложение для организации и проведения любительских автоспортивных мероприятий. Система должна обеспечивать регистрацию и аутентификацию пользователей, управление турнирами, трассами и автомобилями участников, хранение результатов соревнований и взаимодействие пользователей в рамках единой информационной среды. Основными требованиями к системе являются наличие пользовательского интерфейса, реализация прав доступа, обеспечение безопасности работы с данными, стабильная работа клиентской и серверной частей приложения.

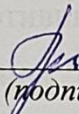
5. Перечень подлежащих разработке вопросов

Требуется провести анализ предметной области любительского автоспорта и существующих информационных систем для организации автоспортивных мероприятий, выявить их преимущества и недостатки. Необходимо выбрать и обосновать используемые технологии и инструменты разработки, спроектировать архитектуру системы, разработать серверную и клиентскую части приложения, реализовать модули управления пользователями, трассами, турнирами и результатами соревнований, а также механизмы аутентификации и авторизации. Дополнительно требуется разработать структуру базы данных и выполнить тестирование разработанной системы.

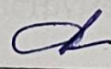
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	07.04.2026- 12.04.2026	
2.	Анализ требований	10	12.04.2026- 13.04.2026	
3.	Выбор технологий, применяемых при проектировании и разработке	50	13.04.2026- 20.04.2026	
4.	Описание алгоритмов	40	20.04.2026- 23.04.2026	
5.	Разработка системы автоспортивных мероприятий	150	23.04.2026- 08.05.2026	
6.	Оформление пояснительной записки	24	08.05.2026- 21.05.2026	

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы  (подпись, дата) Токарев А. Н.

Задание к исполнению принял студент  (подпись, дата) Беляев Д. И.

Работу к защите допустить
Декан факультета  10.06.2026 (подпись, дата) Фионова Л.Р.

Реферат

Работа содержит 127 листов, 41 рисунок, 15 таблиц, 10 источников, 3 приложения.

C#, ASP.NET CORE, TYPESCRIPT, REACT, MICROSOFT SQL SERVER, ENTITY FRAMEWORK CORE, DOCKER, WEB-ПРИЛОЖЕНИЕ

Цель работы – разработка информационной системы для организации и проведения любительских автоспортивных мероприятий.

Разрабатываемая система должна обеспечивать регистрацию и аутентификацию пользователей, управление профилями пилотов, автомобилями, трассами и турнирами, предоставлять возможность подачи и обработки заявок на участие в соревнованиях, хранить результаты мероприятий, а также обеспечивать взаимодействие участников в рамках единой цифровой среды.

Результатом проделанной работы является спроектированное, разработанное и протестированное клиент-серверное веб-приложение для организации любительских автоспортивных мероприятий. При разработке были использованы язык программирования C#, платформа ASP.NET Core, язык TypeScript, библиотека React, СУБД Microsoft SQL Server, технология Entity Framework Core, а также средства контейнеризации Docker и другие вспомогательные библиотеки и инструменты разработки.

ПГУ1.090301.03.001 ПЗ

Изм.	Лист	№ докум.	Подпись	Дата	Лит.	Лист	Листов
Разраб.		Беляев Д. И.		28.05.2026	Информационная клиент-серверная система организации и проведения любительских автотранспортных мероприятий		
Провер.		Токарев А. Н.		28.05.2026			
Н. Контр		Бычков А. С.		28.05.2026			
Утверд.		Митрохин М. А.		28.05.2026			
Пояснительная записка						2	81
					ФВТ, каф. ВТ гр. 22ВВП1		

Содержание

Перечень сокращений и обозначений	5
Введение	7
1 Информационная клиент-серверная система организации и проведения любительских автотранспортных мероприятий	9
1.1 Анализ предметной области	9
1.1.1 Описание предметной области	9
1.1.2 Анализ аналогов	10
Вывод по первому подразделу	18
1.2 Техническое задание. Инструменты разработки	18
1.2.1 Техническое задание	18
1.2.2 Выбор и обоснование инструментов разработки	21
Вывод по второму подразделу	28
1.3 Проектирование и разработка	28
1.3.1 Сервер	29
1.3.2 Клиент	39
1.3.3 База данных	45
1.3.4 Docker и развертывание	54
Вывод по третьему подразделу	57
1.4 Тестирование	58
1.4.1 Проверка регистрации и авторизации пользователя	58
1.4.2 Проверка ведения профиля и управления гаражом	59
1.4.3 Проверка добавления и подтверждения трассы	59
1.4.4 Проверка создания турнира и подачи заявки на участие	60
1.4.5 Проверка обработки заявок и фиксации результатов	61
1.4.6 Проверка загрузки и привязки медиафайлов	62
Выводы по четвертому подразделу	62
1.5 Руководство пользователя	63
1.5.1 Главная страница	63

1.5.2 Регистрация, вход в систему и разграничение прав доступа	64
1.5.3 Работа с личным профилем и цифровой идентичностью пилота	66
1.5.4 Управление виртуальным гаражом	67
1.5.5 Работа с каталогом трасс	68
1.5.6 Работа с турнирами и заявками на участие	71
Вывод по пятому подразделу	74
2 Безопасность жизнедеятельности	75
Заключение	77
Список использованных источников	80
Приложение А – UML-диаграммы	82
Приложение А.1 – UML-диаграмма прецедентов	83
Приложение А.2 – UML-диаграмма последовательностей	84
Приложение А.3 – UML-диаграмма компонентов	85
Приложение А.4 – UML-диаграмма развертывания	86
Приложение Б –Листинг программы	87
Приложение В – Презентация	123

Перечень сокращений и обозначений

В этой работе используются следующие сокращения и обозначения

ИТ – информационные технологии

Минспорт – Министерство спорта

ПЭВМ – персональная электронно-вычислительная машина

РАФ – Российская Автомобильная Федерация

СУБД – система управления базами данных

API – (от англ. Application Programming Interface) – программный интерфейс приложения

CSS – (от англ. Cascading Style Sheets) – формальный язык описания внешнего вида документа (в тексте упоминается версия CSS3)

DOM – (от англ. Document Object Model) – объектная модель документа (в тексте упоминается концепция «виртуального DOM» для эффективного обновления интерфейса)

DTO – (от англ. Data Transfer Object) – объект передачи данных, используемый в архитектуре приложения для связи между слоями

FSD – (от англ. Feature-Sliced Design) – архитектурная методология для фронтенд-проектов, разделяющая код по предметным областям и уровням абстракции

GUID – (от англ. Globally Unique Identifier) – глобально уникальный идентификатор, используемый в качестве базового типа для сущностей базы данных

HTML – (от англ. HyperText Markup Language) – язык гипертекстовой разметки

HTTP – (от англ. HyperText Transfer Protocol) – протокол прикладного уровня передачи данных

JSON – (от англ. JavaScript Object Notation) – текстовый формат обмена данными (в тексте упоминается в контексте производительности ответов сервера)

JVM – (от англ. Java Virtual Machine) – виртуальная машина Java

JWT – (от англ. JSON Web Token) – веб-токен, используемый для поддержания пользовательской сессии и безопасности (хранения ключа)

					<i>ПГУ 1.090301.03.001 ПЗ</i>	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

LINQ – (от англ. Language-Integrated Query) – интегрированный язык запросов для удобной работы с данными на уровне объектной модели

REST – (от англ. Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределённого приложения (в тексте упоминается при построении REST API и REST-сервисов)

RPS – (от англ. Responses Per Second) – количество ответов (запросов) в секунду

SPA – (от англ. Single Page Application) – одностраничное приложение

SQL – (от англ. Structured Query Language) – язык структурированных запросов

UI – (от англ. User Interface) – пользовательский интерфейс

					<i>ПГУ 1.090301.03.001 ПЗ</i>	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В современном мире цифровая трансформация охватывает все сферы жизни, включая физическую культуру и спорт. На государственном уровне прямо указывается на необходимость перевода основных процессов, связанных с деятельностью спортивных организаций, на цифровые технологии. Так, в соответствии со стратегическим направлением в области цифровой трансформации физической культуры и спорта до 2030 года, утвержденным распоряжением Правительства Российской Федерации от 07.02.2024 № 264-р, приоритетной задачей является формирование цифровой среды, ориентированной на потребности различных групп населения [1].

Однако, в отличие от профессионального автомобильного спорта, характеризующегося высокой степенью регламентации и централизованного управления, любительский сегмент остается слабо структурированным. Ему присущи хаотичность информационного взаимодействия, отсутствие единого учета трасс, а также непрозрачность фиксации результатов и гоночного стажа участников. Существующие ИТ-решения преимущественно ориентированы на профессиональные соревнования и не охватывают потребности любительского сообщества, включая ведение истории участия, управление профилем пилота и инструменты самоорганизации мероприятий.

Актуальность темы обусловлена отсутствием комплексных ИТ-решений, объединяющих в рамках единой цифровой среды функции социальной сети для пилотов с инструментами автоматизации управления автоспортивными мероприятиями. Разработка системы позволяет объединить гонщиков, автоматизировать процессы организации и обеспечить прозрачность учета результатов, повысить доступность автоспорта для начинающих участников и способствует реализации государственных инициатив в области цифровизации спорта.

Объект исследования: процесс организации и проведения любительских автоспортивных мероприятий.

					<i>ПГУ 1.090301.03.001 ПЗ</i>	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

Предмет исследования: методы, модели и программные средства автоматизации управления автоспортивными мероприятиями, включая учет участников, трасс и результатов соревнований.

Цель работы: разработка информационной системы для организации и проведения любительских автоспортивных мероприятий, обеспечивающей автоматизацию ключевых процессов и объединение участников в единую цифровую среду.

Задачи исследования:

- провести анализ предметной области любительского автоспорта;
- исследовать существующие аналоги;
- сформулировать требования к разрабатываемой системе;
- выбрать технологический стек и инструменты разработки;
- спроектировать архитектуру системы;
- реализовать программное решение;
- провести тестирование и оценку эффективности разработанной системы.

Научная новизна работы заключается в разработке подхода к объединению функций социальной платформы и системы управления автоспортивными мероприятиями в рамках единой информационной системы, ориентированной на любительский сегмент. Практическая значимость работы заключается в возможности использования разработанной системы для организации и проведения любительских автоспортивных мероприятий, упрощения взаимодействия между участниками и организаторами.

В рамках данной работы рассматривается разработка клиент-серверного веб-приложения для организации любительских автоспортивных мероприятий.

В исследование не включаются:

- вопросы организации профессиональных соревнований;
- разработка мобильных приложений (рассматривается веб-клиент);
- интеграция с государственными информационными системами и лицензированием спортсменов.

1 Информационная клиент-серверная система организации и проведения любительских автотранспортных мероприятий

1.1 Анализ предметной области

1.1.1 Описание предметной области

Автоспорт – категория технических видов спорта, в которых участники соревнуются между собой за рулём автомобилей (прототип, легковой автомобиль, грузовик, внедорожник и так далее) [2].

Любительский автоспорт представляет собой активно развивающуюся, но слабо формализованную область, включающую трек-дни, клубные соревнования и локальные турниры. В условиях общей цифровой трансформации общества вопросы внедрения информационных технологий в сферу спорта приобретают особую значимость. Это подтверждается стратегическим направлением развития физической культуры и спорта до 2030 года, утвержденным распоряжением Правительства Российской Федерации от 07.02.2024 № 264-р, которое прямо указывает на необходимость перевода основных процессов, связанных с деятельностью спортивных организаций, на цифровые технологии и формировании среды, ориентированной на потребности разных групп населения [1].

Несмотря на это, в любительском автоспорте по-прежнему отсутствует единая информационная среда. Взаимодействие участников осуществляется через разрозненные каналы – социальные сети, мессенджеры и форумы. Это приводит к потере информации, усложняет доступ к актуальным данным о мероприятиях и снижает прозрачность процессов. Отдельной проблемой является отсутствие централизованного учета результатов. История участия пилотов, их достижения и гоночный стаж, как правило, не фиксируются в единой системе, что затрудняет объективную оценку уровня участников и их допуска к соревнованиям. Организация мероприятий также остается преимущественно ручным процессом. Регистрация участников, обработка заявок и публикация результатов

					ПГУ 1.090301.03.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

выполняются с использованием разрозненных инструментов, что увеличивает трудозатраты организаторов и повышает вероятность ошибок.

1.1.2 Анализ аналогов

Вопрос автоматизации процессов в любительском автоспорте поднимается не впервые, поэтому на рынке уже существуют различные сервисы и инструменты, частично решающие задачи регистрации участников или публикации расписаний. Их анализ позволит создать систему, опирающуюся на лучшие практики отрасли, но при этом свободную от ограничений, присущих текущим аналогам.

Среди аналогов можно выделить:

- официальный сайт Российской Автомобильной Федерации;
- сайт международного ралли Шелковый путь;
- зарубежная автоспортивная площадка onGRID.

1.1.2.1 Официальный сайт Российской Автомобильной Федерации

Сайт Российской Автомобильной Федерации в первую очередь официальный агрегатора спортивных мероприятий, входящих в календарь Минспорта России, и ориентирован на профессиональный уровень. Он будет актуален действующим спортсменам, обладателям лицензий РАФ и организаторам официальных соревнований. Скриншоты страниц сайта приведены на рисунках 1 – 3.

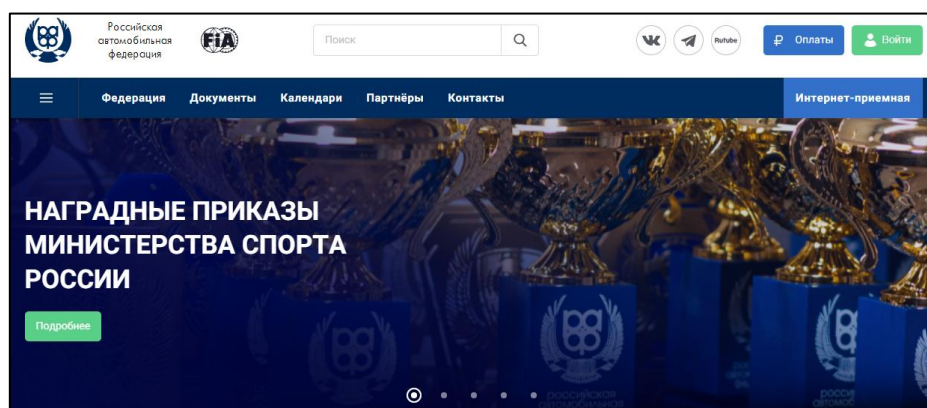


Рисунок 1 – Главная страница

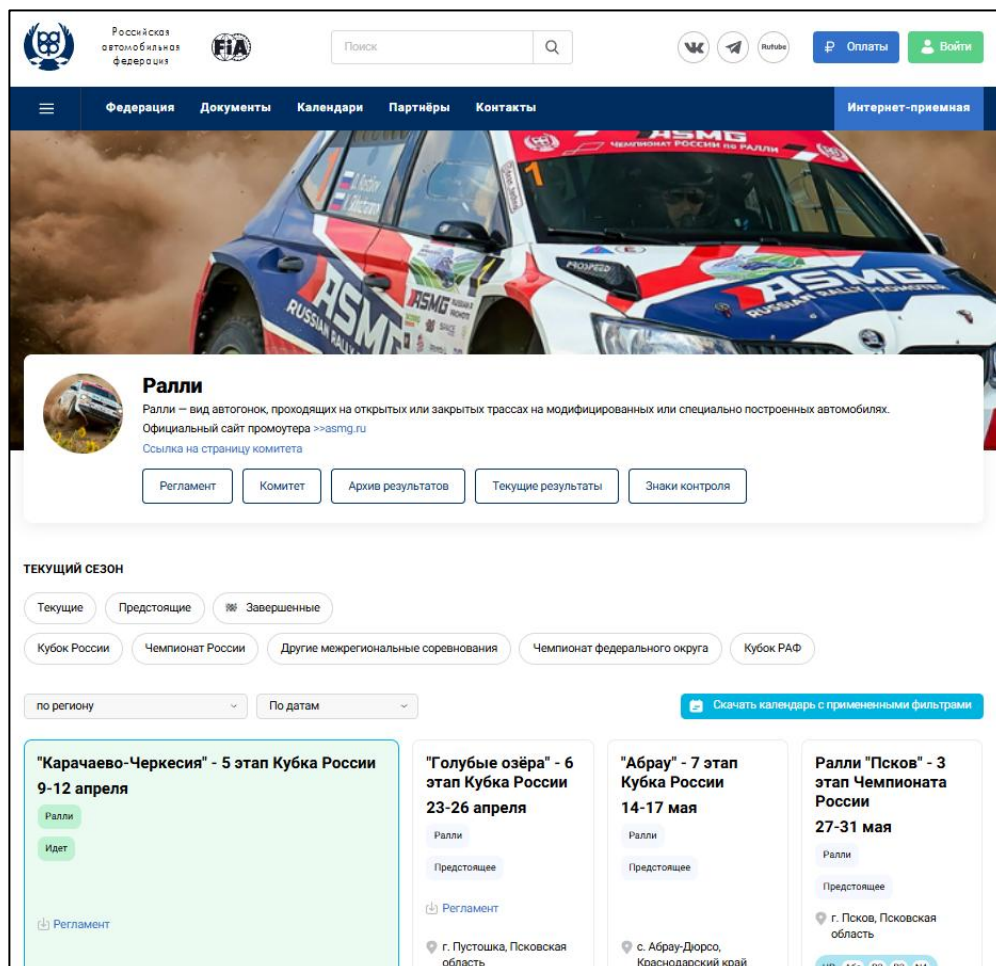


Рисунок 2 – Календарь конкретной дисциплины

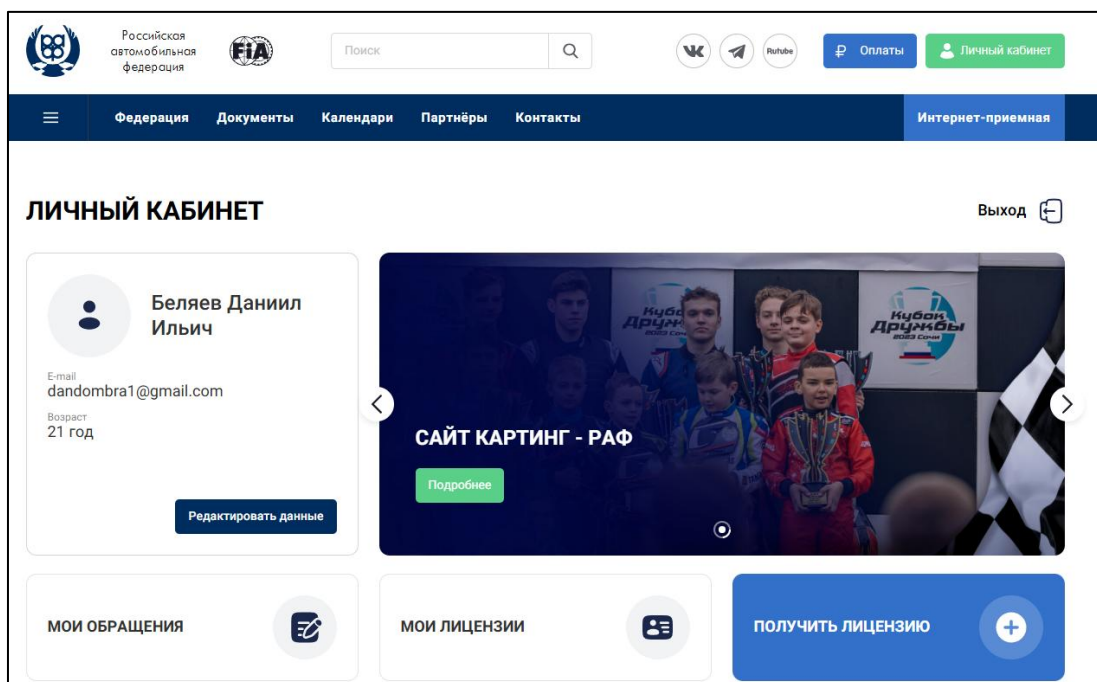


Рисунок 1.1.3 – Личный кабинет

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.03.001 ПЗ

Лист

11

Достоинства:

- удобный и интуитивно понятный интерфейс, обеспечивающий быстрый доступ к ключевым разделам;
- официальная государственная поддержка и статус аккредитованной спортивной федерации;
- наличие функционала для получения спортивной лицензии и оплаты членских взносов и пошлин непосредственно через сайт;
- единая сводка актуальных правил, регламентов, технических требований к автомобилям и тому подобное, что необходимо для участия в официальных соревнованиях под эгидой РАФ;
- публикация официальных результатов всероссийских и межрегиональных соревнований.

Недостатки:

- мероприятия в календаре представлены в виде ссылок на сторонние сайты организаторов, из-за чего отсутствует единообразие в подаче информации, дизайне страниц турниров и процедуре регистрации участников;
- отсутствует возможность для участников сообщества самостоятельно добавлять и публиковать любительские мероприятия;
- не предусмотрено ведение пользовательского «гаража»;
- отсутствует история участия пилота в соревнованиях;
- регистрация и полноценное использование личного кабинета доступны только при наличии действующей спортивной лицензии РАФ.

Таким образом для любительского сегмента функционал сайта не предусматривает инструментов ведения персональной истории выступлений, управления гаражом или самостоятельного создания турниров.

1.1.2.2 Сайт международного ралли Шелковый путь

Сайт международного ралли «Шелковый путь» представляет собой информационный портал конкретного крупного соревнования и нацелен на информирование зрителей и профессиональных участников об этапах гонки,

регламенте и результатах. Регистрация участников осуществляется путём заполнения и отправки отдельных форм, а не через единый личный кабинет с историей выступлений. Скриншоты страниц сайта приведены на рисунках 4 – 6.

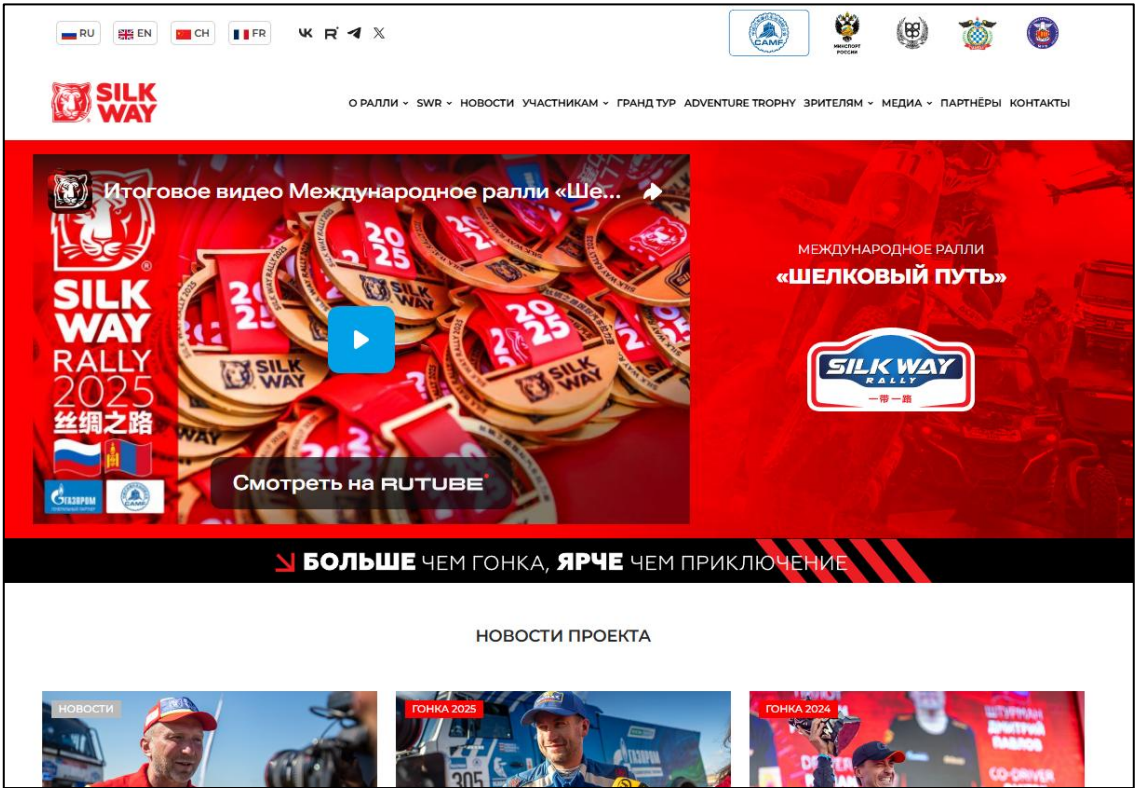


Рисунок 4 – Главная страница

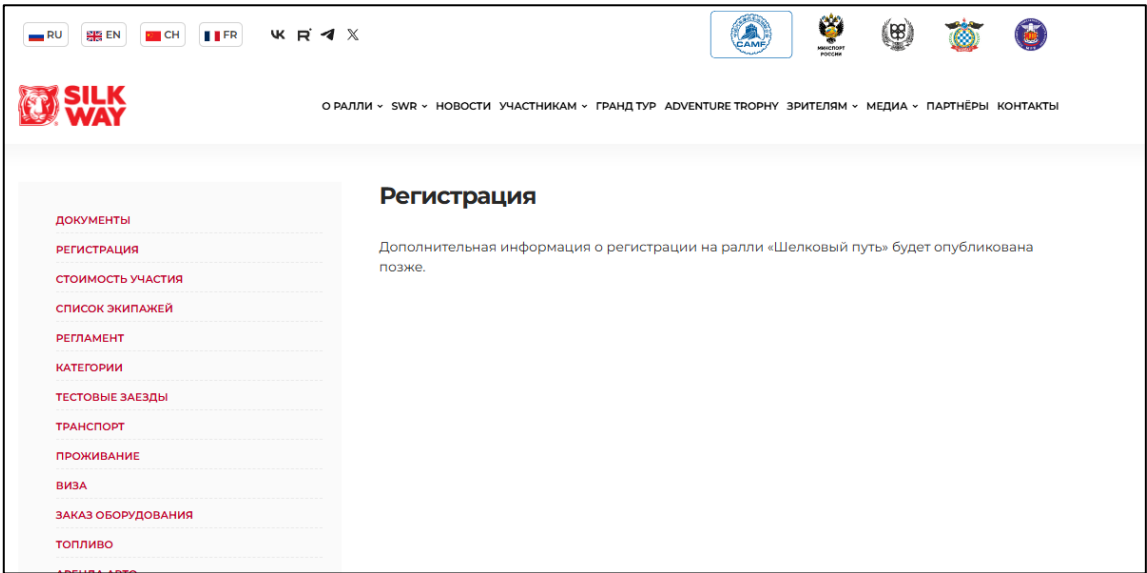


Рисунок 5 – Информация для участников

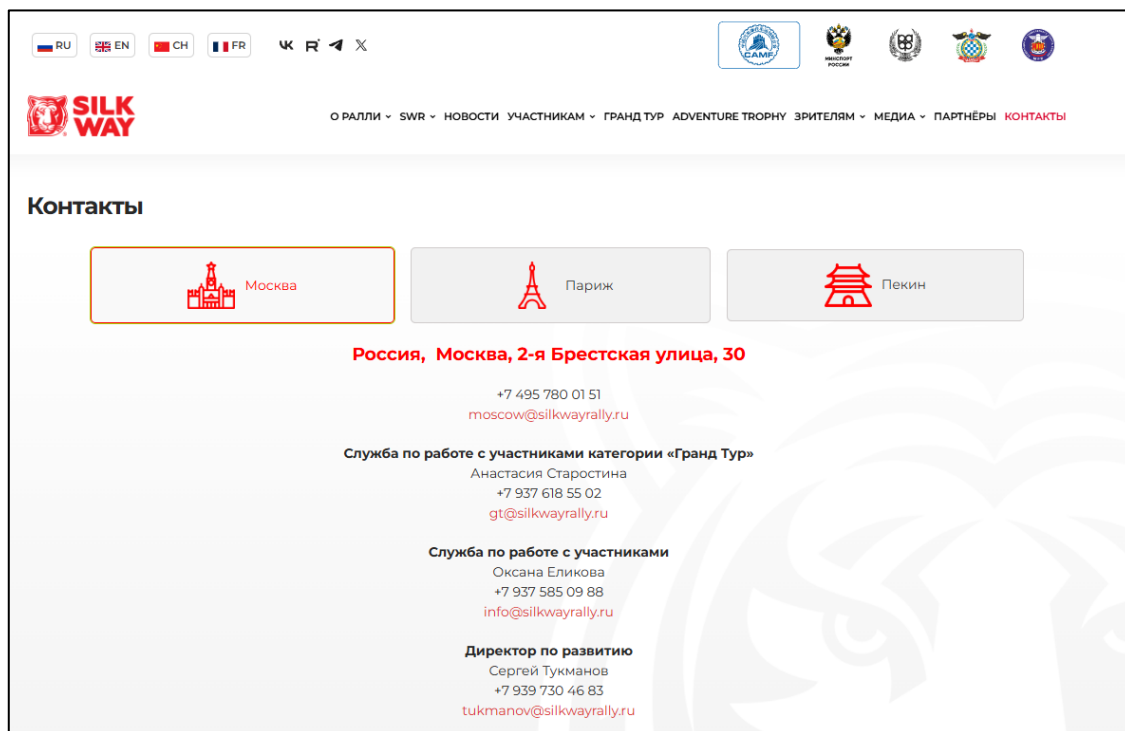


Рисунок 6 – Контактная информация

Достоинства:

- наличие версий сайта на нескольких языках, что делает информацию доступной для международной аудитории;
- подробное и структурированное представление условий регистрации, включая требования к лицензиям пилотов, возрасту, документам и техническому состоянию транспортных средств для каждой категории;
- наличие раздела с актуальными новостями и анонсами, позволяющего отслеживать ключевые события ралли-марафона, этапы приёма заявок, изменения маршрута и другую важную информацию.

Недостатки:

- отсутствие полноценного личного кабинета участника. Регистрация осуществляется через заполнение форм и их отправку по электронной почте, что усложняет и замедляет процесс;
- содержится информация только о конкретном соревновании.

Таким образом, сайт решает задачи освещения одного масштабного события, но не предлагает инфраструктуры для постоянного взаимодействия

пилотов-любителей, накопления гоночного стажа и организации локальных турниров сообществом.

1.1.2.3 Зарубежная площадка onGRID

Платформа onGRID представляет собой сервис для цифрового администрирования автоспортивных мероприятий, ориентированный на профессиональные гоночные серии, клубы и лицензированных пилотов. Она предоставляет инструменты управления событиями, электронного документооборота с юридически значимой подписью и верификации участников. Регистрация пользователей требует прохождения процедуры идентификации и привязки гоночной лицензии, а профиль пилота позволяет хранить информацию об автомобилях. Скриншоты страниц сайта приведены на рисунках 7 – 11.

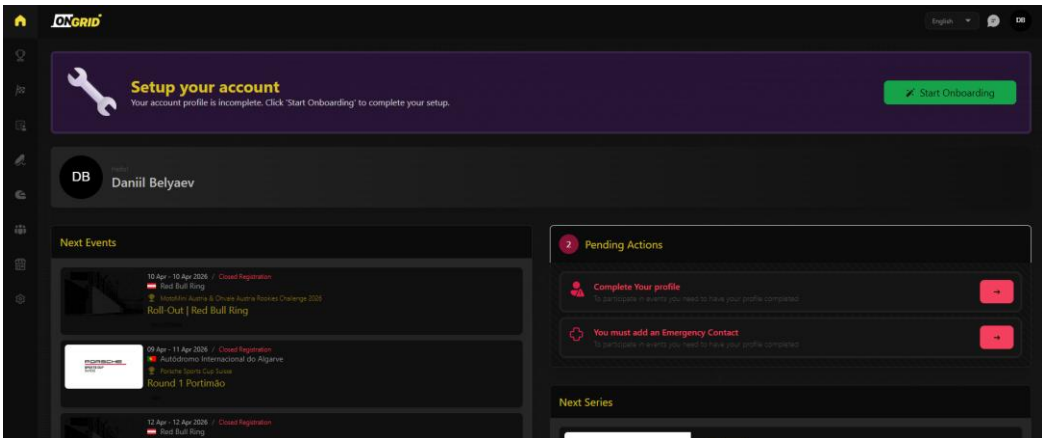


Рисунок 7 – Главная страница

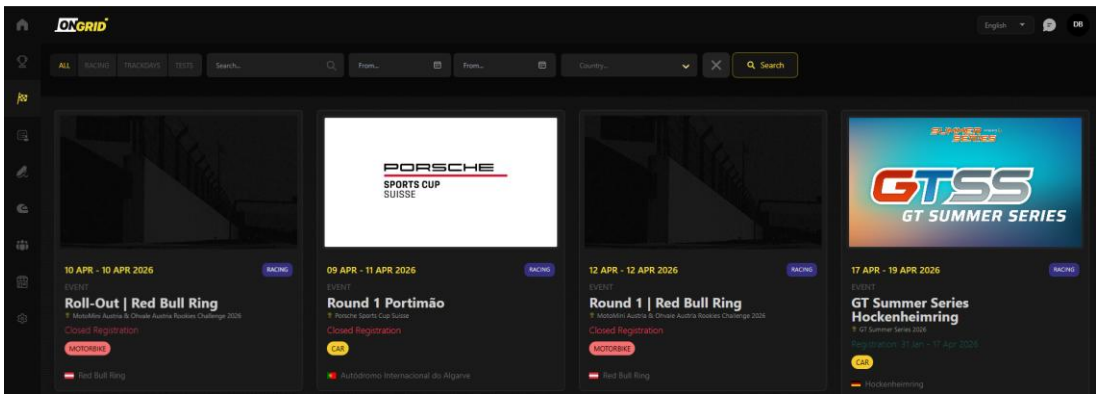


Рисунок 8 – Мероприятия

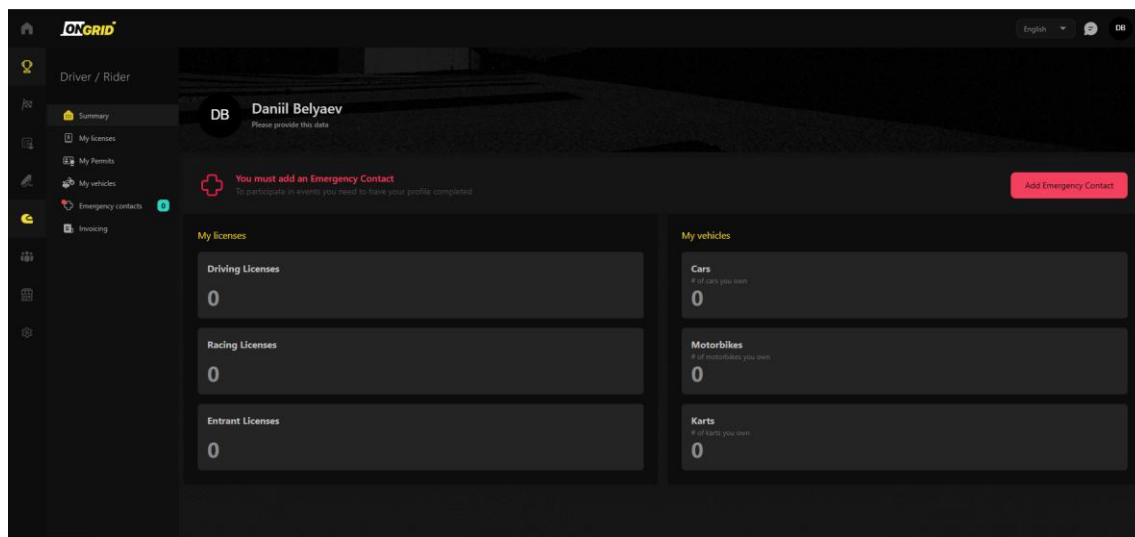


Рисунок 9 – Профиль

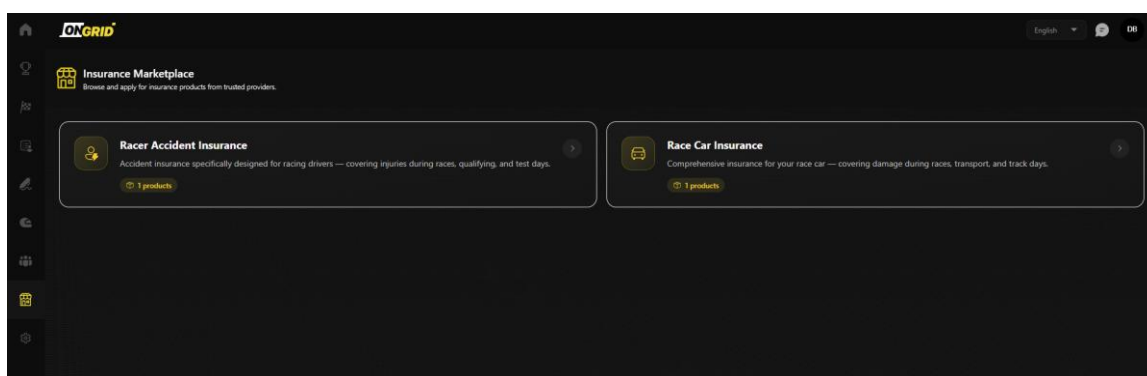


Рисунок 10 – Магазин

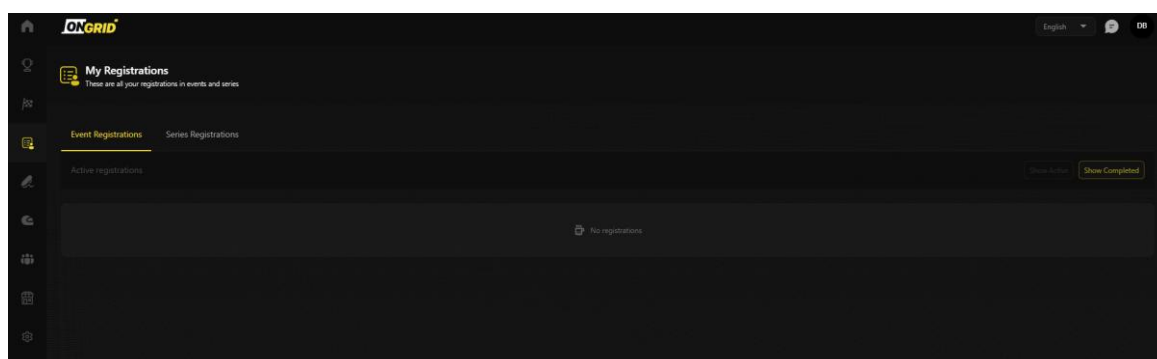


Рисунок 11 – Заявки

Достоинства:

– платформа объединяет все этапы жизненного цикла соревнования: от создания страницы мероприятия и публикации регламента до сбора заявок,

верификации участников и финальной выгрузки протоколов, что исключает необходимость использования разрозненных инструментов;

- использование eIDAS-совместимой квалифицированной электронной подписи и процедур строгой идентификации пользователей придаёт регистрационным формам и подписанным документам статус официальных;

- пользователь имеет возможность хранить личные данные, сведения о гоночной лицензии и привязывать к профилю собственные автомобили, формируя гараж;

- возможность приобретения страховки пилота и автомобиля.

Недостатки:

- добавление новых гоночных площадок осуществляется централизованно организаторами; пользователи не могут самостоятельно предложить трассу, а сообщество проголосовать за подтверждение её статуса, что исключает актуализацию базы площадок без вмешательства администраторов;

- обязательные процедуры идентификации личности и требование наличия действующей гоночной лицензии для полноценного использования функционала создают барьер для любителей, не имеющих официального спортивного статуса и желающих участвовать в неформальных клубных мероприятиях;

- добавление новых турниров производится официальными организаторами; пользователи не могут самостоятельно создать любительский турнир;

- информация о пользователях, их автомобилях и истории выступлений недоступна для просмотра другими участниками сообщества, что исключает прозрачность, важную для любительского сегмента;

- отсутствие мероприятий на территории России.

Таким образом, платформа успешно решает задачи автоматизации административных процессов в автоспорте, но не предлагает функционала для самоорганизации любительского сообщества: отсутствуют механизмы добавления

и подтверждения трасс, возможности создания пользователями турниров, информация о публичной истории выступлений.

Вывод по анализу предметной области

Проведенный анализ показал, что любительский автоспорт характеризуется недостаточным уровнем цифровизации и отсутствием централизованных инструментов управления мероприятиями. Существующие решения ориентированы на профессиональный сегмент либо реализуют отдельные функции, не обеспечивая комплексного подхода. Это обуславливает фрагментацию данных и ограниченность механизмов взаимодействия участников.

Таким образом, возникает необходимость разработки специализированной информационной системы, ориентированной на любительский сегмент и обеспечивающей интеграцию процессов управления мероприятиями и взаимодействия пользователей.

1.2 Техническое задание. Инструменты разработки

1.2.1 Техническое задание

Необходимо разработать клиент-серверное веб-приложение – информационную систему для организации и проведения любительских автоспортивных мероприятий. Приложение должно объединять пилотов, трассы и турниры в единой цифровой среде, обеспечивая автоматизацию процессов регистрации, планирования событий, учёта результатов и социального взаимодействия участников. Диаграммы прецедентов пользователя представлены в приложении А.1. Для этого необходимо реализовать определенные модули.

1.2.1.1 Модуль управления доступом и безопасностью

Цель – обеспечить безопасный вход и разграничение прав в зависимости от опыта и роли пользователя.

Возможности пользователя:

– регистрация нового пилота – пользователь должен иметь возможность создать учетную запись, указав email, пароль и уникальный никнейм;

- идентификация и аутентификация – проверка прав доступа при входе в систему и выдача токенов для поддержания сессии;

- разграничение ролей – Система должна автоматически проверять наличие прав (Racer, Organizer, Moderator, SuperAdmin) при попытке выполнить специфические действия.

1.2.1.2 Модуль управления цифровой идентичностью

Цель – ведение истории достижений пилота и управление его техническим парком.

Возможности пользователя:

- ведение профиля пилота – возможность редактирования биографии и загрузки аватара для узнаваемости в сообществе;

- мониторинг гоночного стажа – автоматическое обновление счетчика завершенных гонок после фиксации результатов турнира;

- управление виртуальным гаражом – добавление автомобилей с указанием марки, модели и технического класса, создание фотогалереи автотранспортного средства;

- валидация при удалении – запрет на удаление автомобиля, если он уже заявлен на активный турнир, для сохранения целостности данных.

1.2.1.3 Модуль гоночных площадок

Цель – формирование базы трасс силами сообщества с механизмом контроля качества.

Возможности пользователя:

- предложение новой трассы – пилот с определенным стажем должен иметь возможность добавить площадку в статусе «Неофициальная»;

- общественное подтверждение – механизм голосования пользователей за трассу для автоматического перевода её в статус «Подтверждена» при достижении порога голосов;

- административный контроль – ручное подтверждение или присвоение трассе статуса «Официальная» модератором системы.

1.2.1.4 Модуль организации мероприятий

Цель – полная автоматизация жизненного цикла автоспортивного события, от создания турнира и ведения заявок, до завершения турнира.

Возможности пользователя:

- создание турнира – настройка параметров гонки (дата, трасса, требования к классу авто и минимальному стажу участников);
- управление организаторским составом – возможность добавления соорганизаторов из числа опытных пользователей системы;
- обработка заявок – подача заявок пилотами с проверкой соответствия класса автомобиля и стажа, а также ручное подтверждение или отклонение заявок организатором;
- управление ходом гонки – смена статусов турнира, отслеживание участниками хода гонки;
- фиксация результатов – внесение итоговых позиций и времени круга для каждого участника после завершения заездов.

1.2.1.5 Модуль управления медиаконтентом

Цель – безопасное хранение и визуализация медиаконтента пользователей системы.

Возможности пользователя:

- загрузка и валидация изображений – проверка формата, размера файла и принадлежности его конкретному пользователю при взаимодействии с фотографиями;
- привязка медиа к сущностям – создание фотогалерей для автомобилей, трасс и конкретных турниров. Добавление аватара пользователя.

1.2.1.6 Минимальные системные требования

Минимальные системные требования для запуска и эксплуатации клиента:

- браузер Google Chrome, Mozilla Firefox, Opera (с поддержкой HTML5, CSS3);
- минимальная скорость интернета: 2-3 Мбит/сек;

- оперативная память: не менее 1 ГБ.

Минимальные системные требования для запуска и эксплуатации сервера:

- операционная система: Windows 10/11 или Linux;
- установленный Docker Desktop / Docker Engine;
- процессор: не менее 2 ядер с тактовой частотой от 2.0 ГГц;
- оперативная память: не менее 4 ГБ;
- минимальная скорость интернета: 2-3 Мбит/сек;
- свободное место на диске: не менее 10 ГБ.

1.2.2 Выбор и обоснование инструментов разработки

В этом пункте проведен анализ языков программирования, фреймворков, СУБД, и сред разработки. Исходя из анализа выбраны технологии, которые обеспечивают наибольшее удобство разработки и поддержки приложения.

1.2.2.1 Выбор платформы исполнения

Выбор платформы для функционирования приложения – первый этап проектирования. Он напрямую влияет на технологический стек разработки и определяет характер взаимодействия пользователей с системой.

К платформе предъявляются следующие основные требования:

- обеспечение максимально широкого охвата пользователей;
- высокая скорость создания продукта и простота его дальнейшего сопровождения.

Для достижения этих целей и упрощения разработки, система строится вокруг сервера с универсальным API, что делает серверную часть независимой от типа клиента и позволяет подключать любые клиентские приложения: мобильные, десктопные и другие.

На начальном этапе, наиболее удобным решением является разработка единого веб-клиента с адаптивной вёрсткой. Такой подход исключает необходимость одновременной поддержки нескольких отдельных приложений, обеспечивая комфортный доступ к системе как с персональных компьютеров, так

и с мобильных устройств, при этом сохраняя возможность для дальнейшего развития экосистемы без изменения серверной логики.

1.2.2.2 Выбор языка программирования

Выбор языка реализации серверной и клиентской частей является вторым этапом проектирования, от которого зависит как скорость разработки, так и эксплуатационные характеристики конечного продукта. В процессе анализа рассматривались несколько технологических стеков, включая Java, Python и связку C# с JavaScript/TypeScript.

Java – это строго типизированный объектно-ориентированный язык программирования, компилируемый в байт-код и исполняемый виртуальной машиной Java (JVM). Одной из ключевых особенностей языка является кроссплатформенность, позволяющая запускать приложения в различных операционных средах без изменения исходного кода. Java широко применяется при создании корпоративных информационных систем благодаря высокой надежности, зрелости технологий и развитой экосистеме [3]. Существенным преимуществом является наличие мощных фреймворков, таких как Spring и Spring Boot, предоставляющих инструменты для внедрения зависимостей, работы с базами данных и построения REST API. Язык поддерживает многопоточность на уровне платформы, что позволяет эффективно разрабатывать высоконагруженные серверные приложения.

Python – высокоуровневый интерпретируемый язык программирования общего назначения, отличающийся лаконичным синтаксисом и высокой скоростью разработки. Динамическая типизация и минималистичный синтаксис позволяют сократить объем кода и упростить поддержку проектов. Python широко применяется в веб-разработке благодаря таким фреймворкам, как Django, FastAPI и Flask. Они предоставляют готовые инструменты для реализации серверной логики и взаимодействия с базами данных. Одним из преимуществ языка является обширная экосистема библиотек, охватывающая широкий спектр задач – от веб-разработки до анализа данных и машинного обучения. Также Python обладает

активным сообществом, что обеспечивает доступность документации и готовых решений [3].

C# – объектно-ориентированный язык программирования, развиваемый в рамках платформы .NET [4]. Он сочетает строгую статическую типизацию с современными возможностями языка, включая поддержку асинхронного программирования. Платформа .NET предоставляет развитую инфраструктуру для создания веб-приложений, которая включает инструменты для работы с базами данных, построения API и обеспечения безопасности. Менеджер пакетов NuGet открывает доступ разработчикам к большому количеству пакетов, которые охватывают многие сценарии разработки – от работы с базами данных и веб-сервисами до машинного обучения и облачных вычислений. C# обладает одним из крупнейших и наиболее активных сообществ разработчиков в мире. Сообщество .NET насчитывает более 5 миллионов разработчиков, которые активно обмениваются опытом [3]. По результатам независимых тестов производительности веб-фреймворков платформа .NET демонстрирует высокую скорость работы [5]. Дополнительным преимуществом является официальная поддержка со стороны Microsoft и наличие версий с долгосрочной поддержкой.

JavaScript – мультипарадигменный язык, который широко используется для реализации клиентской логики в веб-браузерах. Его поддержка популярных браузеров обеспечивает работоспособность интерфейса на любых устройствах – от настольных компьютеров до мобильных телефонов [6]. Для повышения надежности и удобства разработки используется язык TypeScript, являющийся его надмножеством. TypeScript добавляет статическую типизацию. Это позволяет формализовать структуру данных и снизить количество ошибок на этапе разработки. При этом он совместим с экосистемой JavaScript.

В результате анализа, комбинация C# на стороне сервера и TypeScript на стороне клиента формирует технологический стек, обеспечивающий производительность, надёжность, удобство разработки и долгосрочную

поддержку информационной системы для организации любительских автоспортивных мероприятий.

1.2.2.3 Выбор фреймворка

Выбор фреймворков для реализации клиентской и серверной частей системы зависит от требований к производительности, масштабируемости и удобству разработки и выбранным технологическим стеком.

React – это библиотека для разработки пользовательских интерфейсов, основанная на компонентном подходе. Она позволяет разбивать интерфейс на независимые переиспользуемые элементы, что упрощает разработку и сопровождение приложения. Использование виртуального DOM обеспечивает эффективное обновление интерфейса за счёт перерисовки только изменённых элементов. React широко применяется при создании одностраничных приложений и отличается высокой гибкостью, позволяя самостоятельно формировать структуру проекта и выбирать дополнительные инструменты.

Angular – это полнофункциональный фронтенд-фреймворк, предоставляющий готовую архитектуру для построения клиентских приложений. Он включает встроенные средства для маршрутизации, работы с HTTP-запросами и управления состоянием. Angular ориентирован на разработку крупных приложений и использует TypeScript в качестве основного языка. Но его использование связано с более высокой сложностью и строгими архитектурными ограничениями.

ASP.NET Core – это серверный фреймворк, входящий в состав платформы .NET и предназначенный для разработки веб-приложений и REST-сервисов. Он является кроссплатформенным и предоставляет встроенные средства для обработки HTTP-запросов, маршрутизации, аутентификации и авторизации. Фреймворк обеспечивает интеграцию с базами данных, в частности через Entity Framework Core, и демонстрирует высокую скорость, что подтверждается результатами независимых тестов [5]. Сравнение производительности популярных фреймворков показано на рисунке 12.

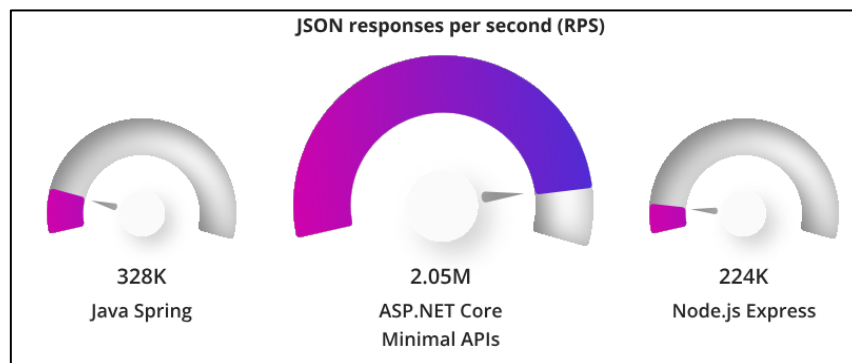


Рисунок 12 – Результаты тестирования производительности

В результате анализа для реализации клиентской части выбран React, что обусловлено его гибкостью, высокой производительностью и широким распространением. Для серверной части выбран ASP.NET Core как производительное и надежное решение, интегрированное с платформой .NET и обеспечивающее удобную разработку REST API.

1.2.2.4 Выбор СУБД

Выбор системы управления базами данных определяется требованиями к надежности хранения информации, производительности обработки запросов и удобству взаимодействия с приложением.

MySQL – одна из наиболее распространённых реляционных СУБД, широко применяемая в веб-разработке [7]. Она отличается простотой развертывания, высокой скоростью работы в типовых сценариях и широкой поддержкой со стороны сообщества. MySQL хорошо подходит для проектов с относительно простой структурой данных, однако при работе со сложными запросами и высоконагруженными системами её возможностей может быть недостаточно.

PostgreSQL представляет собой объектно-реляционную систему управления базами данных с открытым исходным кодом, ориентированную на надежность и соответствие стандартам SQL. Она поддерживает сложные транзакции, расширяемую систему типов данных и механизмы репликации. Подходит для систем, требующих высокой гибкости и работы со сложной логикой обработки данных.

Microsoft SQL Server – реляционная система управления базами данных, разработанная компанией Microsoft. Является одной из наиболее распространённых СУБД [7]. Она ориентирована на использование в корпоративных и веб-приложениях и обеспечивает высокую производительность, надежность и безопасность хранения данных. СУБД поддерживает работу с транзакциями, хранимыми процедурами и другими механизмами обработки данных на стороне сервера. Дополнительным преимуществом является тесная интеграция с платформой .NET, что упрощает разработку и сопровождение приложений.

Entity Framework Core – это технология объектно-реляционного отображения, входящая в экосистему .NET. Она позволяет взаимодействовать с базой данных на уровне объектной модели, минимизируя необходимость написания SQL-запросов вручную. LINQ-запросы, механизмы отслеживания изменений и системы миграций упрощают разработку и поддержку структуры базы данных и снижают вероятность ошибок при работе с данными.

В результате анализа для хранения данных выбрана система Microsoft SQL Server, что обусловлено её надежностью, производительностью и интеграцией с платформой .NET. В качестве инструмента взаимодействия с базой данных используется Entity Framework Core, так как он обеспечивает удобную работу с данными на уровне объектной модели и повышает скорость и эффективность разработки.

1.2.2.5 Выбор среды разработки

Выбор среды разработки оказывает влияние на удобство реализации проекта, скорость разработки и сопровождение программного продукта.

WebStorm представляет собой интегрированную среду разработки, ориентированную на создание клиентских приложений с использованием JavaScript и TypeScript. Среда предоставляет широкий набор встроенных инструментов: автодополнение кода, рефакторинг, отладка, интеграция с системами контроля версий. Поддержка современных фронтенд-фреймворков

делает её удобным решением для разработки интерфейсов. Но платная лицензия может являться ограничивающим фактором.

Rider – интегрированная среда разработки от компании JetBrains, предназначенная для работы с платформой .NET. Она сочетает возможности IntelliJ-платформы с инструментами для разработки на C# и ASP.NET Core. Среда поддерживает анализ кода, рефакторинг и интеграцию с системами контроля версий, а также является кроссплатформенной. Но её распространённость ниже по сравнению с Visual Studio, а использование в коммерческих целях требует приобретения лицензии [8].

Visual Studio Code – лёгкий и расширяемый редактор кода, разработанный компанией Microsoft. Он широко используется в разработке и поддерживает большое количество технологий за счёт системы расширений [8]. Гибкость настройки позволяет адаптировать среду под конкретные задачи, однако часть функциональности требует дополнительных настроек.

Visual Studio – интегрированная среда разработки, ориентированная на создание приложений на платформе .NET. Она обеспечивает глубокую интеграцию с языком C# и фреймворком ASP.NET Core, предоставляет развитые инструменты отладки, тестирования и работы с проектами. Благодаря этому среда является одним из основных инструментов разработки в экосистеме .NET, несмотря на высокие требования к ресурсам [8].

Docker представляет собой платформу контейнеризации, позволяющую запускать приложения в изолированных средах. Использование контейнеров обеспечивает воспроизводимость окружения и упрощает развертывание системы, позволяя запускать серверную часть, базу данных и другие компоненты в единой конфигурации независимо от целевой среды.

В результате анализа для разработки серверной части выбрана среда Visual Studio 2022, обеспечивающая максимальную интеграцию с платформой .NET и удобство разработки. Для реализации клиентской части используется Visual Studio Code как гибкий и лёгкий инструмент работы с современными веб-

технологиями. Для развертывания и обеспечения согласованности среды выполнения применяется Docker.

Вывод по техническому заданию и инструментам разработки

В рамках данной главы было сформировано техническое задание на разработку информационной системы и проведён анализ инструментов, используемых для её реализации.

Определены основные функциональные требования к системе, включающие регистрацию и аутентификацию пользователей, управление профилями пилотов, автомобилями, трассами и турнирами, а также реализацию механизмов взаимодействия участников и разграничения прав доступа, что позволяет сформировать представление о структуре и назначении разрабатываемого приложения.

В результате анализа технологических решений выбран стек, обеспечивающий необходимую производительность, масштабируемость и удобство разработки. Для серверной части выбран язык C# и платформа .NET с использованием ASP.NET Core, для клиентской – TypeScript и библиотека React. В качестве системы управления базами данных выбрана Microsoft SQL Server, а для взаимодействия с данными – Entity Framework Core. Разработка осуществляется с использованием Visual Studio и Visual Studio Code, а развертывание системы обеспечивается с применением контейнеризации Docker.

Таким образом, выбранные технологии и сформулированные требования создают основу для реализации информационной системы, соответствующей задачам организации и проведения любительских автоспортивных мероприятий.

1.3 Проектирование и разработка

Алгоритм работы приложения заключается в том, что пользователь взаимодействует с клиентской частью системы через веб-интерфейс. В зависимости от выполняемого действия клиент формирует HTTP-запрос к серверу и передает необходимые данные, а для защищенных операций также данные

аутентификации. После получения запроса сервер выполняет его обработку: проверяет входные данные, права доступа, после чего обращается к базе данных для выполнения операции над данными. Если на любом этапе возникает ошибка, то сервер возвращает клиенту сообщение об ошибке. В случае успешного выполнения сервер формирует ответ и отправляет клиенту данные или подтверждение выполненного действия. Диаграмма последовательностей, отражающая общий порядок взаимодействия клиента, сервера и базы данных, представлена на рисунке 13 и приложении А.2.

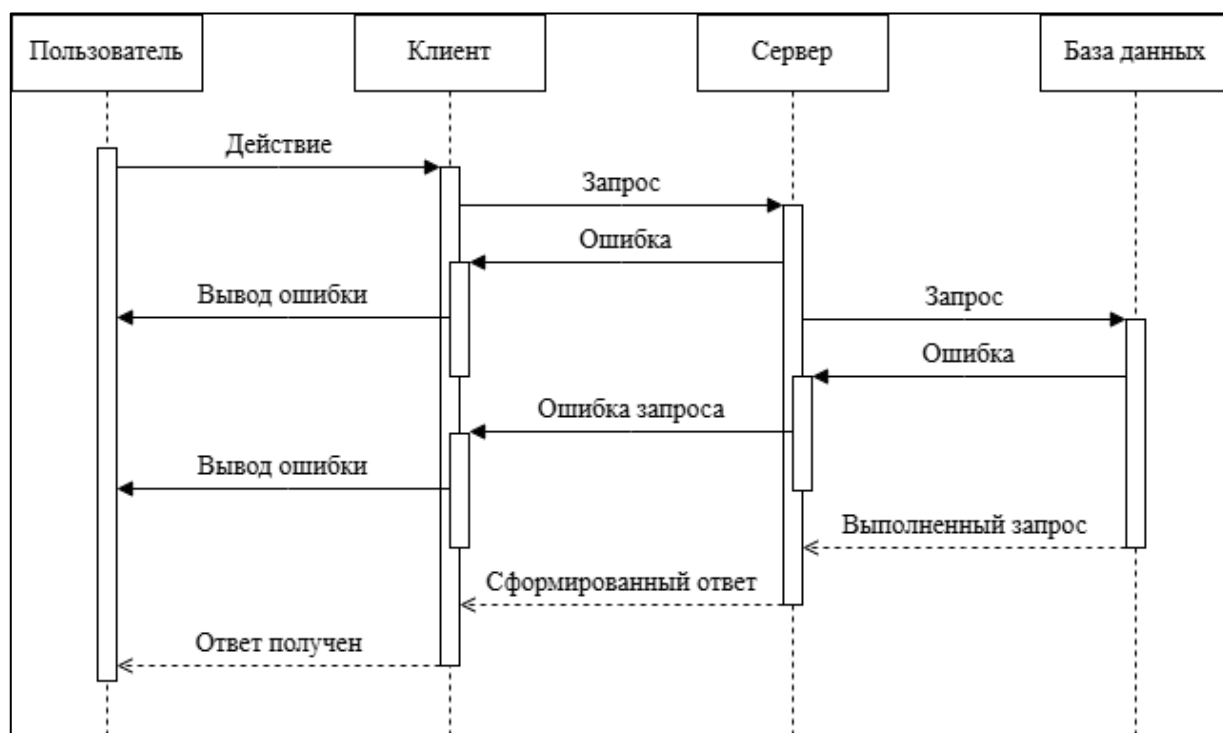


Рисунок 13 – Диаграмма последовательностей

1.3.1 Сервер

1.3.1.1 Проектирование архитектуры

В рамках разработки информационной системы был выбран гибридный архитектурный подход, сочетающий принципы Clean Architecture и методологию Feature-Sliced Design [9, 10]. Такое решение обусловлено необходимостью одновременно обеспечить строгую изоляцию бизнес-логики от инфраструктурных деталей и организовать прикладной код по предметным областям, а не по

техническим слоям. Паттерн «Чистая архитектура» подразумевает разделение проекта на слои. Зависимости в исходном коде могут указывать только внутрь, по направлению к высокоуровневым политикам.

Физически серверная часть разделена на четыре основных проекта:

- MotorsportErp.Domain – слой предметной области – содержит ключевые бизнес-сущности, интерфейсы и базовые типы. Данный слой описывает фундаментальные объекты системы, такие как пользователи, автомобили, трассы, турниры, заявки, результаты и медиафайлы, а также связи между ними. Слой Domain не содержит зависимостей от прикладной логики, инфраструктуры, фреймворков доступа к данным и т.д. Он выполняет роль ядра системы, в котором фиксируются основные понятия предметной области;

- MotorsportErp.Application – слой прикладной логики – содержит бизнес-логику и сценарии использования системы. В данном слое применен Feature-Sliced Design. Код группируется по конкретным предметным областям. Внутри каждой инкапсулированы свои контракты, интерфейсы и сервисы. Этот слой уже опирается на модели предметной области, но не знает, каким именно способом будут храниться данные или как будет осуществляться отправка HTTP-ответов;

- MotorsportErp.Infrastructure – слой инфраструктуры – реализует интерфейсы, объявленные в слое приложения. Включает логику работы с базой данных, которая реализует паттерн Repository, файловой системой, интеграцию с внешними сервисами и провайдерами. Благодаря такому подходу прикладной код не содержит прямой зависимости от конкретной базы данных, файловой системы или конкретной реализации криптографии;

- MotorsportErp.WebApi – слой представления – является входной точкой в приложение. В нем выполняется конфигурация контейнера зависимостей, подключение базы данных, регистрация сервисов и другое. Этот слой отвечает за прием HTTP-запросов, преобразование входных данных в объекты прикладного слоя и формирование HTTP-ответов. Его задача состоит в объединении прикладных сервисов и архитектурных слоев в единое приложение.

На рисунке 14 и приложении А.3 показаны связи между слоями.

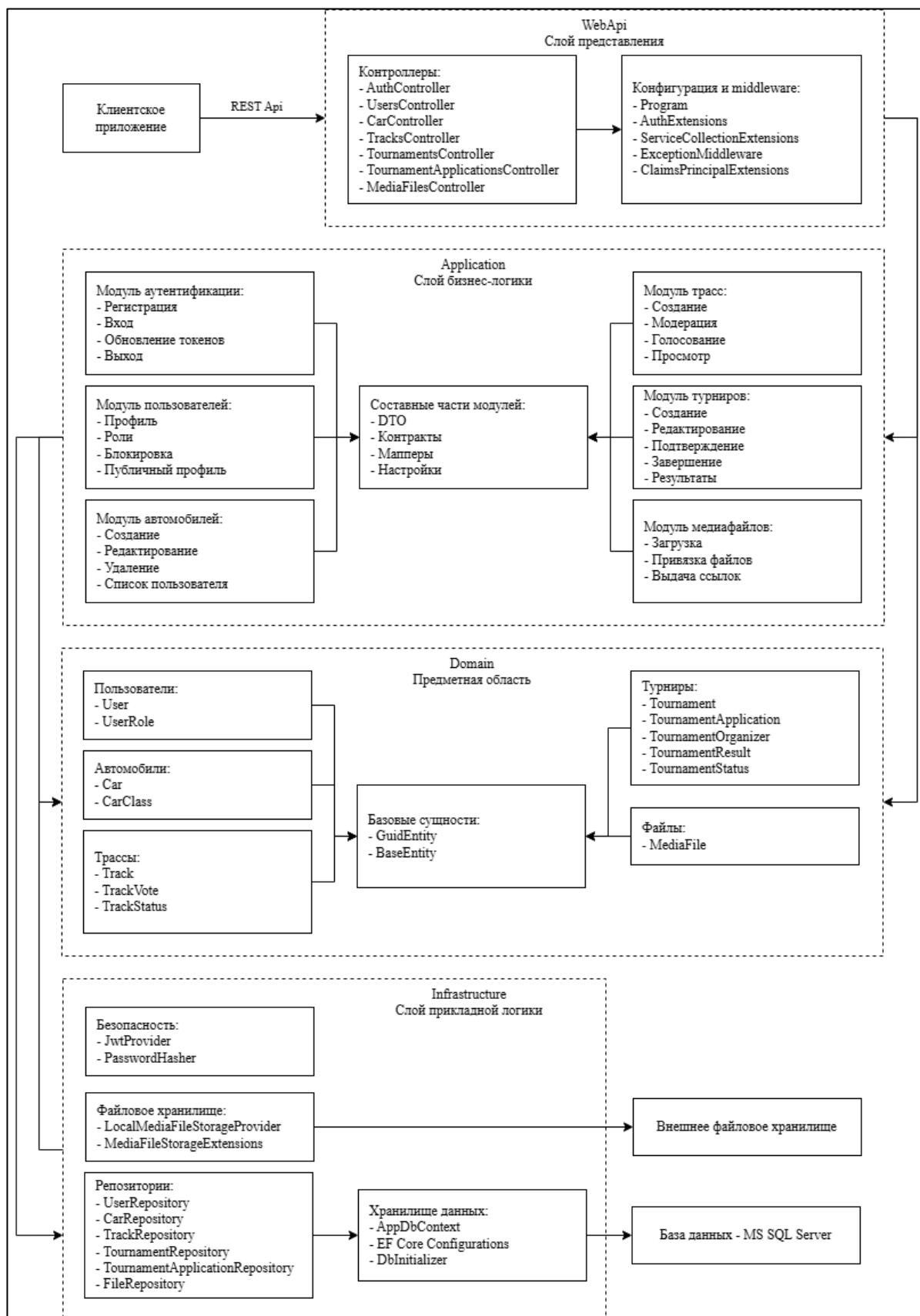


Рисунок 14 – Архитектура сервера

1.3.1.2 Реализация бизнес-логики

Основная бизнес-логика сосредоточена в проекте MotorsportErp.Application и организована по предметным модулям. Такое решение позволяет проследить связь между бизнес-требованием и его реализацией в коде.

Модуль управления доступом и безопасностью реализован в модуле Auth. Он включает сценарии регистрации, входа, обновления пары токенов и завершения пользовательской сессии. Его задачей является обеспечение безопасного входа и регистрации, контроль доступа к защищенным ресурсам приложения. Алгоритм работы входа и регистрации отображен на рисунке 15.

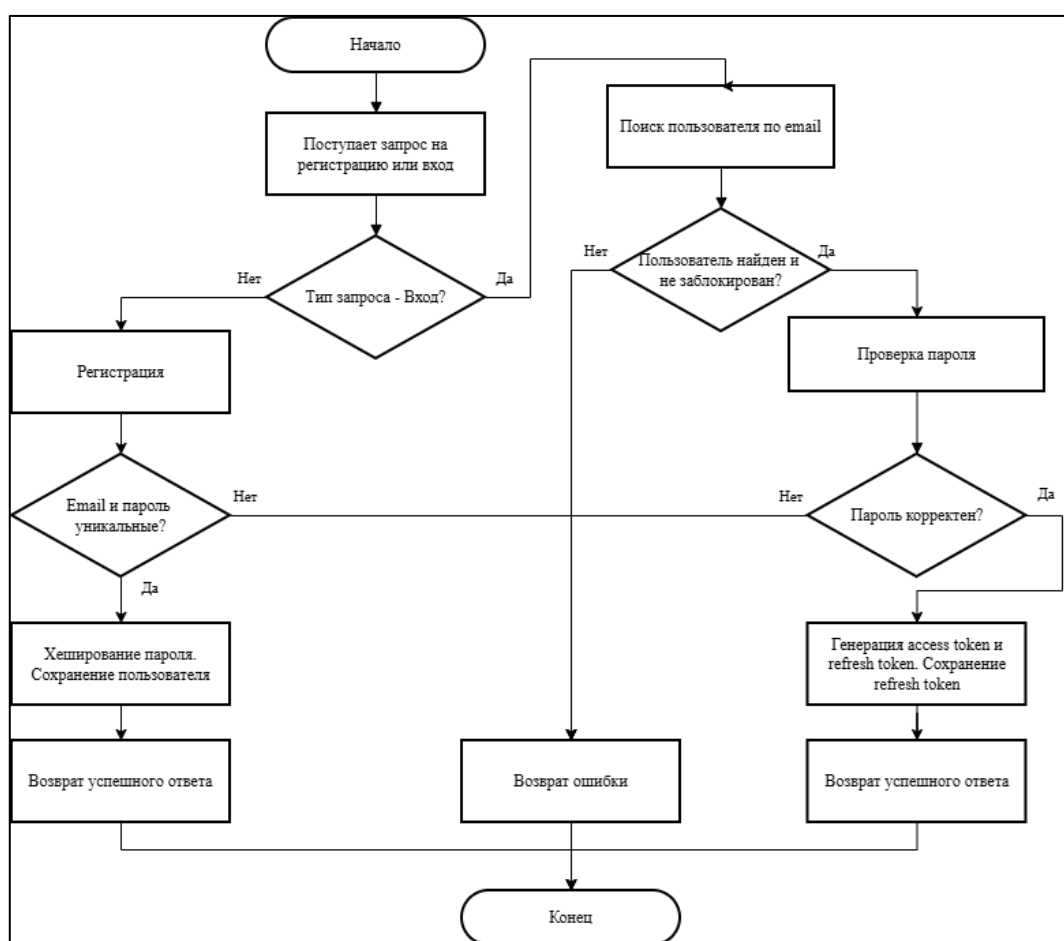


Рисунок 15 – Алгоритм работы входа и регистрации

При регистрации нового пользователя система выполняет проверку уникальности адреса электронной почты и никнейма. После этого пароль преобразуется в криптографический хеш с использованием алгоритма PBKDF2 и

случайной соли. Реализация в проекте продемонстрирована в листинге 1. Такой подход снижает риск компрометации пользовательских данных при утечке базы данных.

Листинг 1 – Хеширование пароля

```
using var algorithm = new Rfc2898DeriveBytes(password,
SaltSize, Iterations, HashAlgorithmName.SHA256);

var salt = algorithm.Salt;
var key = algorithm.GetBytes(KeySize);
```

Проверка пароля выполняется путем повторного вычисления хеша и его сравнения с сохраненным значением. Это видно в листинге 2.

Листинг 2 – Проверка пароля

```
using var algorithm = new Rfc2898DeriveBytes(password, salt,
Iterations, HashAlgorithmName.SHA256);

var keyToCheck = algorithm.GetBytes(KeySize);

return CryptographicOperations.FixedTimeEquals(keyToCheck,
key);
```

При входе система проверяет корректность пароля, статус блокировки пользователя, что продемонстрировано в листинге 3, и только затем выдает access token и refresh token. Access token используется для проверки доступа к защищенным маршрутам, а refresh token хранится в записи пользователя и применяется для продления сессии. Таким образом, требование о безопасном входе и поддержании пользовательской сессии было реализовано не формально, а в виде полноценного механизма авторизации.

Листинг 3 – Проверка данных при входе пользователя

```
if (user.IsBlocked)
    throw new UnauthorizedAccessException("Account is
blocked");
bool isValid = _passwordHasher.Verify(request.Password,
user.PasswordHash);
if (!isValid)
```

```
throw new UnauthorizedAccessException("Invalid email or password");
```

Требование о разграничении ролей также реализовано на сервере. В системе используется перечисление UserRole, в котором пользователь может одновременно обладать несколькими ролями. На уровне Web API настроены политики доступа, а в прикладных сервисах дополнительно выполняются точечные проверки прав на изменение конкретных сущностей.

Модуль управления цифровой идентичностью реализован в сервисах Users и Cars. В UserService описаны сценарии получения личного и публичного профиля, обновления биографии, установки аватара, поиска пользователей, назначения ролей и блокировки учетных записей. Тем самым выполняются требования о ведении профиля пилота и разграничении административных полномочий.

Алгоритм формирования пользовательского профиля основан на объединении данных из нескольких подсистем приложения и представлен в листинге 4. При запросе профиля система получает основную информацию о пользователе, список автомобилей, текущие заявки на турниры, историю участия в соревнованиях и перечень организуемых мероприятий. После этого данные объединяются в единый объект ответа, который отдается клиентскому приложению.

Листинг 4 – Формирование публичного профиля пользователя

```
var publicProfile = BuildPublicProfile(user);
var response = UserMapper.ToProfile(user);
response.CarsCount = publicProfile.CarsCount;
response.TournamentsCount = publicProfile.TournamentsCount;
response.Cars = publicProfile.Cars;
response.CurrentTournaments = publicProfile.CurrentTournaments;
response.TournamentHistory = publicProfile.TournamentHistory;
response.Applications = BuildApplications(user);
response.OrganizedTournaments = user.OrganizedTournaments;
```

Сервис CarService реализует виртуальный гараж пользователя. Пользователь может создавать автомобили, редактировать их, удалять и

прикреплять к ним фотографии. При этом серверная часть выполняет проверки, направленные на обеспечение целостности данных.

Требование о недопустимости удаления автомобиля, который уже используется в активных заявках, реализовано серверной проверкой и продемонстрировано в листинге 5. Это обеспечивает целостность данных и не позволяет разрушить связь между турниром и выбранным для участия автомобилем.

Листинг 5 – Проверка на наличие активных заявок при удалении автомобиля

```
bool hasActive = await
_carRepository.HasActiveApplicationsAsync(carId);
if (hasActive)
    throw new InvalidOperationException("Cannot delete car used
in active tournaments");
```

Модуль гоночных площадок реализован в сервисе TrackService. Здесь отражены требования, связанные с формированием базы трасс силами сообщества. Пользователь с достаточным стажем может создать новую трассу, которая по умолчанию получает статус Unofficial. Затем другие участники могут голосовать за ее подтверждение. После достижения установленного порога положительных голосов система автоматически переводит трассу в статус Confirmed. Алгоритм показан в листинге 6. Модератор, в свою очередь, может вручную подтвердить трассу или перевести ее в статус Official. Требования технического задания реализованы не как разовое добавление трассы, а как полный жизненный цикл гоночной площадки.

Листинг 6 – Автоматическое подтверждение трассы по результатам голосования

```
var positiveVotesCount = track.Votes.Count(v => v.IsPositive);
if (positiveVotesCount >= track.ConfirmationThreshold)
    track.Status = TrackStatus.Confirmed;

await _trackRepository.UpdateAsync(track);
```

Модуль организации мероприятий реализован в сервисе TournamentService и является центральным с точки зрения бизнес-логики системы. Здесь отражены требования по созданию турниров, управлению организаторским составом, обработке заявок, смене статусов турнира и фиксации результатов.

Алгоритм создания турнира включает последовательную проверку сервером корректности дат, числа требуемых участников, существования пользователя и трассы, а также статус трассы. Проверки при создании турнира продемонстрированы в листинге 7.

Листинг 7 – Проверки при создании турнира

```
if (request.StartDate >= request.EndDate)
    throw new ArgumentException("Tournament end date must be
after the start date.");
if (request.RequiredParticipants <= 0)
    throw new ArgumentException("Required participants must be
greater than zero.");
if (track.Status == TrackStatus.Unofficial)
    throw new InvalidOperationException("Tournament can be
created only on confirmed or official tracks.");
```

Если пользователь еще не обладает ролью организатора, система проверяет его гоночный стаж и может автоматически выдать роль Organizer, если установленный порог достигнут. Автоматическое присвоение роли показано в листинге 8.

Листинг 8 – Автоматическое присвоение роли

```
if (!isPrivileged(user))
{
    if (user.RaceCount < _settings.MinRacesToBecomeOrganizer)
        throw new ForbiddenException($"");

    user.Roles |= UserRole.Organizer;
}
```

Подача заявки на турнир также сопровождается прикладными проверками: пользователь не должен быть заблокирован, регистрация на турнир должна быть открыта, автомобиль должен принадлежать пользователю, его класс должен

соответствовать регламенту турнира, а стаж пользователя – требованиям мероприятия. Проверки продемонстрированы в листинге 9. Дополнительно система не допускает повторной подачи заявки на тот же турнир. Алгоритм подачи заявки отображен на рисунке 16.

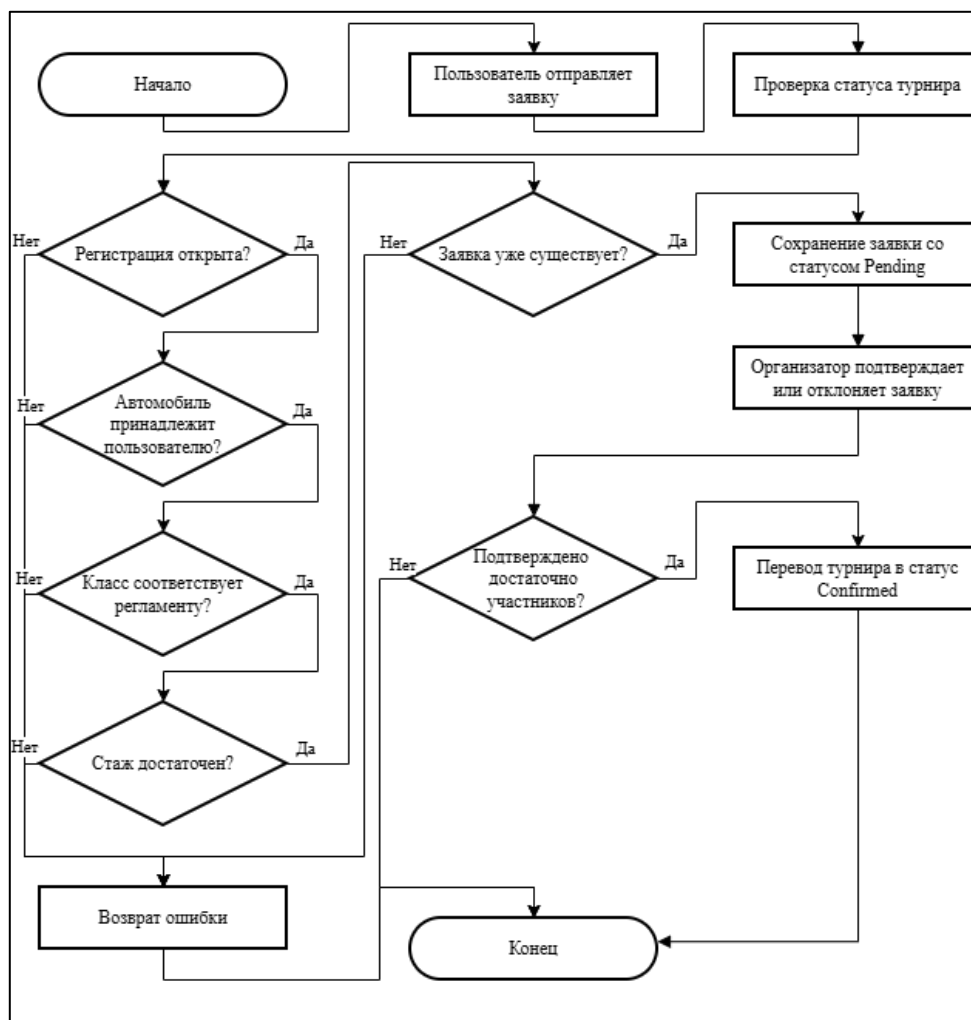


Рисунок 16 – Алгоритм работы заявок

Листинг 9 – Проверка условий подачи заявки на турнир

```

if (tournament.Status != TournamentStatus.RegistrationOpen)
    throw new InvalidOperationException("Registration closed");
if (car.OwnerId != userId)
    throw new ForbiddenException("Not your car");
if (car.CarClass != tournament.AllowedCarClass)
    throw new InvalidOperationException("Car class mismatch");
if (user.RaceCount < tournament.RequiredRaceCount)
    throw new InvalidOperationException("Not enough races");
    
```

После успешного прохождения проверок заявка сохраняется в базе данных и получает статус Pending. Организатор турнира может подтвердить или отклонить участие пилота. При достижении требуемого количества подтвержденных участников турнир автоматически переводится в состояние Confirmed. Это показано в листинге 10.

Листинг 10 – Автоматическое подтверждение турнира

```
if (approvedCount >= tournament.RequiredParticipants)
    tournament.Status = TournamentStatus.Confirmed;
```

После перевода турнира в активную фазу организатор получает возможность вносить результаты. Завершение турнира допускается только при наличии хотя бы одного результата. После завершения система автоматически увеличивает значение RaceCount для участников, что видно в листинге 11. Результаты соревнования становятся основой для дальнейшего стажа пилота и его доступа к новым действиям внутри платформы.

Листинг 11 – Автоматическое увеличение количества гонок в профилях участников

```
foreach (var result in tournament.Results)
    if (participant != null)
    {
        participant.RaceCount += 1;
        await _userRepository.UpdateAsync(participant);
    }
```

Требования к работе с медиаконтентом реализованы в модуле MediaFiles. Загрузка изображений выполняется через отдельный маршрут, а сервер проверяет корректность размера файла, MIME-типа и расширения. Код проверок показан в листинге 12. После физического сохранения изображения создается MediaFile, который затем может быть связан с пользователем, автомобилем, трассой или турниром. Важной частью логики является проверка владения файлом: пользователь не может прикрепить к своей сущности изображение, загруженное другим участником системы.

Листинг 12 – Валидация файлов

```
if (string.IsNullOrEmpty(options.UploadsPath))
    failures.Add("UploadsPath is required.");

if (string.IsNullOrEmpty(options.UploadsRequestPath))
    failures.Add("UploadsRequestPath is required.");
else if (!options.UploadsRequestPath.StartsWith('/'))
    failures.Add("UploadsRequestPath must start with '/'.");

if (options.MaxUploadSizeBytes <= 0)
    failures.Add("MaxUploadSizeBytes must be greater than
zero.");

if (options.PhotoAllowedExtensions.Length == 0)
    failures.Add("PhotoAllowedExtensions must contain at least
one extension.");
else if (options.PhotoAllowedExtensions.Any(ext =>
string.IsNullOrEmpty(ext) || !ext.StartsWith('.')))
    failures.Add("PhotoAllowedExtensions must contain
extensions in '.ext' format.");

if (options.PhotoAllowedContentTypes.Length == 0)
    failures.Add("PhotoAllowedContentTypes must contain at
least one content type.");
```

Таким образом, реализация серверной части показывает, что требования, сформулированные в техническом задании, были разложены на прикладные модули и сценарии. Каждый модуль отвечает за набор бизнес-правил, а вся логика в совокупности формирует работоспособную основу информационной системы.

1.3.2 Клиент

1.3.2.1 Проектирование архитектуры

Для клиентской части приложения, разрабатываемой с использованием библиотеки React, классические архитектурные паттерны в их традиционном виде применимы ограниченно, так как современный пользовательский интерфейс строится не вокруг иерархии классов, а вокруг композиции компонентов, локального состояния и декларативного описания представления. По этой причине был выбран архитектурный подход Feature-Sliced Design (FSD) [10].

FSD – это архитектурная методология для проектов, цель которой сделать проект понятным, структурированным и масштабируемым. Основная идея

подхода заключается в разделении кода одновременно по уровням ответственности и по предметным областям. Код клиентского приложения разделен на следующие слои (от высшего к низшему):

- app (Приложение) – слой инициализации приложения, включающий точку входа, глобальные стили, настройку маршрутизации, подключение провайдеров контекста и общую конфигурацию жизненного цикла приложения;

- pages (Страницы) – композиционный слой, в котором формируются полноценные пользовательские экраны. Здесь определяется структура страницы, объединяются виджеты, пользовательские сценарии и сущности, а также может размещаться состояние, относящееся к конкретному маршруту;

- widgets (Виджеты) – крупные самостоятельные блоки интерфейса, объединяющие несколько сущностей и пользовательских сценариев в единый визуальный модуль;

- features (Пользовательские сценарии) – части, реализующие конкретные действия пользователя – бизнес логика. Каждый сценарий инкапсулирует действия, необходимые для выполнения одной задачи, и может взаимодействовать с хранилищем данных и API;

- entities (Сущности) – представления основных объектов предметной области. Слой содержит компоненты и типизированные структуры, отвечающие за отображение пользователей, автомобилей, трасс и турниров без размещения сложной сценарной логики;

- shared (Общие компоненты) – базовый переиспользуемый слой, содержащий настроенный HTTP-клиент, универсальные UI-компоненты, утилиты форматирования, конфигурационные константы, вспомогательные функции и прочие общие элементы инфраструктуры клиента.

Главное правило FSD заключается в том, что модули одного слоя могут импортировать только модули слоев, расположенных строго ниже. Это означает, что страница может использовать виджеты, пользовательский сценарии, сущности и общие компоненты, но, например, сущность не должна зависеть от

страницы или виджета. Такое ограничение формирует направленный граф зависимостей, предотвращает появление циклических трудно отслеживаемых связей, упрощает сопровождение проекта. На рисунке 17 и приложении А.3 показаны связи между слоями клиентской части.

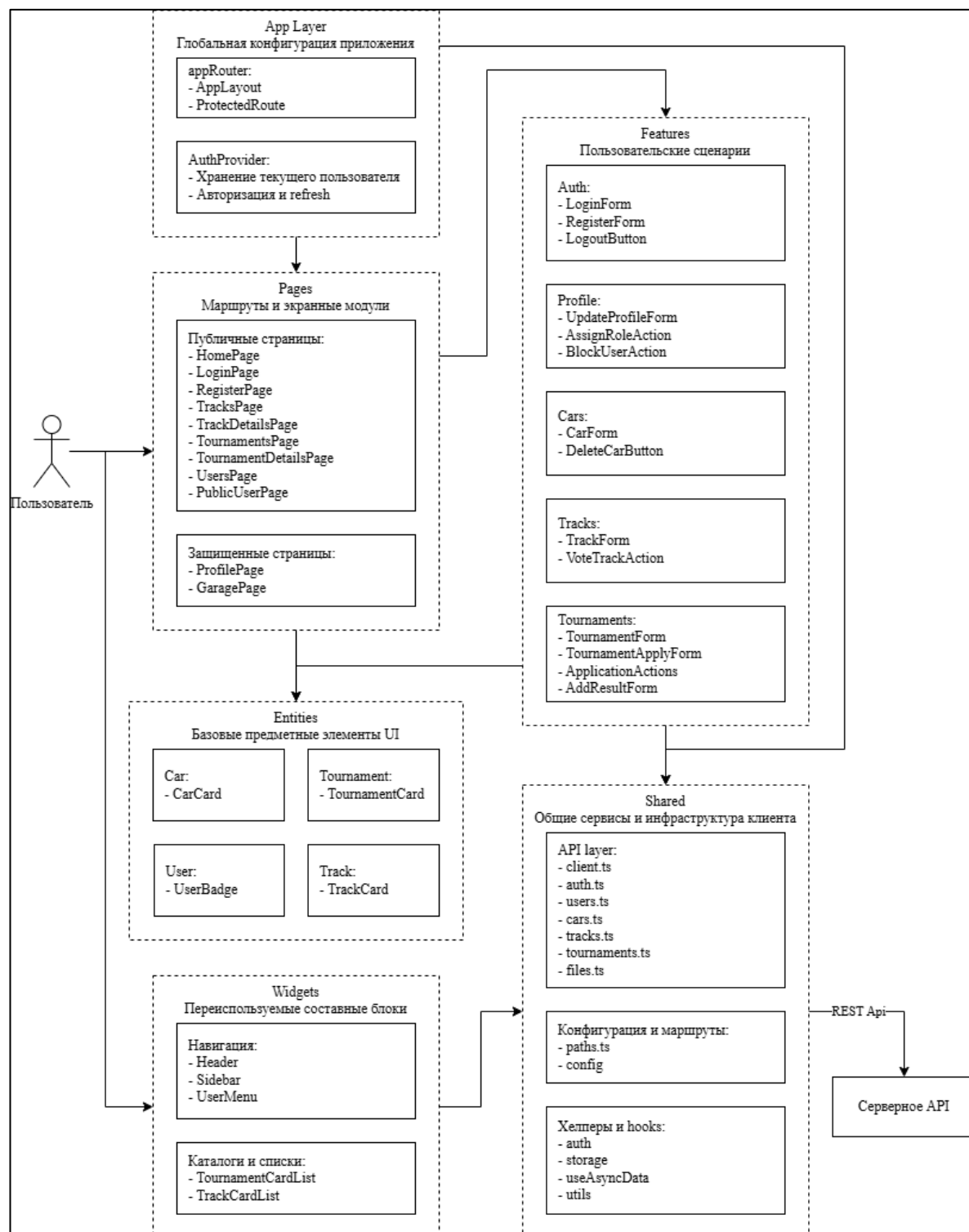


Рисунок 17 – Архитектура клиента

Маршрутизация реализована с помощью react-router-dom. В приложении выделены публичные маршруты, доступные всем пользователям, и защищенные маршруты, требующие авторизации. Защищенные страницы, такие как личный кабинет и гараж пользователя, оборачиваются в компонент ProtectedRoute. Конфигурация маршрутов клиентского приложения показана в листинге 13.

Листинг 13 – Конфигурация маршрутов клиентского приложения

```
path: PATHS.HOME,
element: <AppLayout />,
children: [
  {
    index: true, element: <HomePage />,
  },
  {
    path: PATHS.AUTH.LOGIN, element: <LoginPage />,
  },
  {
    path: PATHS.AUTH.REGISTER, element: <RegisterPage />,
  },
  {
    path: PATHS.TOURNAMENTS.ROOT, element: <TournamentsPage />,
  },
  {
    element: <ProtectedRoute />,
    children: [
      {
        path: PATHS.USERS.PROFILE, element: <ProfilePage />,
      },
      {
        path: PATHS.USERS.GARAGE, element: <GaragePage />,
      },
    ],
  },
],
```

1.3.2.2 Реализация бизнес-логики

Бизнес-правила на клиенте отражаются в виде пользовательских сценариев, экранов и функциональных модулей. Главная задача клиентской части состоит не только в отображении данных, но и в том, чтобы сделать реализованные на сервере правила понятными и управляемыми для пользователя.

Требования модуля управления доступом и безопасностью на клиенте реализованы через AuthProvider, формы входа и регистрации, а также механизм

защищенных маршрутов. Провайдер авторизации хранит сведения о текущем пользователе, признак авторизации, статус начальной загрузки сессии и операции login, register, logout и refreshCurrentUser. Это отображено в листинге 14. За счет этого состояние доступа становится глобальным и доступным для всех страниц приложения.

Листинг 14 – Глобальное состояние авторизации на клиенте

```
type AuthContextValue = {
  currentUser: UserProfileResponse | null;
  isAuthenticated: boolean;
  isBootstrapping: boolean;
  login: (request: LoginRequest) => Promise<void>;
  register: (request: RegisterRequest) => Promise<void>;
  logout: () => Promise<void>;
  refreshCurrentUser: () => Promise<UserProfileResponse>;
  setCurrentUser: (user: UserProfileResponse | null) => void;
};
```

После успешного входа пользователь получает access token и refresh token, которые сохраняются в localStorage. Далее клиент автоматически прикрепляет access token ко всем защищенным запросам, а при получении ответа 401 Unauthorized пытается обновить сессию через refresh token. Функция прикрепления access token показана в листинге 15.

Листинг 15 – Автоматическое добавление access token к каждому запросу

```
apiClient.interceptors.request.use((config) => {
  const accessToken = getStoredAccessToken();

  if (accessToken) {
    config.headers = config.headers ?? {};
    config.headers.Authorization = `Bearer ${accessToken}`;
  }

  return config;
});
```

Модуль цифровой идентичности пользователя на клиенте реализуется через страницы профиля, гаража и публичной карточки пользователя. В личном

кабинете пользователь видит роли, биографию, счетчик завершенных гонок, список заявок, историю участия и перечень организуемых турниров.

Управление гаражом реализовано на отдельной странице `GaragePage` и через пользовательские сценарии `create-car` и `delete-car`. Пользователь может добавить автомобиль, отредактировать его и загрузить фотографии. При этом ограничения на удаление, связанные с участием в активных турнирах, контролируются сервером, а клиент отображает результат операции и сообщения об ошибках.

Требования к модулю гоночных площадок реализованы на страницах списка трасс и детальной карточки трассы. Пользователь может просматривать существующие трассы, создавать новые, голосовать за пользовательские площадки и видеть текущий статус трассы. Для модератора в интерфейсе предусмотрены действия ручного подтверждения и перевода трассы в официальный статус.

Модуль организации мероприятий наиболее полно раскрывается в страницах списка турниров и детальной карточки турнира. Именно здесь пользователь видит параметры соревнования, требования к автомобилю и стажу, список организаторов, участников и результаты. Пользовательские сценарии `apply-to-tournament`, `manage-applications`, `add-result` и `create-tournament` реализуют ключевые действия бизнес-уровня: подачу заявки, подтверждение или отклонение участия, внесение результатов и редактирование турнира. В листинге 16 отображен пример загрузки нескольких источников данных.

Листинг 16 – Пример загрузки нескольких источников данных

```
const [tournament, carReference, tracks, users] = await
Promise.all([
  tournamentsApi.getById(id),
  carsApi.getReferenceData(),
  tracksApi.getAll({}, 0, 100),
  usersApi.getUsers('', 0, 100),
]);
```

Страница турнира объединяет сведения о самом событии, справочники классов автомобилей, список возможных соорганизаторов, набор заявок и результаты. Это отображено на рисунке 18.

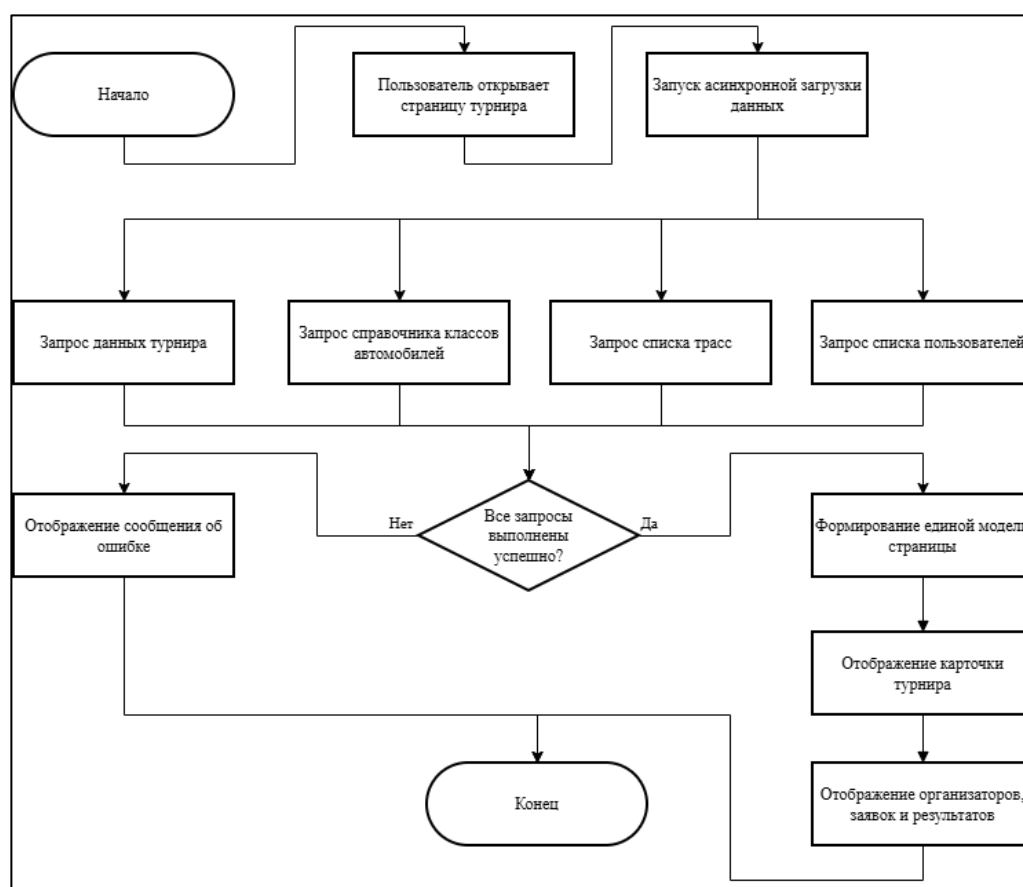


Рисунок 18 – Алгоритм загрузки и обработки данных на странице турнира

Для повторного использования сценария асинхронной загрузки данных в проекте создан хук `useAsyncData`. Он управляет состоянием загрузки, ошибками, результатом и повторным запросом данных. Использование такого хука позволяет унифицировать работу страниц с сервером, сократить дублирование однотипной логики и обеспечить единообразную обработку состояний интерфейса при обращении к API.

1.3.3 База данных

1.3.3.1 Модель данных

Реляционная модель базы данных разрабатывалась исходя из процессов, которые необходимо автоматизировать в любительском автоспорте. Главной

сущностью системы является пользователь. От него зависит большинства операций: регистрация, создание трасс, подача заявок, турниры и другое.

Графически структура базы данных изображена на рисунке 19.

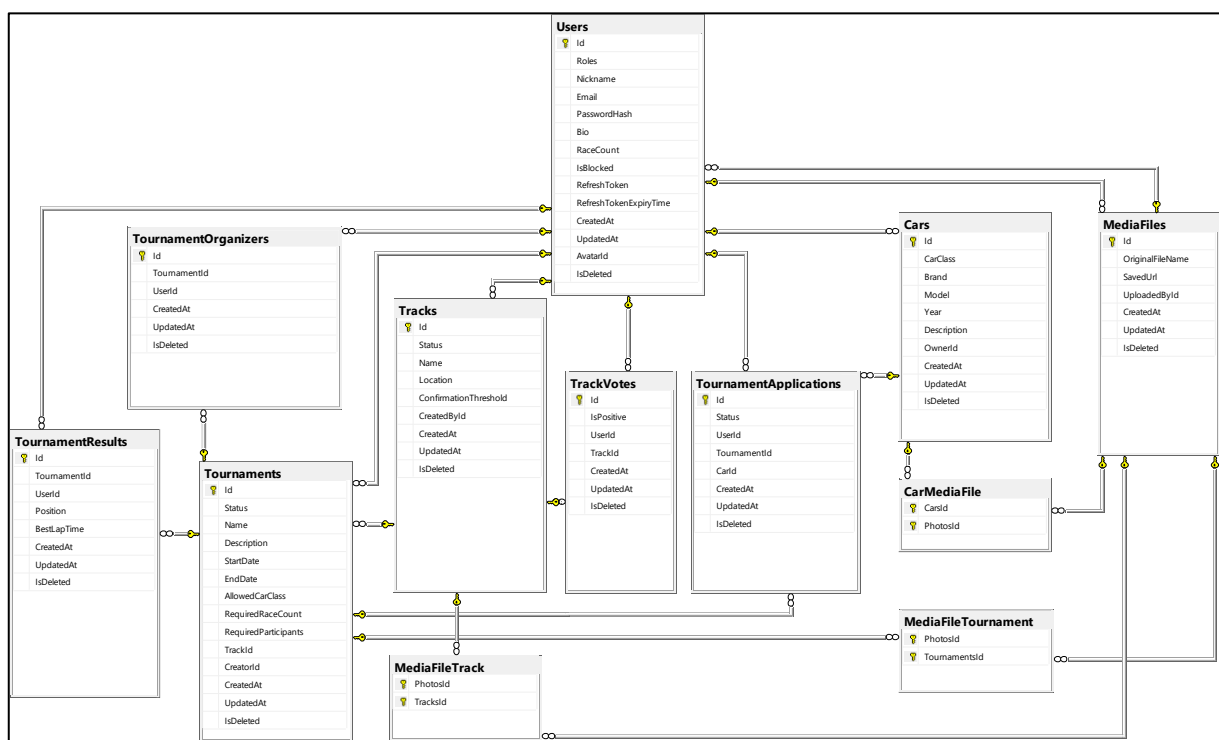


Рисунок 19 – Структура базы данных

Сущность User содержит базовые атрибуты и атрибуты безопасности. RaceCount и Roles отражают накопленный гоночный стаж, который используется при проверке в различных функциях, и набор ролевых полномочий, которые определяют возможность выполнения административных и организационных функций, соответственно, что делает пользователя активным участником предметной области, характеристики которого влияют на допустимые действия в системе. Поля представлены в таблице 1.

Таблица 1 – Users

Поле	Тип данных	Описание	Ключ
Id	uniqueidentifier	Уникальный идентификатор пользователя	РК

Поле	Тип данных	Описание	Ключ
CreatedAt	datetime2	Дата и время создания	
UpdatedAt	datetime2 NULL	Дата и время последнего изменения	
IsDeleted	bit	Признак логического удаления	
Roles	int	Роли пользователя	
Nickname	nvarchar(max)	Отображаемое имя пользователя	
Email	nvarchar(max)	Адрес электронной почты пользователя	UQ
PasswordHash	nvarchar(max)	Хеш пароля пользователя	
Bio	nvarchar(1000) NULL	Краткая информация о пользователе	
RaceCount	int	Количество завершенных гонок пользователя	
IsBlocked	bit	Признак блокировки учетной записи	
RefreshToken	nvarchar(max) NULL	Токен обновления сессии	
RefreshTokenExpiryTime	datetime2 NULL	Срок действия refresh token	
AvatarId	uniqueidentifier NULL	Ссылка на файл аватара	FK-> MediaFiles.Id

Сущность Car отражает персональный гараж пользователя. Содержит базовую информацию об автомобиле, представленную в таблице 2. Один пользователь может иметь несколько автомобилей.

Таблица 2 – Cars

Поле	Тип данных	Описание	Ключ
Id	uniqueidentifier	Уникальный идентификатор автомобиля	PK
CreatedAt	datetime2	Дата создания записи	
UpdatedAt	datetime2 NULL	Дата последнего изменения	
IsDeleted	bit	Признак логического удаления	
CarClass	int	Класс автомобиля	
Brand	nvarchar(max)	Марка автомобиля	
Model	nvarchar(max)	Модель автомобиля	

Поле	Тип данных	Описание	Ключ
Year	int	Год выпуска	
Description	nvarchar(max) NULL	Описание автомобиля	
OwnerId	uniqueidentifier	Владелец автомобиля	FK -> Users.Id

Сущность Track описывает гоночную площадку. В ней хранятся название, местоположение, статус трассы, порог подтверждения и фотографии. Это показано в таблице 3.

Таблица 3 – Tracks

Поле	Тип данных	Описание	Ключ
Id	uniqueidentifier	Уникальный идентификатор трассы	PK
CreatedAt	datetime2	Дата создания записи	
UpdatedAt	datetime2 NULL	Дата последнего изменения	
IsDeleted	bit	Признак логического удаления	
Status	int	Статус трассы	
Name	nvarchar(max)	Название трассы	
Location	nvarchar(max)	Местоположение трассы	
ConfirmationThreshold	int	Порог положительных голосов для подтверждения трассы	
CreatedById	uniqueidentifier	Пользователь, добавивший трассу	FK -> Users.Id

Сущность TrackVote реализует результат голосования пользователя за пользовательскую трассу. Она содержит ссылку на пользователя, ссылку на трассу и булев признак положительного или отрицательного голоса. Поля представлены в таблице 4.

Таблица 4 – TrackVotes

Поле	Тип данных	Описание	Ключ
Id	uniqueidentifier	Уникальный идентификатор голоса	PK
CreatedAt	datetime2	Дата создания записи	
UpdatedAt	datetime2	Дата последнего изменения	

Поле	Тип данных	Описание	Ключ
	NULL		
IsDeleted	bit	Признак логического удаления	
IsPositive	bit	Тип голоса: положительный или отрицательный	
UserId	uniqueidentifier	Пользователь, оставивший голос	FK -> Users.Id
TrackId	uniqueidentifier	Трасса, за которую оставлен голос	FK -> Tracks.Id

Tournament описывает турнир и содержит название, описание, границы, допустимый класс автомобиля, минимальный требуемый стаж, количество необходимых участников, статус, связь с трассой, создателем и организаторами. В таблице 5 описаны поля данной таблицы с типами данных.

Таблица 5 – Tournaments

Поле	Тип данных	Описание	Ключ
Id	uniqueidentifier	Уникальный идентификатор турнира	PK
CreatedAt	datetime2	Дата создания записи	
UpdatedAt	datetime2 NULL	Дата последнего изменения	
IsDeleted	bit	Признак логического удаления	
Status	int	Статус турнира	
Name	nvarchar(max)	Название турнира	
Description	nvarchar(max)	Описание турнира	
StartDate	datetime2	Дата и время начала турнира	
EndDate	datetime2	Дата и время завершения турнира	
AllowedCarClass	int	Допустимый класс автомобиля для участия	
RequiredRaceCount	int	Минимальный стаж участника	
RequiredParticipants	int	Требуемое количество подтвержденных участников	
TrackId	uniqueidentifier	Трасса проведения турнира	FK -> Tracks.Id
CreatorId	uniqueidentifier	Создатель турнира	FK -> Users.Id

Сущность TournamentApplication связывает пользователя, турнир и выбранный автомобиль. Это видно в таблице 6. В ней также хранится статус заявки: Pending, Approved, Rejected или Cancelled.

Таблица 6 – TournamentApplications

Поле	Тип данных	Описание	Ключ
Id	uniqueidentifier	Уникальный идентификатор заявки	РК
CreatedAt	datetime2	Дата создания записи	
UpdatedAt	datetime2 NULL	Дата последнего изменения	
IsDeleted	bit	Признак логического удаления	
Status	int	Статус заявки (Pending, Approved, Rejected, Cancelled)	
UserId	uniqueidentifier	Пользователь, подавший заявку	FK -> Users.Id
TournamentId	uniqueidentifier	Турнир, на который подана заявка	FK -> Tournaments.Id
CarId	uniqueidentifier	Автомобиль для участия	FK -> Cars.Id

Сущность TournamentOrganizer реализует связь многие-ко-многим между турниром и пользователем как организатором. Это показано в таблице 7.

Таблица 7 – TournamentOrganizers

Поле	Тип данных	Описание	Ключ
Id	uniqueidentifier	Уникальный идентификатор связи	РК
CreatedAt	datetime2	Дата создания записи	
UpdatedAt	datetime2 NULL	Дата последнего изменения	
IsDeleted	bit	Признак логического удаления	
TournamentId	uniqueidentifier	Турнир	FK -> Tournaments.Id
UserId	uniqueidentifier	Пользователь-организатор	FK -> Users.Id

Сущность TournamentResult предназначена для хранения результатов турнира. В таблице 5 описаны поля с типами данных.

Таблица 8 – TournamentResults

Поле	Тип данных	Описание	Ключ
Id	uniqueidentifier	Уникальный идентификатор	РК

Поле	Тип данных	Описание	Ключ
CreatedAt	datetime2	Дата создания записи	
UpdatedAt	datetime2 NULL	Дата последнего изменения	
IsDeleted	bit	Признак логического удаления	
TournamentId	uniqueidentifier	Турнир, где зафиксирован результат	FK -> Tournaments.Id
UserId	uniqueidentifier	Пользователь-участник	FK -> Users.Id
Position	int	Итоговая позиция участника	
BestLapTime	time NULL	Лучшее время круга	

Сущность MediaFile описывает загруженные пользователями изображения. Она хранит идентификатор, исходное имя файла, публичный URL и пользователя-владельца. Поля представлены в таблице 9.

Таблица 9 – MediaFiles

Поле	Тип данных	Описание	Ключ
Id	uniqueidentifier	Уникальный идентификатор медиафайла	РК
CreatedAt	datetime2	Дата создания записи	
UpdatedAt	datetime2 NULL	Дата последнего изменения	
IsDeleted	bit	Признак логического удаления	
OriginalFileName	nvarchar(max)	Исходное имя загруженного файла	
SavedUrl	nvarchar(max)	Публичный URL или путь доступа к файлу	
UploadedById	uniqueidentifier NULL	Пользователь, загрузивший файл	FK -> Users.Id

В проекте используются также промежуточные таблицы связей между медиафайлами и сущностями. На уровне модели Entity Framework Core они формируются автоматически как таблицы CarMediaFile, MediaFileTrack и MediaFileTournament.

Все основные сущности наследуются от базового класса GuidEntity. Он содержит идентификатор GUID, дату создания, дату последнего обновления и признак логического удаления IsDeleted. Пример предметной сущности приведен в листинге 17. Такое решение делает структуру модели единообразной и позволяет централизованно поддерживать soft-delete.

Листинг 17 – Пример предметной сущности автомобиля

```
public CarClass CarClass { get; set; }
public required string Brand { get; set; }
public required string Model { get; set; }
public int Year { get; set; }
public string? Description { get; set; }
// Navigation
public Guid OwnerId { get; set; }
public User Owner { get; set; } = null!;
public ICollection<MediaFile> Photos { get; set; } = [];
public ICollection<TournamentApplication> Applications { get;
set; } = [];
```

1.3.3.2 Доступ к данным

В качестве СУБД в проекте выбрана связка Microsoft SQL Server и Entity Framework Core.

Основной точкой взаимодействия приложения с базой является класс `AppDbContext`, который частично приведен в листинге 18. В нем объявлены наборы `DbSet` для пользователей, автомобилей, трасс, голосов, турниров, заявок, результатов, организаторов и медиафайлов. При инициализации приложения контекст подключается через внедрение зависимостей и получает строку подключения из конфигурации.

Листинг 18 – Описание DbSet в AppDbContext

```
public DbSet<User> Users => Set<User>();
public DbSet<Car> Cars => Set<Car>();
public DbSet<Track> Tracks => Set<Track>();
public DbSet<TrackVote> TrackVotes => Set<TrackVote>();
public DbSet<Tournament> Tournaments => Set<Tournament>();
public DbSet<TournamentApplication> TournamentApplications =>
Set<TournamentApplication>();
public DbSet<TournamentResult> TournamentResults =>
Set<TournamentResult>();
public DbSet<TournamentOrganizer> TournamentOrganizers =>
Set<TournamentOrganizer>();
public DbSet<MediaFile> MediaFiles => Set<MediaFile>();
```

Для конфигурирования сущностей используются отдельные классы, реализующие интерфейс `IEntityTypeConfiguration`. Пример функции

конфигурации показан в листинге 19. Это позволяет не перегружать ApplicationDbContext и поддерживать правила для каждой сущности отдельно.

Листинг 19 – Пример конфигурации автомобилей

```
builder.HasOne(c => c.Owner)
    .WithMany(u => u.Cars)
    .HasForeignKey(c => c.OwnerId)
    .onDelete(DeleteBehavior.Cascade);
builder.HasQueryFilter(e => !e.IsDeleted);
```

На уровне доступа к данным в проекте применен паттерн Repository. Для основных предметных областей реализованы UserRepository, CarRepository, TrackRepository, TournamentRepository, TournamentApplicationRepository и FileRepository. Это позволяет отделить прикладные сервисы от деталей запросов к Entity Framework Core. Репозитории инкапсулируют операции доступа к данным: получение по идентификатору, поиск, пагинацию, обновление, удаление и выполнение специализированных выборок.

Для списков пользователей, трасс и турниров реализована постраничная выборка. Это важно при потенциальном росте объема данных, так как интерфейс не должен загружать весь массив записей целиком. Постраничные выборки возвращают список элементов и общее количество записей.

Вместо физического удаления записей из таблиц метод SaveChangesAsync переопределен так, чтобы при удалении сущности она переводилась в состояние Modified, получала IsDeleted true и обновленную дату изменения. Для сущностей задаются глобальные фильтры, которые автоматически скрывают такие записи из обычных выборок. Реализация алгоритма приведена в листинге 19.

Листинг 19 – Реализация логического удаления в ApplicationDbContext

```
case EntityState.Added:
    entityEntry.Entity.CreatedAt = DateTime.UtcNow;
    entityEntry.Entity.UpdatedAt = DateTime.UtcNow;
    entityEntry.Entity.IsDeleted = false;
    break;
case EntityState.Modified:
    entityEntry.Entity.UpdatedAt = DateTime.UtcNow;
```

```

        break;
    case EntityState.Deleted:
        entityEntry.State = EntityState.Modified;
        entityEntry.Entity.IsDeleted = true;
        entityEntry.Entity.UpdatedAt = DateTime.UtcNow;
        break;

```

1.3.3.3 Миграции и инициализация

Для управления структурой базы данных в проекте используются миграции Entity Framework Core.

При запуске приложения применяется класс `DbInitializer`. Он выполняет две задачи. Сначала через `Database.MigrateAsync` обеспечивает актуальность схемы базы данных. Затем при необходимости запускает механизм начального заполнения. Данные этапы показаны в листинге 20.

Листинг 20 – Автоматическое применение миграций и заполнение базы

```

var services = scope.ServiceProvider;
var initializer = services.GetRequiredService<DbInitializer>();
await initializer.InitializeAsync();
await initializer.SeedAsync();

```

Механизм заполнения выполняет заполнение базы начальными данными. В обязательном порядке создается административная учетная запись (пароль берется из переменной окружения). При локальной разработке база заполняется стартовыми данными. Такой набор данных делает приложение пригодным для демонстрации практически сразу после запуска.

1.3.4 Docker и развертывание

1.3.4.1 Контейнеризация

Для обеспечения воспроизводимости среды выполнения и упрощения развертывания в проекте используется контейнеризация Docker. Отдельные образы создаются для серверной и клиентской частей приложения. Такой подход позволяет независимо собирать и запускать backend и frontend.

Серверная часть собирается с помощью файла `Dockerfile.api`. Используется многоэтапная сборка. На первом этапе применяется образ `mcr.microsoft.com/dotnet/sdk:8.0`, внутри которого копируются файлы решения и

проектов, выполняется восстановление зависимостей, а затем публикация приложения. На втором этапе используется более легкий runtime-образ mcr.microsoft.com/dotnet/aspnet:8.0. В него копируются только опубликованные артефакты приложения, что уменьшает размер конечного образа и исключает наличие в контейнере ненужных средств сборки.

Клиентская часть собирается через Dockerfile.frontend. На первом этапе используется образ Node.js, в котором устанавливаются зависимости через npm ci и формируется production-сборка приложения Vite. После этого готовые статические файлы копируются в контейнер на базе Nginx. Nginx выбран в качестве веб-сервера для раздачи фронтенда, поскольку он легковесен, хорошо подходит для публикации статических ресурсов и позволяет корректно настроить поведение SPA. Диаграмма разворачивания системы изображена на рисунке 20 и приложении А.4.

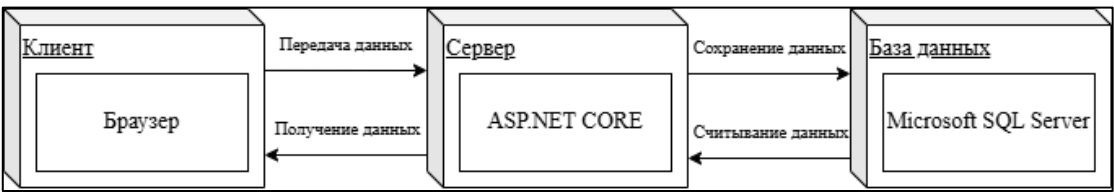


Рисунок 20 – Диаграмма разворачивания

1.3.4.2 Конфигурация окружения

Для гибкой настройки приложения используется внешняя конфигурация среды. В корне проекта присутствуют файлы .env.example и .env.dev. На их основе формируется рабочий файл .env, не изменяя исходный код приложения.

В .env-файле задаются параметры серверной и клиентской частей: внешний и внутренний порт API, окружение, параметры базы данных, секрет JWT, параметры хранения файлов, пароль администратора и адрес API для фронтенда. Пример такого файла приведен в листинге 21.

Листинг 21 – Пример параметров окружения

```
API_PORT=5000
API_INTERNAL_PORT=8080
```

```

ASPNETCORE_ENVIRONMENT=Development
DB_HOST=db
DB_PORT=1433
DB_NAME=MotorsportDb
DB_USER=sa
DB_PASSWORD=Strong_Password_12345!
JwtSettings__Secret=DEV_SUPER_SECRET_KEY
UPLOAD_PATH=/app/wwwroot/uploads
SeedSettings__AdminPassword=Admin_12345!
WEB_PORT=3000
WEB_INTERNAL_PORT=80
VITE_API_URL=http://localhost:5000/api

```

На стороне ASP.NET Core переменные окружения сопоставляются с конфигурационными секциями приложения. Благодаря принятому в .NET соглашению двойное подчеркивание позволяет обращаться к вложенным параметрам.

На стороне клиента переменная VITE_API_URL внедряется на этапе сборки и затем используется в shared/config/env.ts для настройки базового адреса API.

Вынос конфигурации во внешнюю среду важен с точки зрения безопасности. Секретные значения не хранятся в коде. Кроме того, это улучшает сопровождение проекта: изменение порта, строки подключения или JWT-ключа не требует пересборки.

1.3.4.3 Оркестрация сервисов

Для запуска системы как единого комплекса используется файл docker-compose.yml. Состав сервисов из него приведен в листинге 22. В нем описаны три сервиса: db, api и frontend. Сервис db поднимает контейнер Microsoft SQL Server 2022. Сервис api запускает серверную часть ASP.NET Core. Сервис frontend разворачивает собранный React-клиент в Nginx-контейнере. Связь между сервисами организована через зависимости и переменные окружения. API зависит от базы данных и запускается после того, как сервис SQL Server пройдет проверку готовности. Frontend зависит от старта API.

Листинг 22 – Состав сервисов в Docker Compose

```

services:
  frontend:

```

```

    build:
      context: .
      dockerfile: infra/docker/Dockerfile.frontend
  api:
    build:
      context: .
      dockerfile: infra/docker/Dockerfile.api
  db:
    image: mcr.microsoft.com/mssql/server:2022-latest

```

Для сохранения данных используются `sql_data` и `upload_data`. Первый сохраняет данные базы между перезапусками контейнеров, второй используется для хранения загруженных изображений, поэтому, даже при пересоздании контейнера API пользовательские фотографии и содержимое БД не будут потеряны.

Наряду с полным запуском предусмотрены облегченные режимы. Например, можно поднять только базу данных, если сервер и клиент запускаются локально из среды разработки, или поднять только API и БД, если фронтенд разрабатывается отдельно.

Вывод по проектированию и разработке

В данной главе были подробно рассмотрены решения, принятые при проектировании и разработке информационной системы Motorsport ERP. Показано, что серверная часть реализована как многослойное приложение на базе Clean Architecture, где бизнес-логика изолирована от деталей хранения данных и публикации API. Внутри прикладного слоя система разделена на самостоятельные функциональные модули, отвечающие за авторизацию, пользователей, автомобили, трассы, турниры и медиафайлы.

Клиентская часть построена на React и TypeScript по методологии Feature-Sliced Design, что обеспечило логичное разделение интерфейсного кода. Рассмотрены механизмы асинхронного взаимодействия с сервером, автоматического обновления сессии, работы с глобальным и локальным состоянием.

Таким образом, проектирование и разработка системы выполнены как единый инженерный процесс, в котором архитектурные решения, прикладная

логика, модель данных и инфраструктура развертывания взаимно согласованы и обеспечивают создание целостной цифровой платформы для организации и проведения любительских автоспортивных мероприятий.

1.4 Тестирование

Функциональное тестирование разработанной системы проводилось методом ручной проверки ключевых пользовательских сценариев, связанных с бизнес-требованиями. В ходе тестирования проверялись операции регистрации и аутентификации, ведения пользовательского профиля, управления гаражом, работы с трассами, организации турниров, обработки заявок, фиксации результатов и загрузки медиафайлов. Для каждого тест-кейса были зафиксированы выполняемые шаги, используемые данные, ожидаемый результат и фактический результат выполнения. Это позволяет подтвердить корректность реализации основных бизнес-процессов системы.

1.4.1 Проверка регистрации и авторизации пользователя

Данный тест-кейс предназначен для проверки базового сценария работы модуля управления доступом и безопасностью. В рамках проверки оценивается корректность создания учетной записи, входа в систему и доступа к защищенным разделам после успешной аутентификации. Тест-кейс описан в таблице 10.

Таблица 10 – Тест-кейс регистрации и авторизации пользователя

Шаг	Действие	Тестовые данные	Ожидаемый результат	Фактический результат
1	Открыть страницу регистрации		Отображается форма регистрации	Форма регистрации открывается
2	Заполнить и отправить форму регистрации	Nickname: TestPilot01; E-mail: testpilot01@example.com; Password: Qwerty123!	Создается учетная запись, пользователь перенаправляется в личный кабинет	Учетная запись создана, переход в личный кабинет
3	Выйти из системы		Сессия завершается, пользователь возвращается в публичную часть	Выход выполнен, защищенные разделы скрыты

Шаг	Действие	Тестовые данные	Ожидаемый результат	Фактический результат
4	Открыть страницу входа и авторизоваться	E-mail: testpilot01@example.com; Password: Qwerty123!	Пользователь успешно входит в систему	Авторизация выполнена успешно
5	Перейти в защищенный раздел профиля		Страница доступна авторизованному пользователю	Профиль открывается без ошибок

1.4.2 Проверка ведения профиля и управления гаражом

Тест-кейс направлен на проверку модуля цифровой идентичности. Проверяется функционал редактирования персональных данных, добавления автомобиля в гараж, загрузки изображения и соблюдения ограничения на удаление автомобиля. Тест-кейс описан в таблице 11.

Таблица 11 – Тест-кейс ведения профиля и управления гаражом

Шаг	Действие	Тестовые данные	Ожидаемый результат	Фактический результат
1	Обновить профиль пользователя	Nickname: TestPilot01; Bio: Любитель дрифта; Avatar: avatar01.png	Изменения сохраняются и видны в личном кабинете	Данные профиля обновлены и корректны
2	Добавить новый автомобиль в гараж	Brand: Toyota; Model: GT86; Year: 2018; Class: Street!	Автомобиль сохраняется и появляется в списке гаража	Автомобиль добавлен и отображается в гараже
3	Загрузить фотографию автомобиля	Файл: gt86.png	Изображение привязывается к карточке автомобиля	Фотография успешно отображается в карточке
4	Попытаться удалить автомобиль, заявленный на турнир	Автомобиль Toyota GT86; Активная заявка на турнир Spring Drift Cup	Система запрещает удаление и сохраняет связанный автомобиль	Удаление отклонено, автомобиль остается в системе

1.4.3 Проверка добавления и подтверждения трассы

Данный тест-кейс проверяет реализацию модуля гоночных площадок. Проверяется создание новой трассы, изменение ее статуса в результате

общественного голосования и выполнение административного подтверждения.
Тест-кейс описан в таблице 12.

Таблица 12 – Тест-кейс добавления и подтверждения трассы

Шаг	Действие	Тестовые данные	Ожидаемый результат	Фактический результат
1	Авторизоваться под пользователем с достаточным гоночным стажем	RaceCount больше или равно минимального порога для добавления трассы	Пользователю доступна форма добавления трассы	Форма создания трассы доступна
2	Создать новую трассу	Name: Industrial Ring; Location: Тула; Photo: track01.jpg	Трасса создается в статусе «Неофициальная»	Трасса создана со статусом «Неофициальная»
3	Проголосовать за трассу другим пользователем	Голос: положительный	Счетчик голосов увеличивается	Голос принят, счетчик обновлен
4	Подтвердить трассу вручную модератором	Роль: Moderator	Статус трассы изменяется на «Подтвержденная»	Статус изменен на «Подтвержденная»
5	Присвоить трассе официальный статус	Роль: Moderator	Трасса получает статус «Официальная»	Трасса переведена в официальный статус

1.4.4 Проверка создания турнира и подачи заявки на участие

Тест-кейс проверяет основной сценарий работы модуля организации мероприятий. Проверяется создание турнира пользователем с правами организатора и подача заявки пилотом с автомобилем подходящего класса. Тест-кейс описан в таблице 13.

Таблица 13 – Тест-кейс создания турнира и подачи заявки

Шаг	Действие	Тестовые данные	Ожидаемый результат	Фактический результат
1	Авторизоваться под организатором	Роль: Organizer	Пользователю доступна форма создания турнира	Форма создания турнира отображается корректно

Шаг	Действие	Тестовые данные	Ожидаемый результат	Фактический результат
2	Создать турнир	Name: Spring Drift Cup; Track: Industrial Ring; AllowedClass: Street; RequiredRaceCount: 3; RequiredParticipants: 8	Турнир сохраняется и появляется в общем списке	Турнир успешно создан и отображается в каталоге
3	Открыть турнир под учетной записью пилота	Пользователь TestPilot01; RaceCount: 5	Детальная страница турнира открывается, форма заявки доступна	Страница турнира открыта, форма заявки отображается
4	Подать заявку на участие	Автомобиль: Toyota GT86 Class: Street	Заявка создается со статусом ожидания	Заявка создана со статусом Pending

1.4.5 Проверка обработки заявок и фиксации результатов

В данном тест-кейсе проверяются функции организатора по обработке заявок, переводу турнира между состояниями и сохранению результатов заезда. Дополнительно проверяется обновление гоночного стажа пользователя. Тест-кейс описан в таблице 14.

Таблица 14 – Тест-кейс обработки заявок и фиксации результатов

Шаг	Действие	Тестовые данные	Ожидаемый результат	Фактический результат
1	Открыть страницу турнира под учетной записью организатора	Турнир: Spring Drift Cup	Организатору доступен блок управления заявками	Блок управления заявками отображается
2	Подтвердить заявку участника	Заявка TestPilot01	Статус заявки меняется на Approved, пользователь попадает в список участников	Заявка подтверждена, участник отображается в списке
3	Перевести турнир в активное состояние		Статус турнира изменяется на Active	Турнир успешно переведен в статус Active
4	Добавить результат участника	User: TestPilot01; Position: 1; BestLapTime: 00:01:45	Результат сохраняется в карточке турнира	Результат добавлен
5	Завершить турнир		Статус турнира меняется на Finished	Турнир успешно завершен

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.03.001 ПЗ

Лист

61

Шаг	Действие	Тестовые данные	Ожидаемый результат	Фактический результат
6	Проверить обновление гоночного стажа в профиле участника	Профиль TestPilot01	Счетчик завершенных гонок увеличивается	Значение RaceCount увеличено на 1

1.4.6 Проверка загрузки и привязки медиафайлов

Тест-кейс предназначен для проверки модуля управления медиаконтентом. Проверяется загрузка изображений, их привязка к различным сущностям системы и корректное отображение в клиентском интерфейсе. Тест-кейс описан в таблице 15.

Таблица 15 – Тест-кейс загрузки и привязки медиафайлов

Шаг	Действие	Тестовые данные	Ожидаемый результат	Фактический результат
1	Загрузить аватар пользователя	Файл: avatar01.png	Изображение сохраняется и отображается в профиле	Аватар загружен и отображается в личном кабинете
2	Загрузить фото автомобиля	Автомобиль: Toyota GT86; Файл: gt86.webp	Изображение добавляется в карточку автомобиля	Фото автомобиля отображается в гараже
3	Загрузить фото трассы	Трасса: Industrial Ring; Файл: track01.jpg	Изображение появляется в карточке трассы	Фото трассы успешно отображается в карточке
4	Загрузить фото турнира	Турнир: Spring Drift Cup; Файл: event_banner.jpg	Изображение появляется на детальной странице турнира	Фото турнира добавлено и доступно на странице события
5	Удалить собственный файл		Файл удаляется без нарушения связей других сущностей	Удаление выполнено корректно

Вывод по тестированию

Проведенное функциональное тестирование показало, что основные бизнес-процессы системы реализованы корректно. Во всех рассмотренных сценариях

фактические результаты соответствовали ожидаемым. Информационная система обеспечивает создание и аутентификацию пользователей, ведение цифрового профиля гонщика, управление гаражом, работу с трассами, организацию турниров, обработку заявок, фиксацию гоночных результатов и загрузку медиаконтента.

Таким образом, результаты тестирования подтверждают, что разработанное приложение выполняет требования, сформулированные в техническом задании, и может использоваться для автоматизации ключевых процессов организации любительских автоспортивных мероприятий.

1.5 Руководство пользователя

1.5.1 Главная страница

Разработанная информационная система MotorsportERP предназначена для цифровой поддержки любительских автоспортивных мероприятий. В рамках одной веб-платформы пользователь получает доступ к учетной записи пилота, собственному профилю, виртуальному гаражу, каталогу трасс, списку турниров, заявкам на участие и истории результатов.

После открытия приложения пользователь попадает на главную страницу. На ней отображаются ключевые разделы системы, краткое описание платформы, а также обзор ближайших турниров и доступных трасс. Она выполняет навигационную функцию, поскольку объединяет основные точки входа в пользовательские и информационные разделы приложения. За счет этого пользователь может получить общее представление о возможностях системы без предварительного перехода в личный кабинет или административные разделы. В верхней части располагается шапка сайта. В ней содержатся название платформы и меню. Для неавторизованного посетителя в правой части шапки доступны кнопки перехода к авторизации и регистрации. После входа в систему вместо них отображается мини-карточка пользователя с переходом в личный кабинет и кнопкой выхода. Внешний вид главной страницы изображен на рисунке 21.

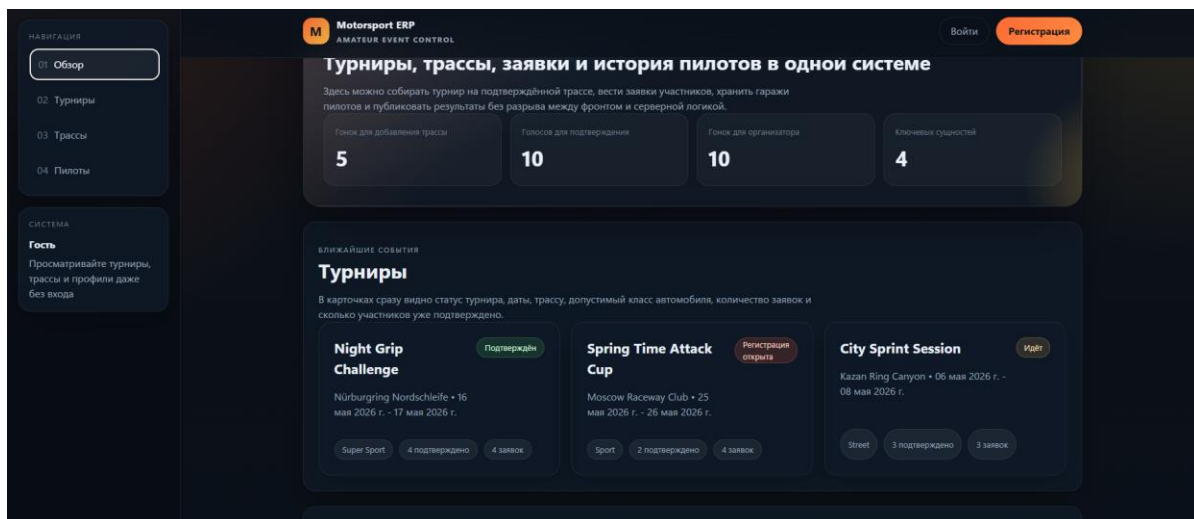


Рисунок 21 – Главная страница приложения MotorsportERP

1.5.2 Регистрация, вход в систему и разграничение прав доступа

Для регистрации новый пользователь открывает страницу регистрации и заполняет три обязательных поля: никнейм, адрес электронной почты и пароль. Внешний вид страницы показан на рисунке 22. После отправки формы сервер создает нового пользователя, присваивает ему базовую роль пилота и выполняет автоматический вход в систему. После успешной регистрации пользователь перенаправляется в личный кабинет, где может приступить к заполнению профиля.

Рисунок 22 – Форма регистрации пользователя

Если учетная запись уже существует, пользователь открывает страницу авторизации и вводит адрес электронной почты и пароль. В случае успешной

проверки данных система сохраняет авторизационную сессию и открывает ранее запрошенную страницу либо главную страницу приложения. Если данные введены неверно, пользователь получает сообщение об ошибке и может повторить попытку входа. Форма входа пользователя приведена на рисунке 23.

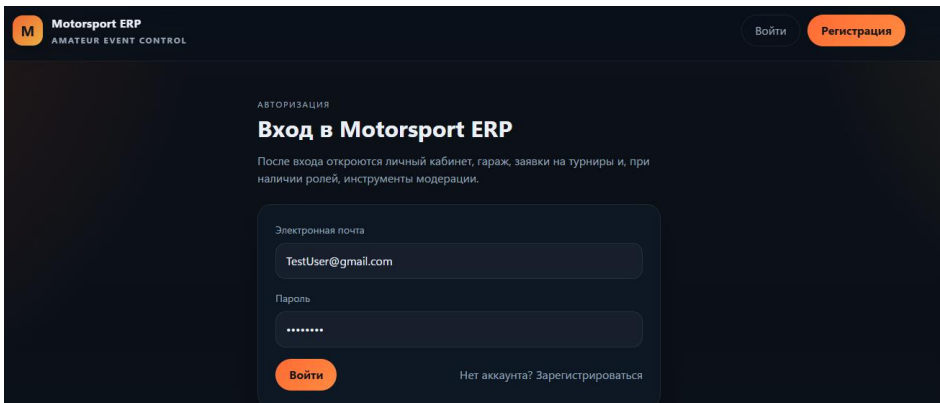


Рисунок 23 – Форма входа пользователя

Авторизованные пользователи не видят административных инструментов пока не назначены соответствующие роли. Каталог пользователей без них изображен на рисунке 24. Например, обычный пользователь может редактировать только собственный профиль и автомобили, тогда как модератор получает дополнительные кнопки ручного подтверждения трасс и доступ к административной панели пользователей. Суперпользователь, в свою очередь, может не только блокировать учетные записи, но и назначать новые роли. Панель управления для администраторов изображена на рисунке 25.

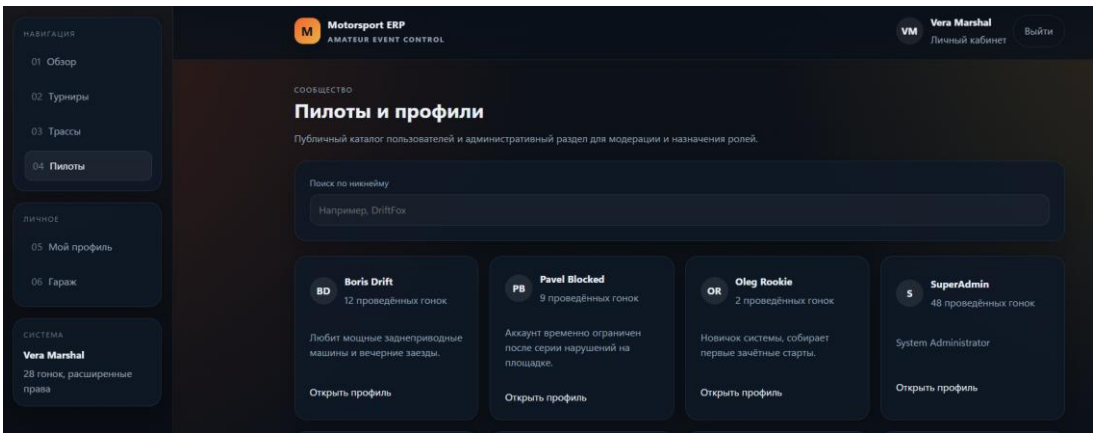


Рисунок 24 – Каталог пользователей

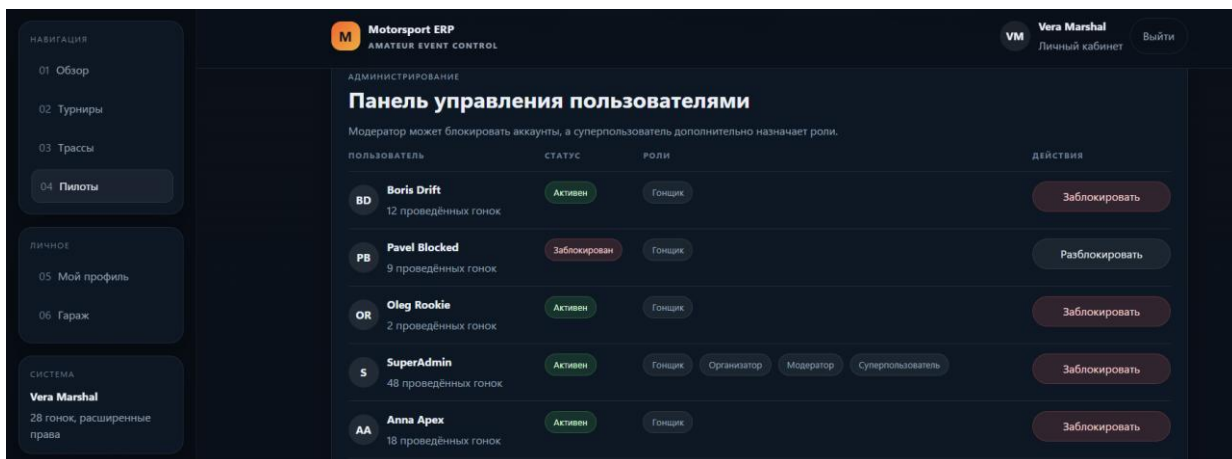


Рисунок 25 – Административная панель управления ролями

1.5.3 Работа с личным профилем и цифровой идентичностью пилота

После регистрации пользователь попадает в личный кабинет. Он изображен на рисунке 26. Этот раздел реализует требования модуля управления цифровой идентичностью и выступает основным местом хранения персональной информации о пилоте. В верхней части страницы отображаются никнейм, роли, текущий статус учетной записи и ключевые числовые показатели: количество завершенных гонок, число автомобилей и количество связанных турниров, что показано на рисунке 27.

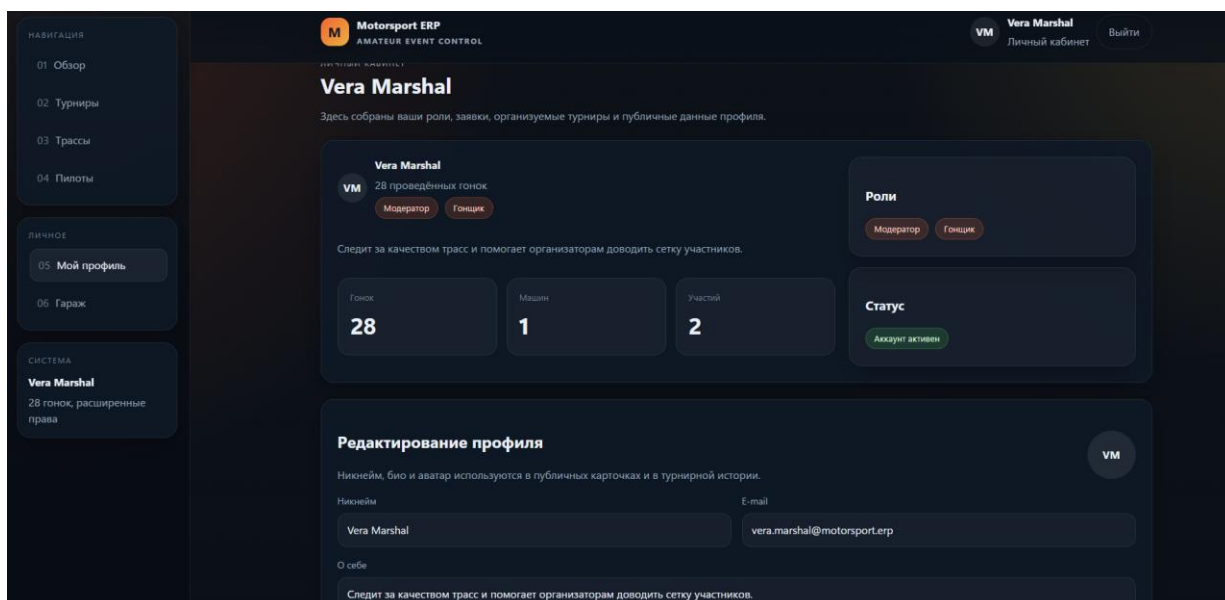


Рисунок 26 – Личный кабинет пользователя

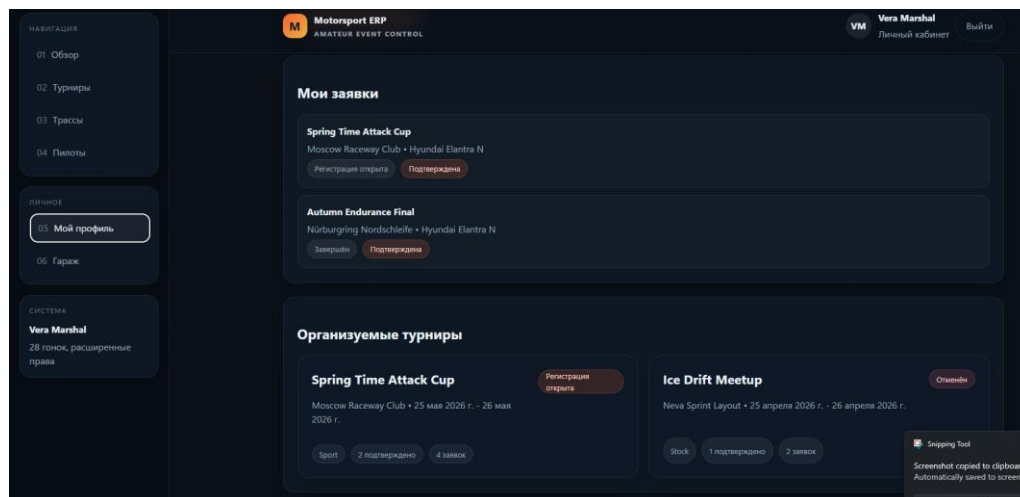


Рисунок 27 – Информация о турнирах и заявках

Ниже располагается форма редактирования профиля. С ее помощью пользователь может изменить текущую информацию о себе. Данные, введенные в этой форме, используются в публичных карточках пользователя, списках участников и истории турниров. За счет этого система формирует единый цифровой профиль пилота, который используется во всех связанных разделах приложения. Форма редактирования профиля изображена на рисунке 28.

Рисунок 28 – Форма редактирования профиля

1.5.4 Управление виртуальным гаражом

Раздел «Гараж» предназначен для добавления транспортных средств, которые затем используются при подаче заявок на турниры. Пример на сайте изображен на рисунке 29. Для каждого автомобиля пользователь указывает марку, модель, год выпуска и технический класс. После сохранения запись появляется в

виде отдельной карточки, которую можно редактировать, дополнять изображениями и, при необходимости, удалять.

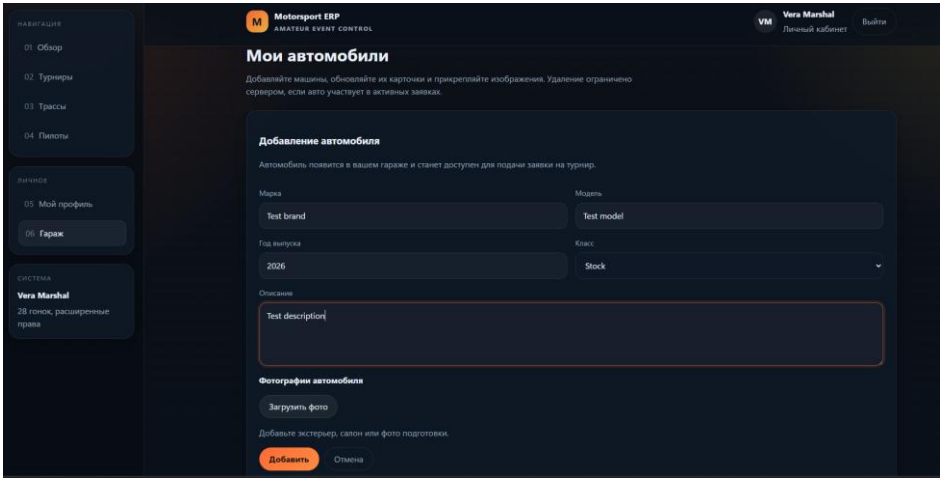


Рисунок 29 – Добавление автомобиля

Гараж позволяет хранить сведения о техническом парке пилота. И обеспечивает последующую автоматическую проверку соответствия автомобиля требованиям турнира. Именно поэтому при подаче заявки пользователь не вводит сведения об автомобиле заново, а выбирает одну из заранее подготовленных карточек. Пример гаража изображен на рисунке 30.

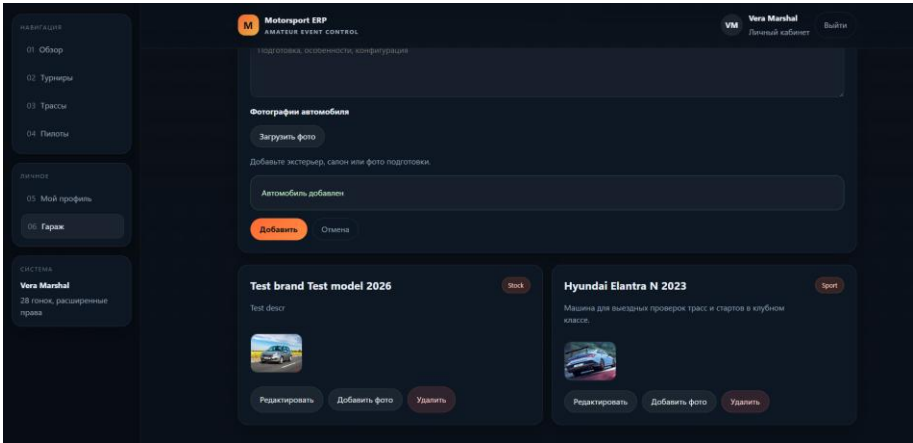


Рисунок 30 – Автомобили пользователя

1.5.5 Работа с каталогом трасс

Страница трасс реализует требования модуля гоночных площадок. В верхней части экрана пользователь может воспользоваться поиском и

фильтрацией по статусу трассы. Это позволяет быстро находить конкретные площадки, просматривать только подтвержденные или официальные объекты и переходить к карточке интересующей трассы. Страница с списком трасс приведена на рисунке 31.

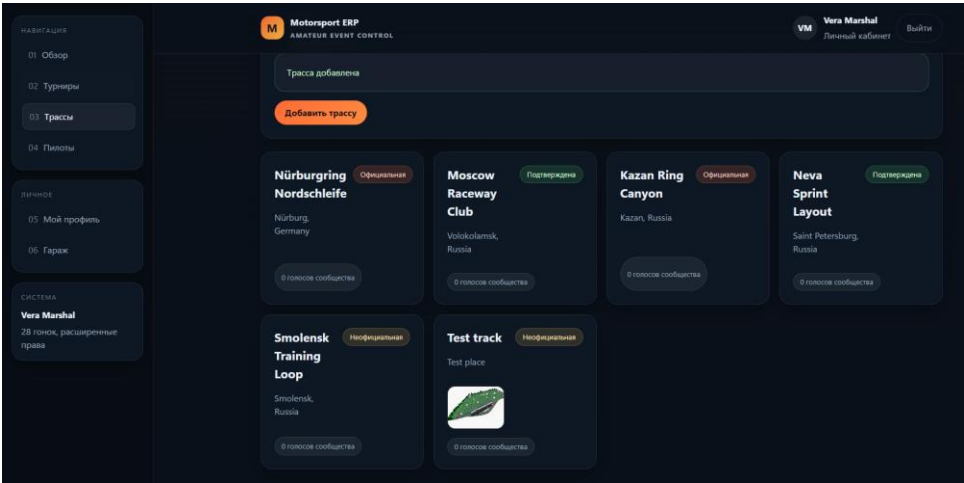


Рисунок 31 – Список трасс

Если пользователь обладает достаточным гоночным стажем либо имеет права модератора, на той же странице для него отображается форма добавления новой трассы. Она показана на рисунке 32. При создании указываются название и местоположение, а также, при необходимости, загружаются фотографии. Для обычного пилота новая трасса создается как неофициальная. Если же операцию выполняет модератор, ему дополнительно может быть доступно управление статусом.

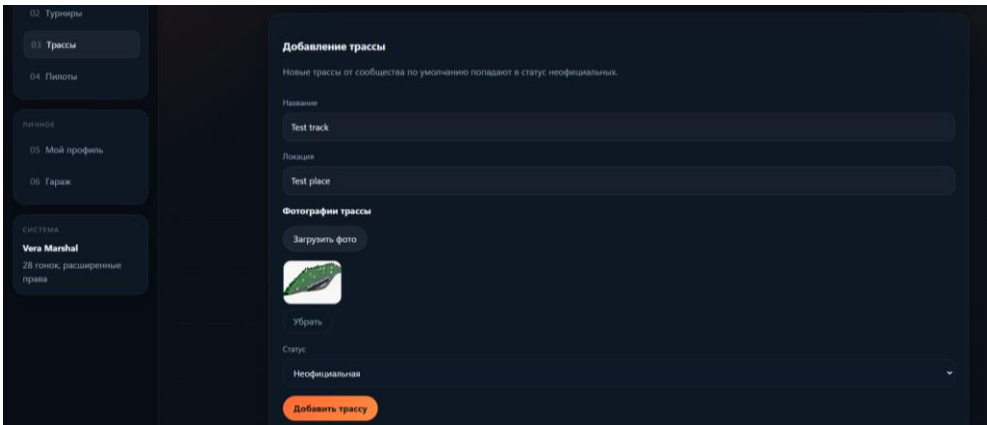


Рисунок 32 – Добавление новой площадки

Детальная страница трассы содержит более полную информацию: название, местоположение, текущий статус, количество набранных голосов, порог подтверждения, сведения об авторе и прикрепленные изображения. Если трасса еще не подтверждена, другие пользователи могут поддержать ее или проголосовать против. Это проиллюстрировано на рисунке 33.

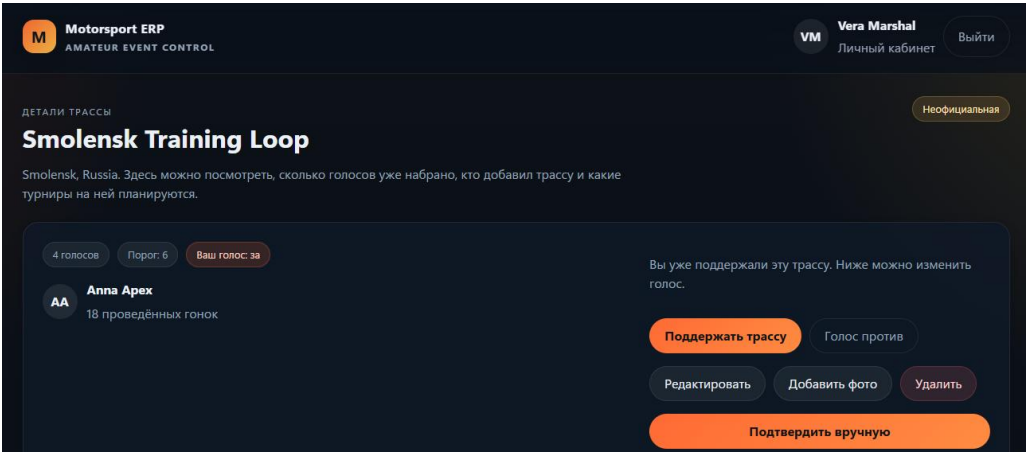


Рисунок 33 – Голосование за трассу

Когда число положительных голосов достигает установленного порога, трасса становится подтвержденной. Кроме того, модератор может подтвердить трассу вручную или присвоить ей статус официальной. Детальная страница трассы показана на рисунке 34.

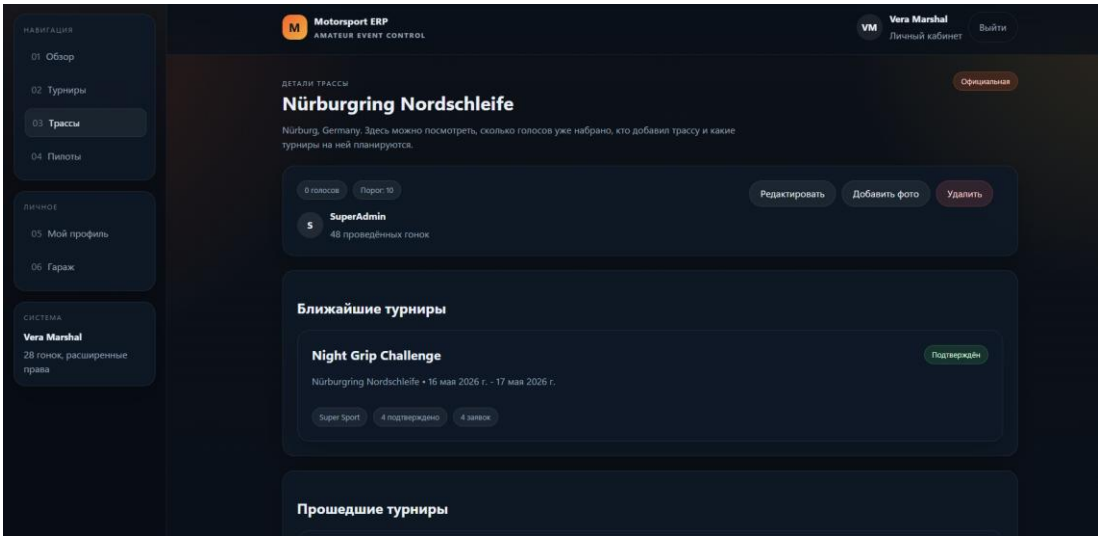


Рисунок 34 – Детальная страница трассы

1.5.6 Работа с турнирами и заявками на участие

На странице турниров пользователь может просматривать общий каталог событий, выполнять поиск по названию, фильтровать записи по статусу и переходить к подробной информации о каждом мероприятии. Страница с турнирами изображена на рисунке 35.

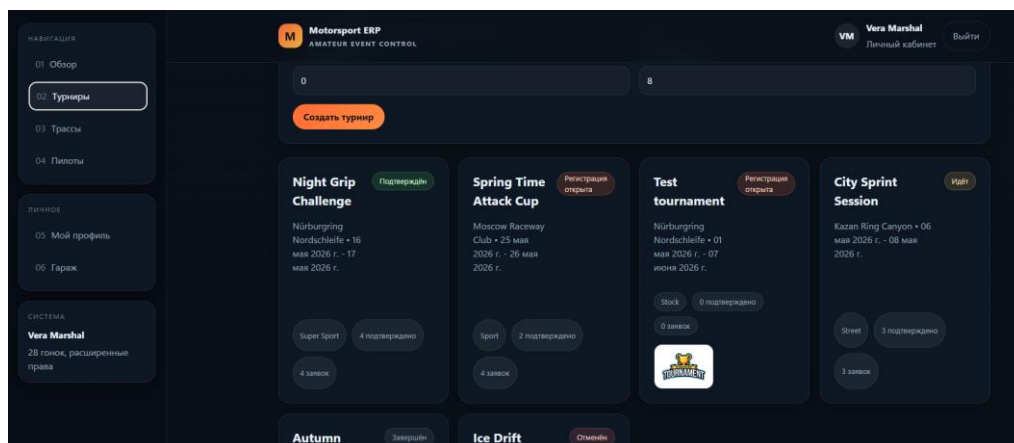


Рисунок 35 – Страница турниров

Если пользователь обладает достаточным стажем либо ролью организатора, на странице для него становится доступна форма создания турнира. При создании организатор задает название, описание, дату начала и окончания, выбирает трассу, допустимый класс автомобиля, минимальный стаж участников и требуемое количество пилотов. Форма создания турнира приведена на рисунке 36.

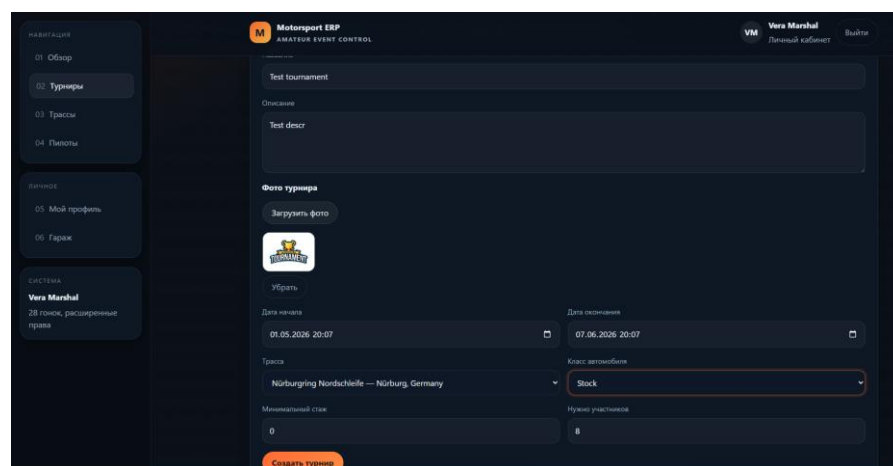


Рисунок 36 – Создание турнира

После открытия детальной страницы турнира пользователь видит основные сведения о событии, что показано на рисунке 37. При наличии загруженных изображений здесь же отображается фотогалерея турнира.

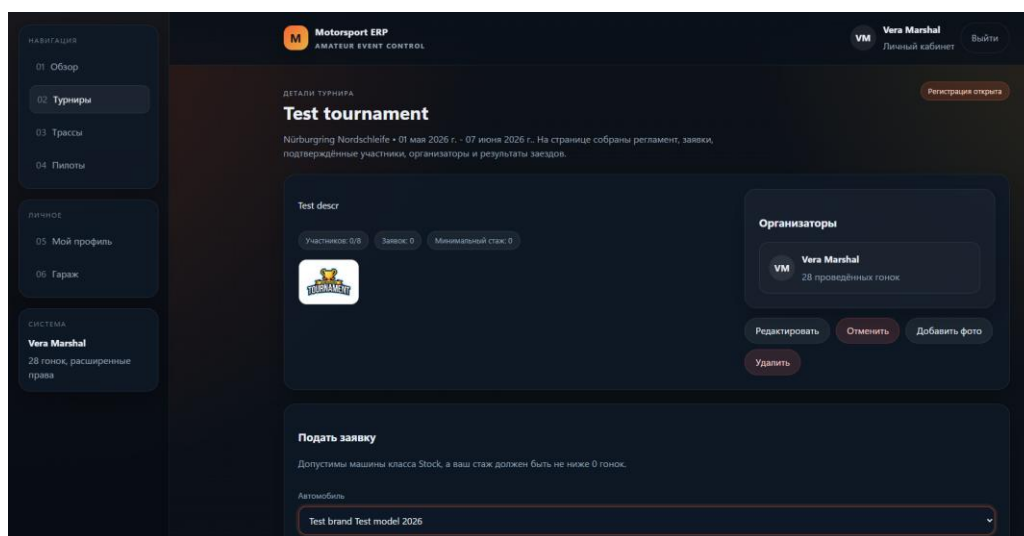


Рисунок 37 – Детальная информация о турнире

Если авторизованный пользователь является пилотом и еще не подал заявку на турнир, а регистрация на событие открыта, на странице отображается форма подачи заявки. Она показана на рисунке 38. В ней можно выбрать только те автомобили из гаража, которые соответствуют допустимому классу турнира.

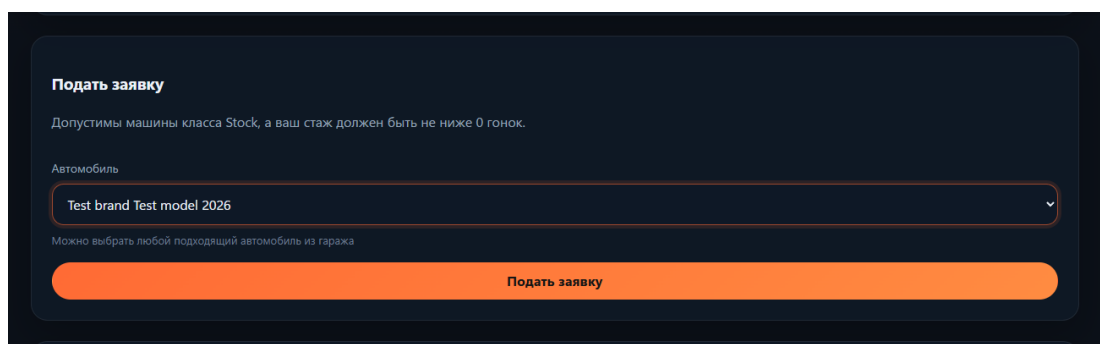


Рисунок 38 – Форма подачи заявки.

После отправки заявки пользователь может отслеживать ее состояние в личном кабинете и на странице самого турнира. Если организатор подтвердит заявку, пользователь перейдет в список участников. Если заявка будет отклонена, соответствующий статус также будет зафиксирован в интерфейсе.

Для организатора или модератора страница турнира содержит расширенный набор инструментов управления. Во-первых, можно редактировать отдельные параметры турнира. Во-вторых, при необходимости допускается назначение соорганизаторов из числа пользователей системы. В-третьих, в отдельном блоке отображаются поступившие заявки, для которых доступны действия подтверждения и отклонения. Эти возможности приведена на рисунке 39.

The screenshot shows a dark-themed interface with two main sections. The top section, titled 'Соорганизаторы' (Co-organizers), includes a 'Добавить пользователя' (Add user) button and a dropdown menu labeled 'Выберите пользователя' (Select user). The bottom section, titled 'Заявки участников' (Participant applications), displays a card for a user named 'Vera Marshal' with a profile picture, '28 проведённых гонок' (28 races completed), and 'Автомобиль: Test brand Test model' (Car: Test brand Test model). To the right of the card is a 'На рассмотрении' (Under review) button. Below the card are two buttons: 'Подтвердить' (Confirm) in orange and 'Отклонить' (Reject) in grey.

Рисунок 39 – Управление заявками и составом организаторов турнира.

В зависимости от текущего состояния организатор может перевести турнир в активную фазу, завершить его или отменить. Эти действия доступны не произвольно, а в соответствии с текущим статусом события, что исключает некорректные переходы между этапами.

После перевода турнира в активное состояние организатор получает доступ к форме внесения результатов. Она изображена на рисунке 40. В этой форме указываются участник, итоговая позиция и лучшее время круга.

The screenshot shows a form titled 'Добавить результат' (Add result). Below the title is a note: 'Результаты можно вводить только для одобренных участников активного турнира.' (Results can only be entered for approved participants of an active tournament). The form has two input fields: 'Участник' (Participant) with a dropdown menu showing 'Kirill Street — Volkswagen Golf GTI', and 'Позиция' (Position) with a numeric input field showing '2'. To the right of the position field is a 'Лучший круг' (Best lap) field with a time input showing '00:01:45'. At the bottom of the form is a large orange button labeled 'Добавить результат' (Add result).

Рисунок 40 – Форма добавления результатов

После завершения турнира результаты отображаются в карточке события, а сведения о завершённом участии учитываются в цифровом профиле пилота. Форма с результатами турнира изображена на рисунке 41.

Результаты				
MS	Maria Sprint	15 проведённых гонок	#1	00:07:58
AA	Анна Арех	18 проведённых гонок	#2	00:08:02
	Vera Marshal	28 проведённых гонок	#3	00:08:07
BD	Boris Drift	12 проведённых гонок	#4	00:08:11

Рисунок 41 – Результаты турнира.

Вывод по руководству пользователя

В четвертой главе были рассмотрены фактические результаты разработки – описано руководство пользователя. Оно подтверждает, что система покрывает все поставленные задачи. Пользовательские действия выстроены в логичную последовательность, а интерфейс отражает реальные процессы предметной области.

Новый участник может зарегистрироваться и войти в систему, заполнить профиль, загрузить аватар и добавить автомобили в виртуальный гараж. После накопления необходимого стажа он получает возможность предлагать новые трассы и, при соблюдении требований, выступать организатором турниров.

Пользователи сообщества могут просматривать каталог трасс, участвовать в голосовании за новые площадки, подавать заявки на турниры и отслеживать статус этих заявок. Организаторы получают инструменты управления жизненным циклом мероприятий: создание события, назначение соорганизаторов, обработка заявок, запуск и завершение турнира, а также фиксация результатов. Модераторы и суперпользователи имеют доступ к административным операциям, связанным с подтверждением трасс, блокировкой учетных записей и назначением ролей.

2 Безопасность жизнедеятельности

В современных условиях значительная часть профессиональной деятельности связана с использованием персональных компьютеров. Это в полной мере относится к разработке программного обеспечения, где работа осуществляется преимущественно в сидячем положении и сопровождается длительным взаимодействием с экраном. Несмотря на отсутствие прямых производственных рисков, такая деятельность связана с рядом факторов, способных негативно влиять на здоровье человека. Для этого выделяют следующие требования к рабочим местам, оборудованным ПЭВМ:

- освещённость должна находиться в пределах 300–500 лк. Свет должен распределяться равномерно, исключая появление резких теней и бликов на экране монитора. Предпочтительно использование рассеянного освещения, а естественный свет должен поступать сбоку от рабочего места;

- эргономика рабочего места должна соответствовать определенным правилам. Монитор должен располагаться на расстоянии 500–700 мм от глаз пользователя и находиться на уровне глаз или немного ниже. Рабочее кресло должно обеспечивать поддержку спины и регулироваться по высоте, что позволяет поддерживать правильную осанку и снижать нагрузку на позвоночник. Конструкция рабочего стола должна обеспечивать удобное размещение оборудования и исключать вынужденные неудобные положения тела. Расстояние между рабочими столами с видеомониторами должно быть не менее 2,0 м, а расстояние между боковыми поверхностями видеомониторов – не менее 1,2 м;

- рекомендуется поддерживать температуру воздуха в пределах 19–23 °С и относительную влажность около 40–60 %. Помещение должно регулярно проветриваться, что способствует поддержанию работоспособности и снижению утомляемости;

- все устройства должны быть корректно подключены и заземлены, что снижает риск поражения электрическим током и обеспечивает стабильную работу

оборудования. Использование исправных кабелей и сетевых фильтров также является обязательным условием безопасной эксплуатации.

Важным аспектом является организация режима труда и отдыха. Рекомендуется делать регулярные перерывы, чередуя работу за компьютером с другими видами деятельности. Оптимальной считается работа с перерывами каждые 45–60 минут продолжительностью 10–15 минут. В течение перерывов рекомендуется выполнять простые упражнения для глаз и разминку, что позволяет снизить напряжение и улучшить общее самочувствие.

Таким образом, правильная организация рабочего места, соблюдение требований к освещению, микроклимату и режиму работы, а также выполнение элементарных правил техники безопасности позволяют снизить негативное воздействие, связанное с работой за персональным компьютером, и обеспечить безопасные условия труда, включая ухудшение зрения, заболевания опорно-двигательного аппарата, повышение уровня стресса и снижение общей работоспособности.

Заключение

В результате выполнения выпускной квалификационной работы разработана информационная система Motorsport ERP – клиент-серверное веб-приложение для организации и проведения любительских автоспортивных мероприятий. Практическим результатом стала цифровая платформа, объединяющая пилотов и организаторов в единой среде. Для участников реализованы инструменты регистрации и авторизации, настройки профиля, управления гаражом автомобилей, просмотра и добавления трасс, участия в голосовании за их подтверждение, подачи заявок на турниры и отслеживания истории активности. Для организаторов реализованы инструменты создания и управления турнирами, обработки заявок и внесения результатов, а для модераторов – функции административного контроля и подтверждения трасс.

Архитектура системы представляет собой backend-комплекс из четырёх серверных проектов, функционирующих как единая инфраструктура. Серверная часть включает семь REST-контроллеров, через которые реализован согласованный HTTP API для всех ключевых функций. В прикладном слое выделены шесть функциональных модулей, каждый из которых отвечает за отдельную предметную область и реализует законченный блок бизнес-логики. Клиентская часть содержит 11 основных экранов. Дополнительно реализованы 14 feature-модулей и 5 виджетов, обеспечивающих модульную структуру интерфейса и поддержку основных пользовательских сценариев без избыточного усложнения системы. На инфраструктурном уровне приложение подготовлено к контейнерному разворачиванию. Разработаны Dockerfile для серверной и клиентской частей, а также общий docker-compose.yml, позволяющий запускать веб-клиент, API и базу данных как единую платформу.

Разработанная система отличается от существующих решений ориентацией именно на потребности любительского автоспорта и сочетает функции социальной платформы и инструмента организационного управления. В отличие от аналогов, где обычно доминирует один аспект, здесь профиль пользователя,

гараж, история участия и управление соревнованиями интегрированы в единую среду. В системе реализована прозрачная модель распределения прав доступа, при которой возможности пользователей зависят от их активности и накопленного опыта. Дополнительным преимуществом является механизм общественного подтверждения трасс. Пользователи могут самостоятельно добавлять площадки, а система голосования выступает инструментом фильтрации и актуализации данных. При этом система сохраняет низкий порог входа для начинающих участников, не требуя профессионального статуса, но обеспечивая управляемость за счет ролей, ограничений, модерации и механизмов блокировки. Реализован целостный жизненный цикл турнира: от создания и набора участников до фиксации результатов, что объединяет все этапы в рамках одной платформы. Наличие персонального гаража и истории участия формирует цифровую идентичность пользователя и влияет на его возможности внутри платформы. Архитектура системы изначально ориентирована на масштабирование: модульный подход позволяет расширять функциональность без переработки базовых компонентов, что делает систему пригодной для дальнейшего развития.

Перспективы дальнейшего развития проекта связаны как с техническими улучшениями, так и с расширением пользовательских возможностей. В дальнейшем целесообразно внедрить автоматизированные unit- и integration-тесты для критически важных бизнес-сценариев, реализовать централизованное логирование ошибок и ключевых событий, а также добавить механизм версионирования API для упрощения поддержки и развития системы. Важным улучшением для платформы будет развитие аналитической и социальной составляющей платформы: внедрение рейтингов пилотов, таблиц лучших времен, статистики выступлений, сравнительного анализа трасс, а также комментариев и чатов. Дополнительно система может быть интегрирована с внешними сервисами, включая системы учета лицензий гонщиков, тайминга и телеметрии, что особенно важно для соревнований с фиксацией времени круга. Возможным направлением развития является создание мобильного клиента или адаптация приложения под

формат Progressive Web App для повышения удобства использования на выездных соревнованиях. Таким образом, разработанная система закладывает основу для формирования полноценной цифровой экосистемы любительского автоспорта.

В ходе тестирования было подтверждено, что приложение соответствует всем поставленным в начале работы требованиям.

Задание на дипломное проектирование выполнено в полном объеме.

					<i>ПГУ 1.090301.03.001 ПЗ</i>	<i>Лист</i>
						79
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		

Список использованных источников

1. Распоряжение Правительства РФ от 07.02.2024 N 264-р (ред. от 21.10.2024) <Об утверждении стратегического направления в области цифровой трансформации физической культуры и спорта на период до 2030 года> / [Электронный ресурс] // Судебные и нормативные акты РФ. – URL: <https://sudact.ru/law/rasporiazhenie-pravitelstva-rf-ot-07022024-n-264-r/> (дата обращения: 10.04.2026).

2. Моторные виды спорта / [Электронный ресурс] // Википедия. – URL: <https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D1%81%D0%BF%D0%BE%D1%80%D1%82>. (дата обращения: 15.04.2026).

3. TIOBE Index / [Электронный ресурс] // TIOBE. – URL: <https://www.tiobe.com/tiobe-index/> (дата обращения: 11.04.2026).

4. Рихтер Джеффри CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Джеффри Рихтер. – 4-е издание. – СПб: ПИТЕР, 2025 – 896 с.

5. Web Framework Benchmarks Round 23 2025-02-24 / [Электронный ресурс] // TechEmpower. – URL: <https://www.techempower.com/benchmarks/#hw=ph&test=json&section=data-r23> (дата обращения: 12.04.2026).

6. Browser support for JavaScript APIs / [Электронный ресурс] // MDN. – URL: https://developer.mozilla.org/en-US/docs/Mozilla/Addons/WebExtensions/Browser_support_for_JavaScript_APIs (дата обращения: 11.04.2026).

7. DB-Engines Ranking / [Электронный ресурс] // DB-Engines. – URL: <https://db-engines.com/en/ranking> (дата обращения: 17.05.2026).

8. Dev IDEs / [Электронный ресурс] // Stackoverflow – URL: <https://survey.stackoverflow.co/2025/technology#most-popular-technologies-dev-envs-dev-envs> (дата обращения: 14.04.2026).

9. Мартин Роберт Чистая архитектура. Искусство разработки программного обеспечения / Роберт Мартин. – СПб: ПИТЕР, 2018 – 352 с.

10. Feature-Sliced Design / [Электронный ресурс] // Feature-Sliced Design. – URL: <https://feature-sliced.design/> (дата обращения: 15.04.2026).

					<i>ПГУ 1.090301.03.001 ПЗ</i>	<i>Лист</i>
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		<i>81</i>

Приложение А

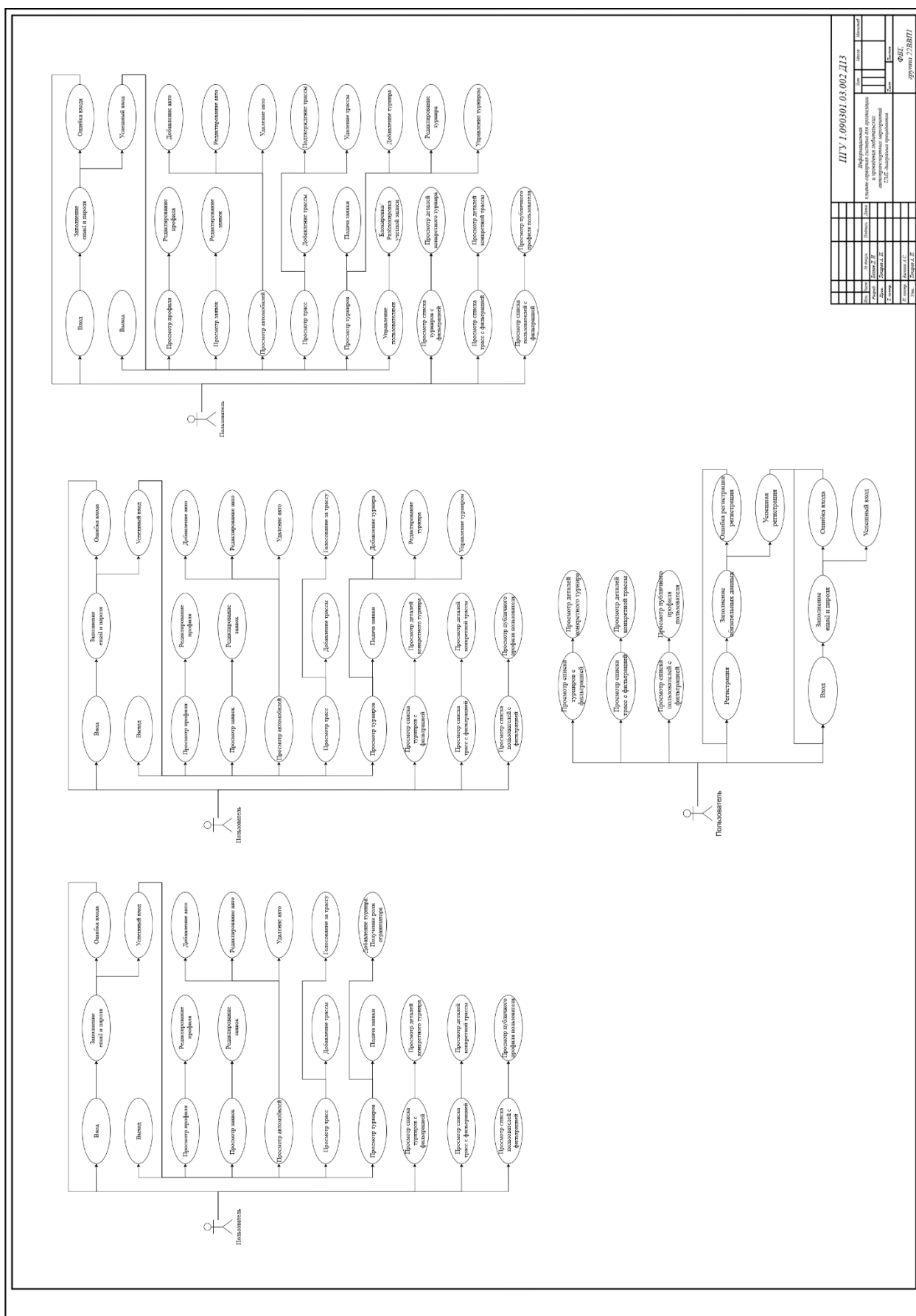
(обязательное)

Информационная клиент-серверная система организации и проведения
любительских автотранспортных мероприятий

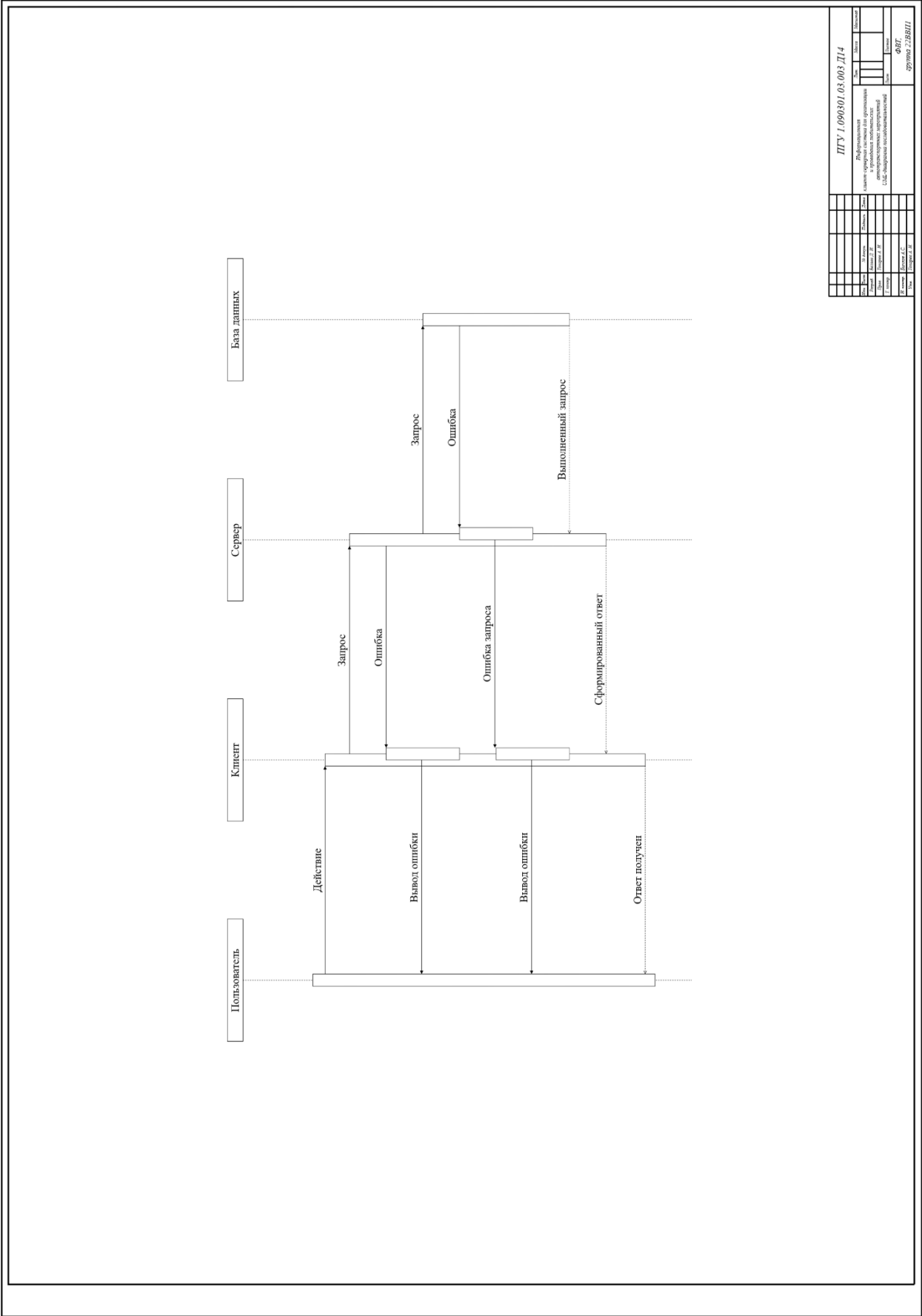
UML-диаграммы

					ПГУ 1.090301.03.001 ПЗ	Лист
						82
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – UML-диаграмма прецедентов



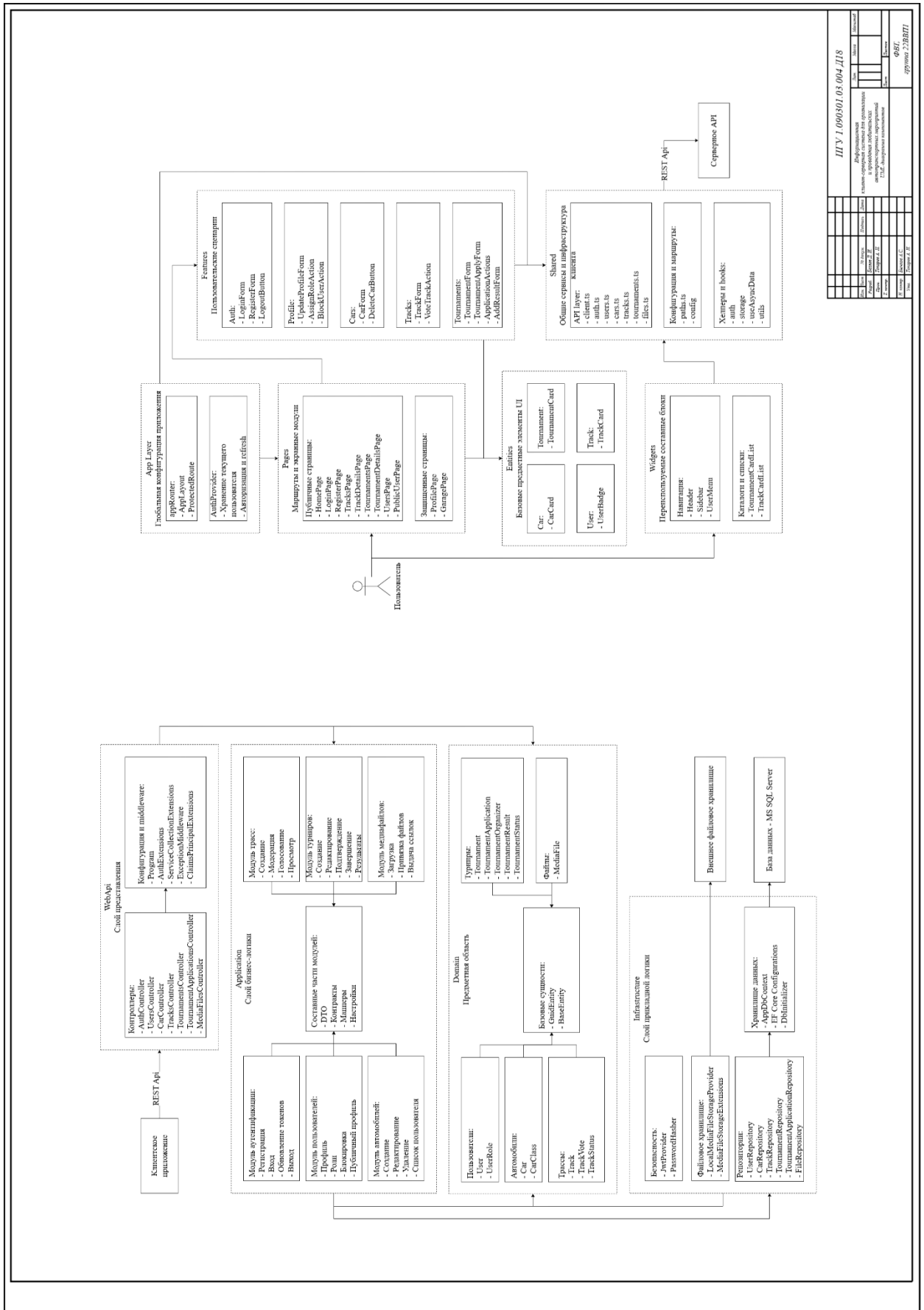
Приложение А.2 – UML-диаграмма последовательностей



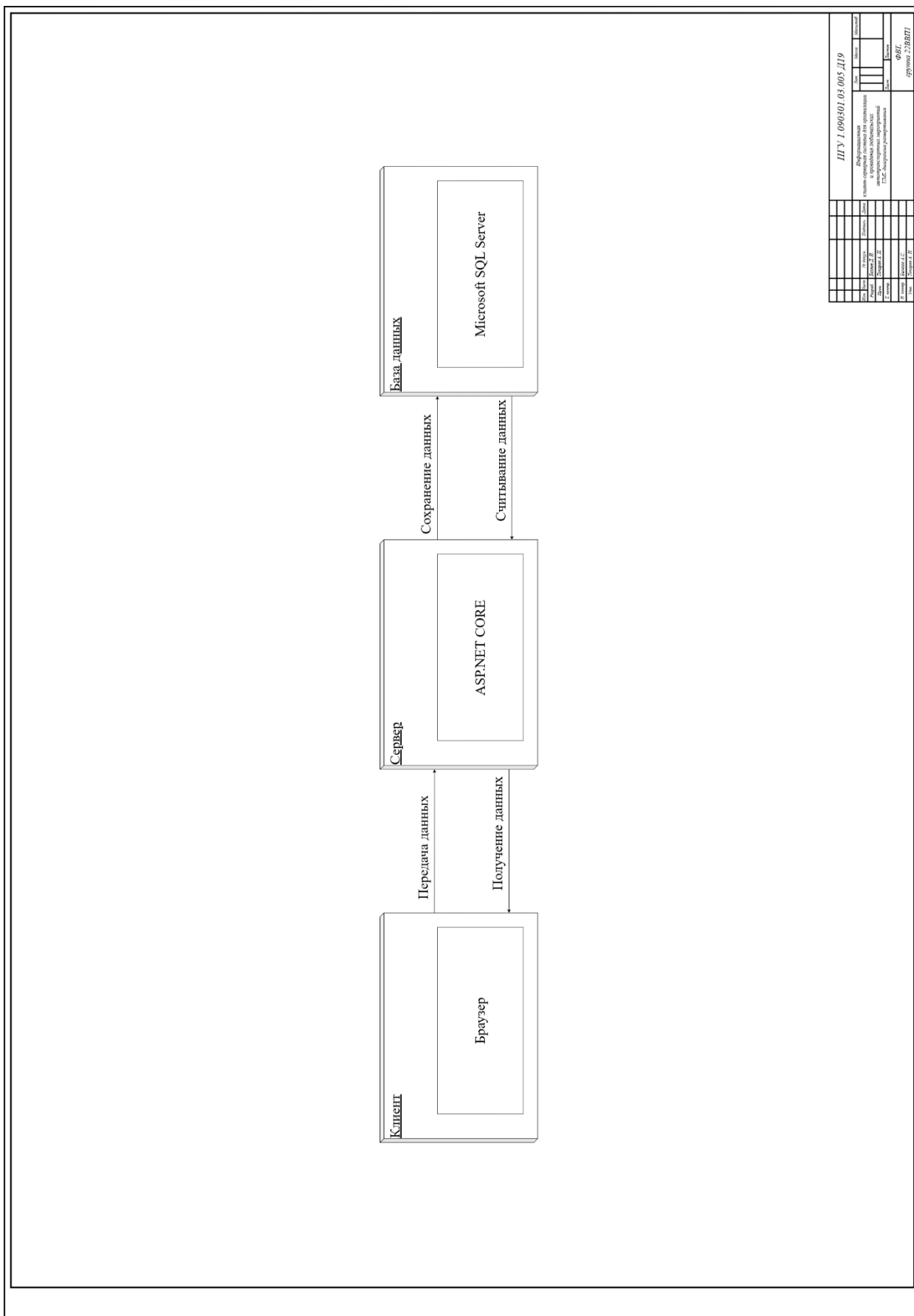
ПГУ 1.090301.03.003 П14									
Информация о документе									
№ докум.	Дата	№ докум.	Дата	№ докум.	Дата	№ докум.	Дата	№ докум.	Дата
Введен	Введен	Введен	Введен	Введен	Введен	Введен	Введен	Введен	Введен
И.И.И.	И.И.И.	И.И.И.	И.И.И.	И.И.И.	И.И.И.	И.И.И.	И.И.И.	И.И.И.	И.И.И.
Ф.И.Т.									
Итого: 2/2/2/1/1									

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.3 – UML-диаграмма компонентов



Приложение А.4 – UML-диаграмма развертывания



																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														</
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

Приложение Б

(обязательное)

Информационная клиент-серверная система организации и проведения
любительских автотранспортных мероприятий

Листинг программы

					ПГУ 1.090301.03.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

Файл: .dockerignore

```
.git
.vs
**/bin
**/obj
apps/frontend/node_modules
apps/frontend/dist
apps/backend/MotorsportErp.WebApi/wwwroot/uploads
```

Файл: .env.example

```
# == SERVER ==
# API
API_PORT=5000
API_INTERNAL_PORT=8080
ASPNETCORE_ENVIRONMENT=Development
# DOCKER BUILD
BUILD_CONFIGURATION=Release
# DATABASE
DB_HOST=db
DB_PORT=1433
DB_INTERNAL_PORT=1433
DB_NAME=MotorsportDb
DB_USER=sa
DB_PASSWORD=CHANGE_ME
# JWT
JwtSettings__Secret=CHANGE_ME
JwtSettings__Issuer=MotorsportApi
JwtSettings__Audience=MotorsportClient
JwtSettings__ExpirationHours=2
# FILE STORAGE
UPLOAD_PATH=/app/wwwroot/uploads
# INITIAL DATA
SeedSettings__AdminPassword=CHANGE_ME

# == WEB CLIENT ==
# WEB
WEB_PORT=3000
WEB_INTERNAL_PORT=80
# API
VITE_API_URL=http://localhost:5000/api
```

Файл: AuthService.cs

```
using Microsoft.IdentityModel.Tokens;
using MotorsportErp.Application.Common.Interfaces.Repositories;
using MotorsportErp.Application.Common.Interfaces.Security;
using MotorsportErp.Application.Features.Auth.Contracts;
```

```

using MotorsportErp.Application.Features.Auth.Interfaces;
using MotorsportErp.Application.Features.Auth.Mappers;
using MotorsportErp.Application.Features.Users.Contracts;
using MotorsportErp.Application.Features.Users.Mappers;
using MotorsportErp.Domain.Users;
using System.Security.Claims;

namespace MotorsportErp.Application.Features.Auth.Services;

public class AuthService : IAuthService
{
    private readonly IUserRepository _userRepository;
    private readonly IPasswordHasher _passwordHasher;
    private readonly IJwtProvider _jwtProvider;

    public AuthService(
        IUserRepository userRepository,
        IPasswordHasher passwordHasher,
        IJwtProvider jwtProvider)
    {
        _userRepository = userRepository;
        _passwordHasher = passwordHasher;
        _jwtProvider = jwtProvider;
    }

    public async Task<UserResponse>
    RegisterAsync(UserRegisterRequest request)
    {
        bool exists = await
        _userRepository.ExistsByEmailAsync(request.Email);
        if (exists)
        {
            throw new InvalidOperationException("User with this
email already exists");
        }

        string passwordHash =
        _passwordHasher.Hash(request.Password);
        User user = UserMapper.ToEntity(request, passwordHash);

        await _userRepository.AddAsync(user);

        return UserMapper.ToResponse(user);
    }

    public async Task<AuthResponse> LoginAsync(UserLoginRequest
request)
    {
        User user = await
        _userRepository.GetByEmailAsync(request.Email) ?? throw new
UnauthorizedAccessException("Invalid email or password");
        if (user.IsBlocked)

```

```

        {
            throw new UnauthorizedAccessException("Account is
blocked");
        }

        bool isValid = _passwordHasher.Verify(request.Password,
user.PasswordHash);
        if (!isValid)
        {
            throw new UnauthorizedAccessException("Invalid
email or password");
        }

        var accessToken = _jwtProvider.GenerateToken(user);
        var refreshToken = _jwtProvider.GenerateRefreshToken();

        user.RefreshToken = refreshToken;
        user.RefreshTokenExpiryTime =
DateTime.UtcNow.AddDays(7);
        await _userRepository.UpdateAsync(user);

        return AuthMapper.ToResponse(accessToken,
refreshToken);
    }

    public async Task<AuthResponse> RefreshTokenAsync(string
accessToken, string refreshToken)
    {
        var principal =
_jwtProvider.GetPrincipalFromExpiredToken(accessToken) ?? throw new
SecurityTokenException("Invalid access/refresh token");

        var userIdStr =
principal.FindFirst(ClaimTypes.NameIdentifier)?.Value;
        if (!Guid.TryParse(userIdStr, out var userId))
        {
            throw new SecurityTokenException("Invalid token
claims");
        }

        var user = await _userRepository.GetByIdAsync(userId)
?? throw new SecurityTokenException("Invalid request");
        if (user.IsBlocked)
        {
            throw new UnauthorizedAccessException("Account is
blocked");
        }

        if (user.RefreshToken != refreshToken ||
user.RefreshTokenExpiryTime <= DateTime.UtcNow)
        {

```

					ИПУ 1.090301.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		90

```

        throw new SecurityTokenException("Invalid or
expired refresh token");
    }

    var newAccessToken = _jwtProvider.GenerateToken(user);
    var newRefreshToken =
_jwtProvider.GenerateRefreshToken();

    user.RefreshToken = newRefreshToken;
    user.RefreshTokenExpiryTime =
DateTime.UtcNow.AddDays(7);
    await _userRepository.UpdateAsync(user);

    return AuthMapper.ToResponse(newAccessToken,
newRefreshToken);
}

public async Task RevokeTokenAsync(Guid userId)
{
    var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");

    user.RefreshToken = null;
    user.RefreshTokenExpiryTime = null;

    await _userRepository.UpdateAsync(user);
}
}

```

Файл: CarService.cs

```

using MotorsportErp.Application.Common.Contracts;
using MotorsportErp.Application.Common.Exceptions;
using MotorsportErp.Application.Common.Extensions;
using MotorsportErp.Application.Common.Interfaces.Repositories;
using MotorsportErp.Application.Features.Cars.Contracts;
using MotorsportErp.Application.Features.Cars.Interfaces;
using MotorsportErp.Application.Features.Cars.Mappers;
using MotorsportErp.Domain.Cars;

namespace MotorsportErp.Application.Features.Cars.Services;

public class CarService : ICarService
{
    private readonly ICarRepository _carRepository;
    private readonly IUserRepository _userRepository;
    private readonly IFileRepository _fileRepository;

    public CarService(
        ICarRepository carRepository,
        IUserRepository userRepository,

```

```

        IFileRepository fileRepository)
    {
        _carRepository = carRepository;
        _userRepository = userRepository;
        _fileRepository = fileRepository;
    }

    public async Task<Guid> CreateAsync(Guid userId,
CarCreateRequest request)
    {
        _ = await _userRepository.GetByIdAsync(userId)
            ?? throw new KeyNotFoundException("User not
found");

        var car = new Car
        {
            Id = Guid.NewGuid(),
            Model = request.Model,
            Brand = request.Brand,
            Year = request.Year,
            Description = request.Description,
            CarClass = request.CarClass,
            OwnerId = userId
        };

        await _carRepository.AddAsync(car);

        return car.Id;
    }

    public async Task<PagedResponse<CarResponse>>
GetUserCarsAsync(Guid userId, int page = 0, int pageSize = 20)
    {
        var (cars, totalCount) = await
_carRepository.GetPagedAsync(c => c.OwnerId == userId, page,
pageSize);

        return new PagedResponse<CarResponse>
        {
            Items = cars.Select(CarMapper.ToResponse).ToList(),
            TotalCount = totalCount,
            Page = page,
            PageSize = pageSize
        };
    }

    public async Task UpdateAsync(Guid userId, Guid carId,
CarUpdateRequest request)
    {
        var car = await _carRepository.GetByIdAsync(carId) ??
throw new KeyNotFoundException("Car not found");
        if (car.OwnerId != userId)

```

```

        {
            throw new ForbiddenException();
        }

        car.Model = request.Model;
        car.Brand = request.Brand;
        car.Year = request.Year;
        car.Description = request.Description;
        car.CarClass = request.CarClass;
        car.OwnerId = userId;

        await _carRepository.UpdateAsync(car);
    }

    public async Task DeleteAsync(Guid userId, Guid carId)
    {
        var car = await _carRepository.GetByIdAsync(carId) ??
        throw new KeyNotFoundException("Car not found");
        if (car.OwnerId != userId)
        {
            throw new ForbiddenException();
        }

        bool hasActive = await
        _carRepository.HasActiveApplicationsAsync(carId);
        if (hasActive)
        {
            throw new InvalidOperationException("Cannot delete
        car used in active tournaments");
        }

        await _carRepository.DeleteAsync(car);
    }

    public async Task AddPhotoAsync(Guid userId, Guid
    targetEntityId, Guid photoId)
    {
        var car = await
        _carRepository.GetByIdAsync(targetEntityId) ?? throw new
        KeyNotFoundException("Car not found");
        if (car.OwnerId != userId)
        {
            throw new ForbiddenException();
        }

        var photo = await _fileRepository.GetByIdAsync(photoId)
        ?? throw new KeyNotFoundException("Photo not found");
        if (photo.UploadedById != userId)
        {
            throw new ForbiddenException("Only owner can use
        self photos");
        }
    }

```

```

        car.Photos.Add(photo);
        await _carRepository.UpdateAsync(car);
    }

    public async Task RemovePhotoAsync(Guid userId, Guid
targetEntityId, Guid photoId)
    {
        var car = await
_carRepository.GetByIdAsync(targetEntityId) ?? throw new
KeyNotFoundException();
        if (car.OwnerId != userId)
        {
            throw new ForbiddenException();
        }

        var photo = await
_fileRepository.GetByIdAsync(photoId);
        if (photo != null)
        {
            _ = car.Photos.Remove(photo);
            await _carRepository.UpdateAsync(car);
        }
    }

    public CarReferenceDataResponse GetReferenceData()
    {
        return new CarReferenceDataResponse()
        {
            CarClasses =
EnumExtensions.GetEnumValues<CarClass>()
                .Select(item => new EnumValueResponse() { Name
= item.Key, Value = item.Value })
                .ToList()
        };
    }
}

```

Файл: MediaFileService.cs

```

using MotorsportErp.Application.Common.Exceptions;
using MotorsportErp.Application.Common.Interfaces.Files;
using MotorsportErp.Application.Common.Interfaces.Repositories;
using MotorsportErp.Application.Features.MediaFiles.Contracts;
using MotorsportErp.Application.Features.MediaFiles.Interfaces;
using MotorsportErp.Domain.Files;

namespace
MotorsportErp.Application.Features.MediaFiles.Services;

public class MediaFileService : IMediaFileService

```

```

    {
        private readonly IFileRepository _fileRepository;
        private readonly IMediaFileStorageProvider
_storageProvider;

        public MediaFileService(IFileRepository fileRepository,
IMediaFileStorageProvider storageProvider)
        {
            _fileRepository = fileRepository;
            _storageProvider = storageProvider;
        }

        public async Task<MediaFileResponse>
UploadImageAsync(Stream fileStream, string fileName, string?
contentType, long fileSize, Guid userId)
        {
            var savedUrl = await
_storageProvider.UploadFileAsync(fileStream, fileName, contentType,
fileSize);

            var mediaFile = new MediaFile
            {
                Id = Guid.NewGuid(),
                OriginalFileName = fileName,
                SavedUrl = savedUrl,
                UploadedById = userId
            };

            await _fileRepository.AddAsync(mediaFile);
            return new MediaFileResponse { Id = mediaFile.Id, Url =
mediaFile.SavedUrl };
        }

        public async Task DeleteFileAsync(Guid fileId, Guid
ownerId)
        {
            var file = await _fileRepository.GetByIdAsync(fileId)
?? throw new KeyNotFoundException();
            if (file.UploadedById != ownerId)
            {
                throw new ForbiddenException();
            }

            _storageProvider.DeleteFile(file.SavedUrl);
            await _fileRepository.DeleteAsync(file);
        }
    }

```

Файл: TournamentService.cs

using Microsoft.Extensions.Options;

					ИГУ 1.090301.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		95

```

using MotorsportErp.Application.Common.Contracts;
using MotorsportErp.Application.Common.Exceptions;
using MotorsportErp.Application.Common.Extensions;
using MotorsportErp.Application.Common.Interfaces.Repositories;
using MotorsportErp.Application.Features.Tournaments.Contracts;
using
MotorsportErp.Application.Features.Tournaments.Interfaces;
using MotorsportErp.Application.Features.Tournaments.Mappers;
using MotorsportErp.Application.Features.Tournaments.Settings;
using MotorsportErp.Domain.Tournaments;
using MotorsportErp.Domain.Tracks;
using MotorsportErp.Domain.Users;

namespace
MotorsportErp.Application.Features.Tournaments.Services;

public class TournamentService : ITournamentService
{
    private readonly ITournamentRepository
_tournamentRepository;
    private readonly ITrackRepository _trackRepository;
    private readonly IUserRepository _userRepository;
    private readonly ICarRepository _carRepository;
    private readonly ITournamentApplicationRepository
_applicationRepository;
    private readonly IFileRepository _fileRepository;

    private readonly TournamentSettings _settings;

    public TournamentService(
        ITournamentRepository tournamentRepository,
        ITrackRepository trackRepository,
        IUserRepository userRepository,
        ICarRepository carRepository,
        ITournamentApplicationRepository applicationRepository,
        IOptions<TournamentSettings> options,
        IFileRepository fileRepository)
    {
        _tournamentRepository = tournamentRepository;
        _trackRepository = trackRepository;
        _userRepository = userRepository;
        _carRepository = carRepository;
        _applicationRepository = applicationRepository;
        _settings = options.Value;
        _fileRepository = fileRepository;
    }

    public async Task<Guid> CreateAsync(Guid userId,
TournamentCreateRequest request)
    {
        if (request.StartDate >= request.EndDate)
        {

```

```

        throw new ArgumentException("Tournament end date
must be after the start date.");
    }

    if (request.RequiredParticipants <= 0)
    {
        throw new ArgumentException("Required participants
must be greater than zero.");
    }

    var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
    var track = await
_trackRepository.GetByIdAsync(request.TrackId) ?? throw new
KeyNotFoundException("Track not found");

    bool isPrivileged =
user.Roles.HasFlag(UserRole.Organizer) ||
                                user.Roles.HasFlag(UserRole.Moderator)
||

user.Roles.HasFlag(UserRole.SuperAdmin);

    if (!isPrivileged)
    {
        if (user.RaceCount <
_settings.MinRacesToBecomeOrganizer)
        {
            throw new ForbiddenException($"You need at
least {_settings.MinRacesToBecomeOrganizer} races to create a
tournament.");
        }

        user.Roles |= UserRole.Organizer;
    }

    if (track.Status == TrackStatus.Unofficial)
    {
        throw new InvalidOperationException("Tournament can
be created only on confirmed or official tracks.");
    }

    var tournament = new Tournament
    {
        Id = Guid.NewGuid(),
        Name = request.Name,
        Description = request.Description,
        TrackId = request.TrackId,
        StartDate = request.StartDate,
        EndDate = request.EndDate,
        RequiredParticipants =
request.RequiredParticipants,

```

```

        AllowedCarClass = request.AllowedCarClass,
        RequiredRaceCount = request.RequiredRaceCount,
        Status = TournamentStatus.RegistrationOpen,
        CreatorId = userId,
        Organizers =
        [
            new TournamentOrganizer
            {
                Id = Guid.NewGuid(),
                UserId = userId
            }
        ]
    };

    await _tournamentRepository.AddAsync(tournament);
    await _userRepository.UpdateAsync(user);

    return tournament.Id;
}

public async Task UpdateAsync(Guid userId, Guid
tournamentId, TournamentUpdateRequest request)
{
    if (request.StartDate >= request.EndDate)
    {
        throw new ArgumentException("Tournament end date
must be after the start date.");
    }

    if (request.RequiredParticipants <= 0)
    {
        throw new ArgumentException("Required participants
must be greater than zero.");
    }

    var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
    var tournament = await
_tournamentRepository.GetByIdAsync(tournamentId) ?? throw new
KeyNotFoundException("Tournament not found");
    if (!HasAccess(tournament, user))
    {
        throw new ForbiddenException();
    }

    if (tournament.Status !=
TournamentStatus.RegistrationOpen)
    {
        throw new InvalidOperationException("Cannot update
after registration");
    }
}

```

```

        tournament.Description = request.Description;
        tournament.StartDate = request.StartDate;
        tournament.EndDate = request.EndDate;
        tournament.RequiredParticipants =
request.RequiredParticipants;

        await _tournamentRepository.UpdateAsync(tournament);
    }

    public async Task CancelAsync(Guid userId, Guid
tournamentId)
    {
        var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
        var tournament = await
_tournamentRepository.GetByIdAsync(tournamentId) ?? throw new
KeyNotFoundException("Tournament not found");
        if (!HasAccess(tournament, user))
        {
            throw new ForbiddenException();
        }

        if (tournament.Status == TournamentStatus.Finished)
        {
            throw new InvalidOperationException("Cannot cancel
finished tournament");
        }

        tournament.Status = TournamentStatus.Cancelled;

        await _tournamentRepository.UpdateAsync(tournament);
    }

    public async Task ApplyAsync(Guid userId, Guid
tournamentId, Guid carId)
    {
        var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
        if (user.IsBlocked)
        {
            throw new UnauthorizedAccessException("User is
blocked");
        }

        var tournament = await
_tournamentRepository.GetByIdAsync(tournamentId) ?? throw new
KeyNotFoundException("Tournament not found");
        if (tournament.Status !=
TournamentStatus.RegistrationOpen)
        {
            throw new InvalidOperationException("Registration
closed");
        }
    }

```

```

    }

    var car = await _carRepository.GetByIdAsync(carId) ??
throw new KeyNotFoundException("Car not found");
    if (car.OwnerId != userId)
    {
        throw new ForbiddenException("Not your car");
    }

    if (car.CarClass != tournament.AllowedCarClass)
    {
        throw new InvalidOperationException("Car class
mismatch");
    }

    if (user.RaceCount < tournament.RequiredRaceCount)
    {
        throw new InvalidOperationException("Not enough
races");
    }

    bool exists = await
_applicationRepository.ExistsAsync(userId, tournamentId);
    if (exists)
    {
        throw new InvalidOperationException("Already
applied");
    }

    var application = new TournamentApplication
    {
        Id = Guid.NewGuid(),
        UserId = userId,
        TournamentId = tournamentId,
        CarId = carId,
        Status = TournamentApplicationStatus.Pending
    };

    await _applicationRepository.AddAsync(application);
}

public async Task ApproveAsync(Guid userId, Guid
applicationId)
{
    var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
    var application = await
_applicationRepository.GetByIdAsync(applicationId) ?? throw new
KeyNotFoundException("Application not found");
    var tournament = await
_tournamentRepository.GetByIdAsync(application.TournamentId) ??
throw new KeyNotFoundException("Tournament not found");

```

```

        if (!HasAccess(tournament, user))
        {
            throw new ForbiddenException("Only organizers can
approve applications.");
        }

        if (application.Status !=
TournamentApplicationStatus.Pending)
        {
            throw new InvalidOperationException("Application is
not pending");
        }

        if (tournament.Status !=
TournamentStatus.RegistrationOpen)
        {
            throw new InvalidOperationException("Applications
can only be approved while registration is open.");
        }

        await
_tournamentRepository.ApproveApplicationAtomicallyAsync(applicationI
d);
    }

    public async Task RejectAsync(Guid userId, Guid
applicationId)
    {
        var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
        var application = await
_applicationRepository.GetByIdAsync(applicationId) ?? throw new
KeyNotFoundException("Application not found");
        var tournament = await
_tournamentRepository.GetByIdAsync(application.TournamentId) ??
throw new KeyNotFoundException("Tournament not found");

        if (!HasAccess(tournament, user))
        {
            throw new ForbiddenException();
        }

        if (application.Status !=
TournamentApplicationStatus.Pending)
        {
            throw new InvalidOperationException("Application is
not pending");
        }

        if (tournament.Status !=
TournamentStatus.RegistrationOpen)

```

```

        {
            throw new InvalidOperationException("Applications
can only be rejected while registration is open.");
        }

        application.Status =
TournamentApplicationStatus.Rejected;

        await _applicationRepository.UpdateAsync(application);
    }

    public async Task StartAsync(Guid userId, Guid
tournamentId)
    {
        var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
        var tournament = await
_tournamentRepository.GetByIdAsync(tournamentId) ?? throw new
KeyNotFoundException("Tournament not found");

        if (!HasAccess(tournament, user))
        {
            throw new ForbiddenException();
        }

        if (tournament.Status != TournamentStatus.Confirmed)
        {
            throw new InvalidOperationException("Tournament not
confirmed");
        }

        if (DateTime.UtcNow < tournament.StartDate)
        {
            throw new InvalidOperationException("Too early to
start");
        }

        tournament.Status = TournamentStatus.Active;

        await _tournamentRepository.UpdateAsync(tournament);
    }

    public async Task FinishAsync(Guid userId, Guid
tournamentId)
    {
        var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
        var tournament = await
_tournamentRepository.GetByIdAsync(tournamentId) ?? throw new
KeyNotFoundException("Tournament not found");

        if (!HasAccess(tournament, user))

```

```

        {
            throw new ForbiddenException();
        }

        if (tournament.Status != TournamentStatus.Active)
        {
            throw new InvalidOperationException("Tournament not
active");
        }

        if (!tournament.Results.Any())
        {
            throw new InvalidOperationException("No results to
finish the tournament");
        }

        tournament.Status = TournamentStatus.Finished;
        await _tournamentRepository.UpdateAsync(tournament);

        foreach (var result in tournament.Results)
        {
            var participant = await
_userRepository.GetByIdAsync(result.UserId);
            if (participant != null)
            {
                participant.RaceCount += 1;
                await _userRepository.UpdateAsync(participant);
            }
        }

        public async Task AddResultAsync(Guid userId, Guid
tournamentId, TournamentResultRequest request)
        {
            var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
            var tournament = await
_tournamentRepository.GetByIdAsync(tournamentId) ?? throw new
KeyNotFoundException("Tournament not found");

            if (!HasAccess(tournament, user))
            {
                throw new ForbiddenException();
            }

            if (tournament.Status != TournamentStatus.Active)
            {
                throw new InvalidOperationException("Tournament not
active");
            }

```

```

        if (tournament.Results.Any(r => r.UserId ==
request.UserId))
        {
            throw new InvalidOperationException("Result already
exists");
        }

        var approvedApplication = tournament.Applications
            .Any(a => a.UserId == request.UserId && a.Status ==
TournamentApplicationStatus.Approved);

        if (!approvedApplication)
        {
            throw new InvalidOperationException("Result can be
added only for approved participants.");
        }

        if (request.Position <= 0)
        {
            throw new ArgumentException("Invalid position");
        }

        tournament.Results.Add(new TournamentResult
        {
            Id = Guid.NewGuid(),
            UserId = request.UserId,
            Position = request.Position,
            BestLapTime = request.BestLapTime
        });

        await _tournamentRepository.UpdateAsync(tournament);
    }

    public async Task<PagedResponse<TournamentResponse>>
GetAllAsync(
        TournamentListQuery query,
        int page = 0,
        int pageSize = 20)
    {
        query ??= new TournamentListQuery();
        var (tournaments, totalCount) = await
_tournamentRepository.GetFilteredPagedAsync(query, page, pageSize);

        return new PagedResponse<TournamentResponse>
        {
            Items = tournaments.Select(t =>
TournamentMapper.ToResponse(t)).ToList(),
            TotalCount = totalCount,
            Page = page,
            PageSize = pageSize
        };
    }
}

```

```

        public async Task<TournamentDetailsResponse>
GetByIdAsync(Guid id)
        {
            var tournament = await
            _tournamentRepository.GetByIdAsync(id)
            ?? throw new KeyNotFoundException("Tournament not
found");

            return TournamentMapper.ToDetails(tournament);
        }

        public async
Task<IReadOnlyCollection<TournamentApplicationResponse>>
GetApplicationsAsync(Guid userId, Guid tournamentId)
        {
            var user = await _userRepository.GetByIdAsync(userId)
            ?? throw new KeyNotFoundException("User not found");
            var tournament = await
            _tournamentRepository.GetByIdAsync(tournamentId) ?? throw new
            KeyNotFoundException("Tournament not found");

            return !HasAccess(tournament, user)
                ? throw new ForbiddenException()
                :
            (IReadOnlyCollection<TournamentApplicationResponse>)tournament.Appli
            cations
                .OrderBy(a => a.Status)
                .ThenBy(a => a.User.Nickname)
                .Select(TournamentMapper.ToApplicationResponse)
                .ToList();
        }

        public async Task DeleteAsync(Guid userId, Guid
tournamentId)
        {
            var user = await _userRepository.GetByIdAsync(userId)
            ?? throw new KeyNotFoundException("User not found");
            var tournament = await
            _tournamentRepository.GetByIdAsync(tournamentId) ?? throw new
            KeyNotFoundException("Tournament not found");

            bool isModerator =
            user.Roles.HasFlag(UserRole.Moderator) ||
            user.Roles.HasFlag(UserRole.SuperAdmin);

            if (!isModerator)
            {
                throw new ForbiddenException("Only moderators can
delete tournaments");
            }
        }

```

```

        await _tournamentRepository.DeleteAsync(tournament);
    }

    public async Task AddOrganizerAsync(Guid userId, Guid
tournamentId, Guid newOrganizerId)
    {
        var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
        var tournament = await
_tournamentRepository.GetByIdAsync(tournamentId) ?? throw new
KeyNotFoundException("Tournament not found");

        if (!HasAccess(tournament, user))
        {
            throw new ForbiddenException("No permission");
        }

        var newOrganizer = await
_userRepository.GetByIdAsync(newOrganizerId) ?? throw new
KeyNotFoundException("New organizer not found");

        if (tournament.Organizers.Any(o => o.UserId ==
newOrganizerId))
        {
            throw new InvalidOperationException("User is
already an organizer");
        }

        tournament.Organizers.Add(new TournamentOrganizer
        {
            Id = Guid.NewGuid(),
            TournamentId = tournamentId,
            UserId = newOrganizerId
        });

        if (!newOrganizer.Roles.HasFlag(UserRole.Organizer))
        {
            newOrganizer.Roles |= UserRole.Organizer;
            await _userRepository.UpdateAsync(newOrganizer);
        }

        await _tournamentRepository.UpdateAsync(tournament);
    }

    public async Task CancelApplicationAsync(Guid userId, Guid
applicationId)
    {
        var application = await
_applicationRepository.GetByIdAsync(applicationId) ?? throw new
KeyNotFoundException("Application not found");
        if (application.UserId != userId)
        {

```

```

        throw new ForbiddenException("You can only cancel
your own applications");
    }

    var tournament = await
    _tournamentRepository.GetByIdAsync(application.TournamentId) ??
    throw new KeyNotFoundException("Tournament not found");
    if (tournament.Status !=
    TournamentStatus.RegistrationOpen)
    {
        throw new InvalidOperationException("Cannot cancel
application after registration is closed");
    }

    var wasApproved = application.Status ==
    TournamentApplicationStatus.Approved;
    application.Status =
    TournamentApplicationStatus.Cancelled;
    await _applicationRepository.UpdateAsync(application);

    if (wasApproved)
    {
        var approvedCount = await
        _applicationRepository.GetApprovedCountAsync(tournament.Id);
        if (approvedCount < tournament.RequiredParticipants
        && tournament.Status == TournamentStatus.Confirmed)
        {
            tournament.Status =
            TournamentStatus.RegistrationOpen;
            await
            _tournamentRepository.UpdateAsync(tournament);
        }
    }

    public async Task AddPhotoAsync(Guid userId, Guid
    targetEntityId, Guid photoId)
    {
        var user = await _userRepository.GetByIdAsync(userId)
        ?? throw new KeyNotFoundException("User not found");
        var tournament = await
        _tournamentRepository.GetByIdAsync(targetEntityId) ?? throw new
        KeyNotFoundException("Tournament not found");

        if (!HasAccess(tournament, user))
        {
            throw new ForbiddenException("No permission");
        }

        if (tournament.Status !=
        TournamentStatus.RegistrationOpen)
        {

```

```

        throw new ForbiddenException("Cannot update after
registration");
    }

    var photo = await _fileRepository.GetByIdAsync(photoId)
    ?? throw new KeyNotFoundException("Photo not found");
    if (photo.UploadedById != userId)
    {
        throw new ForbiddenException("Only owner can use
self photos");
    }

    tournament.Photos.Add(photo);
    await _tournamentRepository.UpdateAsync(tournament);
}

public async Task RemovePhotoAsync(Guid userId, Guid
targetEntityId, Guid photoId)
{
    var user = await _userRepository.GetByIdAsync(userId)
    ?? throw new KeyNotFoundException("User not found");
    var tournament = await
    _tournamentRepository.GetByIdAsync(targetEntityId) ?? throw new
    KeyNotFoundException("Tournament not found");

    if (!HasAccess(tournament, user))
    {
        throw new ForbiddenException("No permission");
    }

    if (tournament.Status !=
    TournamentStatus.RegistrationOpen)
    {
        throw new InvalidOperationException("Cannot update
after registration");
    }

    var photo = await _fileRepository.GetByIdAsync(photoId)
    ?? throw new KeyNotFoundException("Photo not found");
    if (photo != null)
    {
        _ = tournament.Photos.Remove(photo);
        await
        _tournamentRepository.UpdateAsync(tournament);
    }
}

private static bool HasAccess(Tournament tournament, User
user)
{
    return tournament.CreatorId == user.Id ||

```

```

        tournament.Organizers.Any(o => o.UserId ==
user.Id) ||
        user.Roles.HasFlag(UserRole.Moderator) ||
        user.Roles.HasFlag(UserRole.SuperAdmin);
    }

    public TournamentReferenceDataResponse GetReferenceData()
    {
        return new TournamentReferenceDataResponse
        {
            MinRacesToBecomeOrganizer =
_settings.MinRacesToBecomeOrganizer,
            TournamentApplicationStatuses =
EnumExtensions.GetEnumValues<TournamentApplicationStatus>()
                .Select(item => new EnumValueResponse() { Name
= item.Key, Value = item.Value })
                .ToList(),
            TournamentStatuses =
EnumExtensions.GetEnumValues<TournamentStatus>()
                .Select(item => new EnumValueResponse() { Name
= item.Key, Value = item.Value })
                .ToList()
        };
    }
}

```

Файл: TrackService.cs

```

using Microsoft.Extensions.Options;
using MotorsportErp.Application.Common.Contracts;
using MotorsportErp.Application.Common.Exceptions;
using MotorsportErp.Application.Common.Extensions;
using MotorsportErp.Application.Common.Interfaces.Repositories;
using MotorsportErp.Application.Features.Tracks.Contracts;
using MotorsportErp.Application.Features.Tracks.Interfaces;
using MotorsportErp.Application.Features.Tracks.Mappers;
using MotorsportErp.Application.Features.Tracks.Settings;
using MotorsportErp.Domain.Tournaments;
using MotorsportErp.Domain.Tracks;
using MotorsportErp.Domain.Users;

namespace MotorsportErp.Application.Features.Tracks.Services;

public class TrackService : ITrackService
{
    private readonly ITrackRepository _trackRepository;
    private readonly IUserRepository _userRepository;
    private readonly IFileRepository _fileRepository;

    private readonly TrackSettings _settings;

```

```

public TrackService(
    ITrackRepository trackRepository,
    IUserRepository userRepository,
    IOptions<TrackSettings> options,
    IFileRepository fileRepository)
{
    _trackRepository = trackRepository;
    _userRepository = userRepository;
    _settings = options.Value;
    _fileRepository = fileRepository;
}

public async Task<PagedResponse<TrackResponse>>
GetAllAsync(TrackListQuery query, int page = 0, int pageSize = 20)
{
    query ??= new TrackListQuery();
    var (tracks, totalCount) = await
_trackRepository.GetFilteredPagedAsync(query, page, pageSize);

    return new PagedResponse<TrackResponse>
    {
        Items =
tracks.Select(TrackMapper.ToResponse).ToList(),
        TotalCount = totalCount,
        Page = page,
        PageSize = pageSize
    };
}

public async Task<TrackDetailsResponse> GetByIdAsync(Guid
id, Guid? viewerUserId = null)
{
    Track track = await _trackRepository.GetByIdAsync(id)
?? throw new KeyNotFoundException("Track not found");
    TrackDetailsResponse response =
TrackMapper.ToDetails(track);

    if (viewerUserId.HasValue)
    {
        bool? isPositive = await
_trackRepository.GetUserVoteIsPositiveAsync(id, viewerUserId.Value);

        response.CurrentUserVote = new
TrackCurrentUserVoteDto
        {
            HasVoted = isPositive.HasValue,
            IsPositive = isPositive,
        };
    }

    return response;
}

```

```

        public async Task<Guid> CreateAsync(Guid userId,
TrackCreateRequest request)
        {
            var user = await _userRepository.GetByIdAsync(userId)
            ?? throw new KeyNotFoundException("User not found");
            bool isModerator =
user.Roles.HasFlag(UserRole.Moderator) ||
user.Roles.HasFlag(UserRole.SuperAdmin);
            if (!isModerator && user.RaceCount <
_settings.MinRacesToCreate)
            {
                throw new ForbiddenException("Not enough races to
create track");
            }

            Track track = TrackMapper.ToEntity(request, userId);
            track.Status = TrackStatus.Unofficial;
            track.ConfirmationThreshold =
_settings.DefaultConfirmationThreshold;

            if (isModerator && request.Status.HasValue)
            {
                track.Status = request.Status.Value;
            }

            await _trackRepository.AddAsync(track);

            return track.Id;
        }

        public async Task VoteAsync(Guid userId, Guid trackId, bool
isPositive)
        {
            var user = await _userRepository.GetByIdAsync(userId)
            ?? throw new KeyNotFoundException("User not found");
            if (user.IsBlocked)
            {
                throw new UnauthorizedAccessException("User is
blocked");
            }

            var votingInfo =
await _trackRepository.GetVotingInfoAsync(trackId)
            ??
                throw new KeyNotFoundException("Track not found");
            if (votingInfo.Status != TrackStatus.Unofficial)
            {
                throw new InvalidOperationException("Can only vote
for unofficial tracks");
            }
        }

```

```

        if (votingInfo.CreatedById == userId)
        {
            throw new InvalidOperationException("Cannot vote
for your own track");
        }

        await _trackRepository.PersistUserVoteAsync(trackId,
userId, isPositive);
    }

    public async Task ConfirmAsync(Guid userId, Guid trackId)
    {
        var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
        bool isModerator =
user.Roles.HasFlag(UserRole.Moderator) ||
user.Roles.HasFlag(UserRole.SuperAdmin);
        if (!isModerator)
        {
            throw new ForbiddenException("Only moderator can
confirm track");
        }

        var track = await
_trackRepository.GetByIdAsync(trackId) ?? throw new
KeyNotFoundException("Track not found");
        if (track.Status == TrackStatus.Confirmed)
        {
            throw new InvalidOperationException("Track already
confirmed");
        }

        track.Status = TrackStatus.Confirmed;

        await _trackRepository.UpdateAsync(track);
    }

    public async Task UpdateAsync(Guid userId, Guid trackId,
TrackUpdateRequest request)
    {
        var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
        if (user.IsBlocked)
        {
            throw new UnauthorizedAccessException("User is
blocked");
        }

        var track = await
_trackRepository.GetByIdAsync(trackId) ?? throw new
KeyNotFoundException("Track not found");

```

```

        bool isOwner = track.CreatedById == userId;
        bool isModerator =
user.Roles.HasFlag(UserRole.Moderator) ||
user.Roles.HasFlag(UserRole.SuperAdmin);
        if (!isOwner && !isModerator)
        {
            throw new ForbiddenException("You cannot edit this
track");
        }

        track.Name = request.Name;
        track.Location = request.Location;

        await _trackRepository.UpdateAsync(track);
    }

    public async Task DeleteAsync(Guid userId, Guid trackId)
    {
        var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
        var track = await
_trackRepository.GetByIdAsync(trackId) ?? throw new
KeyNotFoundException("Track not found");

        bool isOwner = track.CreatedById == userId;
        bool isModerator =
user.Roles.HasFlag(UserRole.Moderator) ||
user.Roles.HasFlag(UserRole.SuperAdmin);
        if (!isOwner && !isModerator)
        {
            throw new ForbiddenException("You cannot delete
this track");
        }

        if (track.Tournaments.Any(t => t.Status !=
TournamentStatus.Finished))
        {
            throw new InvalidOperationException("Cannot delete
track used in active tournaments");
        }

        await _trackRepository.DeleteAsync(track);
    }

    public async Task MakeOfficialAsync(Guid userId, Guid
trackId)
    {
        var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
        bool isModerator =
user.Roles.HasFlag(UserRole.Moderator) ||
user.Roles.HasFlag(UserRole.SuperAdmin);

```

```

        if (!isModerator)
        {
            throw new ForbiddenException("Only moderator can
make track official");
        }

        var track = await
_trackRepository.GetByIdAsync(trackId) ?? throw new
KeyNotFoundException("Track not found");
        if (track.Status != TrackStatus.Confirmed)
        {
            throw new InvalidOperationException("Track must be
confirmed first");
        }

        track.Status = TrackStatus.Official;

        await _trackRepository.UpdateAsync(track);
    }

    public async Task AddPhotoAsync(Guid userId, Guid
targetEntityId, Guid photoId)
    {
        var user = await _userRepository.GetByIdAsync(userId)
?? throw new KeyNotFoundException("User not found");
        if (user.IsBlocked)
        {
            throw new UnauthorizedAccessException("User is
blocked");
        }

        var track = await
_trackRepository.GetByIdAsync(targetEntityId) ?? throw new
KeyNotFoundException("Track not found");

        bool isOwner = track.CreatedById == userId;
        bool isModerator =
user.Roles.HasFlag(UserRole.Moderator) ||
user.Roles.HasFlag(UserRole.SuperAdmin);
        if (!isOwner && !isModerator)
        {
            throw new ForbiddenException("You cannot edit this
track");
        }

        var photo = await _fileRepository.GetByIdAsync(photoId)
?? throw new KeyNotFoundException("Photo not found");
        if (photo.UploadedById != userId)
        {
            throw new ForbiddenException("Only owner can use
self photos");
        }
    }

```

```

        track.Photos.Add(photo);
        await _trackRepository.UpdateAsync(track);
    }

    public async Task RemovePhotoAsync(Guid userId, Guid
targetEntityId, Guid photoId)
    {
        var user = await _userRepository.GetByIdAsync(userId)
        ?? throw new KeyNotFoundException("User not found");
        if (user.IsBlocked)
        {
            throw new UnauthorizedAccessException("User is
blocked");
        }

        var track = await
_trackRepository.GetByIdAsync(targetEntityId) ?? throw new
KeyNotFoundException("Track not found");

        bool isOwner = track.CreatedById == userId;
        bool isModerator =
user.Roles.HasFlag(UserRole.Moderator) ||
user.Roles.HasFlag(UserRole.SuperAdmin);
        if (!isOwner && !isModerator)
        {
            throw new ForbiddenException("You cannot edit this
track");
        }

        var photo = await
_fileRepository.GetByIdAsync(photoId);
        if (photo != null)
        {
            _ = track.Photos.Remove(photo);
            await _trackRepository.UpdateAsync(track);
        }
    }

    public TrackReferenceDataResponse GetReferenceData()
    {
        return new TrackReferenceDataResponse()
        {
            MinRacesToCreateTrack = _settings.MinRacesToCreate,
            DefaultTrackConfirmationThreshold =
_settings.DefaultConfirmationThreshold,
            TrackStatuses =
EnumExtensions.GetEnumValues<TrackStatus>()
                .Select(item => new EnumValueResponse() { Name
= item.Key, Value = item.Value })
                .ToList()
        };
    }

```

```
}  
}
```

Файл: UserService.cs

```
using MotorsportErp.Application.Common.Contracts;  
using MotorsportErp.Application.Common.Exceptions;  
using MotorsportErp.Application.Common.Extensions;  
using MotorsportErp.Application.Common.Interfaces.Repositories;  
using MotorsportErp.Application.Features.Cars.Mappers;  
using MotorsportErp.Application.Features.Tournaments.Mappers;  
using MotorsportErp.Application.Features.Users.Contracts;  
using MotorsportErp.Application.Features.Users.Interfaces;  
using MotorsportErp.Application.Features.Users.Mappers;  
using MotorsportErp.Domain.Files;  
using MotorsportErp.Domain.Tournaments;  
using MotorsportErp.Domain.Tracks;  
using MotorsportErp.Domain.Users;  
using System.Linq.Expressions;  
  
namespace MotorsportErp.Application.Features.Users.Services;  
  
public class UserService : IUserService  
{  
    private readonly IUserRepository _userRepository;  
    private readonly IFileRepository _fileRepository;  
  
    public UserService(IUserRepository userRepository,  
IFileRepository fileRepository)  
    {  
        _userRepository = userRepository;  
        _fileRepository = fileRepository;  
    }  
  
    public async Task<UserResponse> GetByIdAsync(Guid id)  
    {  
        var user = await _userRepository.GetByIdAsync(id) ??  
throw new KeyNotFoundException("User not found");  
        return UserMapper.ToResponse(user);  
    }  
  
    public async Task<PublicUserProfileResponse>  
GetPublicProfileAsync(Guid id)  
    {  
        var user = await _userRepository.GetByIdAsync(id) ??  
throw new KeyNotFoundException("User not found");  
        return BuildPublicProfile(user);  
    }  
  
    public async Task<UserProfileResponse> GetProfileAsync(Guid  
id)
```

```

        {
            var user = await _userRepository.GetByIdAsync(id) ??
throw new KeyNotFoundException("User not found");
            return BuildProfile(user);
        }

        public async Task<PagedResponse<UserResponse>>
GetAllAsync(string? search, int page, int pageSize)
        {
            Expression<Func<User, bool>>? filter = null;

            if (!string.IsNullOrEmpty(search))
            {
                filter = u => u.Nickname.Contains(search);
            }

            var (users, totalCount) = await
userRepository.GetPagedAsync(filter, page, pageSize);

            return new PagedResponse<UserResponse>
            {
                Items =
users.Select(UserMapper.ToResponse).ToList(),
                TotalCount = totalCount,
                Page = page,
                PageSize = pageSize
            };
        }

        public async Task<PagedResponse<UserAdminResponse>>
GetAllAdminAsync(int page, int pageSize)
        {
            var (users, totalCount) = await
userRepository.GetPagedAsync(null, page, pageSize);

            return new PagedResponse<UserAdminResponse>
            {
                Items =
users.Select(UserMapper.ToAdminResponse).ToList(),
                TotalCount = totalCount,
                Page = page,
                PageSize = pageSize
            };
        }

        public async Task AssignRoleAsync(Guid adminId, Guid
targetUserId, UserRole role)
        {
            if (role == UserRole.SuperAdmin)
            {
                throw new InvalidOperationException("Cannot assign
SuperAdmin role");
            }
        }

```

```

    }

    var admin = await _userRepository.GetByIdAsync(adminId)
    ?? throw new KeyNotFoundException("Admin not found");
    if (!admin.Roles.HasFlag(UserRole.SuperAdmin))
    {
        throw new ForbiddenException("Only super admin can
assign roles");
    }

    var user = await
    _userRepository.GetByIdAsync(targetUserId) ?? throw new
    KeyNotFoundException("User not found");
    if (user.Roles.HasFlag(role))
    {
        throw new InvalidOperationException("User already
has this role");
    }

    user.Roles |= role;

    await _userRepository.UpdateAsync(user);
}

public async Task UpdateProfileAsync(Guid userId,
UserUpdateRequest request)
{
    if (string.IsNullOrWhiteSpace(request.Nickname))
    {
        throw new ArgumentException("Nickname cannot be
empty");
    }

    var user = await _userRepository.GetByIdAsync(userId)
    ?? throw new KeyNotFoundException("User not found");

    user.Nickname = request.Nickname;
    user.Bio = request.Bio;

    MediaFile? file = request.Avatar != null ?
        await
        _fileRepository.GetByIdAsync(request.Avatar.Id) ?? throw new
        KeyNotFoundException("Photo not found") :
        null;
    if (file != null && file.UploadedById != userId)
    {
        throw new ForbiddenException("Only owner can use
self photos");
    }

    user.Avatar = file;
}

```

```

        await _userRepository.UpdateAsync(user);
    }

    public async Task BlockUserAsync(Guid adminId, Guid
targetUserId, bool block)
    {
        var admin = await _userRepository.GetByIdAsync(adminId)
?? throw new KeyNotFoundException("Admin not found");
        if (!admin.Roles.HasFlag(UserRole.Moderator) &&
!admin.Roles.HasFlag(UserRole.SuperAdmin))
        {
            throw new ForbiddenException("No permission to
block users");
        }

        var targetUser = await
_userRepository.GetByIdAsync(targetUserId) ?? throw new
KeyNotFoundException("User not found");
        if (targetUser.Roles.HasFlag(UserRole.Moderator) ||
targetUser.Roles.HasFlag(UserRole.SuperAdmin))
        {
            throw new InvalidOperationException("Can't block
user with role Moderator or SuperAdmin");
        }

        targetUser.IsBlocked = block;
        await _userRepository.UpdateAsync(targetUser);
    }

    private UserProfileResponse BuildProfile(User user)
    {
        var publicProfile = BuildPublicProfile(user);
        var response = UserMapper.ToProfile(user);

        response.CarsCount = publicProfile.CarsCount;
        response.TournamentsCount =
publicProfile.TournamentsCount;
        response.Cars = publicProfile.Cars;
        response.CurrentTournaments =
publicProfile.CurrentTournaments;
        response.TournamentHistory =
publicProfile.TournamentHistory;
        response.Applications = BuildApplications(user);
        response.OrganizedTournaments =
user.OrganizedTournaments
            .Where(o => o.Tournament != null)
            .OrderByDescending(o => o.Tournament.StartDate)
            .Select(o =>
TournamentMapper.ToResponse(o.Tournament))
            .ToList();

        return response;
    }

```

```

    }

    private PublicUserProfileResponse BuildPublicProfile(User
user)
    {
        var response = UserMapper.ToPublicProfile(user);
        var resultLookup = user.Results
            .Where(r => r.Tournament != null)
            .GroupBy(r => r.TournamentId)
            .ToDictionary(g => g.Key, g => g.OrderBy(r =>
r.Position).First());

        response.Cars = user.Cars
            .OrderByDescending(c => c.Year)
            .ThenBy(c => c.Brand)
            .ThenBy(c => c.Model)
            .Select(CarMapper.ToResponse)
            .ToList();

        response.CurrentTournaments = user.Applications
            .Where(IsCurrentParticipation)
            .OrderBy(a => a.Tournament!.StartDate)
            .Select(a => MapTournamentEntry(a, resultLookup))
            .ToList();

        response.TournamentHistory = user.Applications
            .Where(IsHistoryParticipation)
            .OrderByDescending(a => a.Tournament!.EndDate)
            .Select(a => MapTournamentEntry(a, resultLookup))
            .ToList();

        response.CarsCount = response.Cars.Count;
        response.TournamentsCount =
response.CurrentTournaments.Count +
response.TournamentHistory.Count;

        return response;
    }

    private static List<UserTournamentEntryResponse>
BuildApplications(User user)
    {
        var resultLookup = user.Results
            .Where(r => r.Tournament != null)
            .GroupBy(r => r.TournamentId)
            .ToDictionary(g => g.Key, g => g.OrderBy(r =>
r.Position).First());

        return user.Applications
            .Where(a => a.Tournament != null && a.Car != null)
            .OrderByDescending(a => a.Tournament!.StartDate)
            .Select(a => MapTournamentEntry(a, resultLookup))

```

```

        .ToList();
    }

    private static bool
    IsCurrentParticipation(TournamentApplication application)
    {
        return application.Tournament != null &&
            application.Car != null &&
            application.Status ==
TournamentApplicationStatus.Approved &&
            application.Tournament.Status !=
TournamentStatus.Finished &&
            application.Tournament.Status !=
TournamentStatus.Cancelled;
    }

    private static bool
    IsHistoryParticipation(TournamentApplication application)
    {
        return application.Tournament != null &&
            application.Car != null &&
            application.Status ==
TournamentApplicationStatus.Approved &&
            (application.Tournament.Status ==
TournamentStatus.Finished ||
            application.Tournament.Status ==
TournamentStatus.Cancelled);
    }

    private static UserTournamentEntryResponse
    MapTournamentEntry(
        TournamentApplication application,
        IReadOnlyDictionary<Guid, TournamentResult>
resultLookup)
    {
        var result =
resultLookup.GetValueOrDefault(application.TournamentId);

        return new UserTournamentEntryResponse
        {
            ApplicationId = application.Id,
            TournamentId = application.TournamentId,
            TournamentName = application.Tournament!.Name,
            TournamentStatus = application.Tournament.Status,
            StartDate = application.Tournament.StartDate,
            EndDate = application.Tournament.EndDate,
            TrackId = application.Tournament.TrackId,
            TrackName = application.Tournament.Track?.Name ??
string.Empty,
            CarId = application.CarId,
            CarName = $"{application.Car!.Brand}
{application.Car.Model}",

```

					ИПУ 1.090301.03.001 ПЗ	Лист
						121
Изм.	Лист	№ докум.	Подпись	Дата		

```

        ApplicationStatus = application.Status,
        Position = result?.Position,
        BestLapTime = result?.BestLapTime
    };
}

public UserReferenceDataResponse GetReferenceData()
{
    var invisibleRoles = new List<string>() {
UserRole.Moderator.ToString(), UserRole.SuperAdmin.ToString() };
    return new UserReferenceDataResponse
    {
        UserRoles =
EnumExtensions.GetEnumValues<TrackStatus>()
                .Where(item =>
!invisibleRoles.Contains(item.Key))
                .Select(item => new EnumValueResponse() { Name
= item.Key, Value = item.Value })
                .ToList()
    };
}
}

```

Приложение В

(обязательное)

Информационная клиент-серверная система организации и проведения
любительских автотранспортных мероприятий

Презентация

					ПГУ 1.090301.03.001 ПЗ	Лист
						123
Изм.	Лист	№ докум.	Подпись	Дата		

Информационная клиент-серверная система организации и проведения любительских автотранспортных мероприятий

Выполнил студент группы 22ВВП1
Беляев Даниил Ильич
Научный руководитель
к.т.н. доцент кафедры ВТ Токарев А. Н.

Пенза 2026

Информационная клиент-серверная система для организации и проведения любительских автотранспортных мероприятий

Актуальность

Разобщенность организационных процессов в любительском автоспорте и отсутствие единой цифровой среды для взаимодействия пилотов и организаторов.

Цель работы

Разработка информационной клиент-серверной системы для автоматизации организации и проведения любительских автотранспортных мероприятий.

Предмет исследования

Методы, модели и программные средства автоматизации процессов регистрации, ведения каталогов трасс, формирования турниров и фиксации результатов.

Объект исследования

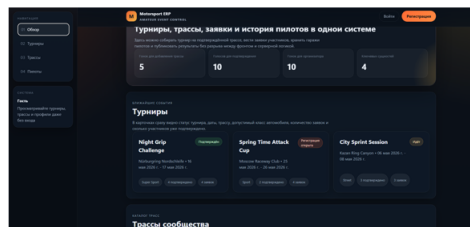
Процессы организации и проведения любительских автотранспортных мероприятий

2

Информационная клиент-серверная система для организации и проведения любительских автотранспортных мероприятий

Задачи исследования

- провести анализ предметной области любительского автоспорта;
- исследовать существующие аналоги;
- сформулировать требования к разрабатываемой системе;
- выбрать технологический стек и инструменты разработки;
- спроектировать архитектуру системы;
- реализовать программное решение;
- провести тестирование и оценку эффективности разработанной системы.



Главная страница приложения

3

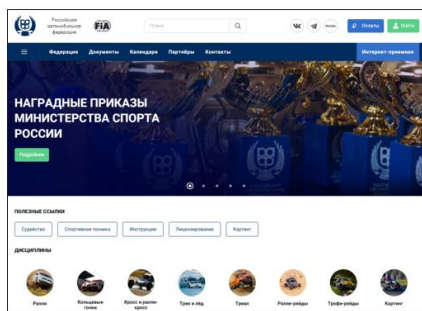
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.03.001 ПЗ

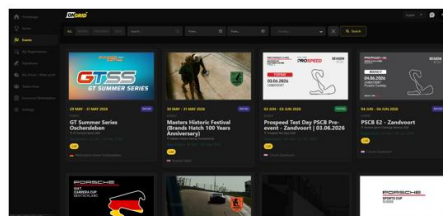
Лист

124

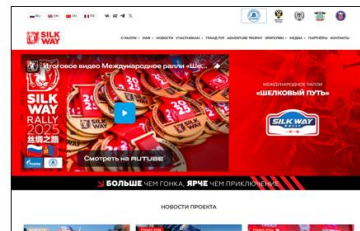
Аналоги



Российская автомобильная федерация



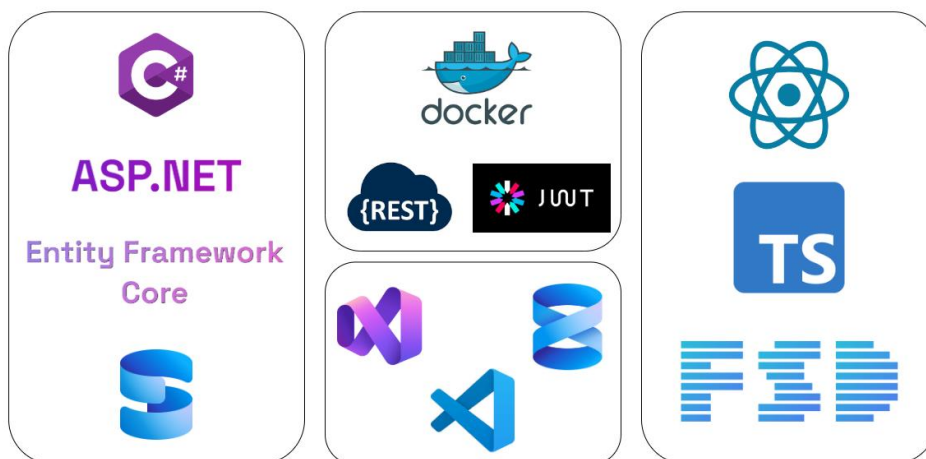
onGRID



Ралли «Шелковый путь»

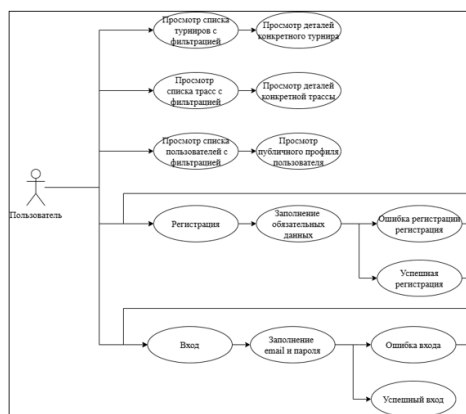
4

Технологический стек проекта

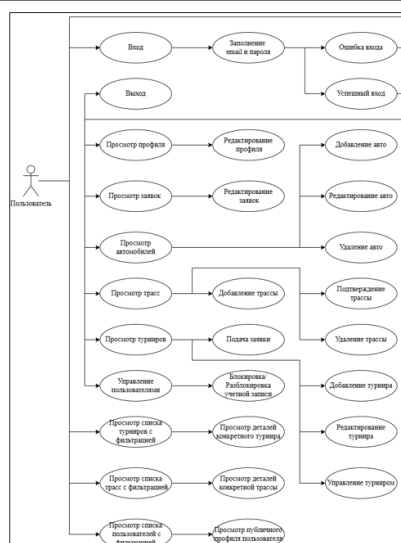


5

Варианты использования



Неавторизованный пользователь



Авторизованный пользователь с правами модератора

6

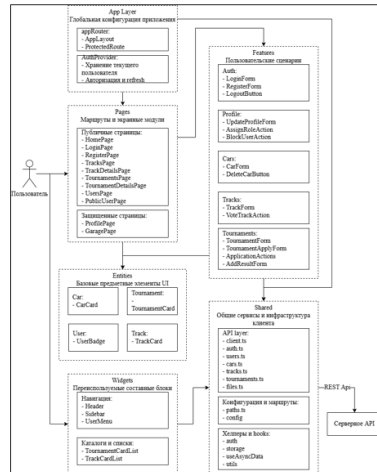
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.03.001 ПЗ

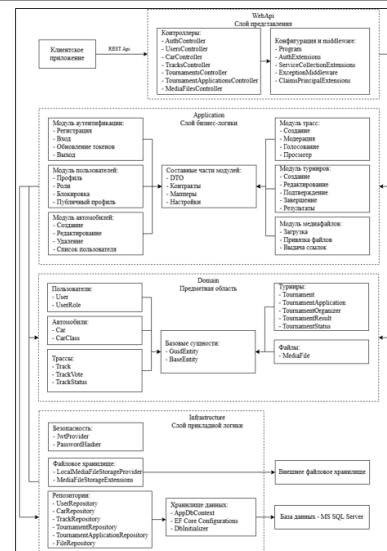
Лист

125

Архитектура системы



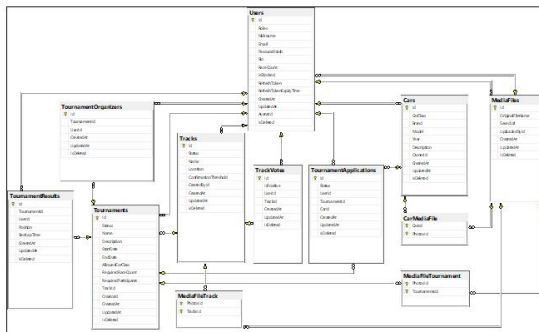
Архитектура клиента



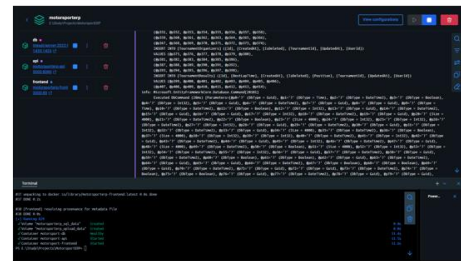
Архитектура Сервера

7

Внешние сервисы, база данных, развертывание



Структура базы данных



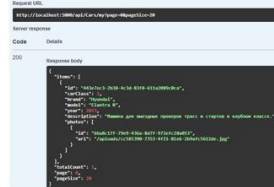
Docker



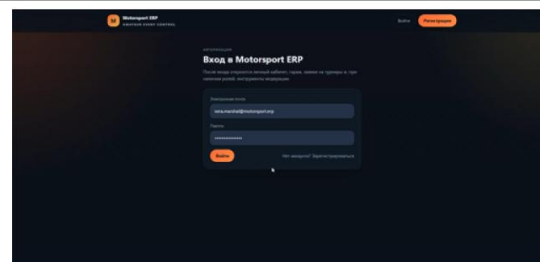
Swagger

8

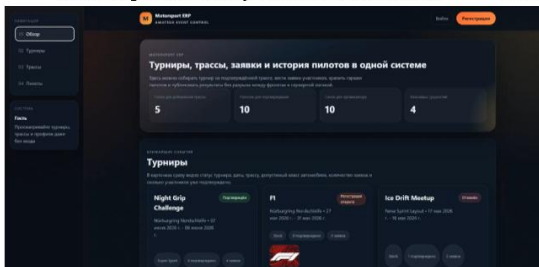
Результат работы



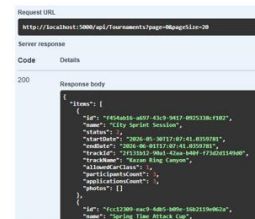
API-запрос для получения автомобилей



Авторизованный пользователь с правами модератора



Неавторизованный пользователь



API-запрос для получения турниров

9

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

ПГУ 1.090301.03.001 ПЗ

Лист

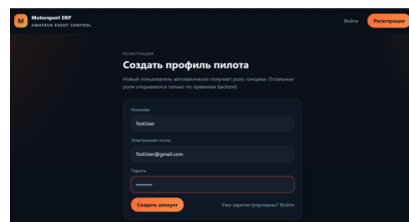
126

Заключение

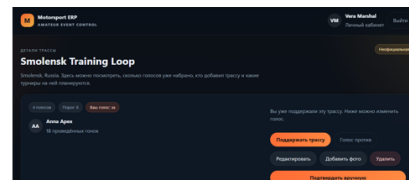
Разработана клиент-серверная информационная система для организации и проведения любительских автоспортивных мероприятий.

Реализованы основные бизнес-модули: авторизация, профиль пилота, виртуальный гараж, работа с трассами, турнирами, заявками, результатами и медиаконтентом.

Проведено тестирование, подтвердившее соответствие разработанного приложения поставленным требованиям.



Страница регистрации



Страница трассы

10

Спасибо за внимание!

Готов ответить на ваши вопросы!

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.03.001 ПЗ

Лист

127