

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

Специальность: 09.05.01 «Применение и эксплуатация
автоматизированных систем специального назначения»

Специализация: «Эксплуатация вычислительных машин, комплексов,
систем и сетей специального назначения»

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ИНФОРМАЦИОННАЯ СИСТЕМА
АВТОМАТИЗАЦИИ ОБРАЗОВАТЕЛЬНОГО
ПРОЦЕССА ВОЕННО-УЧЕБНОГО ЦЕНТРА

Студент

Байнов Р.С.
(подпись, дата)

Байнов Р.С.

Руководитель

Митрохин М.А.
(подпись, дата)

Митрохин М.А.

Нормоконтролёр

А.С. Бычков
(подпись, дата)

А.С. Бычков

Рецензент

зав. кафедрой Программного обеспечения
Пенз. ГТУ М.А.И. Мартышкин
(должность, место работы) (подпись, дата)

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

Заведующий кафедрой

М.А. Митрохин
(подпись)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от 17.06.2026 г. № 14)

Секретарь ГЭК

Гурин Е.И.
(подпись)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Кафедра

вычислительной техники

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

« 11 » 02 2026 г.

ЗАДАНИЕ

НА ДИПЛОМНЫЙ ПРОЕКТ

1. Студент Байнов Руслан Сергеевич гр.21ВОЭ1 факультета Вычислительной техники

2. Тема работы Информационная система автоматизации образовательного процесса
военно-учебного центра

Тема утверждена приказом ПГУ № 122/о от "11.02.2026" 2026 г.

3. Руководитель работы д.т.н., заведующий кафедры ВТ Митрохин М.А.

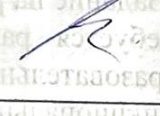
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Требуется разработать веб-ориентированную информационную систему автоматизации образовательного процесса военно-учебного центра (ВУЦ), соответствующую функциональным и техническим требованиям. Система должна обеспечить: ведение расписания занятий, учёт посещаемости и успеваемости курсантов, фиксацию дисциплинарных показателей, хранение учебных материалов, управление личным составом (ведение личных дел), оформление и учёт приказов, формирование регламентированной отчетности, а также предоставление удобных личных кабинетов для курсантов, преподавателей и командиров подразделений с разграничением прав доступа.

5. Перечень подлежащих разработке вопросов

Необходимо провести анализ предметной области — изучить деятельность ВУЦ и действующие процессы учёта (посещаемость, успеваемость, дисциплина, приказы), а также проанализировать существующие аналоги (Moodle, «Дневник.ру», «Сетевой город. Образование») и выявить их ограничения для ВУЦ. Далее следует сформулировать функциональные и нефункциональные требования к системе, включая ролевую модель доступа и критерии безопасности, после чего спроектировать архитектуру системы: структуру базы данных (ER-диаграмма), API (endpoints, методы) и прототипы интерфейса. Затем нужно реализовать основные модули — авторизацию/аутентификацию, управление расписанием, учёт посещаемости/успеваемости/дисциплины, документооборот, хранение материалов и отчётность, — интегрировать компоненты (frontend ↔ backend через REST API, подключение к PostgreSQL) и, наконец, протестировать систему на соответствие требованиям, производительность и безопасность, устранив выявленные ошибки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	15	09.02.2026 – 05.03.2026	
2	Анализ требований	5	06.03.2026 – 24.03.2026	
3	Выбор технологий, применяемых при проектировании и разработке	10	25.03.2026 – 10.04.2026	
4	Описание алгоритмов	20	11.04.2026 – 22.04.2026	
5	Разработка приложения	60	23.04.2026 – 11.05.2026	
6	Оформление пояснительной записки	15	12.05.2026 – 21.05.2026	

Дата выдачи задания "11" "02" 2026 г.

Руководитель работы

(подпись, дата)

Задание к исполнению принял студент

(подпись, дата)

Работу к защите допустить
Декан факультета

15.06.2026

(подпись, дата)

Реферат

Бакалаврская работа содержит 126 листов, 24 рисунка, 1 таблица, 12 источников и 3 приложения.

ИНФОРМАЦИОННАЯ СИСТЕМА, ВОЕННО-УЧЕБНЫЙ ЦЕНТР, АВТОМАТИЗАЦИЯ, УЧЁТ ЛИЧНОГО СОСТАВА, FASTAPI, REACT, POSTGRESQL

Цель работы — разработать веб-ориентированную информационную систему, предназначенную для автоматизации образовательного процесса военно-учебного центра. Система обеспечивает ведение расписания занятий, учёт посещаемости, фиксацию результатов успеваемости, хранение учебных материалов, управление личным составом и формирование отчётности, а также предоставляет курсантам и преподавателям удобный интерфейс для взаимодействия с учебным подразделением.

В процессе выполнения работы проведён анализ предметной области, изучены существующие программные решения, определены их сильные и слабые стороны. На основе анализа сформулированы функциональные и нефункциональные требования к разрабатываемой системе. Выполнено проектирование архитектуры, разработана структура базы данных, реализованы основные модули: авторизация и аутентификация пользователей, личные кабинеты курсантов, преподавателей и командиров, управление расписанием, посещаемостью, успеваемостью, дисциплинарными показателями и учебной документацией.

Для реализации использованы современные технологии: FastAPI в качестве backend-фреймворка, PostgreSQL в качестве системы управления базами данных, React для разработки клиентской части. Архитектура системы обеспечивает масштабируемость, безопасность хранения данных и удобство дальнейшего расширения функционала.

					ПГУ 1.090501.02.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Информационная система автоматизации образовательного процесса военно-учебного центра Пояснительная записка	Лит.	Лист	Листов
Разраб.		Байнов Р.С.		04.06.24				
Провер.		Митрохин М.А.		08.06.24			2	140
Ил. Контр.		Бычков А.С.		08.06.24		ФВТ, каф. ВТ гр. 21ВОЭ1		
Утверд.		Митрохин М.А.		09.06.24				

Содержание

Перечень сокращений и обозначений.....	5
Введение	6
1 Анализ предметной области.....	8
1.1 Общая характеристика учреждения здравоохранения	8
1.1.1 Анализ существующих аналогов	10
1.1.2 Обоснование актуальности разработки программы.....	14
1.1.3 Постановка задачи	16
1.1.4 Выбор технологий и инструментов	19
1.1.5 Выбор серверной платформы	19
1.1.6 Выбор системы управления базами данных	20
1.1.7 Выбор технологий клиентской части	20
1.1.8 Формат обмена данными и взаимодействие компонентов.....	21
1.2 Проектирование и разработка	22
1.2.1 Проектирование базы данных	23
1.2.2 Разработка серверной части	35
1.2.3 Разработка клиентской части	40
1.2.4 Интеграция и взаимодействие компонентов	47
1.2.5 Проектирование пользовательского интерфейса	27
1.2.6 Динамика работы приложения.....	52
1.2.7 Тестирование приложения	54
2 Безопасность жизнедеятельности	58
2.1 Характеристика рабочего места разработчика.....	58

2.2 Пожарная безопасность	59
2.3 Психофизиологические нагрузки и профилактика утомления.....	60
3 Экономика.....	61
Заключение.....	62
Список использованных источников	59
Приложение А – UML-диаграммы.....	60
Приложение Б – Листинг программы.....	65
Приложение В – Презентация	132

Перечень сокращений и обозначений

В данной работе используются следующие сокращения и обозначения.

API (Application Programming Interface) – программный интерфейс для взаимодействия между клиентской и серверной частями системы

CRUD (Create, Read, Update, Delete) – базовые операции над данными в информационных системах

DB (DataBase) – база данных.

JSON (JavaScript Object Notation) – текстовый формат обмена данными, который используется в качестве сохранения игровых настроек и прогресса

DTO (Data Transfer Object) – объект передачи данных между слоями приложения.

HTTP (HyperText Transfer Protocol) – протокол передачи данных между клиентом и сервером.

REST (Representational State Transfer) – архитектурный стиль построения веб-сервисов.

React – библиотека JavaScript для создания пользовательских интерфейсов.

UML (Unified Modeling Language) – унифицированный язык моделирования.

FastAPI – фреймворк для разработки серверной части на Python.

SSL (Secure Sockets Layer) – технология шифрования данных при передаче.

TLS (Transport Layer Security) – протокол защиты данных, используемый в HTTPS.

IP (Internet Protocol) – сетевой протокол, используемый для адресации устройств.

UX (User Experience) – пользовательский опыт, удобство взаимодействия с системой.

Введение

Современная система подготовки военнослужащих предъявляет высокие требования к организации учебного процесса, качеству управления личным составом и оперативности обработки информации. Военно-учебные центры выполняют важную роль в формировании кадрового резерва и обеспечении подготовки специалистов, способных эффективно выполнять задачи в условиях, требующих дисциплины, точности и высокой ответственности. При этом объём данных, связанных с учебной деятельностью, постоянно растёт: необходимо учитывать посещаемость, успеваемость, дисциплинарные показатели, результаты контрольных мероприятий, приказы, распоряжения и множество других сведений, которые должны быть доступны в актуальном и структурированном виде.

Традиционные методы ведения документации — бумажные журналы, разрозненные таблицы и устаревшие локальные программы — уже не соответствуют современным требованиям. Они усложняют обработку информации, увеличивают риск ошибок, затрудняют взаимодействие между подразделениями и не позволяют оперативно формировать отчётность. В условиях, когда учебный процесс требует высокой степени координации и прозрачности, отсутствие единой информационной среды становится серьёзным препятствием для эффективной работы военно-учебного центра.

Цифровая трансформация образовательной сферы делает внедрение современных информационных систем необходимым условием повышения качества подготовки курсантов. Такие системы позволяют автоматизировать ключевые процессы, обеспечить доступ к актуальным данным, снизить нагрузку на преподавателей и командиров подразделений, а также повысить точность и оперативность принятия управленческих решений. Кроме того, использование современных технологий способствует улучшению взаимодействия между участниками образовательного процесса и повышает уровень дисциплины и контроля.

					ПГУ 1.090501.02.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Несмотря на наличие широкого спектра образовательных платформ, большинство из них ориентированы на гражданские учебные заведения и не учитывают специфику военной подготовки. Они не поддерживают работу с личным составом, дисциплинарными показателями, приказами и специализированной отчётностью, что делает их применение в военно-учебных центрах ограниченным. Это создаёт необходимость разработки специализированной информационной системы, адаптированной под реальные процессы и требования военной подготовки.

Целью данной работы является создание веб-ориентированной информационной системы, обеспечивающей автоматизацию учебной деятельности военно-учебного центра, повышение эффективности управления личным составом и улучшение качества взаимодействия между курсантами, преподавателями и командирами подразделений. В рамках исследования проведён анализ предметной области, изучены существующие решения, определены требования к системе, выполнено проектирование архитектуры, разработана структура базы данных и реализованы основные функциональные модули.

1 Анализ предметной области

1.1 Общая характеристика военно-учебного центра

Деятельность военно-учебного центра формируется на основе нормативных документов Министерства обороны и Министерства науки и высшего образования, которые определяют структуру подготовки, порядок ведения документации, требования к учёту личного состава и регламенты организации учебного процесса. ВУЦ работает в соответствии с Положением о военно-учебных центрах, утверждённым приказом Министерства обороны РФ № 565, где подробно описаны функции центра, порядок комплектования учебных взводов, требования к ведению журналов посещаемости, правила оформления приказов по личному составу и структура отчётности. В частности, в пункте 18 данного Положения прямо указано, что командиры подразделений обязаны ежедневно фиксировать посещаемость занятий и своевременно передавать сведения в учебный отдел, а в пункте 32 регламентируется порядок хранения учебной документации, включая журналы, ведомости и приказы.

Организация учебного процесса строится на основе учебных планов, утверждённых Министерством обороны, которые определяют объём теоретической и практической подготовки, последовательность дисциплин и требования к итоговой аттестации. В учебных планах, например в типовой программе подготовки офицеров запаса, содержатся конкретные нормы: количество часов строевой подготовки, огневой подготовки, тактики, военной медицины и других дисциплин, а также требования к проведению контрольных занятий и зачётов. Эти документы определяют структуру расписания, порядок распределения преподавателей и использование учебных аудиторий и полигонов.

Работа с личным составом также строго регламентирована. В соответствии с пунктом 41 Положения о ВУЦ, на каждого обучающегося ведётся личное дело, включающее сведения о прохождении подготовки, дисциплинарных взысканиях, результатах контрольных мероприятий и итоговой аттестации. В приказах по личному составу, которые оформляются в соответствии с требованиями

					ПГУ 1.090501.02.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

делопроизводства Министерства обороны, фиксируются назначения командиров отделений, отчисления, восстановления, дисциплинарные меры и допуски к итоговой аттестации. Эти документы являются обязательными для ведения и подлежат строгому учёту.

Контроль посещаемости и дисциплины занимает особое место в деятельности ВУЦ. Согласно пункту 27 Положения, преподаватели обязаны вести журналы установленного образца, где фиксируются результаты занятий, посещаемость и успеваемость. Журналы имеют унифицированную структуру: разделы для отметок о присутствии, темы занятий, оценок, подписи преподавателя и командира подразделения. В ряде ВУЦ используются также внутренние регламенты, например «Инструкция по ведению журнала посещаемости», где подробно описано, как фиксируются пропуски по уважительной и неуважительной причине, как оформляются служебные записки и какие сроки установлены для передачи данных в учебный отдел.

Учебная документация включает методические материалы, планы занятий, приказы, распоряжения и отчёты. В соответствии с пунктом 35 Положения о ВУЦ, все документы должны храниться в структурированном виде, обеспечивающем быстрый доступ и исключающем возможность утраты. На практике это означает ведение папок с приказами, журналов учёта, ведомостей итоговой аттестации, а также электронных таблиц, которые используются для оперативного обмена данными между преподавателями и командирами подразделений. Например, в большинстве ВУЦ существует «Журнал учёта дисциплинарных взысканий», где фиксируются нарушения, даты, основания и принятые меры, а также «Книга приказов по учебной части», в которой регистрируются все распоряжения, касающиеся учебного процесса.

Анализ официальных документов показывает, что информационные потоки ВУЦ формируются вокруг нескольких ключевых направлений: личный состав, расписание, посещаемость, успеваемость, дисциплина и учебные материалы. Каждое направление имеет собственные регламенты, сроки обновления данных и требования к оформлению документов. Например, сведения о посещаемости

					ПГУ 1.090501.02.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

должны передаваться ежедневно, отчёты по итогам недели — еженедельно, а итоговые ведомости — строго в сроки, установленные учебным планом. Такая структура создаёт высокую нагрузку на командиров подразделений и преподавателей, особенно при использовании бумажных журналов и разрозненных электронных файлов.

В условиях, когда учебный процесс требует точности, оперативности и прозрачности, отсутствие единой информационной системы приводит к дублированию данных, несогласованности сведений и увеличению риска ошибок. Нормативные документы прямо не запрещают использование цифровых систем, но и не предоставляют готовых решений, что делает разработку специализированной информационной системы логичным и необходимым шагом. Такая система должна соответствовать требованиям приказов Министерства обороны, обеспечивать структурированное хранение данных, поддерживать ролевую модель доступа и позволять автоматизировать процессы, которые в официальных документах описаны как обязательные для ежедневного ведения.

Строгая регламентация, закреплённая в федеральных и ведомственных документах, показывает, что работа военно-учебного центра основана на постоянном учёте, контроле и оформлении данных, что требует высокой точности и значительных трудозатрат. Бумажные журналы, книги приказов и служебные записки создают риск ошибок и замедляют обработку информации. В таких условиях специализированная информационная система становится необходимым инструментом, позволяющим привести процессы учёта в соответствие с нормативными требованиями и повысить эффективность управления учебным процессом.

1.1.1 Анализ существующих аналогов

На рынке представлены различные образовательные информационные системы, однако большинство из них ориентированы на гражданские учебные заведения и не учитывают особенности военной подготовки. Анализ существующих решений позволяет определить их сильные стороны, ограничения

и степень соответствия требованиям военно-учебного центра.

Платформа Moodle действительно предоставляет широкий набор инструментов для дистанционного обучения, однако при сравнении с потребностями военно-учебного центра становится очевидно, что её функциональность не покрывает значительную часть процессов, закреплённых в нормативных документах Министерства обороны. Moodle ориентирована на гражданские образовательные программы и не предусматривает механизмов, необходимых для ведения учёта личного состава, оформления приказов, регистрации дисциплинарных взысканий, учёта нарядов, построений, допуска к стрельбам или полевым занятиям. В отличие от требований, описанных в Положении о ВУЦ, система не поддерживает работу с журналами установленного военного образца, не обеспечивает фиксацию ежедневной явки в формате, предусмотренном «Журналом учёта личного состава», и не позволяет оформлять служебные записки и рапорты в соответствии с регламентами строевой части. Кроме того, Moodle не содержит встроенных инструментов для формирования отчётности, соответствующей приказам Министерства обороны, например ведомостей итоговой аттестации по военной подготовке или отчётов по дисциплинарной практике. Всё это делает платформу непригодной для полноценного использования в военно-учебных центрах, где требуется строгая регламентация данных и поддержка специфических процессов, отсутствующих в гражданских образовательных системах.

Внешний вид приложения представлен на рисунке 1.

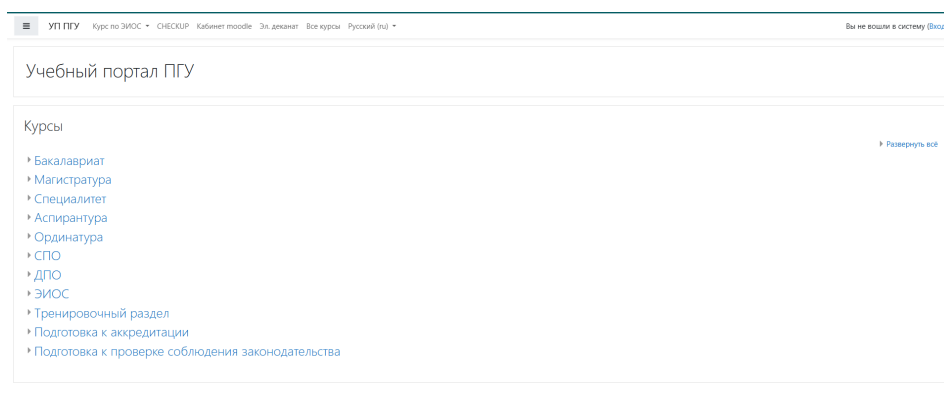


Рисунок 1 – Внешний вид «Moodle»

Система «Дневник.ру» действительно хорошо подходит для школ и колледжей, однако при сравнении с требованиями военно-учебных центров становится очевидно, что её функциональность не покрывает значительную часть процессов, закреплённых в официальных документах Министерства обороны. Платформа не поддерживает работу с личными делами обучающихся в том виде, в котором они ведутся в ВУЦ, не предусматривает фиксацию ежедневной явки в формате, установленном «Журналом учёта личного состава», и не позволяет оформлять служебные записки, рапорты и приказы по строевой части. В отличие от требований, описанных в Положении о ВУЦ, система не обеспечивает регистрацию дисциплинарных взысканий, учёт нарядов, построений, допусков к стрельбам и полевым занятиям. Кроме того, «Дневник.ру» не формирует отчётность, соответствующую военным нормативам, например ведомости итоговой аттестации по военной подготовке или отчёты по дисциплинарной практике. Всё это делает платформу непригодной для использования в военно-учебных центрах, где требуется строгая регламентация данных и поддержка специфических процессов, отсутствующих в гражданских образовательных системах.

Внешний вид приложения представлен на рисунке 2.

Ученик	1				2				Ср.
	ОТБ	ПОВ	ДЗ	КР	ОТБ	ПОВ	СР	КР	
Брунн Кристи	2		5	3				2	3.62
Васильев Артём	5	10.5	5	5	5	5	5		5.00
Васильев Дмитрий	5		5	3	4			2	3.87
Васильев Мария	5		5	5		4		2	4.17
Васильев Ольга	7		5	5				2	4.00
Васильев Артём	5	10.5	5	5		5			3.80
Васильев Игорь				5					5.00
Васильев Глеб	4		5	5				2	3.57
Васильев Анна	5		5	5				2	4.07
Васильев Светлана	4		1-	5				2	3.44

Рисунок 2 – Внешний вид «Дневник.ру»

«Сетевой город. Образование» представляет собой комплексную систему для автоматизации учебной деятельности школ и профессиональных образовательных организаций. Она включает электронные журналы, расписания, отчётность и инструменты для взаимодействия между участниками образовательного процесса. Несмотря на развитый функционал, система не поддерживает работу с личным составом, дисциплинарными показателями, приказами, внутренними распоряжениями и специализированной отчётностью, характерной для военно-учебных центров. Отсутствие механизмов учёта специфики военной подготовки делает её применение в таких условиях ограниченным.

Внешний вид приложения представлен на рисунке 3.

Класс\Параллель		Предмет		Период		Учитель															
7а		Алгебра		3 четверть		Комиссарова Эл															
Ученики	Январь																				
	13	13	15	15	16	20	20	22	22	23	27	27	29	29	30	3	3	5	5	6	10
	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☐	☐	☐	☐	☐	☐	☐	☐
1. Адайкина Диана		4	5	5			5	5				5		УП							
2. Алиева Анастасия	4						4	3		4	УП	УП	УП	УП							
3. Артюхина Алена		3	4		УП	УП	УП	4				4	УП								
4. Аспанов Эдгар			3			4															
5. Глотов Даниил		5		5			5	5		4	УП	УП	УП	УП							
6. Голованёв Рушан	5		3				5	5				5	4								
7. Губатенко Мария	4		4				4	3		5		4	2								
8. Догадин Даниил		2				5	4	4		4		4	2								
9. Иванов Денис	УП	УП	УП	УП			2	3				5	4	УП							
10. Калмыков Сергей		5		4			4	5		5		4	5	УП							
11. Коротин Алексей		3		2			3	2				4	4								
12. Куруков Денис		2	УП	УП			3	4				3	4								
13. Лодиков Андрей		5		3			4	4				4	5								
14. Марамзина Екатерина						4	5	5			УП	УП	УП	УП							

Рисунок 3 – Внешний вид «Сетевой город. Образование»

В таблице 1 представлен анализ аналогов.

Таблица 1 – Анализ аналогов

Название системы	Основные возможности	Преимущества	Недостатки
Moodle	Электронные журналы, расписание, учебные материалы, отчётность, взаимодействие преподавателей и студентов	Широкий функционал, развитая инфраструктура, поддержка крупных образовательных организаций	Высокая стоимость внедрения, сложность адаптации, избыточность для военно-учебных центров
Дневник.ру	Электронный журнал, расписание, успеваемость, обмен сообщениями	Простота использования, доступность, удобный интерфейс	Ориентация на школы; отсутствие инструментов для работы с личным составом и дисциплиной
Сетевой город. Образование	Учёт успеваемости, расписание, отчётность, взаимодействие участников образовательного процесса	Комплексность, интеграция с образовательными учреждениями	Не поддерживает учёт приказов, дисциплинарных показателей, особенностей военной подготовки

Название системы	Основные возможности	Преимущества	Недостатки
Информационная система управления образовательным процессом военно-учебного центра	Учёт личного состава, расписание, посещаемость, успеваемость, дисциплина, приказы, отчётность, учебные материалы	Адаптация под специфику ВУЦ, современная архитектура, гибкость, возможность расширения	Требуется разработки и внедрения, отсутствует промышленная эксплуатация на момент создания

1.1.2 Обоснование актуальности разработки программы

Актуальность разработки информационной системы для военно-учебного центра обусловлена современными требованиями к организации учебного процесса, повышением объёмов данных и необходимостью строгого контроля подготовки личного состава. Традиционные методы ведения документации, основанные на бумажных журналах, разрозненных таблицах и устаревших локальных решениях, уже не обеспечивают необходимого уровня оперативности, точности и доступности информации. Это приводит к увеличению нагрузки на преподавателей и командиров подразделений, усложняет обработку данных, повышает риск ошибок и затрудняет контроль дисциплины и успеваемости.

Военно-учебные центры работают с большим количеством сведений: данными о личном составе, расписаниями занятий, результатами контрольных мероприятий, посещаемостью, дисциплинарными показателями, приказами и отчётностью. При отсутствии единой информационной среды эти данные оказываются распределёнными между различными подразделениями, что

приводит к несогласованности информации, дублированию записей и снижению эффективности управления. В условиях, когда требуется оперативное принятие решений и строгий контроль выполнения учебных программ, подобные проблемы существенно осложняют работу центра.

Дополнительным фактором, определяющим актуальность разработки, является необходимость обеспечения безопасности персональных данных и соблюдения нормативных требований, регулирующих их обработку. Использование устаревших или фрагментированных решений не позволяет обеспечить должный уровень защиты, что создаёт риски утечки информации и нарушения регламентов. Современная информационная система должна обеспечивать надёжное хранение данных, разграничение прав доступа, контроль действий пользователей и соответствие действующим стандартам.

Важную роль играет и повышение требований к прозрачности и управляемости учебного процесса. Командирам подразделений необходимы инструменты для оперативного анализа посещаемости, дисциплины и успеваемости, формирования отчётности и контроля выполнения учебных планов. Преподавателям требуется удобный механизм ведения журналов и фиксации результатов занятий. Курсанты ожидают доступ к актуальному расписанию, учебным материалам и результатам обучения. Отсутствие таких возможностей снижает эффективность подготовки и усложняет взаимодействие между участниками образовательного процесса.

Анализ существующих решений показывает, что большинство из них ориентированы на гражданские учебные заведения и не учитывают специфику военной подготовки. Они либо избыточны по функциональности и стоимости, либо недостаточно гибки и не позволяют адаптировать систему под реальные процессы военно-учебного центра. Это подтверждает необходимость разработки специализированного программного продукта, который сочетал бы современную архитектуру, удобный интерфейс, возможность масштабирования и поддержку ключевых процессов подготовки курсантов.

1.1.3 Постановка задачи

Постановка задачи определяет направления разработки и формирует требования к создаваемой информационной системе. На основании анализа деятельности военно-учебного центра можно выделить ряд проблем, связанных с отсутствием единого цифрового пространства, сложностью обработки данных о личном составе и учебном процессе, а также недостаточной оперативностью обмена информацией между подразделениями. Для решения этих проблем требуется разработать программный продукт, который обеспечит автоматизацию ключевых процессов подготовки курсантов и упростит работу преподавателей и командиров.

Основная задача разработки заключается в создании системы, позволяющей централизованно хранить и обрабатывать данные о курсантском составе, расписаниях занятий, посещаемости, результатах контрольных мероприятий, дисциплинарных показателях и учебных материалах. Система должна предоставить удобные инструменты для преподавателей, включая ведение электронных журналов, фиксацию результатов занятий, формирование отчётности и управление учебными материалами. Курсанты должны иметь доступ к актуальному расписанию, учебным материалам и результатам успеваемости. Командиры подразделений должны получить возможность контролировать дисциплину, отслеживать посещаемость, формировать отчёты и управлять личным составом.

При постановке задачи также учитываются требования к архитектуре и технологической основе будущей системы. Программный продукт должен быть реализован в виде веб-ориентированного решения, обеспечивающего доступ с различных устройств и возможность масштабирования. Особое внимание необходимо уделить вопросам безопасности: защите персональных данных, разграничению прав доступа, контролю действий пользователей и предотвращению несанкционированного вмешательства. Система должна работать стабильно при увеличении количества пользователей и объёма данных, обеспечивая высокую производительность и отказоустойчивость.

					ПГУ 1.090501.02.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		17

Таким образом, задача разработки состоит в создании современной информационной системы, способной повысить эффективность управления учебным процессом, улучшить качество подготовки курсантов и обеспечить прозрачность взаимодействия между участниками образовательной деятельности. Реализация проекта предполагает последовательное выполнение этапов анализа, проектирования, разработки, тестирования и внедрения программного решения.

UML-диаграмма вариантов использования представлена на рисунке 4.

Для уточнения логики взаимодействия пользователей с системой и детализации требований были разработаны пользовательские сценарии (User Flow). Они описывают пошаговое выполнение ключевых задач с точки зрения конечного пользователя, что позволяет выявить потенциальные проблемы интерфейса и логики до начала разработки.

Сценарий 1: Отметка посещаемости занятия преподавателем

Цель: зафиксировать список присутствующих и отсутствующих курсантов на проведённом занятии.

Участник: преподаватель.

Предусловие: пользователь авторизован в системе с ролью «Преподаватель».

Основной поток:

- преподаватель переходит в раздел «Посещаемость»;
- система отображает выпадающий список учебных групп, закреплённых за преподавателем. Преподаватель выбирает нужную группу;
- после выбора группы становится доступен список дисциплин. Преподаватель выбирает дисциплину, по которой проводилось занятие;
- система загружает таблицу посещаемости, где строки соответствуют курсантам группы, а столбцы — датам занятий по выбранной дисциплине;
- преподаватель проставляет отметки о присутствии, устанавливая флажки в соответствующих ячейках таблицы;

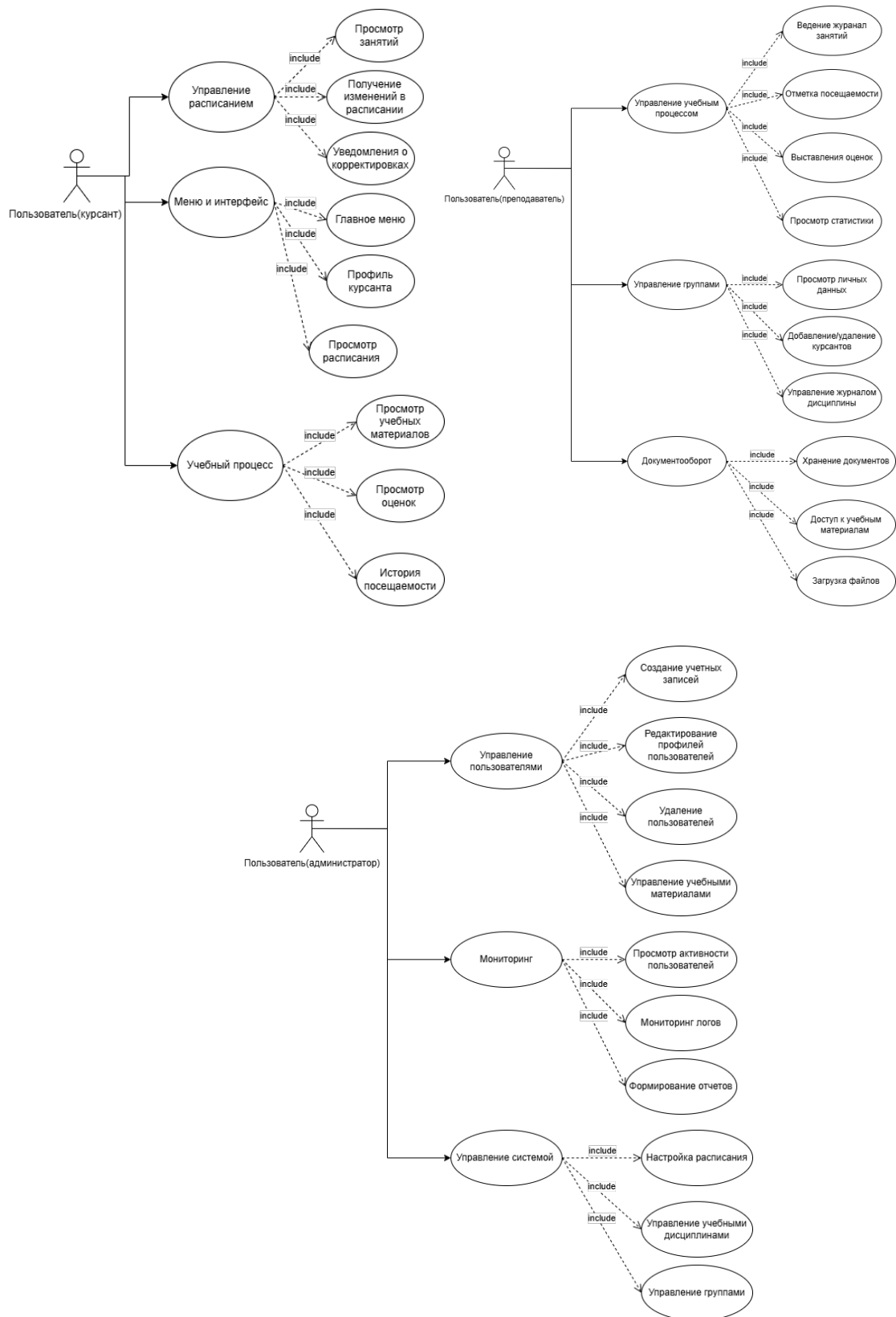


Рисунок 4 – UML-диаграмма вариантов использования

- преподаватель нажимает кнопку «Сохранить посещаемость»;
- система отправляет данные на сервер, сохраняет их в базе данных и выводит уведомление «Посещаемость сохранена».

Сценарий 2: Внесение дисциплинарного взыскания командиром

Цель: зафиксировать факт нарушения дисциплины курсантом и принятые меры.

Участник: командир подразделения.

Предусловие: пользователь авторизован в системе с ролью «Командир».

Основной поток:

- командир переходит в раздел «Дисциплина».
- система отображает список курсантов его подразделения. Командир выбирает нужного курсанта.
- командир нажимает кнопку «Добавить запись» (или использует форму быстрого добавления, если она предусмотрена интерфейсом).
- система открывает форму создания дисциплинарной записи. Командир заполняет обязательные поля: тип нарушения, дата, описание проступка и принятые меры.
- командир нажимает кнопку «Сохранить».
- система выполняет валидацию данных и при успешной проверке создаёт новую запись в «Журнале дисциплины», которая сразу отображается в общем списке.

Альтернативный поток: если заполнены не все обязательные поля или нарушены форматы данных, система подсвечивает поля с ошибками и выводит сообщение с указанием причины. Запись не сохраняется до исправления ошибок.

1.1.4 Выбор технологий и инструментов

Выбор технологий и инструментов является ключевым этапом разработки информационной системы, поскольку от него напрямую зависят производительность, масштабируемость, удобство сопровождения и возможность дальнейшего расширения функционала. Система военно-учебного центра должна обеспечивать стабильную работу при увеличении количества

пользователей, безопасное хранение данных, удобный веб-интерфейс и возможность интеграции с внешними сервисами. Кроме того, важным требованием является поддержка ролевой модели доступа, обработка большого количества структурированных данных и обеспечение высокой надёжности при работе с учебной документацией.

На основании анализа требований и особенностей предметной области был выполнен выбор технологического стека, оптимального для реализации поставленных задач. Для каждого уровня системы рассматривалось несколько альтернативных решений, которые были сравнены по критериям производительности, удобства разработки, зрелости экосистемы и соответствия специфике проекта.

1.1.5 Выбор серверной платформы

Для разработки серверной части системы рассматривались несколько современных веб-фреймворков, среди которых наиболее распространёнными и технологически зрелыми являются Django, Flask, Node.js с использованием Express, а также FastAPI. Каждый из этих инструментов обладает собственными преимуществами и ограничениями, что потребовало детального анализа их возможностей с учётом специфики информационной системы военно-учебного центра.

Django представляет собой мощный и функционально насыщенный фреймворк, включающий встроенный ORM, административную панель и развитую экосистему. Однако его монолитная архитектура и ориентация на серверный рендеринг делают его менее удобным для построения лёгких и высокопроизводительных REST-ориентированных сервисов. В условиях, когда система должна обрабатывать большое количество запросов, обеспечивать гибкость маршрутизации и поддерживать асинхронные операции, использование Django привело бы к избыточной сложности и снижению эффективности разработки.

Другим вариантом являлся Flask — минималистичный Python-фреймворк, который предоставляет разработчику высокую степень свободы и позволяет

самостоятельно формировать архитектуру приложения. Flask хорошо подходит для небольших сервисов и прототипов, однако отсутствие встроенной типизации, автоматической документации и развитых средств валидации данных делает его менее удобным для разработки крупной системы, где требуется строгий контроль корректности данных, безопасность и предсказуемость поведения серверной части. В случае информационной системы военно-учебного центра использование Flask потребовало бы значительных усилий по интеграции сторонних библиотек и созданию инфраструктуры, которая в FastAPI доступна.

Также рассматривалась связка Node.js и Express, обеспечивающая высокую скорость обработки запросов и широкие возможности кастомизации. Однако отсутствие строгой типизации и необходимость ручной реализации многих механизмов — от валидации данных до структурирования API — усложняют сопровождение крупной системы, особенно в условиях, когда требуется высокая надёжность и безопасность. Использование JavaScript на сервере требует дополнительных инструментов для обеспечения типобезопасности, что увеличивает сложность проекта и повышает риск ошибок при работе с критически важными данными.

На фоне этих решений FastAPI продемонстрировал оптимальное сочетание производительности, удобства разработки и строгой организации данных. Его архитектура основана на современном асинхронном подходе, что позволяет эффективно распределять нагрузку и обеспечивать высокую отзывчивость системы даже при большом количестве одновременных пользователей. Строгая типизация, реализованная через Pydantic-модели, обеспечивает автоматическую проверку входящих данных и снижает вероятность ошибок, что особенно важно при работе с учебной документацией, личными делами, посещаемостью и дисциплинарными показателями. Автоматическая генерация интерактивной документации значительно упрощает тестирование и интеграцию клиентской части, а нативная поддержка асинхронных операций делает FastAPI удобным инструментом для построения масштабируемой архитектуры.

					ПГУ 1.090501.02.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		22

Благодаря сочетанию высокой производительности, удобства сопровождения и гибкости FastAPI оказался наиболее подходящим решением для реализации серверной части информационной системы военно-учебного центра. Он обеспечивает необходимый уровень надёжности, позволяет эффективно работать с большими объёмами данных и предоставляет возможности для дальнейшего расширения функционала без существенных изменений архитектуры.

1.1.6 Выбор системы управления базами данных

При выборе системы управления базами данных рассматривались несколько распространённых решений, применяемых в современных веб-ориентированных информационных системах. Наиболее очевидными альтернативами являлись MySQL, MongoDB и PostgreSQL, каждая из которых обладает собственными преимуществами и ограничениями. MySQL традиционно используется в веб-разработке благодаря высокой скорости выполнения простых запросов и широкому распространению, однако её возможности при работе со сложными связями, транзакциями и строгими требованиями к целостности данных ограничены.

В условиях, когда информационная система военно-учебного центра должна обеспечивать корректное хранение данных о личном составе, посещаемости, дисциплинарных показателях и учебной документации, подобные ограничения могут привести к усложнению логики приложения и повышению риска ошибок.

Другим вариантом рассматривалась документно-ориентированная СУБД MongoDB, ориентированная на гибкое хранение данных без строгой схемы. Такой подход удобен для систем, где структура данных часто меняется, однако для военно-учебного центра характерна строго регламентированная и стабильная структура информации. Личные дела, журналы посещаемости, ведомости успеваемости и приказы по личному составу требуют чётко определённых связей, строгой типизации и гарантированной целостности, что делает использование нереляционной модели данных менее подходящим. Кроме того,

отсутствие транзакционной модели уровня ACID усложняет обеспечение корректности данных при одновременной работе большого числа пользователей.

На фоне этих решений PostgreSQL продемонстрировала наибольшее соответствие требованиям проекта. Эта СУБД сочетает высокую производительность, надёжность и богатый набор возможностей, необходимых для работы со сложными структурированными данными. PostgreSQL поддерживает транзакционность, строгую целостность данных, развитую систему индексации и оптимизации запросов, что позволяет эффективно обрабатывать большие объёмы информации и обеспечивать стабильную работу системы при увеличении нагрузки. Наличие расширений, таких как pgcrypto, JSONB и механизмов полнотекстового поиска, делает PostgreSQL гибким инструментом, способным адаптироваться к различным задачам без необходимости перехода на дополнительные решения.

Благодаря сочетанию надёжности, функциональности и масштабируемости PostgreSQL стала оптимальным выбором для реализации базы данных информационной системы военно-учебного центра. Она обеспечивает безопасное хранение критически важной информации, поддерживает сложные связи между сущностями и позволяет развивать систему без ограничений, связанных с архитектурой СУБД.

1.1.7 Выбор технологий клиентской части

При выборе технологий для разработки клиентской части рассматривались несколько современных инструментов, применяемых для создания веб-интерфейсов различной сложности. Наиболее распространёнными решениями являются Angular, Vue.js и React, каждое из которых обладает собственными особенностями и преимуществами. Angular представляет собой полнофункциональный фреймворк с жёсткой структурой и строгими архитектурными требованиями, что делает его удобным для крупных корпоративных систем, но избыточным для проектов, где требуется гибкость и возможность быстрого изменения интерфейса. Его высокая сложность и большое количество встроенных механизмов увеличивают порог входа и усложняют

адаптацию под специфические требования военно-учебного центра.

Vue.js, напротив, отличается простотой освоения и лёгкостью интеграции, что делает его привлекательным для небольших и средних проектов. Однако его экосистема менее развита по сравнению с React, а количество готовых решений и библиотек для сложных интерфейсных задач ограничено. В условиях, когда требуется реализовать многофункциональные личные кабинеты, интерактивные таблицы, расписания, панели управления и системы уведомлений, использование Vue.js может привести к необходимости разработки значительной части функционала вручную.

На фоне этих альтернатив React продемонстрировал наибольшее соответствие требованиям проекта. Его компонентный подход позволяет создавать модульную структуру интерфейса, упрощающую сопровождение и расширение системы. React обеспечивает высокую производительность за счёт виртуального DOM и оптимизации повторных рендеров, что особенно важно при работе с динамическими данными — расписаниями, журналами посещаемости, списками личного состава и другими элементами, которые регулярно обновляются. Богатая экосистема, включающая инструменты управления состоянием, библиотеки визуальных компонентов и средства интеграции с REST-API, позволяет быстро разрабатывать сложные интерфейсные решения без необходимости создавать базовые механизмы с нуля.

Кроме того, React хорошо сочетается с современными инструментами сборки, такими как Webpack и Vite, а также со средствами стилизации, включая CSS-модули, Tailwind и styled-components. Это обеспечивает гибкость при разработке интерфейса и позволяет адаптировать внешний вид системы под требования образовательного учреждения. Благодаря сочетанию производительности, гибкости и развитой экосистемы React стал оптимальным выбором для реализации клиентской части информационной системы военно-учебного центра, обеспечивая удобство работы пользователей и возможность дальнейшего расширения функционала без существенных изменений архитектуры.

					ПГУ 1.090501.02.001 ПЗ	Лист
						23
Изм.	Лист	№ докум.	Подпись	Дата		

1.1.8 Формат обмена данными и взаимодействие компонентов.

При выборе формата обмена данными и архитектурного подхода к взаимодействию клиентской и серверной частей рассматривались несколько современных решений, применяемых в веб-разработке. Одним из альтернативных вариантов являлось использование XML, который долгое время был стандартом для корпоративных систем и обеспечивает строгую структуру данных. Однако его избыточная громоздкость, сложность обработки и большие накладные расходы при передаче делают его менее подходящим для динамичных веб-приложений, где важны скорость и компактность. Другим вариантом рассматривался протокол gRPC, основанный на бинарной сериализации и обеспечивающий высокую производительность при взаимодействии распределённых сервисов. Несмотря на его эффективность, использование gRPC требует более сложной инфраструктуры, а также ограничивает гибкость клиентской части, что делает его менее удобным для браузерных приложений, ориентированных на работу через стандартные HTTP-запросы.

На фоне этих решений наиболее оптимальным оказался формат JSON, который является современным стандартом обмена данными в веб-приложениях. Его компактная структура, удобочитаемость и нативная поддержка в JavaScript позволяют эффективно сериализовать и десериализовать данные, обеспечивая высокую скорость обработки запросов. Это особенно важно при передаче сведений о расписании занятий, личном составе, посещаемости, успеваемости и других элементах учебного процесса, которые регулярно обновляются и требуют оперативного отображения на клиентской стороне.

Взаимодействие между компонентами системы организовано на основе архитектурного стиля REST, который обеспечивает модульность и независимость отдельных подсистем. REST-подход позволяет формировать чётко структурированные конечные точки API, каждая из которых отвечает за выполнение конкретной операции, что упрощает масштабирование и развитие системы. Рассматривалась также архитектура GraphQL, предоставляющая

гибкость в формировании запросов и позволяющая клиенту самостоятельно определять структуру получаемых данных. Однако её использование требует более сложной серверной логики и не всегда оправдано в системах, где структура данных строго регламентирована и редко изменяется, как в случае с информационной системой военно-учебного центра.

Особое внимание уделено вопросам безопасности передачи данных. Для идентификации пользователей и контроля доступа применяется механизм токеной аутентификации на основе JWT, позволяющий надёжно подтверждать подлинность пользователя и разграничивать права доступа в соответствии с его ролью. Такой подход исключает необходимость постоянной передачи конфиденциальных данных и снижает риск несанкционированного доступа. Все запросы передаются по защищённому протоколу, что предотвращает возможность перехвата или подмены данных при взаимодействии клиента и сервера. В совокупности эти меры обеспечивают высокий уровень безопасности и соответствуют требованиям, предъявляемым к информационным системам, работающим с персональными данными и учебной документацией.

					<i>ПГУ 1.090501.02.001 ПЗ</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		25

1.2 Проектирование и разработка

Проектирование информационной системы начинается с определения её функционального назначения, требований к качеству и условий эксплуатации. На этом этапе формируется концептуальная модель решения, включающая структуру подсистем, принципы их взаимодействия и ограничения, накладываемые предметной областью. Важной задачей является обеспечение устойчивой работы системы при увеличении нагрузки, а также возможность дальнейшего развития без необходимости полной переработки архитектуры. Поэтому проектирование ориентируется на модульность, чёткое разделение ответственности и минимизацию связности между компонентами.

При разработке системы используется многоуровневый подход, позволяющий разделить пользовательский интерфейс, бизнес-логику и уровень хранения данных. Такое разделение обеспечивает гибкость и упрощает сопровождение: изменения в одном уровне не требуют переработки остальных. Особое внимание уделяется структуре данных, поскольку корректно спроектированная модель позволяет эффективно выполнять операции поиска, фильтрации, анализа и обеспечивает целостность информации.

Архитектура решения построена на принципе клиент–серверного взаимодействия. Серверная часть реализует обработку запросов, выполнение прикладной логики и управление данными, хранящимися в базе. Клиентская часть отвечает за визуализацию информации, навигацию и взаимодействие пользователя с функциональными модулями. Взаимодействие между клиентом и сервером осуществляется через REST-интерфейс, что обеспечивает стандартизированный обмен данными и упрощает интеграцию с внешними системами.

1.2.1 Проектирование базы данных.

Разработка базы данных включает не только формирование структуры таблиц, но и определение механизмов, обеспечивающих корректность и устойчивость работы системы в условиях реальной эксплуатации. На этапе

					ПГУ 1.090501.02.001 ПЗ	Лист
						27
Изм.	Лист	№ докум.	Подпись	Дата		

проектирования была проведена детальная проработка логической и физической моделей данных, что позволило сформировать оптимальную архитектуру хранения информации, учитывающую особенности предметной области и требования к производительности.

Одним из ключевых аспектов проектирования стало определение набора ограничений целостности. Для всех сущностей были заданы первичные ключи, обеспечивающие уникальность записей, а также внешние ключи, формирующие связи между таблицами. Это позволяет гарантировать согласованность данных и предотвращает появление несвязанных или противоречивых записей. Дополнительно были реализованы ограничения на уровне полей — например, обязательность заполнения критически важных атрибутов, допустимые диапазоны значений и форматы данных.

Особое внимание уделено вопросам транзакционной обработки. Поскольку система выполняет множество параллельных операций — от регистрации посещаемости до обновления расписания — важно обеспечить корректность выполнения запросов при одновременном доступе к одним и тем же данным. Использование транзакций и уровней изоляции PostgreSQL позволяет избежать конфликтов данных и частично выполненных операций. Это особенно важно для модулей, связанных с изменением критически важных данных, таких как дисциплинарные записи или результаты посещаемости.

Дополнительно реализованы механизмы аудита, фиксирующие изменения в ключевых таблицах. Журналирование операций позволяет отслеживать историю действий пользователей, анализировать причины ошибок и обеспечивать прозрачность работы системы. Это особенно важно в образовательных учреждениях, где требуется сохранять историю изменений данных о посещаемости, дисциплине и учебных процессах.

Примеры структур таблиц:

Таблица пользователей системы

```
users (  
    id SERIAL PRIMARY KEY,
```

```

full_name VARCHAR(150) NOT NULL,
email VARCHAR(120) UNIQUE NOT NULL,
password_hash VARCHAR(255) NOT NULL,
role VARCHAR(20) NOT NULL,
created_at TIMESTAMP DEFAULT NOW()
);

```

Таблица групп и их участников;

```

groups (
  id SERIAL PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  commander_id INTEGER REFERENCES users(id)
);

```

```

group_members (
  id SERIAL PRIMARY KEY,
  group_id INTEGER REFERENCES groups(id) ON DELETE CASCADE,
  cadet_id INTEGER REFERENCES users(id) ON DELETE CASCADE
);

```

Таблица посещаемости;

```

attendance (
  id SERIAL PRIMARY KEY,
  schedule_item_id INTEGER REFERENCES schedule_items(id),
  cadet_id INTEGER REFERENCES users(id),
  present BOOLEAN NOT NULL,
  created_at TIMESTAMP DEFAULT NOW()
);

```

Примеры SQL-операций, выполняемых системой:

Получение списка занятий группы на сегодня:

```

SELECT s.id, s.start_time, c.title AS course_title
FROM schedule_items s
JOIN courses c ON c.id = s.course_id
WHERE s.group_id = :group_id
AND DATE(s.start_time) = CURRENT_DATE
ORDER BY s.start_time;

```

Получение дисциплинарных записей по группе:

```
SELECT d.id, d.violation_type, u.full_name AS cadet_name
FROM discipline d
JOIN users u ON u.id = d.cadet_id
WHERE d.group_id = :group_id
ORDER BY d.created_at DESC
LIMIT 10;
```

Пример логики работы серверной части при обращении к базе:

- клиент отправляет запрос: GET /api/attendance?group_id=3;
- FastAPI принимает запрос и вызывает обработчик;
- обработчик обращается к сервисному слою;
- сервис формирует SQL-запрос через SQLAlchemy;
- PostgreSQL выполняет запрос и возвращает данные;
- сервер сериализует результат в JSON и отправляет клиенту.

Проектирование структуры базы данных позволяет формализовать предметную область и определить, какие сущности и процессы должны быть отражены в информационной системе военно-учебного центра. Модель данных должна обеспечивать корректный учёт личного состава, расписания, посещаемости, успеваемости, дисциплинарных сведений и служебной документации, что требует чётко определённых связей между таблицами и строгой логики хранения информации. ER-диаграмма описывает основные объекты системы, их атрибуты и взаимосвязи, обеспечивая целостность данных, исключая дублирование и создавая основу для реализации серверной части, API и клиентского интерфейса. Такая модель позволяет систематизировать данные, поддерживать регламентированные процессы ВУЦ и гарантировать корректную работу всех функциональных модулей приложения.

Представленная на рисунке 5 ER-диаграмма служит основой для последующей реализации серверной части, разработки классов доменной модели и построения API, обеспечивая логическую целостность и согласованность всех компонентов приложения.

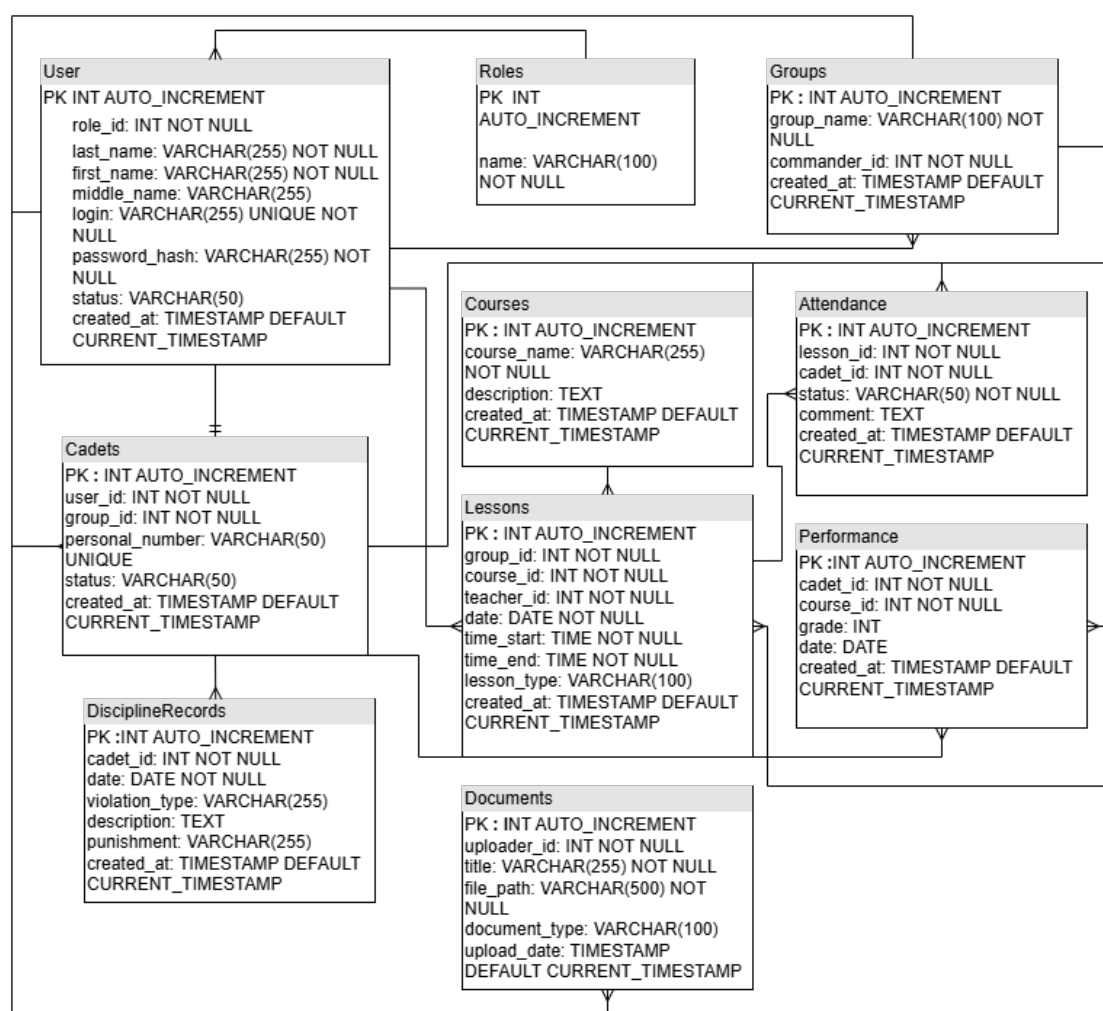


Рисунок 5 – ER-диаграмма БД

1.2.2 Разработка серверной части.

Разработка серверной части информационной системы направлена на обеспечение корректной обработки запросов, выполнение бизнес-логики и безопасное взаимодействие с базой данных. Серверный модуль является центральным компонентом архитектуры, поскольку именно он отвечает за управление данными, реализацию правил предметной области и предоставление клиентскому приложению стандартизированного интерфейса для обмена

информацией. При проектировании серверной части учитывались требования к надёжности, масштабируемости, безопасности и удобству дальнейшего сопровождения.

Основой серверной части выбран фреймворк FastAPI, который сочетает высокую производительность, строгую типизацию и удобство разработки. Использование асинхронной модели обработки запросов позволяет эффективно обслуживать большое количество одновременных подключений, что особенно важно для систем, работающих в условиях постоянного обновления данных — например, при фиксации посещаемости или работе с расписанием. FastAPI обеспечивает автоматическую генерацию документации API, что упрощает тестирование и интеграцию клиентской части.

Структура серверного приложения построена по модульному принципу. Каждый функциональный блок — авторизация, управление пользователями, расписание, посещаемость, дисциплина, документы — выделен в отдельный модуль, содержащий маршруты, схемы данных и бизнес-логику. Такой подход позволяет изолировать изменения и упрощает расширение функциональности. Для описания входных и выходных данных используются Pydantic-модели, обеспечивающие строгую валидацию и предотвращающие передачу некорректных данных между компонентами системы.

Взаимодействие с базой данных реализовано с использованием ORM-библиотеки SQLAlchemy, которая обеспечивает удобный доступ к данным и позволяет абстрагироваться от низкоуровневых SQL-операций. ORM-модели отражают структуру таблиц и связи между ними, что обеспечивает согласованность данных и упрощает выполнение сложных запросов. Для повышения производительности используются ленивые выборки, индексация и оптимизация запросов, что особенно важно при работе с большими объёмами данных, такими как журналы посещаемости или дисциплинарные записи.

Пример инициализации FastAPI-приложения:

```
from fastapi import FastAPI
from app.api.routes import auth, users, schedule, attendance,
discipline
from app.db.session import engine
from app.models import Base

app = FastAPI(title="Military Training Center IS")

# Создание таблиц при первом запуске
Base.metadata.create_all(bind=engine)

# Подключение маршрутов
app.include_router(auth.router, prefix="/api/auth")
app.include_router(users.router, prefix="/api/users")
app.include_router(schedule.router, prefix="/api/schedule")
app.include_router(attendance.router, prefix="/api/attendance")
app.include_router(discipline.router, prefix="/api/discipline")
```

Этот код формирует структуру серверного приложения, регистрирует маршруты и автоматически создаёт таблицы в базе данных при первом запуске.

ORM-модели описывают структуру таблиц и связи между ними. Пример модели занятия:

```
from sqlalchemy import Column, Integer, Date, Time, ForeignKey
from sqlalchemy.orm import relationship
from app.db.session import Base

class Lesson(Base):
    __tablename__ = "lessons"

    id = Column(Integer, primary_key=True)
    group_id = Column(Integer, ForeignKey("groups.id"))
    course_id = Column(Integer, ForeignKey("courses.id"))
    teacher_id = Column(Integer, ForeignKey("users.id"))
    date = Column(Date, nullable=False)
    time_start = Column(Time, nullable=False)
    time_end = Column(Time, nullable=False)

    group = relationship("Group")
    course = relationship("Course")
    teacher = relationship("User")
```

`__tablename__` — имя таблицы в базе данных.

`Column()` — определяет тип данных и ограничения.

`ForeignKey()` — задаёт связь с другой таблицей, обеспечивая целостность данных.

`relationship()` — позволяет получать связанные объекты (например, преподавателя занятия) без ручного SQL-запроса.

Сервисные классы инкапсулируют операции, которые выполняются над данными. Это позволяет отделить логику от API-маршрутов.

Пример сервисного слоя:

```
class AttendanceService:
    def mark(self, db, lesson_id: int, cadet_id: int, present:
bool):

        record = db.query(Attendance).filter_by(
            lesson_id=lesson_id,
            cadet_id=cadet_id
        ).first()
        # Если запись уже существует — обновляем статус
        if record:
            record.present = present
        else:
            # Если записи нет — создаём новую
            record = Attendance(
                lesson_id=lesson_id,
                cadet_id=cadet_id,
                present=present
            )
            db.add(record)
        db.commit()
        db.refresh(record)
        return record
```

Сервис проверяет, существует ли запись посещаемости для конкретного занятия и курсанта. Если запись есть — обновляется только поле present, что предотвращает дублирование. Если записи нет — создаётся новая строка в таблице.

db.commit() — сохраняет изменения в базе данных.

db.refresh() — обновляет объект, чтобы вернуть клиенту актуальные данные.

API-маршруты принимают запросы от клиента, вызывают сервисы и возвращают результат.

Пример API маршрута:

```
@router.post("/mark")

def mark_attendance(data: AttendanceDTO, db: Session =
Depends(get_db)):
    service = AttendanceService()
    return service.mark(
        db=db,
        lesson_id=data.lesson_id,
        cadet_id=data.cadet_id,
        present=data.present
    )
```

`@router.post()` — определяет HTTP-метод и путь.

`AttendanceDTO` — схема валидации входных данных.

`Depends(get_db)` — автоматически создаёт и закрывает соединение с базой.

Валидация данных:

```
class AttendanceDTO(BaseModel):
    lesson_id: int
    cadet_id: int
    present: bool
```

Pydantic автоматически проверяет типы данных. Если клиент отправит некорректные данные — сервер вернёт ошибку 422. Это защищает систему от некорректных запросов и повышает надёжность.

1.2.3 Разработка клиентской части.

Разработка клиентской части информационной системы направлена на создание удобного, интуитивно понятного и функционального интерфейса, обеспечивающего взаимодействие пользователя с серверной частью и базой данных. Клиентский модуль является ключевым элементом пользовательского опыта, поскольку именно он определяет, насколько эффективно и быстро пользователь сможет выполнять свои задачи. В условиях военно-учебного

центра, где требуется оперативная обработка данных, строгая структурированность информации и поддержка различных ролей пользователей, особое внимание уделяется эргономике интерфейса, стабильности работы и адаптивности.

Архитектурные принципы клиентской части. Клиентская часть реализована с использованием библиотеки React, что позволяет применять компонентный подход к построению интерфейса. Каждый функциональный элемент — таблица расписания, форма отметки посещаемости, панель командира, список документов — оформлен в виде независимого компонента, который может быть переиспользован и модифицирован без влияния на остальные части приложения. Такой подход обеспечивает гибкость, упрощает сопровождение и ускоряет разработку новых модулей.

Для управления состоянием используется контекстная модель React, что позволяет централизованно хранить данные о текущем пользователе, его роли, токене авторизации и параметрах интерфейса. Это обеспечивает согласованность данных между страницами и предотвращает необходимость многократного запроса одних и тех же сведений. Взаимодействие с сервером осуществляется через REST-API, а обмен данными выполняется в формате JSON, что обеспечивает лёгкость интеграции и высокую скорость обработки запросов.

На рисунке 20 представлена диаграмма классов для клиентской части приложения.

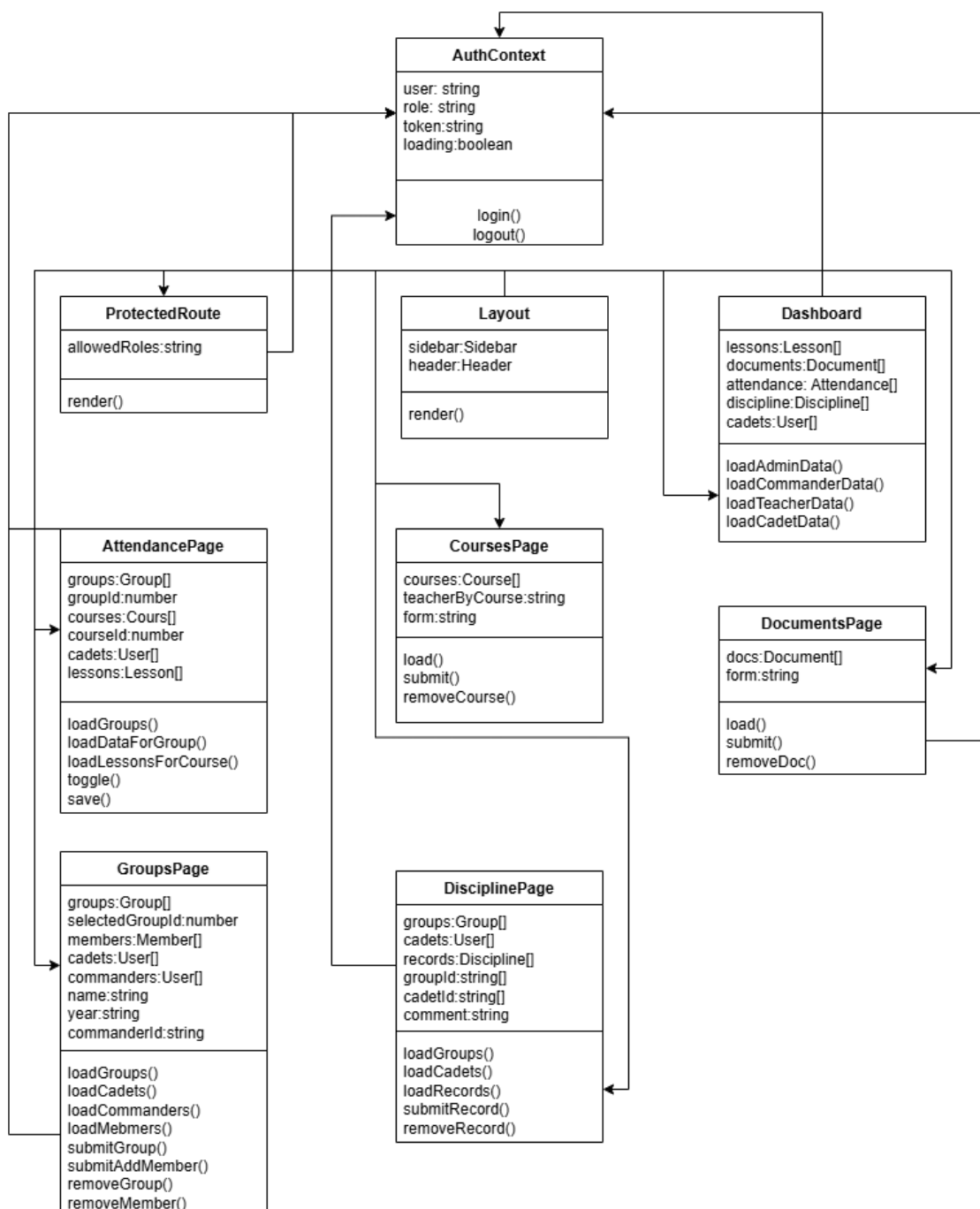


Рисунок 20 – UML-диаграмма классов

Ниже приведён пример компонента, который загружает и отображает расписание занятий для выбранной группы.

```
import { useEffect, useState } from "react";
export default function Schedule({ groupId }) {
  const [lessons, setLessons] = useState([]);
  useEffect(() => {
    fetch(`/api/schedule/${groupId}`)
      .then(res => res.json())
      .then(data => setLessons(data));
  }, [groupId]);
  return (
    <div>
      <h2>Расписание занятий</h2>
      {lessons.map(lesson => (
        <div key={lesson.id} className="lesson-card">
          <p><b>Дата:</b> {lesson.date}</p>
          <p><b>Дисциплина:</b> {lesson.course_name}</p>
          <p><b>Преподаватель:</b> {lesson.teacher_name}</p>
        </div>
      ))}
    </div>
  );
}
```

В этом компоненте используется локальное состояние `lessons`, которое изначально пустое и заполняется после загрузки данных с сервера. Хук `useEffect` выполняет запрос при каждом изменении идентификатора группы, что позволяет автоматически обновлять расписание при переключении между учебными подразделениями. После получения ответа данные преобразуются в формат JSON и сохраняются в состоянии, что вызывает автоматическую перерисовку компонента. Каждое занятие отображается в виде отдельной карточки, где выводятся дата, дисциплина и преподаватель. Такой подход обеспечивает реактивность интерфейса и избавляет пользователя от необходимости вручную обновлять страницу.

Для фиксации посещаемости используется компонент, который отображает строку таблицы с курсантом и чекбоксом, позволяющим отметить присутствие:

```
function AttendanceRow({ cadet, lessonId }) {  
  const [present, setPresent] = useState(cadet.present);  
  const updateAttendance = async (value) => {  
    setPresent(value);  
  
    await fetch("/api/attendance/mark", {  
      method: "POST",  
      headers: { "Content-Type": "application/json" },  
      body: JSON.stringify({  
        lesson_id: lessonId,  
        cadet_id: cadet.id,  
        present: value  
      })  
    });  
  };  
  
  return (  
    <tr>  
      <td>{cadet.last_name} {cadet.first_name}</td>  
      <td>  
        <input  
          type="checkbox"  
          checked={present}  
          onChange={ (e) => updateAttendance(e.target.checked) }  
        />  
      </td>  
    </tr>  
  );  
}
```

Здесь локальное состояние `present` хранит текущую отметку о присутствии. При изменении чекбокса состояние обновляется сразу, что обеспечивает мгновенную визуальную реакцию интерфейса. Параллельно отправляется запрос на сервер, который сохраняет изменения в базе данных. Такой механизм

позволяет преподавателю быстро отмечать посещаемость, не ожидая ответа сервера, а система синхронизирует данные в фоне. Это особенно важно в условиях ВУЦ, где посещаемость фиксируется ежедневно и должна передаваться в учебный отдел своевременно.

Пример компонента с отображением имени текущего пользователя:

```
import { useContext } from "react";
import { AuthContext } from "../context/AuthContext";
export default function UserBadge() {
  const { user } = useContext(AuthContext);
  return (
    <div className="user-badge">
      <span>{user ? `${user.last_name} ${user.first_name}` :
"Гость"}</span>
    </div>
  );
}
```

Этот компонент получает данные авторизованного пользователя из глобального контекста и отображает его фамилию и имя. Если пользователь не авторизован, выводится слово «Гость». Компонент не содержит собственной логики загрузки данных — он просто использует уже существующее состояние, что делает его лёгким, быстрым и удобным для размещения в шапке интерфейса или панели навигации. Такой элемент помогает пользователю ориентироваться в системе и подтверждает, что авторизация выполнена корректно.

Взаимодействие с сервером и обработка данных. Для выполнения запросов к серверу используется библиотека Axios, обеспечивающая удобный механизм отправки HTTP-запросов и обработки ответов. Каждый модуль клиентской части имеет собственный API-слой, содержащий функции для получения, создания, обновления и удаления данных. Такой подход позволяет изолировать сетевую логику от визуальной части и упрощает тестирование.

Особое внимание уделено обработке ошибок. В случае недоступности сервера, истечения срока действия токена или отсутствия прав доступа

пользователю отображаются информативные сообщения. Это снижает вероятность некорректных действий и повышает устойчивость системы.

Ниже представлены примеры взаимодействия с сервером.

Загрузка данных и фильтрация на клиенте:

```
import { useEffect, useState } from "react";
export default function CoursesList() {
  const [courses, setCourses] = useState([]);
  useEffect(() => {
    fetch("/api/courses")
      .then(res => res.json())
      .then(data => {
        const filtered = data.filter(c => c.active === true);
        setCourses(filtered);
      });
  }, []);
  return (
    <div className="courses-list">
      {courses.map(c => (
        <div key={c.id}>{c.name}</div>
      ))}
    </div>
  );
}
```

Компонент загружает список дисциплин с сервера и выполняет простую обработку данных — фильтрацию по признаку активности. Это типичная задача клиентской части: сервер возвращает полный набор данных, а интерфейс оставляет только те элементы, которые нужны пользователю в текущем контексте. После обработки данные сохраняются в состоянии и отображаются в виде списка.

Загрузка дисциплин:

```
import { useEffect, useState } from "react";
export default function CoursesSelect({ onSelect }) {
  const [courses, setCourses] = useState([]);
```

```

useEffect(() => {
  fetch("/api/courses")
    .then(res => res.json())
    .then(data => setCourses(data));
}, []);
return (
  <select onChange={e => onSelect(e.target.value)}>
    <option value="">Выберите дисциплину</option>
    {courses.map(c => (
      <option key={c.id} value={c.id}>
        {c.name}
      </option>
    ))}
  </select>
);

```

Этот компонент выполняет запрос к серверу для получения списка дисциплин и отображает их в виде выпадающего списка. После загрузки данные сохраняются во внутреннем состоянии, что позволяет компоненту автоматически обновляться при изменении информации на сервере. Такой подход обеспечивает корректную интеграцию клиентской части с серверным API: данные запрашиваются, преобразуются в удобный формат и используются для формирования элементов интерфейса, обеспечивая пользователю быстрый доступ к актуальному перечню дисциплин.

1.2.4 Проектирование пользовательского интерфейса.

Проектирование пользовательского интерфейса является ключевым этапом разработки информационной системы, поскольку именно через интерфейс осуществляется взаимодействие пользователя с функциональными модулями приложения. От качества интерфейса зависит скорость выполнения операций, удобство работы, уровень ошибок при вводе данных и общее восприятие системы. В условиях военно-учебного центра, где пользователи обладают различным уровнем подготовки и выполняют широкий спектр задач, интерфейс должен быть интуитивно понятным, структурированным и адаптированным под особенности каждого типа пользователя.

Основной принцип проектирования интерфейса — разделение функциональности по ролям. В системе предусмотрены три категории пользователей: курсанты, преподаватели и командиры подразделений. Каждая категория имеет собственные задачи, поэтому интерфейс формируется с учётом специфики их деятельности. Курсанту требуется быстрый доступ к расписанию, учебным материалам и результатам успеваемости. Преподавателю необходимы инструменты для ведения посещаемости, фиксации результатов занятий и работы с учебной документацией. Командир подразделения использует систему для контроля дисциплины, управления личным составом и формирования отчётности. Такое ролевое разделение позволяет исключить перегруженность интерфейса и повысить удобство работы.

При проектировании интерфейса использовались принципы модульности и единообразия. Все страницы системы имеют общую структуру: навигационное меню, основную рабочую область и вспомогательные элементы управления. Это обеспечивает предсказуемость поведения интерфейса и снижает время адаптации пользователя. Навигационное меню формируется динамически в зависимости от роли пользователя, что позволяет скрывать недоступные функции и предотвращать ошибки, связанные с попыткой доступа к запрещённым разделам.

В процессе проектирования интерфейса учитывались требования к безопасности. Визуально скрытые элементы не только недоступны для взаимодействия, но и полностью исключены из DOM-структуры для предотвращения попыток несанкционированного доступа через инструменты разработчика. Все формы ввода данных снабжены валидацией, предотвращающей отправку некорректной информации.

Внешний вид «главной панели» для курсанта представлен на рисунке 6.

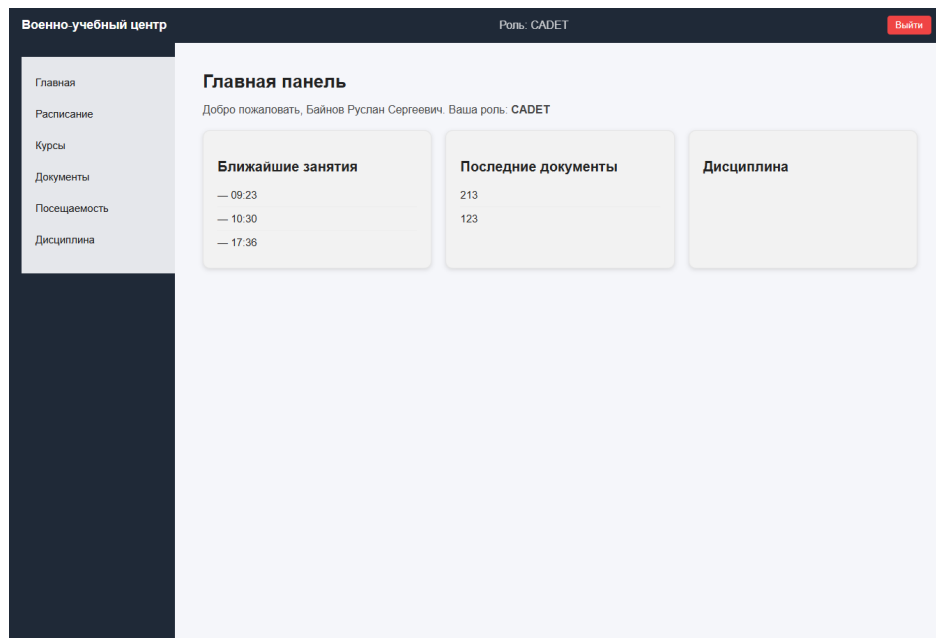


Рисунок 6 – Внешний вид «Главная»

На вкладке «Расписание» курсант может ознакомиться со своим расписанием.

Внешний вид представлен на рисунке 7.

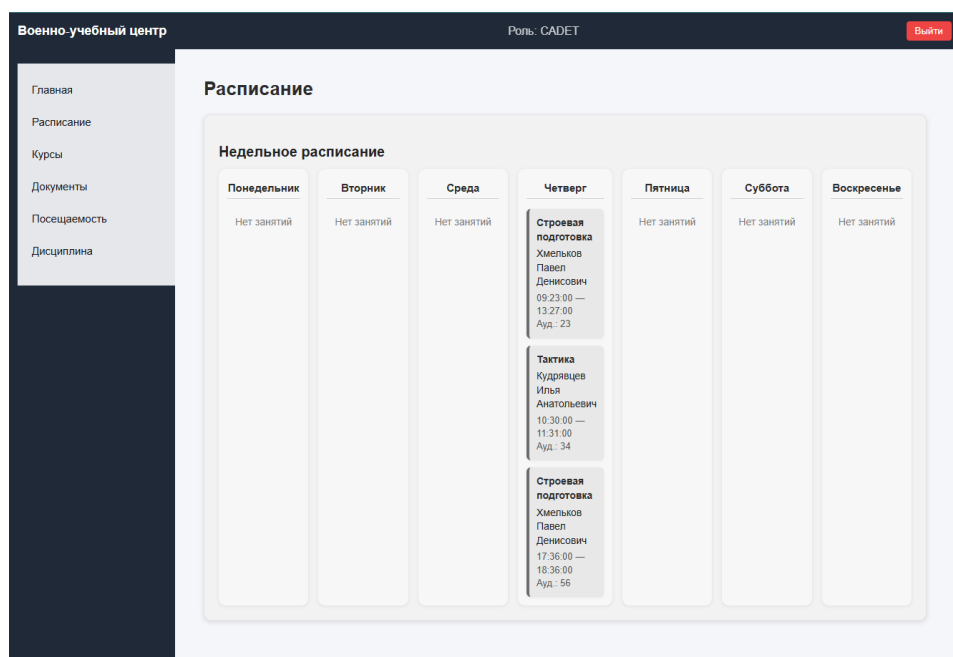


Рисунок 7 – Внешний вид «Расписание»

Перейдя на вкладку «Курсы» курсант может увидеть полный список учебных дисциплин, ознакомиться с их описанием и посмотреть преподавателя.

Внешний вид вкладки представлен на рисунке 8.

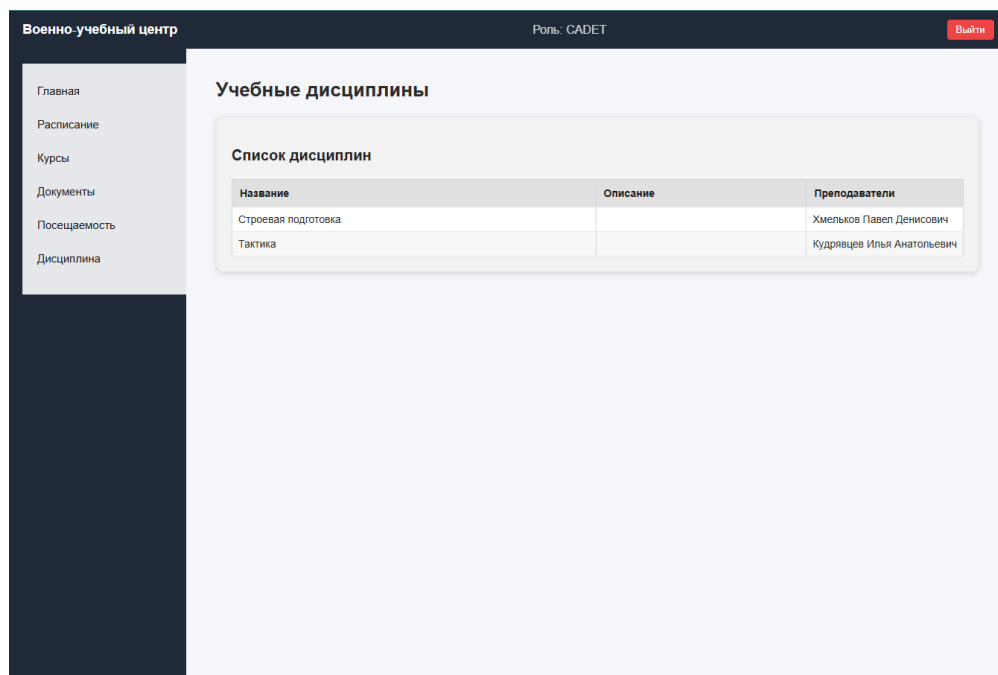


Рисунок 8 – Внешний вид «Курсы»

Вкладка «Документы» предназначена для загрузки документов в систему, например курсовых работ курсанта.

Ознакомиться с представлением вкладки «Документы» можно на рисунке 9.

Документы

Добавить документ

Название
Введите название документа

Категория
Например: отчёт, приказ...

Описание
Краткое описание документа

ID загружившего
Введите ID пользователя

Файл
Выберите файл | файл не выбран

Добавить

Рисунок 9 – Внешний вид «Документы»

На странице «Посещаемость» курсант может посмотреть все свои отметки посещения занятия, по учебным дисциплинам.

Внешний вид страницы представлен на рисунке 10.

Курсант	26.02.2026	26.02.2026
Байнов Руслан Сергеевич	☐	☐

Рисунок 10 – Внешний вид «Посещаемость»

В «Дисциплине» курсант может посмотреть журнал дисциплины и ознакомиться со своими нарушениями.

Представление вкладки «Дисциплина» на рисунке 11.

ID	Группа	Кадет	Командир	Тип	Комментарий	Дата
8	21voe1	Байнов Руслан Сергеевич	Сорокин Дмитрий Евгеньевич	Опоздание	Опоздал на занятие	27.02.2026, 09:43:04

Рисунок 11 – Внешний вид «Дисциплина»

На главной странице пользователя «админ» выведена основная статистическая информация по платформе. Для админа главная страница изображена на рисунке 12.

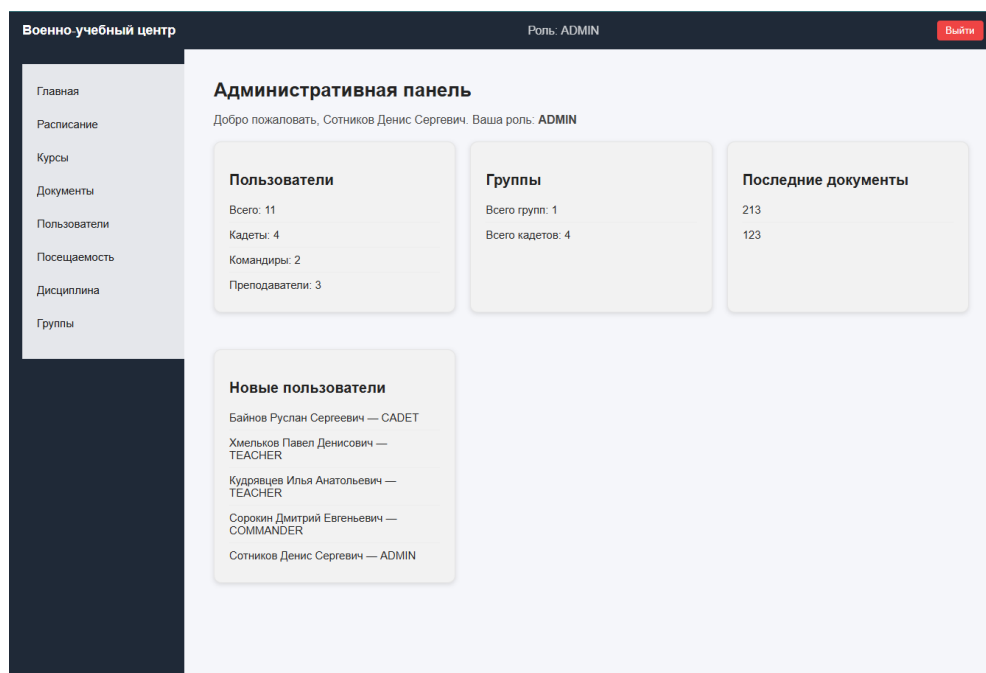


Рисунок 12 – Внешний вид «Главная страница для администратора»

Вкладка «Расписание» для администратора отличается тем, что он может создавать расписание и редактировать его. Вкладка «Расписание для администратора» представлена на рисунке 13.

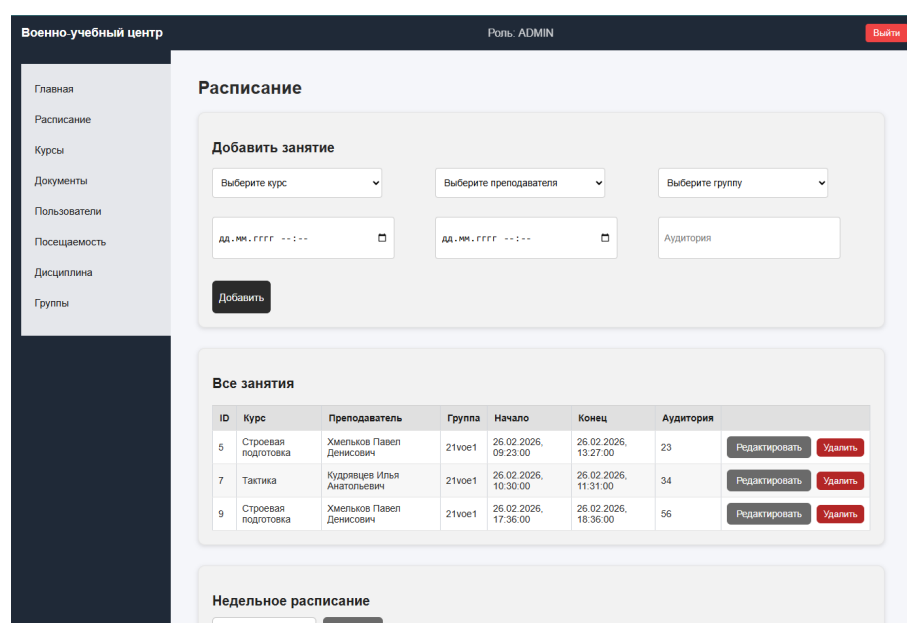


Рисунок 13 – Внешний вид «Расписание для администратора»

После перехода на рабочий экран пользователь открывает модуль расписания, инициируя запрос к серверу на получение актуальных данных. Серверная часть обрабатывает запрос, формирует ответ в формате JSON и передаёт клиенту сведения о занятиях, аудиториях, преподавателях и временных интервалах. Полученные данные отображаются в интерфейсе в виде таблицы или календарной структуры, после чего дальнейшее поведение зависит от роли пользователя.

Курсант имеет доступ только к просмотру расписания и использует модуль для получения информации о предстоящих занятиях. Преподаватель, помимо просмотра, может отмечать посещаемость, фиксируя присутствие или отсутствие курсантов на занятиях. Командир подразделения обладает расширенными правами и может не только просматривать расписание, но и вносить изменения: корректировать время занятий, заменять преподавателей, изменять аудитории или добавлять дополнительные мероприятия. Все внесённые командиром изменения отправляются на сервер, где проходят проверку и сохраняются в базе данных. После успешного обновления данных сервер возвращает подтверждение, и интерфейс автоматически обновляется, отображая актуальное состояние расписания.

Данный сценарий демонстрирует, как один и тот же модуль системы адаптируется под разные роли пользователей, обеспечивая разграничение прав доступа и корректную обработку действий в зависимости от уровня полномочий.

На рисунке 14 изображена диаграмма последовательности для данного сценария.

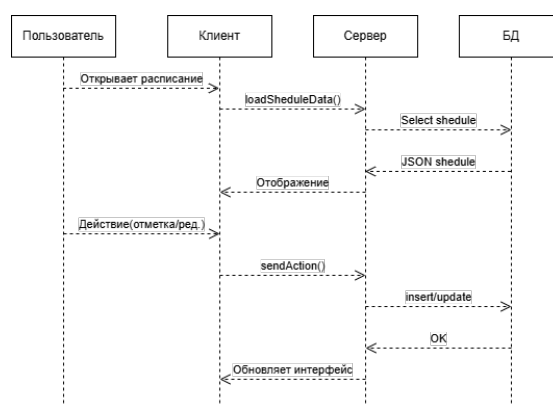


Рисунок 14 – UML-диаграмма последовательности

На странице «Курсы» администратор может редактировать список дисциплин

Внешний вид на рисунке 15.

ID	Название	Описание	Преподаватели	Удалить
4	Стрелковая подготовка		Хмельков Павел Денисович	Удалить
5	Тактика		Кудрявцев Илья Анатольевич	Удалить

Рисунок 15 – Внешний вид «Курсы для администратора»

На рисунке 16 представлена страница «Пользователи», на которой администратор может создавать новых пользователей и назначать им роль.

ID	Логин	ФИО	Роль	Активен
1	admin	Тя-Еёова вВа	ADMIN	Да
2	123	123	CADET	Да
3	1234	аваа	CADET	Да
4	12345	12345	COMMANDER	Да
5	1313	1313	TEACHER	Да
6	222	222	CADET	Да
7	333	Байнов Руслан Сергеевич	CADET	Да
8	444	Хмельков Павел Денисович	TEACHER	Да
9	555	Кудрявцев Илья Анатольевич	TEACHER	Да

Рисунок 16 – Внешний вид «Пользователи»

На странице «Группы» администратор может создавать учебные группы и изменять их состав. Страница изображена на рисунке 17.

Военно-учебный центр Роль: ADMIN Выйти

Учебные группы

Создать группу

Название

Год набора

Командир (опционально)

Создать

Состав группы

Добавить кадета
 Выберите кадета

Добавить

ID	Кадет	
7	Байнов Руслан Сергеевич	Удалить

Список групп

21voe1 (2026)	X
---------------	----------

Рисунок 17 – Внешний вид «Группы»

На главной странице для преподавателя находится информация о ближайших занятиях и посещаемости. Главная страница представлена на рисунке 18. Остальные вкладки имеют такой же функционал, как у администратора.

Военно-учебный центр Роль: TEACHER Выйти

Панель преподавателя

Добро пожаловать, Хмельков Павел Денисович. Ваши занятия на сегодня.

Ближайшие занятия

- 21voe1 — 09:23
- 21voe1 — 10:30
- 21voe1 — 17:36

Группы сегодня

Последняя посещаемость

Рисунок 18 – Внешний вид «Главная для преподавателя»

Сценарий работы с личным составом начинается с того, что командир подразделения переходит в соответствующий модуль из основного рабочего экрана. После открытия модуля система отправляет запрос к серверу для получения актуального списка личных дел, включающего сведения о каждом курсанте. Полученные данные отображаются в интерфейсе в виде структурированного списка, что позволяет командиру быстро ориентироваться в составе подразделения.

Далее пользователь выбирает конкретного курсанта, после чего система загружает и отображает его личное дело. В личном деле содержатся основные сведения о курсанте, данные о прохождении подготовки, дисциплинарные записи, результаты контрольных мероприятий и прикреплённые документы. На этом этапе командиру доступны несколько вариантов действий, каждое из которых инициирует собственный поток обработки данных.

Если требуется изменить персональные данные курсанта или обновить сведения о его подготовке, командир вносит корректировки, которые затем отправляются на сервер. В случае необходимости загрузки нового документа — например, служебной записки, приказа или медицинской справки — файл передаётся на сервер и сохраняется в соответствующем разделе личного дела. При добавлении дисциплинарной записи система формирует запрос, содержащий сведения о нарушении, дате, основании и принятых мерах, после чего отправляет его на сервер для сохранения.

Все действия, связанные с редактированием данных, загрузкой документов или добавлением дисциплинарных сведений, проходят обработку на серверной стороне. После успешного обновления базы данных сервер возвращает подтверждение, и интерфейс автоматически обновляется, отображая актуальное состояние личного дела. Такой подход обеспечивает целостность данных, прозрачность действий пользователя и удобство работы с большим объёмом информации.

На рисунке 19 изображена диаграмма последовательности для данного сценария.

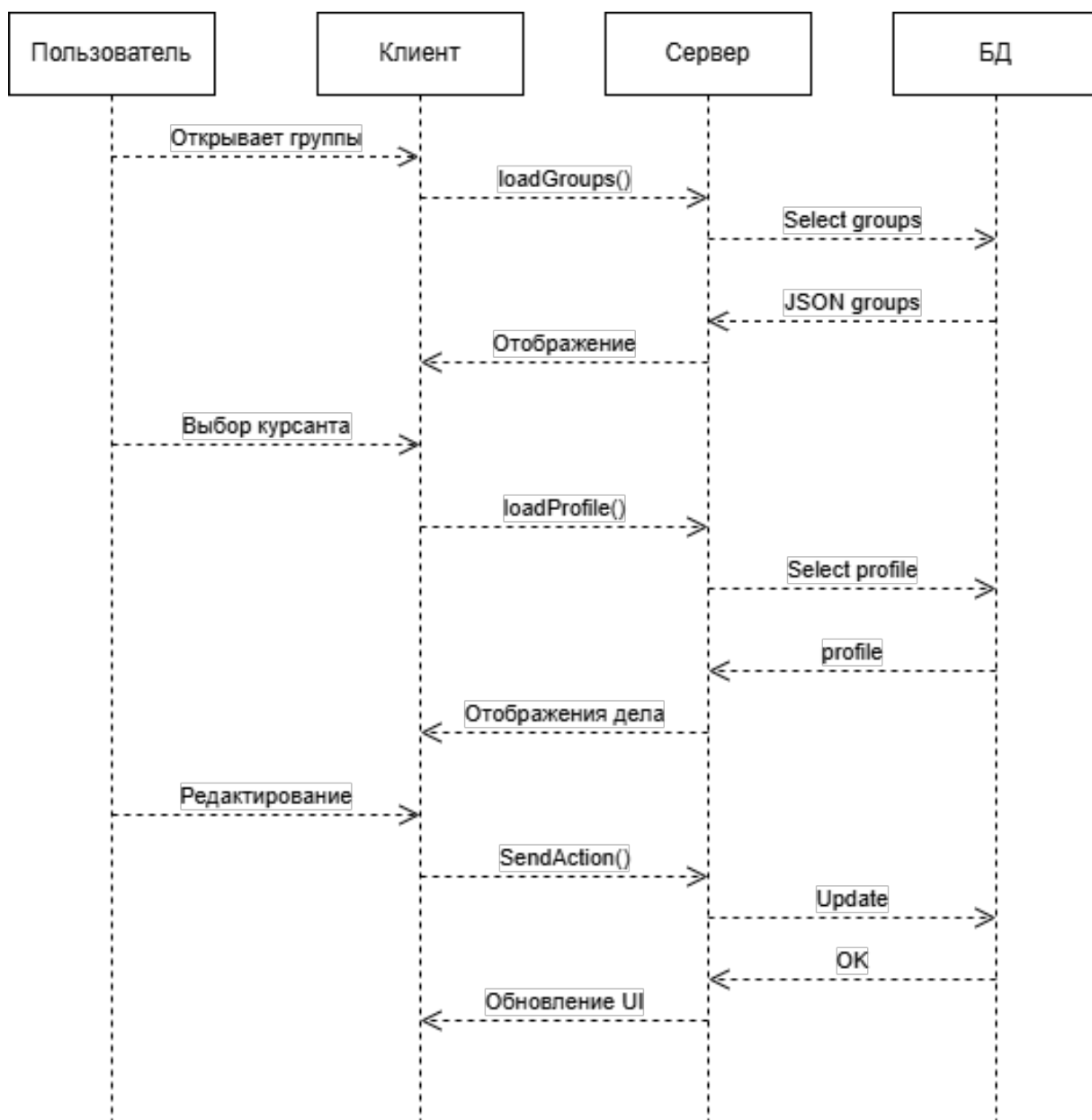


Рисунок 19 – UML-диаграмма последовательности

Для командира группы на главном экране выведена нужная ему статистка. На рисунке 20 можно ознакомиться с внешним видом главного экрана для командира.

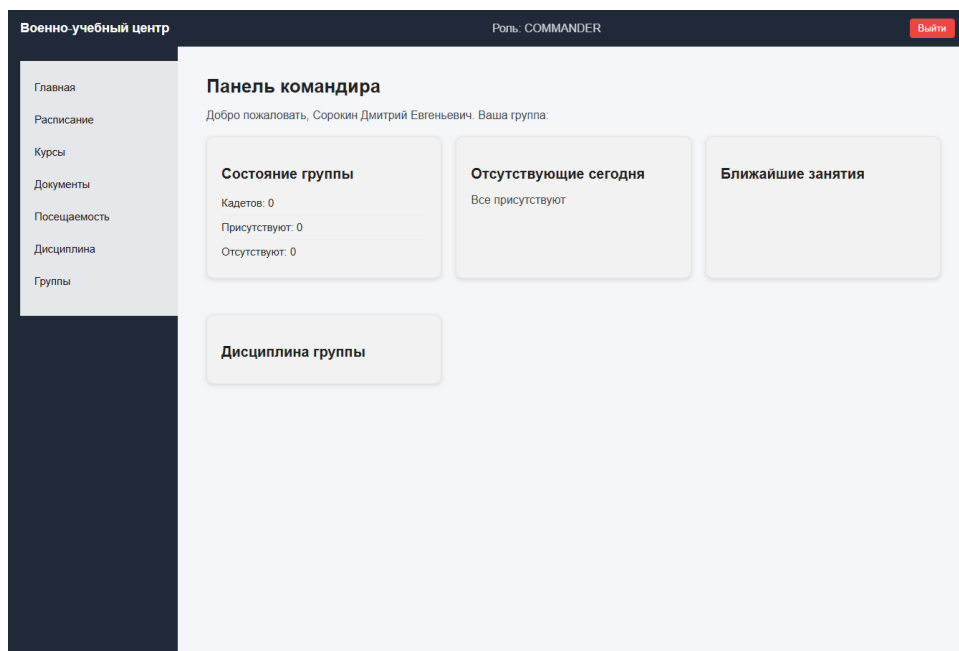


Рисунок 20 – Внешний вид «Отчеты»

На странице дисциплина командир может написать замечание в журнал дисциплины. Рисунок 21 представляет собой страницу «Дисциплина»

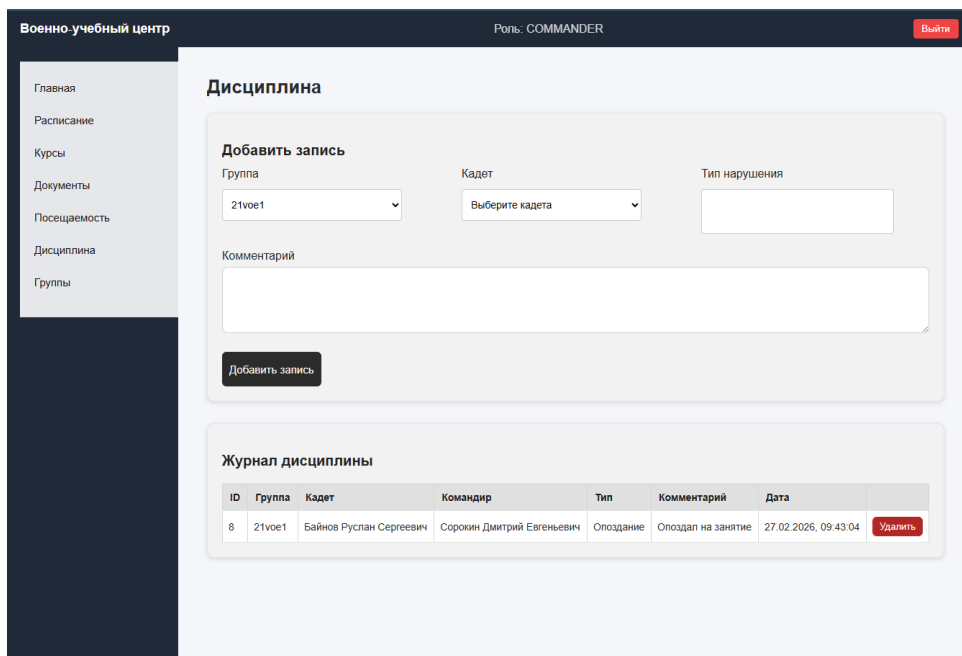


Рисунок 21 – Внешний вид «Дисциплина для на командира»

1.2.5 Интеграция и взаимодействие компонентов

Интеграция компонентов клиентской и серверной частей системы обеспечивает согласованную работу всех модулей, корректный обмен данными и единое поведение приложения для пользователей разных ролей. Взаимодействие реализовано на основе REST-архитектуры, что позволяет разделить ответственность между слоями, упростить масштабирование и обеспечить независимое развитие клиентской и серверной логики.

Архитектурная модель взаимодействия.

Система построена по принципу «клиент–сервер», где:

- клиентская часть (React) отвечает за отображение интерфейса, управление состоянием и обработку пользовательских действий;
- серверная часть (FastAPI) реализует валидацию данных, авторизацию, работу с базой данных и формирование API-ответов;
- база данных (PostgreSQL) хранит структурированные данные о пользователях, группах, дисциплинах, посещаемости, документах и расписании.

Коммуникация между компонентами осуществляется через HTTP-запросы в формате JSON, что обеспечивает простоту интеграции и независимость технологий.

Потоки данных между компонентами формируют единый контур взаимодействия клиентской части, серверной логики и базы данных, обеспечивая согласованность работы всех модулей системы. Авторизация и управление доступом основаны на передаче JWT-токена: после успешного входа клиент получает токен и использует его в каждом запросе к серверу, а сервер проверяет его валидность и определяет роль пользователя, что влияет на доступность маршрутов и объём возвращаемых данных. Работа с учебными группами включает обмен информацией между страницами GroupsPage, AttendancePage и DisciplinePage и сервером, который формирует агрегированные ответы, объединяя данные о группах, составе подразделений и назначенных командирах.

Ниже приведены примеры, демонстрирующие, как клиентская часть взаимодействует с серверной, как данные проходят через API, и как компоненты

системы объединяются в единый рабочий процесс.

Клиент вызывает серверный эндпоинт для получения списка курсантов группы:

```
import { useEffect, useState } from "react";
export default function CadetsList({ groupId }) {
  const [cadets, setCadets] = useState([]);
  useEffect(() => {
    fetch(`/api/users/group/${groupId}`)
      .then(res => res.json())
      .then(data => setCadets(data));
  }, [groupId]);
  return (
    <div className="cadets-list">
      {cadets.map(c => (
        <div key={c.id} className="cadet-item">
          {c.last_name} {c.first_name}
        </div>
      ))}
    </div>
  );
}
```

Серверный маршрут:

```
@router.get("/group/{group_id}")
def get_cadets_by_group(group_id: int, db: Session =
Depends(get_db)):
    return db.query(User).filter(User.group_id ==
group_id).all()
```

Этот пример показывает прямое взаимодействие между компонентом интерфейса и серверным API. Клиент отправляет запрос на получение списка курсантов, сервер выполняет запрос к базе данных и возвращает результат в формате JSON. Компонент автоматически обновляет интерфейс, отображая актуальный состав группы. Такой механизм используется в личных кабинетах преподавателей и командиров, где требуется оперативный доступ к данным личного состава.

Клиент отправляет изменение статуса дисциплинарной записи:

```
import { useState } from "react";
export default function DisciplineStatusToggle({ recordId,
initial }) {
  const [closed, setClosed] = useState(initial);
  const update = async (value) => {
    setClosed(value);

    await fetch("/api/discipline/update-status", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        record_id: recordId,
        closed: value
      })
    });
  };
  return (
    <input
      type="checkbox"
      checked={closed}
      onChange={e => update(e.target.checked)}
    />
  );
}
```

Серверный маршрут:

```
@router.post("/update-status")
def update_discipline_status(data: DisciplineStatusDTO, db:
Session = Depends(get_db)):
  service = DisciplineService()
  return service.update_status(
    db=db,
    record_id=data.record_id,
    closed=data.closed
  )
```

Этот пример иллюстрирует ситуацию, когда клиентская часть не только получает данные, но и передаёт их на сервер. Компонент изменяет состояние элемента интерфейса и отправляет POST-запрос с обновлёнными данными, после чего сервер принимает информацию, вызывает соответствующий сервисный метод и обновляет запись в базе данных. Такой механизм обеспечивает согласованность между действиями пользователя в интерфейсе и фактическим состоянием данных на сервере, что особенно важно при работе с динамическими модулями системы.

Пример вёрстки карточка занятия:

```
export default function LessonCard({ lesson }) {  
  return (  
    <div className="lesson-card">  
      <div className="lesson-header">  
        <span className="lesson-date">{lesson.date}</span>  
        <span className="lesson-time">{lesson.time_start}-  
{lesson.time_end}</span>  
      </div>  
      <div className="lesson-body">  
        <p><b>Дисциплина:</b> {lesson.course_name}</p>  
        <p><b>Преподаватель:</b> {lesson.teacher_name}</p>  
      </div>  
      <div className="lesson-footer">  
        <button className="details-btn">Подробнее</button>  
      </div>  
    </div>  
  )  
}
```

Этот фрагмент демонстрирует пример вёрстки интерфейсного компонента, который отображает информацию о занятии. Разметка разделена на логические блоки: заголовок с датой и временем, основная часть с данными о дисциплине и преподавателе, а также нижняя панель с управляющей кнопкой. Такая структура обеспечивает удобочитаемость и позволяет легко адаптировать компонент под различные сценарии отображения в системе.

Управление учебным процессом реализуется через взаимодействие страниц CoursesPage, SchedulePage и AttendancePage с сервером для получения расписания, списка дисциплин и отметок посещаемости; клиент отображает данные в таблицах и отправляет изменения обратно на сервер, где они проходят валидацию и сохраняются в базе данных. Документооборот обеспечивается страницей DocumentsPage, которая отправляет файлы через FormData, а сервер принимает multipart-запросы, сохраняет файлы и регистрирует их метаданные в базе данных. Дисциплинарные записи обрабатываются через DisciplinePage: клиент получает список нарушений, фильтрует их по группе или пользователю и отправляет новые записи на сервер, где они проверяются и фиксируются в журнале дисциплины

На рисунке 22 представлена диаграмма последовательности.

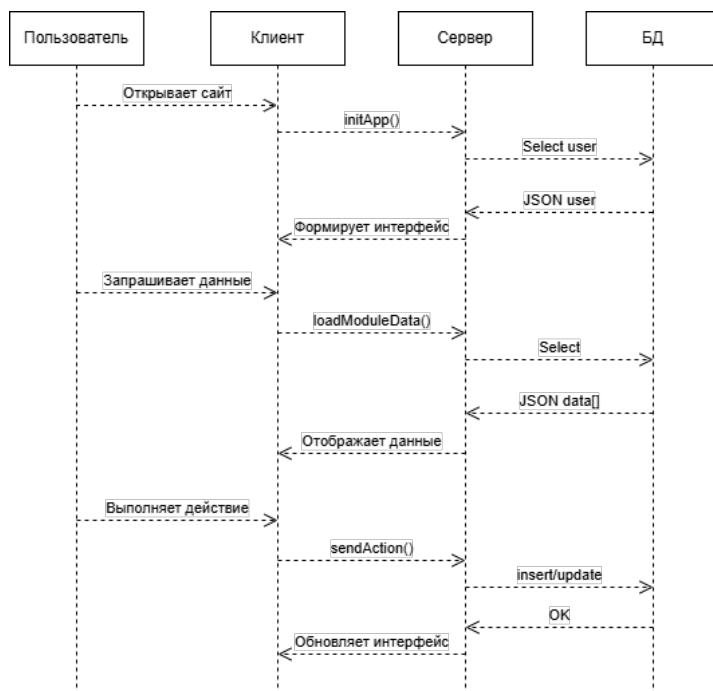


Рисунок 22 – UML-диаграмма последовательности

1.2.6 Динамика работы приложения.

Динамика работы приложения отражает последовательность действий, которые выполняются при взаимодействии пользователя с системой, а также обмен данными между клиентской частью, сервером и базой данных. Работа приложения строится вокруг ролевой модели, что определяет доступные функции и сценарии поведения интерфейса. Каждый пользователь — курсант, преподаватель, командир или администратор — получает доступ только к тем компонентам, которые соответствуют его полномочиям, а интерфейс динамически адаптируется под выбранную роль.

При запуске приложения клиентская часть инициирует загрузку данных о текущем пользователе через контекст авторизации. После успешной проверки токена сервер возвращает сведения о роли и идентификаторе пользователя, что позволяет сформировать индивидуальный набор доступных страниц и элементов интерфейса. Далее начинается загрузка данных, необходимых для работы конкретной роли: расписания, документов, дисциплинарных записей, состава групп или статистики посещаемости. Эти данные подгружаются асинхронно, что обеспечивает плавность работы интерфейса и отсутствие блокировок.

Взаимодействие пользователя с системой происходит через цепочку событий, инициируемых действиями на клиенте. Например, при выборе группы на странице посещаемости клиент отправляет запрос на сервер для получения состава подразделения, затем — запрос на получение списка дисциплин, а после выбора дисциплины — запрос на получение занятий и отметок посещаемости. Все данные отображаются в интерфейсе в виде таблиц, форм и карточек, а изменения фиксируются только после явного подтверждения пользователем. Такой подход обеспечивает предсказуемость поведения и предотвращает случайные изменения данных.

Серверная часть обрабатывает запросы последовательно: проверяет токен, валидирует параметры, выполняет бизнес-логику и обращается к базе данных. PostgreSQL обеспечивает хранение структурированных данных, а сервер формирует агрегированные ответы, объединяя информацию из нескольких таблиц. Это позволяет клиенту получать готовые для отображения данные без необходимости выполнять дополнительную обработку

Особое внимание уделено динамическому обновлению данных. После выполнения операций — добавления дисциплинарной записи, загрузки документа, изменения состава группы или сохранения посещаемости — клиент автоматически инициирует повторную загрузку соответствующих данных. Это гарантирует актуальность отображаемой информации и согласованность между компонентами. Такой механизм обеспечивает реактивность интерфейса и позволяет пользователю сразу видеть результат своих действий.

На рисунке 23 изображена диаграмма состояний.

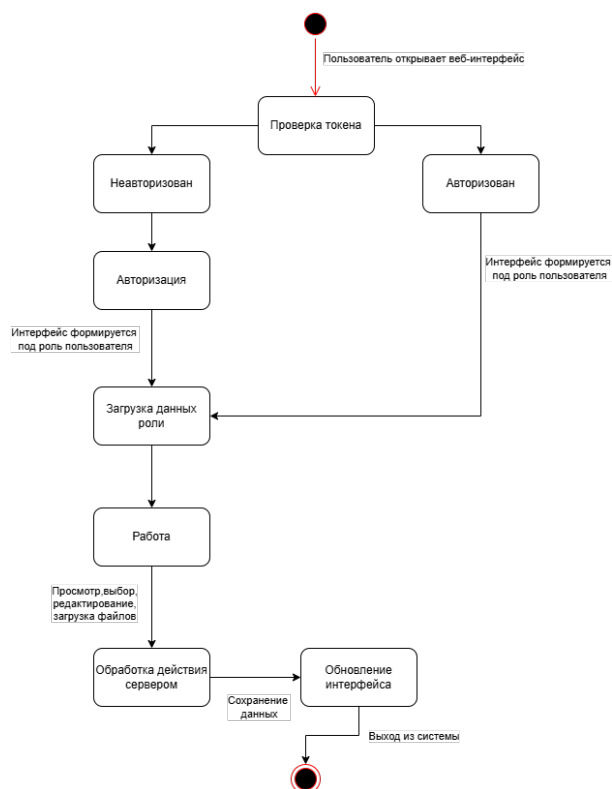


Рисунок 23 – UML-диаграмма состояний

1.2.7 Тестирование приложения.

Тестирование приложения направлено на проверку корректности работы всех модулей системы, выявление ошибок, оценку стабильности и соответствия функциональным требованиям. Поскольку система включает клиентскую часть на React, серверную часть на FastAPI и базу данных PostgreSQL, тестирование охватывает несколько уровней: интерфейс, API, интеграцию компонентов и работу с данными. Комплексный подход позволяет убедиться, что приложение функционирует предсказуемо, корректно обрабатывает пользовательские действия и устойчиво к ошибочным данным.

Функциональное тестирование. Функциональное тестирование проверяет соответствие реализованных функций требованиям, описанным в техническом задании. Для каждой роли пользователя — курсанта, преподавателя, командира и администратора — были проверены доступные сценарии работы: авторизация, просмотр расписания, управление посещаемостью, работа с документами, создание дисциплинарных записей, управление группами и пользователями. Тестирование проводилось вручную, с использованием тестовых данных, что позволило выявить и устранить ошибки в логике отображения, фильтрации данных и обработке форм.

Тестирование пользовательского интерфейса. Проверка интерфейса включала оценку корректности отображения элементов, адаптивности верстки и удобства взаимодействия. Были протестированы таблицы, формы, выпадающие списки, модальные окна и элементы навигации. Особое внимание уделялось ролевой модели: интерфейс должен скрывать недоступные функции и корректно отображать данные в зависимости от роли пользователя. Тестирование проводилось в различных браузерах (Chrome, Firefox, Edge), что позволило убедиться в кроссбраузерной совместимости.

Модульное тестирование. Для обеспечения стабильности и надёжности серверной части разработаны модульные тесты с использованием фреймворка pytest. Тесты покрывают критически важную бизнес-логику, гарантируя корректную работу отдельных компонентов системы в изоляции от остальных

модулей. Применение имитации объектов базы данных (mock-объектов) позволяет проверить логику без подключения к реальной СУБД. На рисунке 24 изображены результаты тестирования.

```
plugins: anyio-4.12.1, mock-3.15.1
collected 2 items

app/tests/test_attendance_service.py::TestAttendanceLogic::test_create_new_attendance_record PASSED [ 50%]
app/tests/test_attendance_service.py::TestAttendanceLogic::test_update_existing_attendance_record PASSED [100%]

===== 2 passed in 0.03s =====
```

Рисунок 24 – Результаты тестов

Тестовые сценарии. Для проверки соответствия реализованной функциональности требованиям технического задания выполнены сквозные тестовые сценарии. В отличие от модульных тестов, они проверяют полный путь пользователя — от действия в интерфейсе до сохранения данных в базе.

Тест-кейс 1: Отметка посещаемости преподавателем.

Цель: проверить, что преподаватель может отметить присутствие курсанта на занятии и сохранить результат в базе данных.

Начальные условия:

- пользователь авторизован в системе с ролью «Преподаватель» (TEACHER);
- в разделе «Посещаемость» выбрана учебная группа и дисциплина;
- на экране отображена таблица со списком курсантов и датами занятий.

Шаги выполнения:

- в открывшейся таблице найти строку, соответствующую курсанту.
- в столбце, соответствующем дате проведённого занятия, щёлкнуть по чекбоксу — должен установиться флажок.
- нажать кнопку «Сохранить посещаемость», расположенную под таблицей.

Ожидаемый результат:

Система выводит всплывающее уведомление с текстом «Посещаемость сохранена».

После обновления страницы (или повторного входа в раздел) ранее установленный флажок остаётся в активном состоянии.

Фактический результат: соответствует ожидаемому — отметка сохраняется, данные не теряются при перезагрузке.

На рисунке 25 изображен результат проверки.

Рисунок 25 – Тест-кейс 1

Тест-кейс 2: Создание нового пользователя администратором

Цель: проверить, что администратор может создать новую учётную запись курсанта, и она корректно отображается в системе.

Начальные условия:

- пользователь авторизован в системе с ролью «Администратор» (ADMIN).
- открыт раздел «Пользователи».

Шаги выполнения:

- нажать кнопку «Создать пользователя» (или «Добавить»).
- в открывшейся форме заполнить обязательные поля:
- ФИО: «Петров Пётр Петрович»;
- пароль: *****.

В выпадающем списке «Роль» выбрать значение CADET.

Ожидаемый результат:

Форма создания закрывается.

В общем списке пользователей появляется новая строка с данными: «Петров Пётр Петрович», роль — «CADET».

При переходе на главную панель администратора счётчик «Пользователи» увеличивается на единицу.

Фактический результат: соответствует ожидаемому — пользователь успешно создан, данные отображаются корректно. На рисунке 26 результат.

Военно-учебный центр

Роль: ADMIN

Выйти

Главная

Расписание

Курсы

Документы

Пользователи

Посещаемость

Дисциплина

Группы

Создать пользователя

petrov

Петров Пётр Петрович;

...

CADET

Создать

Все пользователи

ID	Логин	ФИО	Роль	Активен
1	admin	Тя-ЧЕЕбаа вБа	ADMIN	Да
2	123	123	CADET	Да
3	1234	аваа	CADET	Да
4	12345	12345	COMMANDER	Да
5	1313	1313	TEACHER	Да
6	222	222	CADET	Да
7	333	Байнов Руслан Сергеевич	CADET	Да
8	444	Хмельков Павел Денисович	TEACHER	Да
9	555	Кудрявцев Илья Анатольевич	TEACHER	Да
10	666	Сорокин Дмитрий Евгеньевич	COMMANDER	Да
11	777	Сотников Денис Сергеевич	ADMIN	Да
12	petrov	Петров Пётр Петрович	CADET	Да

Рисунок 26 – тест-кейс 2

2 Безопасность жизнедеятельности

2.1 Характеристика рабочего места разработчика

Рабочее место разработчика относится к категории рабочих мест с длительным использованием персонального компьютера и высокой концентрацией внимания. Поэтому его организация должна обеспечивать не только удобство выполнения профессиональных задач, но и соответствовать требованиям эргономики, санитарных норм и правил охраны труда. Грамотно оборудованное рабочее пространство снижает риск профессиональных заболеваний, повышает работоспособность и способствует поддержанию оптимального уровня комфорта в течение всего рабочего дня.

Рабочее место включает персональный компьютер, монитор, клавиатуру, мышь и офисную мебель. Стол должен иметь достаточную площадь для размещения оборудования, документов и периферии, при этом не создавать излишнего напряжения для рук и плечевого пояса. Высота стола и кресла подбирается таким образом, чтобы предплечья располагались параллельно поверхности стола, а ноги стояли на полу или подставке под углом около 90°. Кресло должно иметь регулируемую спинку и поддержку поясничного отдела, что помогает снизить нагрузку на позвоночник при длительном сидении.

Монитор размещается на расстоянии 50–70 см от глаз, а его верхняя кромка должна находиться на уровне глаз или немного ниже. Такое положение уменьшает нагрузку на мышцы шеи и снижает риск зрительного утомления. Клавиатура располагается на расстоянии 10–30 см от края стола, а мышь — на одном уровне с ней, что обеспечивает естественное положение кистей.

2.2 Пожарная безопасность

Пожарная безопасность рабочего места разработчика определяется особенностями использования компьютерного оборудования, электрических приборов и периферии, которые при неправильной эксплуатации могут стать источником возгорания. Основная задача — создать условия, при которых риск возникновения пожара минимален, а персонал знает порядок действий в нештатной ситуации.

Рабочее помещение должно соответствовать требованиям пожарной безопасности, включающим исправное состояние электропроводки, наличие автоматических выключателей и использование сертифицированных удлинителей и сетевых фильтров. Компьютерная техника должна размещаться так, чтобы обеспечивалась свободная циркуляция воздуха вокруг системного блока и монитора, что предотвращает перегрев оборудования. Важно регулярно очищать технику от пыли, поскольку её накопление ухудшает охлаждение и повышает вероятность возгорания. Не допускается эксплуатация повреждённых кабелей, разъёмов и блоков питания, а также подключение большого количества устройств в одну розетку.

Помещение должно быть оснащено средствами пожаротушения, обычно порошковым или углекислотным огнетушителем, расположенным в зоне быстрого доступа. Сотрудники должны быть ознакомлены с его местоположением и правилами использования. На видных местах размещаются планы эвакуации и указатели направлений выхода. Важно, чтобы пути эвакуации не загромождались мебелью или оборудованием.

При появлении запаха гари, дыма или искрения необходимо немедленно прекратить работу, отключить оборудование от сети, если это безопасно, сообщить ответственному лицу и принять меры по локализации очага возгорания с помощью огнетушителя. В случае распространения огня помещение следует покинуть согласно плану эвакуации.

2.3 Психофизиологические нагрузки и профилактика утомления

Работа разработчика связана с длительным пребыванием за компьютером, высокой концентрацией внимания и выполнением однотипных действий, что приводит к значительным психофизиологическим нагрузкам. Основными факторами, влияющими на состояние организма, являются статическое напряжение мышц, зрительное утомление, монотонность работы и эмоциональное напряжение, возникающее при решении сложных технических задач или работе в условиях ограниченных сроков.

Длительное сидение в одной позе вызывает перенапряжение мышц спины, шеи и плечевого пояса, что может приводить к болевым ощущениям и снижению работоспособности. Высокая нагрузка на зрительный анализатор возникает из-за постоянного фокусирования взгляда на экране монитора, особенно при работе с мелкими элементами интерфейса или текстом. При недостаточном освещении или неправильном расположении монитора зрительное утомление усиливается, что может проявляться в виде сухости глаз, снижения четкости зрения и головных болей. Монотонность работы и необходимость длительной концентрации внимания нередко приводят к эмоциональному напряжению, снижению мотивации и повышенной утомляемости.

Для снижения психофизиологических нагрузок важна правильная организация режима труда и отдыха. Рекомендуется делать короткие перерывы каждые 45–60 минут, во время которых можно выполнить лёгкие упражнения для разминки мышц и переключить внимание на объекты, расположенные вдали, чтобы снизить нагрузку на глаза. Полезно использовать технику «20–20–20», предполагающую каждые 20 минут переводить взгляд на объект, находящийся на расстоянии около 20 футов (6 метров), в течение 20 секунд. Такие короткие паузы помогают восстановить зрительное восприятие и снизить напряжение глазных мышц.

3 Экономика

Расчёт стоимости разработки выполнен на основе модели трудозатрат по этапам жизненного цикла создания минимально жизнеспособного продукта (MVP). При расчёте учитывались работы, выполненные одним разработчиком полного цикла (full-stack developer), владеющим стеком React + FastAPI + PostgreSQL. Расчёт произведён в соответствии с методическими рекомендациями по оценке стоимости программных продуктов.

Исходные данные для расчёта:

- должность исполнителя : инженер-программист (full-stack developer).
- должность исполнителя: инженер-программист (full-stack developer).
- часовая тарифная ставка: 1 200 руб./час (установлена на основании среднерыночной стоимости оплаты труда разработчика соответствующей квалификации в регионе, с учётом страховых взносов и накладных расходов).
- продолжительность рабочего дня: 8 часов.
- общий срок выполнения работ: 3 календарных месяца.

В таблице 2 представлены трудозатраты.

Таблица 2 – Трудозатраты

Наименование этапа	Трудозатраты, чел.-час	Стоимость, руб.
Анализ предметной области, формирование требований	36	43 200

Наименование этапа	Трудозатраты, чел.-час	Стоимость, руб.
Проектирование архитектуры системы, структуры базы данных и API	40	48 000
Реализация серверной части (FastAPI, SQLAlchemy, PostgreSQL)	120	144 000
Реализация клиентской части (React, интерфейсы для всех ролей)	128	153 600
Интеграция компонентов, развёртывание и отладка	36	43 200
Тестирование (модульное, интеграционное, пользовательское) и оформление документации	40	48 000
Итого	400	480 000

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.02.001 ПЗ

Лист

61

Структура трудозатрат по видам работ.

Наибольший объём трудозатрат приходится на реализацию клиентской (32 %) и серверной (30 %) частей, что обусловлено сложностью разработки многофункциональных личных кабинетов, таблиц с динамическими данными и системы управления доступом. В совокупности этапы программирования составляют 62 % от общего объёма работ.

Оценка экономической эффективности.

Таким образом, расчётная стоимость разработки MVP веб-приложения «Информационная система процесса военно-учебного центра» составляет 480 000 рублей.

Внедрение разработанной системы позволит достичь следующих эффектов:

- сокращение временных затрат преподавателей и командиров на ведение журналов, оформление отчётности и поиск информации — ориентировочно на 30–40 %;
- минимизация рисков потери данных за счёт централизованного хранения и автоматического резервирования;
- повышение оперативности принятия управленческих решений благодаря актуальной аналитике по посещаемости и дисциплине;
- снижение количества ошибок при оформлении документации за счёт автоматической валидации и унификации форм.

Указанные факторы позволят окупить затраты на разработку в течение первого года эксплуатации системы за счёт повышения производительности труда сотрудников и улучшения качества управления учебным процессом.

Заключение

Разработанное веб-приложение представляет собой комплексную информационную систему, предназначенную для автоматизации процессов учёта посещаемости, управления учебными группами, документооборота и фиксации дисциплинарных записей. В ходе работы были реализованы все основные функциональные требования, сформирована удобная и интуитивно понятная клиентская часть, а также надёжный серверный модуль, обеспечивающий обработку запросов, хранение данных и контроль доступа на основе ролевой модели.

Использование современных технологий — React на стороне клиента, FastAPI на стороне сервера и PostgreSQL в качестве системы управления базами данных — позволило создать гибкую, масштабируемую и производительную архитектуру. Разделение логики на независимые компоненты обеспечило удобство сопровождения и возможность дальнейшего расширения функционала без существенных изменений в структуре приложения. Реализация REST-интерфейсов упростила взаимодействие между клиентом и сервером, обеспечив прозрачность и предсказуемость обмена данными.

Созданная система готова к эксплуатации и может быть использована в учебных подразделениях для оптимизации административных процессов, повышения прозрачности учёта и улучшения взаимодействия между участниками образовательного процесса. Архитектура приложения предусматривает возможность дальнейшего развития: добавления новых ролей, интеграции с внешними сервисами, расширения функционала расписания и аналитики. Это делает разработанное решение перспективным и адаптируемым под будущие задачи.

Список использованных источников

1. Perplexity AI [Электронный ресурс]. – URL: <https://www.perplexity.ai/> (дата обращения 26.04.2026). (В качестве промптов использовались запросы на поиск источников по заданной тематике и сумаризацию информации из найденных источников)
2. Норт Д., Уудс Д. Эволюционная архитектура. Поддерживаемые изменения в сложных системах. — СПб.: Питер, 2022. — 304 с.
3. Флэнаган Д. JavaScript. Подробное руководство. — 7-е изд. — СПб.: Питер, 2021. — 1088 с.
4. Хант Э., Томас Д. Программист-прагматик. Путь к мастерству. — СПб.: Питер, 2020. — 352 с.
5. Мартин Р. Чистый код. Создание, анализ и рефакторинг. — М.: Вильямс, 2021. — 464 с.
6. React Official Documentation. — URL: <https://react.dev> (дата обращения: 15.02.2026).
7. PostgreSQL Global Development Group. PostgreSQL Documentation. — URL: <https://www.postgresql.org/docs> (дата обращения: 05.02.2026).
8. Axios Documentation. — URL: <https://axios-http.com> (дата обращения: 12.02.2026).
9. Глушаков С. В. Разработка веб-приложений на Python. — М.: БХВ-Петербург, 2022. — 368 с.
10. FastAPI Documentation. — URL: <https://fastapi.tiangolo.com> (дата обращения: 10.02.2026).
11. Федеральный закон «О воинской обязанности и военной службе» № 53-ФЗ от 28.03.1998 г.
12. Методические рекомендации по организации учебного процесса в военно-учебных центрах при вузах. — Минобороны РФ, 2022.

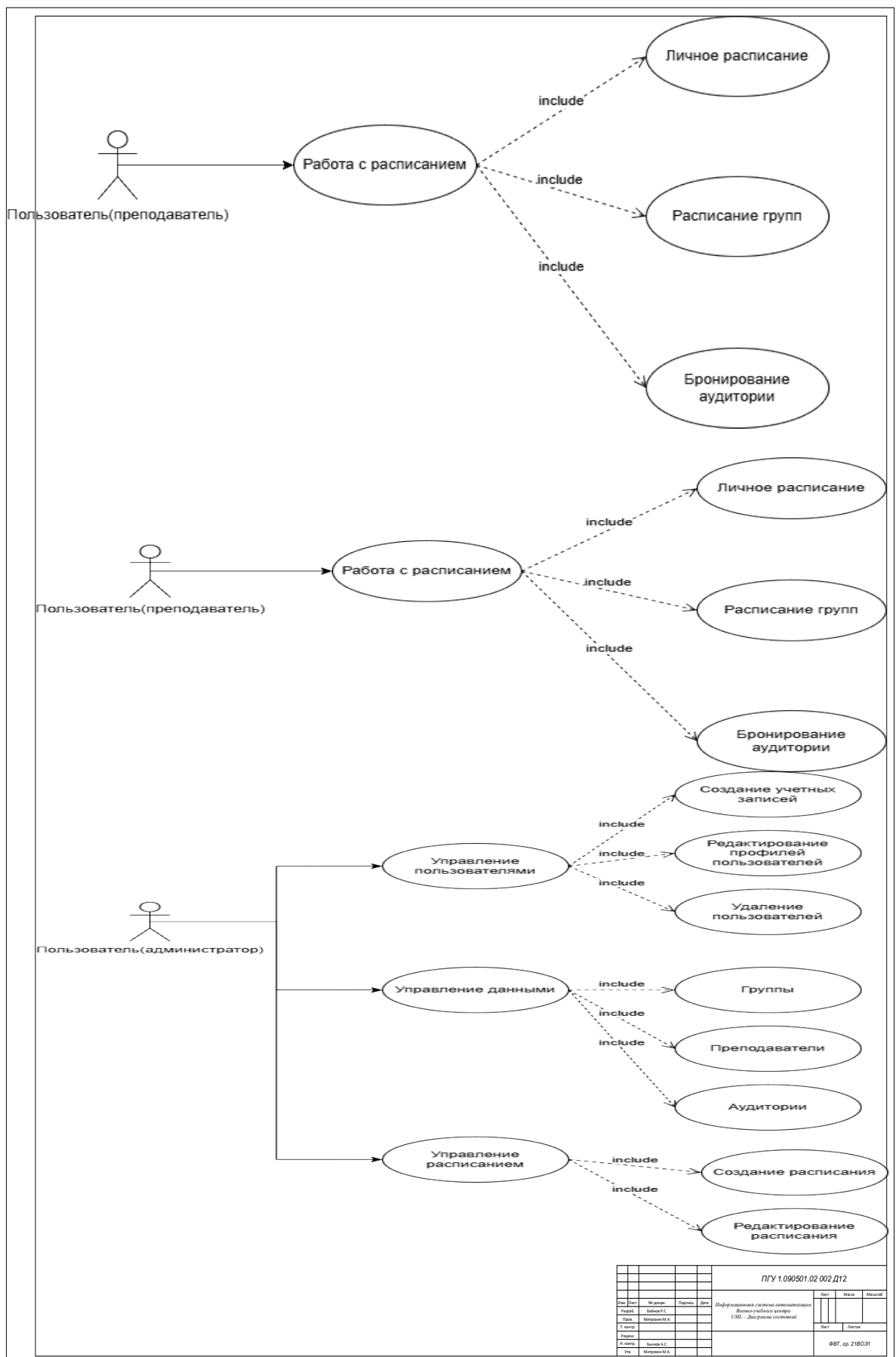
Приложение А

(обязательное)

Информационная система автоматизации образовательного процесса военно-
учебного центра
UML-диаграммы

					ПГУ 1.090501.02.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		63

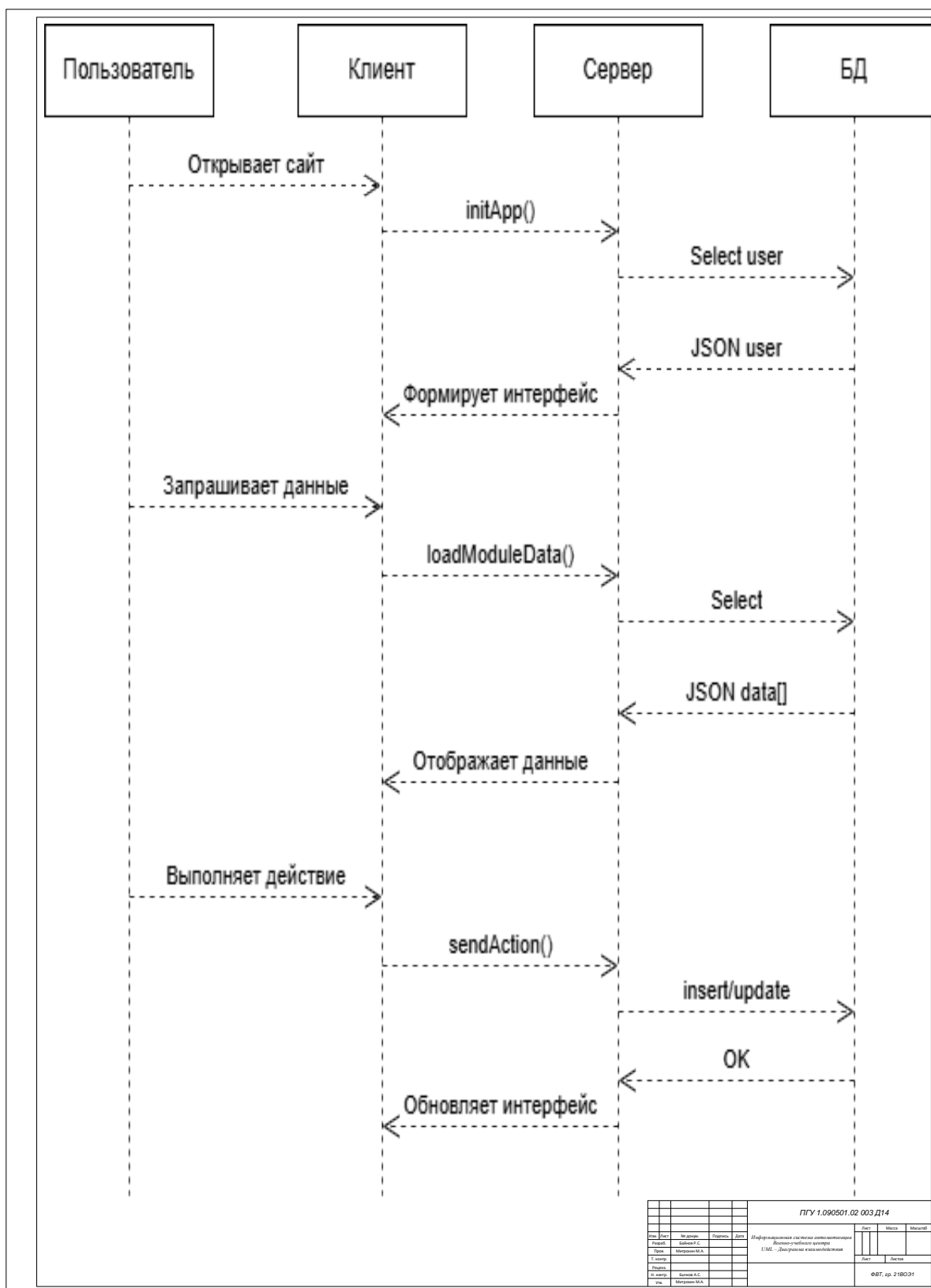
UML – диаграмма вариантов использования



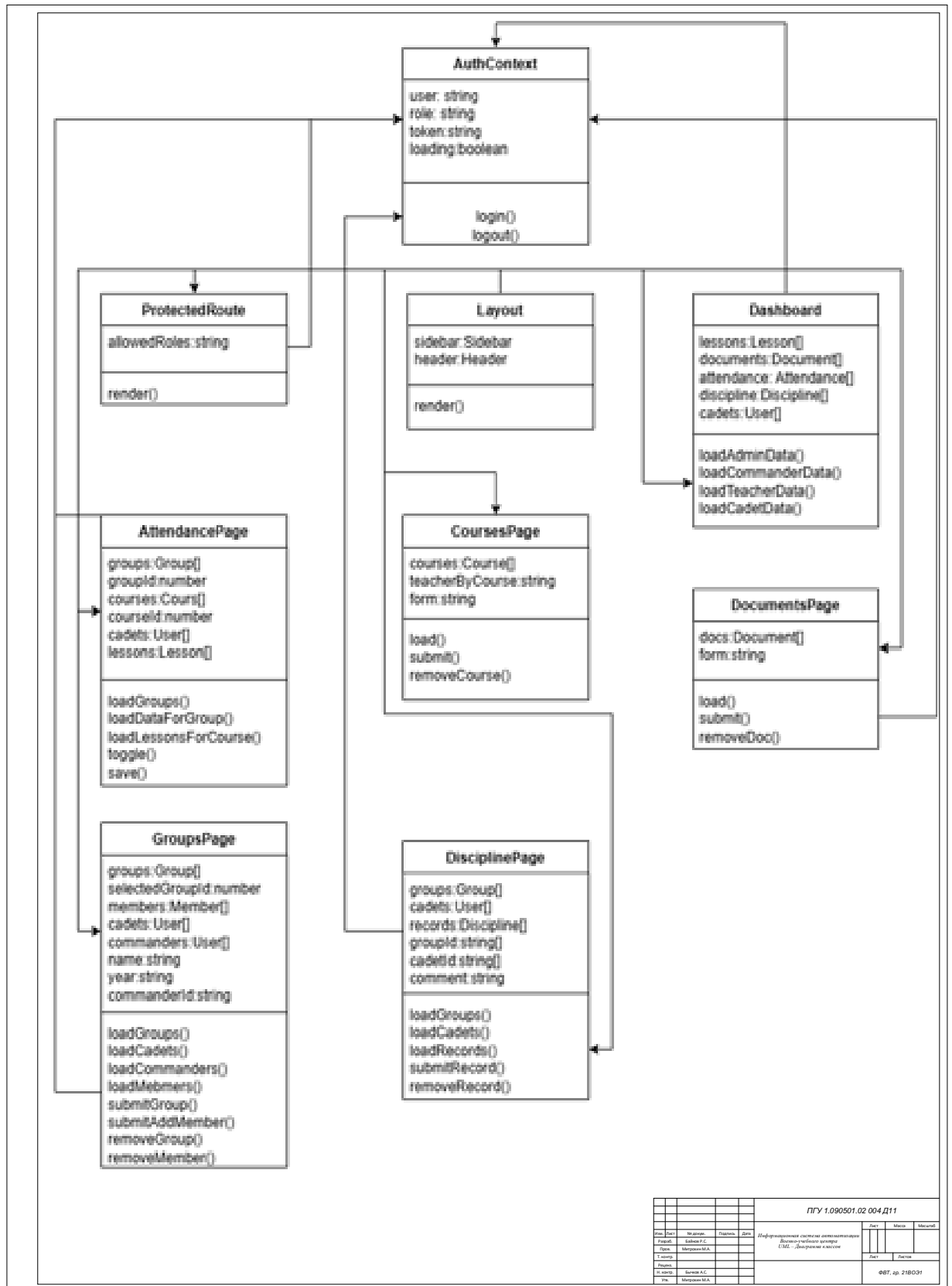
ПГУ 1.090501.02.002.Д12									
Имя	Фамилия	№ докум.	Пароль	Дата	Информационная система автоматизации Вопросно-ответной системы СНС – для работы с системой				
Логин	Пароль	Имя	Фамилия	Дата					
Имя	Фамилия	№ докум.	Пароль	Дата	ПГУ 1.090501.02.002.Д12				
Имя	Фамилия	№ докум.	Пароль	Дата	ПГУ 1.090501.02.002.Д12				

Изм.	Лист	№ докум.	Подпись	Дата

UML – диаграмма взаимодействия



UML – диаграмма классов



ПГУ 1.090501.02.004 Д11									
Имя	Фамилия	Матрица	Пароль	Дата	Информационная система автоматизации Вспомогательная система UML – Диаграмма классов				
Имя	Фамилия	Матрица	Пароль	Дата	Лист	Место	Масштаб		
Имя	Фамилия	Матрица	Пароль	Дата	Лист	Место	Масштаб		
Имя	Фамилия	Матрица	Пароль	Дата	Лист	Место	Масштаб		
Имя	Фамилия	Матрица	Пароль	Дата	Лист	Место	Масштаб		
Имя	Фамилия	Матрица	Пароль	Дата	Лист	Место	Масштаб		
Имя	Фамилия	Матрица	Пароль	Дата	Лист	Место	Масштаб		
Имя	Фамилия	Матрица	Пароль	Дата	Лист	Место	Масштаб		
Имя	Фамилия	Матрица	Пароль	Дата	Лист	Место	Масштаб		

687, стр. 21/03/21

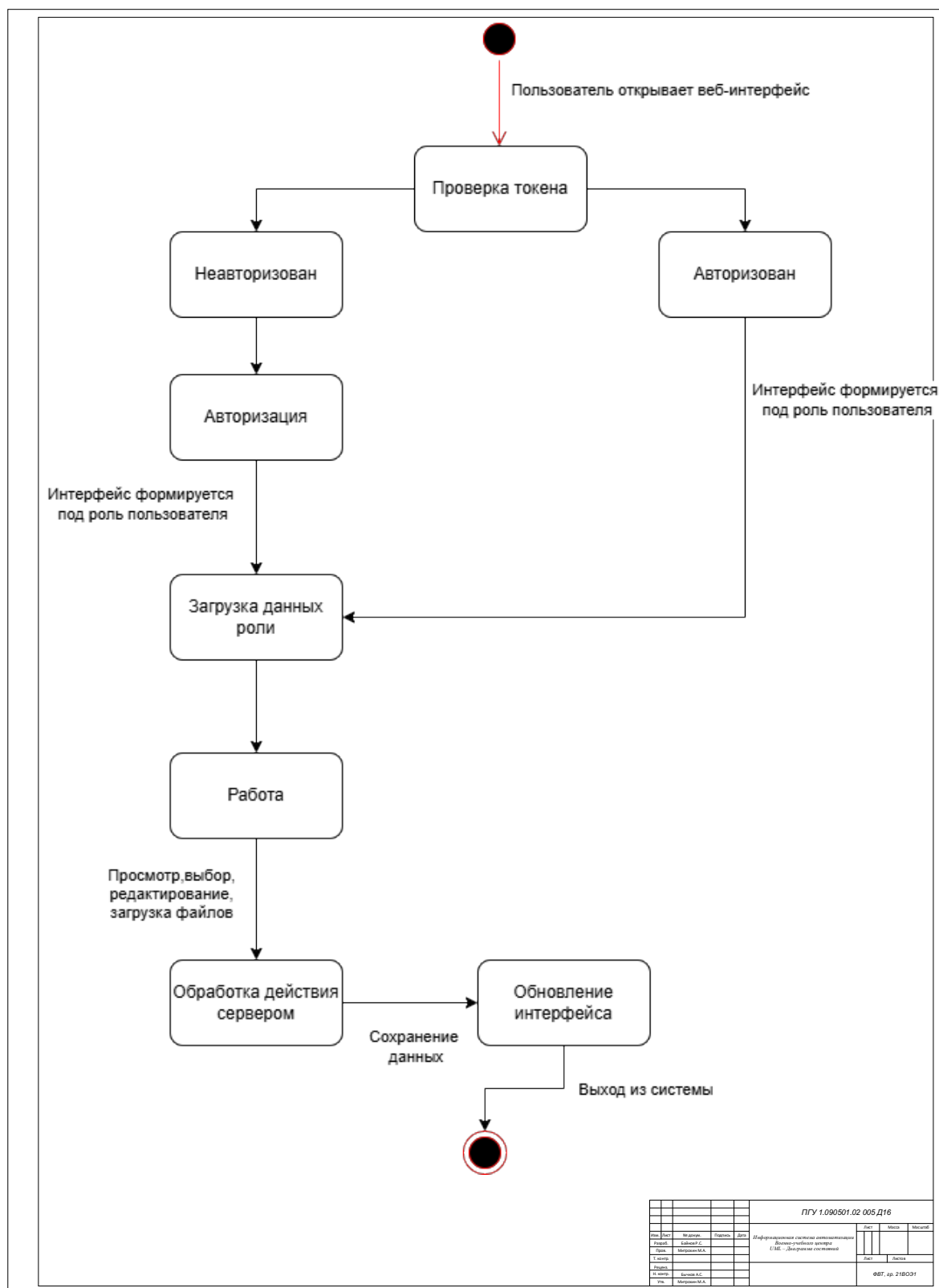
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.02.001 ПЗ

Лист

63

UML – диаграмма состояний



Приложение Б

(обязательное)

Информационная система автоматизации образовательного процесса военно-
учебного центра

Листинг

					ПГУ 1.090501.02.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		65

Файл «AttendancePage.jsx»

```
import React, { useEffect, useState } from "react";
import { getAttendance } from "../api/attendance";
import { getSchedule } from "../api/schedule";
import { getGroups, getGroupMembers } from "../api/groups";
import { getCourses } from "../api/courses";
import { useAuth } from "../hooks/useAuth";
import "../AttendancePage.css";

export default function AttendancePage() {
  const { role, user } = useAuth();
  const isCadet = role === "CADET";

  const [groups, setGroups] = useState([]);
  const [groupId, setGroupId] = useState("");

  const [courses, setCourses] = useState([]);
  const [courseId, setCourseId] = useState("");

  const [cadets, setCadets] = useState([]);
  const [lessons, setLessons] = useState([]);
  const [attendance, setAttendance] = useState({});

  useEffect(() => {
    loadGroups();
  }, []);

  async function loadGroups() {
    const all = await getGroups();
```

```

if (isCadet) {
    // Находим группу кадета
    const group = all.find((g) => g.id === user.group_id);
    if (group) {
        setGroups([group]);
        setGroupId(group.id);
        loadDataForGroup(group.id);
    }
} else {
    setGroups(all);
}

}

    async function loadDataForGroup(id) {
if (!id) return;

const members = await getGroupMembers(id);
const cadets = members.map((m) => ({
    id: m.cadet_id,
    full_name: m.full_name,
}));

const courses = await getCourses();

setCadets(cadets);
setCourses(courses);
setCourseId("");
setLessons([]);
setAttendance({});
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

    async function loadLessonsForCourse(courseId) {
        if (!courseId) return;

        const lessons = await getSchedule({
            group_id: groupId,
            course_id: courseId,
        });

        const records = await getAttendance();

        const map = {};

        cadets.forEach((c) => {
            map[c.id] = {};
            lessons.forEach((l) => {
                const rec = records.find(
                    (r) => r.cadet_id === c.id && r.schedule_item_id ===
l.id
                );
                map[c.id][l.id] = rec ? rec.present : false;
            });
        });

        setLessons(lessons);
        setAttendance(map);
    }

    function toggle(cadetId, lessonId) {
        if (isCadet) return; // Кадет не может менять

        setAttendance((prev) => ({

```

```

    ...prev,
    [cadetId]: {
      ...prev[cadetId],
      [lessonId]: !prev[cadetId][lessonId],
    },
  ));
}

async function save() {
  if (isCadet) return; // Кадет не может сохранять

  for (const cadetId of Object.keys(attendance)) {
    for (const lessonId of Object.keys(attendance[cadetId])) {
      await markAttendance({
        schedule_item_id: Number(lessonId),
        cadet_id: Number(cadetId),
        present: attendance[cadetId][lessonId],
      });
    }
  }

  alert("Посещаемость сохранена");
}

return (
  <div className="attendance-container">
    <h2 className="page-title">Посещаемость по дисциплинам</h2>

    { /* Выбор группы – скрыт для кадета */ }
    { !isCadet && (
      <div className="card">
        <select
          value={groupId}
          onChange={ (e) => {
            const id = e.target.value;
            setGroupId(id);

```

>

```
<option value="">Выберите группу</option>

{groups.map((g) => (

  <option key={g.id} value={g.id}>

    {g.name}

  </option>

))}

</select>

</div>

)}
```

```
{/* Для кадета группа уже выбрана автоматически */}

{isCadet && groupId && (

  <div className="card">

    <div className="readonly-info">

      Группа: <b>{groups[0]?.name}</b>

    </div>

  </div>

)}
```

```
{/* Выбор дисциплины */}

{groupId && (

  <div className="card">

    <select

      value={courseId}

      onChange={(e) => {
```

Изм.	Лист	№ докум.	Подпись	Дата

```

<option value="">Выберите дисциплину</option>
    {courses.map((c) => (
        <option key={c.id} value={c.id}>
            {c.title}
        </option>
    ))}
</select>
</div>
    )}

/* Таблица */}
{courseId && lessons.length > 0 && (
    <div className="card">
        <h3 className="card-title">
            {courses.find((c) => c.id === courseId)?.title}
        </h3>

        <div className="table-wrapper">
            <table className="attendance-table">
                <thead>
                    <tr>
                        <th className="sticky-col">Курсант</th>
                        {lessons.map((l) => (
                            <th key={l.id}>
                                {new
Date(l.start_time).toLocaleDateString()}
                            </th>
                        ))}
                    </tr>
                </thead>

```

```

<tbody>

    {cadets.map((c) => (
        <tr key={c.id}>
            <td className="sticky-col">{c.full_name}</td>

            {lessons.map((l) => (
                <td key={l.id} className="center">
                    <input
                        type="checkbox"
                        className="attendance-checkbox"
                        checked={attendance[c.id]?.[l.id] ||
false}

                        onChange={() => toggle(c.id, l.id)}
                        disabled={isCadet} // Кадет не может
менять

                    />
                </td>
            ))}
        </tr>
    ))}
</tbody>
</table>
</div>

{/* Кнопка сохранения — скрыта для кадета */}
    {!isCadet && (
        <button className="btn-primary save-btn" onClick={save}>
            Сохранить посещаемость
        </button>
    )}
</div>

    )}
</div>

);
}

```

Файл «CoursesPage.jsx»

```
import React, { useEffect, useState } from "react";
import { getCourses, createCourse, deleteCourse } from
"../api/courses";
import { getSchedule } from "../api/schedule";
import { useAuth } from "../hooks/useAuth";
import "../CoursesPage.css";

export default function CoursesPage() {
  const { role } = useAuth();
  const isCadet = role === "CADET";

  const [courses, setCourses] = useState([]);
  const [teachersByCourse, setTeachersByCourse] = useState({});
  const [form, setForm] = useState({ title: "", description: "" });

  async function load() {
    const data = await getCourses();
    setCourses(data);

    // Загружаем расписание, чтобы собрать преподавателей по курсам
    const schedule = await getSchedule();

    const map = {};
    schedule.forEach((item) => {
      if (!map[item.course_id]) map[item.course_id] = new Set();
      map[item.course_id].add(item.teacher_name);
    });
    const normalized = {};
    Object.keys(map).forEach((cid) => {
      normalized[cid] = Array.from(map[cid]);
    });

    setTeachersByCourse(normalized);
  }
}
```

```

useEffect(() => {
    load();
}, []);

async function submit(e) {
    e.preventDefault();
    if (isCadet) return;

    await createCourse(form);
    setForm({ title: "", description: "" });
    load();
}

async function removeCourse(id) {
    if (isCadet) return;

    await deleteCourse(id);
    load();
}

return (
    <div className="courses-container">
        <h2 className="page-title">Учебные дисциплины</h2>

        {/* Форма — скрыта для кадета */}
        {!isCadet && (
            <div className="card">
                <h3 className="card-title">Создать дисциплину</h3>

                <form onSubmit={submit} className="form-grid">
                    <input
                        placeholder="Название"
                        value={form.title}
                        onChange={(e) => setForm({ ...form, title:
e.target.value })}
                    />

```

```

<input
    placeholder="Описание"
    value={form.description}
    onChange={ (e) =>
        setForm({ ...form, description: e.target.value })
    }
/>

<button type="submit" className="btn-primary form-
submit">

    Создать
</button>
</form>
</div>
)}
{ /* Таблица */}
<div className="card">
    <h3 className="card-title">Список дисциплин</h3>

    <table className="styled-table">
        <thead>
            <tr>
                {!isCadet && <th>ID</th>}
                <th>Название</th>
                <th>Описание</th>
                <th>Преподаватели</th>
                {!isCadet && <th></th>}
            </tr>
        </thead>

```

```

<tbody>

    {courses.map((c) => (
        <tr key={c.id}>
            {!isCadet && <td>{c.id}</td>}
            <td>{c.title}</td>
            <td>{c.description}</td>

            { /* Преподаватели */ }
            <td>
                {(teachersByCourse[c.id] || []).join(", ") || "-"}
            </td>

        { /* Удаление — только не для кадета */ }
        {!isCadet && (
            <td>
                <button
                    className="btn-delete"
                    onClick={() => removeCourse(c.id)}
                >
                    Удалить
                </button>
            </td>
        )}
    </tr>
    )})
</tbody>
</table>
</div>
</div>
);
}

```

Файл «DashBoard.jsx»

```
import React, { useEffect, useState } from "react";
import { useAuth } from "../hooks/useAuth";
import { getSchedule } from "../api/schedule";
import { getDocuments } from "../api/documents";
import { getDiscipline } from "../api/discipline";
import { getAttendance } from "../api/attendance";
import { getGroupMembers, getGroups } from "../api/groups";
import { getUsers } from "../api/users";
import "../Dashboard.css";

export default function Dashboard() {
  const { role, user, loading } = useAuth();

  // Общие данные
  const [lessons, setLessons] = useState([]);
  const [documents, setDocuments] = useState([]);
  const [discipline, setDiscipline] = useState([]);
  const [attendance, setAttendance] = useState([]);
  // Командир
  const [cadets, setCadets] = useState([]);

  // Админ
  const [users, setUsers] = useState([]);
  const [groups, setGroups] = useState([]);

  useEffect(() => {
    if (!loading && user) {
      if (role === "ADMIN") loadAdminData();
      else if (role === "COMMANDER") loadCommanderData();
      else if (role === "TEACHER") loadTeacherData();
      else if (role === "CADET") loadCadetData();
    }
  });
}
```

```

    }, [loading, user]);

    // -----
    // ADMIN PANEL
    // -----
    async function loadAdminData() {
        const usersData = await getUsers();
        const groupsData = await getGroups();
        const docs = await getDocuments();

        setUsers(usersData);
        setGroups(groupsData);
        setDocuments(docs.slice(0, 5));
    }
    // -----
    // COMMANDER PANEL
    // -----
    async function loadCommanderData() {
        const groupId = user.group_id;

        const members = await getGroupMembers(groupId);
        setCadets(members);

        const today = new Date().toISOString().split("T")[0];
        const lessonsData = await getSchedule({ date: today, group_id:
groupId });
        setLessons(lessonsData.slice(0, 3));

        const disc = await getDiscipline({ group_id: groupId });
        setDiscipline(disc.slice(0, 3));

        const att = await getAttendance();
        setAttendance(att);
    }

```

Изм.	Лист	№ докум.	Подпись	Дата

```

    }

    // -----
    // TEACHER PANEL
    // -----
    async function loadTeacherData() {
        const lessonsData = await getSchedule({ teacher_id: user.id });
        setLessons(lessonsData.slice(0, 5));

        const att = await getAttendance();
        setAttendance(att.filter((a) => a.teacher_id === user.id));

        const docs = await getDocuments();
        setDocuments(docs.slice(0, 3));
    }

    // -----
    // CADET PANEL
    // -----
    async function loadCadetData() {
        const today = new Date().toISOString().split("T")[0];

        const lessonsData = await getSchedule({
            date: today,
            group_id: user.group_id,
        });
        setLessons(lessonsData.slice(0, 3));

        const docs = await getDocuments();
        setDocuments(docs.slice(0, 3));

        const disc = await getDiscipline({ cadet_id: user.id });
        setDiscipline(disc.slice(0, 3));
    }

    if (loading || !user) {
        return <div className="dashboard">Загрузка...</div>;
    }

```

```

// -----
// ADMIN RENDER
// -----

if (role === "ADMIN") {
    const cadetsCount = users.filter((u) => u.role ===
"CADET").length;
    const commandersCount = users.filter((u) => u.role ===
"COMMANDER").length;
    const teachersCount = users.filter((u) => u.role ===
"TEACHER").length;

    return (
        <div className="dashboard">
            <h2>Административная панель</h2>
            <p>Добро пожаловать, {user.full_name}. Ваша роль:
<b>ADMIN</b></p>

            <div className="dashboard-grid">

                <div className="card">
                    <h3>Пользователи</h3>
                    <div className="item">Всего: {users.length}</div>
                    <div className="item">Кадеты: {cadetsCount}</div>
                    <div className="item">Командиры:
{commandersCount}</div>
                    <div className="item">Преподаватели:
{teachersCount}</div>
                </div>

                <div className="card">
                    <h3>Группы</h3>
                    <div className="item">Всего групп:
{groups.length}</div>
                    <div className="item">Всего кадетов:
{cadetsCount}</div>
                </div>
            </div>
        </div>
    );
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

<div className="card">

    <h3>Последние документы</h3>

    {documents.map((d) => (

        <div key={d.id} className="item">{d.title}</div>

    ))}

</div>

<div className="card">

    <h3>Новые пользователи</h3>

    {users.slice(-5).map((u) => (

        <div key={u.id} className="item">

            {u.full_name} — {u.role}

        </div>

    ))}

</div>

</div>

</div>

);

}

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.02.001 ПЗ

Лист

97

```

return (
    <div className="dashboard">
        <h2>Панель командира</h2>
        <p>Добро пожаловать, {user.full_name}. Ваша группа:
<b>{user.group_name}</b></p>

        <div className="dashboard-grid">

            <div className="card">
                <h3>Состояние группы</h3>
                <div className="item">Кадетов: {cadets.length}</div>
                <div className="item">Присутствуют:
{todayPresent.length}</div>
                <div className="item">Отсутствуют:
{todayAbsent.length}</div>
            </div>

            <div className="card">
                <h3>Отсутствующие сегодня</h3>
                {todayAbsent.length === 0 && <p>Все присутствуют</p>}
                {todayAbsent.map((a) => (
                    <div key={a.id} className="item">{a.cadet_name}</div>
                ))}
            </div>

```

```

<div className="card">
    <h3>Ближайшие занятия</h3>
    {lessons.map((l) => (
        <div key={l.id} className="item">
            <b>{l.course_title}</b> – {new
Date(l.start_time).toLocaleTimeString([], { hour: "2-digit", minute: "2-
digit" })}

        </div>
    ))}
</div>

<div className="card">
    <h3>Дисциплина группы</h3>
    {discipline.map((r) => (
        <div key={r.id} className="item">
            {r.cadet_name}: {r.violation_type}
        </div>
    ))}
</div>

</div>
</div>
);
}

```

```

// -----
// TEACHER RENDER
// -----

if (role === "TEACHER") {
  const today = new Date().toISOString().split("T")[0];
  const todayLessons = lessons.filter(
    (l) => l.start_time.split("T")[0] === today
  );
  const groupsToday = [...new Set(todayLessons.map((l) =>
l.group_name))];

  return (
    <div className="dashboard">
      <h2>Панель преподавателя</h2>
      <p>Добро пожаловать, {user.full_name}. Ваши занятия на
сегодня.</p>

      <div className="dashboard-grid">

        <div className="card">
          <h3>Ближайшие занятия</h3>
          {lessons.map((l) => (
<div key={l.id} className="item">
          <b>{l.course_title}</b> — {l.group_name} —{" "}
            {new Date(l.start_time).toLocaleTimeString([], { hour:
"2-digit", minute: "2-digit" })}
          </div>
          ))}
        </div>

        <div className="card">
          <h3>Прошедшие занятия</h3>
          {lessons.map((l) => (
<div key={l.id} className="item">
          <b>{l.course_title}</b> — {l.group_name} —{" "}
            {new Date(l.start_time).toLocaleTimeString([], { hour:
"2-digit", minute: "2-digit" })}
          </div>
          ))}
        </div>
      )}
    </div>
  );
}

```

```

<div className="card">
  <h3>Группы сегодня</h3>
  {groupsToday.map((g) => (
    <div key={g} className="item">{g}</div>
  ))}
</div>

  <div className="card">
    <h3>Последняя посещаемость</h3>
    {attendance.slice(0, 5).map((a) => (
      <div key={a.id} className="item">
        {a.cadet_name}: {a.present ? "Присутствовал" :
"Отсутствовал"}
      </div>
    ))}
  </div>

</div>
</div>
);
}

// -----
// CADET RENDER
// -----

return (
  <div className="dashboard">
    <h2>Главная панель</h2>
    <p>Добро пожаловать, {user.full_name}. Ваша роль:
<b>CADET</b></p>
    <div className="dashboard-grid">

      <div className="card">
        <h3>Ближайшие занятия</h3>
        {lessons.map((l) => (
          <div key={l.id} className="item">
            <b>{l.course_title}</b> -{" "}
            {new Date(l.start_time).toLocaleTimeString([], {
hour: "2-digit", minute: "2-digit" })}
          </div>
        ))}
      </div>

    )}
  </div>

```

```

<div className="card">
  <h3>Последние документы</h3>
  {documents.map((d) => (
    <div key={d.id} className="item">{d.title}</div>
  ))}
</div>

<div className="card">
  <h3>Дисциплина</h3>
  {discipline.map((r) => (
    <div key={r.id} className="item">
      {r.violation_type}
    </div>
  ))}
</div>

</div>
</div>
);
}

```

Файл «routes_attendance.py»

```

from typing import List
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session

from app.api.deps import get_db, require_role
from app.models.user import UserRole, User
from app.models.attendance import Attendance
from app.models.schedule import ScheduleItem
from app.schemas.attendance import AttendanceCreate, AttendanceOut

router = APIRouter(prefix="/attendance", tags=["attendance"])

# -----
# Создать или обновить посещаемость (только ADMIN, TEACHER,
COMMANDER)
# -----
@router.post("/", response_model=AttendanceOut)
def mark_attendance(
    att_in: AttendanceCreate,
    db: Session = Depends(get_db),
    _: User = Depends(require_role(
        UserRole.ADMIN,
        UserRole.TEACHER,
        UserRole.COMMANDER
    )),
):
    item = db.query(ScheduleItem).get(att_in.schedule_item_id)
    if not item:
        raise HTTPException(404, "Занятие не найдено")

```

```

cadet = db.query(User).get(att_in.cadet_id)
if not cadet or cadet.role != UserRole.CADET:
    raise HTTPException(400, "Указанный пользователь не
является кадетом")

existing = db.query(Attendance).filter(
    Attendance.schedule_item_id == att_in.schedule_item_id,
    Attendance.cadet_id == att_in.cadet_id
).first()

if existing:
    existing.present = att_in.present
    db.commit()
    db.refresh(existing)
    return existing
record = Attendance(**att_in.dict())
db.add(record)
db.commit()
db.refresh(record)
return record

-----
# Получить посещаемость (теперь CADET тоже может)
# -----
@router.get("/", response_model=List[AttendanceOut])
def list_attendance(
    schedule_item_id: int | None = None,
    cadet_id: int | None = None,
    db: Session = Depends(get_db),
    _: User = Depends(require_role(
        UserRole.ADMIN,
        UserRole.COMMANDER,
        UserRole.TEACHER,
        UserRole.CADET,
    )),

```

```

q = db.query(Attendance)

if schedule_item_id:
    q = q.filter(Attendance.schedule_item_id == schedule_item_id)

if cadet_id:
    q = q.filter(Attendance.cadet_id == cadet_id)

return q.all()

```

Файл «routes_auth.py»

```

from datetime import timedelta

from fastapi import APIRouter, Depends, HTTPException, status
from fastapi.security import OAuth2PasswordRequestForm
from sqlalchemy.orm import Session

from app.api.deps import authenticate_user, get_db
from app.core.config import settings
from app.core.security import create_access_token
from app.schemas.auth import Token
from app.schemas.user import UserOut

router = APIRouter(prefix="/auth", tags=["auth"])
@router.post("/login", response_model=Token)
def login(
    form_data: OAuth2PasswordRequestForm = Depends(),
    db: Session = Depends(get_db),
):
    user = authenticate_user(db, form_data.username,
form_data.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Неверное имя пользователя или пароль",
            headers={"WWW-Authenticate": "Bearer"},
        )

    access_token_expires =
timedelta(minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = create_access_token(
        subject=str(user.id),
        expires_delta=access_token_expires,
    )
    return Token(
        access_token=access_token,
        user=UserOut.from_orm(user)
    )

```

Файл «routes_courses.py»

```

from typing import List
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session

from app.api.deps import get_db, require_role
from app.models.user import UserRole, User
from app.models.course import Course
from app.schemas.course import CourseCreate, CourseUpdate,
CourseOut

router = APIRouter(prefix="/courses", tags=["courses"])

    @router.post("/",
    response_model=CourseOut)
    def create_course(
        course_in:
        CourseCreate,
        db: Session =
        Depends(get_db),
        _: User =
        Depends(require_role(UserR
        ole.ADMIN,
        UserRole.COMMANDER)),
    ):
        course =
        Course(**course_in.dict())
        db.add(course)
        db.commit()
        db.refresh(course)
        return course

```

</div>

<div className="dashboard-card">

<div className="dashboard-label">Всего слотов</div>

<div className="dashboard-value">{todayStats.total}</div>

</div>

</div>

{/* ===== Быстрые действия ===== */}

<h2 style={{ marginTop: 30 }}>Быстрые действия</h2>

<div style={{ display: "flex", gap: 20 }}>

<Button onClick={() => navigate("/schedule")}>Моё
расписание</Button>

<Button onClick={() => navigate("/doctor/appointments")}>

Мои приёмы

</Button>

<Button onClick={() => navigate("/patients")}>Мои
пациенты</Button>

<Button onClick={() => navigate("/records")}>Мед.
записи</Button>

</div>

<h2 style={{ marginTop: 40 }}>Ближайшие приёмы</h2>

{upcomingAppointments.length === 0 && (

<div>Нет предстоящих приёмов</div>

)}

<div style={{ display: "flex", flexDirection: "column",
gap: 10 }}>

{upcomingAppointments.slice(0, 5).map((a) => {

const slot = a.slot;

const patient = extractPatient(a);

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.02.001 ПЗ

Лист

107

```

const patientName =
    patient?.last_name && patient?.first_name
    ? `${patient.last_name} ${patient.first_name}`
    : patient?.full_name
    ? patient.full_name
    : patient?.insurance_number
    ? patient.insurance_number
    : "Пациент";

```

```

const patientId = extractPatientId(patient, a);

```

```

return (
    <div
        key={a.id}
        style={{
            padding: 12,
            border: "1px solid #d4d9e5",
            borderRadius: 6,
            display: "flex",
            justifyContent: "space-between"
        }}
    >

```

```

<div>

    <b>
        {new Date(slot.start_time).toLocaleString("ru-
RU", {

            year: "numeric",
            month: "2-digit",
            day: "2-digit",
            hour: "2-digit",
            minute: "2-digit"

        })}
    </b>{" "}
    - {patientName}
    <div style={{ fontSize: 13, color: "#666" }}>
        {a.complaint || "Без жалобы"}
    </div>

</div>

<Button

    onClick={() => {

        if (!patientId) {

            alert("У этого приёма нет

пациента");

            return;

        }

        navigate(`/doctor/patient/${patient
Id}`);

    }}

>

```

Открыть

```
        </Button>
      </div>
    );
  }}}
</div>
</div>
);
}
/ -----
// ADMIN DASHBOARD
// -----
if (role === "admin") {
  return (
    <div>
      <h1 style={{ marginBottom: 20 }}>Панель управления</h1>

      {summary && (
        <div className="dashboard-grid">
          <div className="dashboard-card">
            <div className="dashboard-label">Приёмов
сегодня</div>

            <div className="dashboard-
value">{summary.appointments}</div>
          </div>
```

```


|      |      |          |         |      |                        |      |
|------|------|----------|---------|------|------------------------|------|
|      |      |          |         |      | ПГУ 1.090501.02.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                        | 111  |


```

```

from fastapi import APIRouter

from . import auth_routes, patient_routes, doctor_routes, appointment_routes
from . import medical_record_routes, drug_routes, report_routes
from .specialty_routes import router as specialty_router
from .documents_routes import router as documents_router


api_router = APIRouter(prefix="/api")


api_router.include_router(auth_routes.router)
api_router.include_router(patient_routes.router)
api_router.include_router(doctor_routes.router)
api_router.include_router(appointment_routes.router)
api_router.include_router(medical_record_routes.router)
api_router.include_router(drug_routes.router)
api_router.include_router(report_routes.router)
api_router.include_router(specialty_router)
api_router.include_router(documents_router) #
    
```

Файл «appointment_router.py»

```
# app/api/appointment_routes.py

from datetime import datetime
from fastapi import APIRouter, Depends, Query, status,
HTTPException
from sqlalchemy.orm import Session, joinedload

from ..db import get_db
from ..schemas import (
    AppointmentSlotCreate,
    AppointmentSlotRead,
    AppointmentCreate,
    AppointmentUpdateStatus,
    AppointmentRead,
    ScheduleGenerationRequest,
    AppointmentCancelRequest,
    AppointmentRescheduleRequest,
)
from ..services.appointment_service import AppointmentService
from .deps import get_current_user

from app.models.appointment import Appointment, AppointmentSlot
from app.models.patient import Patient
from app.models.doctor import Doctor

from app.utils.email_sender import send_email

router = APIRouter(prefix="/appointments", tags=["appointments"])
```

```

    @router.post("/slots", response_model=AppointmentSlotRead,
status_code=201)
    def create_slot(
        data: AppointmentSlotCreate,
        db: Session = Depends(get_db),
        user=Depends(get_current_user),
    ):
        return AppointmentService.create_slot(db, data)

    @router.get("/slots/{doctor_id}",
response_model=list[AppointmentSlotRead])
    def list_slots_for_doctor(
        doctor_id: int,
        start_from: datetime | None = Query(None),
        with_appointments: bool = Query(False),
        db: Session = Depends(get_db),
        user=Depends(get_current_user),
    ):
        return AppointmentService.list_slots_for_doctor(
            db, doctor_id, start_from, with_appointments
        )

```

```

# -----
# ГЕНЕРАЦИЯ РАСПИСАНИЯ
# -----
@router.post("/slots/generate/{doctor_id}",
response_model=list[AppointmentSlotRead], status_code=201)
def generate_schedule_for_doctor(
    doctor_id: int,
    data: ScheduleGenerationRequest,
    db: Session = Depends(get_db),
    user=Depends(get_current_user),
):
    return AppointmentService.generate_schedule_for_doctor(db,
doctor_id, data)

@router.post("/", response_model=AppointmentRead, status_code=201)
def create_appointment_for_patient(
    data: AppointmentCreate,
    db: Session = Depends(get_db),
    user=Depends(get_current_user),
):
    appointment =
AppointmentService.create_appointment_for_patient(db, user.id, data)

# Email пациенту
if user.email:
    slot = appointment.slot
    doctor = slot.doctor

```

```

text = (
    f"Вы успешно записались на приём.\n\n"
    f"Врач: {doctor.last_name} {doctor.first_name}\n"
    f"Дата: {slot.start_time}\n"
    f"Статус: {appointment.status}\n\n"
    f"Спасибо, что выбрали MedClinicIS!"
)

send_email(
    to_email=user.email,
    subject="Запись на приём – подтверждение",
    text=text,
)

return appointment

# -----
# ОБНОВЛЕНИЕ СТАТУСА
# -----

@router.patch("/{appointment_id}/status",
response_model=AppointmentRead)
def update_status(
    appointment_id: int,
    data: AppointmentUpdateStatus,
    db: Session = Depends(get_db),
    user=Depends(get_current_user),
):
    return AppointmentService.update_status(db, appointment_id,
data)

@router.get("/patient/{patient_id}",
response_model=list[AppointmentRead])

```

```

def list_appointments_for_patient(
    patient_id: int,
    db: Session = Depends(get_db),
    user=Depends(get_current_user),
):
    return AppointmentService.list_for_patient(db, patient_id)

# -----
# МОИ ПРИЁМЫ (ПАЦИЕНТ)
# -----
@router.get("/my", response_model=list[AppointmentRead])
def list_my_appointments(
    db: Session = Depends(get_db),
    user=Depends(get_current_user),
):
    patient = db.query(Patient).filter(Patient.user_id ==
user.id).first()
    if not patient:
        raise HTTPException(status_code=404, detail="Пациент не
найден")

    return AppointmentService.list_for_patient(db, patient.id)

@router.get("/doctor/my", response_model=list[AppointmentRead])
def get_my_appointments_as_doctor(
    db: Session = Depends(get_db),
    current_user=Depends(get_current_user),

```

1. Находим врача, привязанного к пользователю

```
doctor = db.query(Doctor).filter(Doctor.user_id == current_user.id).first()
```

```
if not doctor:
```

```
    return []
```

2. Возвращаем ВСЕ приёмы врача + подгружаем patient и slot

```
appointments = (
```

```
    db.query(Appointment)
```

```
    .options(
```

```
        joinedload(Appointment.patient),
```

```
        joinedload(Appointment.slot).joinedload(AppointmentSlot.doctor)
```

```
    )
```

```
.join(AppointmentSlot)
```

```
    .filter(AppointmentSlot.doctor_id == doctor.id)
```

```
    .order_by(AppointmentSlot.start_time.asc())
```

```
    .all()
```

```
)
```

```
return appointments
```

```
# -----
```

```
# ОТМЕНА ПРИЁМА
```

```
# -----
```

```
@router.post("/{appointment_id}/cancel", response_model=AppointmentRead)
```

```
def cancel_appointment(
```

```
    appointment_id: int,
```

```
    data: AppointmentCancelRequest,
```

```
    db: Session = Depends(get_db),
```

```
    user=Depends(get_current_user),
```

```
):
```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.02.001 ПЗ

Лист

118

```

        appointment = AppointmentService.cancel_appointment(db,
appointment_id, data.reason)

        if user.email:
            send_email(
                to_email=user.email,
                subject="Отмена записи",
                text=f"Ваша запись была отменена.\nПричина:
{data.reason or 'не указана'}"
            )

        return appointment

# -----
# ПЕРЕНОС ПРИЁМА
# -----

@router.post("/{appointment_id}/reschedule",
response_model=AppointmentRead)
def reschedule_appointment(
    appointment_id: int,
    data: AppointmentRescheduleRequest,
    db: Session = Depends(get_db),
    user=Depends(get_current_user),
):
    appointment = AppointmentService.reschedule_appointment(
        db,
        appointment_id,
        data.new_slot_id,
        data.complaint,
    )

```

```

if user.email:
    slot = appointment.slot
    doctor = slot.doctor

    send_email(
        to_email=user.email,
        subject="Перенос записи",
        text=(
            f"Ваша запись была перенесена.\n\n"
            f"Новый врач: {doctor.last_name}"
            f"{doctor.first_name}\n"
            f"Дата: {slot.start_time}\n"
        )
    )

    return appointment

```

Файл «auth_routes.py»

```

# app/api/auth_routes.py
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session

from app.db import get_db
from app.schemas.auth import LoginRequest, RegisterRequest, Token,
ChangePasswordRequest
from app.services.auth_service import AuthService
from .deps import get_current_user
from app.models import User

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.02.001 ПЗ

Лист

120

```

router = APIRouter(prefix="/auth", tags=["auth"])

@router.post("/login", response_model=Token)
def login(data: LoginRequest, db: Session = Depends(get_db)):
    return AuthService.authenticate(db, data)

@router.post("/register", response_model=Token)
def register(data: RegisterRequest, db: Session = Depends(get_db)):
    return AuthService.register(db, data)

@router.post("/change-password", response_model=Token)
def change_password(
    data: ChangePasswordRequest,
    db: Session = Depends(get_db),
    user: User = Depends(get_current_user)
):
    return AuthService.change_password(db, user, data)

```

Файл «deps.py»

```

from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
from sqlalchemy.orm import Session

from ..db import get_db
from ..security.jwt import decode_access_token
from ..dao.user_dao import UserDAO
from ..models import User

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/auth/login")

```

Приложение В

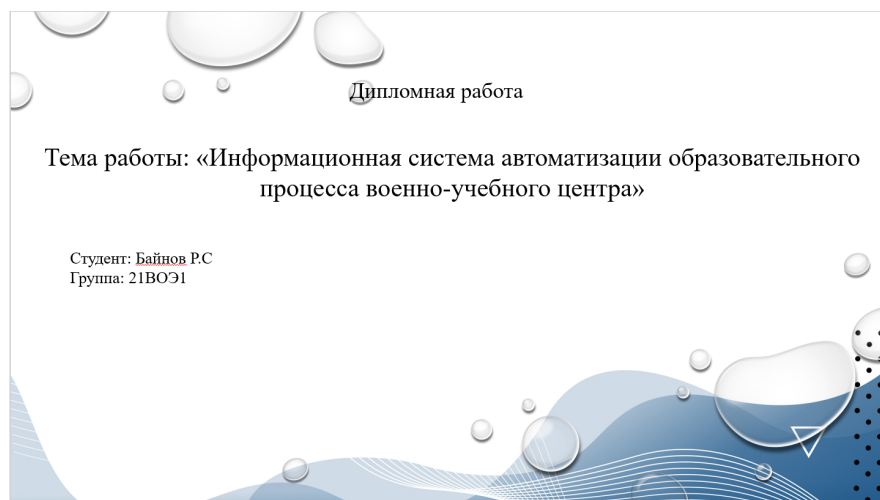
(обязательное)

Информационная система автоматизации образовательного процесса военно-
учебного центра

Презентация

					ПГУ 1.090501.02.001 ПЗ	Лист
						122
Изм.	Лист	№ докум.	Подпись	Дата		

Слайд 1:



Слайд 2:

Актуальность и проблема

Проблема:
Традиционные методы (бумажные журналы, Excel) неэффективны.

Существующие системы (Moodle, «Дневник.ру») не учитывают специфику ВУЦ:

- Нет учёта личного состава.
- Нет дисциплинарных взысканий и приказов.
- Не поддерживается военная отчётность.

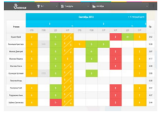
Цель работы: Разработать веб-ориентированную ИС для автоматизации образовательного процесса ВУЦ.

Актуальность:

- Рост объёма данных в военно-учебных центрах (ВУЦ).
- Необходимость строгого контроля посещаемости, успеваемости и дисциплины.
- Требования оперативного формирования отчётности.

Слайд 3:

АНАЛИЗ АНАЛОГОВ



MOODLE:

- +РАЗВИТАЯ ИНФРАСТРУКТУРА, ДИСТАНЦИОННОЕ ОБУЧЕНИЕ
- НЕ ПОДДЕРЖИВАЕТ УЧЁТ ПРИКАЗОВ, ЛИЧНОГО СОСТАВА, ДИСЦИПЛИНЫ

ДНЕВНИК.РУ:

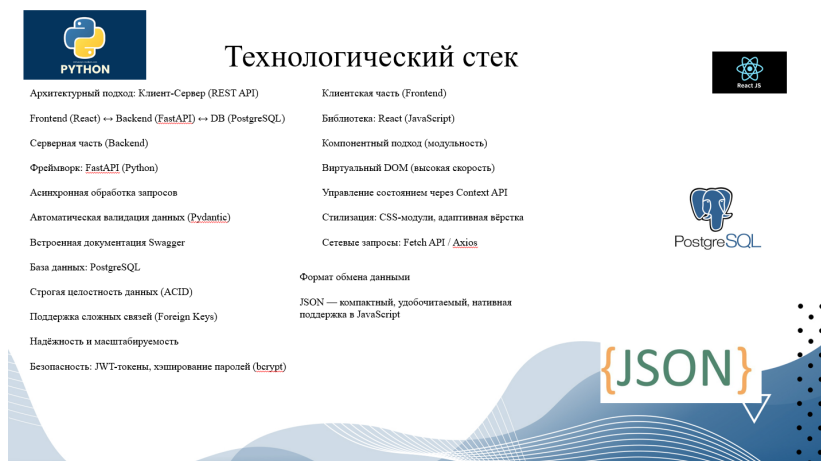
- +ПРОСТОТА, ДОСТУПНОСТЬ
- ОРИЕНТАЦИЯ НА ШКОЛЫ, НЕТ ВОЕННОЙ СПЕЦИФИКИ

СЕТЕВОЙ ГОРОД:

- + КОМПЛЕКСНОСТЬ УЧЁТА УСПЕВАЕМОСТИ
- ОТСУТСТВИЕ ИНСТРУМЕНТОВ ВОЕННОЙ ПОДГОТОВКИ



Слайд 4:



Слайд 5:



Слайд 6:

Проектирование базы данных (ER-диаграмма)

Ключевые сущности:

Users (Расширенная модель: роль, хэш пароля)

Groups (Учебные группы + командир)

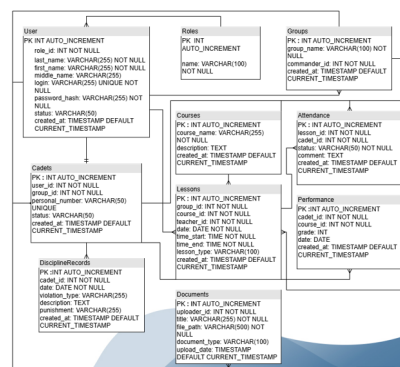
Courses (Дисциплины)

Schedule Items (Расписание: дата, время, связь с группой/преподавателем)

Attendance (Посещаемость: связь студент + занятие)

Discipline (Журнал вызваний)

Особенности: Каскадное удаление (ON DELETE CASCADE), уникальные ключи, транзакционная целостность.



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.02.001 ПЗ

Лист

124

Слайд 7:

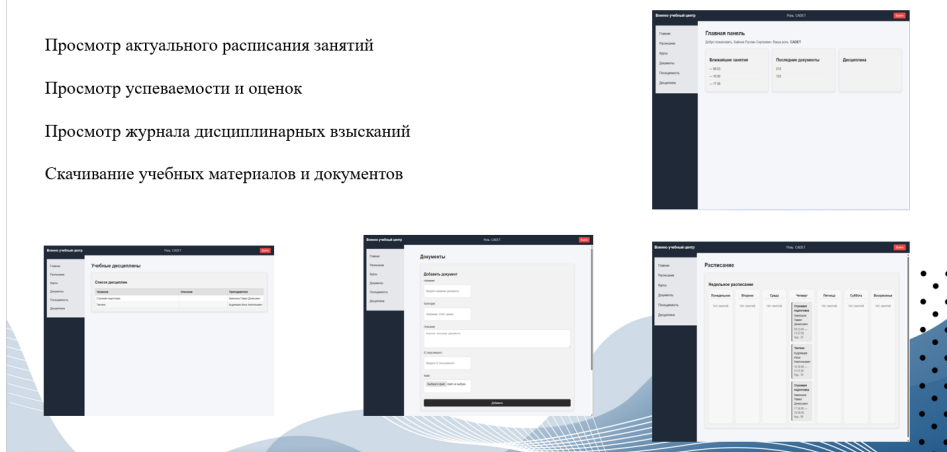
Функционал пользователя(Роль: Курсант)

Просмотр актуального расписания занятий

Просмотр успеваемости и оценок

Просмотр журнала дисциплинарных взысканий

Скачивание учебных материалов и документов



Слайд 8:

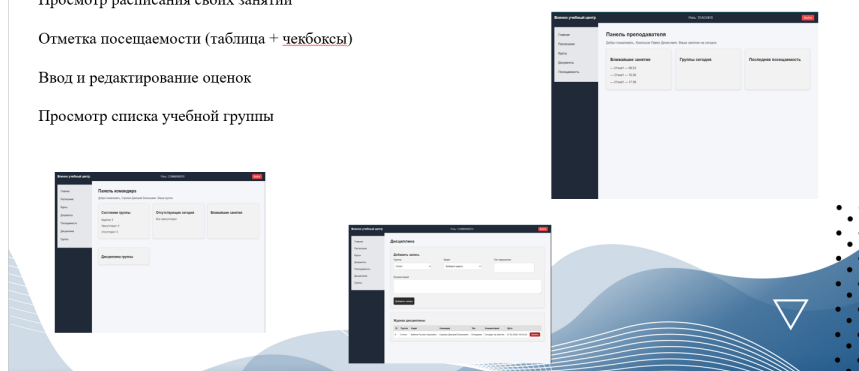
Функционал пользователя(Роль: Преподаватель)

Просмотр расписания своих занятий

Отметка посещаемости (таблица + чекбоксы)

Ввод и редактирование оценок

Просмотр списка учебной группы



Слайд 9:

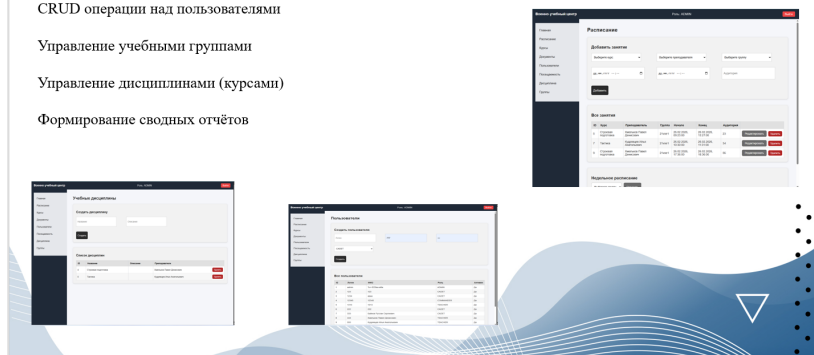
Функционал пользователя(Роль: Администратор)

CRUD операции над пользователями

Управление учебными группами

Управление дисциплинами (курсами)

Формирование сводных отчетов



Слайд 10:

Реализация серверной части (FastAPI)

Разработка серверной части выполнена на основе фреймворка FastAPI, что позволило реализовать высокопроизводительный и структурированный API. В процессе разработки были созданы модули для обработки запросов, управления пользователями, работы с расписанием, посещаемостью, дисциплиной и документами.

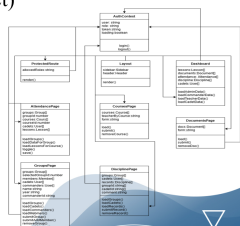
Структура модулей:

- `auth` — регистрация и JWT-аутентификация
- `users` — управление пользователями
- `groups` — управление учебными группами
- `courses` — управление дисциплинами
- `schedule` — работа с расписанием
- `attendance` — учёт посещаемости
- `discipline` — учёт взысканий

Слайд 11:

Реализация клиентской части (React)

Клиентская часть реализована с использованием React, что позволило создать динамичный и отзывчивый интерфейс. Компонентная структура обеспечивает модульность и повторное использование элементов. В процессе разработки были созданы страницы для всех ролей пользователей, а также реализованы механизмы взаимодействия с сервером через REST-API.



Слайд 12:

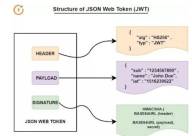
Обеспечение безопасности

Аутентификация: JWT токены (токен хранится в localStorage, срок действия ограничен)

Авторизация: Рольевая модель (CADET, TEACHER, COMMANDER, ADMIN). Middleware проверяет права.

Защита каналов: HTTPS шифрование

Защита данных: Пароли хешируются (bcrypt)



Слайд 13:

Заключение

Итог работы: Разработана готовая к внедрению информационная система для автоматизации образовательного процесса военно-учебного центра.

Решённые проблемы:

- Централизованное хранение данных о личном составе
- Автоматизация учёта посещаемости и успеваемости
- Прозрачность дисциплинарной практики
- Ускорение формирования отчётности

Перспективы развития:

- Интеграция с внешними информационными системами
- Мобильное приложение (React Native)
- Расширенная аналитика и дашборды
- Система уведомлений (Email, Telegram)

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.02.001 ПЗ

Лист

126

