

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего
образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, системы, комплексы и
сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

**КЛИЕНТ-СЕРВЕРНАЯ ИНФОРМАЦИОННАЯ
СИСТЕМА ДЛЯ УЧЁТА ОБОРУДОВАНИЯ**

Студент Ивлюшин Максим Дмитриевич
(подпись, дата) (ФИО полностью)

Руководитель Слепцов Н.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр Бычков А.С.
(подпись, дата) (фамилия, инициалы)

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

Заведующий кафедрой Митрохин М.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой удовлетворительно (протокол заседания ГЭК от 15.06.2026 № 9)

Секретарь ГЭК Гурин Е.И.
(подпись) (фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«7» апреля 2026 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Ивлюшин Максим Дмитриевич гр.22ВВВ1 факультета вычислительной техники

2. Тема работы: «Клиент-серверная информационная система для учета оборудования»

Тема утверждена приказом ПГУ № 347/о от «7» апреля 2026 г.

3. Руководитель бакалаврской работы д.т.н., доцент кафедры ВТ ПГУ Слепцов Н.В.

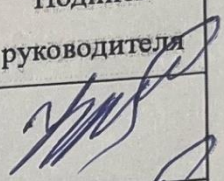
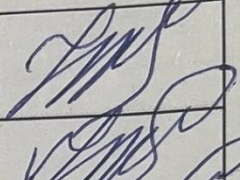
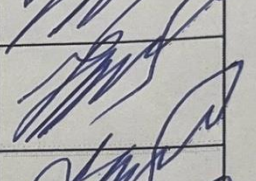
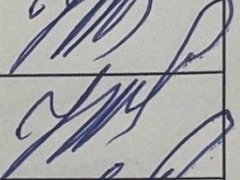
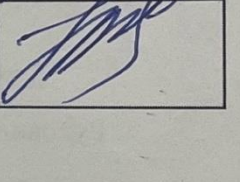
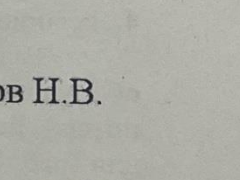
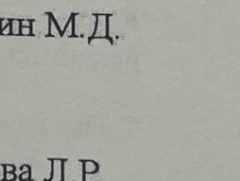
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать веб-приложение «Клиент-серверная информационная система для учета оборудования». Приложение должно обладать графическим и интуитивно понятным интерфейсом для взаимодействия с пользователем. Система должна предоставлять возможность для работы с информацией о пользователях, ролях, и поддерживать аутентификацию. Разрабатываемое приложение должно выполнять задачи по созданию, удалению, редактированию и работы со списком оборудования.

5. Перечень подлежащих разработке вопросов

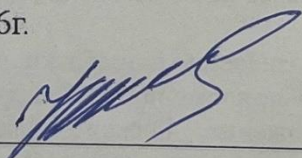
Требуется произвести анализ предметной области систем для учёта оборудования: изучить существующие решения, выделить их достоинства и недостатки. Выбрать язык программирования, серверный фреймворк и технологию для создания графического интерфейса. Определить архитектуру приложения и вспомогательные средства разработки. Изучить функционал библиотек для работы со списками, для реализации аутентификации и для взаимодействия с базой данных.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	30	07.04.2026 – 10.04.2026	
2	Анализ требований к платформе	24	10.04.2026 – 13.04.2026	
3	Выбор технологий, инструментов и проектирование архитектуры	8	13.04.2026 – 14.04.2026	
4	Создание макета графического интерфейса	8	14.04.2026 – 20.04.2026	
5	Разработка серверной логики	60	20.04.2026 – 07.05.2026	
6	Реализация клиентского интерфейса	55	07.05.2026 – 14.05.2026	
7	Оформление пояснительной записки	20	14.05.2026 – 21.05.2026	

Дата выдачи задания «07» апреля 2026г.

Руководитель бакалаврской работы
(подпись, дата)

 Слепцов Н.В.

Задание к исполнению принял студент
(подпись, дата)

 Ивлюшин М.Д.

Работу к защите допустить
Декан факультета

 10.06.2026

(подпись, дата)

Фионова Л.Р.

Реферат

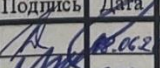
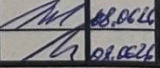
Бакалаврская работа содержит 122 листа, 37 рисунков, 1 таблицу, 6 источников, 3 приложения.

КЛИЕНТ-СЕРВЕРНАЯ СИСТЕМА, УЧЁТ ОБОРУДОВАНИЯ, АВТОМАТИЗИРОВАННАЯ СИСТЕМА, БАЗА ДАННЫХ, PYTHON, FASTAPI, REACT

Цель работы – разработка клиент-серверной информационной системы для учёта оборудования, обеспечивающей централизованное хранение данных об оборудовании, многопользовательский доступ с разграничением прав, учёт перемещений и формирование отчётов.

Для достижения поставленной цели спроектирована и реализована информационная система на основе трёхуровневой архитектуры: клиентское веб-приложение, сервер приложений и сервер базы данных. В процессе разработки реализованы основные функции учёта оборудования: добавление, редактирование, списание и удаление единиц оборудования, выдача оборудования сотрудникам и его возврат, подача и обработка заявок на выдачу, управление пользователями и ролями, поиск и фильтрация оборудования по различным критериям, а также формирование трёх типов отчётов в формате Excel. Для обеспечения безопасности реализована аутентификация с хешированием паролей, а также ролевая модель доступа.

В процессе разработки использовались язык программирования Python, язык JavaScript, система управления базами данных PostgreSQL, инструмент сборки Vite, библиотеки SQLAlchemy, Pydantic, axios, Material UI. Для администрирования базы данных применялся pgAdmin 4, среда разработки – Visual Studio Code. Разработанная система протестирована на корректность выполнения всех функциональных сценариев, подтверждена работоспособность механизмов учёта оборудования, выдачи, заявок и формирования отчётов.

					ПГУ 1.090301.03.001 ПЗ		
Изм.	Лист	№ докум	Подпись	Дата			
Разраб		Ивлошин М.Д.		28.06.22			
Пров		Слепцов Н.В.		28.06.22			
Н. Контр.		Бычков А.С.		28.06.22			
Утв		Митрохин М.А.		28.06.22			
					Литера	Лист	Листов
						2	122
					ФВТ, каф. ВТ гр. 22BBB1		

Клиент-серверная информационная система для учета оборудования
Пояснительная записка

Содержание

Перечень сокращений и обозначений	4
Введение.....	5
1 Клиент-серверная информационная система для учета оборудования	6
1.1 Анализ предметной области	6
1.2 Анализ аналогов информационных систем	8
1.3 Постановка задачи	11
2 Выбор технологий и средств разработки приложения.....	15
2.1 Выбор языка программирования.....	16
2.2 Выбор СУБД.....	18
2.3 Графический клиент pgAdmin 4	19
2.4 Выбор среды разработки	20
3 Реализация	23
3.1 Архитектура приложения.....	23
3.2 Работа с базой данных	24
3.3 Разработка клиентского приложения	27
3.3.1 Основные компоненты и их назначение.....	28
3.4 Разработка серверного приложения	34
3.4.2 Обработка сетевых соединений и протокол обмена.	36
3.4.3 Работа с базой данных.....	37
3.4.4 Обработка действий (бизнес-логика).	39
3.5 Тестирование	40
4 Безопасность жизнедеятельности	48
4.2 Организация рабочего места пользователя	48
4.3 Режим труда и отдыха при работе с информационной системой	49
4.4 Электробезопасность и требования к электрооборудованию	49
4.5 Пожарная безопасность.....	50
Заключение	51
Список использованных источников	53
Приложение А	54
Приложение Б.....	59
Приложение В.....	114

Перечень сокращений и обозначений

В настоящей работе применяют следующие обозначения и сокращения

БД - база данных

ОС - операционная система

ПК - персональный компьютер

СУБД - система управления базами данных

API (Application Programming Interface) - набор правил и интерфейсов, позволяющий различным программным компонентам обмениваться данными и взаимодействовать друг с другом

URL (Uniform Resource Locator) - это уникальный адрес, который указывает на расположение ресурса (сайта, файла, изображения) в интернете

CSS (Cascading Style Sheets) - язык описания внешнего вида веб-страниц, отвечающий за оформление и расположение элементов

HTML (HyperText Markup Language) - язык разметки, применяемый для структурирования содержимого веб-страниц

IDE (Integrated development environment) - интегрированная среда разработки

JS (JavaScript) - язык программирования, используемый преимущественно для создания интерактивных элементов веб-страниц

VS Code (Virtual Studio Code) - бесплатный редактор исходного кода от компании Microsoft, созданный для Windows, macOS и Linux

SQL (Structured Query Language) - язык структурированных запросов

UML (Unified Modeling Language) - унифицированный язык моделирования

CSV (Comma-Separated Values) - это текстовый формат для представления табличных данных

PDF (Portable Document Format) - это популярный универсальный формат электронных документов

HTTP (HyperText Transfer Protocol) — это базовый протокол передачи данных в интернете

					ПГУ 1.090301.03.001 ПЗ	Лист
						4
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

Современное состояние организаций всех форм собственности характеризуется активным внедрением цифровых технологий в процессы управления. Одним из ключевых направлений автоматизации является учёт материальных ценностей, в частности – оборудования (компьютерной техники, оргтехники, серверов, периферийных устройств). Эффективный учёт оборудования позволяет своевременно отслеживать его местоположение, состояние, перемещения между подразделениями и сотрудниками, а также планировать закупки и списание. Однако на практике многие малые и средние предприятия, образовательные учреждения продолжают использовать ручные методы (Excel, бумажные журналы), которые не обеспечивают многопользовательский доступ, контроль целостности данных и оперативное формирование отчётов. Существующие специализированные системы (1С:Управление IT-отделом, GLPI, Snipe-IT) либо избыточно сложны и дороги, либо требуют значительных ресурсов на внедрение и сопровождение. Таким образом, актуальность дипломной работы обусловлена необходимостью создания легковесной, доступной и функциональной клиент-серверной информационной системы учёта оборудования, которая сможет закрыть потребности небольших организаций без существенных финансовых и временных затрат.

Целью дипломной работы является разработка и программная реализация клиент-серверной информационной системы для учёта оборудования, обеспечивающей многопользовательский доступ, разграничение прав, ведение истории перемещений и генерацию отчётов.

1 Клиент-серверная информационная система для учета оборудования

1.1 Анализ предметной области

Предметная область «учёт оборудования» охватывает процессы жизненного цикла технических средств организации - от момента поступления до списания. В любой компании, имеющей парк компьютеров, оргтехники, серверов или производственного оборудования, возникает необходимость фиксировать, где находится каждый инвентарный объект, в каком он состоянии, кто им пользуется, когда проводилось обслуживание.

На практике учёт часто ведётся с помощью инструментов, не предназначенных для этой цели. Самый распространённый способ - таблицы Microsoft Excel или Google Sheets. Сотрудник создаёт файл, вносит колонки: инвентарный номер, название, дата покупки, цена, местонахождение, примечания. Файл хранится на сетевом диске или пересылается по электронной почте. При таком подходе возникают проблемы:

- конфликт версий - два сотрудника одновременно открывают файл, сохраняют свои изменения, и данные одного из них теряются.

- отсутствие аудита - невозможно узнать, кто и когда изменил запись, что критично при инвентаризации.

- сложность поиска - чтобы найти все ноутбуки дороже 50 тыс. руб., нужно вручную настраивать фильтр каждый раз.

- нет истории перемещений - если принтер переставили из кабинета 101 в 203, старая запись просто затирается, и след теряется.

Более продвинутые организации используют специализированные системы - 1С:Управление ИТ-отделом, GLPI, Snipe-IT. Однако они часто избыточны: требуют установки и настройки веб-сервера, поддержки квалифицированным администратором, обучения персонала. Для небольшой образовательной организации или отдела ИТ из трёх человек такие системы слишком комплексные и сложные.

Также важно не забывать про лицензионное программное обеспечение, а также периферию (клавиатуры, мыши, внешние диски). Хорошая система учёта должна позволять добавлять любые типы активов и гибко настраивать их атрибуты.

Актуальность создания клиент-серверной системы учёта оборудования продиктована несколькими современными трендами и практическими потребностями.

Во-первых, повсеместная цифровая трансформация. Организации всех масштабов стремятся заменить бумажные журналы и разрозненные Excel-файлы на автоматизированные системы. Это позволяет не только ускорить работу, но и сделать бизнес-процессы прозрачными для руководства и контролирующих органов. Учёт оборудования - одна из ключевых областей, где контроль даёт быстрый экономический эффект: снижаются потери от неправильного списания, упрощается инвентаризация, исключаются простои.

Во-вторых, рост стоимости и количества оборудования. Современное предприятие может иметь десятки и сотни единиц техники: от серверов стоимостью миллионы рублей до клавиатур и мышей. Без систематизированного учёта невозможно эффективно планировать закупки, амортизацию, техническое обслуживание. Например, если не отслеживать даты покупки, легко пропустить окончание гарантийного срока. Если не фиксировать перемещения, то при увольнении сотрудника его ноутбук может «потеряться».

В-третьих, требования законодательства и внутреннего аудита. При налоговых проверках часто требуют подтвердить наличие основных средств. Акты инвентаризации должны быть подкреплены данными из системы учёта. Кроме того, для государственных учреждений (школ, больниц, вузов) отчётность об использовании бюджетного имущества строго регламентирована - и ручное заполнение форм чревато ошибками.

В-четвёртых, переход на удалённый и гибридный режим работы. Сотрудники могут работать из дома, из филиалов, из командировок. Система учёта должна быть доступна через браузер с любого устройства, без установки

					ПГУ 1.090301.03.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

дополнительного программного обеспечения. Выбранная веб-архитектура (React + FastAPI) удовлетворяет этому требованию полностью: достаточно открыть URL, ввести логин и пароль.

Наконец, анализ рынка готовых решений показывает, что между примитивными таблицами и дорогими корпоративными системами существует «ниша» - легковесные, недорогие (или бесплатные) веб-приложения, которые можно развернуть за один день и которые не требуют выделенного администратора. Разрабатываемая система как раз заполняет эту нишу.

Таким образом, работа становится актуальной: она отвечает на реальный запрос множества небольших организаций - школ, малых предприятий, отделов IT, - которые хотят навести порядок в учёте оборудования без значительных затрат.

1.2 Анализ аналогов информационных систем

Для обоснования необходимости разработки новой системы был проведён анализ существующих решений, используемых для учёта оборудования. Каждый аналог рассмотрен по критериям: функциональность, стоимость, сложность внедрения, безопасность, многопользовательский режим.

1С:Управление IT-отделом (1С:ITIL) - профессиональное решение на платформе 1С:Предприятие. Включает учёт активов, управление заявками, SLA-мониторинг, интеграцию с Active Directory см. рисунок 1.

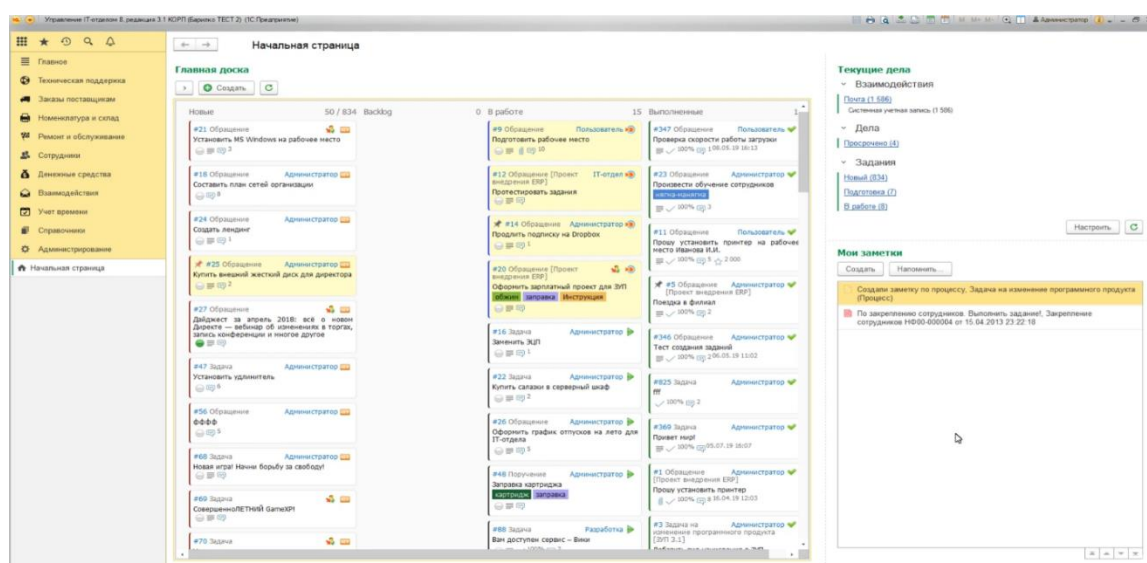


Рисунок 1 – Обзор аналога «1С:Управление IT-отделом (1С:ITIL)»

Достоинства: мощная функциональность, соответствие стандартам ITIL, поддержка крупных предприятий.

Недостатки: высокая стоимость лицензий и сопровождения (десятки и сотни тысяч рублей); сложность настройки; требует квалифицированного 1С-программиста; избыточно для учёта 50–100 единиц оборудования.

Вывод: целесообразна только для крупных организаций с выделенным IT-бюджетом.

GLPI - популярная веб-система для управления IT-активами и технической поддержки. Написана на PHP, использует MySQL см. рисунок 2.

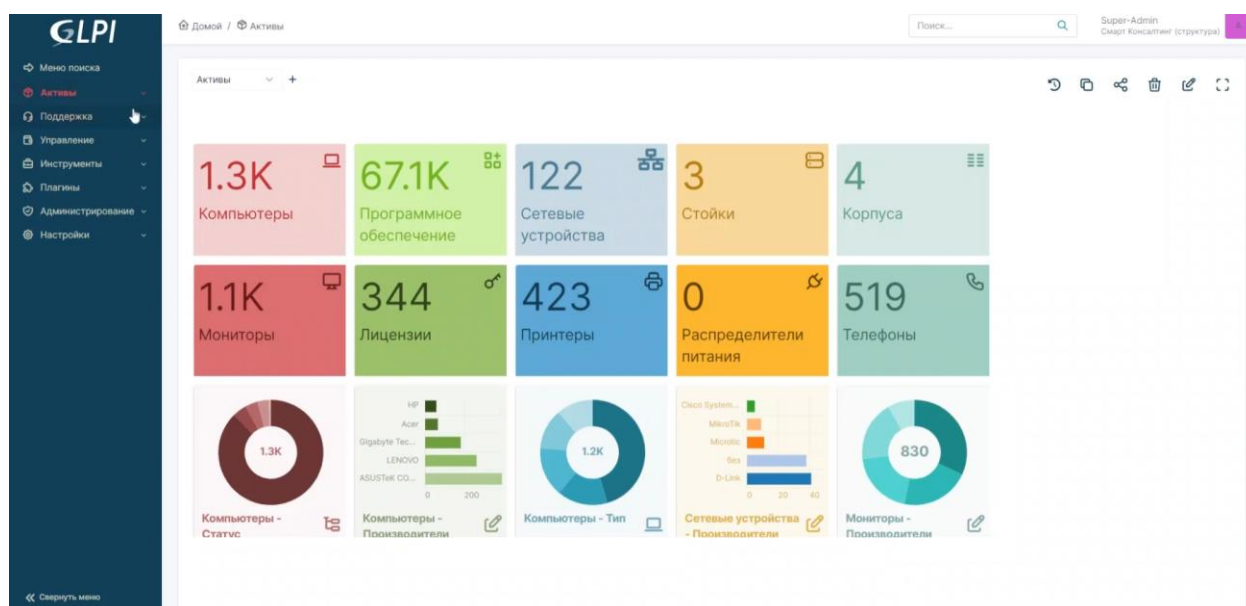


Рисунок 2 – Обзор аналога «GLPI»

Достоинства: бесплатность, богатые возможности (инвентаризация, заявки, база знаний, контракты, финансовый учёт).

Недостатки: сложность установки и настройки (требуется веб-сервер, PHP, настройка прав); интерфейс перегружен, что отпугивает обычных пользователей; локализация на русском языке неполная.

Вывод: для простого учёта оборудования «из коробки» неудобен, требует доработок и обучения.

Snipe-IT - Специализированная система учёта активов, также написанная на PHP (Laravel). Позиционируется как «простой и элегантный способ отслеживать оборудование» см. рисунок 3.

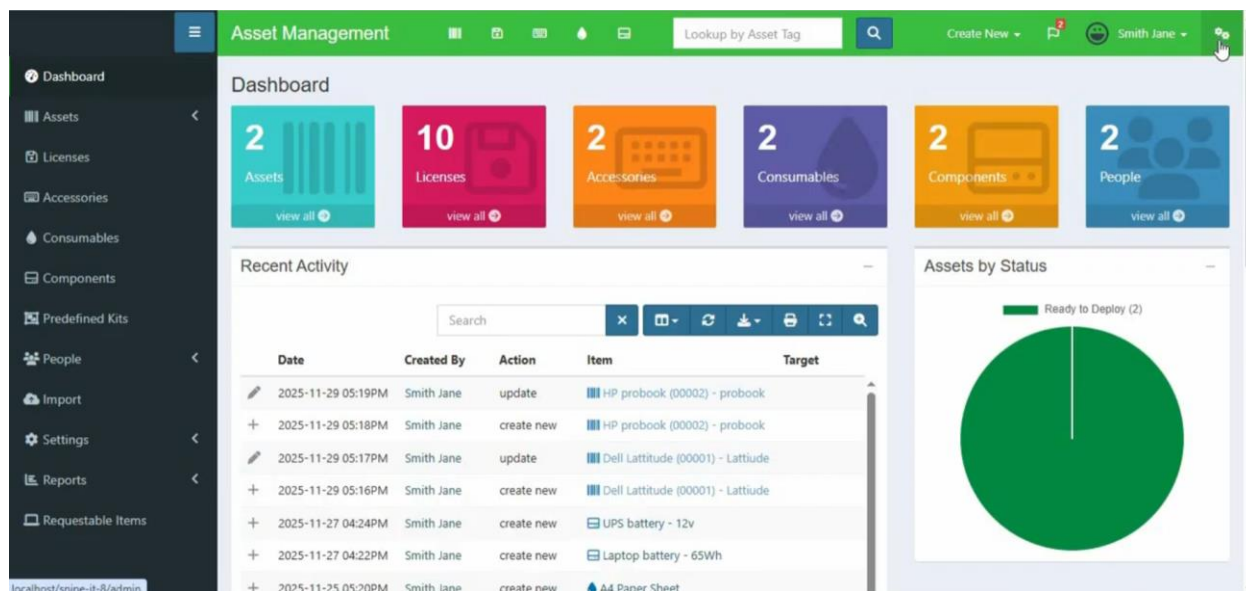


Рисунок 3 – Обзор аналога «Snipe-IT»

Достоинства: красивый современный интерфейс, поддержка штрихкодов, импорт из Excel, API для интеграций.

Недостатки: требует хостинга с PHP 8+ и MySQL; английский интерфейс (русский перевод неофициальный и неполный); для многих организаций функционал избыточен (например, учёт лицензий, контрактов, заявок).

Вывод: хороший вариант, но не всегда подходит по языку и простоте развёртывания.

Таблица 1 – Сравнение аналогов

Критерий	1C:ITIL	GLPI	Snipe-IT
Лицензия	Коммерческая	Open Source	Open Source
Платформа	1C:Предприятие	PHP + MySQL	PHP + MySQL (Laravel)
Основное назначение	Управление ИТ-процессами (ITIL)	Управление активами	Учёт активов
Заявки	Поддерживает	Поддерживает	Нет
Автоинвентаризация	Ограниченно (через сторонние решения)	Агенты FusionInventory/OC S	Нет

Критерий	1С:ITIL	GLPI	Snipe-IT
Управление изменениями	Есть	Есть	Нет
Русский интерфейс	Полный (официальный)	Частичный (неофициальный)	Полный (официальный)
Сложность внедрения	Высокая (требуется 1С-специалист)	Средняя (требуется веб-администратор)	Низкая (Docker / инструкция)
Стоимость	Высокая (лицензия + сопровождение)	Бесплатно (кроме хостинга)	Бесплатно (кроме хостинга)
Подходит для малых организаций	Нет	Условно (избыточно)	Да

Сравнительная характеристика:

Ни один из аналогов не сочетает в себе одновременно бесплатность, веб-интерфейс, простоту установки, поддержку русского языка и современный стек (React + Python). Разрабатываемая система призвана заполнить этот пробел: она будет легко разворачиваться (достаточно установить Python, PostgreSQL и запустить два сервиса), иметь интуитивный интерфейс на русском языке, а при необходимости - дорабатываться силами штатного программиста благодаря открытому исходному коду.

1.3 Постановка задачи

На основе анализа предметной области и существующих аналогов сформулирована следующая задача: разработать клиент-серверную информационную систему для учёта оборудования на основе технологий React, FastAPI и PostgreSQL, которая обеспечит централизованное хранение данных, многопользовательский доступ с разграничением ролей, учёт перемещений и формирование отчётов.

Функциональные требования:

- добавление новой единицы с обязательными полями: название, инвентарный номер, тип (компьютер, принтер и т.п.). Необязательные поля: серийный номер, дата покупки, цена, местоположение, статус (в наличии / выдан / в ремонте / списан);

- редактирование любого поля существующей записи;

- удаление записи (списание) - фактически изменение статуса на «списан», а не физическое удаление из БД, чтобы сохранить историю;

- поиск и фильтрация по всем полям (например, показать все принтеры НР дороже 30 000 руб., находящиеся в кабинете 405);

- регистрация факта выдачи оборудования сотруднику: выбор из списка оборудования, выбор из списка сотрудников (пользователей системы), дата выдачи, примечание;

- регистрация возврата: указывается дата возврата, статус меняется обратно на «в наличии»;

- просмотр истории перемещений для каждой единицы оборудования (кто, когда взял, вернул ли);

- управление пользователями и правами;

- аутентификация по паре логин/пароль. Пароли хранятся в хешированном виде (bcrypt);

- две роли: «администратор» (может добавлять/редактировать/удалять оборудование, управлять пользователями, просматривать отчёты) и «пользователь» (только просмотр списка оборудования, подача заявки на выдачу - если реализовано, или только чтение);

- администратор может создавать новых пользователей, блокировать/разблокировать их.

На рисунке 4 приведена диаграмма вариантов использования

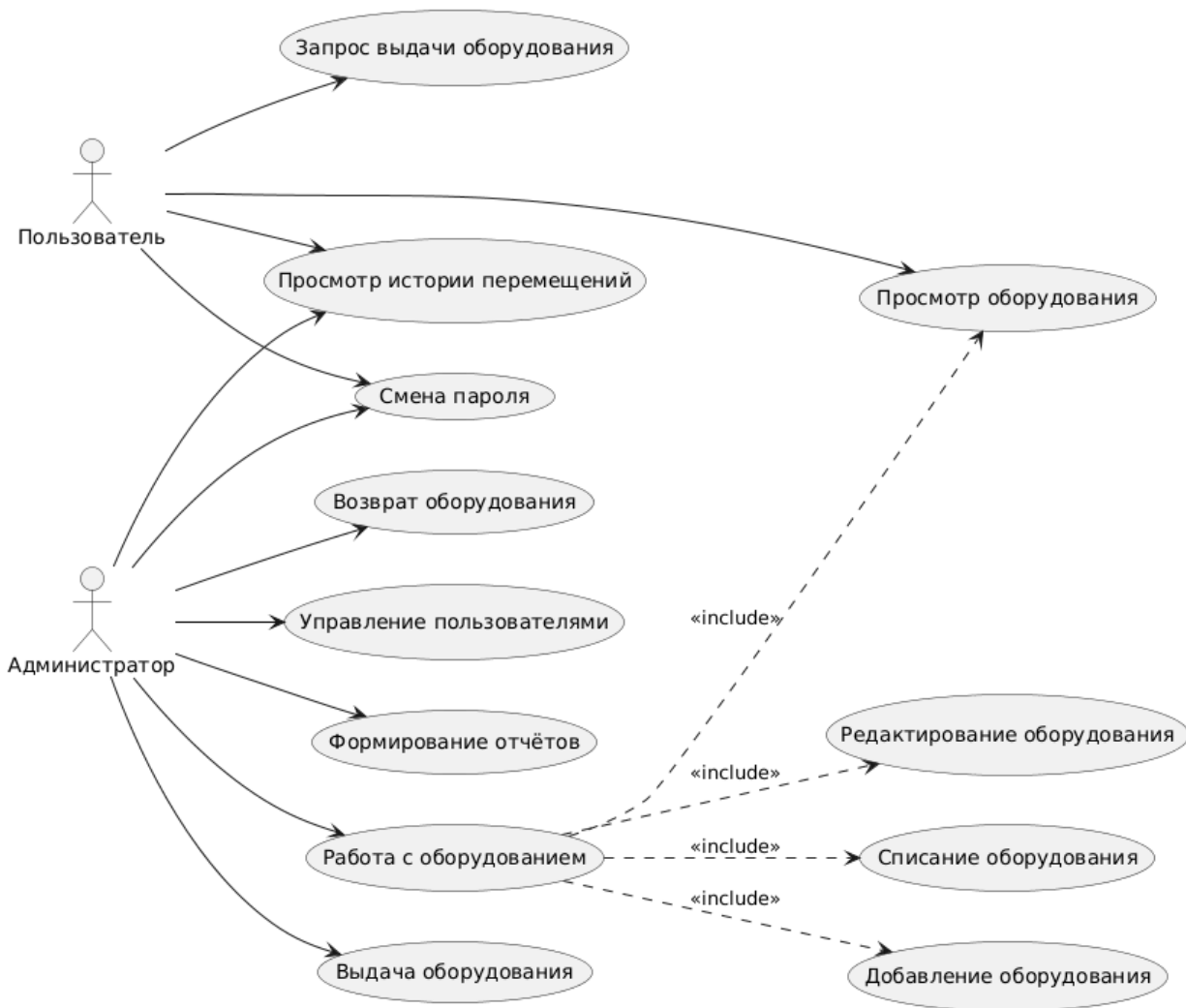


Рисунок 4 – Диаграмма вариантов использования

Формирование отчётов:

- отчёт «Состояние оборудования»: таблица с фильтрами по типу, статусу, местоположению, диапазону цены. Экспорт в форматы CSV или PDF;
- отчёт «Выданное оборудование»: на текущую дату или на произвольную дату – список единиц, которые числятся за сотрудниками;
- отчёт «Списанное оборудование за период»: по месяцам или годам.

Нефункциональные требования:

- производительность: ответ сервера на типовой запрос (получение списка из 100 записей) - не более 500 мс;
- совместимость: Google Chrome – версия 51 и выше, Yandex Browser – версия 22.5 и выше, Mozilla Firefox – версия 53 и выше, Microsoft Edge – версия 79 и выше.

Ограничения и допущения:

- система не взаимодействует с бухгалтерским учётом (проводки, амортизация) – это отдельная задача;
- первоначальное наполнение базы данных (типовое оборудование, тестовые пользователи) создаётся вручную.

					ПГУ 1.090301.03.001 ПЗ	Лист
						14
Изм.	Лист	№ докум.	Подпись	Дата		

2 Выбор технологий и средств разработки приложения

Для автоматизации учёта оборудования необходимо выбрать архитектуру, которая обеспечит централизованное хранение данных, одновременную работу нескольких пользователей, безопасность и удобство. Анализ показывает, что наилучшим образом этим требованиям соответствует клиент-серверная архитектура.

В классической двухуровневой модели (клиент ↔ БД) клиентское приложение напрямую подключается к базе данных. Это просто, но небезопасно: учётные записи БД приходится раздавать всем пользователям, а бизнес-логика (например, проверка того, что инвентарный номер уникален) дублируется в каждом клиенте. Поэтому лучшим решением будет трёхуровневая архитектура: клиент (React) → сервер приложений (FastAPI) → сервер базы данных (PostgreSQL).

Ниже приведен список выбранных технологий и средств разработки:

- Python (FastAPI);
- JavaScript (React);
- PostgreSQL.

На рисунке 5 представлена диаграмма развёртывания приложения.

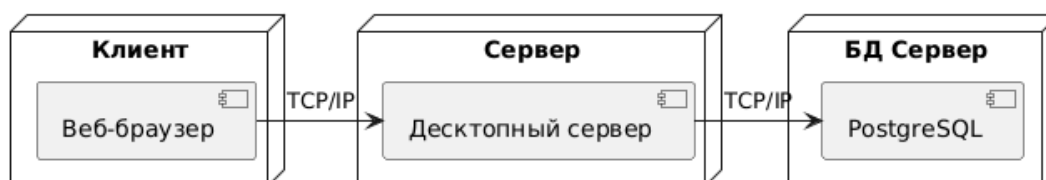


Рисунок 5 – Диаграмма развёртывания приложения

На рисунке 6 представлена диаграмма архитектуры приложения.

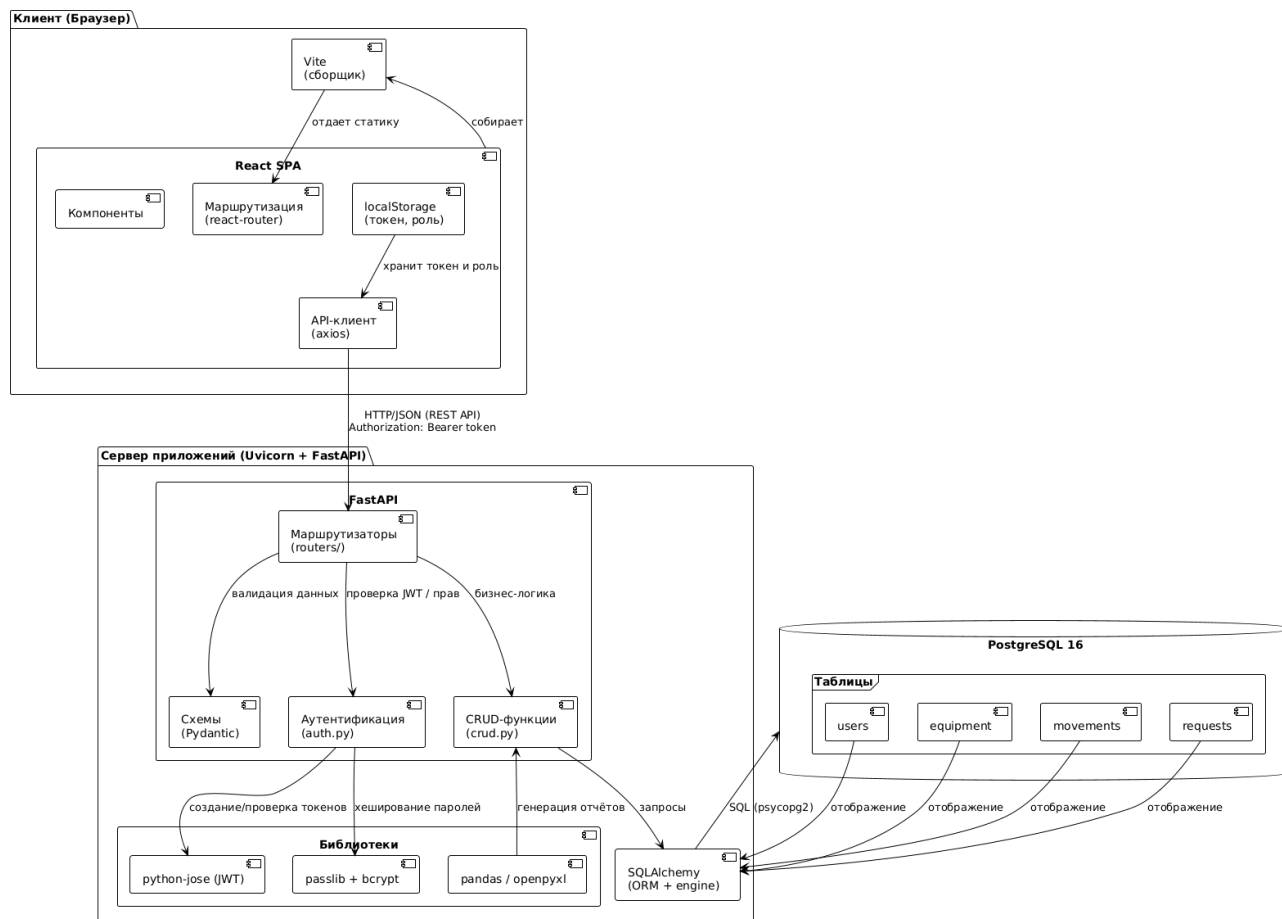


Рисунок 6 – Архитектура приложения

2.1 Выбор языка программирования

Для начала работы с платформой необходимо» пройти процесс регистрации. Разрабатываемая система имеет клиент-серверную архитектуру с веб-интерфейсом, поэтому выбор языков программирования производился отдельно для серверной и клиентской частей.

Серверная часть - для реализации сервера приложений был выбран язык Python версии 3.10 и выше [1]. FastAPI (Python) - современный фреймворк для создания REST API. Его ключевые особенности: асинхронность, автоматическая генерация документации OpenAPI (Swagger UI), встроенная валидация данных через Pydantic, простота написания кода. Выбор обоснован следующими факторами:

- быстрота разработки - Python обладает лаконичным синтаксисом и обширной стандартной библиотекой, что позволяет сократить время написания кода по сравнению с языками вроде Java или C++;

- богатая экосистема веб-фреймворков - наличие FastAPI, Django, Flask даёт возможность выбрать оптимальный инструмент для создания REST API. FastAPI, в частности, обеспечивает высокую производительность (сопоставимую с Node.js и Go) благодаря асинхронной обработке запросов;

- поддержка асинхронности - встроенные механизмы `async/await` позволяют эффективно обрабатывать множество параллельных подключений, что важно для многопользовательской системы;

- простота взаимодействия с PostgreSQL - через драйвер `asyncpg` или ORM `SQLAlchemy`, что ускоряет разработку и снижает риск ошибок при формировании SQL-запросов;

- кроссплатформенность - код на Python одинаково работает в Windows, Linux и macOS, что упрощает развёртывание и тестирование;

- большое сообщество и документация - наличие готовых решений для JWT-аутентификации, валидации данных (`Pydantic`), CORS и других типовых задач.

Альтернативы, такие как Node.js (JavaScript), C# (ASP.NET Core) или Java (Spring Boot), также могли быть использованы, но Python был выбран как наиболее изученный и оптимальный по соотношению сложности разработки и производительности для проекта.

Клиентская часть - для реализации пользовательского интерфейса выбран язык JavaScript (с использованием библиотеки React) [2]. React - библиотека для построения пользовательских интерфейсов. Выбрана потому, что создаёт одностраничное приложение (SPA): после загрузки все переходы между разделами происходят без перезагрузки страницы, что даёт ощущение нативного приложения. React имеет огромное сообщество, множество готовых компонентов (например, таблицы, модальные окна, формы), что ускоряет разработку. Выбор обусловлен:

- универсальность веб-технологий - приложение работает в браузере без установки дополнительного ПО, доступно с любого устройства;
- декларативная компонентная модель упрощает создание интерактивных интерфейсов, управление состоянием и повторное использование кода;
- высокая производительность - виртуальный DOM React минимизирует прямые манипуляции с реальным DOM, что обеспечивает быстрый отклик даже при большом объёме данных (таблица с оборудованием);
- большое количество готовых компонентов (Material UI, Ant Design) ускоряет разработку и обеспечивает современный внешний вид.

Альтернативы (Vue.js, Angular, Svelte) также способны решить поставленную задачу, однако React выбран из-за его популярности, обширной документации.

Таким образом, для серверной части выбран Python, для клиентской - JavaScript с библиотекой React. Это современный и эффективный стек, полностью соответствующий клиент-серверной архитектуре.

2.2 Выбор СУБД

В качестве системы управления базами данных (СУБД) для проекта выбрана PostgreSQL. PostgreSQL - одна из самых надёжных и функциональных свободных СУБД. Поддерживает сложные запросы (JOIN, подзапросы, оконные функции), транзакции с уровнем изоляции, ролевую модель доступа, а также JSON-поля на случай неструктурированных данных. В отличие от других СУБД, PostgreSQL строже соблюдает стандарты SQL и имеет лучшую репутацию в задачах, где важна целостность данных. Решение принято на основе анализа требований к хранению данных об оборудовании, перемещениях и пользователях.

Обоснование выбора PostgreSQL:

- реляционная модель - данные об оборудовании, пользователях и перемещениях имеют чёткие связи (например, оборудование может быть выдано пользователю), что естественно описывается через внешние ключи и нормализованные таблицы. PostgreSQL является одной из самых мощных реляционных СУБД с открытым исходным кодом;

- поддержка стандартов SQL - PostgreSQL строго следует стандарту ANSI SQL, что обеспечивает переносимость запросов и снижает вероятность ошибок при написании сложных выборок (отчёты, история перемещений);

- надёжность и целостность данных - поддержка транзакций (ACID), внешних ключей с каскадными действиями, ограничений уникальности и проверочных ограничений гарантирует, что в базе не появится, например, двух одинаковых инвентарных номеров или записей о выдаче несуществующего оборудования;

- производительность при большом объёме данных - PostgreSQL эффективно работает с индексами (B-tree, hash, GIN), что позволяет быстро выполнять поиск по инвентарному номеру, фильтрацию по типу и статусу даже при десятках тысяч записей;

- расширяемость - поддержка JSON/JSONB полей даёт возможность хранить дополнительные атрибуты оборудования без изменения схемы. Это может пригодиться при будущем расширении функционала;

- бесплатность и лицензия - PostgreSQL распространяется под лицензией PostgreSQL License (аналогичной BSD), что не накладывает ограничений на использование в коммерческих и учебных проектах.

2.3 Графический клиент pgAdmin 4

Для администрирования базы данных PostgreSQL выбран pgAdmin 4 - официальный графический клиент с открытым исходным кодом.

Основные возможности pgAdmin 4, используемые в проекте:

- визуальное управление схемой - создание таблиц, индексов, внешних ключей без написания SQL-кода (хотя SQL также поддерживается);

- редактор запросов - написание и выполнение SQL-скриптов, просмотр результатов, анализ плана выполнения (EXPLAIN);

- просмотр и редактирование данных - удобное отображение содержимого таблиц, фильтрация, сортировка, вставка/обновление записей через интерфейс;

- мониторинг - отслеживание активных сессий, блокировок, производительности;

- резервное копирование и восстановление - встроенные средства для создания дампов базы данных, что важно для тестирования и защиты дипломного проекта;

- кроссплатформенность - pgAdmin 4 работает как веб-приложение, доступное через браузер, что удобно при разработке под разными операционными системами.

Причины выбора именно pgAdmin 4:

- официальная поддержка - разрабатывается командой PostgreSQL, всегда актуален и совместим с последними версиями СУБД;
- бесплатность - не требует лицензионных отчислений;
- удобство - позволяет наглядно демонстрировать структуру базы данных и результаты запросов.

Альтернативы, такие как DBeaver, DataGrip (платный) или командная строка `psql`, также могли быть использованы, но pgAdmin 4 выбран как наиболее простой и функциональный.

В процессе разработки pgAdmin 4 применяется для:

- первоначального создания базы данных `equipment_db`;
- визуального контроля схемы (таблицы `equipment`, `users`, `movements`);
- выполнения тестовых SQL-запросов при отладке;
- создания резервных копий.

2.4 Выбор среды разработки

Для реализации серверной и клиентской частей проекта выбрана среда разработки Visual Studio Code (VS Code). Дополнительно для работы с базой данных используется pgAdmin 4, а для написания React-компонентов – встроенные средства VS Code.

Visual Studio Code (VS Code) - это бесплатный, легкий и очень популярный редактор исходного кода, разработанный Microsoft для Windows, macOS и Linux. Он поддерживает множество языков программирования, отличается высокой скоростью работы и обширной экосистемой расширений, позволяющей превратить его в мощную среду разработки.

Обоснование выбора VS Code:

- лёгкость и быстроедействие - VS Code потребляет меньше оперативной памяти и быстрее запускается по сравнению с тяжеловесными IDE (PyCharm, WebStorm, Visual Studio);

- мощная экосистема расширений - устанавливаются плагины для Python, JavaScript, React, PostgreSQL, Git, что превращает редактор в полноценную среду разработки;

- Python - расширение Microsoft Python (автодополнение, линтер, отладка, тестирование):

- Pylance - быстрый анализатор кода для Python;

- ES7+ React/Redux/React-Native snippets - ускоряет написание React-компонентов;

- Prettier - автоматическое форматирование кода;

- PostgreSQL - подключение к базе данных прямо из редактора (расширение vscode-postgres);

- встроенный терминал - позволяет запускать сервер FastAPI (uvicorn), сборщик Vite для React, выполнять команды Git и Python-скрипты без переключения окон;

- интеграция с системами контроля версий - встроенный клиент Git упрощает отслеживание изменений в дипломном проекте;

- кроссплатформенность - работает в Windows, Linux, macOS, что важно при возможной смене окружения;

- бесплатность - VS Code распространяется бесплатно под лицензией MIT.

Дополнительные средства разработки:

- Git – система контроля версий, репозиторий размещён на GitHub (или локально);

- Postman – для ручного тестирования API-эндпоинтов (альтернатива – встроенная документация FastAPI /docs);

- Vite – сборщик React-приложения.

					ПГУ 1.090301.03.001 ПЗ	Лист
						21
Изм.	Лист	№ докум.	Подпись	Дата		

Таким образом, выбранный набор инструментов – VS Code, pgAdmin 4, Git, Postman – полностью покрывает потребности разработки, отладки и документирования клиент-серверной системы учёта оборудования.

					ПГУ 1.090301.03.001 ПЗ	Лист
						22
Изм.	Лист	№ докум.	Подпись	Дата		

3 Реализация

3.1 Архитектура приложения

Разрабатываемая информационная система построена по трёхуровневой клиент-серверной архитектуре, обеспечивающей разделение уровней представления, бизнес-логики и хранения данных. Основные компоненты системы:

Клиентский уровень (веб-браузер) - одностраничное приложение (SPA), реализованное на библиотеке React. Отвечает за отображение пользовательского интерфейса, обработку действий пользователя и отправку HTTP-запросов к серверу приложений по REST API.

Серверный уровень - приложение на фреймворке FastAPI. Выполняет бизнес-логику, валидацию данных, аутентификацию и авторизацию, взаимодействует с базой данных через ORM SQLAlchemy и предоставляет REST API для клиента.

Уровень данных (СУБД) - реляционная база данных PostgreSQL . Хранит всю информацию об оборудовании, пользователях, перемещениях и заявках.

Взаимодействие между уровнями организовано следующим образом:

Пользователь в браузере выполняет действия (нажатие кнопок, ввод данных), которые преобразуются в асинхронные HTTP-запросы (GET, POST, PUT, DELETE) к серверу приложений.

Сервер приложений принимает запрос, проверяет JWT-токен доступа (если требуется аутентификация), валидирует входящие данные с помощью Pydantic-схем, выполняет необходимые запросы к базе данных через SQLAlchemy и возвращает результат в формате JSON.

Клиент получает ответ и обновляет состояние интерфейса в соответствии с полученными данными.

Для обеспечения безопасности все пароли пользователей хранятся в хешированном виде (алгоритм bcrypt). Доступ к защищённым эндпоинтам API возможен только при предъявлении действительного JWT-токена, передаваемого в

заголовке Authorization: Bearer <token>. Разграничение прав реализовано на основе ролей: admin (администратор) и user (обычный сотрудник). Администратор имеет доступ ко всем функциям системы, включая управление пользователями, выдачу оборудования, обработку заявок и генерацию отчётов; пользователь может просматривать оборудование, запрашивать выдачу и просматривать историю перемещений.

Сервер приложений и клиентская часть обмениваются данными в формате JSON. Для упрощения тестирования и документирования API используется встроенная в FastAPI интерактивная документация Swagger UI, доступная по адресу /docs.

Физическая структура проекта разделена на две независимые части - backend и client, каждая из которых может разрабатываться и разворачиваться отдельно. Клиентская часть собирается с помощью инструмента Vite, серверная - запускается с использованием ASGI-сервера uvicorn. Режим работы

Непрерывная работа за компьютером вызывает зрительное напряжение. При восьмичасовом рабочем дне необходимы перерывы. Рекомендуемая продолжительность перерыва составляет 10-15 минут через каждые два часа работы.

Во время перерыва рекомендуется сменить положение тела, выполнить движения кистями и шеей. Допускается гимнастика для глаз.

3.2 Работа с базой данных

В качестве системы управления базами данных используется PostgreSQL 16 [3]. Схема данных спроектирована с учётом требований предметной области и содержит пять таблиц: users, equipment, movements, requests, а также служебные таблицы для перечислений (enum).

Ниже приведено краткое описание каждой таблицы и её полей.

Таблица users - хранит учётные записи пользователей системы:

- id - первичный ключ, целое число (SERIAL);
- username - уникальное имя для входа (строка, не может быть NULL);

- full_name - полное имя сотрудника (строка);
- hashed_password - хеш пароля, сгенерированный bcrypt (строка);
- role - роль пользователя, перечисление: admin или user (по умолчанию user);
- is_active - статус учётной записи: active или blocked (по умолчанию active).

Между пользователями и перемещениями установлена связь «один-ко-многим» (один пользователь может быть участником нескольких перемещений).

Таблица equipment - содержит информацию об единицах оборудования:

- id - первичный ключ;
- name - наименование (строка);
- inventory_number - уникальный инвентарный номер (строка, обязательный);
- type - тип оборудования (строка, например "ноутбук", "принтер");
- serial_number - серийный номер (строка, необязательный);
- purchase_date - дата приобретения (Date, может отсутствовать);
- price - цена в рублях (Float, может отсутствовать);
- location - текущее местоположение (строка, по умолчанию "Склад");
- status - текущий статус: в наличии, выдан, в ремонте, списан (перечисление, по умолчанию в наличии);
- notes - примечания (строка, необязательное);
- written_off_date - дата списания (DateTime, может отсутствовать; заполняется автоматически при изменении статуса на «списан»).

Связь с таблицей перемещений также «один-ко-многим».

Таблица movements - регистрирует все факты выдачи и возврата оборудования:

- id - первичный ключ;
- equipment_id - внешний ключ к таблице equipment;
- user_id - внешний ключ к таблице users (кому выдано);
- issue_date - дата и время выдачи (DateTime, по умолчанию текущее время UTC);

- expected_return_date - ожидаемая дата возврата (Date, необязательно);
- actual_return_date - фактическая дата возврата (DateTime, заполняется при возврате);
- notes - примечания.

При выдаче оборудования автоматически создаётся запись в этой таблице, а статус оборудования меняется на «выдан». При возврате запись обновляется — проставляется actual_return_date, а статус оборудования возвращается на «в наличии».

Таблица requests - хранит заявки на выдачу, создаваемые пользователями:

- id - первичный ключ;
- equipment_id - внешний ключ к оборудованию;
- user_id - внешний ключ к пользователю, создавшему заявку;
- status - статус заявки: ожидает, одобрена, отклонена (перечисление);
- request_date - дата и время подачи заявки;
- expected_return_date - ожидаемая дата возврата;
- notes – примечания;
- processed_by - внешний ключ к администратору, обработавшему заявку;
- processed_date - дата обработки.

При одобрении заявки администратором автоматически создаётся запись в таблице movements, соответствующая выдача. При отклонении - только меняется статус заявки.

Все операции с базой данных выполняются через функции в модуле crud.py, которые инкапсулируют SQLAlchemy-запросы. Сессии базы данных создаются с помощью SessionLocal - генератора сессий, а для внедрения зависимости используется функция get_db, обеспечивающая корректное закрытие сессии после обработки запроса.

Для наглядного понимания работы БД, на рисунке 7 представлена схема физической модели базы данных.

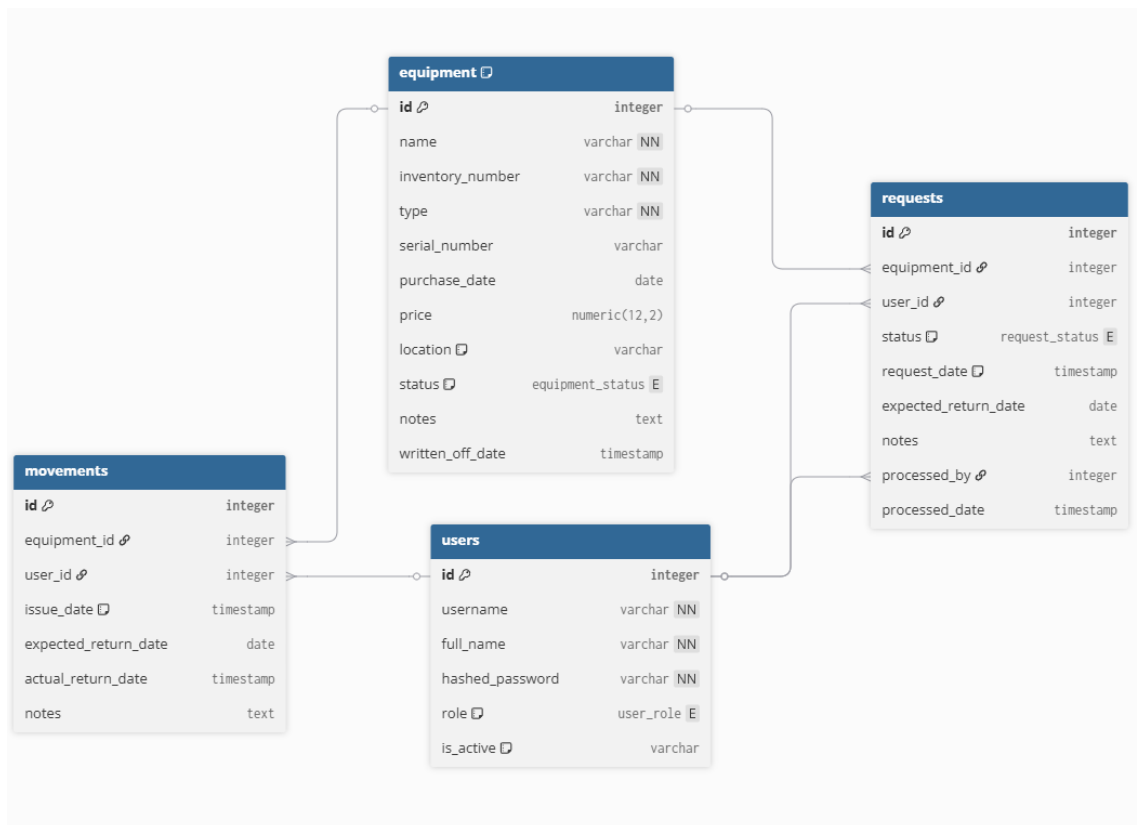


Рисунок 7 – Физическая модель базы данных

3.3 Разработка клиентского приложения

Клиентская часть представляет собой одностраничное веб-приложение (Single Page Application), разработанное с использованием библиотеки React и языка JavaScript (с применением синтаксиса JSX) [4]. Сборка проекта осуществляется инструментом Vite, который обеспечивает быстрый старт разработки и горячую замену модулей.

Структура клиентской части (каталог client/src):

- api/ - модуль для взаимодействия с REST API;
- components/ - набор React-компонентов, реализующих пользовательский интерфейс;
- App.jsx - корневой компонент, управляющий маршрутизацией и состоянием аутентификации;
- main.jsx - точка входа в приложение;

- index.html - базовая HTML-страница, в которую встраивается React-приложение.

Маршрутизация внутри приложения реализована с помощью библиотеки react-router-dom. В зависимости от состояния аутентификации (наличия токена) пользователь видит либо страницу входа, либо основной интерфейс с навигационной панелью и соответствующим содержимым. При успешной аутентификации JWT-токен и роль пользователя сохраняются в localStorage, что позволяет сохранять сессию между обновлениями страницы.

3.3.1 Основные компоненты и их назначение.

Login - форма аутентификации (логин/пароль). Отправляет POST-запрос на и в случае успеха вызывает колбэк onLogin, который сохраняет токен и запрашивает данные текущего пользователя через /api/users/me.

```
import { useState } from "react";
import { login } from "../api";

export default function Login({ onLogin }) {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError("");
    try {
      const res = await login(username, password);
      onLogin(res.data.access_token);
    } catch (err) {
      setError("Неверное имя пользователя или пароль");
    }
  };

  return (
    <div className="d-flex justify-content-center align-items-center" style={{ minHeight: '100vh' }}>
      <div className="card p-4 shadow" style={{ width: '100%', maxWidth: '400px' }}>
        <h3 className="text-center mb-3">Учёт оборудования</h3>
        <form onSubmit={handleSubmit}>
          <div className="mb-3">
            <label className="form-label">Логин</label>
            <input className="form-control" value={username}
              onChange={(e) => setUsername(e.target.value)} required autoFocus />
          </div>
        </form>
      </div>
    </div>
  );
}
```

```

        <div className="mb-3">
            <label className="form-label">Пароль</label>
            <input type="password" className="form-control"
value={password} onChange={(e) => setPassword(e.target.value)}
required />
        </div>
        {error && <div className="alert alert-danger py-
1">{error}</div>}
        <button type="submit" className="btn btn-primary w-
100">Войти</button>
        </form>
    </div>
</div>
);
}

```

Navbar - навигационная панель, адаптирующаяся под роль пользователя: администратор видит ссылки «Добавить», «Выдать», «Заявки», «Пользователи», «Отчёты»; обычный пользователь - «Запрос на выдачу». Кнопка «Выйти» очищает токен и роль.

EquipmentList - таблица со списком оборудования, поддерживающая поиск по текстовому запросу и фильтрацию по типу, статусу, местоположению, диапазону цены. Для администратора доступны кнопки редактирования, списания и полного удаления; для всех пользователей - ссылка на историю перемещений.

EquipmentForm - форма добавления единицы оборудования. Она же используется для редактирования (получает ID из параметров маршрута). Содержит поля, соответствующие модели оборудования. Статус «выдан» исключён из ручного выбора, так как он устанавливается только автоматически при выдаче.

IssueEquipment (доступен только администратору) - форма прямой выдачи оборудования любому сотруднику. Содержит выпадающие списки доступного оборудования и активных пользователей, а также поля даты ожидаемого возврата и примечания.

RequestEquipment (доступен всем пользователям) - форма запроса на выдачу. Обычный пользователь не выбирает получателя - заявка создаётся на его имя. Администратор в этом компоненте не нуждается.

RequestsAdmin - таблица заявок для администратора с возможностью одобрить (создаёт движение и меняет статус оборудования) или отклонить заявку. Обновляется после каждого действия.

UsersManagement - таблица пользователей для администратора с формой добавления нового пользователя и кнопками блокировки/разблокировки. Администраторов нельзя заблокировать.

MovementsHistory - история перемещений для выбранного оборудования. Отображает каждую запись: дату выдачи, сотрудника, ожидаемую и фактическую даты возврата, примечание. Для незавершённых выдач отображается кнопка «Вернуть», которая видна только администратору или тому пользователю, которому выдано оборудование.

Reports - страница отчётов, содержит три блока: «Состояние оборудования» (с фильтрацией), «Выданное оборудование» (на дату) и «Списанное оборудование за период». При скачивании формируется файл Excel (XLSX).

Для взаимодействия с сервером весь код API сосредоточен в модуле api/index.js. Экземпляр axios предварительно настроен: базовый URL http://localhost:8000/api, и ко всем запросам автоматически добавляется заголовок Authorization с сохранённым токеном.

Визуальное оформление выполнено с использованием CSS-фреймворка Bootstrap 5 [5]. Благодаря этому интерфейс является адаптивным и современным без необходимости писать сложные стили.

Модуль взаимодействия с API (api/index.js):

```
import axios from 'axios';

const API = axios.create({
  baseURL: 'http://localhost:8000/api',
});

API.interceptors.request.use(config => {
  const token = localStorage.getItem('access_token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});
```

```

export const login = (username, password) => {
  const params = new URLSearchParams();
  params.append('username', username);
  params.append('password', password);
  return API.post('/auth/login', params, {
    headers: { 'Content-Type': 'application/x-www-form-urlencoded' }
  });
};

export const getMe = () => API.get('/users/me');
export const getEquipment = (params) => API.get('/equipment/', {
  params });
export const createEquipment = (data) => API.post('/equipment/',
  data);
export const updateEquipment = (id, data) =>
  API.put(`/equipment/${id}`, data);
export const deleteEquipment = (id) =>
  API.delete(`/equipment/${id}`);
export const hardDeleteEquipment = (id) =>
  API.delete(`/equipment/${id}/hard`);
export const getUsers = () => API.get('/users/');
export const createUser = (data) => API.post('/users/', data);
export const updateUser = (id, data) => API.put(`/users/${id}`,
  data);
export const issueEquipment = (data) => API.post('/movements/issue',
  data);
export const returnEquipment = (id, data) =>
  API.post(`/movements/return/${id}`, data);
export const getMovementsForEquipment = (id) =>
  API.get(`/movements/equipment/${id}`);
export const createRequest = (data) => API.post('/requests/', data);
export const getRequests = () => API.get('/requests/');
export const approveRequest = (id) =>
  API.put(`/requests/${id}/approve`);
export const rejectRequest = (id) =>
  API.put(`/requests/${id}/reject`);
export default API;

```

Диаграмма последовательностей, показывающая процесс входа, изображена на рисунке 8.

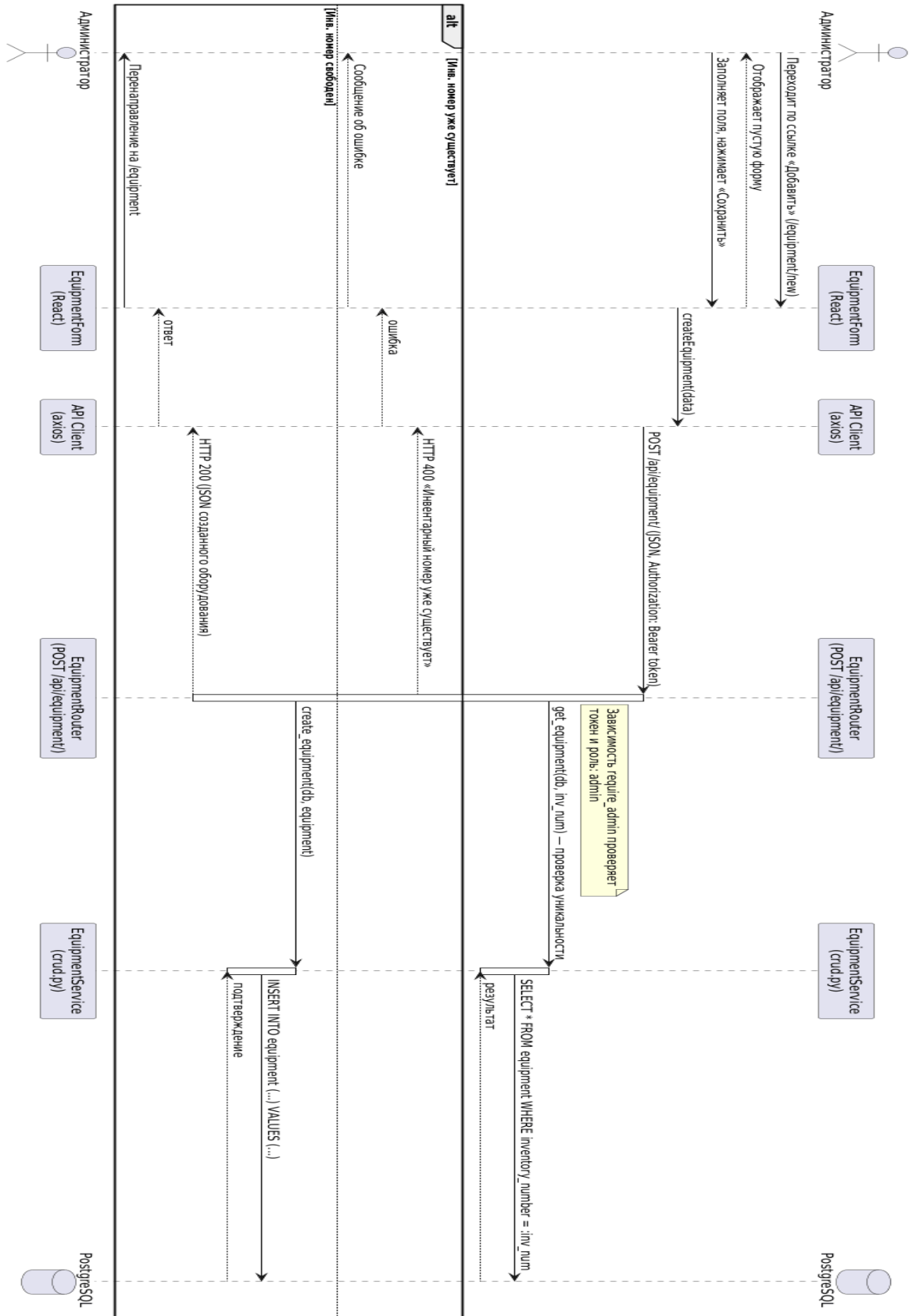


Рисунок 9 - Диаграмма последовательности «Добавление»

3.4 Разработка серверного приложения

Серверная часть реализована на языке Python с использованием веб-фреймворка FastAPI [6]. Вся бизнес-логика сосредоточена в каталоге backend/app.

Основные модули сервера:

- main.py - инициализация приложения FastAPI, подключение CORS, регистрация роутеров, создание таблиц базы данных при старте;
- database.py - настройка подключения к PostgreSQL через SQLAlchemy, создание engine, SessionLocal и декларативного базового класса Base. Также содержит функцию-генератор get_db для внедрения зависимости сессии в маршруты;
- models.py - определение ORM-моделей (User, Equipment, Movement, Request) и перечислений (UserRole, EquipmentStatus, RequestStatus). Описывает связи между таблицами;
- schemas.py - Pydantic-схемы для валидации входящих запросов и сериализации ответов. Включает схемы для создания, обновления и чтения каждой сущности, а также модели для аутентификации (LoginRequest, Token);
- crud.py - функции для работы с БД: создание, чтение, обновление, (мягкое) удаление записей. Например, get_equipments поддерживает поиск (по нескольким полям через ILIKE) и фильтрацию. Функция delete_equipment лишь меняет статус на «списан» и фиксирует дату списания. Для полного удаления используется hard_delete_equipment, которая удаляет запись и связанные перемещения;
- auth.py - модуль аутентификации и авторизации. Содержит функции хеширования и проверки паролей (Passlib + bcrypt), создания JWT-токенов (python-jose), зависимости get_current_user (извлекает пользователя по токену) и require_admin (проверяет роль);
- routers/ - каталог с модулями маршрутизации;
- users.py - эндпоинты регистрации и управления пользователями, а также POST /auth/login (принимает OAuth2PasswordRequestForm или LoginRequest, возвращает JWT-токен);

- equipment.py - CRUD-операции с оборудованием с поддержкой поиска и фильтрации;
- movements.py - выдача и возврат оборудования, просмотр истории;
- requests.py - создание, просмотр, одобрение и отклонение заявок;
- reports.py - генерация отчётов в формате Excel (XLSX) или CSV.

Каждый роутер имеет префикс (/api/equipment, /api/users и т.д.) и зарегистрирован в main.py.

3.4.1 Инициализация сервера.

При старте приложения создаётся объект FastAPI с заголовком «Equipment Accounting System».

Настраивается middleware CORS, разрешающий запросы с адресом клиентского приложения. Это позволяет браузеру свободно отправлять кросс-доменные запросы, включая предварительные OPTIONS, и передавать авторизационные заголовки.

Подключаются роутеры (equipment.router, users.router, movements.router, requests.router, reports.router), каждый из которых отвечает за свою группу эндпоинтов.

Инициализация приложения (main.py):

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from .database import engine, Base
from .routers import equipment, users, movements, requests, reports

Base.metadata.create_all(bind=engine)

app = FastAPI(title="Equipment Accounting System")

origins = ["http://localhost:5173"]
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(equipment.router)
app.include_router(users.router)
app.include_router(movements.router)
```

```
app.include_router(requests.router)
app.include_router(reports.router)
```

3.4.2 Обработка сетевых соединений и протокол обмена.

Сервер общается с клиентом по протоколу HTTP/1.1. Все данные передаются в формате JSON (за исключением эндпоинта логина, который принимает данные формы `application/x-www-form-urlencoded` для совместимости с OAuth2). Каждый ответ содержит заголовок `Content-Type: application/json` и соответствующий код состояния.

Uvicorn, работающий поверх ASGI, обрабатывает входящие TCP-соединения асинхронно, что позволяет эффективно обслуживать множество одновременных клиентов без блокировки потоков. FastAPI, в свою очередь, разбирает URL и HTTP-заголовки, извлекает параметры запроса, тело и передаёт управление соответствующей функции-обработчику.

Аутентификация построена на JWT-токенах. Клиент, получивший токен через `POST /api/auth/login`, обязан передавать его в заголовке `Authorization: Bearer <token>` при каждом защищённом запросе. Зависимость `get_current_user` извлекает токен, декодирует его с помощью секретного ключа, проверяет срок действия и загружает пользователя из базы. Если токен недействителен или пользователь заблокирован, возвращается ошибка 401. Для маршрутов, требующих прав администратора, дополнительно используется зависимость `require_admin`, которая проверяет роль.

Аутентификация и авторизация (auth.py, фрагмент):

```
from datetime import datetime, timedelta
from jose import jwt
from passlib.context import CryptContext
from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
from . import models, database

SECRET_KEY = "секретный-ключ"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 60

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="api/auth/login")
```

```

def create_access_token(data: dict):
    to_encode = data.copy()
    expire = datetime.utcnow() +
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

def get_current_user(token: str = Depends(oauth2_scheme), db:
Session = Depends(database.get_db)):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Не удалось проверить учётные данные",
    )
    try:
        payload = jwt.decode(token, SECRET_KEY,
algorithm=[ALGORITHM])
        username = payload.get("sub")
        if username is None:
            raise credentials_exception
    except JWTError:
        raise credentials_exception
    user = db.query(models.User).filter(models.User.username ==
username).first()
    if user is None or user.is_active != "active":
        raise credentials_exception
    return user

def require_admin(current_user: models.User =
Depends(get_current_user)):
    if current_user.role != models.UserRole.admin:
        raise HTTPException(status_code=403, detail="Требуется права
администратора")
    return current_user

```

3.4.3 Работа с базой данных.

Взаимодействие с PostgreSQL осуществляется через ORM SQLAlchemy. Подключение настраивается в database.py: создаётся engine с параметрами подключения, фабрика сессий SessionLocal и базовый класс Base для декларативного описания моделей.

Для каждого запроса через функцию-генератор get_db создаётся новая сессия базы данных, которая автоматически закрывается после обработки запроса. Это реализовано с помощью системы зависимостей FastAPI: в эндпоинтах параметр db: Session = Depends(database.get_db) получает текущую сессию. Такой подход гарантирует, что соединения не удерживаются дольше необходимого, а в

случае исключения сессия корректно закрывается (благодаря блоку try/finally внутри get_db).

Все операции CRUD вынесены в отдельный модуль crud.py. Каждая функция (get equipments, create_user, create_movement и др.) принимает сессию и необходимые данные, формирует запрос к БД с помощью SQLAlchemy, при необходимости управляет транзакциями (через commit и rollback) и возвращает результат. Например, при выдаче оборудования (create_movement) внутри одной сессии выполняются сразу два действия: вставляется запись в таблицу movements и обновляется статус оборудования. При ошибке на любом этапе транзакция откатывается.

Для поиска и фильтрации используется построение динамических запросов: функция get equipments принимает опциональные параметры search, type, status, location, min_price, max_price и добавляет соответствующие условия WHERE с помощью query.filter(). Поиск реализован с использованием оператора ILIKE (регистронезависимый поиск подстроки) по нескольким полям сразу.

Функция поиска и фильтрации оборудования (crud.py):

```
from sqlalchemy.orm import Session
from sqlalchemy import or_
from . import models

def get equipments(db: Session, skip: int = 0, limit: int = 100,
search: str = None, **filters):
    query = db.query(models.Equipment)

    if search:
        term = f"%{search}%"
        query = query.filter(
            or_(
                models.Equipment.name.ilike(term),
                models.Equipment.inventory_number.ilike(term),
                models.Equipment.serial_number.ilike(term),
                models.Equipment.location.ilike(term),
                models.Equipment.notes.ilike(term),
            )
        )

    if filters.get("type"):
        query = query.filter(models.Equipment.type ==
filters["type"])
    if filters.get("status"):
```

```

        query = query.filter(models.Equipment.status ==
filters["status"])
        if filters.get("location"):
            query =
query.filter(models.Equipment.location.ilike(f"%{filters['location']}
%"))
        if filters.get("min_price") is not None:
            query = query.filter(models.Equipment.price >=
filters["min_price"])
        if filters.get("max_price") is not None:
            query = query.filter(models.Equipment.price <=
filters["max_price"])

    return query.offset(skip).limit(limit).all()

```

3.4.4 Обработка действий (бизнес-логика).

Маршрутизация запросов распределена по файлам в каталоге routers/. Каждый модуль содержит функции-обработчики, декорированные аннотациями HTTP-методов (например, `@router.get("/")`, `@router.post("/issue")`). FastAPI автоматически генерирует интерактивную документацию Swagger UI на основе этих аннотаций и Pydantic-схем.

Входящие данные проходят строгую валидацию через Pydantic-модели, описанные в schemas.py. Если данные не соответствуют ожидаемой структуре (например, отсутствует обязательное поле или неверный тип), клиент немедленно получает ответ 422 с перечнем ошибок, и дальнейшая обработка не выполняется.

После валидации вызываются соответствующие CRUD-функции. В обработчиках реализованы дополнительные проверки бизнес-правил:

- нельзя выдать оборудование, которого нет в наличии (статус не «в наличии»);
- инвентарный номер должен быть уникальным;
- статус «выдан» устанавливается только автоматически при создании движения или одобрении заявки;
- возврат оборудования может выполнить только тот пользователь, которому оно выдано, либо администратор;
- при списании оборудования (через `deleteEquipment`) фактического удаления записи не происходит - статус меняется на «списан» и фиксируется дата списания.

Полное удаление (эндпоинт `hardDeleteEquipment`) доступно только администраторам и удаляет запись вместе со связанными перемещениями;

- обычный пользователь не может одобрять или отклонять заявки.

Все проверки сопровождаются соответствующими HTTP-исключениями (404 – не найдено, 400 – неверный запрос, 403 – недостаточно прав), что позволяет клиенту корректно реагировать на ошибки и отображать пользователю понятные сообщения.

Особенности серверной реализации:

Безопасность: все пароли хешируются `bcrypt`; JWT-токены выпускаются на 60 минут.

Валидация: благодаря `Pydantic` все входные данные строго типизированы; при несоответствии автоматически возвращается ответ 422 с описанием ошибок.

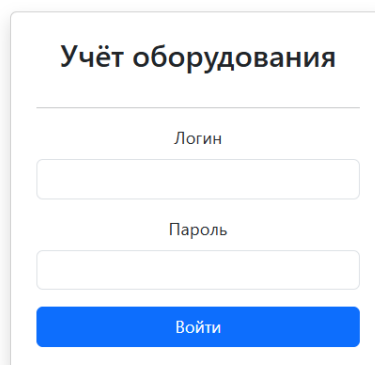
Обработка ошибок: во всех эндпоинтах предусмотрены проверки существования записей и прав доступа, что сопровождается соответствующими HTTP-статусами.

Асинхронная работа: `FastAPI` использует `asyncio`, однако для совместимости с синхронным `SQLAlchemy` операции с БД выполняются через `run_in_threadpool`.

3.5 Тестирование

Тест 1 - Авторизация.

Чтобы начать работу с программой, необходимо пройти авторизацию (рисунок 10). Для этого в поля «Логин» и «Пароль» нужно ввести учетные данные пользователя, после чего нажать кнопку «Войти».



Учёт оборудования

Логин

Пароль

Войти

Рисунок 10 – Окно входа

Учёт оборудования

Логин

Пароль

Неверное имя пользователя или пароль

Войти

Рисунок 11 – Ошибка при входе

Тест 2 – Просмотр списка оборудования.

Если вход прошёл успешно, откроется главная страница пользователя (рисунок 12). В случае, когда пароль не подходит или такого пользователя не существует, программа выдаст предупреждение об ошибке (рисунок 11).

Учёт оборудования									
Оборудование									Выйти
Список оборудования									
Поиск по названию, инв. номеру, местоположению, примечанию									
Введите текст для поиска...									
Найти									
Тип Статус Местоположение Цена от Цена до									
Все Все Кабинет... 0 50000									
Инв. номер	Название	Тип	Серийный номер	Дата покупки	Цена (₽)	Статус	Местоположение	Примечание	Действия
SV-001	Сервер HP ProLiant	сервер	SN333	—	250 000,00	в наличии	Серверная	—	
NB-001	Ноутбук HP ProBook	ноутбук	SN111	—	65 000,00	в наличии	Каб. 101	—	
PR-001	Принтер Canon LBP	принтер	SN222	—	12 000,00	в наличии	Каб. 203	—	

Рисунок 12 – Главная страница пользователя

Тест 3 – Поиск и фильтрация оборудования.

Для поиска оборудования можно воспользоваться строкой поиска. В списке отобразятся подходящие варианты (рисунок 13).

Список оборудования									
Поиск по названию, инв. номеру, местоположению, примечанию									
HP									
Найти									
Тип Статус Местоположение Цена от Цена до									
Все Все Кабинет... 0 50000									
Инв. номер	Название	Тип	Серийный номер	Дата покупки	Цена (₽)	Статус	Местоположение	Примечание	Действия
SV-001	Сервер HP ProLiant	сервер	SN333	—	250 000,00	в наличии	Серверная	—	
NB-001	Ноутбук HP ProBook	ноутбук	SN111	—	65 000,00	в наличии	Каб. 101	—	

Рисунок 13 – Поиск оборудования

При нажатии поля фильтра «Тип» покажется всплывающий список типов (рисунок 14), если оборудования с данным типом нет, то выведется надпись «нет данных» (рисунок 15). При нажатии поля фильтра «Статус» покажется всплывающий список оборудования с данным статусом, если оборудования с данным статусом нет, то выведется надпись «нет данных» (рисунок 15). При написании в поля «Местоположение, Цена от, Цена до», произойдѣи фильтрация по этим данным, в поля цены нельзя написать ничего кроме цифр и запятой. Пример фильтрации на рисунке 16.

Тип

Статус

Местоположение

Цена от

Цена до

ноутбук

Все

Кабинет...

0

50000

Иин. номер	Название	Тип	Серийный номер	Дата покупки	Цена (₽)	Статус	Местоположение	Примечание	Действия
NB-001	Ноутбук HP ProBook	ноутбук	SN111	—	65 000,00	в наличии	Каб. 101	—	

Рисунок 14 – Фильтр «Тип»

Тип		Статус		Местоположение		Цена от		Цена до	
периферия		Все		Кабинет...		0		50000	
Иин. номер	Название	Тип	Серийный номер	Дата покупки	Цена (Р)	Статус	Местоположение	Примечание	Действия
Нет данных									

Рисунок 15 – Пустой список

Тип		Статус	Местоположение		Цена от	Цена до			
ноутбук	▼	в наличии	▼	Кабинет...	12000,01		70000		
Иин. номер	Название	Тип	Серийный номер	Дата покупки	Цена (₽)	Статус	Местоположение	Примечание	Действия
NB-001	Ноутбук HP ProBook	ноутбук	SN111	—	65 000,00	в наличии	Каб. 101	—	

Рисунок 16 – Пример фильтрации

Тест 4 – Запрос на выдачу оборудования.

При нажатии на вкладку «Запрос на выдачу» откроется форма для запроса оборудования (рисунок 17), в строке «Оборудование» будет список доступного оборудования, в строку «Ожидаемая дата возврата» можно писать только цифры, также есть строка для примечания. После заполнения формы и нажатия кнопки «Отправить запрос», выведется уведомления об отправленном запросе (рисунок 18), в случае если оставить поля пустыми, выведется подсказка о необходимости заполнения полей (рисунок 19)

Запрос на выдачу оборудования

Оборудование

-- выберите --

Ожидаемая дата возврата

дд.мм.гггг

Примечание

Отправить заявку

Рисунок 17 – Запрос на выдачу оборудования

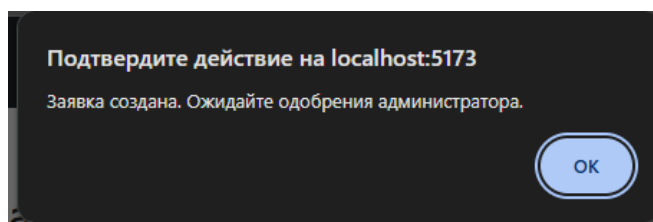


Рисунок 18 – Уведомление об отправке заявки

Запрос на выдачу оборудования

Оборудование

-- выберите --

Выберите один из пунктов списка.

дд.мм.гггг

Примечание

Отправить заявку

Рисунок 19 – Запрос с пустыми полями

Тест 5 – История перемещений.

При нажатии на вкладку «Оборудование» обратно откроется страница со списком оборудования (рисунок 12). При нажатии на кнопку истории (рисунок 19) выведется история перемещений данного оборудования (рисунок 20).



Рисунок 19 – Кнопка истории

Учёт оборудования

Оборудование

Запрос на выдачу

Выйти

История перемещений (ID оборудования: 1)

Дата выдачи	Сотрудник	Ожидаемая дата возврата	Фактическая дата возврата	Примечание	Действия
06.05.2026, 18:30:43	test11	2026-05-08	06.05.2026, 18:36:16	По заявке #1:	
06.05.2026, 17:51:04	test11	2026-05-07	06.05.2026, 18:36:16	—	
06.05.2026, 17:33:28	test11	2026-05-07	06.05.2026, 18:36:17	—	
06.05.2026, 18:39:01	test11	2222-02-22	06.05.2026, 18:39:38	По заявке #2:	

Рисунок 20 – Вкладка истории оборудования

При нажатии на кнопку «Выйти» открывается страница входа (рисунок 10). При входе за администратора появляется новый функционал, а именно вкладки «Добавить», «Выдать», «Заявки», «Пользователи», «Отчёты». А также новые действия с оборудованием, кнопки: «Списать», «Удалить», «Редактировать» (рисунок 21).

Тест 6 – Добавление оборудования.

Учёт оборудования Оборудование Добавить Выдать Заявки Пользователи Отчёты Выйти									
Список оборудования									
Поиск по названию, инв. номеру, местоположению, примечанию									
Введите текст для поиска... Найти									
Тип	Статус	Местоположение	Цена от	Цена до					
Все	Все	Кабинет...	0	50000					
Инв. номер	Название	Тип	Серийный номер	Дата покупки	Цена (₽)	Статус	Местоположение	Примечание	Действия
NB-001	Ноутбук HP ProBook	ноутбук	SN111	—	65 000,00	в наличии	Каб. 101	—	   
PR-001	Принтер Canon LBP	принтер	SN222	—	12 000,00	в наличии	Каб. 203	—	   
SV-001	Сервер HP ProLiant	сервер	SN333	—	250 000,00	в наличии	Серверная	—	   

Рисунок 21 – Страница администратора

При нажатии на вкладку «Добавить», открывается страница с формой добавления оборудования (рисунок 22). Обязательные поля для заполнения: «Название», «Инвентарный номер», «Тип», «Статус», «Местоположение». После заполнения данных полей можно нажать кнопку «Сохранить» для сохранения, после чего откроется список (рисунок 23)

Учёт оборудования Оборудование Добавить Выдать Заявки Пользователи Отчёты Выйти									
Добавление оборудования									
Название					Инвентарный номер				
Тип					Серийный номер				
Дата покупки					Цена (руб.)				
Местоположение					Статус				
Примечания									
Сохранить Отмена									

Рисунок 22 – Страница добавления

Инв. номер	Название	Тип	Серийный номер	Дата покупки	Цена (₽)	Статус	Местоположение	Примечание	Действия
NB-001	Ноутбук HP ProBook	ноутбук	SN111	—	65 000,00	в наличии	Каб. 101	—	   
PR-001	Принтер Canon LBP	принтер	SN222	—	12 000,00	в наличии	Каб. 203	—	   
SV-001	Сервер HP ProLiant	сервер	SN333	—	250 000,00	в наличии	Серверная	—	   
111	тест1	другое	—	—	—	в наличии	Склад	—	   

Рисунок 23 – Список с новым оборудованием

Тест 7 – Выдача оборудования администратором.

При переходе во вкладку «Выдать» открывается форма для выдачи оборудования любому пользователю (рисунок 24), поля «Оборудование» и «Сотрудник» обязательные для заполнения.

Рисунок 24 – Вкладка «Выдать»

При переходе во вкладку «Заявки», открывается список заявок на выдачу оборудования (рисунок 25), заявку можно отклонить, либо одобрить, тогда статус оборудования поменяется на «выдан» (рисунок 26), а у пользователя которому оно было выдано в истории появится кнопка «вернуть» (рисунок 27). После возврата в истории добавиться фактическая дата возвращения, а статус измениться обратно на «в наличии».

Оборудование	Заявитель	Дата заявки	Ожидаемый возврат	Статус	Действия
PR-001 Принтер Canon LBP	test11	07.05.2026, 18:19:07	2026-05-22	ожидает	Одобрить Отклонить
SV-001 Сервер HP ProLiant	test11	07.05.2026, 18:16:01	—	отклонена	
SV-001 Сервер HP ProLiant	test11	07.05.2026, 18:13:07	1111-11-11	одобрена	
PR-001 Принтер Canon LBP	test22	06.05.2026, 18:42:25	2222-02-22	одобрена	
NB-001 Ноутбук HP ProBook	test11	06.05.2026, 18:38:23	2222-02-22	одобрена	
NB-001 Ноутбук HP ProBook	test11	06.05.2026, 18:23:21	2026-05-08	одобрена	

Рисунок 25 – Вкладка «Заявки»

Изм. номер	Название	Тип	Серийный номер	Дата покупки	Цена (₽)	Статус	Местоположение	Примечание	Действия
NB-001	Ноутбук HP ProBook	ноутбук	SN111	—	65 000,00	в наличии	Каб. 101	—	✎ 🗑 ✖ 🔄
SV-001	Сервер HP ProLiant	сервер	SN333	—	250 000,00	в наличии	Серверная	—	✎ 🗑 ✖ 🔄
111	test1	другое	—	—	—	в наличии	Склад	—	✎ 🗑 ✖ 🔄
PR-001	Принтер Canon LBP	принтер	SN222	—	12 000,00	в наличии	Каб. 203	—	✎ 🗑 ✖ 🔄

Рисунок 26 – Статус «выдан»

Дата выдачи	Сотрудник	Ожидаемая дата возврата	Фактическая дата возврата	Примечание	Действия
06.05.2026, 18:42:39	test22	2222-02-22	06.05.2026, 18:45:56	По заявке #3:	
07.05.2026, 18:46:43	test11	2026-05-22	Не возвращено	По заявке #6:	Вернуть

Рисунок 27 – История выданного оборудования

Во вкладке «Пользователи» администратор видит список пользователей, также может добавить нового или заблокировать (рисунок 28).

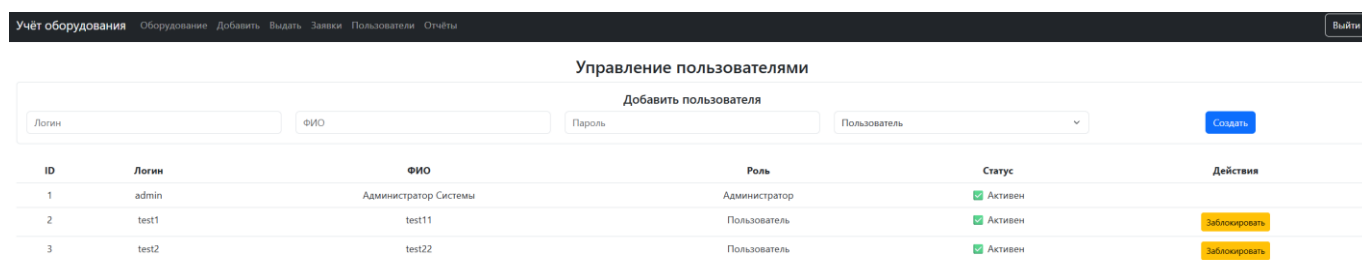


Рисунок 28 – Вкладка «Пользователи»

Тест 8 – Генерирование отчётов.

Во вкладке «Отчёты» можно сгенерировать 3 типа отчётов: «Состояние оборудования», «Выданное оборудование», «Списанное оборудование». Примеры отчётов представлены на рисунках 29-31

equipment_state.xlsx										
	A	B	C	D	E	F	G	H	I	J
1	Инв.номер	Название	Тип	Серийный номер	Дата покупки	Цена	Статус	Местоположение	Примечание	
2	NB-001	Ноутбук HP ProBook	ноутбук	SN111		65000	EquipmentStatus.available	Каб. 101		
3	SV-001	Сервер HP ProLiant	сервер	SN333		250000	EquipmentStatus.available	Серверная		
4	111	тест1	другое				EquipmentStatus.available	Склад		
5	PR-001	Принтер Canon LBP	принтер	SN222		12000	EquipmentStatus.available	Каб. 203		
6										

Рисунок 29 – Отчёт «Состояние оборудования»

equipment_state.xlsx

issued_equipment.xlsx

	A	B	C	D	E
1	Оборудование	Сотрудник	Дата выдачи	Ожидаемый возврат	Примечание
2	PR-001 Принтер Canon LBP	test11	2026-05-07 18:55:25	2026-05-25	Пользуйся аккуратно
3					

Рисунок 30 – Отчёт «Выданное оборудование»

equipment_state.xlsx		issued_equipment.xlsx		written_off_equipment.xlsx					
	A	B	C	D	E	F	G	H	I
1	Инв.номер	Название	Тип	Серийный номер	Дата покупки	Цена	Дата списания	Местоположение	Примечание
2	111	тест1	другое				2026-05-07 18:56:41	Склад	
3									

Рисунок 31 – Отчёт «Списанное оборудование».

Тест 9 – Редактирование оборудования.

При нажатии кнопки «Редактировать» напротив оборудования откроется форма редактирования (рисунок 32)

Редактирование оборудования

Название test1	Инвентарный номер 111
Тип Другое	Серийный номер
Дата покупки	Цена (руб.)
Местоположение	Статус
Склад	Списан
Примечание	

Рисунок 32 – Форма редактирования

Инв. номер	Название	Тип	Серийный номер	Дата покупки	Цена (₽)	Статус	Местоположение	Примечание	Действия
NB-001	Ноутбук HP ProBook	ноутбук	SN111	—	65 000,00	в наличии	Каб. 101	—	  
SV-001	Сервер HP ProLiant	сервер	SN333	—	250 000,00	в наличии	Серверная	—	  
PR-001	Принтер Canon LBP	принтер	SN222	—	12 000,00	в наличии	Каб. 203	—	  
111	test1	другое	—	—	—	в наличии	Склад	Редактирован	  

Рисунок 33 – Редактированное оборудование

Тест 10 – Списание и удаление.

При нажатии кнопки «Списать», появиться уведомление с подтверждением (рисунок 34) и после нажатия кнопки «ок» статус оборудования измениться на «списан», и оно останется в БД (рисунок 35).

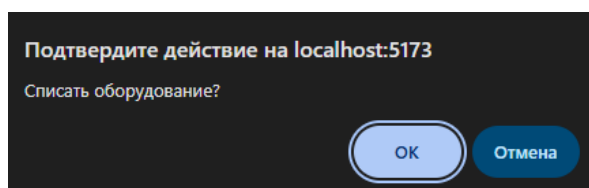


Рисунок 34 – Подтверждение о списании

Инв. номер	Название	Тип	Серийный номер	Дата покупки	Цена (₽)	Статус	Местоположение	Примечание	Действия
NB-001	Ноутбук HP ProBook	ноутбук	SN111	—	65 000,00	в наличии	Каб. 101	—	  
SV-001	Сервер HP ProLiant	сервер	SN333	—	250 000,00	в наличии	Серверная	—	  
PR-001	Принтер Canon LBP	принтер	SN222	—	12 000,00	в наличии	Каб. 203	—	  
111	test1	другое	—	—	—	списан	Склад	Редактирован	  

Рисунок 35 – Статус «списан»

При нажатии кнопки «Удалить навсегда», появиться уведомление с подтверждением (рисунок 36), после подтверждения информация об оборудовании удалиться из списка и из БД (рисунок 37).

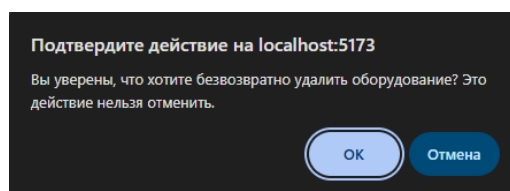


Рисунок 36 – Подтверждение о удалении

Иив. номер	Название	Тип	Серийный номер	Дата покупки	Цена (Р)	Статус	Местоположение	Примечание	Действия
NB-001	Ноутбук HP ProBook	ноутбук	SN111	—	65 000,00	в наличии	Каб. 101	—	  
SV-001	Сервер HP ProLiant	сервер	SN333	—	250 000,00	в наличии	Серверная	—	  
PR-001	Принтер Canon LBP	принтер	SN222	—	12 000,00	в наличии	Каб. 203	—	  

Рисунок 37 – Список после удаления единицы оборудования

4 Безопасность жизнедеятельности

Разработанная клиент-серверная информационная система учёта оборудования предназначена для эксплуатации на ПК в офисных условиях. Пользователи системы проводят за компьютером длительное время, выполняя операции по вводу данных, просмотру таблиц и формированию отчётов. В связи с этим необходимо проанализировать возможные опасные и вредные факторы, возникающие в процессе работы, и разработать мероприятия по их снижению.

4.2 Организация рабочего места пользователя

Основные требования к помещению: площадь на одно рабочее место – не менее 4,5 м²; высота помещения – не менее 2,5 м; естественное и искусственное освещение должно обеспечивать освещённость на рабочем столе не менее 300 лк; отсутствие бликов на экране (размещение монитора перпендикулярно окнам).

Требования к рабочему столу и стулу: рабочий стол должен иметь регулируемую высоту (или соответствовать росту пользователя): оптимальная высота – 725 мм; ширина стола – не менее 800 мм, глубина – не менее 600 мм; стул должен быть подъёмно-поворотным, с регулировкой высоты и наклона спинки;

Расположение монитора: расстояние от глаз до экрана – 50-70 см; верхний край монитора – на уровне глаз или чуть ниже; угол наклона экрана – до 15° для уменьшения бликов.

Параметры монитора: частота обновления – не менее 75 Гц, для ЖК-дисплеев достаточно 60 Гц; яркость и контрастность устанавливаются по ощущению комфорта (обычно яркость – 80–120 кд/м²); изображение должно быть стабильным, без мерцаний и дрожаний.

Клавиатура и мышь: клавиатура – на уровне локтей, угол наклона 5–12°; мышь – рядом с клавиатурой; использование запястных опор для профилактики туннельного синдрома.

4.3 Режим труда и отдыха при работе с информационной системой

Длительная работа за компьютером вызывает зрительное и психофизическое утомление. Для поддержания работоспособности и сохранения здоровья необходимо соблюдать перерывы.

Продолжительность непрерывной работы за ПК без регламентированного перерыва не должна превышать 2 часов.

Через каждые 40–45 минут работы следует делать короткий перерыв (5–10 минут) для отдыха глаз и смены позы.

Обеденный перерыв – 30–60 минут, через 3–4 часа от начала работы.

В перерывах рекомендуется: встать, пройтись, сделать 5–10 простых физических упражнений (повороты головы, вращения плечами); выполнить гимнастику для глаз: перевести взгляд с близкого предмета на дальний, поморгать, закрыть глаза на 1–2 минуты; проветрить помещение (3–5 минут).

Для работы с информационной системой учёта оборудования (ввод данных, поиск, просмотр отчётов) характерна средняя степень напряжённости труда. При интенсивной работе с таблицами рекомендуется каждые 30 минут делать микропаузы по 1–2 минуты (например, нажать клавиши, посмотреть в окно).

4.4 Электробезопасность и требования к электрооборудованию

ПК и серверное оборудование относятся к электроприёмникам с напряжением до 220 В. Поражение электрическим током возможно при неисправной изоляции, отсутствии заземления, работе во влажной среде или с нарушением правил эксплуатации.

Основные меры электробезопасности: все розетки и штепсельные соединения должны иметь защитное заземление; кабель питания системного блока и монитора должен быть исправным, без перегибов и оголённых участков; запрещается самостоятельно разбирать блоки питания или производить ремонт при включённом питании; электрощиты, распределительные коробки должны

быть закрыты; помещение должно быть оснащено автоматическими выключателями (дифференциальными) для защиты от токов короткого замыкания и утечки.

Влажность воздуха в помещении с ПК не должна превышать 60% (при более высокой возможно образование конденсата внутри оборудования). Оптимальная влажность – 40–50%.

Для серверного компьютера (бэкенд и база данных) следует предусмотреть дополнительное бесперебойное питание для корректного завершения работы при внезапном отключении электроэнергии.

4.5 Пожарная безопасность

Эксплуатация ПК и серверного оборудования связана с риском возгорания из-за короткого замыкания, перегрева компонентов (блок питания, процессор), неисправности электропроводки или оставленных без присмотра включённых устройств.

Помещение с ПК должно быть оснащено углекислотным или порошковым огнетушителем (тип ОУ-2 или ОП-3). Использование водных огнетушителей для тушения электрооборудования запрещено.

Автоматическая пожарная сигнализация (дымовые извещатели) в обязательном порядке устанавливается при площади помещения более 100 м². В небольших комнатах достаточно иметь автономный датчик дыма.

Пути эвакуации (дверные проёмы, коридоры) должны быть свободны, ширина прохода – не менее 1 м. Планы эвакуации при пожаре должны быть вывешены на видном месте.

Запрещается: использовать удлинители с повреждённой изоляцией, нагружать одну розетку более 1,5 кВт (через тройник), курить в помещении, оставлять включённую технику без присмотра (кроме серверов). Регулярная уборка от пыли (особенно внутри системных блоков) снижает риск перегрева.

Действия при возникновении пожара:

Немедленно отключить питание (обесточить помещение общим рубильником). Сообщить о пожаре по телефону 101 или 112. Приступить к

					ПГУ 1.090301.03.001 ПЗ	Лист
						50
Изм.	Лист	№ докум.	Подпись	Дата		

тушению первичными средствами (огнетушитель) только если очаг мал и нет угрозы жизни. При распространении огня – эвакуироваться, закрыв за собой дверь.

Заключение

В рамках выполненной выпускной квалификационной работы разработана клиент-серверная информационная система для учета оборудования, обеспечивающая разграничение доступа на основе ролей пользователей.

Программа позволяет добавлять новые единицы оборудования с указанием инвентарного номера, названия, типа, серийного номера, даты покупки, цены, местоположения и статуса. При необходимости любую запись можно отредактировать или списать, причём списанное оборудование остаётся в базе с отметкой о дате списания, а администратор также может удалить запись полностью.

Система поддерживает выдачу оборудования сотрудникам и его возврат с фиксацией всех перемещений. Любой авторизованный пользователь может создать заявку на выдачу, а администратор – одобрить или отклонить её; при одобрении оборудование автоматически переходит в статус «выдан» и создаётся соответствующая запись в истории. Для обычного пользователя доступен просмотр оборудования и истории, а также собственные заявки, для администратора – полное управление пользователями (добавление, блокировка, назначение ролей). В любом списке можно выполнять поиск по тексту и фильтрацию по типу, статусу, местоположению и цене.

Ключевые преимущества разработанной платформы связаны с удобством и безопасностью. Безопасность обеспечивается хешированием паролей, JWT-токенами и разграничением ролей, а веб-интерфейс работает во всех современных браузерах.

Отдельным плюсом является формирование трёх типов отчётов в формате Excel: состояние оборудования, список выданного на текущую или любую другую дату и перечень списанного за выбранный период. Кроме того, программа может быть установлена как на выделенный сервер, так и в локальной сети.

Дальнейшее развитие имеет множество направлений. Полезным будет добавление модуля штрихкодирования для быстрой идентификации оборудования. Еще одно направление – реализация уведомлений по электронной почте о приближении срока возврата. Кроме того, интеграция с системами бухгалтерского учёта для автоматического расчёта амортизации значительно упростит работу пользователей.

					ПГУ 1.090301.03.001 ПЗ	Лист
						52
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Пол Б. Изучаем программирование на Python. – Эксмо: Питер, 2018 – 624 с.
2. Флэнаган Д. JavaScript. Полное руководство. 7-е изд. – СПб: Питер, 2021 – 732 с.
3. Официальная документация PostgreSQL. [Электронный ресурс] / Режим доступа: <https://www.postgresql.org/docs/current/index.html>, свободный (дата обращения 15.04.2026).
4. Официальная документация React. [Электронный ресурс] / Режим доступа: <https://react.dev/learn>, свободный (дата обращения 16.04.2026).
5. Официальная документация Bootstrap 5. [Электронный ресурс] / Режим доступа: <https://getbootstrap.com/docs/5.0/getting-started/introduction/>, свободный (дата обращения 17.04.2026).
6. Официальная документация FastAPI. [Электронный ресурс] / Режим доступа: <https://fastapi.tiangolo.com/learn>, свободный (дата обращения 10.04.2026).

Приложение А

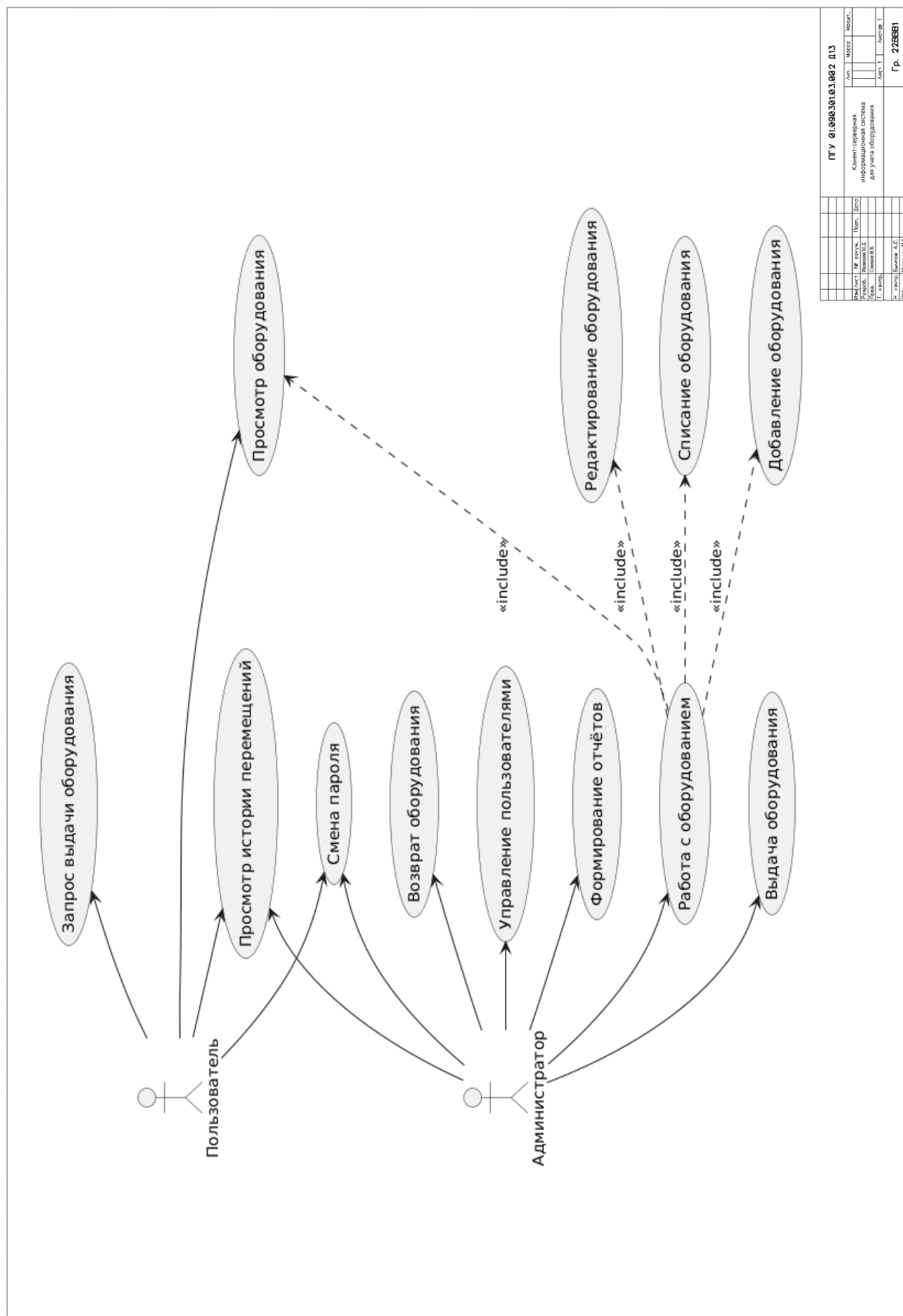
(обязательное)

Клиент-серверная информационная система для учета оборудования

UML-диаграммы

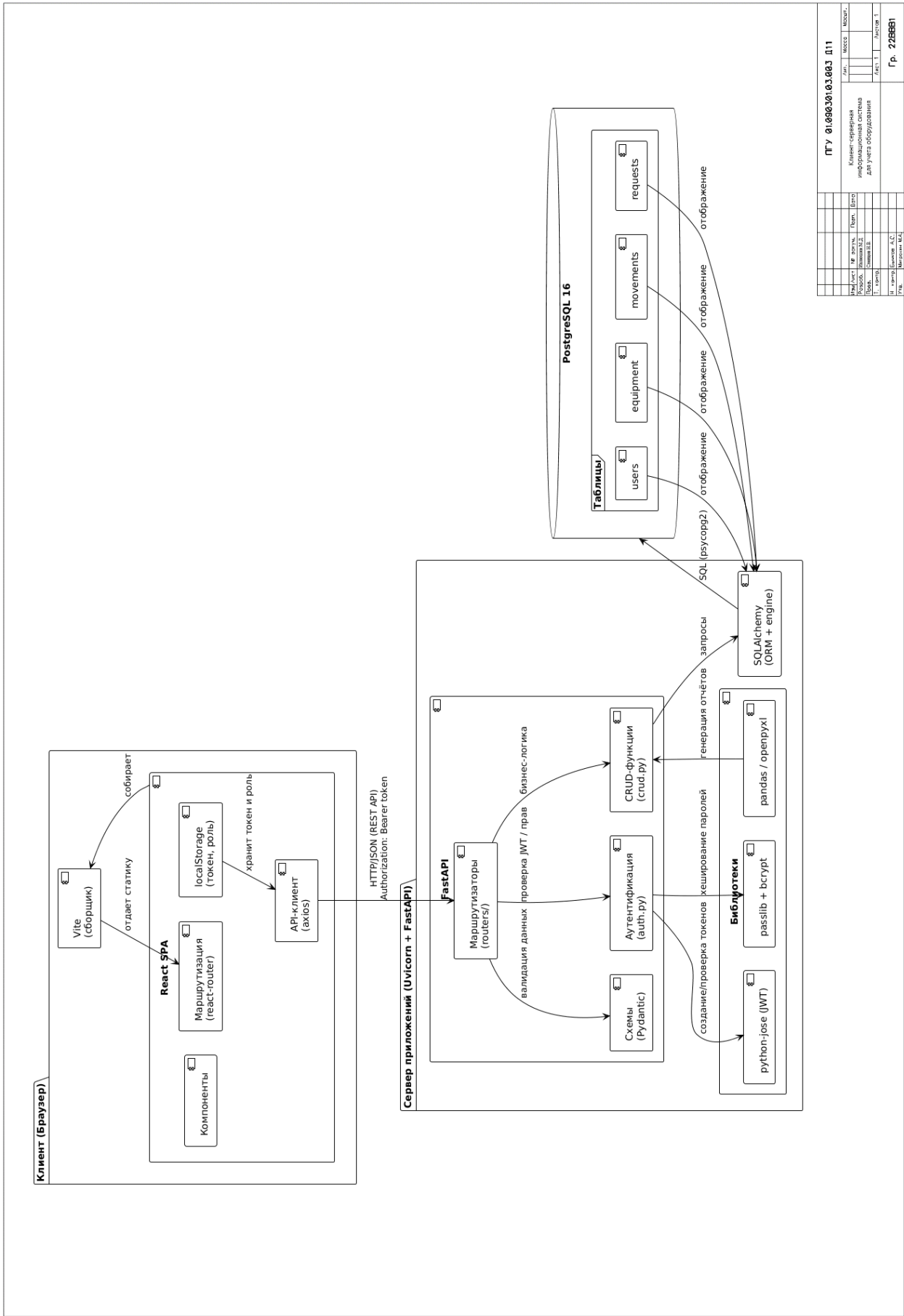
					ПГУ 1.090301.03.001 ПЗ	Лист
						54
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – UML-диаграмма прецедентов использования



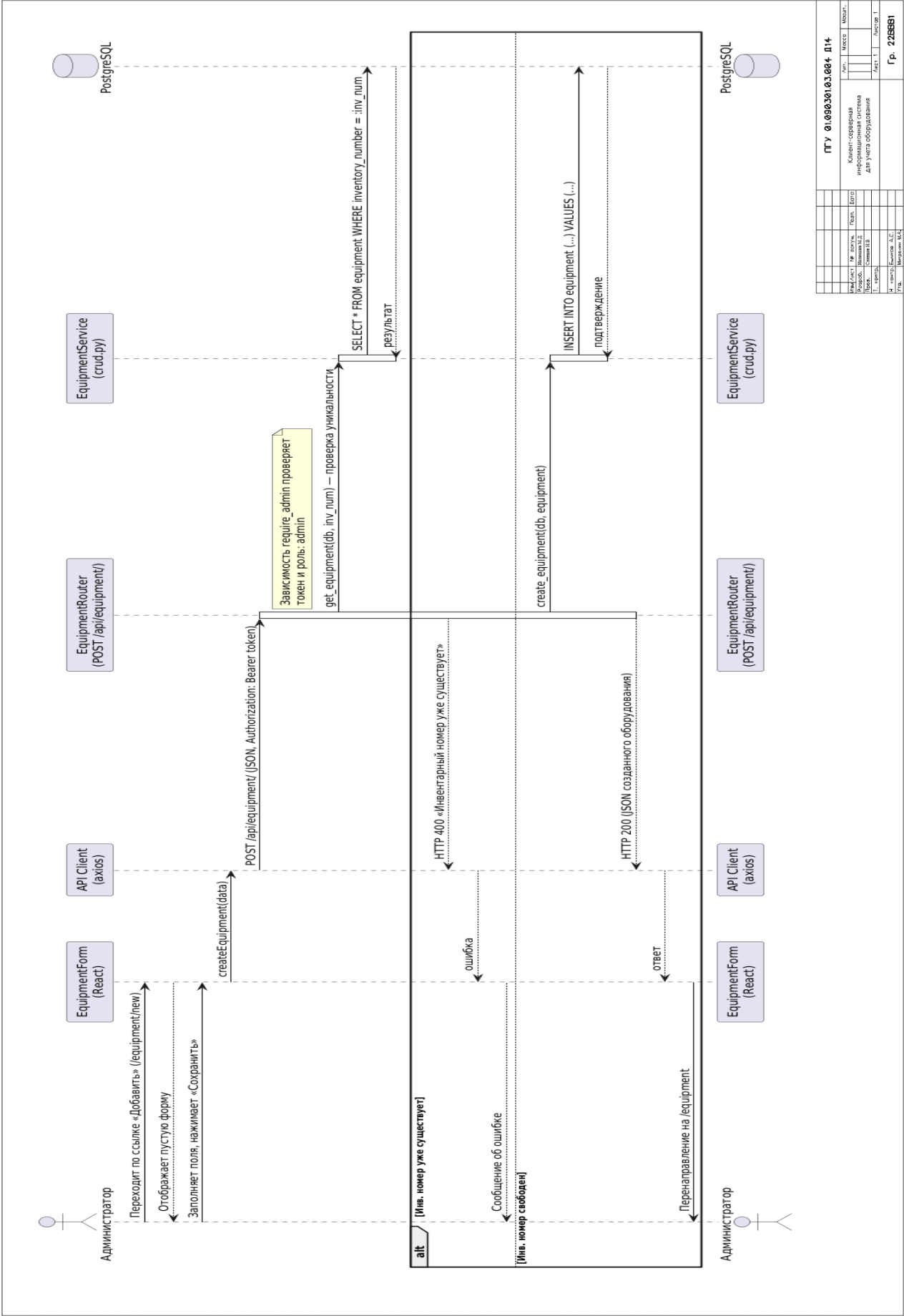
ПГУ 1.090301.03.002 ПЗ									
Изм.	Лист	№ докум.	Подпись	Дата	Изм.	Лист	№ докум.	Подпись	Дата
Клиент-серверная информационная система для учета оборудования					Гр. 220801				

Приложение А.2 – UML-диаграмма классов



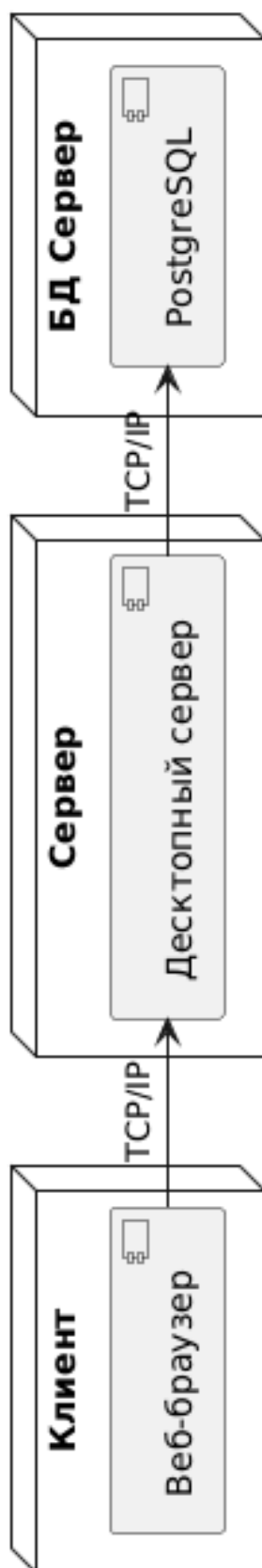
ПГУ 01.090301.03.003 П11			
Имя	Место	Дата	Лист
Иванов И.И.	Место	2023.12.15	1
Гр. 225881			

Приложение А.3 – UML-диаграмма последовательности



ПГУ 1.090301.03.001 ПЗ									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия

Приложение А.4 – UML-диаграмма развертывания

[illegible]

Изм.	Лист	№ докум.	Подпись	Дата

Приложение Б

(обязательное)

Клиент-серверная информационная система для учета оборудования

Листинг программы

					ПГУ 1.090301.03.001 ПЗ	Лист
						59
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «main.py»

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from .database import engine, Base
from .routers import equipment, users, movements, reports, requests

Base.metadata.create_all(bind=engine)

app = FastAPI(title="Equipment Accounting System")

origins = ["http://localhost:5173"]
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(equipment.router)
app.include_router(users.router)
app.include_router(movements.router)
app.include_router(reports.router)
app.include_router(requests.router)
```

Файл «auth.py»

```
from datetime import datetime, timedelta
from jose import JWTError, jwt
from passlib.context import CryptContext
from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
from sqlalchemy.orm import Session
from . import models, database

SECRET_KEY = "секретный-ключ-только-для-разработки"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 60

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="api/auth/login")

def verify_password(plain_password, hashed_password):
    return pwd_context.verify(plain_password, hashed_password)

def get_password_hash(password):
    return pwd_context.hash(password)

def create_access_token(data: dict):
```

					ПГУ 1.090301.03.001 ПЗ	Лист
						60
Изм.	Лист	№ докум.	Подпись	Дата		

```

        to_encode = data.copy()
        expire = datetime.utcnow() +
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
        to_encode.update({"exp": expire})
        return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

def get_current_user(token: str = Depends(oauth2_scheme), db:
Session = Depends(database.get_db)):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Не удалось проверить учётные данные",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
        payload = jwt.decode(token, SECRET_KEY,
algorithms=[ALGORITHM])
        username: str = payload.get("sub")
        if username is None:
            raise credentials_exception
    except JWTError:
        raise credentials_exception
    user = db.query(models.User).filter(models.User.username ==
username).first()
    if user is None or user.is_active != "active":
        raise credentials_exception
    return user

def require_admin(current_user: models.User =
Depends(get_current_user)):
    if current_user.role != models.UserRole.admin:
        raise HTTPException(status_code=403, detail="Требуется права
администратора")
    return current_user

```

Файл «database.py»

```

from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from sqlalchemy.orm import Session

SQLALCHEMY_DATABASE_URL =
"postgresql://postgres:admin@localhost/equipment_db"

engine = create_engine(SQLALCHEMY_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=engine)

Base = declarative_base()
def get_db():
    db = SessionLocal()
    try:

```

```

        yield db
    finally:
        db.close()

```

Файл «models.py»

```

from sqlalchemy import Column, Integer, String, Float, Date,
DateTime, ForeignKey, Enum as SQLAlchemyEnum
from sqlalchemy.orm import relationship
from datetime import datetime
import enum
from .database import Base

class UserRole(str, enum.Enum):
    admin = "admin"
    user = "user"

class EquipmentStatus(str, enum.Enum):
    available = "в наличии"
    issued = "выдан"
    repair = "в ремонте"
    written_off = "списан"

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    username = Column(String, unique=True, index=True,
nullable=False)
    full_name = Column(String, nullable=False)
    hashed_password = Column(String, nullable=False)
    role = Column(SQLAlchemyEnum(UserRole), default=UserRole.user)
    is_active = Column(String, default="active") # "active" /
"blocked"

    movements = relationship("Movement", back_populates="user")

class Equipment(Base):
    __tablename__ = "equipment"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, nullable=False)
    inventory_number = Column(String, unique=True, index=True,
nullable=False)
    type = Column(String, nullable=False) # компьютер, принтер,
etc.
    serial_number = Column(String)
    purchase_date = Column(Date)
    price = Column(Float)
    location = Column(String, default="Склад")
    status = Column(SQLAlchemyEnum(EquipmentStatus),
default=EquipmentStatus.available)

```

```

    notes = Column(String)
    written_off_date = Column(DateTime, nullable=True)

    movements = relationship("Movement", back_populates="equipment")

class Movement(Base):
    __tablename__ = "movements"

    id = Column(Integer, primary_key=True, index=True)
    equipment_id = Column(Integer, ForeignKey("equipment.id"),
    nullable=False)
    user_id = Column(Integer, ForeignKey("users.id"),
    nullable=False)
    issue_date = Column(DateTime, default=datetime.utcnow)
    expected_return_date = Column(Date)
    actual_return_date = Column(DateTime)
    notes = Column(String)

    equipment = relationship("Equipment",
    back_populates="movements")
    user = relationship("User", back_populates="movements")

class RequestStatus(str, enum.Enum):
    pending = "ожидает"
    approved = "одобрена"
    rejected = "отклонена"

class Request(Base):
    __tablename__ = "requests"

    id = Column(Integer, primary_key=True, index=True)
    equipment_id = Column(Integer, ForeignKey("equipment.id"),
    nullable=False)
    user_id = Column(Integer, ForeignKey("users.id"),
    nullable=False)
    status = Column(SQLEnum(RequestStatus),
    default=RequestStatus.pending)
    request_date = Column(DateTime, default=datetime.utcnow)
    expected_return_date = Column(Date)
    notes = Column(String)
    processed_by = Column(Integer, ForeignKey("users.id"))
    processed_date = Column(DateTime)

    equipment = relationship("Equipment")
    user = relationship("User", foreign_keys=[user_id])
    admin = relationship("User", foreign_keys=[processed_by])

```

Файл «schemas.py»

```

from pydantic import BaseModel, Field
from typing import Optional, List
from datetime import date, datetime

```

```

from .models import EquipmentStatus, UserRole

# ----- Auth -----
class Token(BaseModel):
    access_token: str
    token_type: str

class TokenData(BaseModel):
    username: Optional[str] = None

# ----- User -----
class UserBase(BaseModel):
    username: str
    full_name: str
    role: UserRole = UserRole.user

class UserCreate(UserBase):
    password: str

class UserUpdate(BaseModel):
    full_name: Optional[str] = None
    role: Optional[UserRole] = None
    is_active: Optional[str] = None # "active" / "blocked"

class UserOut(UserBase):
    id: int
    is_active: str
    class Config:
        from_attributes = True

# ----- Equipment -----
class EquipmentBase(BaseModel):
    name: str
    inventory_number: str
    type: str
    serial_number: Optional[str] = None
    purchase_date: Optional[date] = None
    price: Optional[float] = None
    location: str = "Склад"
    status: EquipmentStatus = EquipmentStatus.available
    notes: Optional[str] = None

class EquipmentCreate(EquipmentBase):
    pass

class EquipmentUpdate(BaseModel):
    name: Optional[str] = None
    type: Optional[str] = None
    serial_number: Optional[str] = None
    purchase_date: Optional[date] = None
    price: Optional[float] = None
    location: Optional[str] = None
    status: Optional[EquipmentStatus] = None

```

```

        notes: Optional[str] = None

class EquipmentOut(EquipmentBase):
    id: int
    class Config:
        from_attributes = True

# ----- Movement -----
class MovementBase(BaseModel):
    equipment_id: int
    user_id: int
    issue_date: Optional[datetime] = None
    expected_return_date: Optional[date] = None
    notes: Optional[str] = None

class MovementCreate(MovementBase):
    pass

class MovementReturn(BaseModel):
    actual_return_date: Optional[datetime] = None

class MovementOut(MovementBase):
    id: int
    actual_return_date: Optional[datetime] = None
    equipment: EquipmentOut
    user: UserOut
    class Config:
        from_attributes = True

# ----- Reports -----
class EquipmentFilter(BaseModel):
    type: Optional[str] = None
    status: Optional[EquipmentStatus] = None
    location: Optional[str] = None
    min_price: Optional[float] = None
    max_price: Optional[float] = None

# ----- Request -----
class RequestCreate(BaseModel):
    equipment_id: int
    expected_return_date: Optional[date] = None
    notes: Optional[str] = None

class RequestOut(BaseModel):
    id: int
    equipment: EquipmentOut
    user: UserOut
    status: str # используем строку
    request_date: datetime
    expected_return_date: Optional[date] = None
    notes: Optional[str] = None
    processed_by: Optional[int] = None
    processed_date: Optional[datetime] = None

```

```
class Config:
    from_attributes = True
```

Файл «crud.py»

```
from sqlalchemy.orm import Session
from sqlalchemy import or_
from . import models, schemas
from .auth import get_password_hash
from datetime import datetime
from typing import Optional, List

# ----- User CRUD -----
def get_user(db: Session, user_id: int):
    return db.query(models.User).filter(models.User.id ==
user_id).first()

def get_user_by_username(db: Session, username: str):
    return db.query(models.User).filter(models.User.username ==
username).first()

def get_users(db: Session, skip: int = 0, limit: int = 100):
    return db.query(models.User).offset(skip).limit(limit).all()

def create_user(db: Session, user: schemas.UserCreate):
    hashed_pwd = get_password_hash(user.password)
    db_user = models.User(
        username=user.username,
        full_name=user.full_name,
        hashed_password=hashed_pwd,
        role=user.role
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

def update_user(db: Session, user_id: int, updates:
schemas.UserUpdate):
    user = get_user(db, user_id)
    if user:
        for key, value in updates.dict(exclude_unset=True).items():
            setattr(user, key, value)
        db.commit()
        db.refresh(user)
    return user

# ----- Equipment CRUD -----
def get_equipment(db: Session, equipment_id: int):
    return db.query(models.Equipment).filter(models.Equipment.id ==
equipment_id).first()
```

```

def get equipments(db: Session, skip: int = 0, limit: int = 100,
search: str = None, **filters):
    query = db.query(models.Equipment)
    print(f"DEBUG search={search}")
    if search:
        term = f"%{search}%"
        query = query.filter(
            or_(
                models.Equipment.name.ilike(term),
                models.Equipment.inventory_number.ilike(term),
                models.Equipment.serial_number.ilike(term),
                models.Equipment.location.ilike(term),
                models.Equipment.notes.ilike(term),
            )
        )

        if filters.get("type"):
            query = query.filter(models.Equipment.type ==
filters["type"])
        if filters.get("status"):
            query = query.filter(models.Equipment.status ==
filters["status"])
        if filters.get("location"):
            query =
query.filter(models.Equipment.location.ilike(f"%{filters['location']}
%"))
        if filters.get("min_price") is not None:
            query = query.filter(models.Equipment.price >=
filters["min_price"])
        if filters.get("max_price") is not None:
            query = query.filter(models.Equipment.price <=
filters["max_price"])

    return query.offset(skip).limit(limit).all()

def create_equipment(db: Session, equipment:
schemas.EquipmentCreate):
    db_equip = models.Equipment(**equipment.dict())
    db.add(db_equip)
    db.commit()
    db.refresh(db_equip)
    return db_equip

def update_equipment(db: Session, equipment_id: int, updates:
schemas.EquipmentUpdate):
    equip = get_equipment(db, equipment_id)
    if equip:
        for key, value in updates.dict(exclude_unset=True).items():
            setattr(equip, key, value)
        db.commit()
        db.refresh(equip)
    return equip

```

```

def delete_equipment(db: Session, equipment_id: int):
    equip = get_equipment(db, equipment_id)
    if equip:
        equip.status = models.EquipmentStatus.written_off
        equip.written_off_date = datetime.utcnow() # фиксируем дату
        db.commit()
    return equip

# ----- Movement CRUD -----
def create_movement(db: Session, movement: schemas.MovementCreate):
    db_mov = models.Movement(**movement.dict())
    equip = db.query(models.Equipment).filter(models.Equipment.id ==
movement.equipment_id).first()
    if equip:
        equip.status = models.EquipmentStatus.issued
    db.add(db_mov)
    db.commit()
    db.refresh(db_mov)
    return db_mov

def return_equipment(db: Session, movement_id: int,
actual_return_date: datetime):
    mov = db.query(models.Movement).filter(models.Movement.id ==
movement_id).first()
    if mov and mov.actual_return_date is None:
        mov.actual_return_date = actual_return_date
        equip =
db.query(models.Equipment).filter(models.Equipment.id ==
mov.equipment_id).first()
        if equip:
            equip.status = models.EquipmentStatus.available
            db.commit()
            db.refresh(mov)
    return mov

def get_movements_for_equipment(db: Session, equipment_id: int):
    return
db.query(models.Movement).filter(models.Movement.equipment_id ==
equipment_id).all()

def hard_delete_equipment(db: Session, equipment_id: int):
    equip = db.query(models.Equipment).filter(models.Equipment.id ==
equipment_id).first()
    if equip:
        # Удалим связанные перемещения (или можно запретить, если
есть перемещения)

    db.query(models.Movement).filter(models.Movement.equipment_id ==
equipment_id).delete()
        db.delete(equip)
        db.commit()
        return True
    return False

```

```

from .models import Request, RequestStatus

def create_request(db: Session, req: schemas.RequestCreate, user_id:
int):
    db_req = Request(
        equipment_id=req.equipment_id,
        user_id=user_id,
        expected_return_date=req.expected_return_date,
        notes=req.notes,
        status=RequestStatus.pending
    )
    db.add(db_req)
    db.commit()
    db.refresh(db_req)
    return db_req

def get_requests(db: Session, user_id: Optional[int] = None, status:
Optional[str] = None):
    query = db.query(Request)
    if user_id:
        query = query.filter(Request.user_id == user_id)
    if status:
        query = query.filter(Request.status == status)
    return query.order_by(Request.request_date.desc()).all()

def approve_request(db: Session, request_id: int, admin_user_id:
int):
    req = db.query(Request).filter(Request.id == request_id).first()
    if not req or req.status != RequestStatus.pending:
        return None
    # создаём движение
    movement = models.Movement(
        equipment_id=req.equipment_id,
        user_id=req.user_id,
        expected_return_date=req.expected_return_date,
        notes=f"По заявке #{req.id}: {req.notes or ''}"
    )
    db.add(movement)
    # меняем статус оборудования
    equip = db.query(models.Equipment).filter(models.Equipment.id ==
req.equipment_id).first()
    if equip:
        equip.status = models.EquipmentStatus.issued
    # обновляем заявку
    req.status = RequestStatus.approved
    req.processed_by = admin_user_id
    req.processed_date = datetime.utcnow()
    db.commit()
    db.refresh(req)
    return req

```

```

def reject_request(db: Session, request_id: int, admin_user_id:
int):
    req = db.query(Request).filter(Request.id == request_id).first()
    if not req or req.status != RequestStatus.pending:
        return None
    req.status = RequestStatus.rejected
    req.processed_by = admin_user_id
    req.processed_date = datetime.utcnow()
    db.commit()
    db.refresh(req)
    return req

```

Файл «users.py»

```

from fastapi import APIRouter, Depends, HTTPException, status
from fastapi.security import OAuth2PasswordRequestForm
from sqlalchemy.orm import Session
from typing import List
from .. import schemas, crud, auth, database, models

router = APIRouter(
    prefix="/api",
    tags=["auth"] # только auth, чтобы не дублировалось
)

# Аутентификация и получение токена
@router.post("/auth/login", response_model=schemas.Token)
def login(
    form_data: OAuth2PasswordRequestForm = Depends(),
    db: Session = Depends(database.get_db)
):
    user = crud.get_user_by_username(db,
username=form_data.username)
    if not user or not auth.verify_password(form_data.password,
user.hashed_password):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Неверное имя пользователя или пароль",
            headers={"WWW-Authenticate": "Bearer"},
        )
    if user.is_active != "active":
        raise HTTPException(status_code=400, detail="Пользователь
заблокирован")
    access_token = auth.create_access_token(data={"sub":
user.username})
    return {"access_token": access_token, "token_type": "bearer"}

# Создание пользователя (только для администраторов)
@router.post("/users/", response_model=schemas.UserOut)
def create_user(
    user: schemas.UserCreate,
    db: Session = Depends(database.get_db),

```

```

        admin: models.User = Depends(auth.require_admin)
    ):
        db_user = crud.get_user_by_username(db, username=user.username)
        if db_user:
            raise HTTPException(status_code=400, detail="Имя
пользователя уже занято")
        return crud.create_user(db, user)

# Получение списка пользователей (только администратор)
@router.get("/users/", response_model=List[schemas.UserOut])
def read_users(
    skip: int = 0,
    limit: int = 100,
    db: Session = Depends(database.get_db),
    admin: models.User = Depends(auth.require_admin)
):
    return crud.get_users(db, skip=skip, limit=limit)

# Обновление пользователя (администратор)
@router.put("/users/{user_id}", response_model=schemas.UserOut)
def update_user(
    user_id: int,
    updates: schemas.UserUpdate,
    db: Session = Depends(database.get_db),
    admin: models.User = Depends(auth.require_admin)
):
    user = crud.update_user(db, user_id, updates)
    if not user:
        raise HTTPException(status_code=404, detail="Пользователь не
найден")
    return user

@router.get("/users/me", response_model=schemas.UserOut)
def read_current_user(
    current_user: models.User = Depends(auth.get_current_user),
    db: Session = Depends(database.get_db)
):
    return current_user

```

Файл «requests.py»

```

from fastapi import APIRouter, Depends, HTTPException, Query
from sqlalchemy.orm import Session
from typing import List, Optional
from .. import schemas, crud, auth, database, models

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

```

```

@router.post("/", response_model=schemas.RequestOut)
def create_request(
    req: schemas.RequestCreate,
    db: Session = Depends(database.get_db),
    current_user: models.User = Depends(auth.get_current_user)
):
    # Проверим, что оборудование доступно
    equip = crud.get_equipment(db, req.equipment_id)
    if not equip:
        raise HTTPException(status_code=404, detail="Оборудование не
найдено")
    if equip.status != "в наличии":
        raise HTTPException(status_code=400, detail="Оборудование
недоступно")
    return crud.create_request(db, req, current_user.id)

@router.get("/", response_model=List[schemas.RequestOut])
def read_requests(
    status: Optional[str] = None,
    db: Session = Depends(database.get_db),
    current_user: models.User = Depends(auth.get_current_user)
):
    if current_user.role == models.UserRole.admin:
        # админ видит все заявки
        return crud.get_requests(db, status=status)
    else:
        # обычный пользователь видит только свои
        return crud.get_requests(db, user_id=current_user.id,
status=status)

@router.put("/{request_id}/approve")
def approve_request(
    request_id: int,
    db: Session = Depends(database.get_db),
    admin: models.User = Depends(auth.require_admin)
):
    approved = crud.approve_request(db, request_id, admin.id)
    if not approved:
        raise HTTPException(status_code=404, detail="Заявка не
найдена или уже обработана")
    return {"message": "Заявка одобрена, оборудование выдано"}

@router.put("/{request_id}/reject")
def reject_request(
    request_id: int,
    db: Session = Depends(database.get_db),
    admin: models.User = Depends(auth.require_admin)
):
    rejected = crud.reject_request(db, request_id, admin.id)
    if not rejected:
        raise HTTPException(status_code=404, detail="Заявка не
найдена или уже обработана")
    return {"message": "Заявка отклонена"}

```

Файл «reports.py»

```
from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session
from .. import crud, models, database, auth
from typing import Optional
import pandas as pd
import io
from fastapi.responses import StreamingResponse
from datetime import date
from openpyxl import Workbook
from openpyxl.utils import get_column_letter

router = APIRouter(
    prefix="/api/reports",
    tags=["reports"],
    dependencies=[Depends(auth.require_admin)]
)

def auto_width(ws, df):
    """Безопасный автоподбор ширины столбцов"""
    for col_idx, col_name in enumerate(df.columns, 1):
        max_len = len(str(col_name))
        for val in df[col_name]:
            if val is not None:
                max_len = max(max_len, len(str(val)))
        adjusted_width = min(max_len + 2, 60)
        ws.column_dimensions[get_column_letter(col_idx)].width = adjusted_width

def generate_excel(df, filename, sheet_name="Отчёт"):
    output = io.BytesIO()
    wb = Workbook()
    ws = wb.active
    ws.title = sheet_name

    # Заголовки
    for col_idx, col_name in enumerate(df.columns, 1):
        ws.cell(row=1, column=col_idx, value=col_name)

    # Данные
    for row_idx, row in enumerate(df.itertuples(index=False), 2):
        for col_idx, value in enumerate(row, 1):
            ws.cell(row=row_idx, column=col_idx, value=value)

    auto_width(ws, df)
    wb.save(output)
    output.seek(0)
    return output

@router.get("/equipment-state")
def equipment_state_report(
```

					ПГУ 1.090301.03.001 ПЗ	Лист
						73
Изм.	Лист	№ докум.	Подпись	Дата		

```

type: Optional[str] = None,
status: Optional[str] = None,
location: Optional[str] = None,
min_price: Optional[float] = None,
max_price: Optional[float] = None,
search: Optional[str] = None,
format: str = "xlsx",
db: Session = Depends(database.get_db)
):
    filters = {
        "type": type, "status": status, "location": location,
        "min_price": min_price, "max_price": max_price
    }
    equipment = crud.get equipments(db, skip=0, limit=10000,
search=search, **filters)
    df = pd.DataFrame([
        "Инв.номер": e.inventory_number,
        "Название": e.name,
        "Тип": e.type,
        "Серийный номер": e.serial_number,
        "Дата покупки": e.purchase_date,
        "Цена": e.price,
        "Статус": e.status,
        "Местоположение": e.location,
        "Примечание": e.notes
    ] for e in equipment])

    if format == "csv":
        stream = io.StringIO()
        df.to_csv(stream, index=False, sep=';', encoding='utf-8-
sig')
        response = StreamingResponse(iter([stream.getvalue()] ),
media_type="text/csv; charset=utf-8-sig")
        response.headers["Content-Disposition"] = "attachment;
filename=equipment_state.csv"
        return response
    else:
        output = generate_excel(df, "equipment_state.xlsx",
"Состояние")
        response = StreamingResponse(output,
media_type="application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet")
        response.headers["Content-Disposition"] = "attachment;
filename=equipment_state.xlsx"
        return response

@router.get("/issued-equipment")
def issued_equipment_report(
    date: Optional[date] = None,
    format: str = "xlsx",
    db: Session = Depends(database.get_db)
):

```

```

        query =
db.query(models.Movement).filter(models.Movement.actual_return_date
== None)
        if date:
            query = query.filter(models.Movement.issue_date >= date)
        movements = query.all()
        df = pd.DataFrame([{
            "Оборудование": m.equipment.inventory_number + " " +
m.equipment.name,
            "Сотрудник": m.user.full_name,
            "Дата выдачи": m.issue_date,
            "Ожидаемый возврат": m.expected_return_date,
            "Примечание": m.notes
        } for m in movements])

        if format == "csv":
            stream = io.StringIO()
            df.to_csv(stream, index=False, sep=';', encoding='utf-8-
sig')
            response = StreamingResponse(iter([stream.getvalue()]),
media_type="text/csv; charset=utf-8-sig")
            response.headers["Content-Disposition"] = "attachment;
filename=issued_equipments.csv"
            return response
        else:
            output = generate_excel(df, "issued_equipments.xlsx",
"Выданное")
            response = StreamingResponse(output,
media_type="application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet")
            response.headers["Content-Disposition"] = "attachment;
filename=issued_equipments.xlsx"
            return response

@router.get("/written-off-equipment")
def written_off_equipment_report(
    start_date: Optional[date] = None,
    end_date: Optional[date] = None,
    format: str = "xlsx",
    db: Session = Depends(database.get_db)
):
    query =
db.query(models.Equipment).filter(models.Equipment.status ==
models.EquipmentStatus.written_off)
        if start_date:
            query = query.filter(models.Equipment.written_off_date >=
start_date)
        if end_date:
            query = query.filter(models.Equipment.written_off_date <=
end_date)
        equipment = query.all()
        df = pd.DataFrame([{
            "Инв.номер": e.inventory_number,

```

```

        "Название": e.name,
        "Тип": e.type,
        "Серийный номер": e.serial_number,
        "Дата покупки": e.purchase_date,
        "Цена": e.price,
        "Дата списания": e.written_off_date,
        "Местоположение": e.location,
        "Примечание": e.notes
    } for e in equipment])

    if format == "csv":
        stream = io.StringIO()
        df.to_csv(stream, index=False, sep=';', encoding='utf-8-
sig')
        response = StreamingResponse(iter([stream.getvalue()]),
media_type="text/csv; charset=utf-8-sig")
        response.headers["Content-Disposition"] = "attachment;
filename=written_off_equipment.csv"
        return response
    else:
        output = generate_excel(df, "written_off_equipment.xlsx",
"Списанное")
        response = StreamingResponse(output,
media_type="application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet")
        response.headers["Content-Disposition"] = "attachment;
filename=written_off_equipment.xlsx"
        return response

```

Файл «movements.py»

```

from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from typing import List
from datetime import datetime as dt
from .. import schemas, crud, database, auth, models

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

@router.post("/issue", response_model=schemas.MovementOut)
def issue_equipment(
    movement: schemas.MovementCreate,
    db: Session = Depends(database.get_db),
    current_user = Depends(auth.get_current_user)
):
    equip = crud.get_equipment(db, movement.equipment_id)
    if not equip:
        raise HTTPException(status_code=404, detail="Оборудование не
найдено")

```

```

        if equip.status != "в наличии":
            raise HTTPException(status_code=400, detail="Оборудование
недоступно для выдачи")
        user = crud.get_user(db, movement.user_id)
        if not user:
            raise HTTPException(status_code=404, detail="Пользователь не
найден")
        return crud.create_movement(db, movement)

@router.post("/return/{movement_id}",
response_model=schemas.MovementOut)
def return_equipment(
    movement_id: int,
    return_info: schemas.MovementReturn = None,
    db: Session = Depends(database.get_db),
    current_user: models.User = Depends(auth.get_current_user)
):
    # Получаем запись о выдаче
    mov = db.query(models.Movement).filter(models.Movement.id ==
movement_id).first()
    if not mov:
        raise HTTPException(status_code=404, detail="Запись о выдаче
не найдена")
    if mov.actual_return_date is not None:
        raise HTTPException(status_code=400, detail="Оборудование
уже возвращено")

    # Проверка прав: только администратор или тот, кому выдано
    if current_user.role != models.UserRole.admin and mov.user_id !=
current_user.id:
        raise HTTPException(status_code=403, detail="Вы не можете
вернуть чужое оборудование")

    actual_date = return_info.actual_return_date if return_info and
return_info.actual_return_date else dt.utcnow()
    updated_mov = crud.return_equipment(db, movement_id,
actual_date)
    return updated_mov

@router.get("/equipment/{equipment_id}",
response_model=List[schemas.MovementOut])
def get_movements(
    equipment_id: int,
    db: Session = Depends(database.get_db),
    current_user = Depends(auth.get_current_user)
):
    return crud.get_movements_for_equipment(db, equipment_id)

```

Файл «equipment.py»

```

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

```

					ПГУ 1.090301.03.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

```

from .. import schemas, crud, auth, database
from ..auth import require_admin

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

@router.get("/", response_model=List[schemas.EquipmentOut])
def read equipments(
    skip: int = 0,
    limit: int = 100,
    search: Optional[str] = None,
    type: Optional[str] = None,
    status: Optional[schemas.EquipmentStatus] = None,
    location: Optional[str] = None,
    min_price: Optional[float] = None,
    max_price: Optional[float] = None,
    db: Session = Depends(database.get_db),
    current_user = Depends(auth.get_current_user)
):
    filters = {
        "type": type,
        "status": status,
        "location": location,
        "min_price": min_price,
        "max_price": max_price
    }
    return crud.get equipments(db, skip=skip, limit=limit,
search=search, **filters)

@router.post("/", response_model=schemas.EquipmentOut)
def create_equipment(
    equipment: schemas.EquipmentCreate,
    db: Session = Depends(database.get_db),
    admin = Depends(require_admin)
):
    db_equip = db.query(crud.models.Equipment).filter(
        crud.models.Equipment.inventory_number ==
equipment.inventory_number
    ).first()
    if db_equip:
        raise HTTPException(status_code=400, detail="Инвентарный
номер уже существует")
    return crud.create_equipment(db, equipment)

@router.put("/{equipment_id}", response_model=schemas.EquipmentOut)
def update_equipment(
    equipment_id: int,
    updates: schemas.EquipmentUpdate,
    db: Session = Depends(database.get_db),
    admin = Depends(require_admin)
):

```

```

    equip = crud.update_equipment(db, equipment_id, updates)
    if not equip:
        raise HTTPException(status_code=404, detail="Оборудование не
найдено")
    return equip

@router.delete("/{equipment_id}")
def delete_equipment(
    equipment_id: int,
    db: Session = Depends(database.get_db),
    admin = Depends(require_admin)
):
    equip = crud.delete_equipment(db, equipment_id)
    if not equip:
        raise HTTPException(status_code=404, detail="Оборудование не
найдено")
    return {"message": "Оборудование списано"}

@router.delete("/{equipment_id}/hard")
def hard_delete_equipment(
    equipment_id: int,
    db: Session = Depends(database.get_db),
    admin = Depends(require_admin)
):
    success = crud.hard_delete_equipment(db, equipment_id)
    if not success:
        raise HTTPException(status_code=404, detail="Оборудование не
найдено")
    return {"message": "Оборудование полностью удалено"}

```

Файл «index.html»

```

<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>Учёт оборудования</title>
    <!-- Bootstrap 5 CSS -->
    <link
href="https://cdn.jsdelivrivr.net/npm/bootstrap@5.3.0/dist/css/bootstra
p.min.css"
    rel="stylesheet"
    />
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
    <!-- Bootstrap JS (для мобильного меню) -->

```

```

    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.
bundle.min.js"></script>
    </body>
</html>

```

Файл «package.json»

```

{
  "name": "client",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
    "@emotion/react": "^11.14.0",
    "@emotion/styled": "^11.14.1",
    "@mui/material": "^9.0.0",
    "axios": "^1.16.0",
    "react": "^19.2.5",
    "react-dom": "^19.2.5",
    "react-router-dom": "^7.15.0"
  },
  "devDependencies": {
    "@eslint/js": "^10.0.1",
    "@types/react": "^19.2.14",
    "@types/react-dom": "^19.2.3",
    "@vitejs/plugin-react": "^6.0.1",
    "eslint": "^10.2.1",
    "eslint-plugin-react-hooks": "^7.1.1",
    "eslint-plugin-react-refresh": "^0.5.2",
    "globals": "^17.5.0",
    "vite": "^8.0.10"
  }
}

```

Файл main.jsx»

```

import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)

```

					ПГУ 1.090301.03.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «index.css»

```
:root {
  --text: #6b6375;
  --text-h: #08060d;
  --bg: #fff;
  --border: #e5e4e7;
  --code-bg: #f4f3ec;
  --accent: #aa3bff;
  --accent-bg: rgba(170, 59, 255, 0.1);
  --accent-border: rgba(170, 59, 255, 0.5);
  --social-bg: rgba(244, 243, 236, 0.5);
  --shadow:
    rgba(0, 0, 0, 0.1) 0 10px 15px -3px, rgba(0, 0, 0, 0.05) 0 4px
    6px -2px;

  --sans: system-ui, 'Segoe UI', Roboto, sans-serif;
  --heading: system-ui, 'Segoe UI', Roboto, sans-serif;
  --mono: ui-monospace, Consolas, monospace;

  font: 18px/145% var(--sans);
  letter-spacing: 0.18px;
  color-scheme: light dark;
  color: var(--text);
  background: var(--bg);
  font-synthesis: none;
  text-rendering: optimizeLegibility;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;

  @media (max-width: 1024px) {
    font-size: 16px;
  }
}

@media (prefers-color-scheme: dark) {
  :root {
    --text: #9ca3af;
    --text-h: #f3f4f6;
    --bg: #16171d;
    --border: #2e303a;
    --code-bg: #1f2028;
    --accent: #c084fc;
    --accent-bg: rgba(192, 132, 252, 0.15);
    --accent-border: rgba(192, 132, 252, 0.5);
    --social-bg: rgba(47, 48, 58, 0.5);
    --shadow:
      rgba(0, 0, 0, 0.4) 0 10px 15px -3px, rgba(0, 0, 0, 0.25) 0 4px
      6px -2px;

    #social .button-icon {
```

					ПГУ 1.090301.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		81

```

        filter: invert(1) brightness(2);
    }
}

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

html, body, #root {
    width: 100%;
    height: 100%;
}

body {
    margin: 0;
    background-color: white;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

#root {
    width: 100%;
    max-width: 100%;
    margin: 0 auto;
    text-align: center;
    border-inline: 1px solid var(--border);
    min-height: 100svh;
    display: flex;
    flex-direction: column;
    box-sizing: border-box;
}

h1,
h2 {
    font-family: var(--heading);
    font-weight: 500;
    color: var(--text-h);
}

h1 {
    font-size: 56px;
    letter-spacing: -1.68px;
    margin: 32px 0;
    @media (max-width: 1024px) {
        font-size: 36px;
        margin: 20px 0;
    }
}

h2 {
    font-size: 24px;
    line-height: 118%;
    letter-spacing: -0.24px;

```

```

margin: 0 0 8px;
@media (max-width: 1024px) {
  font-size: 20px;
}
}
p {
margin: 0;
}

code,
.counter {
font-family: var(--mono);
display: inline-flex;
border-radius: 4px;
color: var(--text-h);
}

code {
font-size: 15px;
line-height: 135%;
padding: 4px 8px;
background: var(--code-bg);
}

```

Файл «App.jsx»

```

import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom';
import { useState } from 'react';
import Login from './components/Login';
import Navbar from './components/Navbar';
import EquipmentList from './components/EquipmentList';
import EquipmentForm from './components/EquipmentForm';
import UsersManagement from './components/UsersManagement';
import MovementsHistory from './components/MovementsHistory';
import IssueEquipment from './components/IssueEquipment';
import RequestEquipment from './components/RequestEquipment';
import RequestsAdmin from './components/RequestsAdmin';
import Reports from './components/Reports';
import { getMe } from './api';

function App() {
  const [token, setToken] =
useState(localStorage.getItem('access_token'));
  const [role, setRole] = useState(localStorage.getItem('user_role')
|| 'user');

  const handleLogin = async (newToken) => {
    localStorage.setItem('access_token', newToken);
    try {
      const res = await getMe();
      const role = res.data.role;

```

```

        localStorage.setItem('user_role', role);
        setRole(role);
    } catch (e) {
        setRole('user');
    }
    setToken(newToken);
};

const handleLogout = () => {
    localStorage.removeItem('access_token');
    localStorage.removeItem('user_role');
    setToken(null);
    setRole('user');
};

return (
    <BrowserRouter>
        {!token ? (
            <Routes>
                <Route path="*" element={<Login onLogin={handleLogin} />}
            />
            </Routes>
        ) : (
            <>
                <Navbar onLogout={handleLogout} role={role} />
                <div className="container-fluid mt-4">
                    <Routes>
                        <Route path="/equipment" element={<EquipmentList />}
                    />
                        <Route path="/equipment/new" element={<EquipmentForm
                    />} />
                        <Route path="/equipment/:id/edit"
                    element={<EquipmentForm />} />
                        <Route path="/users" element={<UsersManagement />} />
                        <Route path="/movements/:equipmentId"
                    element={<MovementsHistory />} />
                        <Route
                            path="/issue"
                            element={role === 'admin' ? <IssueEquipment /> :
                    <RequestEquipment />}
                        />
                        <Route path="/requests" element={<RequestsAdmin />} />
                        <Route path="/reports" element={<Reports />} />
                        <Route path="*" element={<Navigate to="/equipment" />}
                    />
                    </Routes>
                </div>
            </>
        )}
    </BrowserRouter>
);
}

```

```
export default App;
```

Файл «App.css»

```
.counter {
  font-size: 16px;
  padding: 5px 10px;
  border-radius: 5px;
  color: var(--accent);
  background: var(--accent-bg);
  border: 2px solid transparent;
  transition: border-color 0.3s;
  margin-bottom: 24px;

  &:hover {
    border-color: var(--accent-border);
  }
  &:focus-visible {
    outline: 2px solid var(--accent);
    outline-offset: 2px;
  }
}

.hero {
  position: relative;

  .base,
  .framework,
  .vite {
    inset-inline: 0;
    margin: 0 auto;
  }

  .base {
    width: 170px;
    position: relative;
    z-index: 0;
  }

  .framework,
  .vite {
    position: absolute;
  }

  .framework {
    z-index: 1;
    top: 34px;
    height: 28px;
    transform: perspective(2000px) rotateZ(300deg) rotateX(44deg)
    rotateY(39deg)
    scale(1.4);
  }
}
```

					ПГУ 1.090301.03.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

```

.vite {
  z-index: 0;
  top: 107px;
  height: 26px;
  width: auto;
  transform: perspective(2000px) rotateZ(300deg) rotateX(40deg)
rotateY(39deg)
  scale(0.8);
}
}

#center {
  display: flex;
  flex-direction: column;
  gap: 25px;
  place-content: center;
  place-items: center;
  flex-grow: 1;

  @media (max-width: 1024px) {
    padding: 32px 20px 24px;
    gap: 18px;
  }
}

#next-steps {
  display: flex;
  border-top: 1px solid var(--border);
  text-align: left;

  & > div {
    flex: 1 1 0;
    padding: 32px;
    @media (max-width: 1024px) {
      padding: 24px 20px;
    }
  }

  .icon {
    margin-bottom: 16px;
    width: 22px;
    height: 22px;
  }

  @media (max-width: 1024px) {
    flex-direction: column;
    text-align: center;
  }
}

#docs {
  border-right: 1px solid var(--border);

```

```

@media (max-width: 1024px) {
  border-right: none;
  border-bottom: 1px solid var(--border);
}
}

#next-steps ul {
  list-style: none;
  padding: 0;
  display: flex;
  gap: 8px;
  margin: 32px 0 0;

  .logo {
    height: 18px;
  }

  a {
    color: var(--text-h);
    font-size: 16px;
    border-radius: 6px;
    background: var(--social-bg);
    display: flex;
    padding: 6px 12px;
    align-items: center;
    gap: 8px;
    text-decoration: none;
    transition: box-shadow 0.3s;

    &:hover {
      box-shadow: var(--shadow);
    }

    .button-icon {
      height: 18px;
      width: 18px;
    }
  }
}

@media (max-width: 1024px) {
  margin-top: 20px;
  flex-wrap: wrap;
  justify-content: center;

  li {
    flex: 1 1 calc(50% - 8px);
  }

  a {
    width: 100%;
    justify-content: center;
    box-sizing: border-box;
  }
}

```

```

    }
}

#spacer {
  height: 88px;
  border-top: 1px solid var(--border);
  @media (max-width: 1024px) {
    height: 48px;
  }
}

.ticks {
  position: relative;
  width: 100%;

  &::before,
  &::after {
    content: '';
    position: absolute;
    top: -4.5px;
    border: 5px solid transparent;
  }

  &::before {
    left: 0;
    border-left-color: var(--border);
  }
  &::after {
    right: 0;
    border-right-color: var(--border);
  }
}

```

Файл «index.js»

```

import axios from 'axios';

const API = axios.create({
  baseURL: 'http://localhost:8000/api',
});

API.interceptors.request.use(config => {
  const token = localStorage.getItem('access_token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Auth
export const login = (username, password) => {
  const params = new URLSearchParams();

```

					ПГУ 1.090301.03.001 ПЗ	Лист
						88
Изм.	Лист	№ докум.	Подпись	Дата		

```

    params.append('username', username);
    params.append('password', password);
    return API.post('/auth/login', params, {
      headers: { 'Content-Type': 'application/x-www-form-urlencoded' }
    });
  };

export const getMe = () => API.get('/users/me');

// Equipment
export const getEquipment = (params) => API.get('/equipment/', {
  params });
export const createEquipment = (data) => API.post('/equipment/',
data);
export const updateEquipment = (id, data) =>
API.put(`/equipment/${id}`, data);
export const deleteEquipment = (id) =>
API.delete(`/equipment/${id}`);
export const hardDeleteEquipment = (id) =>
API.delete(`/equipment/${id}/hard`);

// Users
export const getUsers = () => API.get('/users/');
export const createUser = (data) => API.post('/users/', data);
export const updateUser = (id, data) => API.put(`/users/${id}`,
data);

// Movements
export const issueEquipment = (data) => API.post('/movements/issue',
data);
export const returnEquipment = (id, data) =>
API.post(`/movements/return/${id}`, data);
export const getMovementsForEquipment = (id) =>
API.get(`/movements/equipment/${id}`);
export default API;

// Requests
export const createRequest = (data) => API.post('/requests/', data);
export const getRequests = (params) => API.get('/requests/', {
  params });
export const approveRequest = (id) =>
API.put(`/requests/${id}/approve`);
export const rejectRequest = (id) =>
API.put(`/requests/${id}/reject`);

```

Файл «UsersManagement.jsx»

```

import { useEffect, useState } from "react";
import { getUsers, createUser, updateUser } from "../api";

export default function UsersManagement() {
  const [users, setUsers] = useState([]);

```

```

const [form, setForm] = useState({
  username: "",
  full_name: "",
  password: "",
  role: "user",
});

const fetchUsers = async () => {
  try {
    const res = await getUsers();
    setUsers(res.data);
  } catch (err) {
    console.error("Ошибка загрузки пользователей", err);
  }
};

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

const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    await createUser(form);
    setForm({ username: "", full_name: "", password: "", role:
"user" });
    fetchUsers();
  } catch (err) {
    alert("Ошибка создания пользователя: " +
(err.response?.data?.detail || err.message));
  }
};

const handleToggleStatus = async (userId, currentStatus) => {
  const newStatus = currentStatus === "active" ? "blocked" :
"active";
  try {
    await updateUser(userId, { is_active: newStatus });
    fetchUsers();
  } catch (err) {
    alert("Ошибка обновления статуса");
  }
};

return (
  <div>
    <h3>Управление пользователями</h3>

    <form onSubmit={handleSubmit} className="mb-4 p-3 border
rounded">
      <h5>Добавить пользователя</h5>
      <div className="row">
        <div className="col">

```

```

        <input className="form-control" placeholder="Логин"
value={form.username} onChange={(e) => setForm({ ...form, username:
e.target.value })} required />
    </div>
    <div className="col">
        <input className="form-control" placeholder="ФИО"
value={form.full_name} onChange={(e) => setForm({ ...form,
full_name: e.target.value })} required />
    </div>
    <div className="col">
        <input type="password" className="form-control"
placeholder="Пароль" value={form.password} onChange={(e) =>
setForm({ ...form, password: e.target.value })} required />
    </div>
    <div className="col">
        <select className="form-select" value={form.role}
onChange={(e) => setForm({ ...form, role: e.target.value })}>
            <option value="user">Пользователь</option>
            <option value="admin">Администратор</option>
        </select>
    </div>
    <div className="col">
        <button type="submit" className="btn btn-
primary">Создать</button>
    </div>
</div>
</form>

<table className="table">
    <thead>
        <tr>
            <th>ID</th>
            <th>Логин</th>
            <th>ФИО</th>
            <th>Роль</th>
            <th>Статус</th>
            <th>Действия</th>
        </tr>
    </thead>
    <tbody>
        {users.map((u) => (
            <tr key={u.id}>
                <td>{u.id}</td>
                <td>{u.username}</td>
                <td>{u.full_name}</td>
                <td>{u.role === "admin" ? "Администратор" :
"Пользователь"}</td>
                <td>{u.is_active === "active" ? "✔ Активен" : "✗
Заблокирован"}</td>
                <td>
                    {u.role !== "admin" && (
                        <button

```

```

        className={`btn btn-sm ${u.is_active ===
'active' ? 'btn-warning' : 'btn-success'}`}
        onClick={() => handleToggleStatus(u.id,
u.is_active)}
      >
        {u.is_active === 'active' ? 'Заблокировать' :
'Разблокировать'}
      </button>
    )}
  </td>
</tr>
)}}
</tbody>
</table>
</div>
);
}

```

Файл «RequestsAdmin.jsx»

```

import { useEffect, useState } from "react";
import { getRequests, approveRequest, rejectRequest } from "../api";

export default function RequestsAdmin() {
  const [requests, setRequests] = useState([]);

  const fetchRequests = async () => {
    try {
      const res = await getRequests();
      setRequests(res.data);
    } catch (err) {
      console.error("Ошибка загрузки заявок", err);
    }
  };

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

  const handleApprove = async (id) => {
    try {
      await approveRequest(id);
      fetchRequests();
    } catch (err) {
      alert("Ошибка одобрения заявки: " +
(err.response?.data?.detail || err.message));
    }
  };

  const handleReject = async (id) => {
    try {
      await rejectRequest(id);
    }
  };
}

```

```

    fetchRequests();
  } catch (err) {
    alert("Ошибка отклонения заявки: " +
(err.response?.data?.detail || err.message));
  }
};

return (
  <div className="container-fluid mt-4">
    <h3>Заявки на выдачу</h3>
    <table className="table">
      <thead>
        <tr>
          <th>Оборудование</th>
          <th>Заявитель</th>
          <th>Дата заявки</th>
          <th>Ожидаемый возврат</th>
          <th>Статус</th>
          <th>Действия</th>
        </tr>
      </thead>
      <tbody>
        {requests.map((r) => (
          <tr key={r.id}>
            <td>{r.equipment.inventory_number}
{r.equipment.name}</td>
            <td>{r.user.full_name}</td>
            <td>{new Date(r.request_date).toLocaleString()}</td>
            <td>{r.expected_return_date || "-"}</td>
            <td>{r.status}</td>
            <td>
              {r.status === "ожидает" && (
                <>
                  <button
                    className="btn btn-sm btn-success me-1"
                    onClick={() => handleApprove(r.id)}
                  >
                    Одобрить
                  </button>
                  <button
                    className="btn btn-sm btn-danger"
                    onClick={() => handleReject(r.id)}
                  >
                    Отклонить
                  </button>
                </>
              )}
            </td>
          </tr>
        )})}
      </tbody>
    </table>
  </div>

```

```
);
}
```

Файл «RequestEquipment.jsx»

```
< import { useState, useEffect } from "react";
import { getEquipment, createRequest } from "../api";
import { useNavigate } from "react-router-dom";

function formatError(err) {
  const detail = err.response?.data?.detail;
  if (Array.isArray(detail)) {
    return detail.map(d => d.msg).join('; ');
  }
  return detail || err.message;
}

export default function RequestEquipment() {
  const [equipments, setEquipments] = useState([]);
  const [form, setForm] = useState({
    equipment_id: "",
    expected_return_date: "",
    notes: "",
  });
  const [error, setError] = useState("");
  const navigate = useNavigate();

  useEffect(() => {
    getEquipment().then(res => setEquipments(res.data.filter(e =>
e.status === "в наличии")));
  }, []);

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError("");
    try {
      await createRequest({
        equipment_id: parseInt(form.equipment_id),
        expected_return_date: form.expected_return_date || null,
        notes: form.notes || "",
      });
      navigate("/equipment");
      alert("Заявка создана. Ожидайте одобрения администратора.");
    } catch (err) {
      setError(formatError(err));
    }
  };

  return (
    <div className="container-fluid mt-4">
      <h3>Запрос на выдачу оборудования</h3>
      {error && <div className="alert alert-danger">{error}</div>}
    </div>
  );
}
```

```

    <form onSubmit={handleSubmit}>
      <div className="mb-3">
        <label>Оборудование</label>
        <select className="form-select" value={form.equipment_id}
onChange={(e) => setForm({ ...form, equipment_id: e.target.value })}
required>
          <option value="">-- выберите --</option>
          {equipments.map(eq => <option key={eq.id}
value={eq.id}>{eq.inventory_number} - {eq.name}
({eq.location}))</option>)}
        </select>
      </div>
      <div className="mb-3">
        <label>Ожидаемая дата возврата</label>
        <input type="date" className="form-control"
value={form.expected_return_date} onChange={(e) => setForm({
...form, expected_return_date: e.target.value })} />
      </div>
      <div className="mb-3">
        <label>Примечание</label>
        <textarea className="form-control" value={form.notes}
onChange={(e) => setForm({ ...form, notes: e.target.value })} />
      </div>
      <button type="submit" className="btn btn-success">Отправить
заявку</button>
    </form>
  </div>
);
}

```

Файл «Reports.jsx»

```

import { useState } from "react";
import API from "../api";

function buildParams(filters, extra = {}) {
  const params = { ...extra };
  Object.entries(filters).forEach(([key, value]) => {
    if (value !== "" && value !== null && value !== undefined) {
      params[key] = value;
    }
  });
  return params;
}

export default function Reports() {
  const [eqFilters, setEqFilters] = useState({
    type: "",
    status: "",
    location: "",
    min_price: "",
    max_price: "",
  });

```

```

        search: "",
    });
    const [issuedDate, setIssuedDate] = useState("");
    const [writtenOffStart, setWrittenOffStart] = useState("");
    const [writtenOffEnd, setWrittenOffEnd] = useState("");

    const downloadEquipmentReport = async () => {
        try {
            const params = buildParams(eqFilters, { format: "xlsx" });
            const response = await API.get("/reports/equipment-state", {
                params,
                responseType: "blob",
            });
            const blob = new Blob([response.data], { type:
"application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"
});
            const url = window.URL.createObjectURL(blob);
            const link = document.createElement("a");
            link.href = url;
            link.setAttribute("download", "equipment_state.xlsx");
            document.body.appendChild(link);
            link.click();
            link.remove();
            window.URL.revokeObjectURL(url);
        } catch (error) {
            alert("Ошибка загрузки отчёта: " +
(error.response?.data?.detail || error.message));
            console.error(error);
        }
    };

    const downloadIssuedReport = async () => {
        try {
            const extra = { format: "xlsx" };
            if (issuedDate) extra.date = issuedDate;
            const params = buildParams({}, extra);
            const response = await API.get("/reports/issued-equipment", {
                params,
                responseType: "blob",
            });
            const blob = new Blob([response.data], { type:
"application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"
});
            const url = window.URL.createObjectURL(blob);
            const link = document.createElement("a");
            link.href = url;
            link.setAttribute("download", "issued_equipment.xlsx");
            document.body.appendChild(link);
            link.click();
            link.remove();
            window.URL.revokeObjectURL(url);
        } catch (error) {

```

```

        alert("Ошибка загрузки отчёта: " +
(error.response?.data?.detail || error.message));
        console.error(error);
    }
};

const downloadWrittenOffReport = async () => {
    try {
        const params = { format: "xlsx" };
        if (writtenOffStart) params.start_date = writtenOffStart;
        if (writtenOffEnd) params.end_date = writtenOffEnd;
        const response = await API.get("/reports/written-off-
equipment", {
            params,
            responseType: "blob",
        });
        const blob = new Blob([response.data], { type:
"application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"
});
        const url = window.URL.createObjectURL(blob);
        const link = document.createElement("a");
        link.href = url;
        link.setAttribute("download", "written_off_equipment.xlsx");
        document.body.appendChild(link);
        link.click();
        link.remove();
        window.URL.revokeObjectURL(url);
    } catch (error) {
        alert("Ошибка загрузки отчёта: " +
(error.response?.data?.detail || error.message));
        console.error(error);
    }
};

return (
<div className="container-fluid mt-4">
    <h3>Отчёты</h3>
    <div className="row">
        <div className="col-md-6">
            <div className="card p-3 mb-3">
                <h5>Состояние оборудования</h5>
                <div className="row g-2">
                    <div className="col-6">
                        <label>Тип</label>
                        <select
                            className="form-select"
                            value={eqFilters.type}
                            onChange={(e) => setEqFilters({ ...eqFilters,
type: e.target.value })}
                        >
                            <option value="">Все</option>
                            <option value="ноутбук">Ноутбук</option>
                            <option value="компьютер">Компьютер</option>

```

```

        <option value="принтер">Принтер</option>
        <option value="сервер">Сервер</option>
    </select>
</div>
<div className="col-6">
    <label>Статус</label>
    <select
        className="form-select"
        value={eqFilters.status}
        onChange={(e) => setEqFilters({ ...eqFilters,
status: e.target.value })}
    >
        <option value="">Все</option>
        <option value="в наличии">В наличии</option>
        <option value="выдан">Выдан</option>
        <option value="в ремонте">В ремонте</option>
        <option value="списан">Списан</option>
    </select>
</div>
<div className="col-6">
    <label>Местоположение</label>
    <input
        type="text"
        className="form-control"
        value={eqFilters.location}
        onChange={(e) => setEqFilters({ ...eqFilters,
location: e.target.value })}
    />
</div>
<div className="col-3">
    <label>Цена от</label>
    <input
        type="number"
        className="form-control"
        value={eqFilters.min_price}
        onChange={(e) => setEqFilters({ ...eqFilters,
min_price: e.target.value })}
    />
</div>
<div className="col-3">
    <label>Цена до</label>
    <input
        type="number"
        className="form-control"
        value={eqFilters.max_price}
        onChange={(e) => setEqFilters({ ...eqFilters,
max_price: e.target.value })}
    />
</div>
<div className="col-12">
    <label>Поиск</label>
    <input
        type="text"

```

```

        className="form-control"
        value={eqFilters.search}
        onChange={(e) => setEqFilters({ ...eqFilters,
search: e.target.value })}
      />
    </div>
  </div>
  <button className="btn btn-primary mt-3"
onClick={downloadEquipmentReport}>
    Скачать Excel
  </button>
</div>
</div>

<div className="col-md-6">
  <div className="card p-3 mb-3">
    <h5>Выданное оборудование</h5>
    <div className="mb-3">
      <label>На дату (оставьте пустым для текущего
момента)</label>
      <input
        type="date"
        className="form-control"
        value={issuedDate}
        onChange={(e) => setIssuedDate(e.target.value)}
      />
    </div>
    <button className="btn btn-primary"
onClick={downloadIssuedReport}>
      Скачать Excel
    </button>
  </div>

  <div className="card p-3 mb-3">
    <h5>Списанное оборудование</h5>
    <div className="mb-3">
      <label>Дата списания с</label>
      <input
        type="date"
        className="form-control"
        value={writtenOffStart}
        onChange={(e) => setWrittenOffStart(e.target.value)}
      />
    </div>
    <div className="mb-3">
      <label>Дата списания по</label>
      <input
        type="date"
        className="form-control"
        value={writtenOffEnd}
        onChange={(e) => setWrittenOffEnd(e.target.value)}
      />
    </div>
  </div>

```

```

        <button className="btn btn-primary"
onClick={downloadWrittenOffReport}>
            Скачать Excel
        </button>
    </div>
</div>
</div>
</div>
);
}

```

Файл «Navbar.jsx»

```

import { Link, useNavigate } from "react-router-dom";

export default function Navbar({ onLogout, role }) {
    const navigate = useNavigate();

    const handleLogout = () => {
        onLogout();
        navigate("/");
    };

    return (
        <nav className="navbar navbar-expand-lg navbar-dark bg-dark">
            <div className="container-fluid">
                <Link className="navbar-brand" to="/equipment">
                    Учёт оборудования
                </Link>
                <button className="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarNav">
                    <span className="navbar-toggler-icon"></span>
                </button>
                <div className="collapse navbar-collapse" id="navbarNav">
                    <ul className="navbar-nav me-auto">
                        <li className="nav-item">
                            <Link className="nav-link"
to="/equipment">Оборудование</Link>
                        </li>
                        {role === 'admin' && (
                            <li className="nav-item">
                                <Link className="nav-link"
to="/equipment/new">Добавить</Link>
                            </li>
                        )}
                        {role === 'admin' ? (
                            <>
                                <li className="nav-item">
                                    <Link className="nav-link"
to="/issue">Выдать</Link>
                                </li>
                                <li className="nav-item">

```

```

        <Link className="nav-link"
to="/requests">Заявки</Link>
    </li>
    <li className="nav-item">
        <Link className="nav-link"
to="/users">Пользователи</Link>
    </li>
    <li className="nav-item">
        <Link className="nav-link"
to="/reports">Отчёты</Link>
    </li>
</>
) : (
    <li className="nav-item">
        <Link className="nav-link" to="/issue">Запрос на
выдачу</Link>
    </li>
    </ul>
    <button className="btn btn-outline-light"
onClick={handleLogout}>
        Выйти
    </button>
</div>
</div>
</nav>
);
}

```

Файл «MovementsHistory.jsx»

```

import { useEffect, useState } from "react";
import { useParams } from "react-router-dom";
import { getMovementsForEquipment, returnEquipment, getMe } from
"../api";

export default function MovementsHistory() {
    const { equipmentId } = useParams();
    const [movements, setMovements] = useState([]);
    const [currentUser, setCurrentUser] = useState(null);

    useEffect(() => {
        getMe()
            .then(res => setCurrentUser(res.data))
            .catch(err => console.error("Ошибка получения пользователя",
err));
    }, []);

    const fetchMovements = () => {
        if (equipmentId) {
            getMovementsForEquipment(equipmentId)
                .then((res) => setMovements(res.data))

```

```

        .catch((err) => console.error("Ошибка загрузки истории",
err));
    }
};

useEffect(() => {
    fetchMovements();
}, [equipmentId]);

const handleReturn = async (movementId) => {
    try {
        await returnEquipment(movementId, {});
        fetchMovements();
    } catch (err) {
        alert("Ошибка возврата: " + (err.response?.data?.detail ||
err.message));
    }
};

const canReturn = (movement) => {
    if (!currentUser || movement.actual_return_date) return false;
    return currentUser.role === 'admin' || currentUser.id ===
movement.user_id;
};

return (
    <div>
        <h3>История перемещений (ID оборудования: {equipmentId})</h3>
        {movements.length === 0 ? (
            <p>Нет записей о перемещениях</p>
        ) : (
            <table className="table">
                <thead>
                    <tr>
                        <th>Дата выдачи</th>
                        <th>Сотрудник</th>
                        <th>Ожидаемая дата возврата</th>
                        <th>Фактическая дата возврата</th>
                        <th>Примечание</th>
                        <th>Действия</th>
                    </tr>
                </thead>
                <tbody>
                    {movements.map((m) => (
                        <tr key={m.id}>
                            <td>{new Date(m.issue_date).toLocaleString()}</td>
                            <td>{m.user?.full_name || "-"}</td>
                            <td>{m.expected_return_date || "-"}</td>
                            <td>{m.actual_return_date ? new
Date(m.actual_return_date).toLocaleString() : "Не возвращено"}</td>
                            <td>{m.notes || "-"}</td>
                            <td>
                                {canReturn(m) && (

```

```

        <button onClick={() => handleReturn(m.id)}
className="btn btn-sm btn-warning">Вернуть</button>
        )}
      </td>
    </tr>
  )})
</tbody>
</table>
)}
</div>
);
}

```

Файл «Login.jsx»

```

import { useState } from "react";
import { login } from "../api";

export default function Login({ onLogin }) {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError("");
    try {
      const res = await login(username, password);
      onLogin(res.data.access_token);
    } catch (err) {
      if (err.response?.status === 401) {
        setError("Неверное имя пользователя или пароль");
      } else if (err.response?.status === 422) {
        setError("Ошибка валидации данных");
      } else {
        setError("Ошибка соединения с сервером");
      }
    }
  };

  return (
    <div className="d-flex justify-content-center align-items-center" style={{ minHeight: '100vh' }}>
      <div className="card p-4 shadow" style={{ width: '100%',
maxWidth: '400px' }}>
        <h3 className="text-center mb-3">Учёт оборудования</h3>
        <hr />
        <form onSubmit={handleSubmit}>
          <div className="mb-3">
            <label className="form-label">Логин</label>
            <input
              className="form-control"

```

```

        value={username}
        onChange={ (e) => setUsername(e.target.value)}
        required
        autoFocus
      />
    </div>
    <div className="mb-3">
      <label className="form-label">Пароль</label>
      <input
        type="password"
        className="form-control"
        value={password}
        onChange={ (e) => setPassword(e.target.value)}
        required
      />
    </div>
    {error && <div className="alert alert-danger py-1">{error}</div>}
    <button type="submit" className="btn btn-primary w-100">
      Войти
    </button>
  </form>
</div>
</div>
);
}

```

Файл «IssueEquipment.jsx»

```

import { useState, useEffect } from "react";
import { getEquipment, getUsers, issueEquipment } from "../api";
import { useNavigate } from "react-router-dom";

export default function IssueEquipment() {
  const [equipments, setEquipments] = useState([]);
  const [users, setUsers] = useState([]);
  const [form, setForm] = useState({
    equipment_id: "",
    user_id: "",
    expected_return_date: "",
    notes: "",
  });
  const [error, setError] = useState("");
  const navigate = useNavigate();

  useEffect(() => {
    getEquipment().then(res => setEquipments(res.data.filter(e =>
e.status === "в наличии")));
    getUsers().then(res => setUsers(res.data.filter(u => u.is_active
=== "active")));
  }, []);

  const handleSubmit = async (e) => {
    e.preventDefault();

```

```

try {
  await issueEquipment({
    equipment_id: parseInt(form.equipment_id),
    user_id: parseInt(form.user_id),
    expected_return_date: form.expected_return_date || null,
    notes: form.notes,
  });
  navigate("/equipment");
} catch (err) {
  setError(err.response?.data?.detail || "Ошибка выдачи");
}
};

return (
  <div className="container-fluid mt-4">
    <h3>Выдача оборудования</h3>
    {error && <div className="alert alert-danger">{error}</div>}
    <form onSubmit={handleSubmit}>
      <div className="mb-3">
        <label>Оборудование</label>
        <select className="form-select" value={form.equipment_id}
onChange={(e) => setForm({ ...form, equipment_id: e.target.value })}
required>
          <option value="">-- выберите --</option>
          {equipments.map(eq => <option key={eq.id}
value={eq.id}>{eq.inventory_number} - {eq.name}
({eq.location})</option>)}
        </select>
      </div>
      <div className="mb-3">
        <label>Сотрудник</label>
        <select className="form-select" value={form.user_id}
onChange={(e) => setForm({ ...form, user_id: e.target.value })}
required>
          <option value="">-- выберите --</option>
          {users.map(u => <option key={u.id}
value={u.id}>{u.full_name} ({u.username})</option>)}
        </select>
      </div>
      <div className="mb-3">
        <label>Ожидаемая дата возврата</label>
        <input type="date" className="form-control"
value={form.expected_return_date} onChange={(e) => setForm({
...form, expected_return_date: e.target.value })} />
      </div>
      <div className="mb-3">
        <label>Примечание</label>
        <input type="text" className="form-control"
value={form.notes} onChange={(e) => setForm({ ...form, notes:
e.target.value })} />
      </div>
      <button type="submit" className="btn btn-
success">Выдать</button>

```

```

        </form>
      </div>
    );
  }
}
Файл «EquipmentList.jsx»

import { useEffect, useState } from "react";
import { getEquipment, deleteEquipment, hardDeleteEquipment } from
"../api";
import { Link } from "react-router-dom";

const EQUIPMENT_TYPES = [
  "ноутбук", "компьютер", "принтер", "сервер", "периферия", "другое"
];

const EQUIPMENT_STATUSES = [
  "в наличии", "выдан", "в ремонте", "списан"
];

export default function EquipmentList() {
  const [equipment, setEquipment] = useState([]);
  const [filters, setFilters] = useState({
    type: "",
    status: "",
    location: "",
    min_price: "",
    max_price: "",
  });
  const [search, setSearch] = useState("");
  const [searchValue, setSearchValue] = useState("");
  const userRole = localStorage.getItem('user_role') || 'user';
  const isAdmin = userRole === 'admin';

  const fetchEquipment = async () => {
    try {
      const params = {};
      if (search) params.search = search;
      if (filters.type) params.type = filters.type;
      if (filters.status) params.status = filters.status;
      if (filters.location) params.location = filters.location;
      if (filters.min_price) params.min_price = filters.min_price;
      if (filters.max_price) params.max_price = filters.max_price;
      const res = await getEquipment(params);
      setEquipment(res.data);
    } catch (err) {
      console.error("Ошибка загрузки:", err);
    }
  };

  useEffect(() => {
    fetchEquipment();
  }, [filters, search]);

```

```

const handleFilterChange = (e) => {
  const { name, value } = e.target;
  setFilters({ ...filters, [name]: value });
};

const handleSearch = () => {
  setSearch(searchValue.trim());
};

const handleKeyDown = (e) => {
  if (e.key === "Enter") {
    handleSearch();
  }
};

const handleDelete = async (id) => {
  if (window.confirm("Списать оборудование?")) {
    await deleteEquipment(id);
    fetchEquipment();
  }
};

const formatDate = (dateStr) => {
  if (!dateStr) return "-";
  const d = new Date(dateStr);
  return d.toLocaleDateString("ru-RU");
};

const formatPrice = (price) => {
  if (price == null) return "-";
  return Number(price).toLocaleString("ru-RU", {
    minimumFractionDigits: 2,
    maximumFractionDigits: 2,
  });
};

const handleHardDelete = async (id) => {
  if (window.confirm("Вы уверены, что хотите безвозвратно удалить оборудование? Это действие нельзя отменить.")) {
    try {
      await hardDeleteEquipment(id);
      fetchEquipment();
    } catch (err) {
      alert("Ошибка удаления: " + (err.response?.data?.detail || err.message));
    }
  }
};

return (
  <div>
    <h3>Список оборудования</h3>

```

```

<div className="row mb-3">
  <div className="col-md-6 col-lg-4">
    <label className="form-label">Поиск по названию, инв.
номеру, местоположению, примечанию</label>
    <div className="input-group">
      <input
        type="text"
        className="form-control"
        placeholder="Введите текст для поиска..."
        value={searchValue}
        onChange={(e) => setSearchValue(e.target.value)}
        onKeyDown={handleKeyDown}
      />
      <button className="btn btn-primary"
onClick={handleSearch}>
        🔍 Найти
      </button>
    </div>
  </div>
</div>

<div className="row mb-3 g-2">
  <div className="col-md-2">
    <label className="form-label">Тип</label>
    <select className="form-select" name="type"
value={filters.type} onChange={handleFilterChange}>
      <option value="">Все</option>
      {EQUIPMENT_TYPES.map((t) => (
        <option key={t} value={t}>{t}</option>
      ))}
    </select>
  </div>
  <div className="col-md-2">
    <label className="form-label">Статус</label>
    <select className="form-select" name="status"
value={filters.status} onChange={handleFilterChange}>
      <option value="">Все</option>
      {EQUIPMENT_STATUSES.map((s) => (
        <option key={s} value={s}>{s}</option>
      ))}
    </select>
  </div>
  <div className="col-md-2">
    <label className="form-label">Местоположение</label>
    <input
      type="text"
      className="form-control"
      placeholder="Кабинет..."
      name="location"
      value={filters.location}
      onChange={handleFilterChange}
    />
  </div>
</div>

```

```

</div>
<div className="col-md-2">
  <label className="form-label">Цена от</label>
  <input
    type="number"
    className="form-control"
    placeholder="0"
    name="min_price"
    value={filters.min_price}
    onChange={handleFilterChange}
  />
</div>
<div className="col-md-2">
  <label className="form-label">Цена до</label>
  <input
    type="number"
    className="form-control"
    placeholder="50000"
    name="max_price"
    value={filters.max_price}
    onChange={handleFilterChange}
  />
</div>
</div>

<div className="table-responsive">
  <table className="table table-striped table-hover align-
middle">
    <thead className="table-dark">
      <tr>
        <th>Инв. номер</th>
        <th>Название</th>
        <th>Тип</th>
        <th>Серийный номер</th>
        <th>Дата покупки</th>
        <th>Цена (₽)</th>
        <th>Статус</th>
        <th>Местоположение</th>
        <th>Примечание</th>
        <th>Действия</th>
      </tr>
    </thead>
    <tbody>
      {equipment.length === 0 ? (
        <tr>
          <td colspan={10} className="text-center">Нет
данных</td>
        </tr>
      ) : (
        equipment.map((e) => (
          <tr key={e.id}>
            <td>{e.inventory_number}</td>
            <td>{e.name}</td>

```

```

        <td>{e.type}</td>
        <td>{e.serial_number || "-"}</td>
        <td>{formatDate(e.purchase_date)}</td>
        <td className="text-
end">{formatPrice(e.price)}</td>
        <td>
            <span className={`badge bg-${e.status === 'в
наличии' ? 'success' : e.status === 'выдан' ? 'warning' : e.status
=== 'в ремонте' ? 'info' : 'secondary'}`}>
                {e.status}
            </span>
        </td>
        <td>{e.location}</td>
        <td>{e.notes || "-"}</td>
        <td>
            <div className="d-flex gap-1">
                {isAdmin && (
                    <>
                        <Link to={` /equipment/${e.id}/edit`}
className="btn btn-sm btn-outline-primary"
title="Редактировать">✎</Link>
                        <button onClick={() => handleDelete(e.id)}
className="btn btn-sm btn-outline-danger"
title="Списать">🗑️</button>
                        <button onClick={() =>
handleHardDelete(e.id)} className="btn btn-sm btn-outline-dark"
title="Удалить навсегда">✖️</button>
                    </>
                )}
                <Link to={` /movements/${e.id}`} className="btn
btn-sm btn-outline-info" title="История">📅</Link>
            </div>
        </td>
    </tr>
</tbody>
</table>
</div>
</div>
);
}

```

Файл «EquipmentForm.jsx»

```

// src/components/EquipmentForm.jsx
import { useState, useEffect } from "react";
import { useNavigate, useParams } from "react-router-dom";
import { createEquipment, getEquipment, updateEquipment } from
"../api";

const emptyForm = {
    name: "",

```

```

    inventory_number: "",
    type: "",
    serial_number: "",
    purchase_date: "",
    price: "",
    location: "Склад",
    status: "в наличии",
    notes: "",
  };

export default function EquipmentForm() {
  const { id } = useParams();
  const isEdit = Boolean(id);
  const navigate = useNavigate();
  const [form, setForm] = useState(emptyForm);
  const [error, setError] = useState("");

  useEffect(() => {
    if (isEdit) {
      const fetchEquip = async () => {
        try {
          const res = await getEquipment(); // загружаем все, но
          можно доработать получение одного
          const equip = res.data.find((e) => e.id === parseInt(id));
          if (equip) {
            setForm({
              ...equip,
              purchase_date: equip.purchase_date || "",
            });
          }
        } catch (err) {
          setError("Ошибка загрузки оборудования");
        }
      };
      fetchEquip();
    }
  }, [id, isEdit]);

  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const payload = {
        ...form,
        price: form.price ? parseFloat(form.price) : null,
        purchase_date: form.purchase_date || null,
      };
      if (isEdit) {
        await updateEquipment(id, payload);
      } else {

```

```

        await createEquipment(payload);
    }
    navigate("/equipment");
} catch (err) {
    setError("Ошибка сохранения: " + (err.response?.data?.detail
|| err.message));
}
};

return (
    <div>
        <h3>{isEdit ? "Редактирование оборудования" : "Добавление
оборудования"}</h3>
        {error && <div className="alert alert-danger">{error}</div>}
        <form onSubmit={handleSubmit}>
            <div className="row mb-2">
                <div className="col">
                    <label>Название</label>
                    <input className="form-control" name="name"
value={form.name} onChange={handleChange} required />
                </div>
                <div className="col">
                    <label>Инвентарный номер</label>
                    <input className="form-control" name="inventory_number"
value={form.inventory_number} onChange={handleChange} required />
                </div>
            </div>

            <div className="row mb-2">
                <div className="col">
                    <label>Тип</label>
                    <select className="form-select" name="type"
value={form.type} onChange={handleChange} required>
                        <option value="">--выберите--</option>
                        <option value="ноутбук">Ноутбук</option>
                        <option value="компьютер">Компьютер</option>
                        <option value="принтер">Принтер</option>
                        <option value="сервер">Сервер</option>
                        <option value="периферия">Периферия</option>
                        <option value="другое">Другое</option>
                    </select>
                </div>
                <div className="col">
                    <label>Серийный номер</label>
                    <input className="form-control" name="serial_number"
value={form.serial_number} onChange={handleChange} />
                </div>
            </div>

            <div className="row mb-2">
                <div className="col">
                    <label>Дата покупки</label>

```

```

        <input type="date" className="form-control"
name="purchase_date" value={form.purchase_date}
onChange={handleChange} />
    </div>
    <div className="col">
        <label>Цена (руб.)</label>
        <input type="number" step="0.01" className="form-
control" name="price" value={form.price} onChange={handleChange} />
    </div>
</div>

<div className="row mb-2">
    <div className="col">
        <label>Местоположение</label>
        <input className="form-control" name="location"
value={form.location} onChange={handleChange} />
    </div>
    <div className="col">
        <label>Статус</label>
        <select className="form-select" name="status"
value={form.status} onChange={handleChange}>
            <option value="в наличии">В наличии</option>
            <option value="в ремонте">В ремонте</option>
            <option value="списан">Списан</option>
        </select>
    </div>
</div>

<div className="mb-2">
    <label>Примечания</label>
    <textarea className="form-control" name="notes"
value={form.notes} onChange={handleChange} />
</div>

    <button type="submit" className="btn btn-
success">Сохранить</button>
    <button type="button" className="btn btn-secondary ms-2"
onClick={() => navigate("/equipment")}>Отмена</button>
</form>
</div>
);
}

```

Приложение В

(обязательное)

Клиент-серверная информационная система для учета оборудования

Презентация

					ПГУ 1.090301.03.001 ПЗ	Лист
						114
Изм.	Лист	№ докум.	Подпись	Дата		

Клиент-серверная информационная система для учёта оборудования

Автор: Ивлюшин М.Д.
Руководитель: Слепцов Н.В.
Пенза, 2026

Цель и задачи работы

Цель: разработка клиент-серверной информационной системы для учёта оборудования, обеспечивающей централизованное хранение данных об оборудовании, многопользовательский доступ с разграничением прав, учёт перемещений и формирование отчётов

Задачи:

- Анализ предметной области и аналогов
- Выбор технологического стека
- Проектирование архитектуры и базы данных
- Реализация серверной части
- Реализация клиентского интерфейса
- Тестирование системы

Изм.	Лист	№ докум.	Подпись	Дата

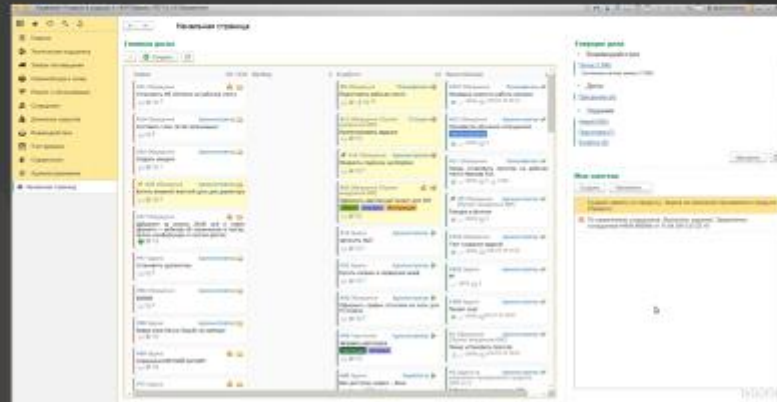
ПГУ 1.090301.03.001 ПЗ

Лист

115

Анализ аналогов

1С:Управление IT-отделом (1С:ITIL) : дорого, сложно внедрять, требует специалиста по 1С, избыточно для малых организаций. Включает учёт активов, управление заявками, SLA-мониторинг, интеграцию с Active Directory.



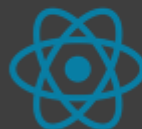
Анализ аналогов

GLPI – трудоёмкая установка, устаревший перегруженный интерфейс, неполная поддержка русского языка.



Стек технологий

- Бэкенд: Python + FastAPI
- Фронтенд: React + JavaScript
- СУБД: PostgreSQL



Функциональные требования

- Аутентификация (логин/пароль + токен)
- Добавление, редактирование, списание оборудования
- Учёт выдачи и возврата, история перемещений
- Заявки на выдачу (пользователь → администратор)
- Управление пользователями и ролями (админ/пользователь)
- Поиск и фильтрация оборудования
- Формирование отчётов в Excel

Изм.	Лист	№ докум.	Подпись	Дата

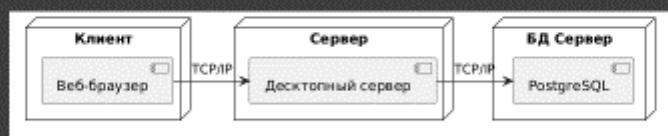
Архитектура приложения

Клиент ↔ Сервер приложений ↔ База данных

Взаимодействие через REST API, обмен JSON

Аутентификация: JWT-токены, пароли хешируются (bcrypt)

Ролевая модель: администратор / пользователь



Физическая модель базы данных

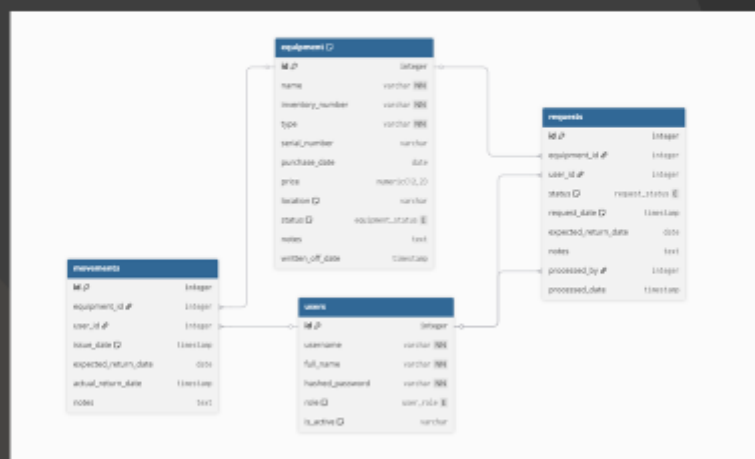


Диаграмма вариантов использования



Интерфейс: страница входа

Учёт оборудования

Логин

Пароль

Войти

Учёт оборудования

Логин

Пароль

Неверное имя пользователя или пароль

Войти

Интерфейс: список оборудования

Удп. оборудование | Оборудование | Запрос на выдачу

После нажатия на кнопку «Запрос на выдачу»

Введите текст для поиска: [Найти](#)

Всего: 3 | Вкл: 3 | Выкл: 0

Статус: | Оборудование: | Страница: | Страница 10

№ инв.	Наименование	Тип	Серийный номер	Дата покупки	Цена (Р)	Статус	Информация	Примечание	Действия
10-001	Система ИТ-оборудования	система	10001	—	10 000,00	Активен	Система ИТ	—	Изменить
10-002	Система ИТ-оборудования	система	10002	—	10 000,00	Активен	Система ИТ	—	Изменить
10-003	Система ИТ-оборудования	система	10003	—	10 000,00	Активен	Система ИТ	—	Изменить

Интерфейс: выдача и заявки

Запрос на выдачу оборудования

Оборудование:

Описание оборудования:

Примечание:

[Отправить заявку](#)

Удп. оборудование | Оборудование | Запрос на выдачу | Заявки | Оборудование | Статус

Выдача оборудования

Оборудование:

Описание оборудования:

Примечание:

[Отправить заявку](#)

Удп. оборудование | Оборудование | Запрос на выдачу | Заявки | Оборудование | Статус

Заявки на выдачу

Оборудование	Информация	Дата заявки	Информация о заявке	Статус	Действия
10-001 Система ИТ-оборудования	10001	10.09.2019 10:00:00	10.09.2019 10:00:00	Активен	Изменить
10-002 Система ИТ-оборудования	10002	10.09.2019 10:00:00	10.09.2019 10:00:00	Активен	Изменить
10-003 Система ИТ-оборудования	10003	10.09.2019 10:00:00	10.09.2019 10:00:00	Активен	Изменить
10-004 Система ИТ-оборудования	10004	10.09.2019 10:00:00	10.09.2019 10:00:00	Активен	Изменить
10-005 Система ИТ-оборудования	10005	10.09.2019 10:00:00	10.09.2019 10:00:00	Активен	Изменить
10-006 Система ИТ-оборудования	10006	10.09.2019 10:00:00	10.09.2019 10:00:00	Активен	Изменить

Пример отчёта

№	A	B	C	D	E	F	G	H	I	J
1	Имя.номер	Название	Тип	Серийный номер	Дата покупки	Цена	Статус	Местоположение	Примечание	
2	NS-001	Ноутбук HP ProBook	ноутбук	SN111		85000	EquipmentStatus.available	Каб. 101		
3	SV-001	Сервер HP ProLiant	сервер	SN333		250000	EquipmentStatus.available	Серверная		
4	111	test1	другое				EquipmentStatus.available	Склад		
5	PR-001	Принтер Canon LBP	принтер	SN222		12000	EquipmentStatus.available	Каб. 203		

№	A	B	C	D	E
1	Оборудование	Сотрудник	Дата выдачи	Ожидаемый возврат	Примечание
2	PR-001. Принтер Canon LBP	test11	2026-05-07 18:55:25	2026-05-25	Пользуясь аккуратно
3					

Преимущества и перспективы развития

Преимущества разработанной программы:

- Бесплатное открытое решение
- Простота развёртывания (Python + PostgreSQL)
- Современный веб-интерфейс на русском языке
- Разграничение прав (админ / пользователь)
- Гибкая фильтрация и поиск
- Выгрузка отчётов в Excel

Перспективы:

- Штрихкодирование оборудования
- Уведомления по email о возврате
- Мобильное приложение для инвентаризации
- Интеграция с бухгалтерскими системами

Спасибо за внимание!

					ПГУ 1.090301.03.001 ПЗ	Лист
						122
Изм.	Лист	№ докум.	Подпись	Дата		