

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

ПРОГРАММНО-АППАРАТНОЕ РЕШЕНИЕ
СИСТЕМЫ УПРАВЛЕНИЯ И КОНТРОЛЯ
ДОСТУПОМ

Студент

10.06.2026 Ганин Иван Романович
(подпись, дата) (ФИО полностью)

Руководитель

10.06.2026 Митрохин М.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

10.06.2026 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

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

Заведующий кафедрой

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

Работа защищена с отметкой отлично (протокол заседания ГЭК от

15 июня 2026 № 14)

Секретарь ГЭК

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

Пенза 2026

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

Факультет
вычислительной техники

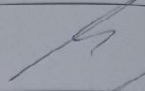
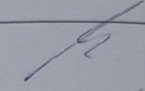
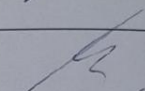
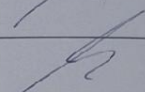
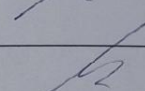
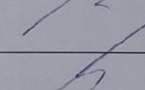
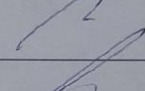
Кафедра
«Вычислительная техника»

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

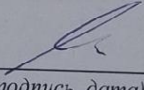
**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

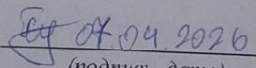
1. Студент Ганин Иван Романович гр. 22BBV1 факультета вычислительной техники.
2. Тема работы: «Программно-аппаратное решение системы управления и контроля доступом»
Тема утверждена приказом ПГУ № 347/о от "7" апреля 2026 г.
3. Руководитель бакалаврской работы д.т.н., зав. кафедры ВТ ПГУ Митрохин М.А.
4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать программно-аппаратную систему управления и контроля доступом. Система должна осуществлять идентификацию пользователей по ТМ-ключам, централизованно управлять правами доступа, вести журнал событий и обеспечивать удалённое администрирование через веб-интерфейс..
5. Перечень подлежащих разработке вопросов
Произвести анализ существующих программно-аппаратных решений в области систем контроля доступа и обосновать актуальность разработки. Определить назначение, требования и общий состав разрабатываемой СКУД. Выполнить выбор аппаратных средств (микроконтроллер, считыватель, замок, источники питания). Разработать общую структуру системы. Разработать программное обеспечение для управления комплексом. Произвести опытную проверку работы системы и оценить её эффективность.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	07.04.2026 – 11.04.2026	
2	Анализ аналогов, актуальности, постановка задачи	24	11.04.2026 – 14.04.2026	
3	Выбор и настройка средств разработки	4	14.04.2026 – 15.04.2026	
4	Определение состава комплекса	30	15.04.2026 – 19.04.2026	
5	Разработка аппаратной части комплекса	100	19.04.2026 – 04.05.2026	
6	Разработка программного обеспечения для управления комплексом	55	04.05.2026 – 11.05.2026	
7	Опытная проверка и оценка эффективности	54	11.05.2026 – 18.05.2026	
8	Оформление пояснительной записки	25	18.05.2026 – 21.05.2026	

Дата выдачи задания " 07 " апреля 2026г.

Руководитель бакалаврской работы  Митрохин М.А.
(подпись, дата)

Задание к исполнению принял студент  07.04.2026 Ганин И.Р.
(подпись, дата)

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

Реферат

Бакалаврская работа содержит 89 листов, 54 рисунка, 1 таблицу, 16 источников и 4 приложения.

СИСТЕМА УПРАВЛЕНИЯ И КОНТРОЛЯ ДОСТУПОМ, СКУД, ПРОГРАММНО-АППАРАТНЫЙ КОМПЛЕКС, МИКРОКОНТРОЛЛЕР ESP32, СЧИТЫВАТЕЛЬ TOUCH MEMORY, ТМ, ЭЛЕКТРОМАГНИТНЫЙ ЗАМОК, ВЕБ-ИНТЕРФЕЙС, БАЗА ДАННЫХ SQLite

Цель работы заключается в разработке программно-аппаратной системы управления и контроля доступом (СКУД), обеспечивающей идентификацию пользователей по ТМ-ключам, централизованное управление правами доступа, ведение журнала событий и удалённое администрирование через веб-интерфейс.

Объектом исследования в работе являются процессы идентификации и аутентификации пользователей в системах контроля доступа, методы централизованного логирования событий и способы организации каналов связи между контроллером и сервером.

В работе применены методы программирования микроконтроллера ESP32 на языке C++ в среде Arduino IDE, методы создания асинхронного веб-приложения с использованием фреймворка FastAPI и языка Python, методы проектирования реляционной базы данных SQLite и разработки веб-интерфейса администратора на HTML/CSS/JS с применением Bootstrap 5.

В работе разработан опытный образец СКУД, включающий контроллер на базе ESP32 с ТМ-считывателем, серверное приложение с REST API и веб-интерфейс администратора, а также обеспечено централизованное хранение журнала событий, удалённое открытие двери, динамическое обновление белого списка ключей и режим обучения для добавления новых ключей прикладыванием к считывателю.

Изм	Лист	№ докум.	Подпись	Дата
Разраб.		Ганин И.Р.		10.04.20
Пров.		Митрохин М.А.		10.04.20
Н. Контр.		Бычков А.С.		10.04.20
Утв.		Митрохин М.А.		11.04.20

ПГУ 1.090301.01.001 ПЗ		
Литера	Лист	Листов
	2	89

Программно-аппаратное решение
системы управления и контроля
доступом
Пояснительная записка

ФВТ, каф. ВТ
гр. 22BBB1

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	7
1 Анализ предметной области.....	9
1.1 Анализ существующих аналогов.....	11
1.2 Анализ актуальности разработки	16
1.3 Постановка задачи	18
2 Выбор технологий и средств разработки.....	22
2.1 Выбор языка программирования.....	22
2.2 Выбор СУБД.....	24
2.3 Выбор среды разработки	26
2.4 Настройка среды разработки	28
3 Реализация.....	40
3.1 Архитектура системы.....	40
3.2 Работа с базой данных.....	42
3.3 Разработка аппаратной части.....	44
3.3.1 Главный цикл опроса и идентификации.....	44
3.3.2 Отправка событий на сервер.....	45
3.4 Разработка серверного приложения и веб-интерфейса	47
3.4.1 Модуль приёма событий от контроллера	47
3.4.2 Модуль журнала событий (access_log).....	48
3.4.3 Модуль управления ключами доступа	48
3.4.4 Модуль удалённого открытия двери.....	49
3.5 Тестирование	51
4 Безопасность жизнедеятельности.....	58
4.1 Характеристика потенциальных рисков при работе с комплексом.....	58
4.2 Требования безопасности при эксплуатации комплекса	59
4.3 Организация рабочего места оператора	60
4.4 Электробезопасность	60

4.5 Пожарная безопасность	61
4.6 Санитарно-гигиенические требования.....	61
Заключение	63
Список использованных источников	65
Приложение А – Схемы электрические	67
Приложение Б – UML-диаграммы.....	69
Приложение В – Листинг программы.....	73
Приложение Г - Презентация.....	85

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

СКУД – система управления и контроля доступом

СУБД – система управления базами данных

TM (Touch Memory) – технология однопроводных ключей-таблеток (iButton), используемая для идентификации

API (Application Programming Interface) – программный интерфейс приложения

ASGI (Asynchronous Server Gateway Interface) – асинхронный шлюзовой интерфейс сервера

CRC (Cyclic Redundancy Check) – алгоритм нахождения кода ошибки для контроля целостности данных

CSS (Cascading Style Sheets) – формальный язык описания внешнего вида документа

CSV (Comma-Separated Values) – текстовый формат, предназначенный для представления табличных данных

ESP32 – серия микроконтроллеров с низким энергопотреблением, используемая в проекте как аппаратная платформа

GPIO (General Purpose Input/Output) – линии ввода/вывода общего назначения микроконтроллера

HTML (HyperText Markup Language) – стандартизированный язык разметки документов

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

IDE (Integrated Development Environment) – интегрированная среда разработки

IP (Internet Protocol) – сетевой протокол

JS (JavaScript) – мультипарадигменный язык программирования

JSON (JavaScript Object Notation) – текстовый формат обмена данными

JWT (JSON Web Token) – стандарт создания токенов

доступа OAuth2 – протокол авторизации

ORM (Object-Relational Mapping) – технология программирования, связывающая базы данных с концепциями объектно-ориентированных языков

PDF (Portable Document Format) – кроссплатформенный формат электронных документов

PoE (Power over Ethernet) – технология питания удаленных устройств по витой паре

REST (Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределенного приложения в сети

RS-485 – стандарт физического уровня интерфейса передачи данных

SDK (Software Development Kit) – комплект средств разработки программного обеспечения

SPA (Single Page Application) – одностраничное веб-приложение

UID (Unique Identifier) – уникальный идентификатор (ключа или записи)

USB (Universal Serial Bus) – универсальная последовательная шина для подключения периферийных устройств

VS Code (Visual Studio Code) – редактор исходного кода

Wi-Fi (Wireless Fidelity) – стандарт беспроводной локальной вычислительной сети

Введение

Обеспечение физической безопасности объектов – от офисных зданий и промышленных предприятий до учебных заведений и складских комплексов – в современных условиях невозможно без надёжной системы управления и контроля доступом (СКУД). Ежедневно возникают угрозы несанкционированного проникновения, хищения имущества, утечки конфиденциальной информации, а также риски, связанные с человеческим фактором. Система контроля доступа выполняет ключевые функции: ограничение входа только для авторизованных лиц, учёт рабочего времени сотрудников, логирование всех попыток прохода (успешных и неуспешных), а также интеграцию с видеонаблюдением и пожарной сигнализацией. Дополнительно система оснащается герконовым датчиком, позволяющим фиксировать момент открытия двери и её последующее закрытие, что повышает безопасность и позволяет обнаруживать несанкционированное вторжение или оставленную открытой дверь.

Традиционные решения часто представляют собой закрытые, дорогостоящие программно–аппаратные комплексы, привязанные к конкретному производителю и требующие регулярных лицензионных отчислений. Облачные СКУД создают зависимость от внешних сервисов и стабильности интернет–соединения, а хранение данных о перемещениях сотрудников на сторонних серверах несёт риски утечки чувствительной информации. Всё это подрывает предсказуемость и управляемость корпоративной системы безопасности, особенно в условиях растущих требований к цифровому суверенитету и защите персональных данных.

Чтобы избежать данных проблем, правильным решением является разработка собственной программно–аппаратной системы, которая позволит развернуть автономный контроллер доступа с клиент–серверной архитектурой, снизить зависимость от внешних поставщиков и обеспечить полный контроль над событиями входа. Разрабатываемое решение предлагает

					ПГУ 1.090301.01.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

именно такую альтернативу. Аппаратная часть на основе микроконтроллера SAMIORE ROBOT ESP32 DevKit[1] и промышленного считывателя «Считыватель–2 исп.01» взаимодействует с центральным сервером. Программное обеспечение полностью написано на языке Python, что обеспечивает простоту развёртывания, кроссплатформенность и широкие возможности дальнейшей модификации. Для администрирования и мониторинга используется веб–интерфейс, доступный из любого браузера, а при необходимости может быть создана десктопная обёртка. Такой подход позволяет развернуть контроллер на собственном оборудовании и исключить утечку данных за пределы контролируемой инфраструктуры.

Целью данной работы является разработка инструмента, автоматизирующего процесс контроля доступа в закрытой среде с полным контролем над данными. Создаваемое программно–аппаратное решение реализует клиент–серверную архитектуру с использованием микроконтроллера ESP32 и серверной части на Python с веб–интерфейсом. Такой подход обеспечивает унификацию среды управления доступом, исключает утечку данных за пределы контролируемой инфраструктуры и предоставляет администраторам привычный интерфейс, свободный от ограничений коммерческих аналогов. Таким образом, создание собственного программно–аппаратного комплекса СКУД на базе ESP32 и Python позволяет получить современный, безопасный и экономически эффективный инструмент контроля доступа.

1 Анализ предметной области

Предметной областью разработки является деятельность по организации и контролю физического доступа людей на охраняемые объекты с целью обеспечения безопасности, учёта рабочего времени и предотвращения несанкционированного проникновения. В частности, рассматривается процесс идентификации и аутентификации сотрудников и посетителей по электронному ключу Touch Memory (iButton, ТМ–таблетка) с последующим принятием решения о предоставлении или запрете доступа через исполнительное устройство (электромагнитный замок) и централизованным логированием всех попыток прохода.

Контингент пользователей включает сотрудников различных категорий (руководство, административный персонал, инженерно–технические работники, обслуживающий персонал), а также временный персонал и посетителей. Каждый пользователь получает один или несколько ключей Touch Memory с уникальным 48–битным идентификатором, записанным производителем и не подлежащим изменению. Номенклатура точек доступа охватывает входные двери в здание, двери в служебные и технические помещения, лаборатории, серверные, склады, а также калитки на пропускных пунктах. Каждая точка доступа оснащается контроллером на базе ESP32, считывателем ТМ–ключей, электромагнитным замком 12В и герконовым датчиком состояния двери.

Процесс управления доступом включает решение задачи авторизации на основе списка разрешённых ключей, хранящегося как в локальной памяти контроллера, так и на центральном сервере. Основные ограничения можно классифицировать следующим образом:

Пространственные ограничения: одна дверь (управляемая контроллером) в один момент времени может быть открыта только одним ключом. При этом система фиксирует не только факт открытия, но и фактическое состояние двери (открыта/закрыта) с помощью герконового

датчика. Для помещений с повышенными требованиями безопасности может требоваться ограниченный список лиц, имеющих доступ. Кроме того, система должна предотвращать «хвост» – проход постороннего лица вслед за авторизованным сотрудником, для чего в настройках контроллера задаётся минимальное время удержания замка в открытом состоянии.

Для контроля фактического состояния двери предусмотрен герконовый датчик. При открытии двери датчик размыкается, и ESP32 фиксирует событие «дверь открыта». Это позволяет системе отслеживать, закрыта ли дверь после прохода, а также обнаруживать попытки удержания двери. Данные о состоянии двери могут отправляться на сервер и отображаться в веб-интерфейсе

Субъектные ограничения: один ключ должен быть привязан к одному конкретному сотруднику; выдача одного ключа двум лицам не допускается, так как это нарушает аудит событий. Кроме того, существуют персональные ограничения, задаваемые вручную администратором: для некоторых сотрудников доступ может быть временно заблокирован. Некоторые ключи могут быть настроены как «мастер-ключи», дающие доступ во все помещения независимо от ограничений. Эти индивидуальные настройки вводятся через веб-интерфейс и синхронизируются с контроллерами по сети Wi-Fi.

Существующий процесс управления доступом на многих малых и средних объектах носит преимущественно ручной или полуавтоматический характер. Сотрудник службы безопасности или комендант получает заявки на выдачу ключей, заносит ID в локальную базу данных и вручную программирует контроллеры – обычно через мастер-карту или последовательное подключение к компьютеру. При этом проверка вышеперечисленных ограничений выполняется визуально или с помощью простейших формул, что не гарантирует отсутствия ошибок. Особенно остро проблема проявляется при частом изменении состава сотрудников, когда требуется оперативно обновлять права доступа на нескольких точках, и ручная настройка каждого контроллера отнимает часы и чревата конфликтами. Кроме

того, в таких системах отсутствует централизованный журнал событий – попытки прохода фиксируются только локально в памяти контроллера, и их сбор требует физического обхода всех точек.

Таким образом, предметная область характеризуется высокой структурной сложностью, наличием большого числа разнородных сущностей, необходимостью оперативного обновления прав доступа и централизованного логирования, что делает актуальной разработку автоматизированной программно–аппаратной СКУД с возможностью удалённого управления через веб–интерфейс и единой базой данных на сервере.

1.1 Анализ существующих аналогов

Для обоснования необходимости разработки собственного программно–аппаратного решения проведём сравнительный анализ существующих на рынке систем контроля доступа, которые могут быть использованы для автоматизации управления доступом в малых и средних организациях. Рассмотрим аналоги.

Первый аналог – «Sigur».



Рисунок 1 – Обзор аналога «Sigur»

Данный продукт представляет собой профессиональную российскую систему контроля доступа, предлагаемую как в локальном, так и в облачном исполнении. Система включает широкий спектр контроллеров (Sigur S2, S3, S4), мощное серверное ПО (Sigur Client) и веб–интерфейс для администрирования.

Основные функциональные возможности и особенности: поддержка различных типов считывателей (Wiegand 26/34, Touch Memory, EM–Marine, RFID); управление до двух точек на одном контроллере; связь с ПК по Ethernet или RS–485; локальная память на 30 000 событий; централизованное логирование; разграничение прав доступа по временным зонам.

Достоинства: высокая надёжность и отказоустойчивость; развитая логика управления доступом; интеграция с учётом рабочего времени; мощное серверное ПО с веб–интерфейсом; поддержка большого количества считывателей и замков.

Недостатки: закрытый протокол взаимодействия между контроллером и сервером; высокая стоимость лицензий; привязка к оборудованию вендора; невозможность самостоятельной модификации или интеграции с несертифицированным оборудованием; для малого объекта система избыточна как по цене, так и по функционалу.

Вывод по продукту Sigur: система является мощным профессиональным инструментом для крупных предприятий с высокими требованиями к безопасности и готовностью финансировать лицензии. Однако она экономически неэффективна и технически избыточна для решения задач малой организации или образовательного учреждения, а также не позволяет вносить изменения в логику работы без привлечения разработчика.

Второй аналог – «RusGuard».

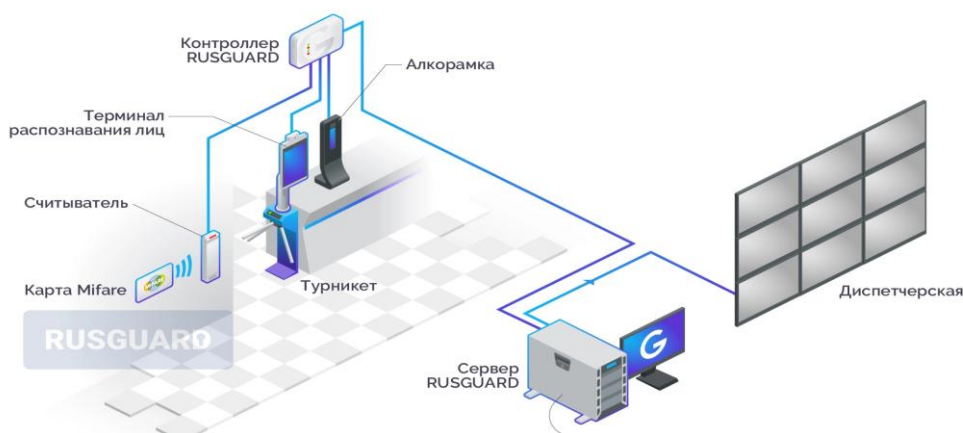


Рисунок 2 – Обзор аналога «RusGuard»

RusGuard – это отечественная платформа, сочетающая управление доступом с системами видеонаблюдения и учёта рабочего времени. Продукт существует в виде отдельных контроллеров и серверного ПО, которое может работать под управлением Windows. Система ориентирована на крупные объекты с развитой инфраструктурой безопасности.

Основные функциональные возможности и особенности: поддержка PoE; память на 50 000 событий; управление до двух дверей; поддержка считывателей Wiegand, Touch Memory; встроенный веб-сервер для базовой настройки; интеграция с IP-камерами.

Достоинства: единая экосистема, объединяющая контроль доступа и видеонаблюдение; гибкая система отчётов; мощное серверное ПО с широкими возможностями настройки; поддержка PoE упрощает монтаж; возможность масштабирования до сотен точек доступа.

Недостатки: перегруженный интерфейс программного обеспечения; высокая ресурсоёмкость сервера; платные обновления и ежегодная техническая поддержка; закрытость протоколов; невозможность использования с несертифицированным оборудованием; сложность настройки для неспециалиста.

Вывод по продукту RusGuard: система подходит для крупных предприятий, где уже используется видеонаблюдение того же производителя, и есть штатный администратор для обслуживания. Однако для малого объекта или учебного учреждения она избыточна, сложна в настройке и имеет высокую стоимость владения. Отсутствие открытого API и возможности самостоятельной доработки делает её непригодной для дипломного проекта, предполагающего изучение и модификацию системы.

Третий аналог – «Proxway (LarTech)»



Рисунок 3 – Обзор аналога «Proxway (LarTech)»

Proxway – это продукт компании LarTech (Китай), широко представленный на российском рынке в среднем ценовом сегменте. Система включает контроллеры (PW-2000, PW-400 mod.23), ПО Proxway Access Manager, а также SDK для разработчиков. Отличается относительно открытой архитектурой.

Основные функциональные возможности и особенности: поддержка считывателей Wiegand, Touch Memory, RFID; Ethernet или RS-485 для связи; память на 20 000 событий; поддержка до 2–4 дверей; возможность интеграции через SDK на C++/C#.

Достоинства: доступная стоимость оборудования; наличие SDK и документации для сторонних разработчиков; возможность интеграции с другими системами через Wiegand; веб-интерфейс для базового администрирования; относительно низкий порог входа.

Недостатки: устаревший дизайн программного обеспечения; ограниченный функционал тонкой настройки временных зон и уровней доступа; для использования SDK требуется знание C++/C# и опыт работы с низкоуровневыми протоколами; документация на русском языке неполная;

при обновлении прошивки контроллера возможна потеря совместимости с собственными разработками.

Вывод по продукту Proxway: система привлекательна своей ценой и наличием SDK, что даёт возможность для самостоятельной интеграции. Однако для полноценной адаптации под нужды конкретной организации потребуется квалифицированный разработчик, разбирающийся в C++ и протоколах. Отсутствие готового веб-интерфейса с современным дизайном и ограниченная поддержка со стороны вендора делают систему не лучшим выбором для образовательного или малого предприятия, которое не может позволить себе штатного программиста.

Вывод по результатам анализа аналогов.

Существующие на рынке программно-аппаратные решения в области СКУД можно разделить на следующие категории:

- профессиональные и дорогие (Sigur, RusGuard) – обладают широкими возможностями, но имеют высокую стоимость лицензий, закрытую архитектуру и требуют значительных затрат на внедрение и сопровождение;
- бюджетные, но ограниченные (Proxway) – доступны по цене, но предоставляют лишь базовый функционал и устаревший интерфейс, а для расширения требуют глубоких знаний в программировании.

Ни одно из рассмотренных решений не удовлетворяет совокупности требований, предъявляемых к моему проекту: открытость и возможность самостоятельной модификации, централизованное логирование, удобный веб-интерфейс, возможность работы в локальной сети без обязательного подключения к интернету, простота развёртывания.

Это подтверждает целесообразность разработки собственного программно-аппаратного комплекса СКУД на базе ESP32, ТМ-считывателя, электромагнитного замка и серверной части на Python с веб-интерфейсом, который будет лишён указанных недостатков и сможет эффективно использоваться в малых организациях, учебных заведениях и на объектах с особыми требованиями к безопасности.

1.2 Анализ актуальности разработки

Обеспечение физической безопасности объектов различного назначения (офисных зданий, промышленных предприятий, учебных заведений и складских комплексов) в современных условиях невозможно без надежной автоматизации процессов контроля доступа. Система управления и контроля доступом выполняет критически важные функции: ограничение входа только для авторизованных лиц, учет рабочего времени, централизованное логирование всех попыток прохода и интеграция с другими системами безопасности.

Актуальность темы обусловлена следующими ключевыми факторами:

- Недостатки существующих ручных и полуавтоматических процессов. На многих малых и средних объектах управление доступом до сих пор осуществляется вручную или с использованием примитивных средств. Такой подход подвержен высокому риску человеческого фактора: возможны ошибки при выдаче ключей, дублирование идентификаторов, несвоевременная блокировка прав уволенных сотрудников, а также отсутствие централизованного журнала событий, что делает аудит безопасности неэффективным;

- Экономическая и техническая избыточность коммерческих решений. Существующие на рынке профессиональные системы (такие как Sigur или RusGuard) обладают широким функционалом, но характеризуются высокой стоимостью лицензий, закрытой архитектурой, жесткой привязкой к оборудованию вендора и высокой ресурсоемкостью серверного ПО. Для малых организаций или учебных заведений такие системы являются экономически неэффективными и избыточными. Бюджетные аналоги часто имеют устаревший интерфейс, ограниченный функционал тонкой настройки и требуют от персонала глубоких знаний в программировании для интеграции;

- Риски использования облачных СКУД. Современные облачные решения создают критическую зависимость от стабильности интернет-соединения и внешних сервисов. Хранение чувствительных данных о

перемещениях сотрудников на сторонних серверах несет прямые риски утечки конфиденциальной информации, что противоречит растущим требованиям к цифровому суверенитету и защите персональных данных (в том числе в рамках законодательства РФ);

– Потребность в открытых и адаптируемых решениях. Отсутствует доступный программно-аппаратный комплекс, который сочетал бы в себе низкую стоимость внедрения, открытую архитектуру для самостоятельной модификации, современный веб-интерфейс и возможность работы исключительно в локальной сети организации без обязательного выхода в глобальный интернет.

Предлагаемое решение и его значимость:

Разработка собственной программно-аппаратной СКУД на базе микроконтроллера ESP32 и серверной части на языке Python позволяет устранить вышеуказанные противоречия. Использование доступной аппаратной платформы в связке с высокопроизводительным асинхронным фреймворком FastAPI и легковесной СУБД SQLite обеспечивает:

- отказ от дорогостоящих лицензий и проприетарного оборудования;
- полное хранение базы данных и журналов событий на локальном сервере организации, исключающее утечку данных за пределы контролируемой инфраструктуры;
- наличие современного веб-интерфейса для удаленного управления правами доступа, мониторинга событий в реальном времени и оперативного реагирования на инциденты;
- кроссплатформенность Python и модульная архитектура позволяют легко адаптировать систему под специфические требования конкретного объекта силами одного IT-специалиста.

Таким образом, актуальность разработки подтверждается острой потребностью малого и среднего бизнеса, а также образовательных учреждений в недорогом, безопасном, автономном и легко модифицируемом инструменте

контроля доступа, лишенном недостатков как ручного управления, так и избыточных коммерческих аналогов.

1.3 Постановка задачи

Требуется разработать программно–аппаратное решение системы управления и контроля доступом, включающее микроконтроллерный блок управления точкой прохода на базе ESP32 и программное обеспечение на языке Python. Ниже приведён паспорт системы, перечень устройств и функциональные задачи. Для достижения поставленной цели решаются следующие задачи, разделённые по компонентам приложения.

Аппаратная часть должна обеспечивать:

- взаимодействие со считывателем ТМ–ключей по протоколу 1–Wire[2];
- управление электромагнитным замком через реле;
- светодиодную и звуковую индикацию;
- подключение к Wi-Fi и обмен данными с сервером по HTTP (REST API);
- обработку команд от сервера (открыть дверь, блокировать, обновить белый список);
- опрос герконового датчика для контроля состояния двери (открыта/закрыта) и отправка соответствующего события на сервер;
- отправку событий на сервер в реальном времени.

Серверная часть на Python должна реализовывать:

- приём и обработку HTTP–запросов от контроллера (FastAPI[3]);
- управление базой данных сотрудников и ключей (SQLite[4]);
- ведение централизованного журнала событий с фильтрацией и экспортом;
- маршрутизацию команд от веб–интерфейса к контроллеру;
- мониторинг состояния системы.

Клиентская часть (веб–интерфейс) должна включать:

					ПГУ 1.090301.01.001 ПЗ	Лист
						18
Изм.	Лист	№ докум.	Подпись	Дата		

- одностраничное приложение (HTML/CSS/JS);
 - модуль управления ключами сотрудников (регистрация, блокировка, привязка);
 - отображение плана объекта с состоянием двери и ручным управлением;
- модуль отчётов по событиям доступа с экспортом в PDF;
- возможность упаковки в десктопную обёртку (pywebview).

Инструменты разработки:

- Arduino IDE для ESP32;
- Python 3.10+, FastAPI, SQLite для сервера;
- PyCharm для разработки ПО.

В таблице 1 представлен паспорт –программно–аппаратного решения.

Таблица 1 – Сравнительная таблица аналогов

Система	Преимущества	Недостатки
Sigur	Высокая надёжность, развитая логика, интеграция с учётом рабочего времени	Закрытый протокол, высокая стоимость лицензий, привязка к вендору
RusGuard	Единая экосистема с видео, мощный софт, гибкие отчёты	Перегруженный интерфейс, высокая ресурсоёмкость, платные обновления
Proxway	Доступная стоимость, открытость SDK	Устаревший дизайн ПО, ограниченная настройка временных зон

На рисунках 4-5 приведены диаграмма вариантов использования



Рисунок 4 – Диаграмма вариантов использования веб-интерфейса

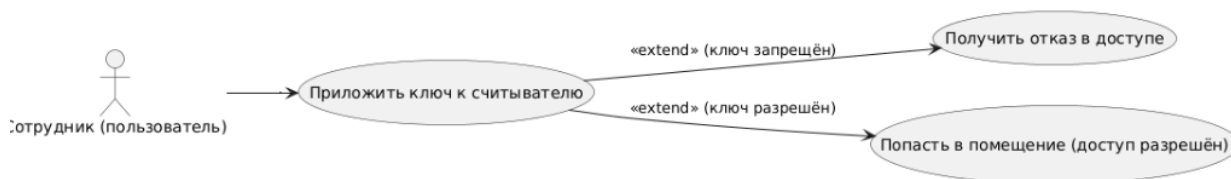


Рисунок 5 – Диаграмма вариантов использования аппаратной части

Ожидаемый результат:

Работоспособное программно–аппаратное решение СКУД, включающее:

- микроконтроллер ESP32 с прошивкой на C++ (Arduino IDE), обеспечивающий считывание ТМ–ключей, управление электромагнитным замком, отправку событий на сервер по HTTP;
- серверную часть на Python (FastAPI + SQLite) с веб–интерфейсом, позволяющую администратору просматривать журнал доступа в реальном времени, добавлять и блокировать ключи, удалённо открывать дверь;

– локальную сетевую инфраструктуру (Wi-Fi), обеспечивающую связь контроллера с сервером без необходимости подключения к глобальному интернету.

Разработанная система позволит автоматизировать процесс контроля доступа в малых организациях, учебных заведениях и на объектах с особыми требованиями к информационной безопасности, сократить временные затраты на выдачу и отзыв ключей, а также обеспечить централизованное хранение и анализ всех попыток прохода.

					ПГУ 1.090301.01.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		21

2 Выбор технологий и средств разработки

Выбор технологического стека является критически важным этапом проектирования программно–аппаратной системы управления доступом, развёртываемой в масштабе организации. Разрабатываемая СКУД должна обслуживать десятки или сотни сотрудников и посетителей, обеспечивать обработку событий доступа с гарантией целостности и конфиденциальности данных при одновременной работе нескольких администраторов и контроллеров. В настоящем разделе обосновывается выбор языка программирования серверной части, системы управления базами данных, инструментов разработки и аппаратной платформы контроллера с учётом масштаба и требований к надёжности системы.

2.1 Выбор языка программирования

Для реализации серверной части разрабатываемой СКУД выбран язык Python в сочетании с высокопроизводительным асинхронным веб–фреймворком FastAPI. Данный выбор обусловлен необходимостью создания масштабируемой, легко поддерживаемой и безопасной системы, способной обрабатывать конкурентные запросы от нескольких контроллеров ESP32 (до 10–20 штук) и одновременно обслуживать веб–интерфейс администратора.

Высокая производительность при асинхронной обработке. FastAPI работает поверх ASGI–сервера (Uvicorn)[5] и использует нативный асинхронный цикл событий Python (asyncio). Это позволяет эффективно обслуживать десятки одновременных HTTP–запросов от контроллеров и администратора при минимальном потреблении оперативной памяти. Для системы, где в пиковые моменты (например, при утреннем проходе сотрудников) ESP32 могут отправлять события каждые несколько секунд, асинхронная модель предпочтительнее классической многопоточной (как в Flask без асинхронности) с точки зрения утилизации ресурсов.

Лёгкость реализации REST API. FastAPI предоставляет декларативный способ описания эндпоинтов с автоматической валидацией входных данных через Pydantic[6]. Для СКУД требуются всего несколько эндпоинтов: POST /log (приём события от ESP32), GET /logs (получение журнала), GET / (веб-интерфейс). Быстрота прототипирования позволяет за считанные минуты реализовать и отладить API, а автоматическая генерация документации Swagger UI упрощает тестирование.

Богатая экосистема для работы с базами данных и JSON. Python имеет встроенную поддержку SQLite (модуль sqlite3), а также мощные ORM (SQLAlchemy)[7] для работы с реляционными базами данных. Формат JSON (основной для обмена данными с ESP32) обрабатывается стандартной библиотекой json, что исключает необходимость подключения дополнительных тяжёлых зависимостей.

Простота интеграции с аппаратной частью. На стороне ESP32 используется C++, но сервер на Python не требует низкоуровневого взаимодействия с железом – достаточно принимать HTTP-запросы. Python-сервер можно легко развернуть на любом компьютере в локальной сети, а его код прост для понимания и модификации силами одного разработчика (дипломника).

Безопасность и авторизация. FastAPI изначально поддерживает OAuth2 и JWT, но для локальной СКУД в пределах организации достаточно простой аутентификации администратора по паролю. Тем не менее, фреймворк предоставляет все необходимые инструменты для внедрения защиты от несанкционированного доступа в будущем.

Горизонтальная масштабируемость. Благодаря stateless-архитектуре экземпляры FastAPI могут быть легко развернуты за балансировщиком нагрузки при необходимости обслуживания большого числа контроллеров. Хотя для дипломного проекта это избыточно, сам стек позволяет масштабироваться в будущем.

Таким образом, выбор Python + FastAPI представляет собой оптимальный вариант, совмещающий производительность, скорость разработки и простоту поддержки для серверной части СКУД.

2.2 Выбор СУБД

В качестве системы управления базами данных для разрабатываемой СКУД выбрана SQLite. Этот выбор продиктован требованиями к лёгкости развёртывания, достаточной надёжности для хранения журнала событий и возможности работы без выделенного сервера БД в условиях малой организации или образовательного учреждения.

Обоснование выбора SQLite для СКУД

Строгая согласованность данных при конкурентном доступе. В системе контроля доступа записи в журнал событий (успешные и неудачные попытки прохода) могут поступать одновременно от нескольких контроллеров ESP32, а администратор в это же время может просматривать журнал или добавлять новые ключи. SQLite поддерживает транзакции с уровнем изоляции **SERIALIZABLE**, что гарантирует, что одновременные записи не приведут к повреждению базы данных или появлению «рваных» записей. Для типовой нагрузки (до 10 контроллеров, отправляющих события с интервалом не менее 500 мс) SQLite обеспечивает бесконфликтную работу.

Эффективная работа с JSON. Контроллер ESP32 отправляет на сервер события в формате JSON (`{"uid": "...", "status": "GRANTED"}`). SQLite начиная с версии 3.9 поддерживает функции работы с JSON, что позволяет хранить пришедшие данные в текстовом поле, а при необходимости – извлекать отдельные атрибуты. Однако для простоты и производительности события можно хранить в отдельных столбцах (`uid`, `status`, `timestamp`), что является более предпочтительным для СКУД.

Лёгкость развёртывания и отсутствие отдельного сервера. В отличие от PostgreSQL, MySQL или MS SQL, SQLite не требует установки, настройки и администрирования отдельного сервера баз данных. База данных представляет

собой один файл (skud.db), который может храниться в папке с серверным приложением. Это упрощает развёртывание СКУД на обычном компьютере (даже на ноутбуке диспетчера) и не требует выделенного ИТ-специалиста для обслуживания БД.

Достаточная производительность для задач СКУД. В журнале событий за год может накопиться несколько десятков тысяч записей (например, $100 \text{ сотрудников} \times 2 \text{ прохода в день} \times 200 \text{ рабочих дней} = 40\,000 \text{ записей}$). SQLite легко справляется с таким объёмом: выборка последних 100 записей для веб-интерфейса выполняется за миллисекунды. При типовом количестве одновременных соединений нагрузка не достигает пределов производительности SQLite.

Интеграция с Python-стеком. Модуль sqlite3 входит в стандартную поставку Python, поэтому не требуется установка дополнительных драйверов. SQLAlchemy (ORM) также поддерживает SQLite «из коробки». Асинхронные библиотеки (aiosqlite) позволяют интегрировать SQLite с FastAPI без блокировки цикла событий, что важно для приёма параллельных запросов от контроллеров.

Отсутствие лицензионных затрат. SQLite распространяется под лицензией Public Domain, что для государственного учреждения или малого предприятия с ограниченным бюджетом является существенным преимуществом.

Резервное копирование и переносимость. Файл skud.db можно просто скопировать на флешку или в облачное хранилище, а при необходимости – восстановить систему на другом компьютере, перекопировав файл. Это не требует сложных процедур дампа и восстановления, характерных для клиент-серверных СУБД.

Альтернативы (PostgreSQL, MySQL, MS SQL) избыточны для данного проекта: они требуют установки и настройки отдельного сервера, выделенных ресурсов (память, диск), а также квалифицированного администратора. Для

СКУД, обслуживающей до 1000 сотрудников и до 10 точек доступа, такая мощность не нужна и только усложнит эксплуатацию.

Таким образом, SQLite является СУБД, сочетающей в себе достаточную надёжность транзакций, встроенную поддержку в Python, простоту развёртывания и нулевую стоимость владения, что делает её оптимальным выбором для разрабатываемой СКУД.

2.3 Выбор среды разработки

Разработка программно–аппаратной системы контроля доступа включает создание прошивки для микроконтроллера ESP32 (на C++ в Arduino IDE) и серверной части на Python (FastAPI) с веб–интерфейсом администратора. Для обеспечения единой, гибкой и производительной среды разработки серверной части был выбран Visual Studio Code (VS Code) – бесплатный, кроссплатформенный редактор кода с широкими возможностями расширения. Для программирования ESP32 используется Arduino IDE как наиболее простой и надёжный инструмент. Данный выбор обусловлен стремлением унифицировать рабочий процесс, снизить порог входа для последующего сопровождения проекта и эффективно использовать ресурсы компьютера.

Обоснование выбора Visual Studio Code для серверной части

Единая среда для разработки на Python и веб–интерфейса. Использование VS Code для backend (Python) и frontend (HTML/CSS/JS) исключает необходимость переключения между разными редакторами. Это ускоряет навигацию по проекту, упрощает отладку сквозных сценариев (например, тестирование API и отображение данных в таблице) и снижает когнитивную нагрузку на разработчика. В рамках дипломного проекта, где кодовая база невелика, унификация инструмента является значительным преимуществом.

Мощная экосистема расширений. VS Code обладает крупнейшим маркетплейсом расширений, позволяющих превратить редактор в полноценную IDE под конкретные задачи. Для серверной части на Python используются:

– Python – поддержка IntelliSense, отладки, тестирования, виртуальных окружений;

– Pylance – быстрый и точный сервер языка для проверки типов и автодополнения, незаменимый при работе с аннотациями типов Pydantic и FastAPI;

– REST Client – выполнение HTTP-запросов непосредственно из редактора для тестирования API (например, отправка POST-запроса на /log);

Для веб-интерфейса (простые HTML/JS):

– Prettier и ESLint – автоматическое форматирование и проверка синтаксиса JavaScript;

– Live Server – запуск локального сервера для предпросмотра HTML-страницы в реальном времени.

Все расширения работают в едином окне, не конфликтуя друг с другом.

Встроенная поддержка отладки для Python. VS Code предоставляет унифицированный интерфейс отладки. Можно запустить серверную часть через конфигурацию Python: FastAPI, устанавливать точки останова в коде и исследовать состояние переменных. Для отладки достаточно добавить конфигурацию с типом python и указать "request": "launch", "module": "uvicorn". Это позволяет эффективно исследовать проблемы приёма событий от ESP32.

Работа с базами данных. Через расширение SQLTools или встроенную поддержку SQLite VS Code позволяет подключаться к файлу базы данных skud.db, выполнять произвольные SQL-запросы, просматривать структуру таблиц и редактировать данные. Это удобно для оперативной проверки содержимого журнала событий без установки отдельного инструмента.

Лёгковесность и низкое потребление ресурсов. По сравнению с полнофункциональными IDE VS Code потребляет значительно меньше оперативной памяти и быстрее запускается.

Бесплатность и открытость. VS Code распространяется под свободной лицензией MIT и не требует никаких лицензионных отчислений. Это делает его

идеальным выбором для дипломного проекта, выполняемого без финансирования.

Кроссплатформенность. VS Code работает на Windows, macOS и Linux, что позволяет разрабатывать сервер на любой операционной системе, доступной студенту.

Обоснование выбора Arduino IDE для аппаратной части.

Простота настройки. Поддержка ESP32 добавляется через менеджер плат в два клика.

Библиотеки. Через менеджер библиотек устанавливаются OneWire, WiFi, HTTPClient, ArduinoJson – всё необходимое для работы с ТМ-считывателем и отправки HTTP-запросов.

Монитор порта. Встроенный инструмент для просмотра отладочных сообщений из ESP32, критически важный при настройке чтения ключей.

2.4 Настройка среды разработки

Необходимо установить среду разработки Arduino IDE.

Сайт Arduino IDE [8] представлен на рисунке 6.

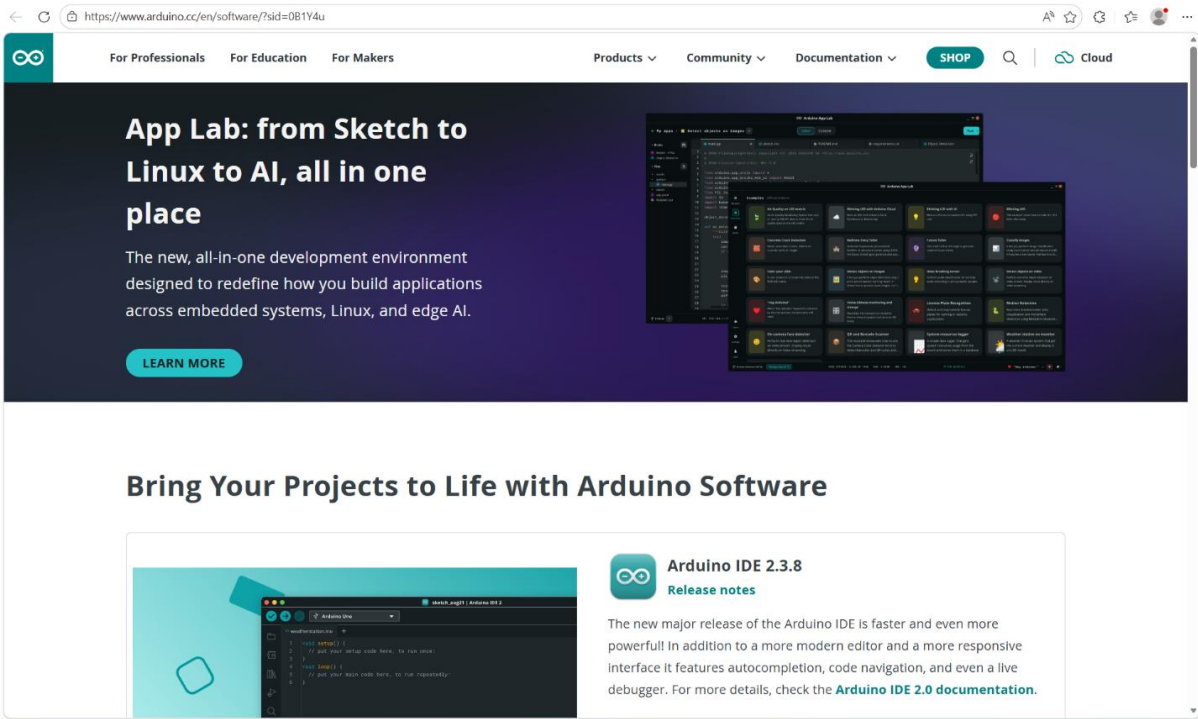


Рисунок 6 – Главная страница сайта Arduino IDE

Страница загрузки Arduino IDE содержит подробное описание функциональных возможностей среды, технические требования к системе, архив предыдущих версий и элементы управления для загрузки установщика. Для начала работы нажал кнопку скачивания.

Bring Your Projects to Life with Arduino Software

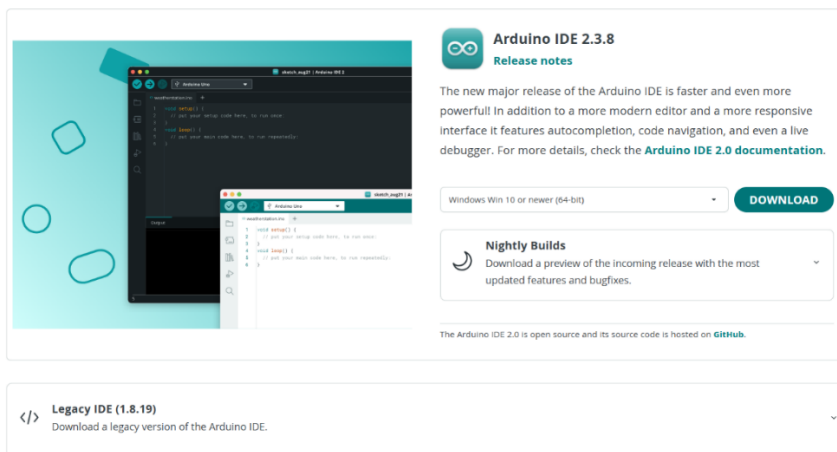


Рисунок 7 – Кнопка загрузки установщика

После перехода на страницу загрузки начинается скачивание исполняемого файла, который затем запустил.

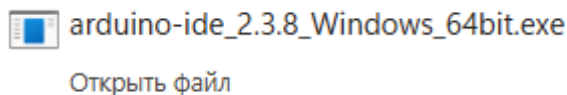


Рисунок 8 – Скачивание установщика

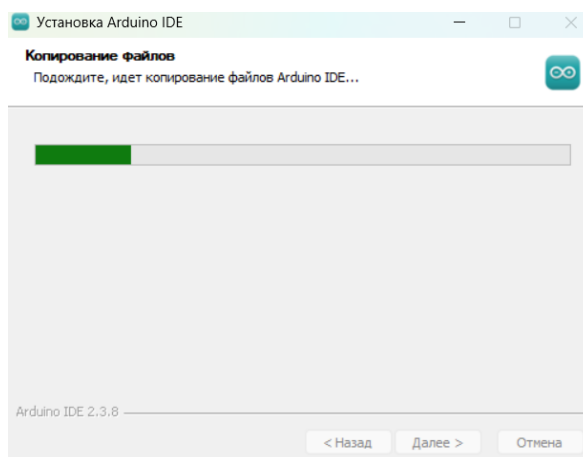


Рисунок 9 – Запуск установщика

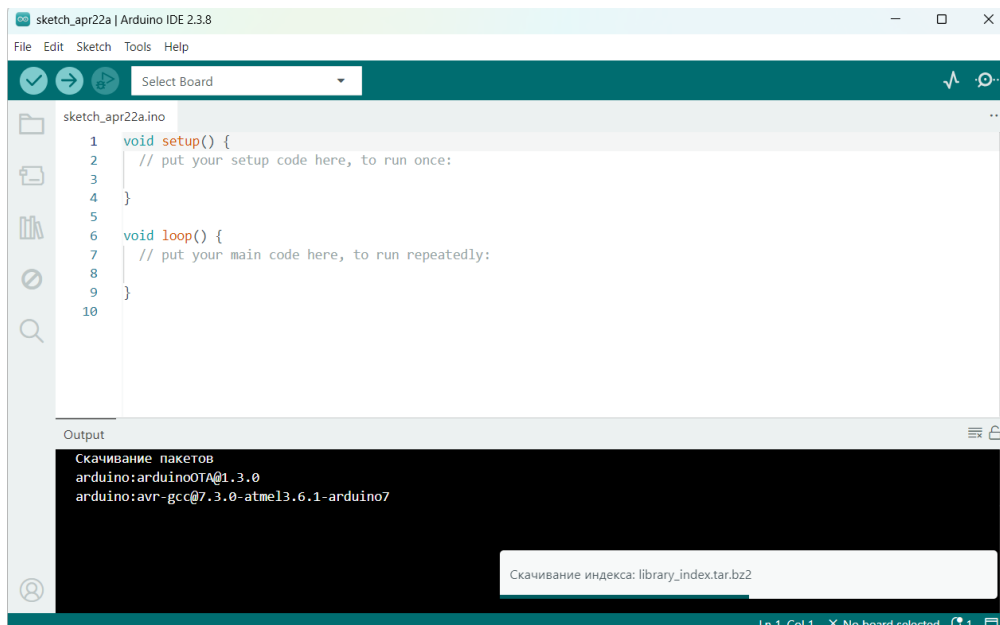


Рисунок 10 – Начальное окно Arduino IDE

Для разработки программ на базе ESP32 добавил поддержку этой платы. Для этого используется меню «Файл → Настройки». В поле «Дополнительные ссылки для менеджера плат» вставил ссылку на пакет ESP32. Затем через «Инструменты → Плата → Менеджер плат» установил пакет esp32 by Espressif Systems. Результат успешной установки показан на рисунке 11.

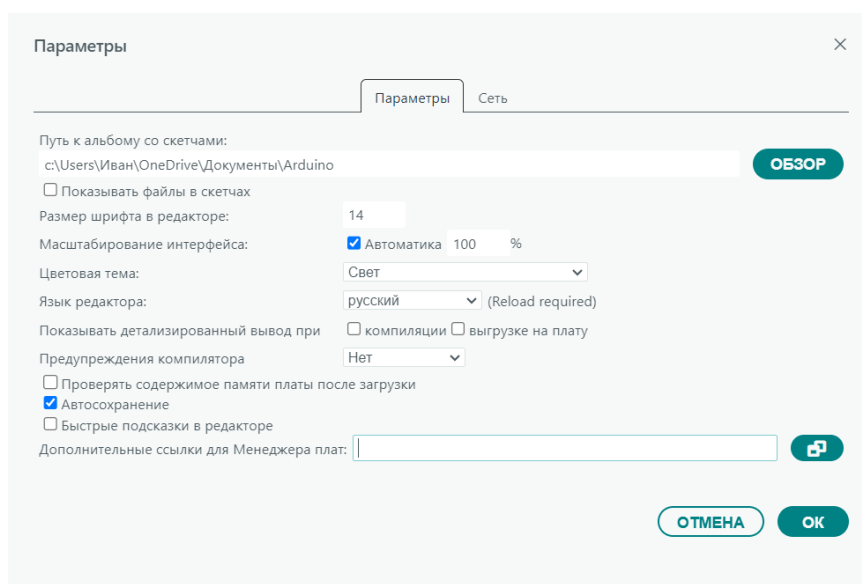


Рисунок 11– Меню настроек

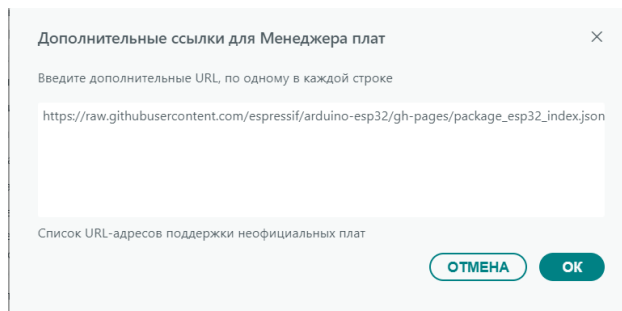


Рисунок 11– Добавление ссылки на ESP32

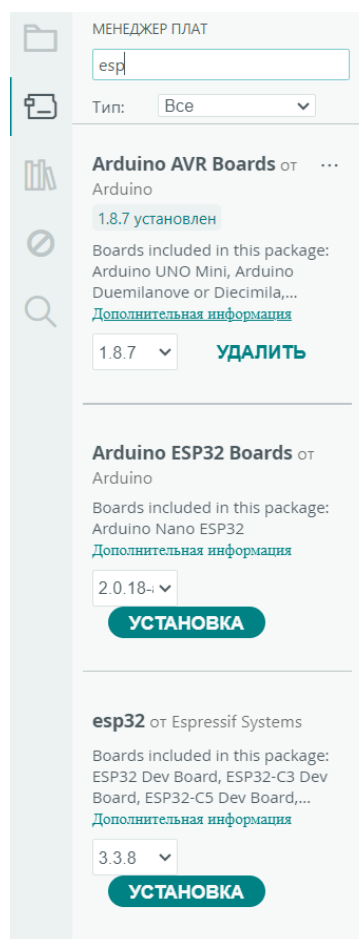


Рисунок 13 – Установка пакета ESP32 в менеджере плат

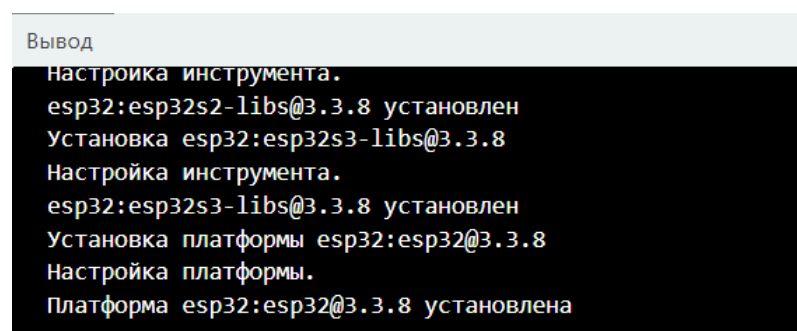


Рисунок 14 – Пакет ESP32 установлен

Помимо настройки IDE требуется подключение самой платы ESP32 к компьютеру через USB-кабель. В процессе выбора платы в меню «Инструменты → Плата» выбрал ESP32 Dev Module , а в меню «Инструменты → Порт» – соответствующий COM-порт.

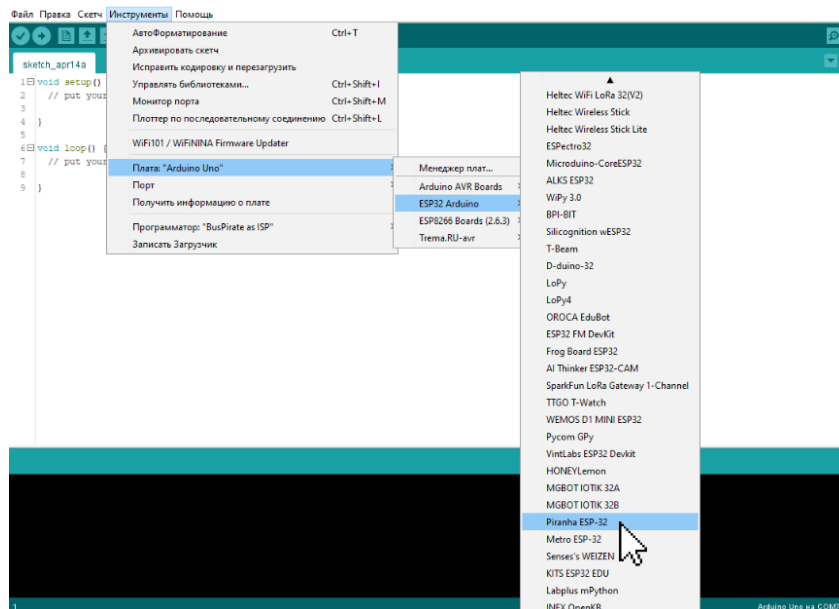


Рисунок 15 – Выбор платы ESP32 Dev Module

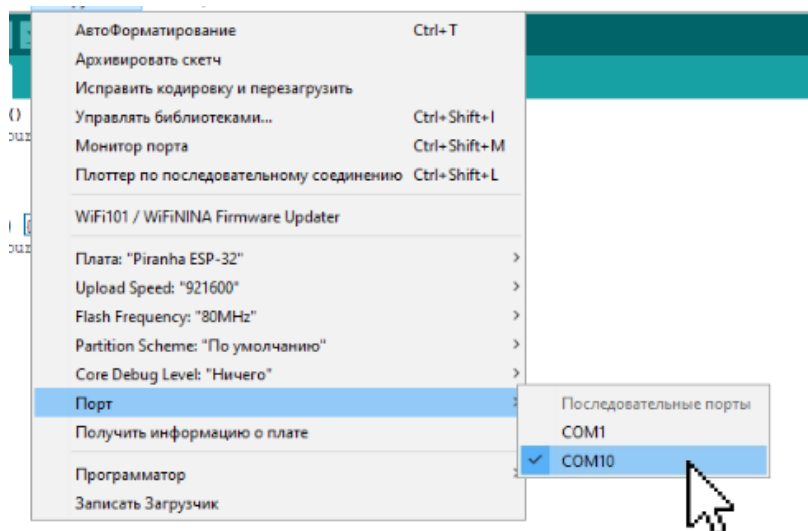


Рисунок 16 – Выбор порта

По завершении установки создал тестовый скетч, «Файл → Примеры → 01.Basics → Blink». Если среда разработки не обнаружила плату ESP32, необходимо проверить правильность выбора платы и порта.

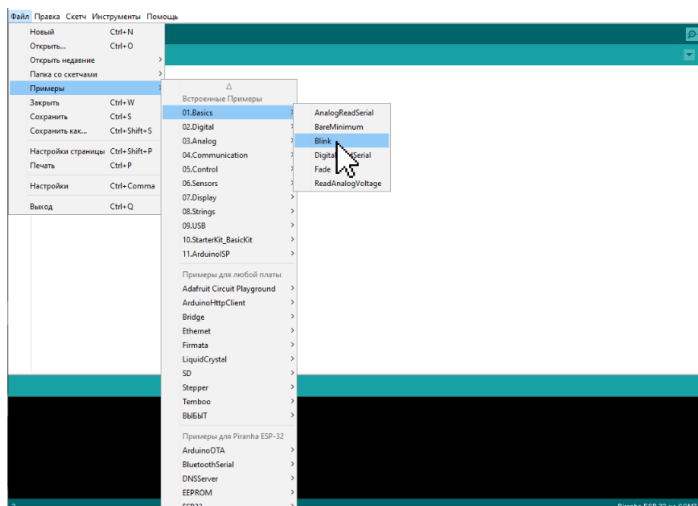


Рисунок 17– Открытие примера Blink

Для проверки корректности настройки среды выполним сборку и загрузку прошивки. Для этого в панели инструментов нажал кнопку «Загрузка» (стрелка вправо). Появление сообщения «Done uploading» в логе и мигающий встроенный светодиод на плате ESP32 подтверждает корректную настройку среды разработки.

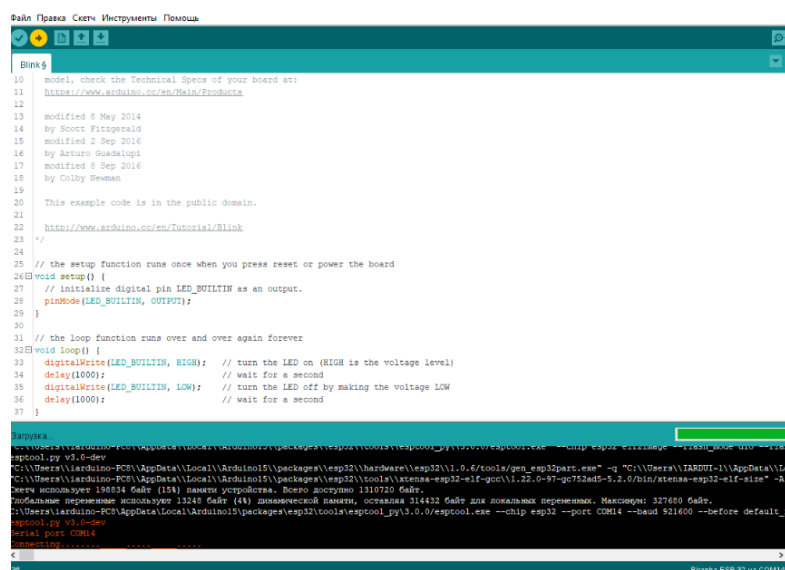


Рисунок 18 – Загрузка прошивки на плату

Для разработки серверной части на Python выбрана среда Visual Studio Code. Сайт Visual Studio Code [9] представлен на рисунке 18.

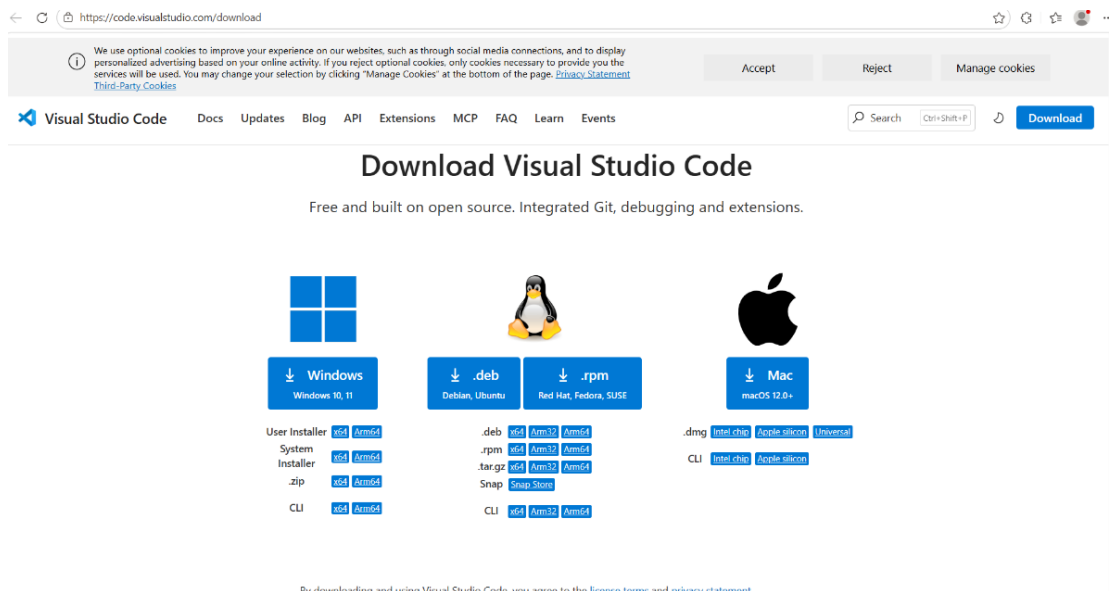


Рисунок 19– Главная страница сайта Visual Studio Code

Страница загрузки Visual Studio Code содержит подробное описание функциональных возможностей среды, технические требования к системе, архив предыдущих версий и элементы управления для загрузки установщика. Для начала работы необходимо нажать кнопку скачивания.

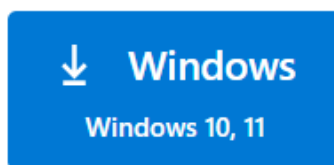


Рисунок 20 – Кнопка загрузки установщика

После перехода на страницу загрузки начинается скачивание исполняемого файла, который затем необходимо запустить.

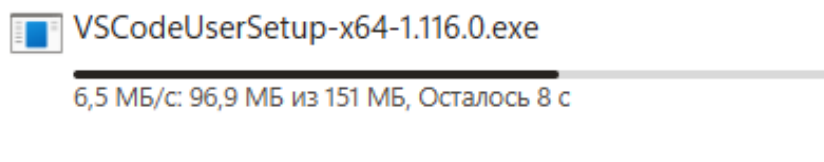


Рисунок 21 – Скачивание установщика

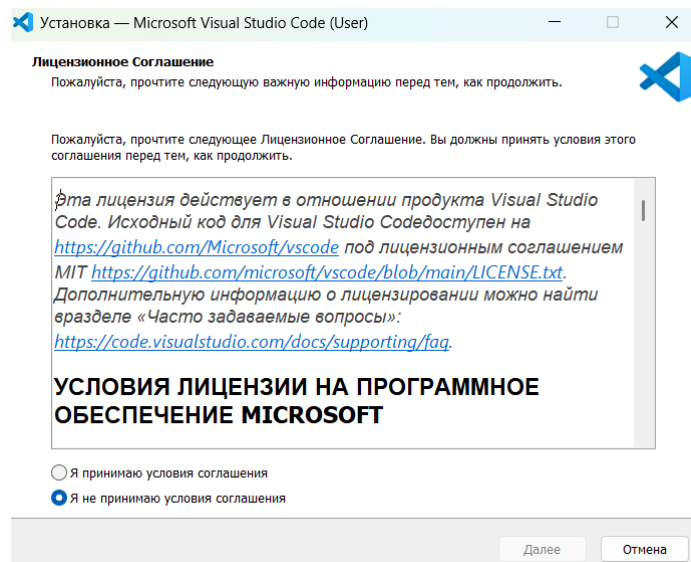


Рисунок 22 – Запуск установщика

В окне выбора компонентов отметил галочки «Add to PATH» и «Create a desktop icon».

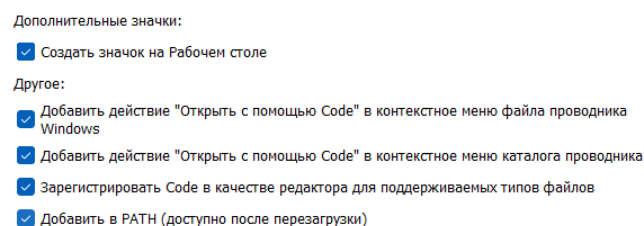


Рисунок 23 – Выбор компонентов в установщике Visual Studio Code

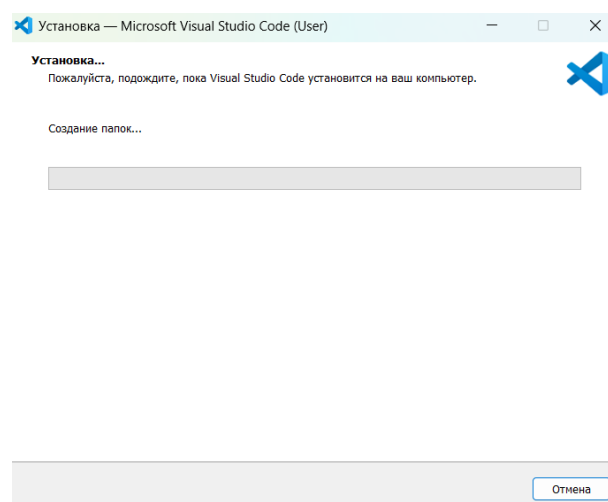


Рисунок 24–Установка Visual Studio Code

Для разработки программ на Python установил соответствующее расширение. Для этого использую меню «Расширения». В окне управления расширениями нашел и установил «Python» от Microsoft. Результат успешной установки показан на рисунке 24.

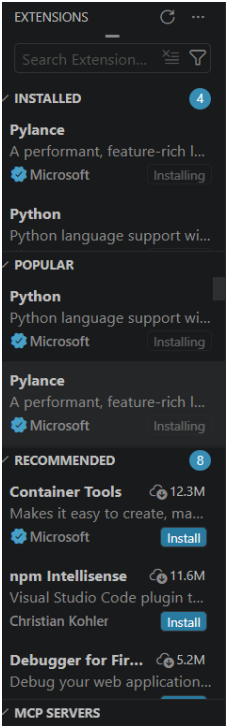


Рисунок 25 – Меню расширений

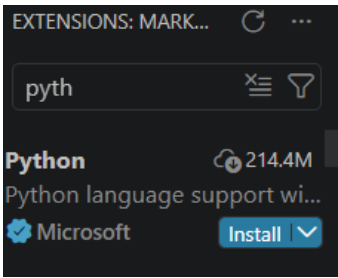


Рисунок 26– Установка расширения Python

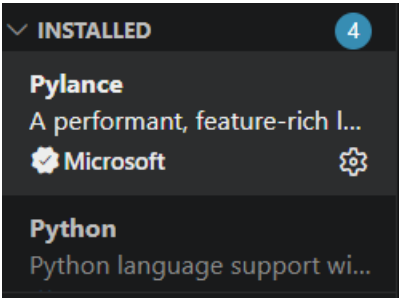


Рисунок 27 – Расширение Python установлено

Помимо настройки IDE требуется установка самого интерпретатора Python с официального сайта [10]. В процессе установки отметил галочку «Add Python to PATH».

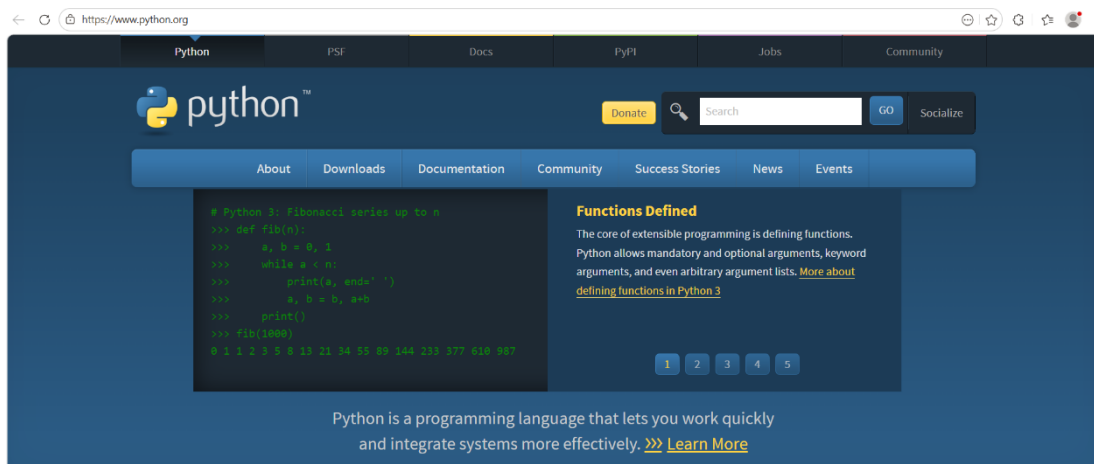


Рисунок 28 – Официальный сайт Python

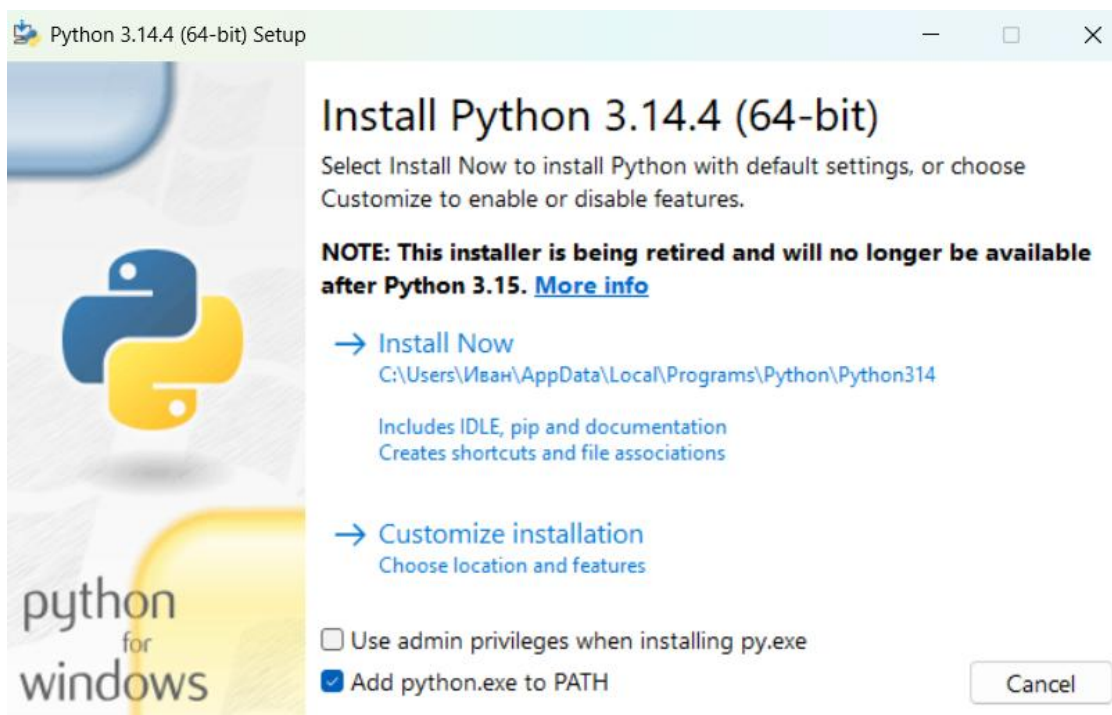


Рисунок 29 – Выбор опции добавления Python в PATH

На этапе создания проекта открываю папку проекта в Visual Studio Code. Затем через встроенный терминал создаю виртуальное окружение и устанавливаю необходимые пакеты: fastapi, uvicorn, sqlalchemy, aiosqlite.

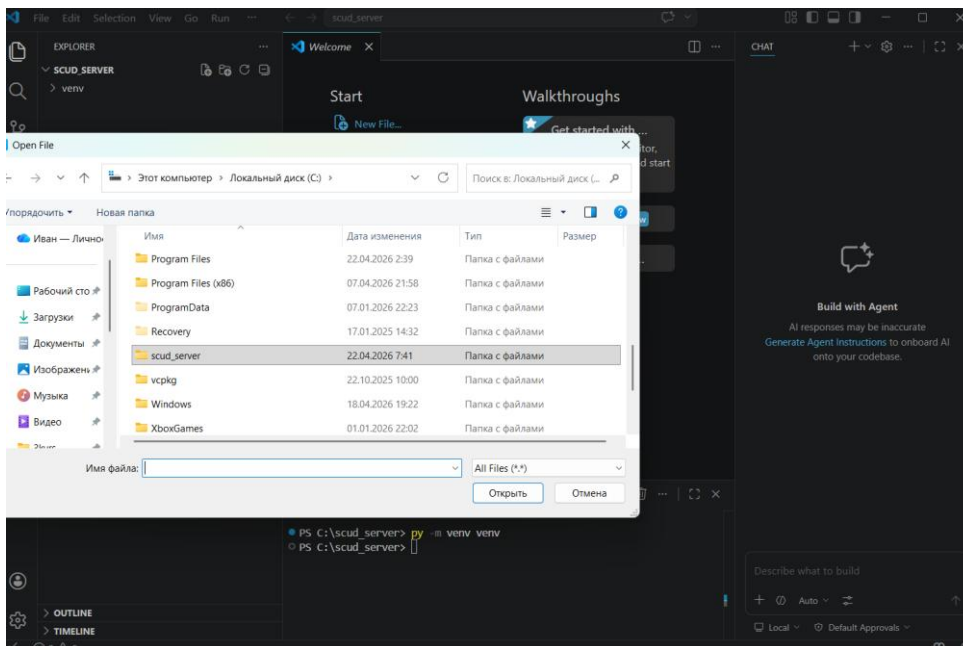


Рисунок 30 – Открытие папки проекта



Рисунок 31 – Создание виртуального окружения в терминале

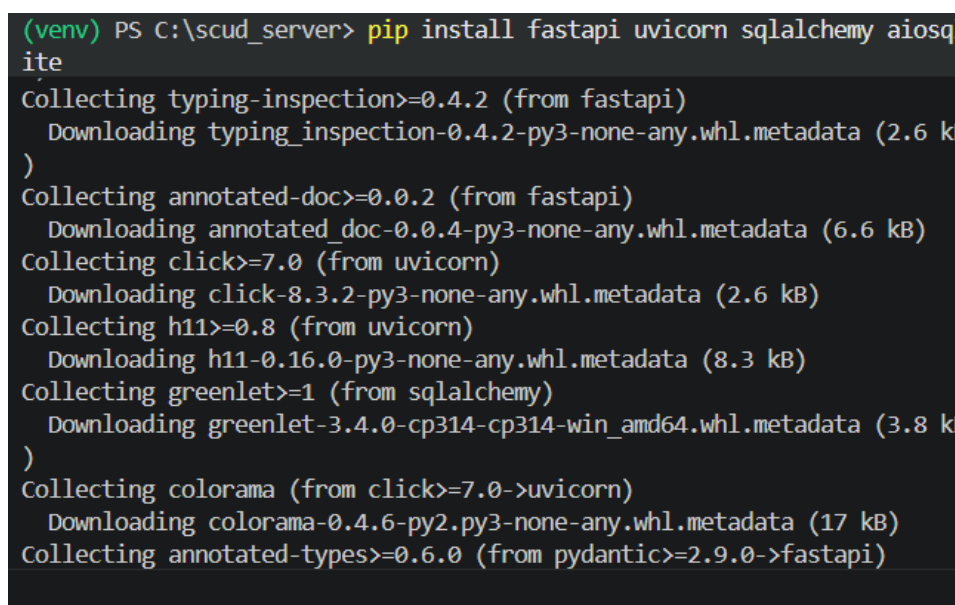


Рисунок 32 – Установка пакетов через pip

После успешной конфигурации создал тестовый файл main.py с простейшим FastAPI-сервером. Для проверки корректности настройки среды

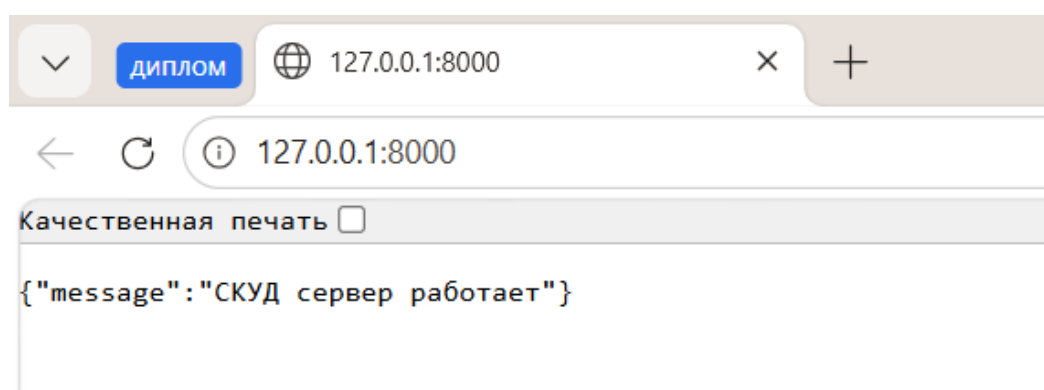
выполнил запуск сервера командой `uvicorn main:app --reload`. Появление сообщения о запуске в терминале и открытие страницы `http://127.0.0.1:8000` в браузере подтверждает корректную настройку среды разработки.

```
main.py / root
1  from fastapi import FastAPI
2
3  app = FastAPI()
4
5  @app.get("/")
6  def root():
7      return {"message": "СКУД сервер работает"}
```

Рисунок 33 – Код простейшего FastAPI-сервера

```
(venv) PS C:\scud_server> uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\scud_server']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [16672] using StatReload
INFO:     Started server process [24940]
INFO:     waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:52323 - "GET / HTTP/1.1" 200 OK
```

Рисунок 34 – Запуск Uvicorn в терминале



Качественная печать ☐

```
{"message": "СКУД сервер работает"}
```

Рисунок 35 – Ответ сервера в браузере

3 Реализация

3.1 Архитектура системы

Разработанная система построена по модели «клиент–сервер» с использованием архитектурного стиля REST. Система состоит из трёх взаимодействующих компонентов.

Аппаратный клиент (контроллер доступа) – реализован на микроконтроллере ESP32 (модель SAMIORE ROBOT ESP32 DevKit) с прошивкой на языке C++ в среде Arduino IDE. Контроллер подключается к локальной сети по Wi-Fi и управляет следующими периферийными устройствами: считыватель Touch Memory (TM-ключей, 3 провода: питание, земля, данные), электромагнитный замок 5В (через модуль реле). При прикладывании ключа ESP32 считывает его 64-битный идентификатор, проверяет по локальному белому списку и в случае разрешения подаёт сигнал на реле, открывая замок на заданное время. Одновременно контроллер отправляет на сервер HTTP POST-запрос с данными о событии (идентификатор ключа, статус «GRANTED»/«DENIED»). Питание ESP32 и электромагнитного замка осуществляется от одного внешнего блока питания на 5В.

Серверная часть – реализована на языке Python с применением асинхронного веб-фреймворка FastAPI. Сервер предоставляет REST API для выполнения всех бизнес-операций:

- приём и сохранение событий от контроллера (POST /log);
- получение журнала событий с фильтрацией и пагинацией (GET /logs);
- управление ключами доступа;
- удалённое открытие двери (POST /open);
- аутентификация администратора для доступа к веб-интерфейсу.

Клиентская часть – одностраничное приложение, разработанное на HTML5, CSS3 и JavaScript с использованием библиотеки Bootstrap 5[11] для адаптивной вёрстки. Взаимодействие с сервером осуществляется через асинхронные Fetch API запросы. Веб-интерфейс позволяет администратору

просматривать журнал событий в реальном времени, добавлять и блокировать ТМ-ключи, удалённо открывать дверь.

База данных – в качестве системы управления базами данных используется SQLite. Взаимодействие серверного приложения с БД организовано через объектно-реляционный маппер SQLAlchemy 2.0 с асинхронным драйвером aiosqlite. SQLite обеспечивает надёжное хранение данных без необходимости развёртывания отдельного сервера БД, что упрощает эксплуатацию системы в условиях малой организации или учебного заведения.

Все компоненты системы взаимодействуют по протоколу HTTP: контроллер отправляет события на сервер, сервер отдаёт данные веб-интерфейсу и по запросу передаёт команды контроллеру. Такая архитектура обеспечивает централизованное логирование, удалённое администрирование и возможность масштабирования.

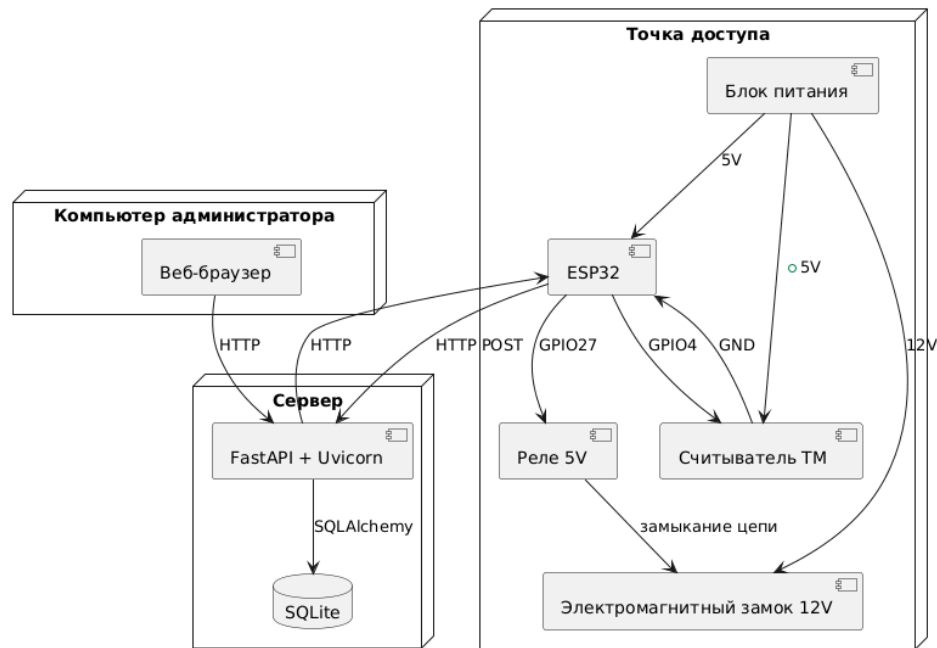


Рисунок 36– Диаграмма развертывания приложения

– Архитектурная схема системы представлена на рисунке 36. HTTP-запросы от веб-интерфейса администратора и от контроллера ESP32 поступают на слой маршрутизации FastAPI. В зависимости от эндпоинта запрос

направляется в соответствующий сервис: обработка событий доступа (/log), управление ключами (/keys), получение журнала (/logs), удалённое открытие двери (/open). Сервисы выполняют валидацию входных данных (через Pydantic-схемы), реализуют бизнес-логику (сохранение события в журнал, добавление или удаление ключа в базе данных, отправка команды контроллеру) и обращаются к базе данных через модели SQLAlchemy. Для обеспечения целостности данных критически важные операции выполняются в рамках атомарных транзакций базы данных.

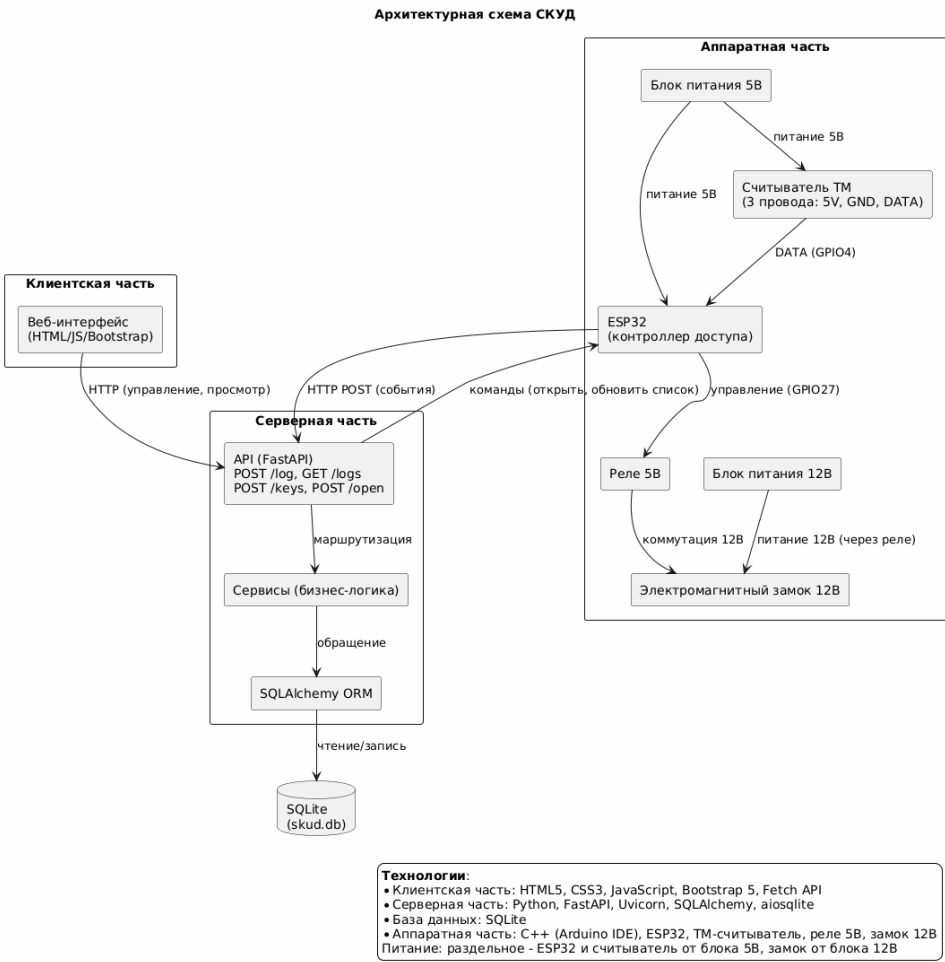


Рисунок 37 – Архитектура приложения

3.2 Работа с базой данных

Для хранения информации о ключах доступа, сотрудниках и событиях спроектирована реляционная база данных. Схема базы данных включает следующие основные сущности:

– access_keys – электронные ключи доступа (ТМ–ключи). Поля: идентификатор, уникальный 64–битный идентификатор ключа (UID), идентификатор пользователя (владельца), статус (активен/заблокирован), дата назначения, дата окончания действия (опционально);

– access_log – журнал событий доступа. Содержит поля: идентификатор, идентификатор ключа (UID), статус попытки (GRANTED/DENIED/BUTTON), временная метка, идентификатор контроллера (для масштабирования);

– controllers – список контроллеров. Поля: идентификатор, имя, IP–адрес (или идентификатор), статус (онлайн/офлайн), дата последней активности.

Для обеспечения производительности и целостности данных используются индексы по полю uid в таблицах access_keys и access_log, по полю timestamp (для быстрой выборки последних событий), а также внешние ключи: access_keys.user_id ссылается на users.id, access_log.uid — на access_keys.uid (при желании). CHECK–ограничения для статусов (GRANTED/DENIED/BUTTON) и статуса активности ключа (0/1).

Создание и обновление схемы базы данных автоматизировано с помощью инструмента миграций Alembic[12] (для SQLite). Все миграции хранятся в каталоге alembic/versions и применяются последовательно при развёртывании приложения.

Для наглядного понимания работы БД на рисунке 37 представлена схема физической модели базы данных.

Схема физической модели базы данных СКУД

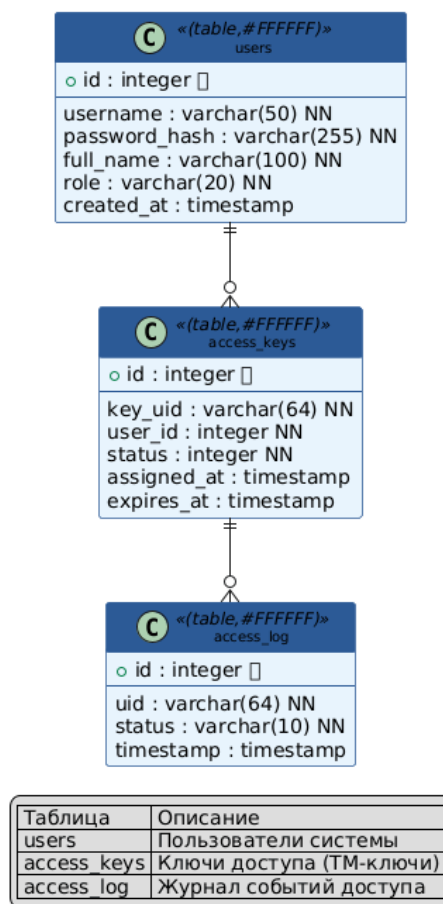


Рисунок 38 – Физическая модель базы данных

3.3 Разработка аппаратной части

Аппаратная часть реализована на микроконтроллере ESP32 (модель SAMIORE ROBOT ESP32 DevKit, 30 Pin, Type-C) с прошивкой на языке C++ в среде Arduino IDE. Система управляет считывателем Touch Memory (ТМ-ключей) и электромагнитным замком 12В через модуль реле 5В. Контроллер подключается к локальной Wi-Fi сети и обменивается данными с сервером по протоколу HTTP.

3.3.1 Главный цикл опроса и идентификации

Основным компонентом прошивки является бесконечный цикл loop(), который непрерывно опрашивает шину 1-Wire, к которой подключён ТМ-считыватель (пин GPIO4). При обнаружении ключа считывается его 64-битный уникальный идентификатор (UID) и проверяется контрольная

сумма CRC8. Далее UID сравнивается с локальным белым списком, хранящимся в памяти ESP32 (массив allowedKey). В зависимости от результата:

Ключ разрешён – подаётся сигнал на реле (GPIO27) на 2 секунды, замок открывается; на сервер отправляется событие GRANTED.

Ключ запрещён – замок не открывается, на сервер отправляется событие DENIED.

Определения реакции на ключ:

// Основной цикл обработки

```
void loop() {
  byte address[8];
  if (oneWire.search(address)) {
    if (OneWire::crc8(address, 7) == address[7]) {
      bool granted = (memcmp(address, allowedKey, 8) == 0);
      if (granted) {
        digitalWrite(RELAY_PIN, HIGH);
        delay(2000);
        digitalWrite(RELAY_PIN, LOW);
        sendEvent(address, "GRANTED");
      } else {
        sendEvent(address, "DENIED");
      }
    } else {
      Serial.println("CRC error");
    }
    oneWire.reset_search();
  }
  delay(100);
}
```

3.3.2 Отправка событий на сервер

Функция sendEvent() формирует HTTP POST-запрос в формате JSON и отправляет его на сервер (FastAPI). Адрес сервера и порт задаются константами. В случае недоступности сети событие не сохраняется локально, но при восстановлении соединения новые попытки будут передаваться.

Отправка события:

```
void sendEvent(byte* uid, String status) {
  if (WiFi.status() != WL_CONNECTED) return;
```

```

HTTPClient http;
http.begin("http://192.168.1.100:8000/log");
http.addHeader("Content-Type", "application/json");
char uidStr[17];
for (byte i = 0; i < 8; i++) sprintf(&uidStr[i*2], "%02X", uid[i]);
String payload = "{\"uid\\":\"" + String(uidStr) + "\", \"status\\":\"" + status + "\"}";
http.POST(payload);
http.end();
}

```

При получении команды от сервера ESP32 может перезагрузить свой локальный список.

Диаграмма последовательности, показывающая процесс успешной идентификации ТМ-ключа, открытия замка и отправки события на сервер, изображена на рисунке 39

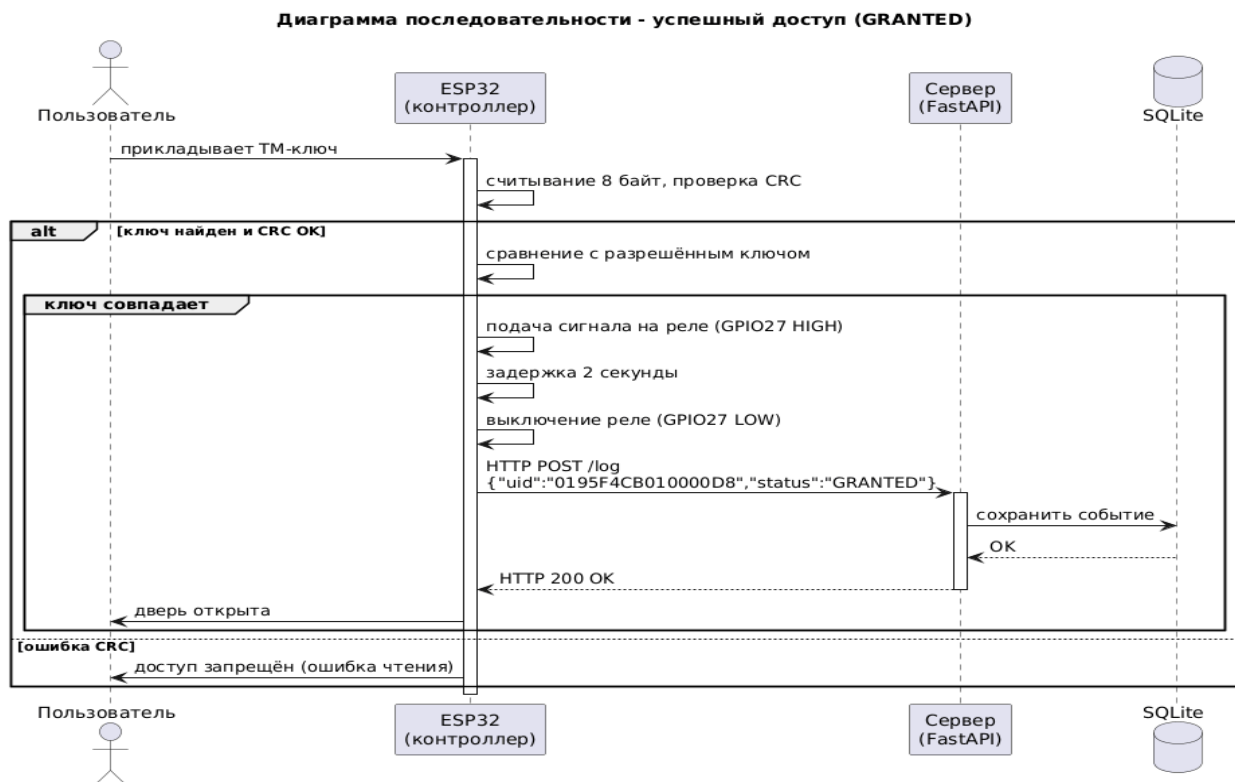


Рисунок 39 – Диаграмма последовательности «Открытие»

Процесс отклонения ТМ-ключа показан на диаграмме последовательностей (рисунок 40).

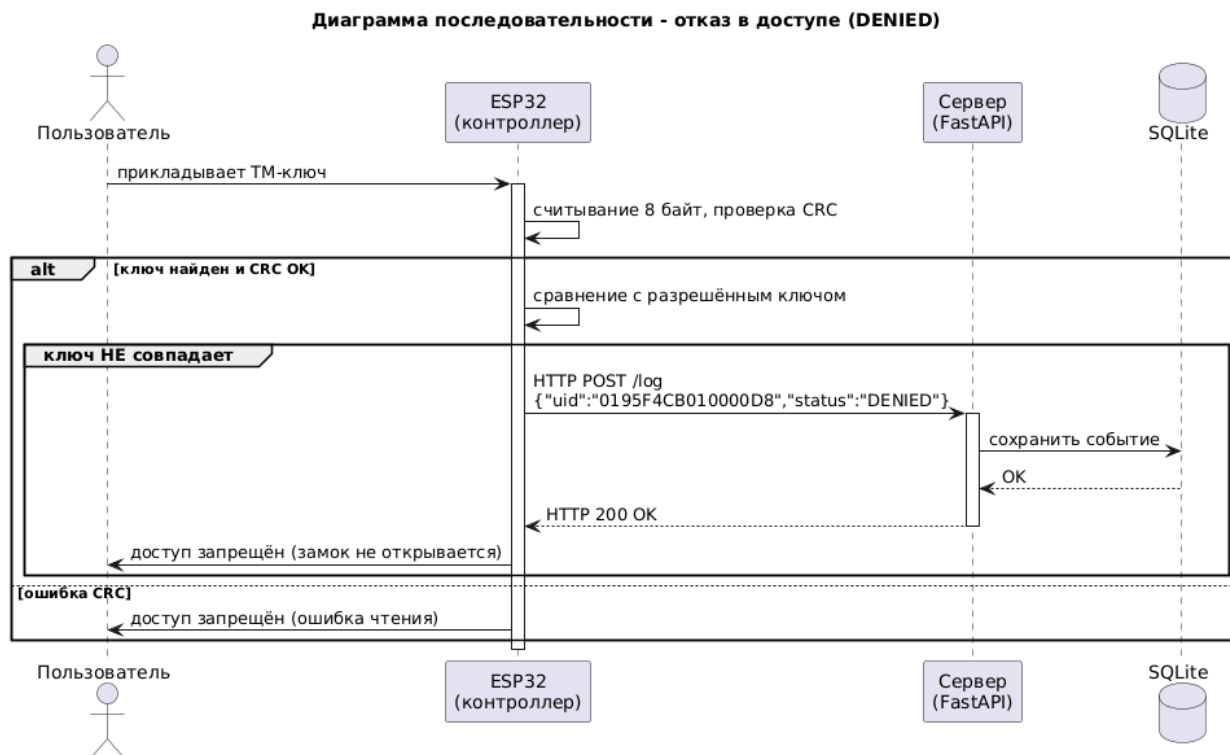


Рисунок 40 – Диаграмма последовательности «Отклонение ключа»

3.4 Разработка серверного приложения и веб-интерфейса

Серверная часть реализована на языке Python с использованием асинхронного веб-фреймворка FastAPI и объектно-реляционного маппера SQLAlchemy 2.0 (асинхронный режим с драйвером aiosqlite для SQLite). Веб-интерфейс администратора представляет собой одностраничное приложение с навигацией через вкладки, разработанное на HTML5, CSS3 и JavaScript с использованием библиотеки Bootstrap 5 для адаптивной вёрстки.

3.4.1 Модуль приёма событий от контроллера

Контроллер ESP32 отправляет на сервер POST-запросы с данными о каждой попытке доступа. Эндпоинт /log принимает JSON следующего формата:

```
{"uid": "0195F4CB010000D8", "status": "GRANTED"}
```

Сервис event_service.py выполняет:

- проверка входных данных через Pydantic-схему (проверка, что uid – строка длиной 16 символов, status – одно из трёх значений: GRANTED, DENIED, BUTTON);

- сохранение записи в таблицу `access_log` с добавлением временной метки (Unix timestamp);
- возврат HTTP 200 OK в случае успеха, иначе – 400 Bad Request.

Прием событий:

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import time
app = FastAPI()
class Event(BaseModel):
    uid: str
    status: str
@app.post("/log")
async def log_event(event: Event):
    if event.status not in ("GRANTED", "DENIED", "BUTTON"):
        raise HTTPException(400, "Invalid status")
    # Сохранение в БД
    async with async_session() as db:
        db.add(AccessLog(uid=event.uid, status=event.status, timestamp=int(time.time())))
        await db.commit()
    return {"message": "ok"}
```

3.4.2 Модуль журнала событий (access_log)

Сервер предоставляет эндпоинт GET /logs для получения списка событий с поддержкой пагинации. Параметры запроса:

- skip – количество пропускаемых записей;
- limit – максимальное количество записей (по умолчанию 100).

Реализована асинхронная функция, возвращающая список событий в формате JSON, отсортированный по убыванию времени:

```
@app.get("/logs")
async def get_logs(skip: int = 0, limit: int = 100):
    async with async_session() as db:
        result = await db.execute(select(AccessLog).order_by(AccessLog.timestamp.desc()).of
fset(skip).limit(limit))
        logs = result.scalars().all()
    return logs
```

3.4.3 Модуль управления ключами доступа

Администратор через веб-интерфейс может управлять ТМ-ключами. Сервер предоставляет эндпоинты:

- GET /keys – получение списка всех ключей;
- POST /keys – добавление нового ключа (тело: {"key_uid": "...", "user_id": 1, "expires_at": 0});
- DELETE /keys/{id} – удаление ключа;
- PATCH /keys/{id}/block – блокировка/разблокировка ключа (изменение поля status).

Все изменения ключей сохраняются в таблице access_keys.

3.4.4 Модуль удалённого открытия двери

Веб-интерфейс реализован в виде одностраничного приложения с тремя основными вкладками, доступными без входа в систему:

- Журнал событий – таблица с записями доступа, автообновление каждые 5 секунд;
- Управление ключами – список ключей с возможностью добавления, удаления, блокировки;
- Управление дверью – кнопка удалённого открытия замка.

Навигация между вкладками осуществляется через панель с кнопками-ссылками. При нажатии на кнопку «Открыть дверь» отправляется POST-запрос на /open; в случае успеха выводится всплывающее сообщение, а журнал автоматически обновляется, показывая запись ADMIN_OPEN:

```
<div class="container mt-4">
  <ul class="nav nav-tabs" id="myTab" role="tablist">
    <li class="nav-item" role="presentation">
      <button class="nav-link active" id="logs-tab"
data-bs-toggle="tab" data-bs-target="#logs" type="button"
role="tab">Журнал событий</button>
    </li>
    <li class="nav-item" role="presentation">
      <button class="nav-link" id="keys-tab"
data-bs-toggle="tab" data-bs-target="#keys" type="button"
role="tab">Управление ключами</button>
    </li>
    <li class="nav-item" role="presentation">
      <button class="nav-link" id="door-tab"
data-bs-toggle="tab" data-bs-target="#door" type="button"
role="tab">Управление дверью</button>
    </li>
  </ul>
</div>
```

```

        </li>
    </ul>
    <div class="tab-content mt-3">
        <div class="tab-pane fade show active" id="logs"
role="tabpanel">
            <table class="table table-striped">

<thead><tr><th>ID</th><th>UID</th><th>Статус</th><th>Время</th><
/tr></thead>

            <tbody id="logBody"></tbody>
        </table>
    </div>
    <div class="tab-pane fade" id="keys" role="tabpanel">
        <form id="addKeyForm">
            <div class="mb-3"><input type="text"
id="keyUid" placeholder="UID (16 hex)" required></div>
            <div class="mb-3"><input type="number"
id="userId" placeholder="ID пользователя" required></div>
            <button type="submit" class="btn
btn-primary">Добавить ключ</button>
        </form>
        <div id="keysList"></div>
    </div>
    <div class="tab-pane fade" id="door" role="tabpanel">
        <button id="openDoorBtn" class="btn
btn-danger">Открыть дверь</button>
        <div id="doorResult" class="mt-2"></div>
    </div>
</div>
</div>

```

Функции для загрузки журнала, управления ключами и открытия замка реализованы через JS:

```

async function loadLogs() {
    const response = await fetch('/logs?limit=50');
    const logs = await response.json();
    const tbody = document.getElementById('logBody');
    tbody.innerHTML = "";
    logs.forEach(log => {
        const row = tbody.insertRow();
        row.insertCell(0).innerText = log.id;
        row.insertCell(1).innerText = log.uid;
        row.insertCell(2).innerText = log.status;
        row.insertCell(3).innerText = new Date(log.timestamp * 1000).toLocaleString();
    });
}

```

```

    });
  }
  setInterval(loadLogs, 5000);

  document.getElementById('openDoorBtn').addEventListener('click', async () => {
    const response = await fetch('/open', { method: 'POST' });
    const resultDiv = document.getElementById('doorResult');
    if (response.ok) {
      resultDiv.innerHTML = '<div class="alert alert-success">Дверь открыта</div>';
      loadLogs(); // обновить журнал
    } else {
      resultDiv.innerHTML = '<div class="alert alert-danger">Ошибка открытия</div>';
    }
  });
});

```

3.5 Тестирование

В процессе разработки программно–аппаратной СКУД был проведён комплекс мероприятий по тестированию, направленных на проверку корректности работы функциональных требований, надёжности и производительности. Тестирование проводилось на всех уровнях: модульное тестирование прошивки ESP32, интеграционное тестирование взаимодействия ESP32 с сервером через HTTP, системное тестирование веб–интерфейса администратора. В качестве средств тестирования использовались: отладка через Serial Monitor (Arduino IDE), ручное тестирование API через Postman[13] и встроенную документацию FastAPI (/docs), а также инструменты разработчика браузера (консоль, вкладка Network) для проверки запросов веб–интерфейса.

UI–тестирование проводилось вручную в браузерах Google Chrome (версия 90+), Mozilla Firefox (версия 88+) и Microsoft Edge (версия 90+).

Тест 1 – Считывание ТМ–ключа и реакция ESP32:

При прикладывании ТМ–ключа, предварительно внесённого в белый список, ESP32 считывает его 64–битный идентификатор по протоколу 1–Wire, проверяет контрольную сумму CRC и сравнивает с локальным списком разрешённых ключей, загруженным из энергонезависимой памяти.

					ПГУ 1.090301.01.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		51

Совпадение найдено – контроллер подаёт сигнал на реле (GPIO27) на 2 секунды. Электромагнитный замок щёлкает и открывает дверь. Одновременно ESP32 формирует HTTP POST–запрос на сервер `http://192.168.50.100:8000/log` с телом `{ "uid" : "0195F4CB010000D8" , "status" : "GRANTED" }`. В мониторе порта выводятся строки `Key: 0195F4CB010000D8, GRANTED` и `Lock opened by KEY`. Сервер сохраняет запись в таблицу `access_log` с текущим временем. В веб–интерфейсе (вкладка «Журнал событий») через 1–2 секунды появляется новая строка с полем `GRANTED` и корректным `UID`.

```
Found key: 0184D5CB01000081
Access GRANTED
Lock opened by KEY
HTTP POST success, code: 500
```

Рисунок 41 – Монитор порта ESP32

ID	UID	Статус	Время
2	017231CD0100008D	GRANTED	20.05.2026, 21:27:16

Рисунок 42 – Журнал событий с записью GRANTED

Тест 2 – Отклонение неизвестного ключа:

При прикладывании ТМ–ключа, отсутствующего в белом списке, ESP32 считывает его `UID`, проверяет `CRC` и не находит совпадения в локальном списке. Реле не срабатывает, замок остаётся закрытым. В монитор порта выводится `Key: ...` и `DENIED`. ESP32 отправляет на сервер `POST`–запрос `{ "uid": "017231CD0100008D", "status": "DENIED" }`. Сервер сохраняет. В веб–интерфейсе через несколько секунд появляется запись с красным фоном и полем `DENIED`. Дверь не открывается.

```
Found key: 017231CD0100008D
Access DENIED
HTTP POST success, code: 500
```

Рисунок 43 – Отказ в доступе

3	0195F4CB010000D8	DENIED	20.05.2026, 21:29:26
---	------------------	--------	----------------------

Рисунок 44 – Журнал событий с записью DENIED

Тест 3 – Просмотр журнала событий в веб-интерфейсе

Администратор открывает веб-интерфейс по адресу <http://192.168.50.100:8000>. По умолчанию активна вкладка «Журнал событий». Страница отправляет асинхронный GET-запрос на /logs каждые 5 секунд (JavaScript setInterval). Таблица заполняется данными из базы SQLite, отсортированными по убыванию времени. В колонках отображаются ID, UID ключа, статус и локальное время события. При появлении новых событий таблица обновляется без перезагрузки страницы.

СКУД Панель управления

<div> <div>Журнал событий</div> <div>Управление ключами</div> <div>Управление дверью</div> </div>			
ID	UID	Статус	Время
4	0195F4CB010000D8	DENIED	20.05.2026, 21:29:26
3	0195F4CB010000D8	DENIED	20.05.2026, 21:29:26
2	017231CD0100008D	GRANTED	20.05.2026, 21:27:16
1	017231CD0100008D	LEARNED	20.05.2026, 21:27:13

Рисунок 45 – Вкладка «Журнал событий» с записями

Тест 4 – Добавление нового ключа через веб-интерфейс.

Администратор активирует режим обучения через веб-интерфейс. Сервер отправляет команду ESP32 POST /learn_mode. ESP32 устанавливает флаг learnMode = true. Далее любой приложенный к считывателю ТМ-ключ вызывает следующие действия:

ESP32 считывает UID, открывает замок;

UID добавляется в локальный белый список и сохраняется в энергонезависимую память;

ESP32 отправляет на сервер POST-запрос /learn_key с UID ключа;

Сервер сохраняет ключ в таблицу access_keys (status=1);

Режим обучения автоматически выключается после первого приложенного ключа.

После этого данный ключ уже работает как разрешённый.

СКУД Панель управления

[Журнал событий](#) [Управление ключами](#) [Управление дверью](#)

Добавить ключ (поднесите ключ к считывателю)

ID	UID	Статус	Действия
1	017231CD0100008D	Активен	Заблокировать Удалить

Рисунок 46 – Кнопка активации режима обучения в веб-интерфейсе

```
Learn mode activated
Found key: 0195F4CB010000D8
Whitelist saved to flash
HTTP POST success, code: 500
Learn mode disabled, key added
Found key: 0195F4CB010000D8
Access GRANTED
Lock opened by KEY
HTTP POST success, code: 500
```

Рисунок 47 – Лог ESP32 при добавлении ключа через обучение

Тест 5 – Блокировка и удаление ключа.

На вкладке «Управление ключами» администратор видит список ключей. При нажатии «Блокировать» отправляется PATCH /keys/{id}/block. Сервер меняет статус ключа в БД и вызывает update_esp_whitelist(), отправляя на ESP32 актуальный список активных ключей. ESP32 сохраняет новый список, и заблокированный ключ перестаёт открывать замок. При нажатии «Удалить» ключ удаляется из БД, белый список на ESP32 также обновляется. В журнале событий фиксируются попытки прохода заблокированным ключом как DENIED.

СКУД Панель управления

[Журнал событий](#)[Управление ключами](#)[Управление дверью](#)

Добавить ключ (поднесите ключ к считывателю)

ID	UID	Статус	Действия
2	0195F4CB010000D8	Активен	<button>Заблокировать</button> <button>Удалить</button>
3	0184D5CB01000081	Активен	<button>Заблокировать</button> <button>Удалить</button>

Рисунок 48 – Список ключей с кнопками блокировки и удаления

```
INFO: 127.0.0.1:60511 - "GET /logs HTTP/1.1" 200 OK
Whitelist sent to ESP32: 1 keys
```

Рисунок 49 – Обновление списка ключей на плате

Тест 6 – Удалённое открытие двери.

На вкладке «Управление дверью» администратор нажимает кнопку «Открыть дверь». Сервер отправляет ESP32 команду POST /open. ESP32 активирует реле на 2 секунды – замок открывается. Сервер записывает в журнал событие ADMIN_OPEN со статусом GRANTED. В веб-интерфейсе появляется всплывающее сообщение «Дверь открыта».

СКУД Панель управления

[Журнал событий](#)[Управление ключами](#)[Управление дверью](#)

Открыть дверь

Рисунок 50– кнопка удаленного октрытия

28	ADMIN_OPEN	GRANTED	25.05.2026, 22:07:09
----	------------	---------	----------------------

Рисунок 51– удаленное открытие администратором

					ПГУ 1.090301.01.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпись	Дата		

Lock opened by REMOTE

Рисунок 52– открытие администратором

Тест 7 – Синхронизация белого списка при старте ESP32.

При включении ESP32 после загрузки прошивки она подключается к Wi-Fi и отправляет GET-запрос на сервер /keys. Сервер возвращает список активных ключей (status=1). ESP32 сохраняет их в локальную память (Preferences). Благодаря этому добавленные ранее через режим обучения ключи остаются рабочими после перезагрузки контроллера.

```
rst:0x1 (POWERON_RESET),boot:0x17 (SPI_FAST_FLASH_BOOT)
config: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:4876
ho 0 tail 12 room 4
load:0x40078000,len:16560
load:0x40080400,len:3500
entry 0x400805b4
Loaded whitelist: 1
```

Рисунок 53 – синхронизация списка ключей

Тест 8 – Производительность и стабильность.

Система тестировалась при последовательных прикладываниях ключа. ESP32 успешно обрабатывала каждое событие, отправляла запрос на сервер. Журнал заполнялся без потерь.

```
INFO: 192.168.50.134:54100 - "POST /log HTTP/1.1" 200 OK
INFO: 192.168.50.134:54101 - "POST /log HTTP/1.1" 200 OK
INFO: 192.168.50.134:54102 - "POST /log HTTP/1.1" 200 OK
INFO: 192.168.50.134:54103 - "POST /log HTTP/1.1" 200 OK
INFO: 192.168.50.134:54104 - "POST /log HTTP/1.1" 200 OK
INFO: 192.168.50.134:54105 - "POST /log HTTP/1.1" 200 OK
INFO: 192.168.50.134:54106 - "POST /log HTTP/1.1" 200 OK
INFO: 192.168.50.134:54107 - "POST /log HTTP/1.1" 200 OK
INFO: 192.168.50.134:54108 - "POST /log HTTP/1.1" 200 OK
INFO: 192.168.50.134:54109 - "POST /log HTTP/1.1" 200 OK
INFO: 192.168.50.134:54110 - "POST /log HTTP/1.1" 200 OK
```

Рисунок 54– сервер принимает события

Все функциональные требования выполнены. Система обеспечивает надёжную идентификацию ТМ–ключей, централизованное логирование, удалённое управление замком и динамическое обновление белого списка без перепрошивки контроллера. Веб-интерфейс администратора предоставляет удобные инструменты для мониторинга и администрирования. Система устойчива к сетевым сбоям и некорректным входным данным.

					ПГУ 1.090301.01.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		57

4 Безопасность жизнедеятельности

В данном разделе рассматриваются вопросы обеспечения безопасности жизнедеятельности при разработке, сборке и тестировании программно-аппаратного комплекса системы управления и контроля доступом (СКУД). Разрабатываемый комплекс включает микроконтроллер ESP32, считыватель Touch Memory (iButton), электромагнитное реле, электромагнитный замок, источники постоянного напряжения (5 В и 12 В), а также персональный компьютер. При этом ПК используется на этапах разработки программного обеспечения на Python, написания прошивки для ESP32, настройки и отладки комплекса. Поэтому требования безопасности должны учитывать как работу разработчика за ПК, так и эксплуатацию электрооборудования при коммутации и тестировании аппаратной части.

4.1 Характеристика потенциальных рисков при работе с комплексом

В соответствии с классификацией ГОСТ 12.0.003-2015[14] к опасным и вредным производственным факторам при работе с комплексом могут относиться физические, психофизиологические и организационные факторы. Наиболее существенными для рассматриваемого комплекса являются возможность поражения электрическим током, повышенная зрительная нагрузка при работе с ПК и мелкими электронными компонентами, а также монотонность выполняемых операций.

К электрическим факторам относятся наличие цепей питания ПК от сети 220 В переменного тока, а также источников постоянного напряжения 5 В и 12 В для питания микроконтроллера и электромагнитного замка. Существует вероятность короткого замыкания при повреждении изоляции проводов, неправильном подключении разъёмов на макетной плате или попадании посторонних токопроводящих предметов в зону электрических соединений. Несмотря на то, что управляющие цепи являются низковольтными и условно

безопасными для человека, блоки питания и цепи управления реле требуют соблюдения правил электробезопасности для предотвращения перегрева и возгорания.

К психофизиологическим факторам относятся длительное статическое напряжение мышц при работе за компьютером, повышенная нагрузка на зрительный анализатор из-за необходимости концентрации внимания на мониторе и мелких деталях при коммутации, а также умственное переутомление при отладке программного кода.

4.2 Требования безопасности при эксплуатации комплекса

К работе с комплексом допускается персонал, ознакомленный с назначением устройства, инструкцией по эксплуатации, порядком запуска и действиями при возникновении нештатных ситуаций. Перед началом работы разработчик должен убедиться в исправности кабелей, разъёмов, отсутствии видимых повреждений изоляции проводов, а также в отсутствии посторонних предметов в рабочей зоне.

Сборка и изменение схемы коммутации (подключение считывателя, реле, замка к выводам GPIO микроконтроллера) должны выполняться только при полностью отключенном питании. Запуск тестирования допускается после визуальной проверки правильности соединений и надежности контактов. Во время подачи питания запрещается касаться токоведущих частей, перемещать провода под напряжением или вмешиваться в работу схемы.

При появлении запаха гари, дыма, искрения, нехарактерного нагрева компонентов (например, стабилизатора напряжения на плате ESP32 или блока питания) работа комплекса должна быть немедленно остановлена. После остановки необходимо отключить питание от сети и сообщить ответственному лицу о неисправности. Повторный запуск допускается только после выявления и устранения причины нештатной ситуации.

4.3 Организация рабочего места оператора

Рабочее место разработчика должно обеспечивать удобное наблюдение за ходом отладки, свободный доступ к органам управления ПК и безопасную зону для сборки аппаратной части. Персональный компьютер, блоки питания и макетная плата должны располагаться так, чтобы соединительные кабели не находились в проходах, не подвергались натяжению, перегибу и не мешали действиям оператора.

Монитор должен быть установлен на расстоянии 60–70 см от глаз, на уровне или чуть ниже линии зрения, чтобы исключить чрезмерный наклон головы. Рабочая поверхность должна быть чистой, сухой и иметь достаточную площадь для размещения документации, инструментов и тестируемого оборудования.

4.4 Электробезопасность

Электробезопасность при эксплуатации комплекса должна обеспечиваться в соответствии с требованиями ГОСТ 12.1.019-2017[15], устанавливающего общие требования и виды защиты персонала от воздействия электрического тока. Все элементы комплекса, питающиеся от сети 220 В (персональный компьютер, сетевой адаптер), должны иметь исправную изоляцию, защитные кожухи и подключаться к сети с обязательным защитным заземлением.

Подключение и отключение модулей (считывателя, реле, электромагнитного замка) к микроконтроллеру должны выполняться только при отключённом питании соответствующих цепей. Запрещается эксплуатировать комплекс при повреждении кабелей, отсутствии защитных крышек на блоках питания, следах перегрева или неустойчивом контакте разъёмов.

Обслуживание источников питания и коммутация силовых цепей (12 В для замка) должны выполняться аккуратно, с использованием инструмента с

изолированными рукоятками. Оператор должен знать порядок экстренного отключения питания комплекса и основные действия при поражении электрическим током.

4.5 Пожарная безопасность

Пожарная безопасность комплекса должна обеспечиваться системой предотвращения пожара и организационно-техническими мероприятиями, что соответствует общим положениям ГОСТ 12.1.004-91[16]. Основными причинами пожароопасных ситуаций при разработке СКУД могут быть короткое замыкание в цепи электромагнитного замка, перегрев импульсного блока питания или стабилизатора напряжения на плате ESP32, перегрузка электрической сети, использование неисправных удлинителей или накопление пыли внутри системного блока ПК.

Для снижения риска возгорания необходимо использовать только исправные, сертифицированные источники питания, не допускать перегрузки сетевых фильтров, не закрывать вентиляционные отверстия блоков питания и системного блока, а также не размещать горючие материалы (бумагу, растворители) рядом с нагревающимися элементами.

В помещении должны быть предусмотрены первичные средства пожаротушения (углекислотный огнетушитель, пригодный для тушения электроустановок под напряжением), свободные проходы и доступ к средствам отключения электропитания. После окончания работы рекомендуется полностью обесточивать оборудование, если дальнейшая работа не планируется.

4.6 Санитарно-гигиенические требования

В помещении необходимо поддерживать нормируемые параметры микроклимата, обеспечивать естественную или принудительную вентиляцию, регулярную влажную уборку пыли и достаточную освещённость рабочей зоны в соответствии с СанПиН 1.2.3685-21.

					ПГУ 1.090301.01.001 ПЗ	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

Освещённость рабочего места должна быть достаточной (не менее 300 лк) для уверенного наблюдения за LCD-дисплеем монитора, а также для комфортной работы с мелкими электронными компонентами и выводами микросхем при коммутации. Следует исключить блики на экране монитора и затенение рабочей зоны с макетной платой.

Температурный режим в помещении рекомендуется поддерживать на уровне 20–22°C, что обеспечивает комфортную работу оператора и стабильную работу электронного оборудования. Необходимо исключать перегрев компонентов, обеспечивать свободную циркуляцию воздуха около корпуса ПК и блоков питания. Для снижения психофизиологической нагрузки и предотвращения зрительного утомления необходимо соблюдать режим труда и отдыха: делать перерывы на 10–15 минут каждый час работы за компьютером, выполняя простую производственную гимнастику для глаз и мышц спины.

Соблюдение перечисленных требований позволяет снизить вероятность травматизма и профессиональных заболеваний, уменьшить влияние вредных факторов, обеспечить безопасные условия труда разработчика и повысить надёжность выполнения операций по созданию и тестированию программно-аппаратного комплекса СКУД.

Заключение

В ходе выполнения дипломной работы были изучены способы автоматизации контроля доступа на основе ТМ-ключей (Touch Memory), а также рассмотрены существующие решения для систем управления доступом (Sigur, RusGuard, Proxway) и их недостатки, такие как закрытость протоколов, высокая стоимость лицензий и невозможность самостоятельной модификации.

В процессе работы был разработан и собран опытный образец программно-аппаратной системы управления и контроля доступом (СКУД). В состав комплекса вошли микроконтроллерная плата SAMIORE ROBOT ESP32 DevKit, ТМ-считыватель Болид «Считыватель-2 исп.01», модуль реле 5В, электромагнитный замок 12В, опционально герконовый датчик, а также два блока питания (5В и 12В). Система дополнена серверным приложением на Python с использованием фреймворка FastAPI и базы данных SQLite, а также веб-интерфейсом администратора на HTML/CSS/JS с адаптивной вёрсткой (Bootstrap 5).

Была разработана прошивка для ESP32 на языке C++ в среде Arduino IDE, реализующая чтение ТМ-ключа по протоколу 1-Wire, проверку CRC, сравнение с локальным белым списком, управление реле, отправку HTTP-событий на сервер и приём команд (обновление белого списка, удалённое открытие двери, режим обучения). Также написано серверное ПО, предоставляющее REST API для управления ключами и получения журнала событий, а также веб-интерфейс, позволяющий администратору просматривать журнал в реальном времени, добавлять, блокировать и удалять ключи, удалённо открывать дверь и активировать режим обучения.

Работоспособность комплекса была подтверждена в ходе опытной проверки. Система успешно идентифицирует разрешённые и запрещённые ТМ-ключи, открывает замок при успешном проходе, фиксирует все попытки доступа в базе данных, а также обеспечивает удалённое управление и

автоматическое обновление белого списка на контроллере. Дополнительно опциональный герконовый датчик позволяет отслеживать фактическое состояние двери.

Применение разработанной СКУД позволяет сократить время на администрирование прав доступа (за счёт режима обучения и веб-интерфейса), исключить ручное программирование ключей на контроллере, а также обеспечить централизованное хранение журнала событий и удалённый контроль. Для малого объекта с 10–20 пользователями экономия времени на управление ключами может достигать нескольких часов в месяц, а стоимость владения существенно ниже по сравнению с коммерческими аналогами.

В дальнейшем комплекс можно развивать путём добавления поддержки нескольких контроллеров (масштабирование), интеграции с видеонаблюдением, введения временных зон доступа, использования шифрования канала связи (HTTPS) и модернизации веб-интерфейса для построения аналитических отчётов.

					ПГУ 1.090301.01.001 ПЗ	Лист
						64
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. espressif.com.: [Электронный ресурс]. – URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (дата обращения 19.05.2026).
2. analog.com.: [Электронный ресурс]. – URL: <https://www.analog.com/en/technical-articles/1wire-communication-through-software.html> (дата обращения 19.05.2026).
3. fastapi.tiangolo.com.: [Электронный ресурс]. – URL: <https://fastapi.tiangolo.com/> (дата обращения 19.05.2026).
4. sqlite.org.: [Электронный ресурс]. – URL: <https://www.sqlite.org/docs.html> (дата обращения 19.05.2026).
5. uvicorn.org.: [Электронный ресурс]. – URL: <https://www.uvicorn.org/> (дата обращения 19.05.2026).
6. pydantic.dev.: [Электронный ресурс]. – URL: <https://docs.pydantic.dev/> (дата обращения 19.05.2026).
7. sqlalchemy.org.: [Электронный ресурс]. – URL: <https://www.sqlalchemy.org/> (дата обращения 19.05.2026).
8. arduino.cc.: [Электронный ресурс]. – URL: <https://www.arduino.cc/en/software/> (дата обращения 19.05.2026).
9. code.visualstudio.com.: [Электронный ресурс]. – URL: <https://code.visualstudio.com/download> (дата обращения 19.05.2026).
10. python.org.: [Электронный ресурс]. – URL: <https://www.python.org/downloads/> (дата обращения 19.05.2026).
11. getbootstrap.com.: [Электронный ресурс]. – URL: <https://getbootstrap.com/docs/5.3/getting-started/introduction/> (дата обращения 19.05.2026).
12. alembic.sqlalchemy.org.: [Электронный ресурс]. – URL: <https://alembic.sqlalchemy.org/> (дата обращения 19.05.2026).

13. postman.com.: [Электронный ресурс]. – URL:
<https://www.postman.com/> (дата обращения 19.05.2026).

14. docs.cntd.ru (ГОСТ 12.0.003-2015): [Электронный ресурс]. – URL:
<https://docs.cntd.ru/document/1200136071> (дата обращения 19.05.2026).

15. docs.cntd.ru (ГОСТ 12.1.019-2017): [Электронный ресурс]. – URL:
<https://docs.cntd.ru/document/1200161238> (дата обращения 19.05.2026).

16. docs.cntd.ru (ГОСТ 12.1.004-91): [Электронный ресурс]. – URL:
<https://docs.cntd.ru/document/9051953> (дата обращения 19.05.2026).

					ПГУ 1.090301.01.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		66

Приложение А

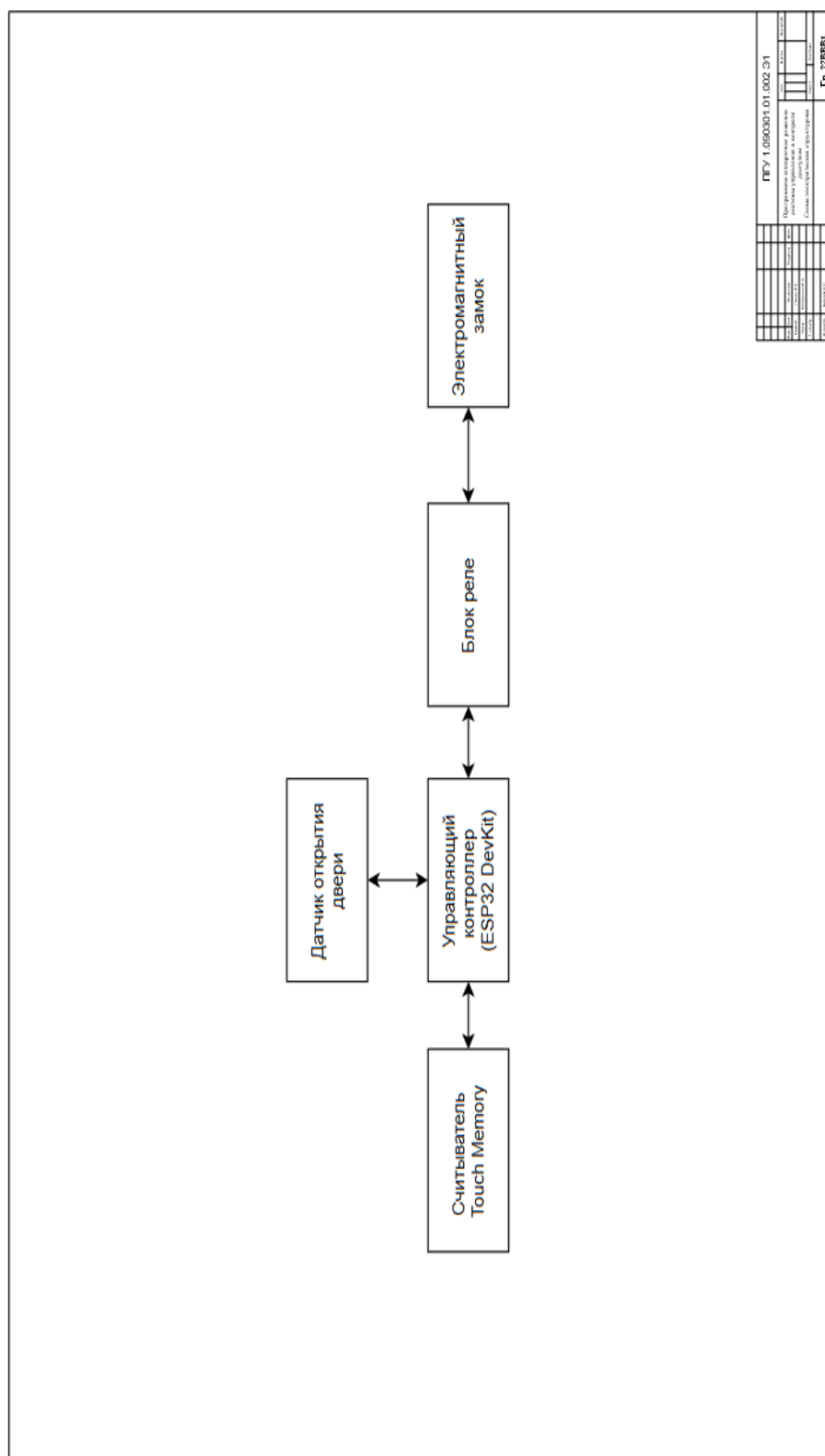
(обязательное)

Программно-аппаратное решение системы управления и контроля доступом

Схемы электрические

					ПГУ 1.090301.01.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		67

Приложение А.1 – Схема электрическая структурная системы



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.01.001 ПЗ

Лист

68

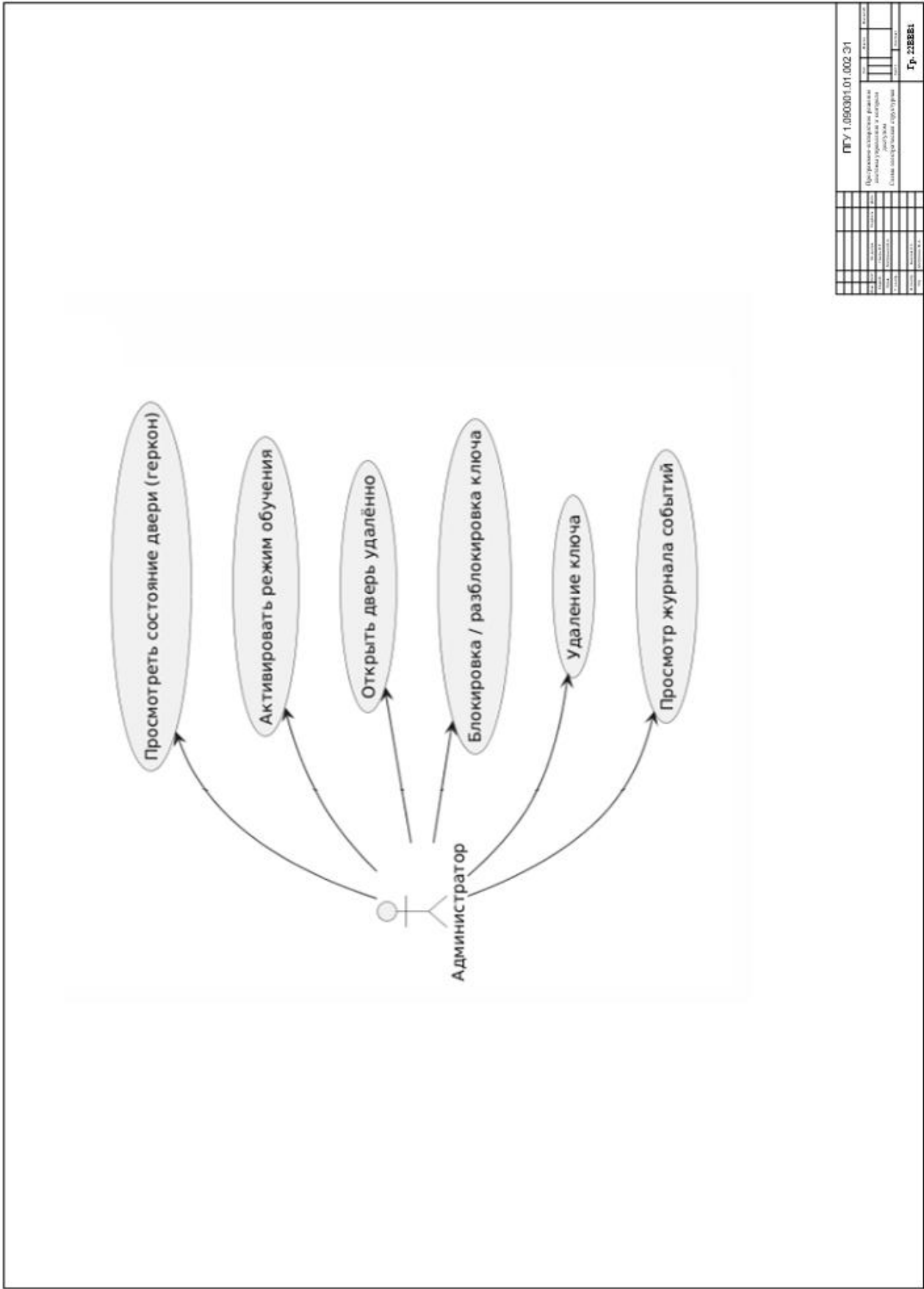
Приложение Б

(обязательное)

Программно-аппаратное решение системы управления и контроля доступом
UML-диаграммы

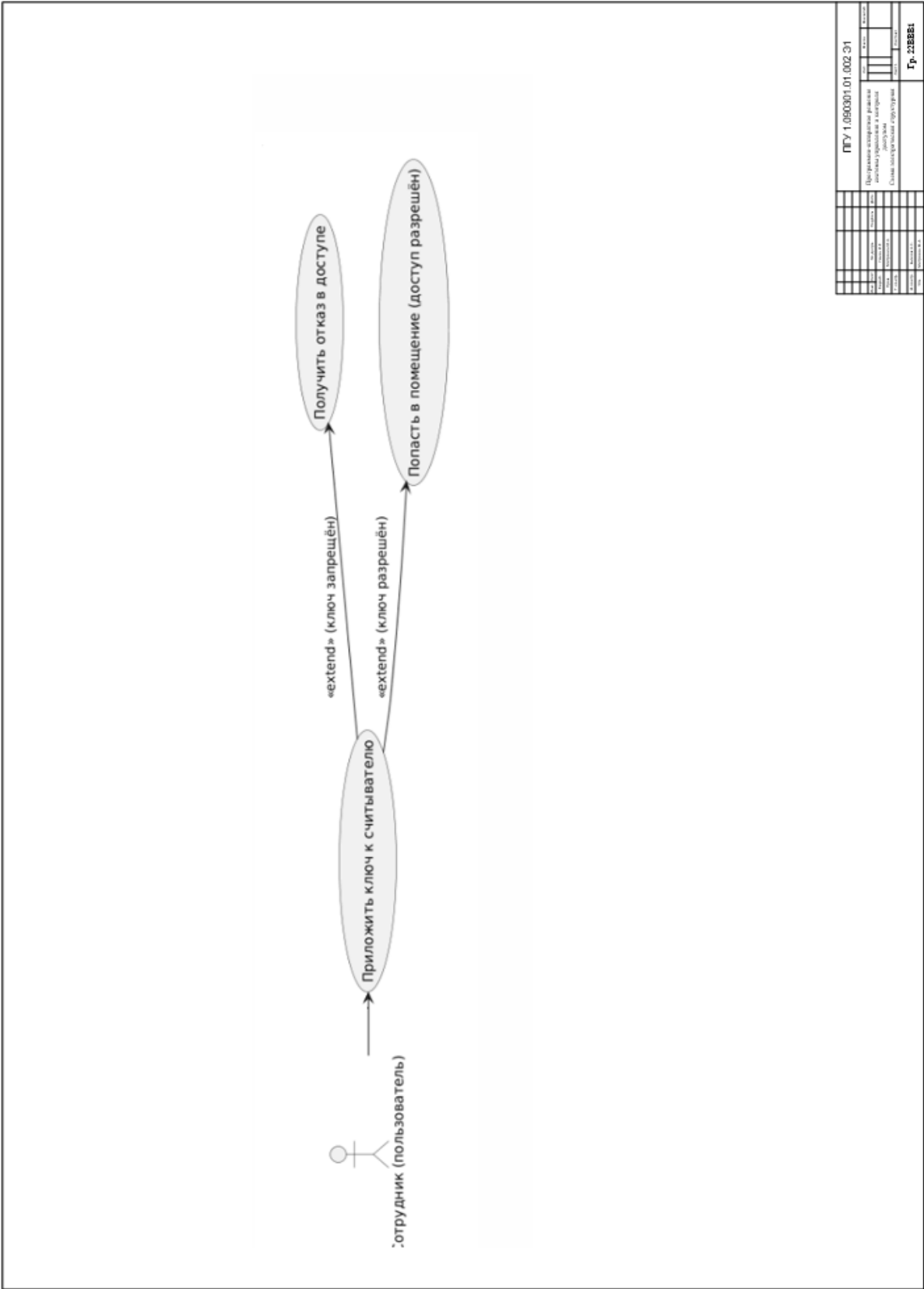
					ПГУ 1.090301.01.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		69

Приложение Б.1 – UML – диаграмма прецедентов интерфейса веб-интерфейса

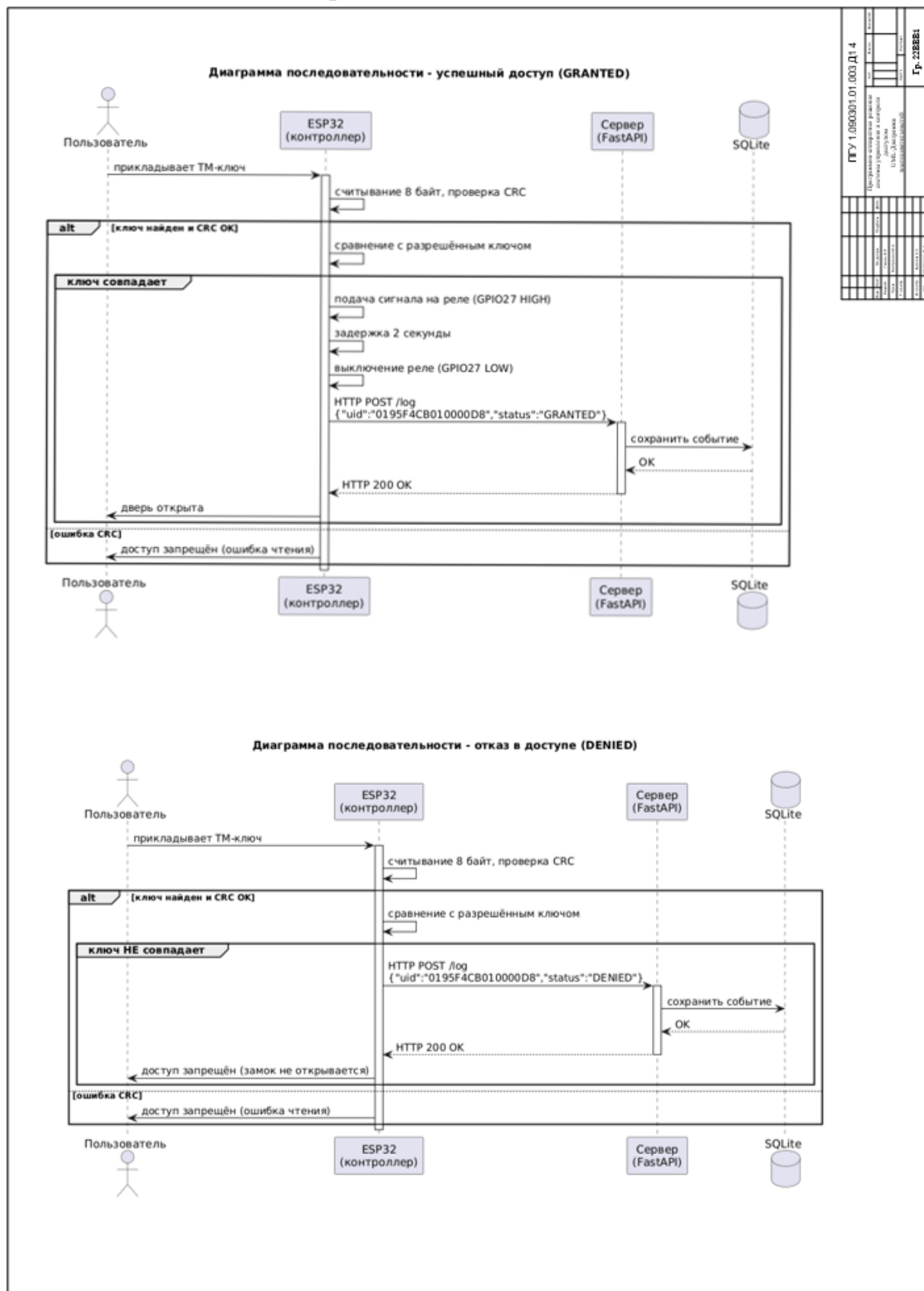


ПГУ 1.090301.01.002 31	
Описание системы, раздела, документа, функции, процедуры	Имя исполнителя, структура
Гр. 228881	

Приложение Б.2 – UML – диаграмма прецедентов интерфейса аппаратной части



Приложение Б.3 – UML – диаграммы последовательности



Приложение В

(обязательное)

Программно-аппаратное решение системы управления и контроля доступом

Листинг программы

					ПГУ 1.090301.01.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		73

Файл «server.py»

```

from fastapi import FastAPI, HTTPException, Depends
from fastapi.responses import FileResponse
from pydantic import BaseModel
from sqlalchemy import create_engine, Column, Integer, String, DateTime
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, Session
from datetime import datetime
import requests
import os

BASE_DIR = os.path.dirname(os.path.abspath(__file__))
DATABASE_URL = f"sqlite:///{'os.path.join(BASE_DIR, 'skud.db')}"
engine = create_engine(DATABASE_URL, connect_args={"check_same_thread": False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

# БД
class AccessLog(Base):
    __tablename__ = "access_log"
    id = Column(Integer, primary_key=True, index=True)
    uid = Column(String, index=True)
    status = Column(String)
    timestamp = Column(DateTime, default=datetime.utcnow)

class AccessKey(Base):
    __tablename__ = "access_keys"
    id = Column(Integer, primary_key=True, index=True)
    key_uid = Column(String, unique=True, index=True)
    user_id = Column(Integer, nullable=True)
    status = Column(Integer, default=1)
    assigned_at = Column(DateTime, default=datetime.utcnow)
    expires_at = Column(DateTime, nullable=True)

Base.metadata.create_all(bind=engine)

# Pydantic-схемы
class LogEvent(BaseModel):
    uid: str
    status: str

class KeyCreate(BaseModel):
    key_uid: str
    user_id: int = None
    expires_at: datetime = None

# Настройки ESP32
ESP32_IP = "192.168.50.42"
app = FastAPI()

```

					ПГУ 1.090301.01.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		74

```

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

def update_esp_whitelist():
    db = SessionLocal()
    try:
        keys = db.query(AccessKey).filter(AccessKey.status == 1).all()
        uid_list = [key.key_uid for key in keys]
        try:
            response = requests.post(f"http://{ESP32_IP}/update_whitelist",
json=uid_list, timeout=2)
            if response.status_code == 200:
                print(f"Whitelist sent to ESP32: {len(uid_list)} keys")
            else:
                print(f"ESP32 returned {response.status_code}")
        except Exception as e:
            print(f"Failed to update ESP32: {e}")
    finally:
        db.close()

@app.get("/")
def read_root():
    return FileResponse("index.html")

@app.post("/log")
def log_event(event: LogEvent, db: Session = Depends(get_db)):
    db_log = AccessLog(uid=event.uid, status=event.status)
    db.add(db_log)
    db.commit()
    return {"message": "ok"}

@app.post("/learn_key")
def learn_key(data: dict, db: Session = Depends(get_db)):
    uid = data.get("uid")
    if not uid:
        raise HTTPException(400, "uid required")
    existing = db.query(AccessKey).filter(AccessKey.key_uid == uid).first()
    if not existing:
        new_key = AccessKey(key_uid=uid, user_id=None, status=1)
        db.add(new_key)
        db.commit()

```

```

        update_esp_whitelist()
    else:
        if existing.status == 0:
            existing.status = 1
            db.commit()
            update_esp_whitelist()
        return {"message": "key added"}
@app.get("/logs")
def get_logs(limit: int = 100, db: Session = Depends(get_db)):
    logs =
db.query(AccessLog).order_by(AccessLog.timestamp.desc()).limit(limit).all()
    return logs
@app.get("/keys")
def get_keys(db: Session = Depends(get_db)):
    keys = db.query(AccessKey).all()
    return keys
@app.post("/keys")
def add_key(key: KeyCreate, db: Session = Depends(get_db)):
    existing = db.query(AccessKey).filter(AccessKey.key_uid ==
key.key_uid).first()
    if existing:
        raise HTTPException(400, "Key already exists")
    db_key = AccessKey(key_uid=key.key_uid, user_id=key.user_id, status=1,
expires_at=key.expires_at)
    db.add(db_key)
    db.commit()
    update_esp_whitelist()
    return db_key
@app.delete("/keys/{key_id}")
def delete_key(key_id: int, db: Session = Depends(get_db)):
    key = db.query(AccessKey).filter(AccessKey.id == key_id).first()
    if not key:
        raise HTTPException(404, "Key not found")
    db.delete(key)
    db.commit()
    update_esp_whitelist()
    return {"message": "deleted"}
@app.patch("/keys/{key_id}/block")
def block_key(key_id: int, db: Session = Depends(get_db)):
    key = db.query(AccessKey).filter(AccessKey.id == key_id).first()
    if not key:
        raise HTTPException(404, "Key not found")

```

```

        key.status = 0 if key.status == 1 else 1
        db.commit()
        update_esp_whitelist()
        return {"status": key.status}
@app.post("/open")
def remote_open():
    try:
        requests.post(f"http://{ESP32_IP}/open", timeout=2)
    except Exception:
        raise HTTPException(500, "ESP32 unreachable")
    db = SessionLocal()
    try:
        db_log = AccessLog(uid="ADMIN_OPEN", status="GRANTED")
        db.add(db_log)
        db.commit()
    finally:
        db.close()
    return {"message": "door opened"}
@app.post("/enter_learn_mode")
def enter_learn_mode():
    try:
        requests.post(f"http://{ESP32_IP}/learn_mode", timeout=2)
    except Exception:
        raise HTTPException(500, "ESP32 unreachable")
    return {"message": "learn mode activated"}

```

Файл «index.html»

```

<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>СКУД Администратор</title>
    <link
href="https://cdn.jsdelivrivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
    <style>
        .granted { background-color: #d4edda !important; }
        .denied { background-color: #f8d7da !important; }
        .button { background-color: #fff3cd !important; }
        table { margin-top: 20px; }
        .nav-tabs { margin-bottom: 20px; }
    </style>
</head>
<body>
<div class="container mt-4">
    <h1>Управление СКУД</h1>
    <ul class="nav nav-tabs" id="myTab" role="tablist">
        <li class="nav-item"><button class="nav-link active" id="logs-tab" data-
bs-toggle="tab" data-bs-target="#logs" type="button" role="tab">Журнал
событий</button></li>

```

					ПГУ 1.090301.01.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

```

        <li class="nav-item"><button class="nav-link" id="keys-tab" data-bs-
toggle="tab" data-bs-target="#keys" type="button" role="tab">Управление
ключами</button></li>
        <li class="nav-item"><button class="nav-link" id="door-tab" data-bs-
toggle="tab" data-bs-target="#door" type="button" role="tab">Управление
дверью</button></li>
    </ul>
    <div class="tab-content mt-3">
        <!-- Журнал -->
        <div class="tab-pane fade show active" id="logs" role="tabpanel">
            <h3>Журнал доступа</h3>
            <table class="table table-bordered" id="logTable">

<thead><tr><th>ID</th><th>UID</th><th>Статус</th><th>Время</th></tr></thead>
                <tbody id="logBody"></tbody>
            </table>
        </div>
        <!-- Управление ключами -->
        <div class="tab-pane fade" id="keys" role="tabpanel">
            <h3>Добавить ключ</h3>
            <form id="addKeyForm">
                <div class="mb-3"><label class="form-label">UID ключа (16 hex
символов)</label><input type="text" id="keyUid" class="form-control" required
pattern="[A-Fa-f0-9]{16}"></div>
                <div class="mb-3"><label class="form-label">ID пользователя
(необязательно)</label><input type="number" id="userId" class="form-
control"></div>
                <button type="submit" class="btn btn-primary">Добавить
ключ</button>
            </form>
            <hr>
            <h3>Список ключей</h3>
            <table class="table table-bordered" id="keysTable">

<thead><tr><th>ID</th><th>UID</th><th>Статус</th><th>Действия</th></tr></thead>
                <tbody id="keysBody"></tbody>
            </table>
            <button id="learnModeBtn" class="btn btn-warning mt-3">Режим
обучения</button>
            <div id="learnMsg" class="mt-2"></div>
        </div>
        <!-- Управление дверью -->
        <div class="tab-pane fade" id="door" role="tabpanel">
            <button id="openDoorBtn" class="btn btn-danger">Открыть дверь</button>
            <div id="doorResult" class="mt-2"></div>
        </div>
    </div>
</div>

<script>
    async function loadLogs() {
        try {
            const response = await fetch('/logs?limit=50');
            const logs = await response.json();
            const tbody = document.getElementById('logBody');
            tbody.innerHTML = '';
            logs.forEach(log => {
                const row = tbody.insertRow();
                row.insertCell(0).innerText = log.id;
                row.insertCell(1).innerText = log.uid;
                const statusCell = row.insertCell(2);
                statusCell.innerText = log.status;
                let cssClass = '';

```

```

        if (log.status === 'GRANTED') cssClass = 'granted';
        else if (log.status === 'DENIED') cssClass = 'denied';
        else if (log.status === 'DOOR_OPEN') cssClass = 'button';
        else if (log.status === 'DOOR_CLOSED') cssClass = 'button';
        statusCell.className = cssClass;
        row.insertCell(3).innerText = new
Date(log.timestamp).toLocaleString();
    });
    } catch(e) { console.error(e); }
}
async function loadKeys() {
    const response = await fetch('/keys');
    const keys = await response.json();
    const tbody = document.getElementById('keysBody');
    tbody.innerHTML = '';
    keys.forEach(key => {
        const row = tbody.insertRow();
        row.insertCell(0).innerText = key.id;
        row.insertCell(1).innerText = key.key_uid;
        row.insertCell(2).innerText = key.status === 1 ? 'Активен' :
'Заблокирован';
        const actions = row.insertCell(3);
        const blockBtn = document.createElement('button');
        blockBtn.innerText = key.status === 1 ? 'Блокировать' :
'Разблокировать';
        blockBtn.className = 'btn btn-sm btn-secondary me-1';
        blockBtn.onclick = async () => {
            await fetch(`/keys/${key.id}/block`, { method: 'PATCH' });
            await loadKeys(); await loadLogs();
        };
        const delBtn = document.createElement('button');
        delBtn.innerText = 'Удалить';
        delBtn.className = 'btn btn-sm btn-danger';
        delBtn.onclick = async () => {
            if (confirm('Удалить ключ?')) {
                await fetch(`/keys/${key.id}`, { method: 'DELETE' });
                await loadKeys(); await loadLogs();
            }
        };
        actions.appendChild(blockBtn);
        actions.appendChild(delBtn);
    });
}
document.getElementById('addKeyForm').addEventListener('submit', async (e) =>
{
    e.preventDefault();
    const keyUid = document.getElementById('keyUid').value;
    const userId = document.getElementById('userId').value || null;
    const response = await fetch('/keys', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ key_uid: keyUid, user_id: userId })
    });
    if (response.ok) {
        alert('Ключ добавлен');
        await loadKeys(); await loadLogs();
        document.getElementById('addKeyForm').reset();
    } else {
        alert('Ошибка: ключ уже существует или неверный формат');
    }
});
document.getElementById('openDoorBtn').addEventListener('click', async () => {
    const resultDiv = document.getElementById('doorResult');

```

```

        try {
            const response = await fetch('/open', { method: 'POST' });
            if (response.ok) {
                resultDiv.innerHTML = '<div class="alert alert-success">Дверь
открыта</div>';
                await loadLogs();
            } else {
                resultDiv.innerHTML = '<div class="alert alert-danger">Ошибка
открытия</div>';
            }
        } catch (err) {
            resultDiv.innerHTML = '<div class="alert alert-danger">Сервер
недоступен</div>';
        }
        setTimeout(() => resultDiv.innerHTML = '', 3000);
    });
    document.getElementById('learnModeBtn').addEventListener('click', async () =>
    {
        const msgDiv = document.getElementById('learnMsg');
        try {
            const response = await fetch('/enter_learn_mode', { method: 'POST' });
            if (response.ok) {
                msgDiv.innerHTML = '<div class="alert alert-info">Режим обучения
активирован. Приложите ключ к считывателю.</div>';
                setTimeout(() => msgDiv.innerHTML = '', 5000);
            } else {
                msgDiv.innerHTML = '<div class="alert alert-danger">Ошибка
активации режима обучения</div>';
            }
        } catch {
            msgDiv.innerHTML = '<div class="alert alert-danger">ESP32
недоступна</div>';
        }
    });
    setInterval(loadLogs, 5000);
    loadLogs();
    loadKeys();
</script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"
></script>
</body>
</html>

```

Файл «Plata.ino»

```

#include <WiFi.h>
#include <WebServer.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <Preferences.h>
#include <OneWire.h>

// НАСТРОЙКИ СЕТИ
const char* ssid = "Asus-RT-AX53U";
const char* password = "119607Vg11!";
const char* server_host = "192.168.50.100";
const int server_port = 8000;

// ПИНЫ
#define ONE_WIRE_BUS 4          // DATA ТМ-считывателя
#define RELAY_PIN 27           // управление реле (замок)
#define DOOR_PIN 13            // геркон (датчик двери)

```

					ПГУ 1.090301.01.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

```

// БЕЛЫЙ СПИСОК
#define MAX_KEYS 100
String whitelist[MAX_KEYS];
int whitelistCount = 0;
Preferences prefs;

bool learnMode = false;

OneWire oneWire(ONE_WIRE_BUS);
WebServer server(80);

void loadWhitelist();
void saveWhitelist();
void addKey(String uid);
bool checkKey(String uid);
void sendEvent(String uid, String status);
void openLock(String source);
void handleUpdateWhitelist();
void handleOpen();
void handleLearnMode();
void fetchWhitelistFromServer();

// SETUP
void setup() {
    Serial.begin(115200);
    pinMode(RELAY_PIN, OUTPUT);
    digitalWrite(RELAY_PIN, LOW);
    pinMode(DOOR_PIN, INPUT_PULLUP);

    loadWhitelist();

    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nWiFi connected");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());

    server.on("/update_whitelist", HTTP_POST, handleUpdateWhitelist);
    server.on("/open", HTTP_POST, handleOpen);
    server.on("/learn_mode", HTTP_POST, handleLearnMode);
    server.begin();
    Serial.println("HTTP server started");

    fetchWhitelistFromServer();
}

// ОСНОВНОЙ ЦИКЛ
void loop() {
    server.handleClient();

    // Обработка геркона (состояние двери)
    static int lastDoorState = 1;
    int doorState = digitalRead(DOOR_PIN);
    if (doorState != lastDoorState) {
        lastDoorState = doorState;
        String status = (doorState == HIGH) ? "DOOR_OPEN" : "DOOR_CLOSED";
        sendEvent("DOOR_SENSOR", status);
        Serial.println(status);
    }
}

```

```

// Поиск ТМ-ключа
byte address[8];
if (oneWire.search(address)) {
    if (OneWire::crc8(address, 7) == address[7]) {
        char uidStr[17];
        for (byte i = 0; i < 8; i++) {
            sprintf(&uidStr[i*2], "%02X", address[i]);
        }
        String uid = String(uidStr);
        Serial.print("Key: ");
        Serial.println(uid);

        if (learnMode) {
            addKey(uid);
            sendEvent(uid, "LEARNED");
            learnMode = false;
            Serial.println("Learn mode OFF, key added");
            HTTPClient http;
            String url = "http://" + String(server_host) + ":" + String(server_port) +
"/learn_key";
            http.begin(url);
            http.addHeader("Content-Type", "application/json");
            String payload = "{\"uid\":\"" + uid + "\"}";
            http.POST(payload);
            http.end();
        } else {
            if (checkKey(uid)) {
                Serial.println("GRANTED");
                openLock("KEY");
                sendEvent(uid, "GRANTED");
            } else {
                Serial.println("DENIED");
                sendEvent(uid, "DENIED");
            }
        }
    } else {
        Serial.println("CRC error");
    }
    oneWire.reset_search();
    delay(500);
}
delay(100);
}

// РАБОТА С WHITELIST
void loadWhitelist() {
    prefs.begin("whitelist", true);
    whitelistCount = prefs.getInt("count", 0);
    for (int i = 0; i < whitelistCount && i < MAX_KEYS; i++) {
        whitelist[i] = prefs.getString(("key" + String(i)).c_str(), "");
    }
    prefs.end();
    Serial.print("Loaded whitelist: ");
    Serial.println(whitelistCount);
}

void saveWhitelist() {
    prefs.begin("whitelist", false);
    prefs.putInt("count", whitelistCount);
    for (int i = 0; i < whitelistCount; i++) {
        prefs.putString(("key" + String(i)).c_str(), whitelist[i]);
    }
}

```

```

    for (int i = whitelistCount; i < MAX_KEYS; i++) {
        prefs.remove(("key" + String(i)).c_str());
    }
    prefs.end();
}

void addKey(String uid) {
    if (whitelistCount >= MAX_KEYS) {
        Serial.println("Whitelist full");
        return;
    }
    for (int i = 0; i < whitelistCount; i++) {
        if (whitelist[i] == uid) return;
    }
    whitelist[whitelistCount++] = uid;
    saveWhitelist();
}

bool checkKey(String uid) {
    for (int i = 0; i < whitelistCount; i++) {
        if (whitelist[i] == uid) return true;
    }
    return false;
}

// ОТПРАВКА НА СЕРВЕР
void sendEvent(String uid, String status) {
    if (WiFi.status() != WL_CONNECTED) return;
    HTTPClient http;
    String url = "http://" + String(server_host) + ":" + String(server_port) +
"/log";
    http.begin(url);
    http.addHeader("Content-Type", "application/json");
    StaticJsonDocument<200> doc;
    doc["uid"] = uid;
    doc["status"] = status;
    String payload;
    serializeJson(doc, payload);
    http.POST(payload);
    http.end();
}

// УПРАВЛЕНИЕ ЗАМКОМ
void openLock(String source) {
    digitalWrite(RELAY_PIN, HIGH);
    delay(2000);
    digitalWrite(RELAY_PIN, LOW);
    Serial.print("Lock opened by ");
    Serial.println(source);
}

// HTTP-ОБРАБОТЧИКИ
void handleUpdateWhitelist() {
    if (!server.hasArg("plain")) {
        server.send(400, "text/plain", "No body");
        return;
    }
    String body = server.arg("plain");
    DynamicJsonDocument doc(4096);
    DeserializationError error = deserializeJson(doc, body);
    if (error) {
        server.send(400, "text/plain", "Invalid JSON");
        return;
    }
}

```

```

    }
    JSONArray arr = doc.as<JSONArray>();
    int newCount = 0;
    for (JsonVariant v : arr) {
        if (newCount < MAX_KEYS) {
            whitelist[newCount++] = v.as<String>();
        }
    }
    whitelistCount = newCount;
    saveWhitelist();
    server.send(200, "text/plain", "Whitelist updated");
    Serial.println("Whitelist updated via HTTP");
}

void handleOpen() {
    openLock("REMOTE");
    sendEvent("ADMIN_OPEN", "GRANTED");
    server.send(200, "application/json", "{\"status\":\"ok\"}");
}

void handleLearnMode() {
    learnMode = true;
    server.send(200, "application/json", "{\"status\":\"learn_mode_activated\"}");
    Serial.println("Learn mode activated");
}

void fetchWhitelistFromServer() {
    if (WiFi.status() != WL_CONNECTED) return;
    HTTPClient http;
    String url = "http://" + String(server_host) + ":" + String(server_port) +
"/keys";
    http.begin(url);
    int httpCode = http.GET();
    if (httpCode == 200) {
        String payload = http.getString();
        DynamicJsonDocument doc(4096);
        DeserializationError error = deserializeJson(doc, payload);
        if (!error && doc.is<JSONArray>()) {
            JSONArray arr = doc.as<JSONArray>();
            int cnt = 0;
            for (JsonVariant v : arr) {
                if (v["status"].as<int>() == 1 && cnt < MAX_KEYS) {
                    whitelist[cnt++] = v["key_uid"].as<String>();
                }
            }
            whitelistCount = cnt;
            saveWhitelist();
            Serial.print("Fetched whitelist from server: ");
            Serial.println(whitelistCount);
        }
    }
    http.end();
}

```

Приложение Г
(обязательное)

Программно-аппаратное решение системы управления и контроля доступом
Презентация

					ПГУ 1.090301.01.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		85

Программно-аппаратное решение системы управления и контроля доступом

Автор работы:

Студент гр. 22ВВВ1 Ганин Иван Романович

Научный руководитель:

д.т.н., зав. Каф. ВТ Митрохин Максим Александрович

Пенза, 2026

Цель и задачи работы

Цель – разработка программно-аппаратной СКУД, обеспечивающей идентификацию по ТМ-ключам, централизованное управление правами доступа, ведение журнала событий и удалённое администрирование через веб-интерфейс.

Задачи:

- Выполнить анализ существующих решений;
- Выбрать аппаратные средства;
- Разработать архитектуру системы;
- Разработать прошивку ESP32 и серверную часть;
- Создать веб-интерфейс администратора;
- Провести тестирование и оценить эффективность.



Достоинства: <ul style="list-style-type: none">• Высокая надёжность;• развитая логика;• интеграция с учётом рабочего времени.	Достоинства: <ul style="list-style-type: none">• Единая экосистема с видеонаблюдением;• мощный софт;• гибкие отчёты.	Достоинства: <ul style="list-style-type: none">• Доступная стоимость;• открытость SDK для интеграции.
Недостатки: <ul style="list-style-type: none">• Закрытый протокол;• высокая стоимость лицензий;• привязка к вендору.	Недостатки: <ul style="list-style-type: none">• Перегруженный интерфейс;• высокая ресурсоёмкость;• платные обновления.	Недостатки: <ul style="list-style-type: none">• Устаревший дизайн ПО;• ограниченная настройка временных зон.

Используемые технологии

Программные средства



Аппаратные средства



Архитектура приложения



Принцип работы системы

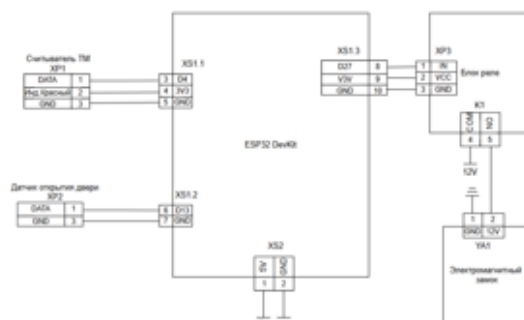
- 1) ESP32 подключается к Wi-Fi, загружает белый список ключей с сервера, запускает HTTP-сервер для команд.
- 2) Пользователь прикладывает ТМ-ключ к считывателю. ESP32 считывает UID, проверяет CRC и сравнивает с белым списком.
- 3) Если ключ есть в базе данных, то ESP32 подаёт сигнал на реле, замок открывается на 2 секунды и отправляет на сервер событие GRANTED.
- 4) Если ключа в базе нет или он заблокирован, то замок не открывается и на сервер отправляется событие DENIED.
- 5) При изменении состояния двери ESP32 отправляет события DOOR_OPEN / DOOR_CLOSED.
- 6) Когда администратор активирует режим обучения, следующий приложенный ключ автоматически добавляется в белый список и БД.
- 7) Все события сохраняются в SQLite и отображаются в веб-интерфейсе в реальном времени.

Серверная часть

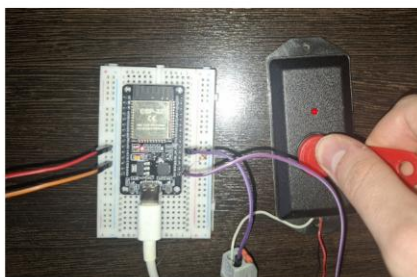
СКУД Панель управления

Журнал событий Управление системой Управление дверью

ID	UID	Статус	Время
40	0195F4CB010000D8	GRANTED	25.05.2026, 22:15:22
41	0195F4CB010000D8	GRANTED	25.05.2026, 22:15:19
44	0195F4CB010000D8	GRANTED	25.05.2026, 22:15:16
43	0195F4CB010000D8	GRANTED	25.05.2026, 22:15:13
42	0195F4CB010000D8	GRANTED	25.05.2026, 22:15:09
41	0195F4CB010000D8	GRANTED	25.05.2026, 22:15:06
40	0195F4CB010000D8	GRANTED	25.05.2026, 22:15:03
39	0195F4CB010000D8	GRANTED	25.05.2026, 22:14:59



Аппаратная часть



```
Learn mode activated
Found key: 0195F4CB010000D8
Whitelist saved to flash
HTTP POST success, code: 500
Learn mode disabled, key added
Found key: 0195F4CB010000D8
Access GRANTED
Lock opened by KEY
HTTP POST success, code: 500
```

Тестирование системы

Тестируемый сценарий	Результат
Прикладывание разрешённого ключа	Замок открывается, в журнал пишется GRANTED
Прикладывание запрещённого ключа	Замок не открывается, в журнал DENIED
Режим обучения: активация и прикладывание нового ключа	Ключ добавляется в БД и белый список ESP32, замок открывается
Удалённое открытие двери через веб-интерфейс	ESP32 получает команду, замок открывается, в журнале ADMIN_OPEN
Контроль геркона (опционально)	При открытии двери в журнале DOOR_OPEN, при закрытии – DOOR_CLOSED
Просмотр журнала в реальном времени	Таблица обновляется каждые 5 секунд, цветовое кодирование строк

Результаты

- Проведён анализ предметной области и существующих аналогов;
- Разработана архитектура программно-аппаратной системы;
- Разработана и отлажена прошивка ESP32 на C++;
- Разработано и отлажено серверное приложение на Python и базой данных SQLite;
- Разработан и отлажен веб-интерфейс;
- Выполнено тестирование системы;
- Система обеспечивает централизованное логирование, динамическое обновление белого списка на ESP32 и удалённое администрирование.

В дальнейшем проект можно расширить, добавив временные зоны доступа, фрование канала связи, поддержку нескольких контроллеров, экспорта отчётов в PDF/Excel. Также можно интегрировать видеонаблюдение, ши (HTTPS) и усиления аутентификации;

Спасибо за внимание!

					ПГУ 1.090301.01.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		89