



Министерство науки и высшего образования Российской Федерации  
Калужский филиал федерального государственного автономного  
образовательного учреждения высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(КФ МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ «ИНФОРМАТИКА И УПРАВЛЕНИЕ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ, ИНФОРМАЦИОННЫЕ  
ТЕХНОЛОГИИ»

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## К КУРСОВОМУ ПРОЕКТУ

### НА ТЕМУ:

*Разработка клиент-серверного приложения для сбора и*  
*анализа телеметрических данных движения автомобиля на*  
*базе ESP32*

Студент группы ИУК4-72Б

\_\_\_\_\_  
(подпись, дата)

Е.В. Губин

\_\_\_\_\_  
(И.О. Фамилия)

Руководитель курсового проекта

\_\_\_\_\_  
(подпись, дата)

Е.В. Красавин

\_\_\_\_\_  
(И.О. Фамилия)

Калуга, 2025

Министерство науки и высшего образования Российской Федерации  
Калужский филиал федерального государственного автономного образовательного  
учреждения высшего образования  
«Московский государственный технический университет имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(КФ МГТУ им. Н.Э. Баумана)

---

УТВЕРЖДАЮ  
Заведующий кафедрой ИУК4  
( Ю.Е. Гагарин )  
« 05 » сентября 2025г.

**ЗАДАНИЕ**  
**на выполнение курсового проекта**

по дисциплине **Компьютерные сети и Интернет-технологии**

Студент группы **ИУК4-72Б Губин Егор Вячеславович**  
(фамилия, имя, отчество)

Тема курсового проекта **Разработка клиент-серверного приложения для сбора и анализа телеметрических данных движения автомобиля на базе ESP32**

Направленность КП **учебный**  
Источник тематики **кафедра ИУК4**

**Задание**

*Провести анализ требований и технологий разработки программного обеспечения.*

*Выполнить проектирование программного обеспечения.*

*Осуществить интеграция компонентов программного обеспечения.*

**Оформление курсового проекта**

*Расчетно-пояснительная записка на \_\_\_\_\_ листах формата А4.*

*Перечень графического материала КП (плакаты, схемы, чертежи и т.п.):*

- *Функциональная модель сервиса – 1 лист формата А3;*
- *Структура базы данных – 1 лист формата А3.*
- *Схема подключения – 1 лист формата А3;*

Дата выдачи задания « 05 » сентября 2025 г.

Руководитель \_\_\_\_\_ 05.09.2025  
(подпись, дата)

Студент \_\_\_\_\_ 05.09.2025  
(подпись, дата)

Е.В. Красавин

(И.О. Фамилия)

Е.В. Губин

(И.О. Фамилия)

Министерство науки и высшего образования Российской Федерации  
Калужский филиал федерального государственного автономного образовательного  
учреждения высшего образования  
«Московский государственный технический университет имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(КФ МГТУ им. Н.Э. Баумана)

## КАЛЕНДАРНЫЙ ПЛАН на выполнение курсового проекта

по дисциплине **Компьютерные сети и Интернет-технологии**

Студент группы **ИУК4-72Б Губин Егор Вячеславович**  
(фамилия, имя, отчество)

Тема курсового проекта **Разработка клиент-серверного приложения для сбора и анализа телеметрических данных движения автомобиля на базе ESP32**

№	Наименование этапов	Сроки выполнения этапов		Отметка о выполнении	
		план	факт	Руководитель	Куратор
1	Задание на выполнение	1-я нед.			
2	Выполнение логического проектирования программного обеспечения	10-я нед.			
3	Выполнение и окончательное оформление графической части и расчетно-пояснительной записки	14-я нед.			
4	Защита	17-я нед.			

Студент \_\_\_\_\_ 05.09.2025г. \_\_\_\_\_ 05.09.2025г.  
(подпись, дата) (подпись, дата)

## РЕФЕРАТ

Расчетно-пояснительная записка 50 с., 2 рисунка, 16 источников.

Разработка клиент-серверной системы мониторинга телеметрии транспортного средства на основе протокола MQTT

Объектом разработки является программно-аппаратный комплекс, обеспечивающий сбор, передачу, хранение и анализ телеметрических данных, получаемых от транспортных средств через бортовой контроллер и модемную связь.

Цель проекта – создание клиент-серверной системы, позволяющей в реальном времени получать данные о местоположении, состоянии и параметрах работы транспортного средства, визуализировать их в удобном пользовательском интерфейсе, а также анализировать исторические данные для последующего мониторинга и диагностики.

Поставленные задачи решаются путем проектирования и разработки клиент-серверного взаимодействия серверного и клиентского приложений с помощью выбранных инструментов и технологий.

Результатом работы является полноценная система, обеспечивающая прием телеметрических данных, их обработку и сохранение, отображение информации об активной сессии на карте, вывод основных параметров в реальном времени, просмотр маршрутов и графиков телеметрии за выбранный период.

## СОДЕРЖАНИЕ

РЕФЕРАТ .....	4
СОДЕРЖАНИЕ.....	5
ВВЕДЕНИЕ.....	6
1. АНАЛИЗ ТРЕБОВАНИЙ И ТЕХНОЛОГИЙ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. 9	
1.1 Техническое задание .....	9
1.2 Анализ существующих аналогов и прототипов .....	12
1.3 Обоснование выбора инструментов и платформы для разработки .....	16
1.4 Методологии и стандарты, используемые при проектировании системы .....	19
2. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.....	22
2.1 Разработка структуры системы.....	22
2.2 Разработка клиент-серверной архитектуры системы.....	23
2.3 Разработка структуры базы данных.....	26
2.4 Проектирование модуля сбора телеметрии на базе ESP32 .....	28
2.5 Проектирование модуля обработки и упаковки данных.....	31
2.6 Проектирование канала передачи данных на основе MQTT .....	34
2.7 Проектирование клиентского приложения для мониторинга и анализа .....	37
3. КОНТРОЛЬ КАЧЕСТВА И ИНТЕГРАЦИЯ КОМПОНЕНТОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ .....	40
3.1 Методика тестирования аппаратного и программного комплекса .....	40
3.2 Интеграционное тестирование модулей системы .....	44
3.3 Тестирование клиентского приложения.....	46
ЗАКЛЮЧЕНИЕ.....	48
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	49

## ВВЕДЕНИЕ

Современные транспортные средства оснащаются большим количеством электронных систем, обеспечивающих безопасное и эффективное управление автомобилем. Эти системы генерируют обширные телеметрические данные, содержащие информацию о скорости, частоте вращения двигателя, состоянии датчиков, параметрах работы силовой установки и других характеристиках. Дополнительно к этому, навигационные модули позволяют отслеживать местоположение транспортного средства в реальном времени, формируя тем самым полноценную картину движения. Анализ подобных данных открывает широкие возможности для оптимизации технического состояния автомобиля, мониторинга стиля вождения, повышения безопасности дорожного движения, а также создания интеллектуальных сервисов.

Актуальность разработки систем сбора и анализа телеметрии обусловлена ростом популярности решений в области IoT (Internet of Things) и транспортной телематики. Компактные микроконтроллеры с поддержкой сетевых протоколов позволяют реализовать недорогие и функциональные устройства, способные собирать параметры работы транспортного средства и передавать их на сервер для последующей обработки. Одним из таких контроллеров является ESP32, сочетающий высокую вычислительную производительность, поддержку беспроводных интерфейсов и гибкую программную архитектуру. Использование MQTT-протокола для обмена сообщениями обеспечивает лёгкость масштабирования, устойчивость системы к сетевым задержкам и возможность реального времени отображать данные на стороне клиента.

Объектом исследования является процесс сбора, передачи и анализа телеметрических данных транспортного средства.

Предметом исследования является архитектура и программная реализация системы телеметрии на базе микроконтроллера ESP32, включающая

сбор данных с CAN-шины, получение навигационной информации, формирование аналитических данных и их визуализацию.

Цель работы — разработка программно-аппаратного комплекса для сбора и анализа телеметрических данных движения автомобиля на базе микроконтроллера ESP32 с использованием OBD2/CAN, модуля NEO-6M, GSM-модема SIMCom и клиент-серверной архитектуры на основе MQTT.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Проанализировать существующие решения и выделить их преимущества и недостатки.
2. Сформировать техническое задание на разработку программно-аппаратного комплекса.
3. Обосновать выбор аппаратной платформы, инструментов разработки и сетевых технологий.
4. Разработать архитектуру системы, включающую модули сбора телеметрии, обработки данных, передачи сообщений и клиентскую часть.
5. Спроектировать структуру базы данных для хранения сессий, телеметрии и навигационных данных.
6. Реализовать прототип устройства на ESP32, обеспечивающего:
  - считывание CAN-кадров через модуль SN65HVD230,
  - получение координат от NEO-6M,
  - передачу данных на MQTT-брокер через SIMCom-модем.
7. Разработать клиентское приложение для мониторинга движения автомобиля в реальном времени и анализа завершённых сессий.
8. Провести тестирование и интеграцию всех компонентов системы.

9. Сформировать выводы о работоспособности, эффективности и возможностях дальнейшего развития разрабатываемого решения.

Результатом выполнения курсовой работы является программно-аппаратный комплекс, позволяющий в реальном времени собирать телеметрические и геоданные автомобиля, передавать их на сервер и отображать на клиентском приложении, обеспечивая основу для дальнейшего анализа и расширения функциональности системы.



# **1. АНАЛИЗ ТРЕБОВАНИЙ И ТЕХНОЛОГИЙ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## **1.1 Техническое задание**

### **Требования к функциям, выполняемым системой**

Система должна обеспечивать сбор, передачу, хранение, отображение и анализ телеметрических данных и навигационной информации транспортного средства. Для этого она должна предоставлять следующие функции:

- Сбор телеметрических данных автомобиля через интерфейс OBD2/CAN с использованием стандарта SAE J1979 и CAN 2.0.
- Считывание GPS-координат с модуля NEO-6M в формате NMEA-сообщений.
- Получение и обработку поддерживаемых PID-команд автомобиля по протоколу OBD2 (сервис 0x01, диапазон PID 0x01–0x20).
- Циклический опрос поддерживаемых PID с формированием корректных диагностических запросов и обработкой полученных значений (скорость, частота вращения двигателя и др.).
- Формирование структуры данных для передачи: телеметрические данные, геоданные, служебные сообщения о начале и завершении сессии.
- Установление связи с серверной частью через GSM-модем SIMCom и отправка данных по протоколу MQTT.
- Отображение в реальном времени положения транспортного средства на карте и его текущей телеметрии на стороне клиентского приложения.
- Обеспечение возможности просмотра завершённых сессий, их анализа и визуализации данных в виде графиков.
- Хранение телеметрических данных, геоданных и информации о сессиях в базе данных MySQL.

- Идентификация устройств по уникальному MAC-адресу ESP32 и выбор активного контроллера для мониторинга.

### **Требования к клиентской части системы**

Клиентская часть (приложение Windows на C# WinForms) должна обеспечивать:

- Удобный и интуитивно понятный графический интерфейс для отображения геолокации автомобиля и телеметрических данных в реальном времени.
- Возможность выбора контроллера по MAC-адресу для подключения к текущей активной сессии.
- Построение графиков телеметрических параметров по завершённым сессиям.
- Проверку корректности и полноты получаемых через MQTT данных, уведомление пользователя об ошибках связи или отсутствии ответов от сервера.
- Отображение карты с использованием стороннего API (например, GMap.NET или аналогов), включая масштабирование, перемещение, обновление данных в реальном времени.
- Высокую производительность при отображении данных, включая плавное обновление карты и графиков, стабильную обработку входящих MQTT-сообщений.
- Надёжную работу при нестабильном интернет-соединении, автоматическое восстановление связи с брокером.
- Возможность экспорта собранных данных в популярные форматы (CSV).

## **Требования к серверной части системы**

Серверная часть, включающая MQTT-брокер и базу данных MySQL, должна обеспечивать:

- Приём всех сообщений от телеметрических модулей ESP32 по протоколу MQTT.
- Незамедлительную рассылку входящих данных всем подписанным клиентам (режим реального времени).
- Сохранение данных в базе MySQL согласно структуре: контроллеры, сессии, телеметрия, геоданные, PID-команды и ответы.
- Обработку исключительных ситуаций, ведение журналов ошибок и уведомление клиентов при невозможности обработки данных.
- Масштабируемость системы за счёт лёгкости подключения дополнительных телеметрических устройств.
- Возможность развертывания серверной части на ОС Windows или Linux.
- Минимальные задержки обработки данных благодаря оптимизации структуры БД и корректным настройкам MQTT-брокера.
- Надёжность канала связи, устойчивость к сетевым перепадам и пакетным потерям.
- Контроль уникальности устройств по MAC-адресу и корректное создание сессий при подключении ESP32.

## **Требования к аппаратной части системы**

Аппаратная часть устройства на базе ESP32 должна обеспечивать:

- Подключение к CAN-шине автомобиля через трансивер SN65HVD230.
- Электрическую безопасность и питание от бортовой сети автомобиля через модуль LM2596S с понижением напряжения до 5 В.

- Стабильную работу микроконтроллера ESP32 под управлением FreeRTOS.
- Подключение GPS-антенны к модулю NEO-6M и получение данных с частотой 5 Гц.
- Подключение GSM-модуля SIMCom для передачи данных на сервер.
- Корректное определение начала и конца сессии на основе подачи питания.
- Минимальную задержку при обработке CAN-кадров и NMEA-сообщений, использование очередей FreeRTOS.

### **Требования к надёжности и качеству работы**

Система должна обеспечивать:

- Устойчивую передачу данных при нестабильной GSM-связи.
- Защиту данных при передаче через MQTT с использованием уникальных идентификаторов устройств.
- Минимизацию задержек между моментом получения телеметрии и её отображением на карте.
- Автоматическую перезагрузку ESP32 при отсутствии ответа от CAN-шины.
- Корректную работу в течение длительных периодов времени (часы/сутки).

## **1.2 Анализ существующих аналогов и прототипов**

Системы телеметрии и мониторинга транспортных средств широко представлены на рынке решений для автодиагностики и IoT-мониторинга. Однако комплексных систем, полностью аналогичных разрабатываемому программно-аппаратному комплексу и основанных на микроконтроллере

ESP32, сравнительно мало. Тем не менее существует ряд коммерческих и открытых решений, частично пересекающихся по функциональности, которые можно рассматривать как аналоги или прототипы.

### **Аналог 1: OBD-II Bluetooth/Wi-Fi адаптеры (ELM327 и производные)**

ODR-адаптеры на основе микросхемы ELM327 являются наиболее распространённым решением для диагностики автомобилей с использованием протокола OBD2.

#### **Преимущества:**

- низкая стоимость и широкая доступность;
- поддержка основных OBD2-протоколов;
- возможность подключения к мобильным устройствам через Bluetooth/Wi-Fi;
- совместимость со множеством приложений (Torque, Car Scanner, OBD Auto Doctor).

#### **Недостатки:**

- неполная и часто нестабильная поддержка CAN-протокола из-за подделок ELM327;
- ориентированность на разовое диагностирование, а не на постоянную телеметрию;
- отсутствие встроенного GPS-модуля или средств мобильной связи;
- отсутствие серверной части для анализа данных;
- нельзя использовать для длительного мониторинга или хранения сессий.

Данные адаптеры являются аналогами по принципу получения PID-команд, но не предоставляют функционал телеметрии в реальном времени, передачи данных на сервер или анализа сессий.

## **Аналог 2: Автомобильные трекеры (GPS+GSM)**

Коммерческие устройства, такие как GalileoSky, StarLine Маяк, АвтоФон, Teltonika FMB920, широко используются для мониторинга транспорта.

Преимущества:

- стабильный GSM-канал передачи данных;
- встроенный GPS-модуль;
- ориентация на длительное использование;
- поддержка серверных платформ для анализа и построения отчётов.

Недостатки:

- большинство устройств не читают OBD2 PID и не предоставляют телеметрию двигателя;
- закрытая прошивка и невозможность кастомизации;
- высокая стоимость по сравнению с ESP32-базированными решениями;
- передача данных осуществляется в закрытом проприетарном формате.

Эти устройства обеспечивают GPS-трекинг, но не выполняют глубокой диагностической телеметрии, что делает их частичными, а не полными аналогами.

## **Аналог 3: OBD-II трекеры (GPS + OBD + GSM)**

Пример: Xenon OBD2 GPS Tracker, FleetComplete, VjoyCar OBD Tracker.

Такие устройства сочетают трекер и OBD-клиент, позволяя получать скорость, RPM, ошибки двигателя.

Преимущества:

- измерение телеметрии и GPS-данных одновременно;
- удобство «вставил и работает»;
- ориентированы на коммерческие автопарки (логистику, такси).

Недостатки:

- поддержка только ограниченного набора PID;
- закрытые протоколы обмена;
- отсутствие возможности модификации или изучения работы системы;
- нет открытой серверной части для кастомизации анализа данных.

Это наиболее близкие прототипы, но они не предоставляют открытой архитектуры, не позволяют разработчику контролировать внутренние процессы и не подходят для исследовательских целей.

#### **Аналог 4: Open-source проекты на Arduino/ESP32 с OBD-II**

Существуют проекты энтузиастов, такие как:

- Arduino OBD-II GPS Logger,
- ESP32 OBD2 with CAN bus,
- Freematics OBD-II Telematics Kit.

Преимущества:

- открытый исходный код;
- использование CAN-трансиверов и GPS-модулей, аналогично данной работе;
- возможность кастомизации и расширения функционала.

Недостатки:

- отсутствует полноценная серверная часть или она слишком упрощена;
- не реализованы механизмы анализа завершённых сессий;
- не используется MQTT в качестве универсального транспорта;
- зачастую ограничены только локальным логированием (на SD-карту или в память ESP32).

Несмотря на техническую близость, ни один из проектов не обеспечивает целостной архитектуры, включающей CAN-сбор, GPS, MQTT, сервер, БД и клиентское приложение.

### **1.3 Обоснование выбора инструментов и платформы для разработки**

При разработке системы было выбрано сочетание аппаратных и программных средств, оптимально подходящее для решения задачи мониторинга и передачи данных в реальном времени с использованием микроконтроллера и настольного программного обеспечения. Рассмотрим обоснование выбора каждого инструмента.

#### **Выбор аппаратной платформы**

Для разработки аппаратной части системы был выбран микроконтроллер ESP32 DevKitC v4. Данный контроллер обладает рядом преимуществ:

- высокая производительность благодаря двухъядерному процессору Tensilica LX6;
- поддержка шины I2C, SPI, UART, что упрощает подключение внешних датчиков;
- низкое энергопотребление;
- широкая популярность и развитое сообщество, что облегчает разработку и поиск документации.



ESP32 является оптимальным выбором для систем мониторинга, удалённой передачи данных и автоматизации, что полностью соответствует задачам проекта.

### **Выбор средств разработки встроенного ПО**

В качестве основной SDK использован ESP-IDF (Espressif IoT Development Framework), поскольку он является официальным и наиболее функциональным инструментарием для разработки под ESP32. Выбор обоснован следующими преимуществами:

- низкоуровневый доступ ко всем возможностям ESP32;
- высокая стабильность, официальная поддержка и регулярные обновления;
- встроенные библиотеки для работы с MQTT, TCP/IP, датчиками и периферией;
- поддержка FreeRTOS для реализации многозадачности.

Разработка велась в Espressif IDE, основанной на Eclipse, поскольку она:

- полностью интегрирована с ESP-IDF;
- поддерживает отладку, прошивку и мониторинг через USB;
- обеспечивает удобную структуру проекта.

### **Выбор протокола обмена данными**

Для передачи данных между ESP32 и ПК выбран протокол MQTT, так как он идеально подходит для IoT-устройств:

- низкое потребление трафика;
- поддержка работы в реальном времени;
- высокая устойчивость в условиях нестабильной сети;

- возможность обработки большого количества сообщений при минимальной нагрузке.

В качестве брокера используется Mosquitto, так как:

- он бесплатный и открытый;
- отличается высокой стабильностью;
- прост в установке и настройке;
- поддерживает аутентификацию, SSL и широкие возможности конфигурации.

MQTT обеспечивает надёжную доставку данных от микроконтроллера к настольному приложению.

### **Выбор базы данных**

Для хранения данных используется реляционная СУБД MySQL, выбранная по следующим причинам:

- высокая производительность при работе с большими объёмами данных;
- открытость и бесплатность;
- простота интеграции с C#;
- широкая распространённость и наличие инструментов администрирования (phpMyAdmin, MySQL Workbench);
- устойчивость и надёжность при длительной эксплуатации.

MySQL оптимально подходит для хранения логов, показаний сенсоров и данных мониторинга.

### **Выбор платформы и языка для настольного приложения**

Графический интерфейс системы реализован в виде настольного приложения на C# WinForms, что обусловлено следующими факторами:

- простота создания интерфейсов со множеством форм, таблиц, графиков и элементов управления;
- быстрая разработка благодаря инструментам Visual Studio;
- широкая поддержка .NET-библиотек, включая средства для работы с MQTT и MySQL;
- высокая стабильность и совместимость с Windows.

Использование Visual Studio 2022 позволяет реализовать современное, удобное и легко расширяемое приложение.

#### **1.4 Методологии и стандарты, используемые при проектировании системы**

При проектировании системы использовались современные инженерные подходы, направленные на обеспечение надёжности, расширяемости и удобства сопровождения как встроенного, так и настольного программного обеспечения. Основой разработки стал модульный принцип, позволяющий разделять систему на независимые функциональные части: прошивку микроконтроллера, брокер обмена сообщениями, клиентское приложение и базу данных. Такой подход упростил отладку, интеграцию и дальнейшее развитие системы.

Разработка велась итерационным методом: сначала был создан прототип обмена данными между ESP32 и брокером, затем — базовая версия настольного приложения, после чего выполнялась интеграция с MySQL и последовательное расширение функциональности. Итерационный процесс позволил своевременно корректировать архитектуру и требования, постепенно улучшая систему и устраняя выявленные недостатки.

При написании клиентской части на языке C# учитывались ключевые принципы объектно-ориентированного проектирования, включая SOLID. Логика приложения разделена на независимые компоненты, что позволило улучшить читаемость и тестируемость кода. Интерфейсная часть отделена от функциональной, что соответствует рекомендациям Microsoft по созданию

настольных приложений на базе WinForms. В коде соблюдены стандартные соглашения о наименовании и организации файлов, что облегчает поддержку проекта.

Встроенное программное обеспечение для ESP32 разрабатывалось в соответствии с рекомендациями Espressif и общими стандартами для микроконтроллеров. Использование ESP-IDF обеспечило единообразную структуру проекта, применение FreeRTOS — корректную реализацию многозадачности, а следование стандартам языка C и соглашениям производителя — стабильную работу прошивки. Дополнительно учитывались требования к управлению памятью и обработке ошибок, что особенно важно для встроенных систем.

При реализации сетевого взаимодействия использовалась спецификация протокола MQTT 3.1.1. Соблюдение правил организации тем, уровня качества доставки сообщений, механизма keep-alive и корректного восстановления соединений гарантирует устойчивый обмен данными между устройством и приложением. В качестве брокера применялся Mosquitto, соответствующий рекомендациям по конфигурации, безопасности и обработке сообщений.

Проектирование базы данных MySQL велось с учётом нормализации структуры, выделения первичных и внешних ключей, правильного выбора типов данных и использования индексов. Благодаря этому база данных получилась устойчивой, оптимизированной и пригодной для расширения по мере увеличения объёма регистрируемой информации.

Тестирование системы включало проверку корректности работы отдельных компонентов и их взаимодействия. Проводились проверки обмена данными через MQTT, устойчивости работы ESP32 при длительной работе, корректности обработки данных в MySQL и стабильности настольного приложения при взаимодействии с брокером и базой. Комбинация модульных и

интеграционных испытаний позволила обнаружить и устранить ошибки на ранних этапах разработки.

## **2. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

### **2.1 Разработка структуры системы**

Структура разрабатываемой системы представляет собой комплекс аппаратных и программных компонентов, обеспечивающих сбор, передачу, обработку и визуализацию телеметрических данных транспортного средства. Основу архитектуры составляет микроконтроллер ESP32 DevKitC v4, выполняющий функции центрального управляющего узла и обеспечивающий взаимодействие всех периферийных модулей.

Данные о состоянии автомобиля получаются через интерфейс OBD2 по CAN-шине. Для приема и передачи сигналов CAN используется трансивер SN65HVD230, подключенный к аппаратному контроллеру CAN (TWAI) в составе ESP32. Он обеспечивает гальваническое согласование и корректную работу с бортовой CAN-шиной автомобиля, формируя поток диагностических сообщений, содержащих параметры двигателя, скорость, обороты, температуру и другие индикаторы.

Параллельно с этим осуществляется получение данных о местоположении с помощью GPS-модуля NEO-6M. Он подключается к ESP32 по интерфейсу UART и обеспечивает вычисление координат, скорости движения и времени. Информация GPS объединяется с телеметрией OBD2, формируя единый поток данных о транспортном средстве.

Для обеспечения удаленной передачи данных используется модем SIMCom (SIM800/SIM7600), также работающий через UART. Он отвечает за передачу пакетов по сотовой сети и выступает связующим звеном между микроконтроллером и облачной частью системы. Основным протокол передачи данных — MQTT, что позволяет организовать надежный обмен сообщениями с брокером Mosquitto при минимальном трафике и низких задержках.

MQTT-брокер выступает центральным коммуникационным узлом программной части системы. Он принимает телеметрические сообщения от ESP32 и

обеспечивает их дальнейшую доставку серверному приложению. Серверная часть выполняет обработку, нормализацию и сохранение данных в базе MySQL. Использование реляционной базы данных позволяет хранить историческую информацию о поездках, положении автомобиля, диагностических параметрах и состоянии сетевого взаимодействия, обеспечивая дальнейший анализ и формирование отчетности.

Для отображения данных пользователю применяется клиентское приложение, разработанное с использованием C# и WinForms в Visual Studio 2022. Оно подключается к MQTT-брокеру для получения данных в реальном времени и к MySQL для просмотра архивных записей. Пользовательский интерфейс обеспечивает визуализацию параметров автомобиля, показ карты с координатами, а также выдачу диагностических уведомлений.

В результате структура системы представляет собой многоуровневую архитектуру, включающую аппаратный слой (ESP32 с периферией), коммуникационный слой (MQTT), серверный слой обработки и хранения данных (MySQL), а также клиентский интерфейс визуализации (WinForms). Такое распределение ролей делает систему масштабируемой, модульной и удобной для сопровождения, а также упрощает дальнейшее расширение функционала за счет добавления новых сенсоров, сетевых модулей или сервисов.

## **2.2 Разработка клиент-серверной архитектуры системы**

Клиент-серверная архитектура разработанной системы основана на распределённом взаимодействии между несколькими логическими уровнями, каждый из которых решает свою задачу: сбор телеметрии, транспорт сообщения, обработка данных и их визуализация. Такое разделение позволяет обеспечить независимость компонентов, гибкость масштабирования и устойчивость системы к сбоям отдельных узлов. В контексте использования

MQTT-брокера (рис 2.1) узлы системы подразделяются на 2 типа: подписчики (subscribers) и публикующие (publishers).

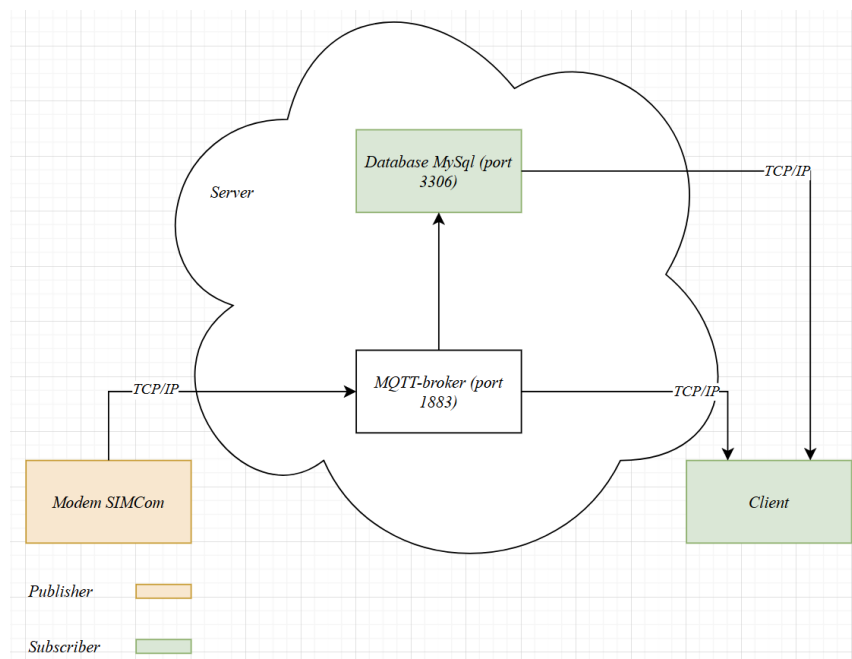


Рис 2.1 Клиент-серверная архитектура

На нижнем уровне находится аппаратный клиент — микроконтроллер ESP32, выполняющий функции устройства сбора данных. Он объединяет в себе три источника информации: данные CAN-шины автомобиля через SN65HVD230, координаты GPS с модуля NEO-6М и состояние сетевого соединения через модем SIMCom. Все данные предварительно нормализуются, объединяются в единый формат сообщения и передаются на серверную сторону по протоколу MQTT. Такой подход позволяет гарантировать потоковую передачу телеметрии с минимальной задержкой даже при нестабильной мобильной связи.

Связующим звеном между всеми компонентами является MQTT-брокер Mosquitto, выполняющий роль коммуникационного центра системы. Он обеспечивает получение, хранение и маршрутизацию сообщений от ESP32 к серверному приложению и клиентским интерфейсам. Использование MQTT позволяет эффективно разделять данные по тематическим каналам (topics), разгружая сеть и предоставляя возможность подключения нескольких



подписчиков без изменения логики устройства. Публикующие размещают информацию на топики (topics), а подписчики сразу же считывают эту информацию.

Серверная часть системы отвечает за обработку и долговременное хранение данных. Она подписывается на необходимые MQTT-темы, принимает поступающие от устройства сообщения и преобразует их в структуру, пригодную для сохранения в базе данных. В качестве СУБД используется MySQL, обеспечивающая хранение координат, диагностических параметров, идентификаторов сообщений OBD2, временных меток, состояний связи и другой телеметрии. Сервер реализует контроль целостности данных, обработку ошибок и управление бизнес-логикой, а также предоставляет методы доступа к информации клиентским приложениям.

Клиентская часть выполнена в виде настольного приложения на C# (WinForms), которое имеет два канала взаимодействия с системой: прямое подключение к MQTT-брокеру для получения данных в реальном времени и обращение к MySQL для просмотра архивной информации. Такой подход позволяет одновременно наблюдать текущее состояние автомобиля и выполнять анализ исторических поездок, диагностических параметров и событий. Интерфейс обеспечивает отображение GPS-положения, графиков телеметрии, диагностических статусов и уведомлений об ошибках.

Таким образом, клиент-серверная архитектура системы реализует четкое разделение функций между аппаратным уровнем, коммуникационным брокером, сервером обработки данных и клиентским приложением. MQTT обеспечивает эффективную и надежную передачу телеметрических сообщений, MySQL — структурированное хранение данных, а WinForms — удобное средство визуализации. Подобная архитектура легко масштабируется и позволяет расширять систему за счет подключения новых устройств, серверных сервисов или сторонних аналитических инструментов без изменения архитектуры основной платформы.

## 2.3 Разработка структуры базы данных

Для хранения телеметрических и географических данных, получаемых от контроллера на базе ESP32, была разработана реляционная база данных на основе СУБД MySQL. Структура базы данных проектировалась с учетом необходимости хранения нескольких ключевых сущностей: контроллеров, сессий их работы, телеметрии (ответов на PID-запросы), GPS-данных (NMEA), а также справочной информации о PID-командах и единицах измерения. Разработанная модель обеспечивает масштабируемость, целостность данных и независимость справочных таблиц от логических сущностей.

При проектировании использовалась ER-диаграмма (рис. 2.2), которая позволила определить связи между таблицами и выявить основные сущности предметной области.

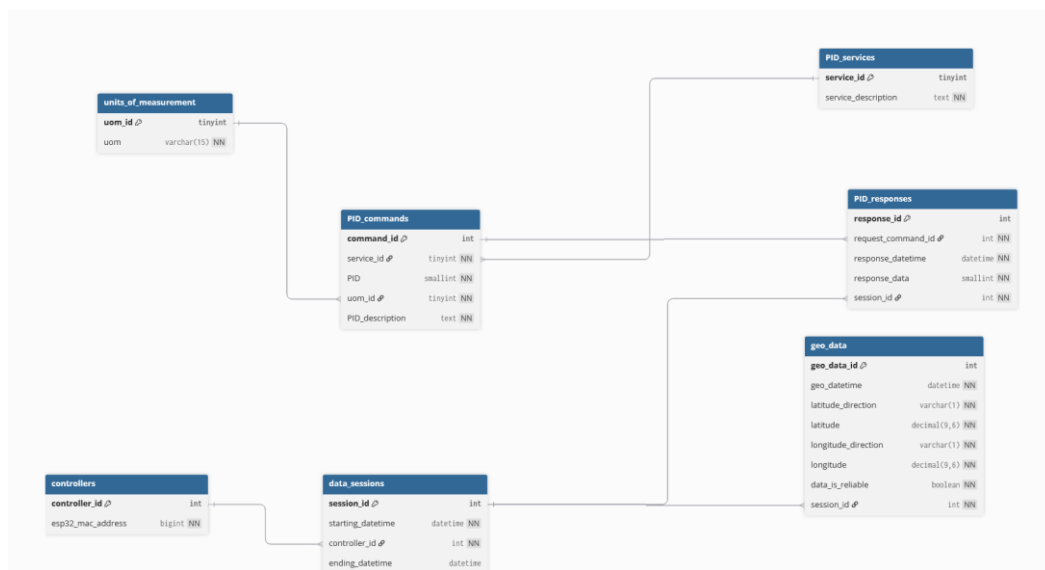


Рис. 2.2 ER-диаграмма

Каждому физическому устройству (контроллеру) соответствует запись в таблице *controllers*, содержащая его уникальный MAC-адрес. Это позволяет идентифицировать поступающие через MQTT данные и привязывать их к конкретному устройству.

Рабочий период устройства, определяемый временем включения и последующего отключения питания, описывается сущностью сессии. Для этих

целей используется таблица *data\_sessions*, содержащая дату начала, дату завершения и ссылку на контроллер. Таким образом, реализуется возможность хранить и анализировать многократные периоды работы одного и того же контроллера.

GPS-координаты, получаемые от навигационного модуля NEO-6M, сохраняются в таблице *geo\_data*. Для уменьшения избыточности данные привязываются не к контроллеру напрямую, а к конкретной сессии работы устройства. Таблица содержит:

- широту и долготу в десятичном формате,
- направление широты и долготы,
- метку достоверности данных (параметр Valid из строки GLL),
- точное время измерения.

Разделение геоданных и сессий позволяет строить траекторию движения по каждой сессии независимо.

Параметры, получаемые из бортовой сети автомобиля по протоколу OBD-II, разделены на две логические части: справочные сведения о PID и фактические ответы системы.

Для описания поддерживаемых сервисов OBD-II используется таблица *PID\_services*, содержащая описание каждого сервиса (например, сервис 0x01 — текущие данные автомобиля).

Информация о командах (PID), их описании и единицах измерения хранится в таблице *PID\_commands*, связанной с таблицами *PID\_services* и *units\_of\_measurement*. В *units\_of\_measurement* хранятся обозначения физических единиц (км/ч, об/мин, °C и т.д.). Такой подход обеспечивает гибкость и позволяет легко расширять базу новыми PID без изменения структуры данных.

Фактические телеметрические данные сохраняются в таблице *PID\_responses*. Каждая запись включает:

- ссылку на PID-команду,
- значение ответа,
- дату и время получения,
- ссылку на активную сессию.

Таким образом, можно построить временные графики любых параметров: скорость, обороты двигателя, температура, расход топлива и т.д.

Для поддержания связности и корректности данных используются внешние ключи:

- удаление контроллера приводит к удалению всех его сессий и связанных данных,
- удаление сессии автоматически очищает связанные GPS-данные и PID-ответы,
- удаление PID-команды приводит к удалению её измерений.

Это достигается за счет применения ON DELETE CASCADE, что делает систему устойчивой к неаккуратному удалению записей и позволяет поддерживать консистентность базы данных без дополнительных скриптов.

## **2.4 Проектирование модуля сбора телеметрии на базе ESP32**

Модуль сбора телеметрии является ключевым компонентом системы, обеспечивающим получение данных как из бортовой сети автомобиля через интерфейс CAN, так и от навигационного модуля NEO-6M по протоколу NMEA.

CAN (Controller Area Network) представляет собой широко распространенный протокол, применяемый в автомобильной промышленности

для обмена данными между электронными блоками управления. В рамках проекта используется классический CAN 2.0, включающий два варианта: CAN 2.0A (11-битовый идентификатор) и CAN 2.0B (29-битовый идентификатор). В автомобилях OBD-II, как правило, применяется режим CAN 11 бит с частотой 500 кбит/с. Физический и канальный уровни стандартизованы документами ISO-11898-1 (канальный уровень) и ISO-11898-2 (физический уровень).

CAN-шина работает по дифференциальному принципу и использует две линии — CANH и CANL. При передаче доминантного уровня разность потенциалов между линиями увеличивается ( $CANH \approx 3.5\text{ V}$ ,  $CANL \approx 1.5\text{ V}$ ), а при рецессивном уровне обе линии стремятся к примерно 2.5 V. Такой подход обеспечивает высокую помехоустойчивость, что особенно важно в условиях автомобильной среды. Для подключения ESP32 используется внешний трансивер SN65HVD230, обеспечивающий преобразование логических уровней контроллера к дифференциальным уровням шины.

Кадры CAN имеют фиксированную структуру, включающую идентификатор, управляющие биты, длину данных (DLC), до восьми байт полезной нагрузки и контрольную сумму CRC. На уровне микроконтроллера ESP32, работающего в режиме TWAI (Two-Wire Automotive Interface — аппаратная реализация CAN), кадры доступны в виде структурированных объектов с полями для идентификатора, длины данных и массива байтов. Таким образом, ESP32 получает телеметрию уже в разобранном формате, готовом для дальнейшей обработки.

Для получения параметров автомобиля используется стандарт SAE J1979, регламентирующий набор PID-команд (Parameter ID), доступных через диагностический разъем OBD-II. Запросы в CAN-сеть формируются как кадры с идентификатором 0x7DF (широковещательный запрос), где первый байт определяет длину сообщения, второй — номер сервиса (в данном проекте используется сервис 0x01 — текущие параметры автомобиля), а третий — номер PID. Ответы от ЭБУ поступают на идентификаторы 0x7E8-0x7EF и

содержат значение параметра в виде одного или нескольких байтов. Идентификатор ЭБУ определяет, какой блок управления дал ответ. В автомобилях стоят стандартные блоки управления (например, силовой), но могут стоять и другие блоки – это устанавливает производитель ЭБУ. Так же производитель определяет какой идентификатор принадлежит конкретному ЭБУ.

Контроллер ESP32 после инициализации CAN-интерфейса выполняет начальный опрос поддерживаемых PID из диапазона 0x01–0x20. Запрос формируется циклически, с интервалом 1500 мс, до получения ответа. Если ни один ЭБУ не отвечает после пяти попыток, устройство автоматически перезагружается, поскольку отсутствие ответа свидетельствует о неисправном подключении или неподдерживаемом протоколе.

После определения поддерживаемых PID ESP32 переходит к циклическому опросу нужных параметров. Каждый PID запрашивается по очереди, а после достижения конца списка система возвращается к началу, обеспечивая непрерывный сбор телеметрии.

В составе системы используется навигационный модуль NEO-6M, который передает GPS-информацию в текстовом формате NMEA-0183 через интерфейс UART. Стандарт NMEA определяет набор строк, каждая из которых содержит координаты, скорость, высоту, время, статус фиксации и другие параметры. Наиболее распространенные типы строк включают GGA, RMC, GLL, GSA, GSV и VTG.

Для проекта используется только строка GLL, содержащая минимально необходимый набор параметров: широту, долготу, направление координат, а также флаг достоверности данных. Такой выбор позволяет снизить нагрузку на UART и процессор, поскольку большая часть перегружающей информации (высота, скорость, качество GPS-фиксации) для цели курсовой работы не требуется.

NEO-6M был предварительно настроен через USB-to-TTL конвертер с использованием программного обеспечения u-center, входящего в экосистему u-blox. В рамках конфигурации модуля была установлена частота обновления 5 Гц, а также отключены все типы строк, кроме GLL. Это обеспечивает высокую частоту обновления координат при минимальном объеме данных.

ESP32 создает отдельную задачу FreeRTOS для приема UART-потока. Пакеты NMEA поступают в виде ASCII-строк, завершающихся символами новой строки. Контроллер выделяет строку GLL, анализирует поля и преобразует координаты из формата DDMM.MMMM в десятичный вид, что удобно для последующей записи в MySQL и отображения на карте.

## **2.5 Проектирование модуля обработки и упаковки данных**

Модуль обработки и упаковки данных является ключевым компонентом клиентской части системы, обеспечивающим преобразование телеметрии, поступающей от GPS-приёмника NEO-6M и данных, получаемых от электронного блока управления (ЭБУ) автомобиля по CAN-шине, в унифицированный транспортный формат. В качестве формата обмена для передачи данных на MQTT-брокер выбран JSON, что позволяет обеспечить гибкость, читаемость, расширяемость структуры сообщений и совместимость с серверной частью, реализующей запись данных в MySQL.

Разрабатываемый модуль реализуется на контроллере ESP32 и выполняет следующие задачи:

1. Предобработка и нормализация данных телеметрии.
2. Формирование JSON-пакетов различных типов (геоданные, PID-данные, служебные сообщения).
3. Контроль корректности данных (валидность GPS, проверка диапазонов значений PID).
4. Управление структурой сообщений и их передачи модему по UART.

## 5. Буферизация данных на случай временной недоступности связи.

Модуль обработки и упаковки данных должен:

- поддерживать формирование трёх основных типов сообщений:
  1. служебные сообщения (инициализация и запуск сессии),
  2. геоданные (geo),
  3. данные PID-ответов (pid);
- использовать единый стиль именования полей;
- обеспечивать временную метку в формате ISO 8601 или в виде Unix-timestamp;
- генерировать сообщения, совместимые со структурой таблиц в MySQL;
- быть устойчивым к сбоям связи и потере отдельных пакетов;
- минимизировать объём JSON-структур для экономии трафика через сотовый модем.

После получения списка поддерживаемых PID и завершения инициализации контроллер отправляет на MQTT-брокер служебный пакет для открытия новой сессии. Он содержит MAC-адрес устройства и временную метку:

```
{  
  "type": "session_start",  
  "mac": "AA:BB:CC:DD:EE:FF",  
  "timestamp": "2025-01-01 12:00:00"  
}
```

Брокер создаёт запись в таблице `data_sessions` и возвращает контроллеру `session_id`. Модуль сохраняет его в энергонезависимую или оперативную память и использует в последующих JSON-структурах.



Геолокация поступает от модуля NEO-6М. Модуль обработки преобразует данные NMEA в структуру, согласованную с таблицей geo\_data.

JSON-сообщение имеет вид:

```
{  
  "type": "geo",  
  "session_id": 1234,  
  "geo_datetime": "2025-01-01 12:00:05",  
  "latitude": 55.755825,  
  "latitude_direction": "N",  
  "longitude": 37.617298,  
  "longitude_direction": "E",  
  "data_is_reliable": true  
}
```

Особенности обработки:

- Модуль проверяет флаг валидности данных NMEA (A/V).
- В случае недостоверности позиции поле data\_is\_reliable устанавливается в false, но данные продолжают передаваться для анализа сервером.
- Дробная часть координат сохраняется в формате decimal(9,6), как в MySQL.

После получения запросов SAE J1979 по CAN 2.0B модуль получает «сырые» значения (например, скорость, обороты двигателя, температура ОЖ). Затем данные нормализуются в соответствии с формулами PID и упаковываются в JSON, согласованный с таблицей PID\_responses.

### JSON-структура:

```
{  
  "type": "pid",  
  "session_id": 1234,  
  "request_command_id": 22,  
  "response_datetime": "2025-01-01 12:00:06",  
  "response_data": 3102  
}
```

Каждому PID-ответу присваивается время получения. Поле `request_command_id` связывает ответ с командой из таблицы `PID_commands`. Значение `response_data` передаётся в виде нормализованного числа, пригодного для сохранения в MySQL.

Таблица 2.1 Обязательные поля в JSON-пакете

Поле	Назначение
type	Тип данных (session_start, geo, pid)
session_id	Привязка к сессии
timestamp	Временная метка

Такой подход позволяет серверной части легко различать форматы сообщений и маршрутизировать их в соответствующие таблицы MySQL.

## 2.6 Проектирование канала передачи данных на основе MQTT

Канал передачи данных между телеметрическим устройством на базе ESP32 и серверной частью системы организуется с использованием технологии обмена сообщениями через MQTT-брокер. Такая архитектура обеспечивает асинхронность, масштабируемость и минимальный сетевой трафик, что является критичным при работе через сотовый модем SIMCom.

MQTT реализует модель взаимодействия «публикующий — брокер — подписчик». Устройство ESP32 выступает в роли публикующего узла, MQTT-брокер (Mosquitto) — в роли центрального узла маршрутизации, а серверное приложение и клиентская программа на ПК — в роли подписчиков.

Для разделения типов передаваемых данных используются четыре логических канала:

```
devices/<mac_address>/session/start
```

```
devices/<mac_address>/geo
```

```
devices/<mac_address>/pid
```

```
devices/<mac_address>/status
```

Назначение топиков:

- session/start — сообщение об открытии новой сессии, содержащее MAC-адрес устройства и время начала работы.
- geo — публикация геоданных в формате JSON (широта, долгота, направление координат, достоверность, время).
- pid — публикация телеметрических значений, полученных от ЭБУ автомобиля (например, RPM, скорость, температура).
- status — служебные сообщения о состоянии устройства, уровне связи, ошибках или диагностике.

Разделение позволяет серверу корректно классифицировать данные при обработке и распределять нагрузку.

Устройство подключено к сети через модем SIMCom и периодически публикует данные в соответствующие топики. Каждый тип данных имеет заранее определённый JSON-формат, соответствующий структурам, используемым в базе данных MySQL.

ESP32 не знает ничего о сервере или базе данных — оно лишь публикует сообщения в MQTT-брокер, что упрощает логику прошивки и делает архитектуру модульной.

На стороне сервера существует отдельный программный компонент — модуль-подписчик.

Он подписан сразу на весь набор топиков:

```
devices/+/session/start
```

```
devices/+/geo
```

```
devices/+/pid
```

```
devices/+/status
```

Символ `+` позволяет автоматически принимать данные от любого контроллера, независимо от его MAC-адреса.

Функции серверного подписчика:

- принимать сообщения от MQTT-брокера,
- анализировать тип данных,
- преобразовывать JSON в структуру, соответствующую таблицам MySQL,
- выполнять запись в базу данных.

Клиентская программа на ПК также является подписчиком MQTT:

- подписывается на телеметрию выбранного контроллера,
- получает геоданные в реальном времени,
- получает значения PID-параметров,
- отображает автомобиль на карте и обновляет показатели.

Пользователь предварительно выбирает MAC-адрес активного контроллера, и приложение подписывается только на его топики, что снижает трафик и повышает отзывчивость интерфейса.

На сервере данные сохраняются следующим образом:

#### 1. Сообщение session/start

Создаёт новую запись в таблице data\_sessions. Сессия считается активной до получения команды об остановке либо до пропадания устройства.

#### 2. Сообщения geo

Каждое сообщение преобразуется в строку таблицы geo\_data с указанием:

- времени,
- широты и долготы,
- направления координат,
- достоверности данных,
- идентификатора текущей сессии.

#### 3. Сообщения pid

Телеметрические данные записываются в PID\_responses, включая:

- идентификатор PID,
- время получения,
- значение раскрытого параметра,
- идентификатор сессии.

### **2.7 Проектирование клиентского приложения для мониторинга и анализа**

Клиентское приложение выполняет роль основного интерфейса оператора, обеспечивая доступ к данным устройств в реальном времени, а также к историческим записям, хранящимся в базе данных. При проектировании интерфейса учитываются следующие ключевые задачи: оперативный мониторинг, визуализация телеметрии, анализ прошедших

поездки (сессий), выбор активного контроллера и интерактивная работа с картографическими данными.

Поскольку система поддерживает множество устройств, клиентскому приложению необходим механизм выбора «активного контроллера». После подключения к серверу пользователю предоставляется список всех зарегистрированных контроллеров. Выбранный контроллер становится источником данных реального времени: приложение автоматически подписывается на соответствующие MQTT-топики. Смена контроллера приводит к отписке от предыдущих топиков и подписке на новые, что предотвращает избыточный трафик и обеспечивает актуальность отображаемой информации.

Клиентское приложение содержит картографический модуль, интегрированный через API выбранного картографического сервиса. На карту выводится текущая геопозиция контроллера.

Одновременно с координатами система отображает сокращённый набор основных телеметрических параметров реального времени, наиболее важных для мониторинга:

- фактическая скорость движения автомобиля;
- обороты двигателя (RPM).

Данные выводятся в компактной панели, обновляются по мере поступления сообщений MQTT и позволяют оператору оценивать ситуацию в режиме онлайн.

Для более глубокого анализа предусмотрен режим отображения дополнительных параметров, предоставляемых сервисом OBD-II. К таким параметрам относятся температурные показатели, расход топлива, положение дроссельной заслонки, нагрузки двигателя и другие диагностические данные.

Помимо данных в реальном времени, пользователь может работать с историческими данными. После выбора контроллера приложение запрашивает у сервера список записанных сессий (по данным таблицы sessions базы данных). Каждая сессия включает время начала, окончания и дополнительные характеристики.

При выборе конкретной сессии клиент получает:

- сохранённый маршрут движения (набор GPS-точек);
- временную шкалу телеметрии;
- диагностические параметры, соответствующие моментам времени.

Для детального анализа телеметрические данные сессии выводятся на графики. Пользователь может изучить изменение параметров во времени, такие как:

- скорость;
- обороты двигателя;
- температура охлаждающей жидкости;
- мгновенный расход топлива;
- другие диагностические параметры, если они были доступны.

Клиентское приложение взаимодействует сразу с двумя источниками данных:

- MQTT-брокером — для приёма данных реального времени;
- сервером REST/WS — для запросов к базе данных (исторические сессии, маршруты, телеметрия).

### **3. КОНТРОЛЬ КАЧЕСТВА И ИНТЕГРАЦИЯ КОМПОНЕНТОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

#### **3.1 Методика тестирования аппаратного и программного комплекса**

Методика тестирования разрабатываемого решения направлена на проверку корректности работы всех компонентов: контроллера сбора данных (аппаратная часть), модема передачи данных, MQTT-брокера, серверной части и клиентского приложения. Тестирование проводится поэтапно — от проверки отдельных модулей до комплексной оценки работы всей системы в условиях, максимально приближённых к реальной эксплуатации автомобиля.

Основными целями тестирования являются:

- подтверждение корректности сбора данных о геолокации и телеметрии автомобиля;
- проверка устойчивости передачи данных модемом через сеть сотового оператора;
- верификация корректной работы MQTT-брокера и серверной части, включая обработку, распаковку и запись данных в базу MySQL;
- оценка своевременности доставки данных клиентскому приложению в режиме реального времени;
- проверка корректности визуализации маршрута и телеметрии в клиентском интерфейсе;
- выявление возможных ошибок, задержек, некорректных форматов данных или перегрузок каналов связи.

#### **Тестирование аппаратного контроллера**

Проверяемые функции:

- получение и парсинг геоданных (широта, долгота, валидность);



- взаимодействие с ЭБУ автомобиля и получение значений PID (скорость, обороты, температура, нагрузка двигателя и др.);
- формирование JSON-пакетов для отправки;
- корректность запуска и остановки сессии.

Методика:

1. Контроллер подключается к диагностическому разъёму OBD-II.
2. Производится запрос списка PID, проверяется корректность ответа.
3. Генерируются тестовые PID-запросы и сравниваются полученные данные с эталонными значениями автомобиля.
4. В фиксированные интервалы времени проверяется стабильность работы GPS-модуля.
5. Измеряется частота обновления данных и отсутствие пропусков сообщений.

### **Тестирование канала связи через модем**

Проверяемые параметры:

- стабильность подключения к сети оператора;
- корректность отправки JSON-пакетов на MQTT-брокер;
- работа в условиях низкого уровня сигнала;
- обработка обрывов связи и переподключений.

Методика:

1. Выполняется публикация тестовых сообщений в соответствующие топики:
  - devices/<mac>/session/start
  - devices/<mac>/geo

- devices/<mac>/pid
  - devices/<mac>/status
2. На стороне сервера ведётся логирование входящих сообщений.
  3. Проверяются ситуации:
    - кратковременное отсутствие сигнала,
    - задержки в сети,
    - публикация крупных пакетов (PID + GPS одновременно).
  4. Оценивается корректность повторной отправки сообщений после восстановления связи.

### **Тестирование MQTT-брокера и серверной части**

Проверяемые функции:

- подключение клиентов (модемов) и удержание соединений;
- публикация данных и передача подписчикам;
- обработка JSON-пакетов сервером;
- запись данных в MySQL;
- отсутствие потерь сообщений.

Методика:

1. Запуск MQTT-брокера на VPS или локальной машине.
2. Подключение тестового клиента и публикация заранее подготовленных JSON-сообщений.
3. Проверка:
  - корректности парсинга;
  - валидности данных;

- соответствия формата структуре БД;
  - сохранения данных в geo\_data и PID\_responses.
4. Проведение нагрузочного теста: отправка сотен сообщений подряд.
  5. Проверка логов на предмет ошибок: неверный топик, повреждённый JSON, нарушение формата.

### **Тестирование клиентского приложения**

Проверяемые функции:

- выбор активного контроллера;
- отображение данных в реальном времени;
- показ положения автомобиля на карте;
- отображение скорости, оборотов и других параметров;
- переход в режим просмотра сессий;
- отображение маршрута на карте;
  - построение графиков телеметрии по времени.

Методика:

1. Клиент подписывается на топики всех активных устройств.
2. Проверяется корректность:
  - подключения и получения обновлений без задержек,
  - отображения GPS-точек,
  - обновления скорости/оборотов в реальном времени.
3. Выполняется демонстрационный заезд, в процессе которого клиент должен:
  - показывать перемещение автомобиля,

- своевременно обновлять данные.
4. После завершения используется режим просмотра сессии:
- загружается маршрут по данным из БД,
  - визуализируются графики (скорость, обороты и т.д.)

### **Комплексное тестирование**

Проводится сквозная проверка всей системы, включающая:

1. Запуск контроллера → начало сессии.
2. Сбор данных + передача через модем → MQTT-брокер.
3. Обработка сервером → запись в БД.
4. Клиентское приложение:
  - видит устройство онлайн,
  - отображает телеметрию,
  - показывает маршрут.

### **3.2 Интеграционное тестирование модулей системы**

Интеграционное тестирование направлено на проверку корректности совместной работы аппаратных и программных компонентов системы после их предварительной модульной отладки. На данном этапе анализируется не внутренняя логика отдельных модулей, а взаимодействие между ними, целостность передаваемых данных, устойчивость соединений и корректность обработки передаваемой информации на всех стадиях прохождения от контроллера до клиентского приложения.

Тестирование начинается с анализа связи между контроллером на базе ESP32 и модемом, а также их взаимодействия с MQTT-брокером. Особое внимание уделяется формированию контроллером корректных JSON-пакетов и их передаче через сеть оператора связи. Проверяется, что модем стабильно

публикует сообщения в заданные топики, а брокер корректно принимает эти данные и не теряет сообщения при разрывах соединения, слабом сигнале или повторных публикациях.

Следующим этапом является интеграция брокера и серверного приложения. Здесь анализируется механизм подписки сервера на соответствующие топики, корректность получения сообщений и отсутствие искажений в передаваемой структуре данных. Проверяется, что сервер способен обрабатывать JSON-пакеты, выявлять ошибки формата, выполнять валидацию данных и формировать внутренние объекты для дальнейшей работы. Особое внимание уделяется поведению системы при некорректных входных данных, устаревших или отсутствующих полях, а также при высокой частоте публикаций.

После проверки взаимодействия брокера и сервера тестирование переходит к анализу связи между сервером и базой данных MySQL. Оценивается корректность формирования SQL-запросов, соответствие структуры данных схемам таблиц, целостность внешних ключей и отсутствие дублирования записей. Тесты показывают, насколько быстро база данных принимает записи телеметрии и геолокации, правильно ли ведутся сессии и корректно ли обрабатываются ситуации временной недоступности СУБД.

Интеграционное тестирование также включает этап анализа передачи данных от базы данных к клиентскому приложению. Проверяется, способен ли сервер корректно агрегировать и подготавливать данные для отображения на стороне клиента, включая маршруты перемещения автомобиля, историю телеметрии, список сессий и отображение данных в реальном времени. Оценивается скорость получения данных клиентом, корректность их визуализации и отсутствие искажений при передаче.

Отдельное внимание уделяется проверкам системы в условиях отказов. В ходе тестирования имитируются обрывы связи модема, недоступность брокера,

перегрузка канала или база данных, а также отправка повреждённых или неполных JSON-пакетов. Анализируется способность системы к восстановлению соединения, возобновлению сессии и сохранению данных без потерь. Важной частью является и проверка поведения системы при увеличении нагрузки, когда телеметрия поступает с высокой частотой или одновременно от нескольких контроллеров. Это позволяет оценить устойчивость брокера, производительность сервера и отклик базы данных при реальных эксплуатационных сценариях.

Интеграционное тестирование считается успешным, если данные на всём пути — от контроллера до клиента — передаются корректно, структура сообщений сохраняется, задержка обновления минимальна, сервер правильно обрабатывает все сообщения, а база данных надёжно фиксирует телеметрию и геолокацию без искажений. Кроме того, система должна успешно восстанавливаться после сбоев и обеспечивать стабильную работу в условиях высокой нагрузки.

### **3.3 Тестирование клиентского приложения**

Тестирование клиентского приложения проводилось с целью проверки корректности его работы при взаимодействии с серверной частью, оценки удобства использования, стабильности и производительности. В процессе тестирования рассматривались функциональные и нефункциональные аспекты, включая корректность отображения данных, работу интерфейса, обработку ошибок и поведение приложения в нестандартных условиях.

В рамках функционального тестирования проверялась реализация основных пользовательских сценариев:

- Просмотр и обновление данных — корректность запроса информации на сервер, отображение полученных данных, их обновление в соответствии с действиями пользователя.

- Навигация по интерфейсу — проверка переходов между экранами, корректности работы кнопок и элементов управления.
- Обработка ошибок — проверка поведения приложения при отсутствии соединения, некорректных данных, длительной задержке ответа сервера.

Все основные сценарии были последовательно выполнены, и приложение корректно реагировало на действия пользователя.

Проводилась оценка интерфейса (UI/UX) следующих аспектов:

- удобство взаимодействия пользователя с интерфейсом;
- корректность отображения элементов на разных размерах экрана;
- читаемость текста и доступность основных функций;
- отсутствие визуальных артефактов, некорректно отображающихся компонентов и пересекающихся элементов.

Тестирование производительности, а именно:

- время запуска приложения;
- скорость отклика интерфейса;
- время получения и отображения данных после сетевых запросов;
- стабильность работы при длительной эксплуатации.

Замеры показали, что клиентское приложение функционирует устойчиво, время отклика находится в пределах нормы, задержки при взаимодействии с сервером минимальны.

## ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана программная система, включающая серверное и клиентское приложения, обеспечивающая взаимодействие с базой данных и предоставляющая пользователю удобные средства для работы с информацией. Итоговый проект полностью соответствует требованиям, определённым в техническом задании, и демонстрирует корректную и устойчивую работу всех ключевых компонентов.

Созданное клиентское приложение обладает интуитивно понятным интерфейсом, что делает его доступным для пользователей с разным уровнем подготовки. Серверная часть обеспечивает надёжную обработку запросов, стабильное хранение данных и безопасное взаимодействие между компонентами системы.

Разработанная архитектура позволяет легко расширять и модифицировать функциональность приложения. При дальнейшем развитии проекта могут быть реализованы следующие возможности:

- Добавление новых модулей для расширения функционала;
- Поддержка дополнительных сущностей и расширение структуры базы данных;
- Реализация расширенной системы ролей и прав доступа;
- Введение более гибких инструментов аналитики и визуализации данных;
- Оптимизация производительности с использованием продвинутых механизмов кеширования и обработки данных;
- Разработка мобильной версии приложения или адаптация для работы на различных платформах;
- Улучшение системы логирования и мониторинга для повышения отказоустойчивости.



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ford, A. Applied MQTT — Practical Guide for IoT Messaging / A. Ford. — O'Reilly Media, 2020. — 210 p. — ISBN 978-1-49207-765-9.
2. Trottier, C. Building Distributed Applications with .NET / C. Trottier. — Addison-Wesley, 2018. — 352 p. — ISBN 978-0-321-48682-5.
3. Антонов А. И., Галкин В. А., Аксенов А. Н. Сетевые технологии в автоматизированных системах обработки информации и управления : учебное пособие / Антонов А. И., Галкин В. А., Аксенов А. Н. - Москва : МГТУ им. Н. Э. Баумана, 2020. - 148 с. - ISBN 978-5-7038-5221-7.
4. Ачилов, Р. Н. Построение защищенных корпоративных сетей : учебное пособие / Р. Н. Ачилов. — Москва : ДМК Пресс, 2013. — 250 с. — ISBN 978-5-94074-884-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/66472>
5. Басараб М. А., Колесников А. В., Коннова Н. С. Моделирование компьютерных сетей : учебно-методическое пособие / Басараб М. А., Колесников А. В., Коннова Н. С. ; МГТУ им. Н. Э. Баумана (национальный исследовательский ун-т). - М. : Изд-во МГТУ им. Н. Э. Баумана, 2021. - 82 с. : ил. - Библиогр. в конце кн. - ISBN 978-5-7038-5729-8.
6. Бондарев, В. В. Программирование клиент-серверных приложений : учебное пособие / В. В. Бондарев. — Санкт-Петербург : Лань, 2020. — 304 с. — ISBN 978-5-8114-4788-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/175234>
7. Гольдштейн, Б. С. Защита информации в компьютерных сетях : учеб. пособие / Б. С. Гольдштейн, С. В. Палутенко. — Санкт-Петербург : Лань, 2021. — 368 с. — ISBN 978-5-8114-4561-6.
8. Ибе, О. Компьютерные сети и службы удаленного доступа : справочник / О. Ибе. — Москва : ДМК Пресс, 2007. — 336 с. — ISBN 5-94074-080-4. —

Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/1169>

9. Красильщиков, М. Н. Введение в современные сетевые технологии : учебное пособие / М. Н. Красильщиков. — Москва : БИНОМ, 2019. — 224 с. — ISBN 978-5-9963-3630-4.
10. Курочкин, А. В. Сетевое программирование в .NET : учебное пособие / А. В. Курочкин. — Москва : ДМК Пресс, 2021. — 320 с. — ISBN 978-5-97060-961-2.
11. Нисси, М. MQTT — The Messaging Protocol for the IoT : учебное пособие / M. Nissi. — Хельсинки : Aalto University Press, 2020. — 112 p.
12. Рихтер, Д. CLR via C#. Программирование на платформе Microsoft .NET Framework / Д. Рихтер. — 4-е изд. — Москва : Вильямс, 2019. — 896 с. — ISBN 978-5-8459-1970-9.
13. Сергеев, А. Н. Основы локальных компьютерных сетей : учебное пособие для вузов / А. Н. Сергеев. — 4-е изд., стер. — Санкт-Петербург : Лань, 2022. — 184 с. — ISBN 978-5-507-44766-4. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/242867>
14. Сетевые технологии и Интернет Учебное пособие / Семенов А.А. - 2017. - URL: <http://www.iprbookshop.ru/66840.html>.
15. Смит, П. MQTT Essentials — A Lightweight IoT Protocol / P. Smith. — Birmingham : Packt Publishing, 2019. — 176 p. — ISBN 978-1-78913-224-5.
16. Стивенс, Р. UNIX: разработка сетевых приложений / Р. Стивенс. — Москва : Вильямс, 2018. — 944 с. — ISBN 978-5-8459-1974-7.