# **КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА**

С развитием сетевых технологий клиент-серверная архитектура становится неотъемлемой частью информационных систем, обеспечивая эффективное взаимодействие между клиентскими и серверными компонентами.

Эта архитектура позволяет создавать гибкие, масштабируемые и безопасные приложения, удовлетворяющие потребности современных пользователей. Применение клиент-серверных технологий способствует оптимизации обработки данных, повышению производительности и улучшению пользовательского опыта.

Стремительное развитие интернета вещей, облачных вычислений и мобильных устройств диктует необходимость постоянного совершенствования клиент-серверных технологий для обеспечения надёжной и масштабируемой работы приложений в современном информационном ландшафте.

В архитектуре «Клиент – Сервер» несколько компьютеров-клиентов, (удалённые системы), посылают запросы и получают некоторые услуги от централизованной служебной машины – сервера, которая так же может называться хост-системой. Клиентская машина предоставляет пользователю так называемый «дружественный интерфейс», (*user-friendly interface*), чтобы облегчить его взаимодействие с сервером [14].

Клиент-серверная архитектура включает в себя следующие компоненты:

– клиенты: это устройства или приложения, которые запрашивают информацию или услуги у сервера;

– серверы: это компьютеры, которые предоставляют запрашиваемую информацию или услуги клиентам. Серверы могут выполнять различные функции, такие как хранение данных, обработка запросов, вычисления и т. д.;

– протоколы обмена данными: это правила или наборы инструкций, которые определяют, как клиенты и серверы обмениваются информацией. Некоторые из наиболее распространенных протоколов: *HTTP/HTTPS* (гипертекстовые протоколы), стек протоколов *TCP/IP* (набор правил, описывающих, как компьютеры соединяются и передают информацию друг другу), протоколы отправки и получения почты (*SMTP, POP3, IMAP*);

– базы данных: это хранилища информации, которые используются на серверной стороне для хранения и управления данными. Базы данных позволяют серверу эффективно хранить, организовывать и извлекать информацию по запросу клиента;

– сеть: это инфраструктура, которая обеспечивает связь между клиентами и серверами. Упрощенно говоря, сети могут быть локальными (*LAN*) и глобальными (*WAN*). Сеть обеспечивает передачу данных между клиентами и серверами по протоколам обмена данными;

– система безопасности: это компонент, который обеспечивает защиту данных, передаваемых между клиентами и серверами. Этот компонент может включать в себя шифрование данных, аутентификацию и авторизацию клиентов, защиту от несанкционированного доступа и другие меры безопасности;

– хранение и обработка данных: это компоненты, связанные с хранением данных на серверах и их обработкой. Это может включать в себя серверные операционные системы, системы управления базами данных (СУБД), серверы приложений и другие компоненты, необходимые для эффективной работы клиент-серверной архитектуры.

Клиент-серверная архитектура широко применяется во многих областях, включая веб-разработку, область работы с базами данных, область облачных вычислений. Клиент-серверная архитектура также применяется в системах Интернета вещей, (*IoT*), где устройства клиентов обмениваются данными с сервером для управления и мониторинга [15].

Преимуществами клиент-серверной архитектуры является следующее:

– масштабируемость, клиент-серверная архитектура позволяет распределить нагрузку на сервера и может масштабироваться по мере необходимости. Благодаря этому можно значительно улучшить производительность системы и обрабатывать большое количество запросов от клиентов;

– централизованное управление, то есть сервер является центральным узлом, который контролирует всю систему, обеспечивает безопасность и управление доступом к данным. Это позволяет легко обновлять и модифицировать систему;

– надежность, при использовании клиент-серверной архитектуры, отказ одного компонента системы не влияет на работу других компонентов. Это значительно повышает надежность всей системы;

– централизованное управление сервером обеспечивает возможность контроля доступа и защиты данных, что делает клиент-серверную архитектуру более безопасной по сравнению с другими *IT*-архитектурами.

Недостатки клиент-серверной архитектуры:

– зависимость от сервера, клиент не может работать без сервера. Если сервер недоступен, все клиенты будут неработоспособны или испытывать проблемы с функциональностью;

– затраты на инфраструктуру, клиент-серверная архитектура требует наличия серверного и сетевого оборудования и поддержки, что может потребовать затрат на инфраструктуру и обслуживание;

– зависимость от сети, клиент-серверная архитектура требует постоянного подключения к сети. Если сеть недоступна, это может существенно ограничить возможности работы системы;

– ограниченность: при использовании клиент-серверной архитектуры возникают ограничения на количество одновременно подключенных клиентов и на пропускную способность сети.

Это может привести к ограничениям в расширении системы и обработке большого количества запросов.

Возможность использования архитектуры «Клиент – Сервер» для приема-передачи измерительной информации будет отражена в следующем: клиенты, такие как датчики или измерительные устройства, могут отправлять данные на сервер посредством сетевого подключения. В свою очередь серверы, как центральные узлы, могут принимать данные от клиентов и обеспечивать их хранение, обработку и анализ. Серверная сторона может включать в себя базы данных для хранения полученных измерений и системы обработки данных для анализа и визуализации результатов.

Клиентские устройства могут непрерывно собирать данные о погоде, такие как температура, влажность, скорость ветра, и отправлять их на серверы. Серверы могут анализировать и обрабатывать эти данные в реальном времени, предоставляя актуальную информацию о погоде на интерфейсе пользователя. Возможность использования архитектуры «Клиент – Сервер» для передачи-приема статистических данных, полученных из архива по результатам измерений, будет заключаться в том, что клиенты могут запрашивать статистические данные из архива на сервере, например, среднее значение температуры за определенный период времени или суммарные осадки за месяц.

Серверы могут обеспечивать доступ к архивным статистическим данным, хранящимся в базах данных, и передавать запрошенную информацию клиентам через сетевое соединение.

Безопасность передачи данных может быть обеспечена с использованием шифрования и аутентификации на уровне протоколов, таких как *HTTPS*, чтобы предотвратить несанкционированный доступ к статистическим данным.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 4.1.1 – Схема архитектуры «Клиент-сервер»

Работа архитектуры клиент-сервер, согласно представленной выше схеме, построена следующим образом:

– база данных в виде набора файлов находится на жестком диске специально выделенного компьютера (сервера сети), как и СУБД;

– существует локальная сеть, состоящая из клиентских компьютеров, на каждом из которых установлено клиентское приложение для работы с БД;

– на каждом из клиентских компьютеров пользователи имеют возможность запустить приложение. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к СУБД, расположенной на сервере, на выборку/обновление информации. Для общения используется специальный язык запросов *SQL*, т.е. по сети от клиента к серверу передается лишь текст запроса;

– СУБД инкапсулирует внутри себя все сведения о физической структуре БД, расположенной на сервере;

– СУБД инициирует обращения к данным, находящимся на сервере, в результате которых на сервере осуществляется вся обработка данных и лишь результат выполнения запроса копируется на клиентский компьютер. Таким образом СУБД возвращает результат в приложение;

– приложение, используя пользовательский интерфейс, отображает результат выполнения запросов[16].

Функции приложения-клиента:

– посылка запросов серверу;

– интерпретация результатов запросов, полученных от сервера;

– представление результатов пользователю в некоторой форме (интерфейс пользователя).

Функции серверной части:

– прием запросов от приложений-клиентов;

– интерпретация запросов;

– оптимизация и выполнение запросов к БД;

– отправка результатов приложению-клиенту;

– обеспечение системы безопасности и разграничение доступа;

– управление целостностью БД;

– реализация стабильности многопользовательского режима работы.

В заключении стоит сказать, что разделение задач между клиентами и сервером позволяет более эффективно использовать ресурсы информационных систем. Сервера, как правило, более мощные и надёжные устройства, способные обрабатывать сравнительно большие объемы данных, в то время как клиенты могут быть менее мощными устройствами.

В целом, клиент-серверная архитектура обеспечивает гибкость, масштабируемость и надежность сетевых приложений, позволяя клиентам получать доступ к ресурсам и услугам серверов.

## **Описание задачи обработки данных**

В контексте клиент-серверного приложения для модуля метеоконтроля с беспроводным интерфейсом передачи данных, задача обработки данных заключается в следующем:

1 Получение данных о погодных условиях: приложение должно принимать запросы от клиентов через веб-интерфейс или *API*. Эти запросы могут содержать параметры, такие как местоположение или период времени, для которых необходимы данные о погоде.

2 Извлечение данных из источника: после получения запроса, приложение должно извлечь соответствующие данные о погоде из базы данных или другого источника. Эти данные могут включать в себя информацию о температуре, влажности, давлении, скорости ветра и других метеорологических параметрах.

3 Обработка и форматирование данных: полученные данные необходимо обработать и представить в удобном для клиента формате. Это может включать в себя преобразование единиц измерения, округление значений, фильтрацию выбросов и другие операции для обеспечения точности и читаемости данных.

4 Передача данных клиенту: окончательно обработанные данные должны быть отправлены клиенту через веб-интерфейс или *API*. Это позволит пользователям получить актуальную информацию о погоде в выбранном ими местоположении или времени.

5 Обеспечение безопасности и конфиденциальности: важным аспектом обработки данных является обеспечение их безопасности и конфиденциальности. Приложение должно учитывать требования к защите данных и обеспечивать безопасную передачу информации между клиентами и сервером.

6 Обеспечение отказоустойчивости и масштабируемости: для эффективной работы приложения необходимо обеспечить его отказоустойчивость и способность масштабироваться для обработки больших объемов данных и одновременных запросов от пользователей. Это может потребовать использования распределенных систем и механизмов резервного копирования данных.

## **Разработка блок-схемы алгоритма**

Блок-схема должна отражать весь процесс обработки данных от отправки данных модулем метеоконтроля на сервер до их отображения на клиентском приложении.

Для разработки блок-схемы алгоритма обработки данных, согласно стандартам, нужно разделить процесс на основные этапы, которые будут включать взаимодействие клиентской части, серверной части, и устройства.

Разрабатываемый алгоритм предполагает собой следующее:

– модуль портативной метеостанции, благодаря находящемся на нём программном обеспечении, посредством *TCP* соединения посылает данные в универсальном формате на персональный компьютер, на котором развёрнута серверная часть приложения;

– серверная часть приложения на персональном компьютере принимает данные, используя соответствующий сетевой протокол, и проверяет целостность и правильность полученной информации, что включает в себя проверку формата данных и контрольных сумм;

– после успешной проверки целостности, сервер распаковывает данные и производит их первичную обработку;

– обработанные данные сохраняются в базе данных, развернутой на сервере. Для хранения данных может использоваться реляционная база данных (например, *SQLAlchemy*) в зависимости от требований к структуре и объёму данных;

– после сохранения, данные подвергаются более детальному анализу, который включает вычисление статистических показателей, построение зависимостей, при необходимости могут быть использованы алгоритмы машинного обучения для предсказания будущих значений или обнаружения аномалий;

– результаты анализа и обработанные данные становятся доступными для клиентских приложений через веб-приложение. Клиентское приложение могут быть реализованы в виде веб-интерфейса, мобильного приложения или настольного программного обеспечения;

– клиентское приложение визуализируют данные для конечных пользователей. Визуализация может включать графики, таблицы, карты и другие способы представления информации, что позволяет пользователю легко интерпретировать и использовать данные;

– алгоритм предусматривает циклический сбор и обработку данных, что позволяет получать и обрабатывать новую информацию в реальном времени;

Таким образом, разрабатываемый алгоритм обеспечивает эффективный и надежный сбор, передачу, обработку, хранение, анализ и визуализацию данных с модуля метеоконтроля, что позволяет получать актуальную и точную информацию о погодных условиях в режиме реального времени.

Блок схема алгоритма обработки данных приведена на рисунке 7.2.1

Изображение выглядит как текст, снимок экрана, Шрифт, графический дизайн

Автоматически созданное описание

Рисунок 7.2.1 – Блок-схема алгоритма обработки данных

## **Реализация алгоритма**

Алгоритм будет реализован на языке программирования *Python,* так как это высокоуровневый язык программирования общего назначения, который направлен на повышение производительности разработчика, а также на обеспечение переносимости написанных на нём программ, что отвечает поставленной задаче.

Программный код серверной части, а также программный код клиентского приложения приведён в приложениях А и Б соответственно.

Подтверждение правильной работы серверной части приведена на рисунке 7.3.1

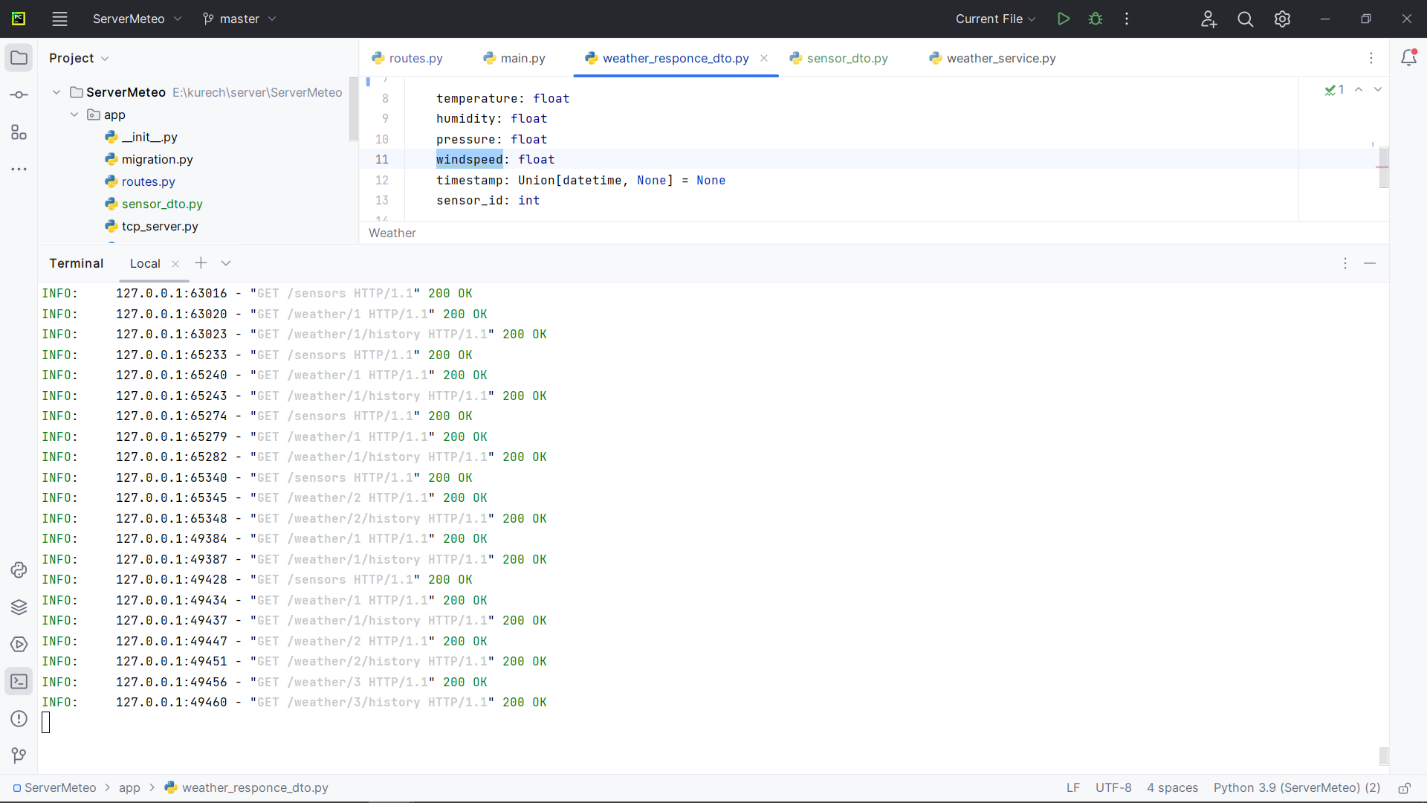


Рисунок 7.3.1 – Подтверждение правильной работы серверной части

Следующий этап – разработка клиентского приложения, которое будет взаимодействовать с сервером, используя *POST*-запросы и *GET*-запросы, в его разработке будет использовать *Tkinter* – графическая библиотека для языка программирования *Python.*

Клиентское приложение должно быть наглядным и интуитивно понятным, для упрощения работы с ним пользователю.

Изображение выглядит как текст, снимок экрана, диаграмма, График

Автоматически созданное описание

Рисунок 7.3.2 – Внешний вид клиентского приложения

В разрабатываемом клиентском приложении предусмотрена возможность просмотра истории изменения параметров окружающей среды, поступающих с модуля метеоконтроля.

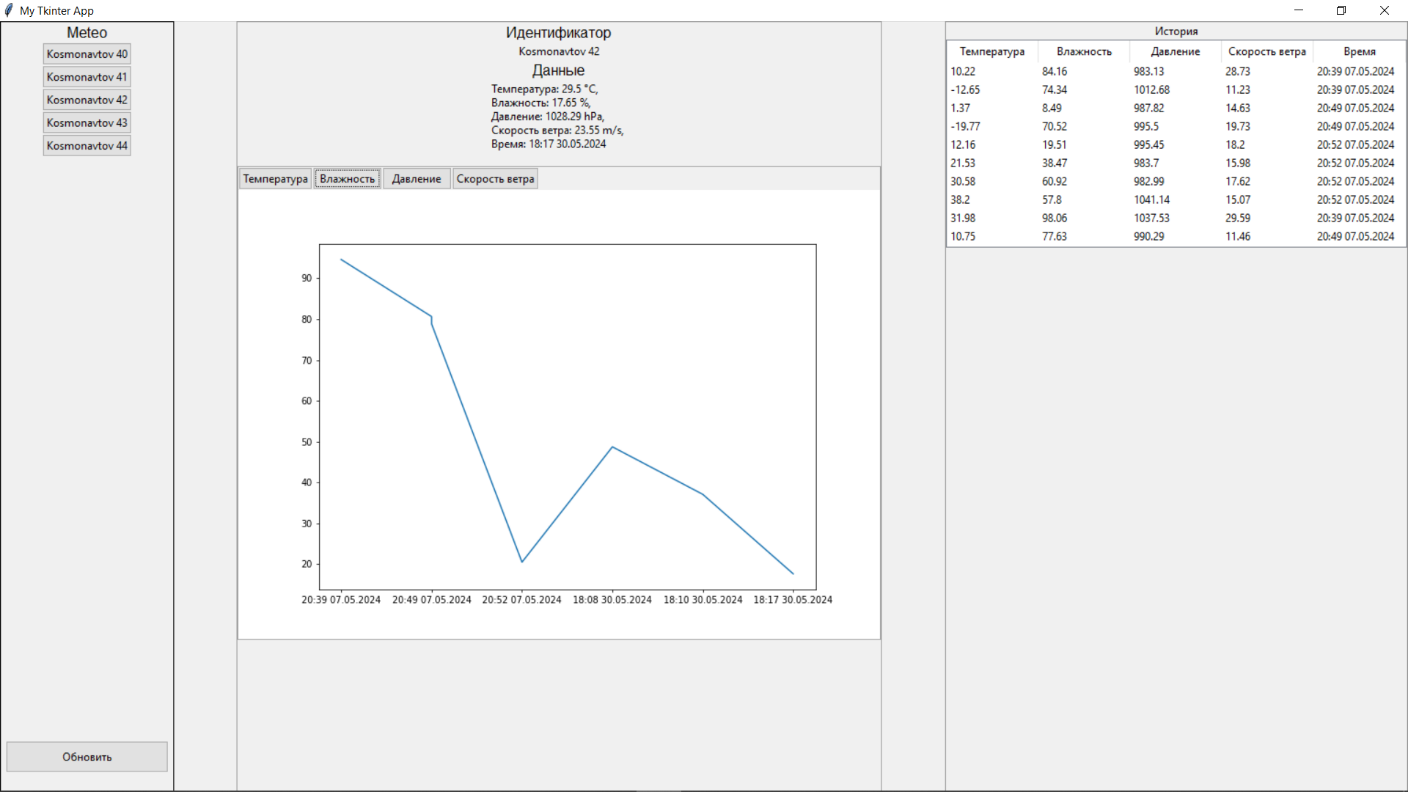


Рисунок 7.3.3 – Результат обработки запроса параметра влажности

Метеоданные представлены как в табличном виде, так и в виде математических графиков, которые показывают динамику их изменения в течение времени.

У пользователя есть возможность выбрать конкретный модуль метеоконтроля, которому соответствует определённый идентификатор, чтобы получать метеоданные с него, а также предусмотрена возможность отслеживания изменений каждого параметра.

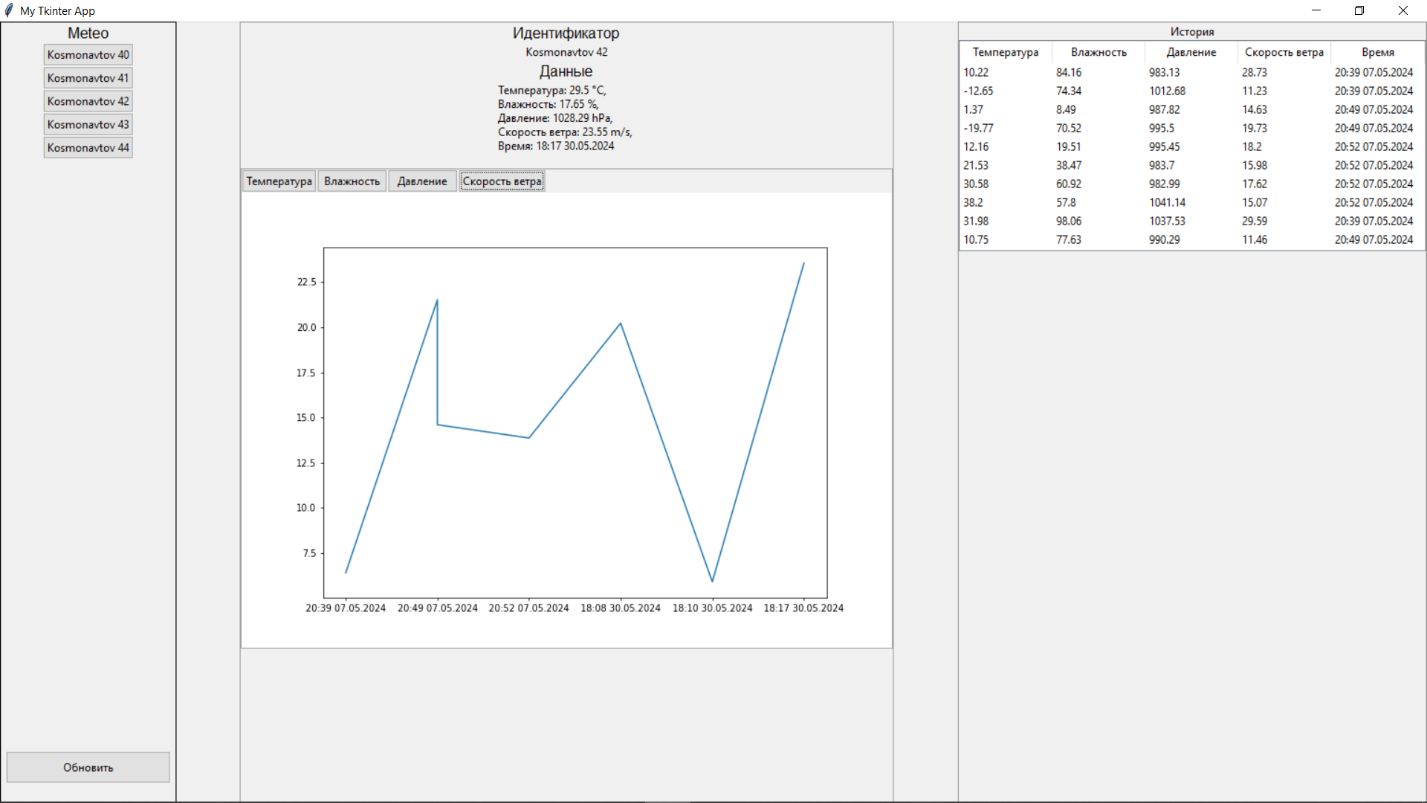


Рисунок 7.3.4. – Результат обработки запроса параметра скорости ветра

Таким образом, был разработан алгоритм обработки данных, который позволяет пользователю эффективно, надежно и удобно получать доступ к текущим метеоданным, к истории изменения параметров окружающей среды, получает также представление о метеоданных в виде математических графиков в режиме реального времени, что сильно упрощает работу с модулем метеоконтроля.

Проведенные тестирования подтвердили правильность работы алгоритма. Сервер корректно обрабатывает запросы от клиента и возвращает ожидаемые результаты.