КОМИТЕТ ПО ОБРАЗОВАНИЮ ПРАВИТЕЛЬСТВА САНКТ-ПЕТЕРБУРГА

**САНКТ- ПЕТЕРБУРГСКОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ   
«КОЛЛЕДЖ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»**

Специальность 09.02.03 «Программирование в компьютерных системах»

**УТВЕРЖДАЮ**

**Зам. директора по УМР**

**\_\_\_\_\_\_\_\_\_\_\_\_О.Н. Венедиктова**

**«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2018 г.**

**ДИПЛОМНЫЙ ПРОЕКТ**

***Пояснительная записка***

**Тема: Система управления базами данных для микроконтроллеров**

Группа 443

Рецензент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нормоконтроль \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.Н. Алексеева

Консультант по экономической части \_\_\_\_\_\_\_\_\_\_\_\_\_ Т.В. Львова

Руководитель дипломного проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Фомин

Выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Бельцов

Санкт-Петербург 2018

# ОГЛАВЛЕНИЕ

Оглавление

[ОГЛАВЛЕНИЕ 2](#_Toc517055606)

[ВВЕДЕНИЕ 6](#_Toc517055607)

[1. АНАЛИТИЧЕСКАЯ ЧАСТЬ 9](#_Toc517055608)

[1.1 Исследование предметной области 9](#_Toc517055609)

[1.2 Обзор и анализ аналогичных решений 10](#_Toc517055610)

[1.3 Анализ средств разработки 16](#_Toc517055611)

[1.3.1 Обзор методов решения 16](#_Toc517055612)

[1.3.1.1 Языки программирования для разработки СУБД 16](#_Toc517055613)

[1.3.1.2 Среды разработки для создания СУБД 21](#_Toc517055614)

[1.3.1.3 Языки программирования для разработки клиента 24](#_Toc517055615)

[1.3.1.4 Среды разработки для создания клиента 27](#_Toc517055616)

[1.3.1.5 Среда разработки для микроконтроллера 29](#_Toc517055617)

[1.3.2 Обоснования выбора методов 31](#_Toc517055618)

[1.3.2.1 Язык программирования для разработки СУБД 31](#_Toc517055619)

[1.3.2.2 Среда разработки для создания СУБД 32](#_Toc517055620)

[1.3.2.3 Язык программирования для разработки клиента 32](#_Toc517055621)

[1.3.2.4 Среда разработки для создания клиента 33](#_Toc517055622)

[1.3.2.5 Микроконтроллер и среда разработки для него 33](#_Toc517055623)

[2. ПРОЕКТНАЯ ЧАСТЬ 34](#_Toc517055624)

[2.1 Техническое задание 34](#_Toc517055625)

[2.1.1 Постановка задачи 34](#_Toc517055626)

[2.1.2 Назначение программного продукта 35](#_Toc517055627)

[2.1.3 Основание для разработки 35](#_Toc517055628)

[2.1.4 Требования к программе 35](#_Toc517055629)

[2.1.4.1 Требования к функциональным характеристикам 36](#_Toc517055630)

[2.1.4.2 Требования к аппаратным и программным средствам 36](#_Toc517055631)

[2.1.5 Требования к надежности 38](#_Toc517055632)

[2.1.5.1 Требования к надежной функциональности системы 38](#_Toc517055633)

[2.1.5.2 Отказ из-за некорректных действий пользователей системы 38](#_Toc517055634)

[2.1.5.3 Требования к пользователям системы 39](#_Toc517055635)

[2.1.6 Защита и сохранность данных 39](#_Toc517055636)

[2.2 Описание функционально-логической структуры программы 39](#_Toc517055637)

[2.2.1 Диаграмма потоков данных (DFD) 40](#_Toc517055638)

[2.2.2 Функциональная схема программы 41](#_Toc517055639)

[2.2.3 Функциональные диаграммы работы программы (IDEF0) 41](#_Toc517055640)

[2.2.4 Диаграмма вариантов использования (UML) 42](#_Toc517055641)

[2.3 Описание физико-математической модели 43](#_Toc517055642)

[2.3.1 Хэширование (SHA-256) 43](#_Toc517055643)

[2.4 Логическая модель данных 47](#_Toc517055644)

[2.4.1 Структуры и классы на сервере 47](#_Toc517055645)

[2.4.2 Структуры и классы на визуальном клиенте 53](#_Toc517055646)

[2.4.3 Структура базы данных СУБД 63](#_Toc517055647)

[2.5 Разработка аппаратной части 65](#_Toc517055648)

[2.5.1 Алгоритм работы устройства 69](#_Toc517055649)

[2.5.2 Сборка устройства 69](#_Toc517055650)

[2.5.3 Настройка устройства 70](#_Toc517055651)

[2.6 Алгоритм работы программы 74](#_Toc517055652)

[2.6.1 Общий алгоритм работы программы 74](#_Toc517055653)

[2.6.2 Алгоритмы отдельных модулей 80](#_Toc517055654)

[2.6.2.1 Алгоритм инициализации СУБД 81](#_Toc517055655)

[2.6.2.2 Алгоритм работы с клиентом 82](#_Toc517055656)

[2.7 Программная реализация 84](#_Toc517055657)

[2.7.1 Пользовательский интерфейс 84](#_Toc517055658)

[2.7.1.1 СУБД 84](#_Toc517055659)

[2.7.1.2 Telnet подключение 85](#_Toc517055660)

[2.7.1.3 Визуальный клиент 86](#_Toc517055661)

[2.7.2 Тестирование и отладка программы 100](#_Toc517055662)

[2.7.3 Локализация программного обеспечения 102](#_Toc517055663)

[2.8 Инструкция пользователя 104](#_Toc517055664)

[2.8.1 Настройка и запуск СУБД 104](#_Toc517055665)

[2.8.1.1 Настройка конфигураций сервера 105](#_Toc517055666)

[2.8.1.2 Запуск сервера 105](#_Toc517055667)

[2.8.2 Сценарий работы Telnet подключения 106](#_Toc517055668)

[2.8.2.1 Подключение к серверу 107](#_Toc517055669)

[2.8.2.2 Регистрация и вход 107](#_Toc517055670)

[2.8.2.3 Создание базы данных 107](#_Toc517055671)

[2.8.2.4 Добавление и получение данных 108](#_Toc517055672)

[2.8.3 Команды 108](#_Toc517055673)

[2.8.3.1 Справочные команды 108](#_Toc517055674)

[2.8.3.2 Команды работы с пользователями 109](#_Toc517055675)

[2.8.3.3 Команды работы с базами данных 109](#_Toc517055676)

[2.8.3.4 Команды работы с данными 109](#_Toc517055677)

[2.8.3.5 Команды прослушивания 110](#_Toc517055678)

[2.8.3.6 Команды работы с языками 110](#_Toc517055679)

[2.8.3.7 Команды для разграничения прав 110](#_Toc517055680)

[2.8.3.8 Информационные команды 111](#_Toc517055681)

[2.8.3.9 Команды для управления сервером 111](#_Toc517055682)

[2.8.3.10 Другие команды 112](#_Toc517055683)

[2.8.4 Сценарий работы визуального клиента 112](#_Toc517055684)

[2.8.4.1 Открытие справки 113](#_Toc517055685)

[2.8.4.2 Добавление подключения 114](#_Toc517055686)

[2.8.4.3 Добавление данных в базу данных 115](#_Toc517055687)

[2.9 Анализ результатов работы 116](#_Toc517055688)

[3. ЭКОНОМИЧЕСКАЯ ЧАСТЬ 118](#_Toc517055689)

[3.1 Описание продукта 118](#_Toc517055690)

[3.2 Описание и расчет затрат на выполнение проекта 118](#_Toc517055691)

[3.3 Расчет расходов на заработную плату 120](#_Toc517055692)

[3.4 Расчет затрат на материалы 120](#_Toc517055693)

[3.5 Расчет амортизации 121](#_Toc517055694)

[3.6 Расчет электроэнергии 121](#_Toc517055695)

[3.7 Расчет сметы затрат 121](#_Toc517055696)

[3.8 Расчет предполагаемой прибыли 122](#_Toc517055697)

[ЗАКЛЮЧЕНИЕ 124](#_Toc517055698)

[Приложение А1. Исходный текст программы СУБД 126](#_Toc517055699)

[Приложение А2. Исходный текст программы визуального менеджера 148](#_Toc517055700)

[Приложение А3. Исходный текст программы микроконтроллера 167](#_Toc517055701)

[Приложение Б. Рисунки 169](#_Toc517055702)

# ВВЕДЕНИЕ

Еще с древних времен у человека возникла потребность в сохранении данных. Для начала люди использовали рисунки, которыми исписывали стены, затем пришли к какой бы то ни было письменности, которую сохраняли на камнях, спустя долгий промежуток времени уже появились пергамент и бумага. Но факт остается фактом, у людей возникла нужда в создании более долговременных, максимально удобных и систематизированных мест для сохранения данных.

До того, как появились базы данных, какими мы понимаем их сейчас, примером может служить – библиотека. В них хранятся книги систематизировано, отделы с определенными жанрами, сортировка книг по алфавиту, можно спокойно найти нужную информацию для дальнейшего использования. Но книг появляется все больше, не все способны поддерживать актуальность библиотеки, информацию становится найти с каждым разом сложнее, а уж если это научная литература, то и проверка достоверности требуется на основании нескольких источников, что требует немало времени. К тому же, если взяли из нее книгу, то более экземпляров этой книги может и не быть, и требуется ждать, пока вернет читатель эту книгу. Так что, библиотеки сейчас имеют смысл лишь для художественной литературы, ее актуальность не имеет значения.

Еще до возникновения всемирной интернет паутины, разрабатывались первые базы данных, для хранения электронной информации, сначала на перфокартах, потом и на жестких дисках. А уж с появлением интернета, как мы его сейчас знаем, базы данных используются практически везде, для хранения практически любых данных, от простых чисел до сложных мультимедийных данных. Благодаря вычислительным технологиям запись, поиск, сортировка и группировка информации осуществляется очень быстро. Информация доступна откуда угодно, лишь нужен доступ в интернет, что в наш век – редкость быть без интернета. Для управления базами данных используется система управления базами данных (СУБД).

На текущий момент существует огромное количество СУБД, в зависимости от потребностей пользователя, программного обеспечения, скорости работы, модели данных, хранимых данных и т.д.

Все эти СУБД отлично подходят для прикладного программного обеспечения, где скорость управления данными не заметна. А вот на низкоуровневом уровне, для тех же микроконтроллеров, где требуется молниеносная передача значений, для дальнейшей обработки, они не подходят. Для их работы порой требуется установка, дополнительные компоненты, много места и из-за своей громоздкости менее производительны.

Сейчас стремительно развивается такое направление, как «интернет вещей», когда устройства могут взаимодействовать друг с другом независимо от человека для какой-либо задачи. Задачи, которые они выполняют, могут быть разные, но передача данных должна быть стремительная, а подключение не должно представлять сложности из-за того, что затрачиваемые ресурсы должны пойти на их обработку, а обработка тоже занимает время. Время в наше время представляет собой самый ценный ресурс, ведь его у нас, по сравнению со вселенскими масштабами, ничтожно, и чем меньше времени отнимает та или иная задача, тем его больше можно потратить на другую, более значимую задачу.

Цель проекта: создать СУБД специально для максимально быстрого управления данными типа «Ключ – значение», запуском на большом количестве устройств, иметь способы, обеспечивающие безопасность баз данных, включать возможности изменения кода под свои потребности, быть как можно более простой в подключении и в тоже время быть незаменимым инструментом.

Данный диплом подтверждает компетенции профессиональных модулей:

1. ПМ 01. Разработка программных модулей программного обеспечения для компьютерных систем:

– ПК 1.1. Выполнять разработку спецификаций отдельных компонент.

– ПК 1.2. Осуществлять разработку кода программного продукта на основе готовых спецификаций на уровне модуля.

– ПК 1.3. Выполнять отладку программных модулей с использованием специализированных программных средств.

– ПК 1.4. Выполнять тестирование программных модулей.

– ПК 1.5. Осуществлять оптимизацию программного кода модуля.

– ПК 1.6. Разрабатывать компоненты проектной и технической документации с использованием графических языков спецификаций.

2. ПМ 02. Разработка и администрирование баз данных:

– ПК 2.1. Разрабатывать объекты базы данных.

– ПК 2.2. Реализовывать базу данных в конкретной системе управления базами данных.

– ПК 2.3. Решать вопросы администрирования базы данных.

– ПК 2.4. Реализовывать методы и технологии защиты информации в базах данных.

3. ПМ 03. Участие в интеграции программных модулей:

– ПК 3.1. Анализировать проектную и техническую документацию на уровне взаимодействия компонент программного обеспечения.

– ПК 3.2. Выполнить интеграцию модулей в программную среду.

– ПК 3.3. Выполнить отладку программного продукта с использованием специализированных программных средств.

– ПК 3.4. Осуществлять разработку тестовых наборов и тестовых сценариев.

– ПК 3.5. Производить инспектирование компонент программного продукта на предмет соответствия стандартам кодирования.

– ПК 3.6. Разрабатывать техническую документацию.

## АНАЛИТИЧЕСКАЯ ЧАСТЬ

В этом разделе указана цель разработки данного проекта; проведен анализ существующих аналогов; описаны средства, использованные для разработки дипломного проекта.

1. Исследование предметной области

В современной технологии баз данных предполагается, что создание базы данных, её поддержка и обеспечение доступа пользователей к ней осуществляются с помощью специального программного инструментария – системы управления базами данных (СУБД).

Система управления базами данных (СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Современные СУБД используют технологии для подключения пользователей к базам данных. В зависимости от СУБД предполагается её использование в режиме графического клиента (для удобного управления базами данных, создание структуры данных, производить манипуляции с данными, производить настройки подключения пользователей и разграничения прав доступа).

Микроконтроллер – это микросхема, предназначенная для управления электронными устройствами. По сути это маленький компьютер, имеющий на своем борту процессор, ОЗУ и ПЗУ, но он способен выполнять только лишь относительно простые задачи, из-за небольшой производительности процессора и небольшого количества оперативной памяти.

Чтобы была возможность коммуникации микроконтроллеров между собой, будет использована «Клиент – Сервер» сетевая архитектура, где сервер – это поставщик услуг, то есть на него возложена основная вычислительная нагрузка, а клиент – это заказчик услуг, наши микроконтроллеры, на которые требуется как можно меньшая нагрузка. Если мы хотим получать или передавать какие либо данные, то должны использовать как можно более простой протокол для лучшей быстродейственности.

В целом можно использовать имеющиеся СУБД, написав лишь дополнительные программы для передачи данных в простом виде от микроконтроллеров с последующей переработкой ее в тот вид, в котором принимает СУБД и обратно, но это скажется на быстродействие.

Целью разработки данного проекта является создание СУБД для микроконтроллеров.

Были поставлены следующие задачи:

* обеспечить быструю производительность;
* использовать простой протокол передачи;
* возможность запуска СУБД на большом количестве устройств;
* создать графический клиент для управления базами данных.

1. Обзор и анализ аналогичных решений

Существует множество СУБД, такие как Oracle, MySQL, Microsoft SQL, MariaDB, PostgreSQL, IBM DB2, MongoDB, Firebird, Redis и т.д. Многие СУБД используют одинаковые модели данных, наиболее популярная сейчас модель – реляционная модель данных. Рассмотрим некоторые модели и наиболее популярную базу данных с ней.

1. MySQL (Реляционная модель данных)

MySQL – свободная реляционная система управления базами данных. Компания Oracle осуществляет её разработку и поддержку. Является отличным решением для небольших и средних приложений. Входит в состав множества серверов. Поддерживает много типов таблиц, а благодаря её открытой архитектуре постоянно появляются новые типы таблиц.

Реляционные системы управления базами данных поддерживают реляционную (ориентированную на таблицу) модель данных. Схема таблицы определяется именем таблицы и фиксированным числом атрибутов с фиксированными типами данных. Запись или объект соответствует строке в таблице и состоит из значений каждого атрибута. Таким образом, таблица состоит из набора единых записей.

На рисунке 1 представлена таблица, в которой содержится информация о людях, работающих на каком-то предприятии. Строкой, можно сказать, является описание объекта из реального мира, а столбцы - атрибуты объекта. Каждый атрибут имеет свой тип данных.



Рисунок 1 - Компоненты реляционных отношений

Один или несколько атрибутов должны идентифицировать объект, их называют ключевыми, с помощью них образуются отношения между таблицами. Пример схемы отношений можно увидеть на рисунке 2.



Рисунок 2 - Отношения в базе данных

Для удобной настройки баз данных используется визуальный клиент, наиболее популярный от компании Oracle, именуемы MySQL Workbanch, который представлен на рисунке 3.

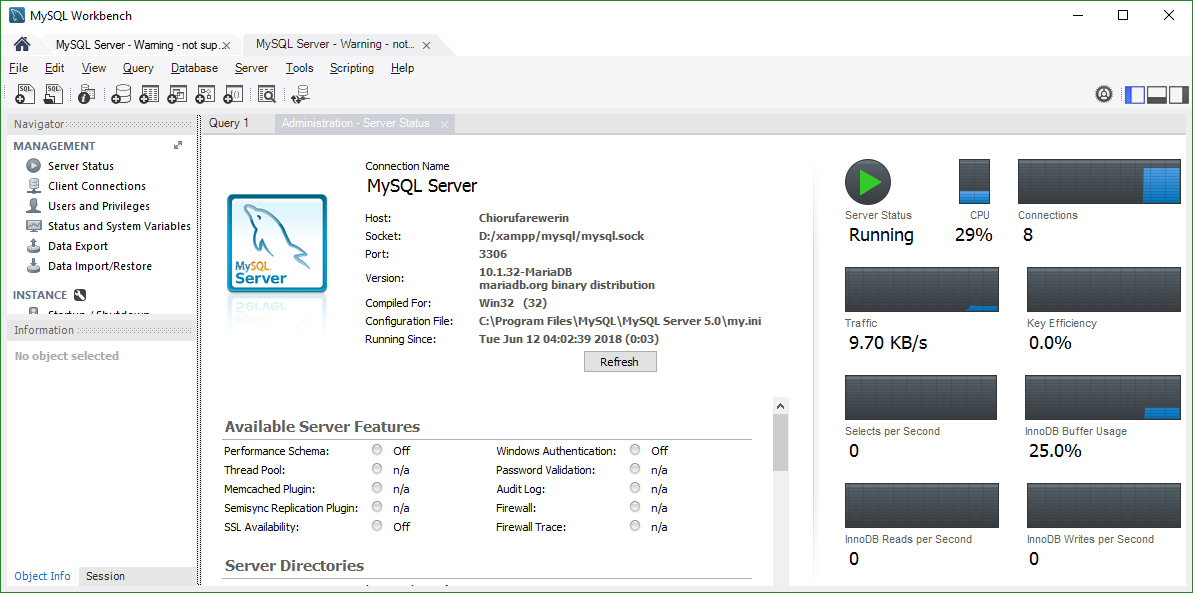


Рисунок 3 - Интерфейс MySQL Workbanch

Достоинства и недостатки MySQL, как и реляционной базы данных в целом можно увидеть в таблице 1.

**Таблица 1 - Анализ MySQL (реляционной базы данных)**

|  |  |
| --- | --- |
| Достоинства | Недостатки |
| Простота, понятность и удобство физической реализации. | Трудоемкость разработки в результате логического проектирования. |
| Применяются строгие правила, основанные на развитом математическом аппарате, который позволяет достаточно лаконично описать операции над данными. | Низкая скорость доступа к данным. |

1. MongoDB (Документированная СУБД)

MongoDB — документоориентированная система управления базами данных с открытым исходным кодом, не требующая описания схемы таблиц. Классифицирована как NoSQL, использует JSON-подобные документы и схему базы данных. Написана на языке C++.

Документоориентированная СУБД — СУБД, специально предназначенная для хранения иерархических структур данных (документов):

* Записи не обязательно должны иметь единую структуру, то есть разные записи могут иметь разные столбцы.
* Типы значений отдельных столбцов могут отличаться для каждой записи.
* Столбцы могут иметь более одного значения (массивы).
* Записи могут иметь вложенную структуру.

Так как, для записи данных используется JSON, то и соответственно информацию об объектах можно представить в виде JSON (см. рисунок 4).

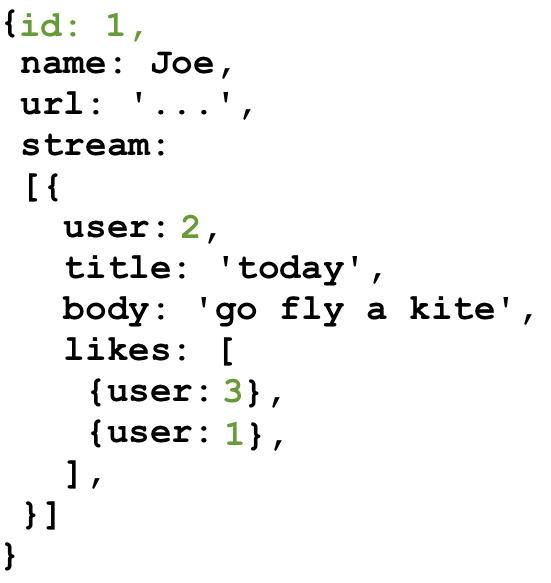


Рисунок 4 - Представление объекта в MongoDB

Для того, чтобы удобно производить настройки СУБД, для MongoDB существует множество средств, наиболее удобное из них – это MongoVUE, интерфейс которого представлен на рисунке 5.

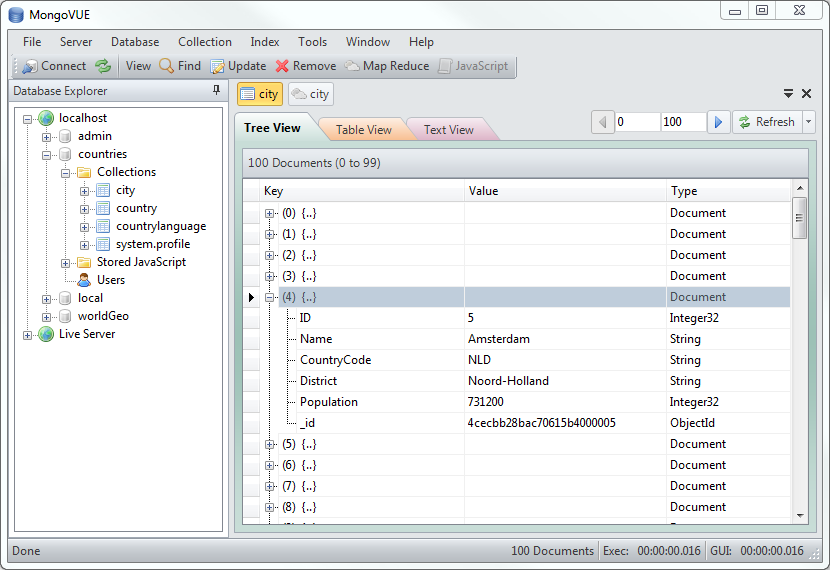


Рисунок 5 - Интерфейс MongoVUE

Достоинства и недостатки MongoDB и других документоориентированных СУБД можно увидеть в таблице 2.

**Таблица 2 - Анализ MongoDB (документоориентированная СУБД)**

|  |  |
| --- | --- |
| Достоинства | Недостатки |
| Гибкая схема хранения данных, что идеально подходит для хранения документов. | Обновление данных негативно сказывается на производительности. |
| Простота разработки и администрирования. | При извлечении значений записи, происходит получение огромного количества данных. |
| Масштабируемость. |

1. Redis (Хранилище «ключ – значение»

Redis — сетевое журналируемое хранилище данных типа «ключ — значение» с открытым исходным кодом. Нереляционная высокопроизводительная СУБД.

Redis считается самой быстрой СУБД в мире, снижает сложность приложений, упрощает разработку, ускоряет выход на рынок и обеспечивает беспрецедентную гибкость для разработчиков с его проницательными структурами данных и модулями.

Хранилища «ключ – значение», вероятно, являются простейшей формой систем управления базами данных. Они могут хранить только пары ключей и значений, а также извлекать значения, когда ключ известен.

Эти простые системы обычно не подходят для сложных приложений. С другой стороны, именно эта простота делает такие системы привлекательными в определенных обстоятельствах. Например, ресурсосберегающие хранилища ключевых значений часто применяются во встроенных системах или в качестве высокопроизводительных баз данных.

Для того, чтобы удобно производить настройку этой СУБД существует несколько визуальных менеджеров, наиболее известный из них – это FastoRedis, интерфейс которого представлен на рисунке 6.

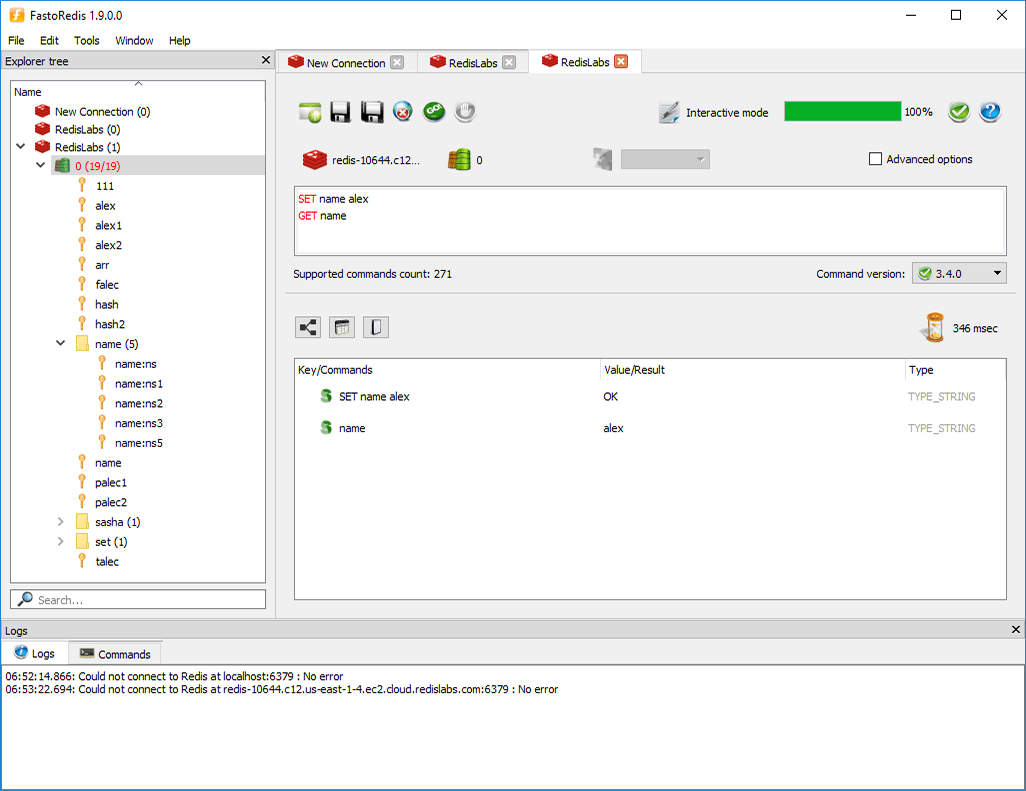


Рисунок 6 - Интерфейс FastoRedis

Достоинства и недостатки Redis и других хранилищ «ключ – значение» можно увидеть в таблице 3.

**Таблица 3 - Анализ Redis «ключ – значение»**

|  |  |
| --- | --- |
| Достоинства | Недостатки |
| Высокое быстродействие благодаря хранению данных в оперативной памяти и простой модели данных. | Примитивная структура данных. |

1. Анализ средств разработки

Прежде чем приступать к разработки программного обеспечения, был произведен анализ языков программирования и средств разработки.

1. Обзор методов решения

В данном разделе были рассмотрены следующие средства разработки: языки программирования и среды разработки для серверной части (СУБД), языки программирования и среды разработки клиентской части (визуальный клиент), а также среда разработки для микроконтроллера.

1. Языки программирования для разработки СУБД

Так как СУБД планируется быть быстродейственной, но в то же время легко изменяемой, небольшой по объему и иметь возможность запуска на большом количестве платформ, решено было выбрать интерпретируемый язык программирования.

На сегодняшний день наиболее популярными интерпретируемыми языками программирования являются:

* Python
* PHP
* JavaScript

1. Python



Рисунок 7 - Логотип языка программирования Python

Python (см. рисунок 7)– интерпретированный язык программирования высокого. Создан Guido van Rossum и впервые выпущен в 1991 году. Python имеет философию программирования, которая подчеркивает читаемость кода, используя значащие пробельные символы. Он предоставляет конструкции, которые обеспечивают создание программ для малых, так и для больших проектов.

Python имеет динамическую типизацию и автоматическое управление памятью. Он поддерживает несколько парадигм программирования: объектно-ориентированный, императивный, функциональный и процедурный.

Интерпретаторы Python доступны практически для всех платформ, такие как Windows, Linux, Mac OS, Solaris, BSD, на мобильных устройствах и, даже, есть его реализация для микроконтроллеров. CPython – эталонная реализация Python, являющаяся программным обеспечением с открытым исходным кодом, имеющая поддержку сообщества.

Благодаря своему синтаксису, код на этом языке очень удобочитаемый, пример можно увидеть на рисунке 8.



Рисунок 8 - Пример кода на языке Python

В своем распоряжении Python имеет множество встроенных библиотек, многие из которых имеют реализацию на C для скорости работы. Из-за большой поддержки разработчиков, имеет огромное количество сторонних библиотек, которые можно свободно использовать в своем программном коде. У этого языка программирования есть своя философия программирования, называемая «The Zen of Python»:

* Красивое лучше, чем уродливое.
* Явное лучше, чем неявное.
* Простое лучше, чем сложное.
* Сложное лучше, чем запутанное.
* Плоское лучше, чем вложенное.
* Разреженное лучше, чем плотное.
* Читаемость имеет значение.
* Особые случаи не настолько особые, чтобы нарушать правила.
* При этом практичность важнее безупречности.
* Ошибки никогда не должны замалчиваться.
* Если не замалчиваются явно.
* Встретив двусмысленность, отбрось искушение угадать.
* Должен существовать один — и, желательно, только один — очевидный способ сделать это.
* Хотя он поначалу может быть и не очевиден, если вы не голландец.
* Сейчас лучше, чем никогда.
* Хотя никогда зачастую лучше, чем прямо сейчас.
* Если реализацию сложно объяснить — идея плоха.
* Если реализацию легко объяснить — идея, возможно, хороша.
* Пространства имён — отличная вещь! Давайте будем делать их больше!

1. PHP



Рисунок 9 - Логотип языка программирования PHP

PHP: Hypertext Preprocessor (см. рисунок 9)– это серверный скриптовый язык, предназначенный для веб-разработки, но также используемый в качестве языка программирования общего назначения. Он был первоначально создан Расмусом Лердорфом в 1994 году. PHP первоначально обозначал Personal Home Page, но теперь он стоит за рекурсивный акроним PHP: Hypertext Preprocessor.

Стандартный PHP-интерпретатор, основанный на Zend Engine, является свободным программным обеспечением, выпущенным под лицензией PHP. PHP широко портирован и может быть использован практически на всех операционных системах и платформах, таких как Windows, Linux, BSD, Mac OS и Solaris.

Пример кода на PHP можно увидеть на рисунке 10.

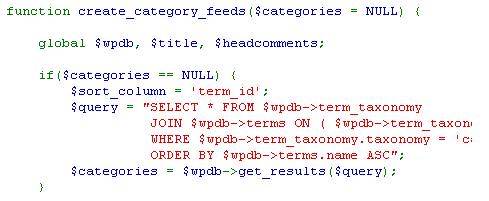


Рисунок 10 - Пример кода на языке PHP

3) JavaScript



Рисунок 11 - Логотип языка программирования JavaScript

JavaScript (см. рисунок 11) – высокоуровневый, интерпретированный язык программирования. Это язык, который характеризуется как динамический, слабо типизированный, прототипный и мультипарадигмальный.

JavaScript, наряду с HTML и CSS, является одной из трех основных технологий всемирной паутины. JavaScript позволяет создавать интерактивные веб-страницы и, таким образом, является неотъемлемой частью веб-приложений. Подавляющее большинство веб-сайтов используют его, и все основные веб-браузеры имеют специальный механизм для его выполнения.

Поддерживает функциональные и императивные (в том числе объектно-ориентированные и прототипные) стили программирования. Он имеет API для работы с текстом, массивами, датами, регулярными выражениями и базовыми манипуляциями с DOM (получать содержимое веб документов и изменять его).

Изначально исполнялся только на клиентской стороне в веб-браузерах, в последствии стал использоваться на веб-серверах, в базах данных, а также для написания настольных и мобильных приложений.

Для написания приложений используется программная платформа node.js, превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Имеет поддержку следующих платформ: Windows, Mac OS, Linux, SunOS и устройствах с архитектурой ARM.

Пример кода на JavaScript можно увидеть на рисунке 12.

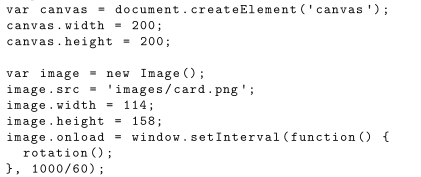


Рисунок 12 - Пример кода на языке JavaScript

1. Среды разработки для создания СУБД

Были рассмотрены следующие среды разработки:

* PyCharm;
* WebStorm;
* Visual Studio Code.

1)PyCharm

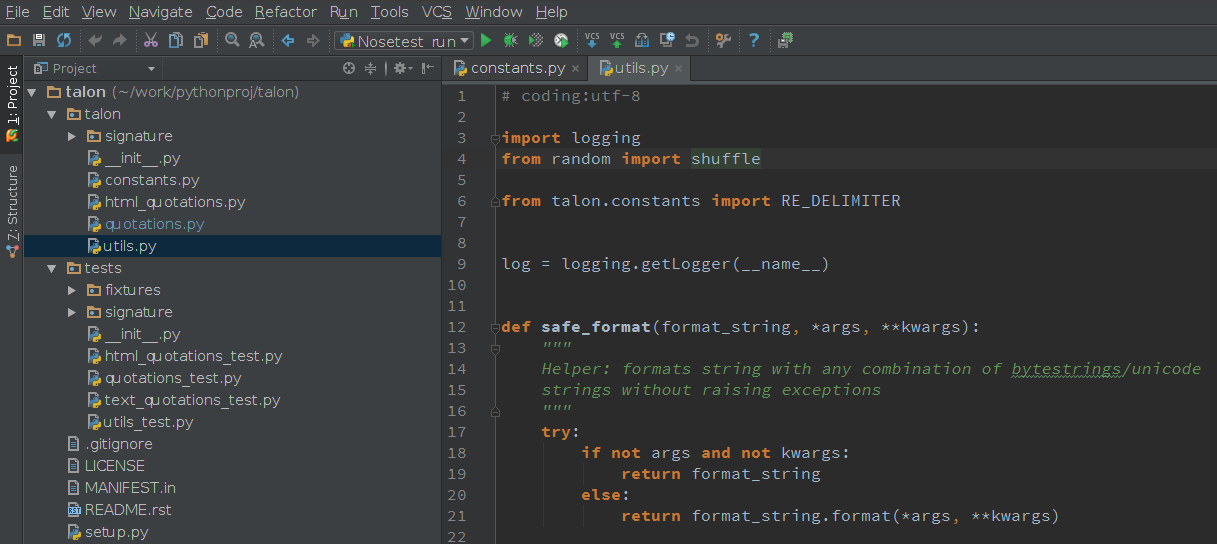


Рисунок 13 - Интерфейс PyCharm

PyCharm (см. рисунок 13) – это интегрированная среда разработки, используемая для программирования на языке Python.

Особенностями этой среды является:

- анализ написания кода и помощь с завершением;

- подсветка синтаксиса и ошибок;

- удобная навигация по проекту;

- рефакторинг кода;

- поддержка веб-фреймворков: Django, web2py и Flask;

- интегрированный отладчик Python.

2)WebStorm

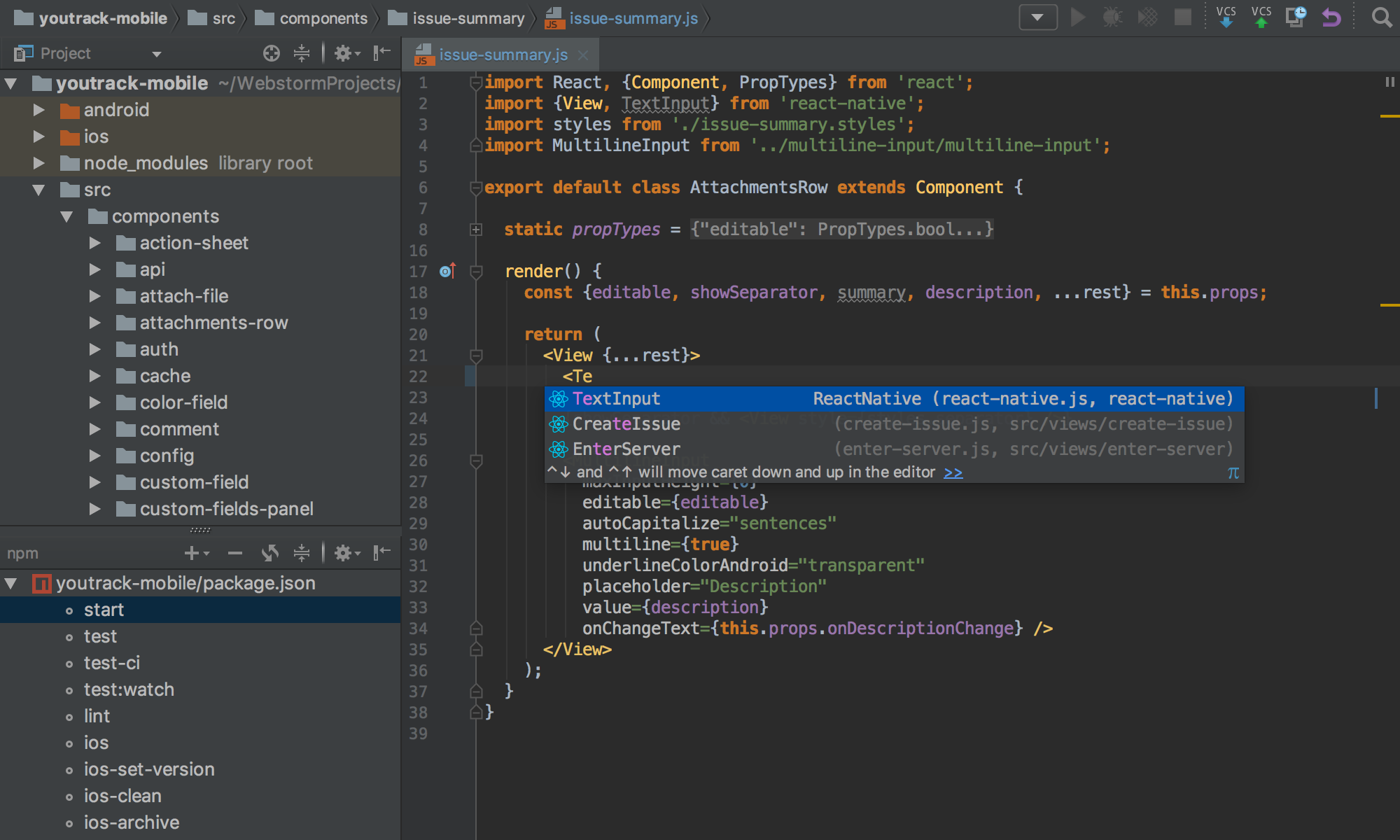


Рисунок 14 - Интерфейс WebStorm

WebStorm (см. рисунок 14) – это интегрированная среда разработки от JetBrains для языка программирования JavaScript. WebStorm поддерживает HTML5, Node.js, Bootstrap, AngularJS, TypeScript, PhoneGap/Cordova и Dart.

Особенностями этой среды являются:

* интеллектуальное автозаполнение;
* проверка кода и быстрое исправление;
* быстрая навигация по коду и поиск кода;
* рефакторинг кода;
* отладка JavaScript.

1. Visual Studio Code

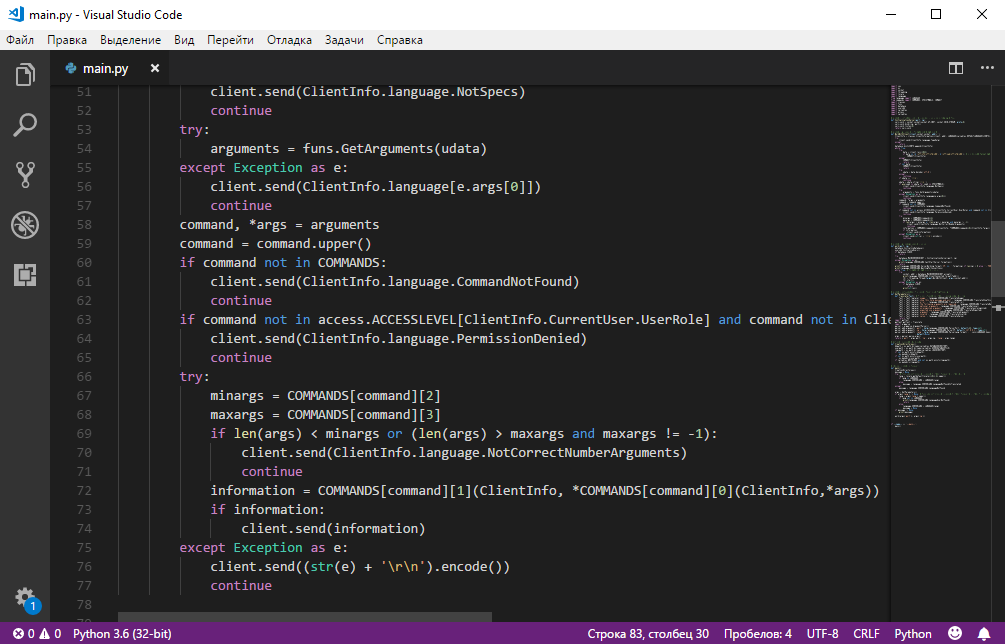


Рисунок 15 - Интерфейс Visual Studio Code

Visual Studio Code (см. рисунок 15) – это редактор исходного кода, разработанный Microsoft для Windows. В обзоре разработчиков Stack Overflow 2018 Visual Studio Code был признан самым популярным инструментом среды для разработчиков, причем 34,9% из 75398 опрашиваемых заявили, что используют его.

Особенностями этой среды является:

* поддержка множества языков программирования;
* расширение возможностей с помощью плагинов;
* поддержка отладки;
* встроенный контроль GIT;
* подсветка синтаксиса;
* интеллектуальное завершение кода;
* рефакторинг.

1. Языки программирования для разработки клиента

Для удобного управления базами данных необходим визуальный клиент.

Рассмотрим наиболее популярные языки программирования для создания оконных приложений:

* Java;
* C#.

1. Java



Рисунок 16 - Логотип языка программирования Java

Java (см. рисунок 16) – сильно типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle).

Предполагается, что разработчики «пишут один раз, а запускают где угодно», это означает, что скомпилированный Java-код может работать на всех платформах, поддерживающих Java, без необходимости перекомпиляции.

Для разработки оконных приложений существует 3 библиотеки: Swing, JavaFX и AWT.

Abstract Window Toolkit (AWT) – это исходная платформо-независимая оконная библиотека графического интерфейса языка Java.

Swing – это набор инструментов для создания GUI на языке программирования Java. Он имеет более мощные и гибкие компоненты, которые написаны полностью на Java и не зависят от платформы, в отличии от AWT. Swing обеспечивает внешний вид приложений, который эмулирует платформу, на которой работает программа.

JavaFX – это программная платформа для создания настольных приложений, которые могут работать на самых разных устройствах. Позволяет использовать xml файлы разметки для GUI. JavaFX предназначен для замены Swing, как стандартной библиотеки.

Пример кода на Java можно увидеть на рисунке 17.

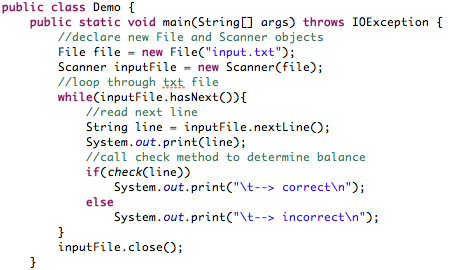


Рисунок 17 - Пример кода на языке программирования Java

1. C#

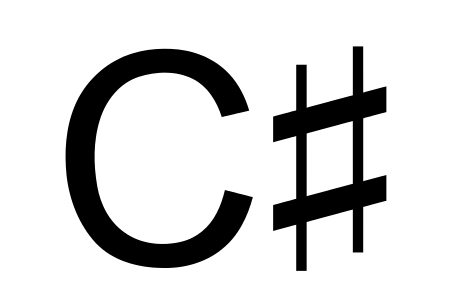


Рисунок 18 - Логотип языка программирования C#

C# (см. рисунок 18) – это объектно-ориентированный язык программирования из семейства с C-подобным синтаксисом, включающий статическую типизацию, полиморфизм, перегрузку операторов, атрибуты, делегаты, события, итераторы, анонимные функции, исключения. Он был разработан Microsoft в рамках .NET, а затем одобрен в качестве стандарта. Опираясь на многие проблемы других языков программирования, исключает некоторые модели, например, в C# в отличии от C++ нет множественного наследования классов.

C# использует среду .NET, которая работает в основном на Windows, но есть ее реализации и на других платформах. Программы, написанные для .NET Framework, выполняются в программной среде Common Language Runtime (CLR), обеспечивающая безопасность, управление памятью и обработку исключений.

Существует 2 основных компонента для создания оконных приложений: WinForms и WPF.

Windows Forms (WinForms) – библиотека классов для создания графических приложений. Будучи первой технологией построения GUI, макет интерфейса WinForms очень упрощен и ограничен. Элементы управления размещаются на расстоянии от верхней левой части окна с указанием размера. Размер элементов управления не может автоматически компенсироваться при изменении размеров окон без неуклюжего и неудобного кода обработчика события.

Windows Presentation Foundation (WPF) – это графическая подсистема Microsoft для визуализации пользовательских интерфейсов в приложениях на базе Windows. WPF, ранее известный как «Авалон», был первоначально выпущен как часть .NET Framework 3.0 в 2006 году. WPF использует модель программирования, позволяющая отделить интерфейс и бизнес-логику приложений, используя XAML (язык на основе XML).

Пример программного кода на языке C# можно увидеть на рисунке 19.



Рисунок 19 - Пример кода на языке программирования C#

1. Среды разработки для создания клиента

Будут рассмотрены следующие среды разработки:

* NetBeans IDE;
* Microsoft Visual Studio.

1. NetBeans IDE

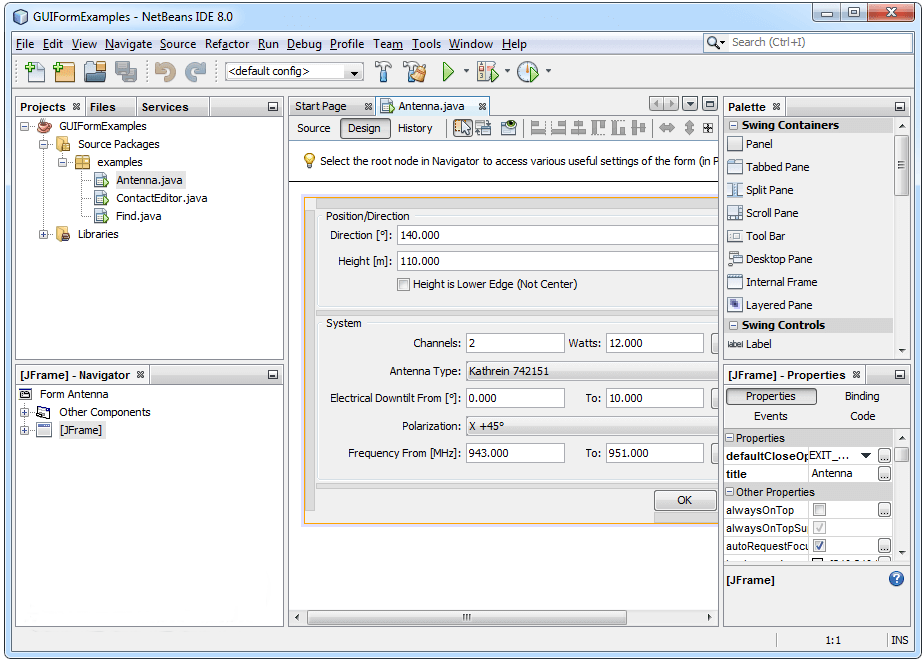


Рисунок 20 - Интерфейс NetBeans

NetBeans IDE (см. рисунок 20) – это интегрированная среда разработки для Java. NetBeans позволяет разрабатывать приложения из набора модульных программных компонентов, называемых модулями. NetBeans работает под управлением Microsoft Windows, MacOS, Linux и Solaris. В дополнение к разработке Java он имеет расширения для других языков, таких как PHP, C, C++, HTML5, Javadoc и JavaScript.

Платформа NetBeans является основой для упрощения разработки настольных приложений Java Swing.

* Особенности платформы:
* подсветка синтаксиса;
* отладка кода;
* поддержка GIT;
* управление пользовательским интерфейсом;
* управление настройками пользователей;
* управление окнами;
* фреймворк для создания пошаговых диалогов;
* GUI Builder - инструмент для визуальной разработки интерфейса;
* Profiler - инструмент для Java-приложений, помогающий искать утечки памяти и оптимизировать скорость работы.

1. Microsoft Visual Studio

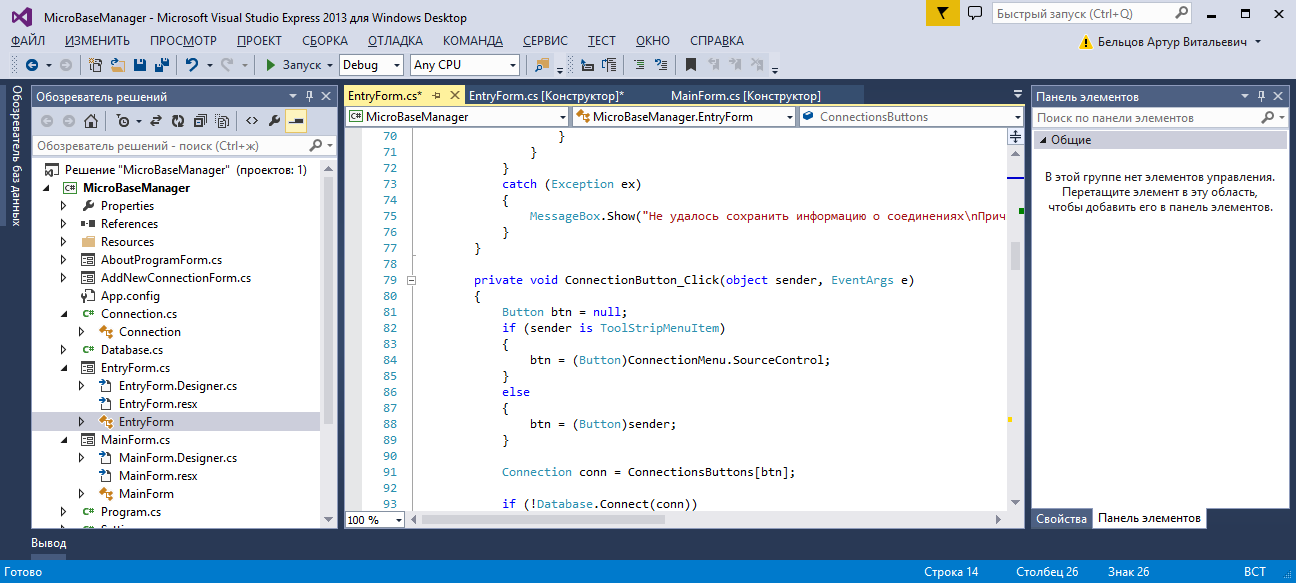


Рисунок 21 - Интерфейс Microsoft Visual Studio

Microsoft Visual Studio (см. рисунок 21) – это интегрированная среда разработки от Microsoft. Она используется для разработки компьютерных программ, а также веб-сайтов, веб-приложений, веб-сервисов и мобильных приложений. Visual Studio использует разработки программного обеспечения Microsoft, такие как Windows API, Windows Forms, Windows Presentation Foundation (WPF), Windows Store и Microsoft Silverlight.

Особенности среды разработки:

* подсветка синтаксиса;
* поддержка IntelliSense (помощь в завершении);
* рефакторинг кода;
* отладчик;
* поддержка 36 различных языков программирования;
* интегрированный отладчик;
* различные конструкторы (форм для графических приложений, классов, схем баз данных);
* поддержка плагинов;
* поддержка систем управления версиями (Subversion и GIT).

Visual Studio не бесплатна для коммерческой разработки и имеет следующие цены:

Visual Studio Enterprise: 2999$/год или 250$/месяц.

Visual Studio Professional: 539$/год или 45$/месяц.

1. Среда разработки для микроконтроллера

Будут рассмотрены следующие среды разработки:

* ESPlorer;
* uPyCraft.

1. ESPlorer

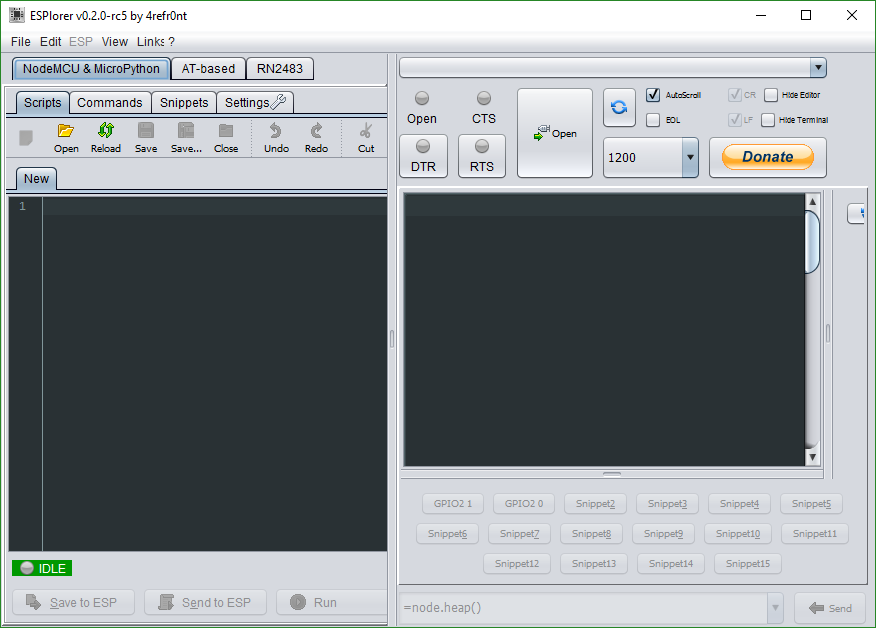


Рисунок 22 - Интерфейс ESPlorer

ESPlorer (см. рисунок 22) – это интегрированная среда разработки для разработчиков, пишущих программы для микроконтроллеров NodeMCU ESP8266. Позволяет использовать несколько языков программирования: LUA и MicroPython. Имеет поддержку таких платформ, как Windows, Linux, Solaris и Mac OS.

Особенности среды разработки:

* подсветка синтаксиса;
* автозаполнение кода;
* интеллектуальная передача данных в ESP8266, с проверкой правильного ответа после каждой строки.

1. uPyCraft

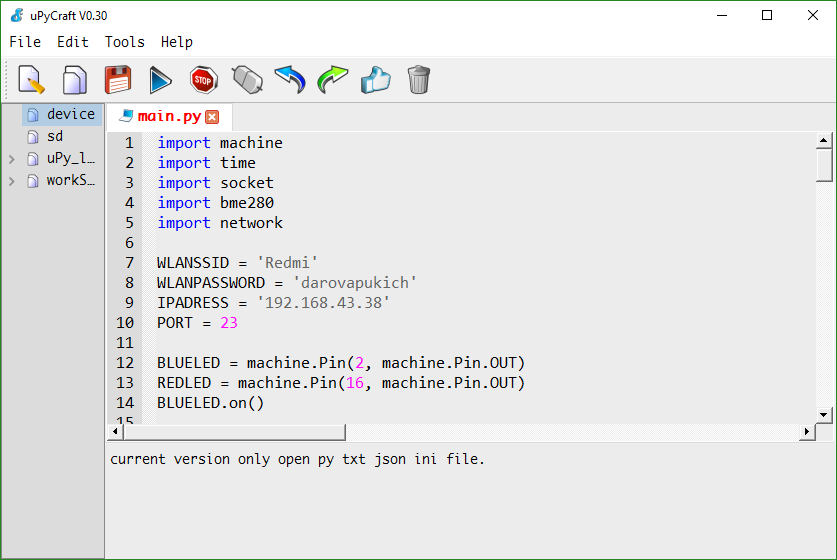


Рисунок 23 - Интерфейс uPyCraft

uPyCraft (см. рисунок 23) – это интегрированная среда разработки, позволяющая разрабатывать программы для микроконтроллеров, используя исключительно MicroPython.

Особенности среды разработки:

* подсветка синтаксиса;
* загрузка файлов на микроконтроллер;
* возможность установки прошивки;
* удобный доступ к файлам микроконтроллера с возможностью их изменения.

1. Обоснования выбора методов

Произведя анализ доступных методов, были выбраны наиболее подходящие решения.

1. Язык программирования для разработки СУБД

В ходе анализа доступных языков программирования, был выбран язык программирования Python.

Python подходит по многим критериям.

Во-первых, качество программного кода. Синтаксис языка подразумевает использования пробельных символов для выделения блоков кода, что вынуждает программировать в едином стиле. На каждую строку приходится отдельная конструкция. Благодаря этому код на этом языке удобочитаем, а его обслуживание выполняется гораздо проще, в отличии от других языков программирования.

Во-вторых, высокая скорость разработки. В отличии от других языков программирования, таких как C, C++ или Java, объем программного кода на языке Python составляет треть, а порой даже и пятую часть эквивалентного кода. Это означает меньший объем работы, меньший ввод данных и меньшее количество времени на отладку.

В-третьих, переносимость программ. Большая часть программ выполняется без изменений на всех основных платформах, все что требуется от разработчика, это перенести программный код с одного устройства на другое.

В-четвертых, интеграция компонентов. Сценарии Python могут легко взаимодействовать с другими частями приложений. Эта интеграция позволяет не только использовать библиотеки Python, но и возможность вызывать функции из библиотек на языке C/C++.

В-пятых, высокое быстродействие. Программы на python, использующие встроенные конструкции и функции языка, фактически выполняются со скоростью, которую может дать язык C.

1. Среда разработки для создания СУБД

Исходя из анализа средств разработки, была выбрана Visual Studio Code.

Visual Studio Code прежде всего удобный редактор кода с подсветкой синтаксиса. Потенциал среды разработки раскрывается благодаря установки расширений, которые позволяют использовать ее как полноценную среду разработки, с использованием IntelliSense (помощника для автозаполнения), дебаггера, рефакторинга и т.д.

Эта среда разработки с открытым исходным кодом, поэтому она постоянно дорабатывается благодаря сообществу, которое сформировалось уже немалое, а популярность этой среды стремится лишь только вверх.

1. Язык программирования для разработки клиента

Исходя из анализа языков программирования, был выбран C#.

C# относительно молодой, но в то же время является мощным инструментом для разработки программного обеспечения. Этот язык программирования вобрал в себя лучшее из таких языков, как Java, C++, Pascal, Modula и Smalltalk.

.NET Framework позволяет разработчикам использовать виртуальную машину, обеспечивая безопасность, контроль памяти и обработку исключений.

Такая работа приложений обеспечивается благодаря исполняющей среде для байт-кода (Common Language Runtime, или сокращенно – CLR). Она исполняет «высокоуровневый ассемблер» – промежуточный язык, разработанный Microsoft, в который транслируют все компиляторы, поддерживающие платформу .NET Framework.

1. Среда разработки для создания клиента

Исходя из анализа средств разработки и выбранного языка программирования, была выбрана Microsoft Visual Studio.

Microsoft Visual Studio является мощной интегрированной средой разработки программного обеспечения, захватившая лидирующие позиции на рынке программных средств, используемых для создания программ.

Эта среда для разработки очень удобна в использовании, содержит в себе множество средств для разработки: IntelliSense (помогает программисту в завершении команды, а также осуществляет поиск ошибок и предлагает варианты их решения), отладчик, рефакторинг кода, средства для создания графических интерфейсов, веб-редактор, дизайнер классов и дизайнер схем базы данных.

1. Микроконтроллер и среда разработки для него

Была поставлена задача – выбрать микроконтроллер, позволяющий передавать информацию по сети. Было предложено множество вариантов, и условия, которые должен выполнять микроконтроллер – это иметь в наличии Wi‑Fi модуль, а также прошиваемость, позволяющую программировать контроллер на более высокоуровневом языке программирования, нежели C, который поддерживают практически все.

Был вариант купить микроконтроллер и припаять к нему модуль, поддерживающий Wi-Fi, но уже есть готовое решение NodeMCU ESP8266, который имеет на борту и интегральную схему, и модуль Wi-FI, а также чип ESP8266, который позволяет ставить прошивку не только NodeMCU с языком Lua, но и MicroPython с языком Python.

Так как имеется в распоряжении NodeMCU ESP8266, то была установлена прошивка MicroPython, из-за этого выбор пал на uPyCraft, наиболее удобную IDE для этого языка программирования.

## ПРОЕКТНАЯ ЧАСТЬ

В данном разделе рассмотрены основные требования к программе, функциональные требования, технические средства.

Также будут представлены инструкции для пользователя при первом знакомстве с программным продуктом.

1. Техническое задание

Целью данного проекта является создание программы, а конкретно систему управления базами данных для микроконтроллеров, которая позволила бы передавать информацию с максимальным возможным быстродействием, с возможностью запуска на как можно большем количестве устройств, занимала сравнительно небольшой объем на жестком диске, а также, при необходимости, имела возможность быстрой доработки для специфичных задач.

1. Постановка задачи

Задачей дипломного проекта является создание системы управления базами данных, а также создание программы для удобного управления базами данных в СУБД.

Система управления базами данных должна иметь следующие функции:

* подключение клиентов посредством сетевого протокола;
* возможность использования Telnet для подключения;
* регистрация пользователей и разграничение ролей;
* создание баз данных и их управление при помощи команд;
* выбор языка;
* возможность осуществлять редактирование программного кода;
* сохранение баз данных в файлы, а также обеспечить их нахождение в оперативной памяти для лучшего быстродействия;
* запись и получение данных;
* выводить справочную информацию;
* осуществлять прослушивание баз данных, для получения актуальной информации об текущих изменения;
* осуществлять логирование.

Клиент, для визуального управления базами данных, должен осуществлять следующие функции:

* осуществлять подключение к СУБД;
* сохранение подключений;
* управление базами данных и данными при помощи визуальных средств;
* вести графики изменений данных;
* выдача информации в удобном для пользователя виде.

1. Назначение программного продукта

Программа, а конкретно СУБД, должна быть использована для сохранения и выдачи данных для клиентов с наименьшей задержкой, осуществлением управления базами данных (создания, редактирования, удаления, изменения прав доступа), возможностью запуска на как можно большем количестве устройств.

Визуальный клиент должен служить незаменимым инструментом для удобного управления базами данных в СУБД.

1. Основание для разработки

Основанием для разработки послужили: задание на преддипломную практику и задание на диплом.

1. Требования к программе

В данном разделе описываются требования к программному обеспечению и аппаратным характеристикам для правильного функционирования программ.

1. Требования к функциональным характеристикам

Требования для системы управления базами данных:

* подключение клиентов посредством сетевого протокола;
* возможность использования Telnet для подключения;
* регистрация пользователей и разграничение ролей;
* создание, управление и удаление баз данных при помощи команд;
* возможность выбора языка;
* обеспечить нахождение баз данных в ОЗУ;
* использовать периодическое сохранения баз данных в файл и возможность принудительного их сохранения с помощью команды;
* отправка и получение информации;
* выводить справочную информацию;
* отправлять данные об текущих изменениях в базах данных;
* вести историю изменений.

Требования для визуального клиента:

* осуществлять подключение к системе управления базами данных;
* сохранять подключения для последующего использования;
* использование визуальных средств для осуществления управления базами данных и пользователями;
* использовать графики для вывода информации об изменениях данных;
* выдача информации в удобном для пользователя виде.

1. Требования к аппаратным и программным средствам

Для работы СУБД необходимы следующие программные компоненты:

* Python 3, рекомендуется использовать последнюю версию, на момент написания – это 3.6.5;
* дополнительные модули для Python: gettext, psutil и argparse.

Для работы менеджера по управления базами данных необходимо установить .NET Framework 4.5.

Минимальные и рекомендуемые системные требования для СУБД представлены в таблице 4, а для менеджера в таблице 5.

**Таблица 4 - Системные требования для СУБД**

|  |  |  |
| --- | --- | --- |
| Характеристика | Минимальные | Рекомендуемые |
| Операционная система | * Windows * Linux * Mac OS * iOS * Solaris | * Windows * Linux * Mac OS * iOS * Solaris |
| Процессор | 800 MHz | 1,3 GHz |
| Свободное место на ПЗУ | 10 Мб | 50 Мб |
| Оперативная память | 256 Мб | 528 Мб |

**Таблица 5 - Системные требования для визуального менеджера**

|  |  |  |
| --- | --- | --- |
| Характеристика | Минимальные | Рекомендуемые |
| Операционная система | * Windows 7 * Windows 8 * Windows 10 | * Windows 7 * Windows 8 * Windows 10 |
| Процессор | 1,7 MHz | 2,4 GHz |
| Свободное место на ПЗУ | 50 Мб | 100 Мб |
| Оперативная память | 1024 Мб | 2048 Мб |

Для демонстрации реализации передачи данных на сервер, необходимо использовать микроконтроллер, а также какой-либо модуль, позволяющий собирать информацию, например, о температуре, атмосферном давлении.

Требования к микроконтроллеру:

* беспроводной интерфейс: Wi-Fi;
* частота процессора: 80 МГц;
* ОЗУ: 96 Кб;
* ПЗУ: 64 Кб;
* порты ввода-вывода.

1. Требования к надежности

Так как это система управления базами данных, то требования к надежности тут должны быть на самом высоком уровне.

Здесь используются следующие средства:

* проверка введённых команд;
* проверка целостности баз данных;
* разграничение прав доступа для пользователей;
* использование шифрования для пароля.

Если произойдет отказ системы, что несомненно может привести к потере данных, используется периодическое автоматическое сохранение данных через определенные промежутки времени.

Для менеджера по управлению базами данных происходит проверка введенных данных от пользователя, чтобы не возникало непредвиденных ошибок из-за некорректной информации.

1. Требования к надежной функциональности системы

Интерпретируемый язык программирования хорош тем, что при возникновении каких-либо ошибок, полноценно оповещает о том, что пошло не так, и в каком месте произошла эта ошибка, поэтому при отсутствии каких-либо данных или компонентов, об этом будет сразу сообщено.

Но следует, для большего удобства, обработать наиболее вероятные события, которые могут возникнуть, с предложением вариантов их решения. Поэтому, если какая-либо база данных повреждена, пользователю предлагается ее пересоздать, либо не трогать и попробовать решить проблему самому.

1. Отказ из-за некорректных действий пользователей системы

На всем протяжении работоспособности программы необходимо осуществлять проверку всех введённых данных от пользователей, поступающих в систему.

Так как это СУБД, то в нее могут приходить такие данные, которые могут привести к незапланированным ошибкам, возникает необходимость к проверке получаемых команд, количестве введённых аргументов, вводу запрещенных символов. Если это происходит, то сообщать об этом пользователю.

В клиентском приложении также необходимо проверять корректность заполнения всех данных, чтоб не возникало непредвиденных ошибок, и приложение аварийно не завершало свою работу.

1. Требования к пользователям системы

При подключении к серверу, пользователю предлагается получить справку по коммандам, благодаря этому ему можно легко разобраться, как работать с этой СУБД, если уже имел дело с подобным программным обеспечением.

1. Защита и сохранность данных

Для защиты получения информации от пользователей, которые не должны иметь доступа к ней, используется разграничение прав доступа. Администратор СУБД может настраивать это под свое усмотрение, при необходимости, а также давать права доступа на некоторые команды для определенных пользователей. Пользователи, создающие свои базы данных, автоматически становятся ее владельцами, и, если необходимо, могут в нее добавлять других пользователей, и назначать права конкретно на эту базу данных.

Чтобы никто не узнал используемый пароль, даже администратор СУБД, используется его шифрование с помощью SHA 256 – наиболее защищенной на сегодняшний день метод криптографического хеширования.

1. Описание функционально-логической структуры программы

Для описания схемы работы программы, используются различные диаграммы, представленные ниже.

1. Диаграмма потоков данных (DFD)

DFD (data flow diagrams) – диаграммы потоков данных, являющаяся основным инструментом структурного анализа и проектирования информационных систем. Цель этого описать внешние по отношению к системе источники и адресаты данных, логические функции, потоки данных и хранилища данных, а также увидеть, как в процессе входные данные преобразуются в выходные. Диаграммы можно увидеть на рисунке 24 или ее крупный аналог на рисунке Б.1 и рисунке 25 или крупном аналоге Б.2.

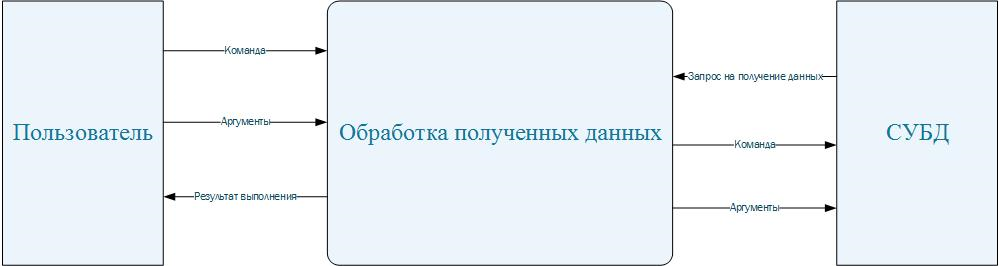


Рисунок 24 - Концептуальная DFD диаграмма

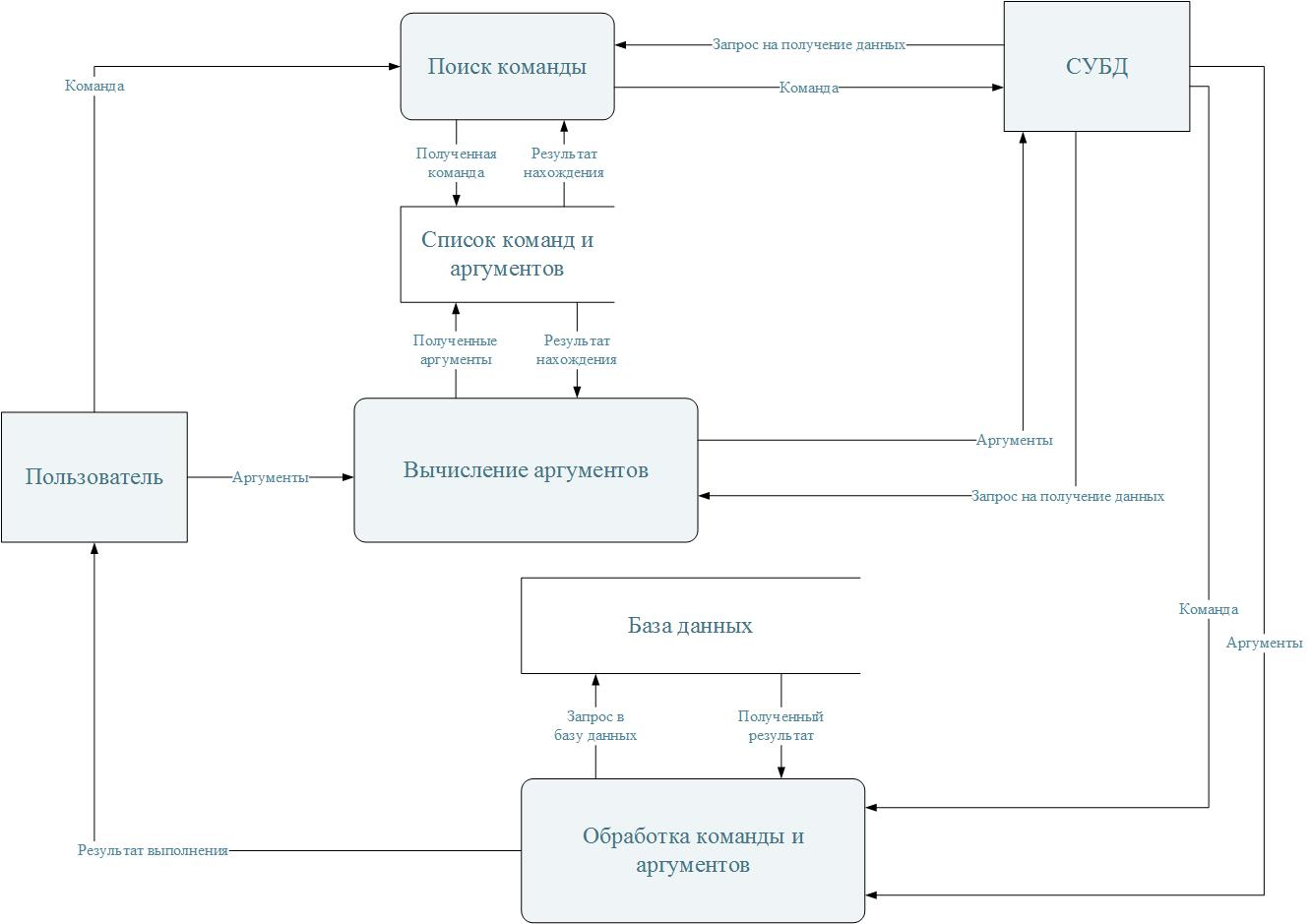


Рисунок 25 - Детализированная DFD диаграмма

1. Функциональная схема программы

Функциональная схема или схема данных – схема, разъясняющая процессы, в отдельных функциональных компонентах программного обеспечения с описанием информационных потоков, состава данных в потоках и указанием используемых файлов и устройств.

Функциональную схему можно увидеть на рисунке 26 или крупном аналоге Б.3.

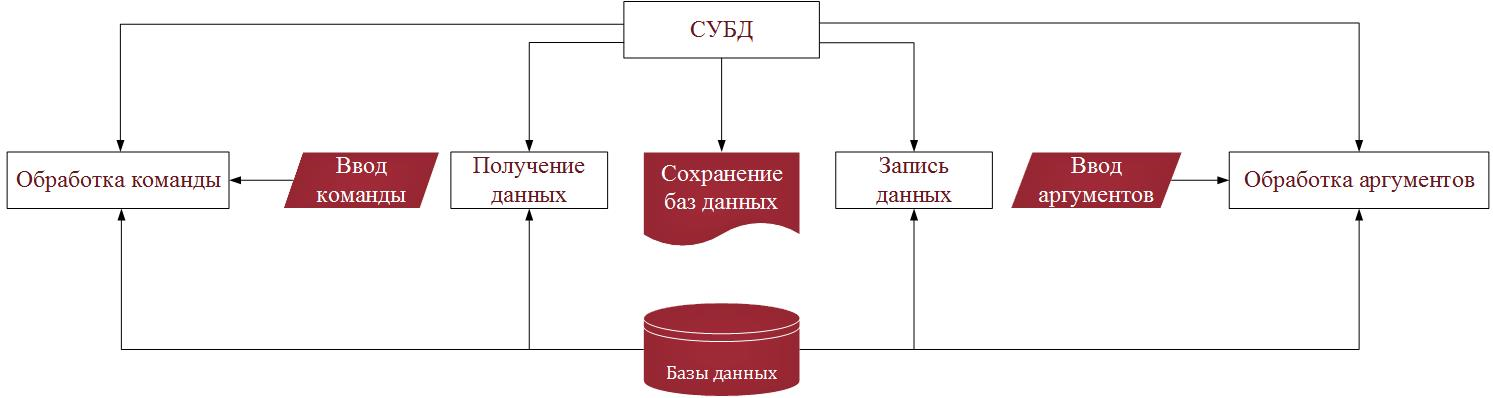


Рисунок 26 - Функциональная схема программы

1. Функциональные диаграммы работы программы (IDEF0)

В основе Integrated Definition Function Modeling (IDEF0) методологии лежит понятие блока, который отображает некоторую бизнес-функцию. Четыре стороны блока имеют разную роль: левая сторона имеет значение «входа», правая – «выхода», верхняя – «управления», нижняя – «механизма».

Контекстная IDEF0 диаграмма представлена на рисунке 27, а детализированная на рисунке 28 или крупном аналоге Б.4.



Рисунок 27 - Контекстная диаграмма IDEF0

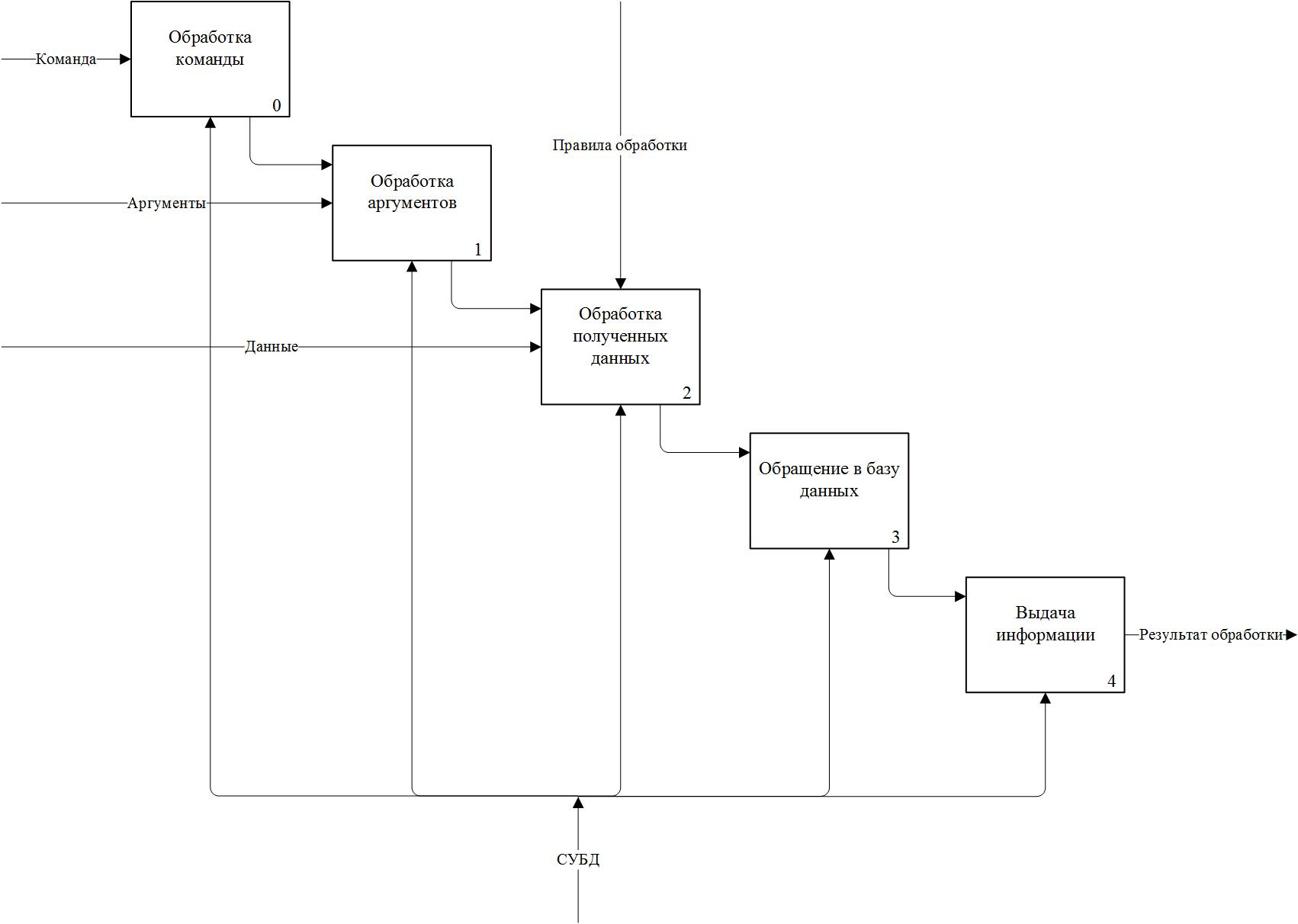


Рисунок 28 - Детализированная диаграмма IDEF0

1. Диаграмма вариантов использования (UML)

Диаграмма вариантов использования, также известная как диаграмма прецедентов – это диаграмма, отражающая отношения между актерами и прецедентами, позволяющая описать программу на концептуальном уровне.

Актером может быть любая сущность, которая взаимодействует с системой, а прецедент – это часть функциональности системы, благодаря которому актер может получить результат.

В данном случае актером может быть незарегистрированный пользователь, зарегистрированный пользователь или администратор. У каждого актера свои возможности, которые могут как быть доступны им всем, или же только некоторым. Администратор системы при необходимости может менять варианты использования для актеров, но по умолчанию в системе используется модель, отраженная на рисунке 29 или крупном аналоге Б.5.

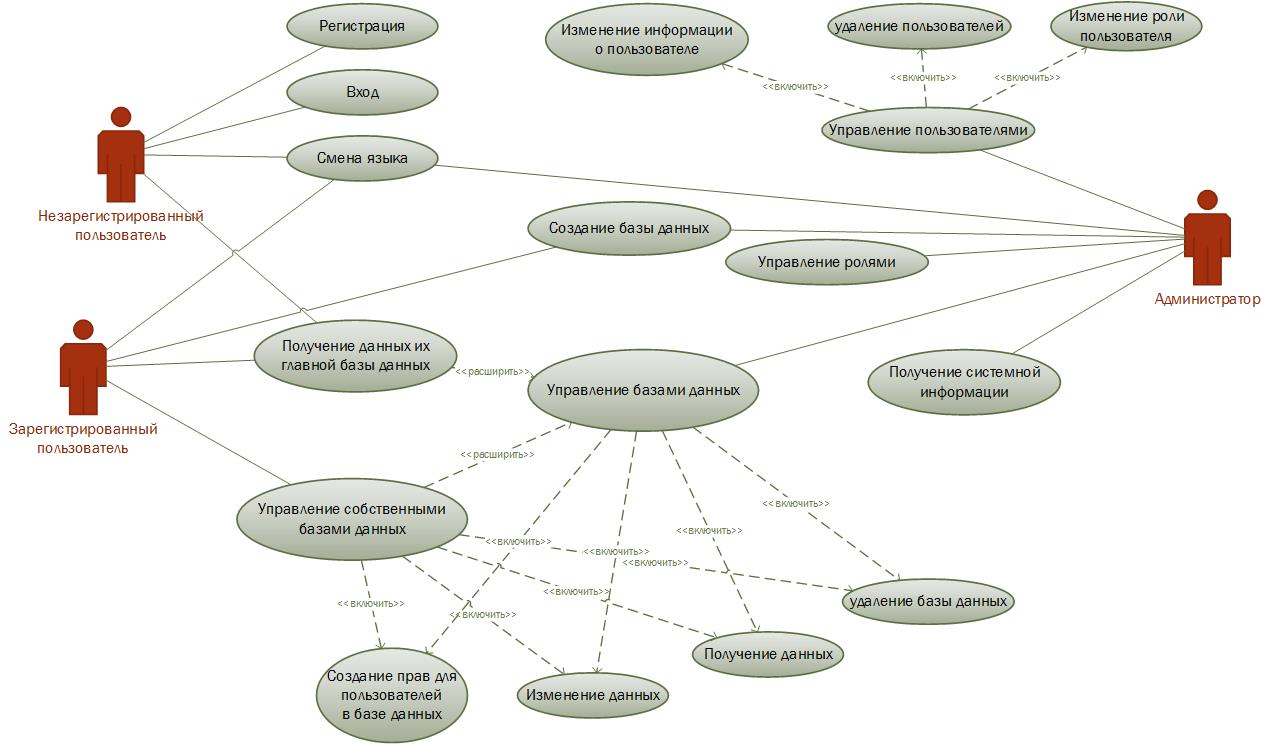


Рисунок 29 - Диаграмма вариантов использования

1. Описание физико-математической модели

В данном разделе описаны основные теоретические аспекты, термины и формулы, используемые в приложении.

1. Хэширование (SHA-256)

Хэширование – это преобразование данных произвольной длины в выходную строку фиксированной длины, выполняемое определенным алгоритмом. Оно нужно для вычисления хэша, например, файла, и если файл мы кому-то передали, то он должен иметь такой же хэш. Так что оно нужно прежде всего для проверки целостности.

Также хэширование отлично подходит для сохранения паролей, чтобы нельзя было его узнать или подобрать, ведь при изменении даже бита данных, уже на выходе получается совершенно другое значение. Проблема, которая преследует все хэш функции – это вычисление коллизий, получение одних и тех же данных на выходе, при поступлении двух и более разных данных, поэтому чем сложнее алгоритм и большее количество выходных байтов, тем сложнее вычислять эти коллизии.

В программе будет использоваться алгоритм хэширования, как SHA-256, он входит в семейство алгоритмов хэширования SHA-2. Все эти хэш функции разработаны на структуре Меркла-Дамгарда, которая подразумевает разбиение сообщения на блоки фиксированной длины, с каждым таким блоком функция работает по очереди, используя значения полученные функцией из предыдущего блока и новым поступающим блоком.

Пример работы хэш-функций, построенных на структуре Меркла-Дамгарда, можно увидеть на рисунке 30. Используются уже изначально заготовленное значение, которое лежит в VAL, дальше используется первый блок сообщения и преобразовывается с изначальным значением в новое значение, в последствии со вторым блоком будет использоваться уже это значение. Каждое такое преобразование называется раундом, и после всех уже этих раундов значение идет в финальную функцию, где уже может обрезаться до нужной длины (как, например, SHA-224, которая очень похожа на SHA-256, только последние 4 байта отбрасываются) и преобразовать значение битов в одну строку, что и будет называться хэшем.

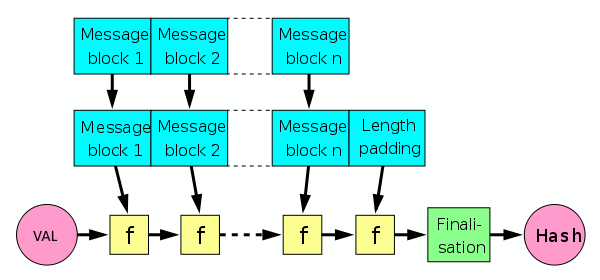


Рисунок 30 - Пример работы алгоритма однонаправленного хэширования

SHA-256 имеет на выходе 32 байта, или же строку из 64 символов, найти коллизии будет очень ресурсозатратно на данный момент времени и в ближайшие несколько лет это наиболее защищенная и быстрая хэш-функция.

Исходное сообщение разбивается на блоки, каждый блок — на 16 слов. Алгоритм пропускает каждый блок сообщения через цикл с 64 или 80 итерациями (раундами). На каждой итерации 2 слова преобразуются, функцию преобразования задают остальные слова. Результаты обработки каждого блока складываются, сумма является значением хеш-функции. Тем не менее, инициализация внутреннего состояния производится результатом обработки предыдущего блока.

Алгоритм использует следующие битовые операции:

* ǁ — конкатенация,
* + — сложение,
* and — побитовое «И»,
* xor — исключающее «ИЛИ»,
* shr (shift right) — логический сдвиг вправо,
* rotr (rotate right) — циклический сдвиг вправо.

Как работает алгоритм SHA-256:

* Инициализация исходных значений и констант. Исходные значения нужны для последующего преобразования с блоками сообщения, а константы в работе функциях при преобразовании значений.
* Предварительная обработка, которая разбивает сообщение на одинаковые блоки размером 512 бит, и записывает длину сообщения
* Далее происходит работа над каждым блоком, для начала он разбивается на 16 слов, и записывается в некоторый массив w. Так как работа происходит с 64 словами, то для начала вычисляются эти недостающие 48 слов:

**для** i **от** 16 **до** 63

s0 = (w[i-15] **rotr** 7) **xor** (w[i-15] rotr 18) **xor** (w[i-15] shr 3)

s1 = (w[i-2] **rotr** 17) **xor** (w[i-2] rotr 19) **xor** (w[i-2] shr 10)

w[i] = w[i-16] + s0 + w[i-7] + s1

где rotr – можно представить, как:

((x >> y) | (x << (32-y))) & 0xFFFFFFFFL

shr – можно представить, как

x >> y

xor – можно представить, как:

x & !y | !x & y

Далее, используя эти 64 слова, происходит преобразование:

**для** i **от** 0 **до** 63

Σ0 = (a **rotr** 2) **xor** (a **rotr** 13) **xor** (a **rotr** 22)

Ma = (a **and** b) **xor** (a **and** c) **xor** (b **and** c)

t2 = Σ0 + Ma

Σ1 = (e **rotr** 6) **xor** (e **rotr** 11) **xor** (e **rotr** 25)

Ch = (e **and** f) **xor** ((**not** e) **and** g)

t1 = h + Σ1 + Ch + k[i] + w[i]

h = g

g = f

f = e

e = d + t1

d = c

c = b

b = a

a = t1 + t2

* После всех этих манипуляций с каждым блоком, последняя функция преобразует полученные биты данных в текст, длинной 64 символа.

Наглядно процесс одной итерации можно увидеть на рисунке 31:

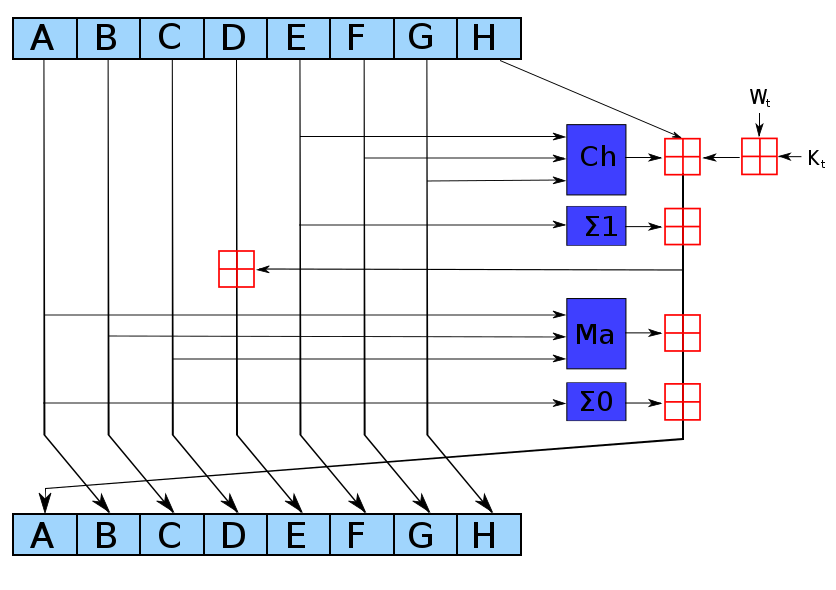


Рисунок 31 - Схема итерации алгоритма SHA-256

1. Логическая модель данных

Для того, чтобы удобно хранить данные, а также к ним обращаться, было решено создать отдельные структуры или классы, для тех объектов, которые имеют наибольшую значимость в программе.

1. Структуры и классы на сервере

Для удобства хранения были разработаны такие структуры или классы, которые лежат в файле classes.py.

1. Класс DataBase

Класс представляет собой хранилище данных для представления базы данных в программе. В класс, который можно увидеть на рисунке 32, включены основные поля и метод инициализации. Описание класса можно увидеть в таблице 6.

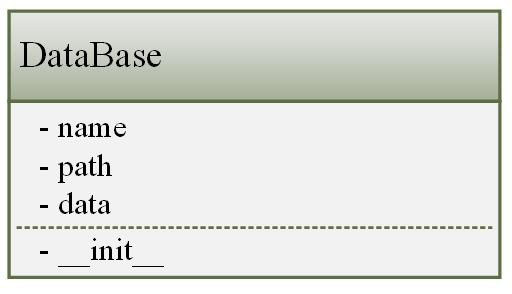


Рисунок 32 – Структура класса DataBase

**Таблица 6 - Описание класса DataBase**

|  |  |
| --- | --- |
| Название | Описание |
| Переменная name | Название базы данных |
| Переменная path | Место хранения базы данных на внешней памяти |
| Переменная data | Представляет из себя ассоциативный массив, где ключом является название переменной, а значением является значение переменной. |
| Метод \_\_init\_\_ | Стандартный конструктор в языке python, используется для инициализации значений |

1. Класс AccessDataBase

С помощью данного класса хранится информация о привилегиях пользователя в какой-либо базы данных. Структуру класса можно увидеть на рисунке 33, а описание класса в таблице 7.

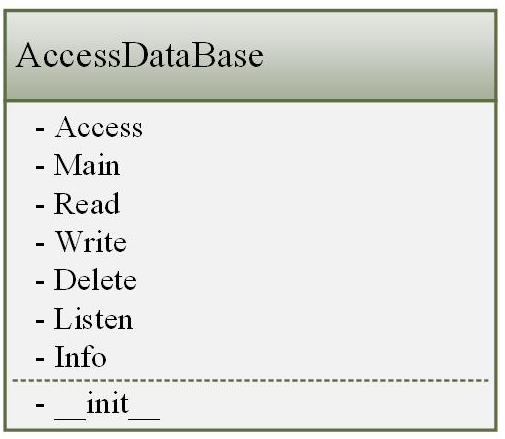


Рисунок 33 - Структура класса AccessDataBase

**Таблица 7 - Описание класса AccessDataBase**

|  |  |
| --- | --- |
| Название | Описание |
| Переменная Access | Возможность входа |
| Переменная Main | Возможности администратора |
| Переменная Read | Возможность получать данные |
| Переменная Write | Возможность записывать и изменять данные |
| Переменная Delete | Возможность удалять данные |
| Переменная Listen | Возможность прослушивания базы данных |
| Переменная Info | Возможность получать информацию о базе данных |
| Метод \_\_init\_\_ | Стандартный конструктор в языке python, используется для инициализации значений |

1. Класс User

С помощью данного класса хранится информация о пользователе. Структуру класса можно увидеть на рисунке 34, а описание класса в таблице 8.

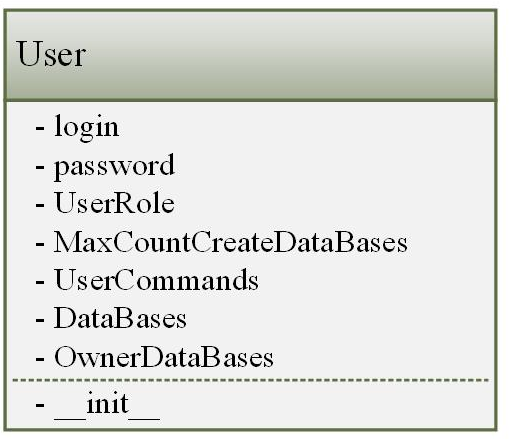


Рисунок 34 – Структура класса User

**Таблица 8 - Описание класса User**

|  |  |
| --- | --- |
| Название | Описание |
| Переменная login | Имя пользователя |
| Переменная password | Пароль пользователя (хранится в виде значения хэш-функции SHA-256) |
| Переменная UserRole | Роль пользователя в СУБД |
| Переменная MaxCountCreateDataBases | Максимальное количество баз данных, которые может создать пользователь |
| Переменная UserCommands | Дополнительные разрешенные команды пользователю, помимо команд доступных его ролью |
| Переменная DataBases | Ассоциативный массив баз данных, хранящий информацию о том, какие возможности имеет пользователь в базе данных. Возможности находятся в структуре AccessDataBase |
| Переменная OwnerDataBases | Список баз данных, которые создал этот пользователь |
| Метод \_\_init\_\_ | Стандартный конструктор в языке python, используется для инициализации значений |

1. Класс ClientInformation

С помощью данного класса хранится информация о подключениях к СУБД. Структуру класса можно увидеть на рисунке 35, а описание класса в таблице 9.

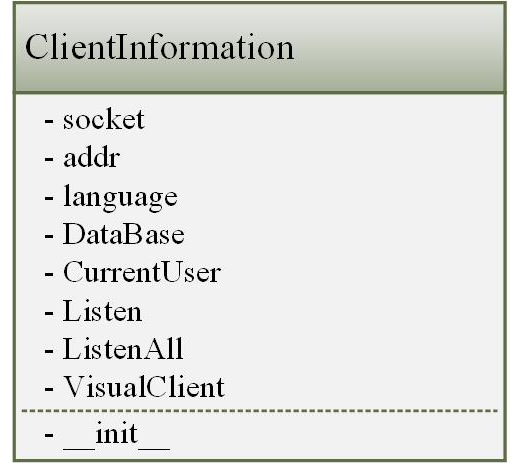


Рисунок 35 - Структура класса ClientInformation

**Таблица 9 - Описание класса ClientInformation**

|  |  |
| --- | --- |
| Название | Описание |
| Переменная socket | Сокет подключения |
| Переменная addr | Адрес и порт подключения |
| Переменная language | Язык, используемый клиентом |
| Переменная DataBase | Текущая база данных, в которой находится клиент |
| Переменная CurrentUser | Пользователь, используемый клиентом |
| Переменная Listen | Информация, слушает ли клиент текущую базу данных |
| Переменная ListenAll | Информация, слушает ли клиент все изменения в СУБД |
| Переменная VisualClient | Информация, является ли подключения с визуального менеджера по управлению базами данных |
| Метод \_\_init\_\_ | Стандартный конструктор в языке python, используется для инициализации значений |

1. Класс Language

Этот класс нужен лишь для определения метода, который позволяет возвращать значения атрибута объекта с помощью квадратных скобок (как будто берем значение из массива). От этого класса уже будут наследоваться другие языки, которые, в том числе, унаследуют этот метод. Структуру класса можно увидеть на рисунке 36, а описание класса в таблице 10.



Рисунок 36 - Структура класса Language

**Таблица 10 - Описание класса Language**

|  |  |
| --- | --- |
| Название | Описание |
| Метод \_\_getitem\_\_ | С помощью этого метода реализована возможность получить значение атрибута объекта с помощью обращения через квадратные скобки, как если бы это было обращение к массиву. |

1. Класс EnglishLanguage

Этот класс нужен для хранения в атрибутах строки сообщений, отправляемые и выводимые пользователю. Является основным языком программы, хранящий данные о всех возможных сообщениях на английском языке. Остальные языки должны наследоваться от этой структуры, для последующего переназначения атрибутов. Структуру класса можно увидеть на рисунке 37, а описание класса в таблице 11.



Рисунок 37 - Структура класса EnglishLanguage

**Таблица 11 - Описание класса EnglishLanguage**

|  |  |
| --- | --- |
| Название | Описание |
| Переменная SavingDatabases | Строка “Saving databases…” |
| Переменная OK | Строка “<OK>\r\n” |
| Переменная PermissionDenied | Строка “Permission denied\r\n” |
| Множество других переменных | В этой структуре находятся все возможные выводимые сообщения на английском языке. Каждое такое сообщение щаносится в отдельный атрибут класса. |

1. Класс RussianLanguage

Этот класс унаследован от EnglishLanguage и имеет аналогичные атрибуты с сообщениями для пользователей. Отличие ее в том, что все сообщения написаны на русском языке. Структуру класса можно увидеть на рисунке 38, а описание класса в таблице 12.

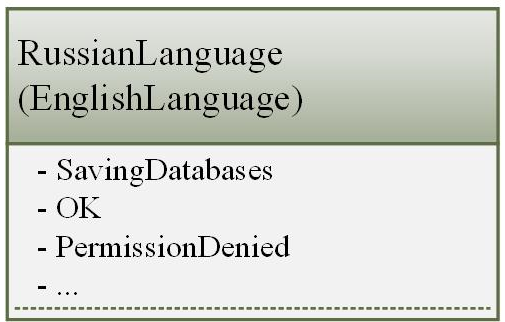


Рисунок 38 - Структура класса RussianLanguage

**Таблица 12 – Описание класса RussianLanguage**

|  |  |
| --- | --- |
| Название | Описание |
| Переменная SavingDatabases | Строка “Сохранение баз данных” |
| Переменная OK | Строка “<OK>\r\n” |
| Переменная PermissionDenied | Строка “Доступ запрещен\r\n” |
| Множество других переменных | В этой структуре находятся все возможные выводимые сообщения на русском языке. Каждое такое сообщение щаносится в отдельный атрибут класса. |

1. Класс Inf

Этот класс нужен для хранения фиксированных значений обработки сообщений базой данных, располагается в commands.py. Используется при передаче информации визуальному клиенту по управлению базами данных. Структуру класса можно увидеть на рисунке 39, а описание класса в таблице 13.

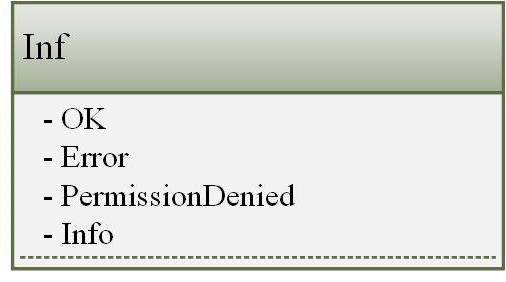


Рисунок 39 - Класс Inf

**Таблица 13 - Описание класса Inf**

|  |  |
| --- | --- |
| Название | Описание |
| Переменная OK | Строка “<OK>”, показывающая, что полученная команда была корректно выполнена |
| Переменная Error | Строка “<ERROR>”, показывающая, что при выполнении команды произошла ошибка |
| Переменная PermissionDenied | Строка “<PERMISSION DENIED>”, показывающая, что для выполнения команды недостаточно прав |
| Переменная Info | Строка “<INFO>”, показывающая какую-то информацию, при выполнении команды |

1. Структуры и классы на визуальном клиенте

Визуальный клиент представляет из себя менеджер по управлению базами данных с помощью визуальных средств, а также позволяет анализировать данные и выводить их не требуя набивать эти команды пользователю с помощью клавиатуры. Некоторые классы представляют из себя хранение процедур и функций по группам, как, например, статический класс Database, имеющий в себе исключительно статические объекты.

1. Класс Database

Данный класс служит для соединения с базой данных и хранения ее подключения. Имеет в себе множество функций, которые использованы для получения и отправки данных. Объект этого класса нельзя создать, так как он статический, как и все его «внутренности», есть лишь несколько переменных, служащий для хранения текущего соединения. Функции, которые позволяют получать данные не от текущего соединения нужны там, где создается новое соединение в базу данных для получения данных не мешая основному подключению. Структура этого класса представлена на рисунке 40, а описание находится в таблице 14.

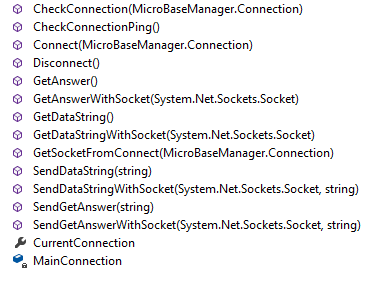


Рисунок 40 - Структура класса Database

**Таблица 14 - Описание класса Database**

|  |  |
| --- | --- |
| Название | Описание |
| CheckConnection | Проверка возможности соединения с сервером. |
| CheckConnectionPing | Получения задержки подключения. |
| Connect | Подключение к серверу и сохранение соединение в CurrentConnection. |
| Disconnect | Отключение от сервера. |
| GetAnswer | Получение ответа в виде объекта Answer от сервера текущего соединения. |
| GetAnswerWithSocket | Получение ответа в виде объекта Answer от сервера другого подключения. |
| GetDataString | Получение ответа от сервера в виде строки текущего подключения. |
| GetDataStringWithSocket | Получение ответа от сервера в виде строки другого подключения. |
| GetSocketFromConnect | Получить сокет подключения, используя объект Connection |
| SendDataString | Отправка данных серверу в виде строки для текущего подключения. |
| SendDataStringWithSocket | Отправка данных серверу в виде строки для другого подключения. |
| SendGetAnswer | Отправка и получение сгруппированы в одном месте, служит для удобства, так как сервер получая данные, сразу отправляет ответ. Возвращает объект Answer. Использует текущее соединение. |
| SendGetAnswerWithSocket | Аналогична функции SendGetAnswerString, но может использовать любое другое подключение. |
| CurrentConnection | Переменная для хранения текущего соединения в виде сокета |
| MainConnection | Переменная для хранения текущего соединения Connection |

1. Класс Connection

Данный класс служит для создания объектов соединений и хранит в себе информацию о подключении, такие как IP и порт. Структура этого класса представлена на рисунке 41, а описание находится в таблице 15.

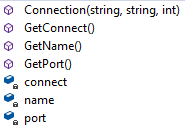


Рисунок 41 - Структура класса Connection

**Таблица 15 - Описание класса Connection**

|  |  |
| --- | --- |
| Название | Описание |
| Connection | Конструктор класса, получающий на вход имя подключения, IP и порт |
| GetConnection | Возвращает IP |
| GetName | Возвращает имя подключения |
| GetPort | Возвращает число – порт |
| connect | Переменная для хранения IP |
| name | Переменная для хранения имени подключения |
| port | Переменная для хранения порта |

1. Класс Answer

Так как реализована работа в СУБД с визуальным клиентом, то есть представление однотипных сообщений, которые можно обрабатывать в едином месте и получать на выход лишь нужную информацию. Информация от сервера может представлять из себя как строку, так и множество данных, поэтому есть реализация как для получения больших данных, так и единого сообщения, но вся полученная информация хранится в виде массива. Структура этого класса представлена на рисунке 42, а описание находится в таблице 16.

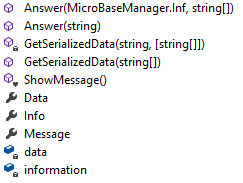


Рисунок 42 - Структура класса Answer

**Таблица 16 - Описание класса Answer**

|  |  |
| --- | --- |
| Название | Описание |
| Answer (Inf, string) | Конструктор класса, получающий на вход информацию ответа от сервера и массив данных. |
| Answer (string) | Конструктор класса, получающий на вход данные от сервера без обработки. Данные обрабатываются и сохраняются в переменные, такие как data – хранится массив строк и information – информация об ответе. |
| GetSerializedData (string, string[]) | Рекурсивная приватная функция для обработки данных и возвращающая их в публичную с аналогичным именем, которая в свою очередь отправляет объект AnswerData пользователю. На вход получает данные, которые нужно обработать и их формат. |
| GetSerializedData (string[]) | Получение объекта AnswerData, служащий для удобного получения данных. На вход получает формат ответа от сервера. |
| ShowMessage | Показать сообщение, полученное от сервера. |
| Data | Свойство, возвращающее массив строк – ответ от сервера |
| Info | Свойство, возвращающее информацию об обработке команды. |
| Message | Свойство аналогичное Data, но возвращает первый элемент массива. |
| data | Переменная для хранения ссылки на массив строк данных, полученных от сервера. |
| information | Переменная для хранения информации о выполнении команды |

1. Класс AnswerData

Чтобы обработку данных, получаемы от сервера объединить в одно место используется класс AnswerData, который хранит значения в указанном формате. Сам объект формируется в классе Answer в функции GetSerializedData. С помощью данного класса возможно получать объекта из глубины, используя обращения, как к массиву используя квадратные скобки. Сами объекты могут быть вложены друг в друга, но возвращают в итоге строки либо массив строк Структура этого класса представлена на рисунке 43, а описание находится в таблице 17.

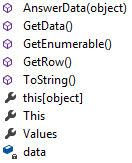


Рисунок 43 - Структура класса AnswerData

**Таблица 17 - Описание класса AnswerData**

|  |  |
| --- | --- |
| Название | Описание |
| AnswerData | Конструктор класса, получающий на вход любой объект, но, так как, создать объект возможно исключительно из класса Answer, то возможно передачи строки, AnswerData, массива и объекта Dictionary. |
| GetData | Возвращает значение data |
| GetEnumerable | Создает генератор из объекта data. Так как data может представлять собой не только строку, а массив или Dictionary, то возвращаются либо строки массива, либо ключи для словаря. Генератор – это такой объект, представляющий из себя последовательную генерацию других объектов. Можно представить как динамический массив, где массив создается только когда его вызывают, поэтому его удобно использовать в конструкции foreach. |
| GetRow | Похож на GetEnumerable, но возвращает не строку, а объект AnswerData. |
| ToString() | Метод, возвращающий строковое представление объекта data |
| this[] | Свойство, позволяющее обращаться к объекту, используя квадратные скобки. Возвращает объекты, хранящиеся в data, в зависимости от того, что из себя представляет data – словарь или массив. Ежели это строка или AnswerData, то не обращается к этому объекту с параметром, а возвращает сам объект AnswerData. Это служит для того, чтобы данные вернулись, независимо от того возможно это или нет. |
| This | Свойство, возвращает данные, аналогичные методу ToString |
| Values | Свойство, возвращает массив строк используя GetEnumerable. |
| data | Переменная, хранящая данные, которые из себя представляют либо сам объект AnswerData, либо строку, либо словарь Dictionary, где ключом является строка, а значение – объект AnswerData, либо массив объектов AnswerData |

1. Перечисление Inf

Данное перечисление ничто иное, как аналогичное представление информации об успешности выполнения команды, представленная в структурах СУБД. Структура этого перечисоения представлена на рисунке 44, а описание находится в таблице 18.

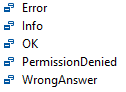


Рисунок 44 - Структура перечисления Inf

**Таблица 18 - Описание перечисления Inf**

|  |  |
| --- | --- |
| Название | Описание |
| Error | Сервер вернул ошибку при выполнении команды |
| Info | Сервер вернул, что команда выполнена и представляет из себя данные |
| OK | Сервер вернул, что команда выполнена успешно |
| PermissionDenied | Сервер вернул, что у пользователя, под которым вы находитесь, не имеет возможности выполнить эту команду |
| WrongAnswer | В отличии от структуры в СУБД, есть значение WrongAnswer, означающая, что ответ от сервера был неожидаемый. |

1. Класс User

СУБД разграничение прав, используя пользователей, так что пользователей также желательно хранить в одном месте, как и их обработку. Данный класс служит как для хранения текущего для соединения пользователя, так и для других функций и форм, использующих пользователей. Информацию о пользователе возвращает сервер. Структура этого класса представлена на рисунке 45, а описание находится в таблице 19.

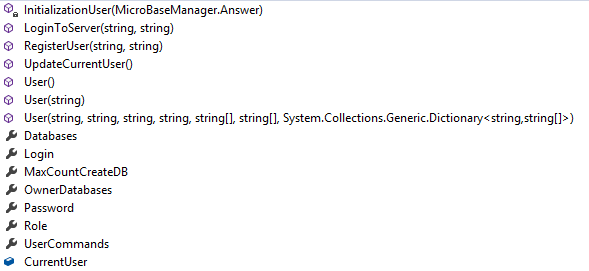


Рисунок 45 - Структура класса User

**Таблица 19 - Описание класса User**

|  |  |
| --- | --- |
| Название | Описание |
| InitializationUser | Инициализирует пользователя с использованием ответа от сервера, то есть объекта Answer. Так как метод приватный, то используется внутри класса другими функциями. |
| LoginToServer | Получает на вход логин и пароль и устанавливает нового текущего пользователя подключения. |
| RegisterUser | Регистрирует нового пользователя на сервере, передавая логин и пароль. |
| UpdateCurrentUser | Обновляет текущего пользователя подключения. |
| User | Конструктор, создающий информацию о текущем пользователе, использующего соединение |
| User (string) | Конструктор, получающий на вход логин пользователя и создающий объект этого пользователя. |
| User (string, string…) | Конструктор, использующий все поля и инициализирующий их. |
| Databases | Словарь, для хранения информации о базе данных(ключ) и доступе к ней(значение). |
| Login | Имя пользователя. |
| MaxCountCreateDB | Число максимально возможного создания количества баз данных этим пользователем. |
| OwnerDatabases | Базы данных, созданные этим пользователем |
| Password | Пароль для текущего соединения хранится в открытом виде, а для других пользователей в поле хранится закодированный пароль в SHA256 |
| Role | Роль пользователя |
| UserCommands | Список команд, возможных для пользователя |
| CurrentUser | Статическая переменная, хранящая текущего пользователя соединения |

1. Класс DataBaseWork

Данный класс служит для хранения информации о базе данных, а также ее управлению. Используются объекты базы данных при их управлении из визуального клиента. Структура этого класса представлена на рисунке 46, а описание находится в таблице 20.

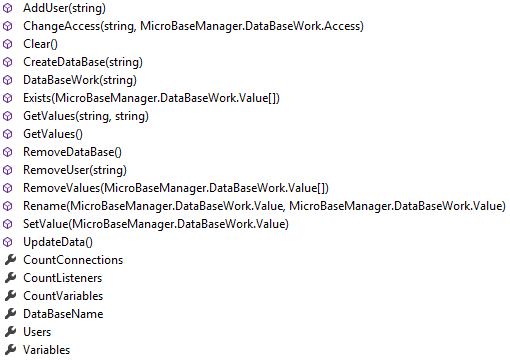


Рисунок 46 - Структура класса DataBaseWork

**Таблица 20 - Описание класса DataBaseWork**

|  |  |
| --- | --- |
| Название | Описание |
| AddUser | Добавить пользователя. |
| ChangeAccess | Изменить доступ к базе данных для пользователя. |
| Clear | Очистить базу данных. |
| CreateDataBase | Единственный статический метод. Создает базу данных и возвращает ее объект. |
| DataBaseWork | Конструктор, получающий на вход имя базы данных. |
| Exists | Проверяет на наличие переменных. |
| GetValues(string,string) | Получить переменные из базы данных. Использует параметры, такие как сортировка (по возрастанию, по убыванию или не сортировать) и фильтр. В фильтре можно указать «\*» – любое количество символов и «?» – любой одиночный символ. |
| GetValues | Получить переменные из базы данных. |
| RemoveDataBase | Удалить базу данных. |
| RemoveUser | Удалить пользователя из базы данных. |
| RemoveValues | Удалить переменные из базы данных. |
| Rename | Переименовать переменную. |
| SetValue | Установить переменную в базу данных. |
| UpdateData | Обновить информацию о базе данных. |
| CountConnections | Количество соединений в базу данных. |
| CountListeners | Количество слушателей. |
| CountVariables | Количество переменных в базе данных. |
| DataBaseName | Имя базы данных. |
| Users | Словарь пользователей базы данных с именем – ключом и доступом(объект Access) – значение. |
| Variables | Массив объектов Value, который является переменными в базе данных. |

1. Класс Value

Для хранения переменных в базе данных могут быть использованы не еденичные значения, а несколько. Чтобы была возоможность удобно ими управлять был создан класс Value. Структура этого класса представлена на рисунке 47, а описание находится в таблице 21.

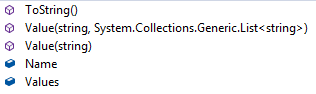


Рисунок 47 - Структура класса Value

**Таблица 21 - Описание класса Value**

|  |  |
| --- | --- |
| Название | Описание |
| ToString | Переопределенный метод, для того чтобы была возможность выводить значения в таблицу, которая при показе данных пользователю использует именно этот метод. |
| Value (string, List <string>) | Конструктор, получающий на вход и инициализирующий имя переменной и значения. |
| Value (string) | Конструктор, который инициализирует только имя переменной. |
| Name | Имя переменной. |
| Values | Список значений переменной. |

1. Класс Access

Аналогично, как в СУБД создаю структуру для хранения информации о доступах пользователя в базу данных. Структура этого класса представлена на рисунке 48, а описание находится в таблице 22.

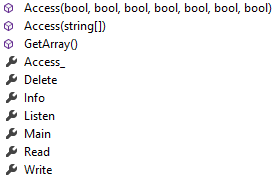


Рисунок 48 - Структура класса Access

**Таблица 22 - Описание класса Access**

|  |  |
| --- | --- |
| Название | Описание |
| Access (bool, bool, …) | Конструктор, инициализирующий все переменный у объекта. |
| Access (string[]) | Конструктор, который получает на вход массив название переменных, которые будут иметь значение |
| GetArray | Получить массив строк – наименований значений, которые хранят true. |
| Access\_ | Возможность доступа в базу данных. |
| Delete | Возможность удалять значения из базы данных. |
| Info | Возможность использовать команды по доступу к информации о базе данных. |
| Listen | Возможность прослушивания базы данных. |
| Main | Возможность администрировать базу данных. |
| Read | Возможность получать данные из базы данных. |
| Write | Возможность записи значений в базу данных. |

1. Класс ImportExportData

Название класса говорит само за себя, с помощью него осуществляется импорт и экспорт подключений и баз данных. Класс и все его методы статические, поэтому его объекты создать невозможно. Структура этого класса представлена на рисунке 49, а описание находится в таблице 23.

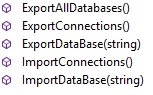


Рисунок 49 – Класс ImportExportData

**Таблица 23 - Описание класса ImportExportData**

|  |  |
| --- | --- |
| Название | Описание |
| ExportAllDatabases | Экспортировать все базы данных в файлы. |
| ExportConnections | Экспортировать подключения в файлы. |
| ExportDataBase | Экспортировать одну базу данных. |
| ImportConnections | Импортировать подключения. |
| ImportDataBase | Импортировать базу данных, указывая в какую базу данных будет происходить запись. |

1. Структура базы данных СУБД

Так как моей задачей уже и является создание программы по управлению базами данных, то тут структуру хранения самой базы данных можно представить в виде ассоциативного массива, где «ключ» – это название переменной, а «значение» – это значение переменной. Значение представляет собой массив значений, ведь в переменной может хранится не одно значение. Каждая такая база данных хранится в памяти, а также сохраняется на внешней памяти, в директорию, по умолчанию которая databases, где имеет расширение .dat.

Структуру базы данных можно представить на  
рисунке 50. Как можно наглядно увидеть, что база данных и есть по сути ассоциативный массив, где ключом является название переменной. Тут можно увидеть такие переменные, как dsad, A, dsakdad, temp, VAL0, VAL13. Каждая переменная содержит значение, описываемое массивом переменных-строк. Вот, например, в переменную dsad записаны такие значения: “2”, “3”, а в переменной temp имеются такие значения: “141”, “67”, “34”, “0”, “69”, “24”, “78”, “58”, “62”, “64”. Хранятся все значения в виде строк, поэтому при обработке значений функциями, работающими с числовыми значениями, используется преобразование строк к числу, с последующим использованием в вычислении, а полученный результат записывается обратно в виде строки.



Рисунок 50 - Представление базы данных

В СУБД присутствует дополнительная база данных, для пользователей, в которой, примерно по такой же структуре, как и в базах данных. То есть используется ассоциативный массив, где «ключ» – это имя учетной записи пользователя, а «значение» – это данные о пользователе, которые хранятся в виде структуры User, описание которой встречалось ранее. По умолчанию база данных хранится в директории users с названием файла USERS.dat.

Структуру хранения пользователей можно представить на рисунке 51. Как видно, некоторый пользователь SomeUser содержит информацию о себе, а конкретно login, password и т.д. Кроме того, переменная DataBases представляет собой ассоциативный массив, который хранит название базы данных и роли пользователя, определяющие, что может делать в ней пользователь.

Структуры были представлены в виде JSON (JavaScript Object Notation) – текстовый обмен данными, основанный на JavaScript, но несмотря на это, он стал универсальным и независимым, и его поддерживают множество других языков программирования. Данные не хранятся в виде JSON, это лишь удобное представление информации. Данные хранятся в виде бинарного кода, ведь это позволяет более компактно хранить их, быстро загружать и сохранять.



Рисунок 51 - Представление базы данных для пользователей

1. Разработка аппаратной части

Исходя из требований к аппаратной части были выбраны следующие комплектующие:

* микроконтроллер NodeMCU (ESP8266);
* датчик BME280.

Выбранный микроконтроллер имеет в своем снаряжении Wi-Fi модуль, благодаря которому микроконтроллер может передавать информацию откуда угодно, главное присутствие питания, которое может быть обеспеченно с помощью портативного аккумулятора, и присутствие в диапазоне Wi-Fi сети. Также имеется возможность использовать это устройство в качестве небольшого сервера, например, HTTP, что позволяет получать с него информацию с помощью веб-обозревателей. Благодаря используемому чипу, а именно ESP8266, можно устанавливать различные прошивки, в которых можно программировать на других языках программирования и использовать различные модули. Микроконтроллеры на базе чипа ESP8266 завоевали огромный успех у большинства людей, благодаря сравнительно дешевой цене и легкостью настройки, что, если вы уже с ними знакомы, сможете их настроить для нужной работы уже через 15 минут.

Сам микроконтроллер можно увидеть на рисунке 52, а его характеристики в таблице 24.



Рисунок 52 - Микроконтроллер NodeMCU (ESP8266)

**Таблица 24 – Характеристики микроконтроллера NodeMCU (ESP8266)**

|  |  |
| --- | --- |
| Наименование характеристики | Значение |
| Частота процессора | 80 МГц |
| Битность процессора | 32-битный |
| ОЗУ | 96 КБ |
| ПЗУ | 4 МБ |
| Wi-Fi | 802.11 b/g/n |
| Внутреннее напряжение | 3,3 В |
| Внешнее питание | 3,6 – 20 В |
| Потребление | 60 – 200 мА |
| Тип подключения | microUSB |
| Количество портов ввода/вывода | 11 |
| Поддерживаемые языки | * C (Arduino) * Lua (NodeMCU) * Python (MicroPython) |

Существуют облачные сервисы, позволяющие собрать вам нужную прошивку из тех модулей, которые вами будут использованы. На одном из таких сервисов, генерирующих прошивку из последней версии исходных кода NodeMCU firmware, можно выбрать модули (см. рисунок 53), где по умолчанию выбраны основные, позволяющие работать с сетью, файлами, Wi-Fi модулем и портами на устройстве. В итоге вы получаете готовую прошивку, с нужными модулями, и сразу можете приступать к программированию на языке Lua.

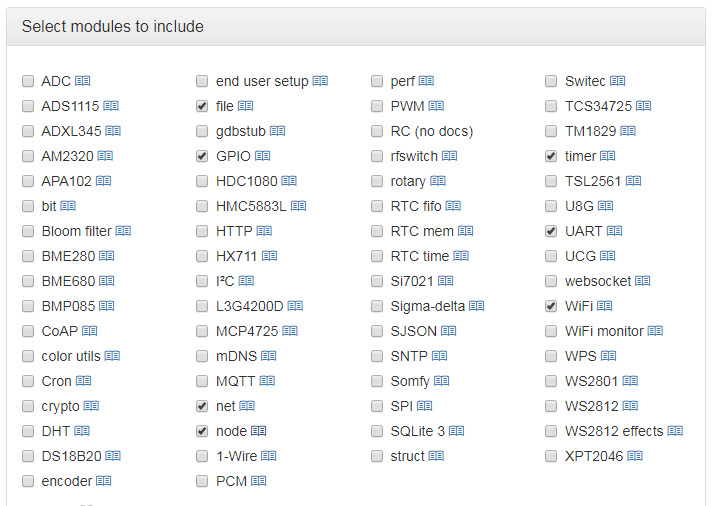


Рисунок 53 - Возможность выбора нужных модулей

Моим выбором оказалась прошивка MicroPython – это программная реализация языка программирования Python 3, написанного на C, которая оптимизирована для работы на микроконтроллере. MicroPython – это компилятор Python и среда выполнения, работающая на аппаратном обеспечении микроконтроллера. Прошивка, имеющая размер чуть больше 1 МБ, включает основные библиотеки Python и модули, дающие программисту доступ к низкоуровневому оборудованию. MicroPython активно разрабатывается сообществом. Существует также урезанная версия MicroPython, имеющая размер около 500 КБ, которая используется в микроконтроллерах, имеющих небольшой объем внешней памяти.

BME280 – это интегрированный датчик окружающей среды, разработанный специально для мобильных устройств и микроконтроллеров, где размер и энергопотребление играют ключевую роль. В устройстве используются датчики высокой точности для измерения давления, влажности и температуру, имеющие быстрое время отклика. Алгоритм устройства основан на считывании калибровочных коэффициентов с датчиков, с последующем преобразованием с помощью формул в метрические единицы. Все эти преобразования реализованы в модуле для BME280.

Датчик BME280 можно увидеть на рисунке 54, а его характеристики доступны в таблице 25.

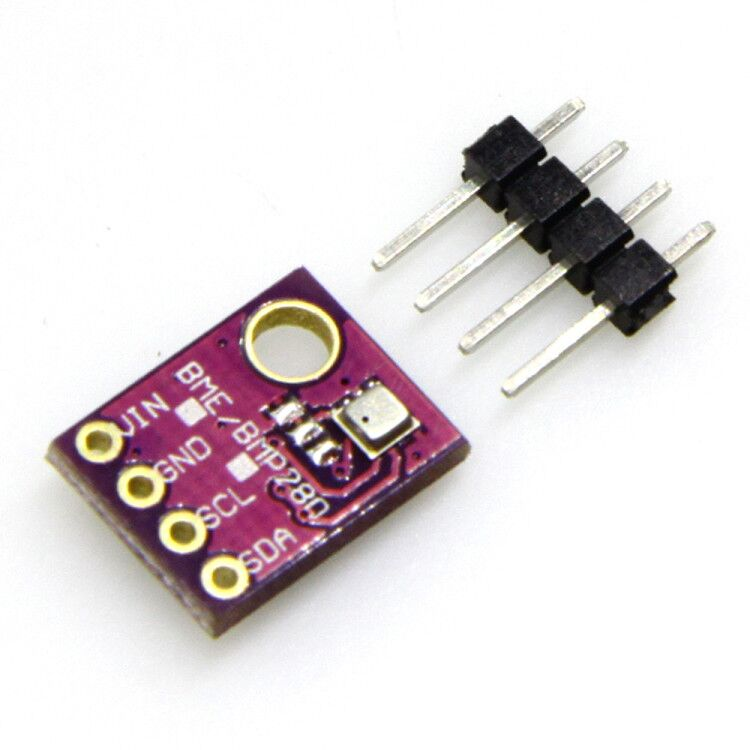


Рисунок 54 - Датчик BME280

**Таблица 25 - Характеристики датчика BME280**

|  |  |
| --- | --- |
| Наименование характеристики | Значение |
| Интерфейс | I²C и SPI |
| Входное напряжение | 1,71 – 3,6 В |
| Потребление тока | 1,8 – 3,6 мкА |
| Рабочий диапазон для температуры | -40 – +85 °C |
| Рабочий диапазон для влажности воздуха | 0 – 100 % |
| Рабочий диапазон для давления | 300 – 1100 гПа |
| Время отклика | 1 с |
| Погрешность для влажности | ±3 % |
| Погрешность для температуры | ±0,5°C |
| Погрешность для давления | ±1,0 гПа |

1. Алгоритм работы устройства

В момент включения устройства происходит соединение с Wi-Fi сетью, при неудачном подключении, будет монотонно светиться красный датчик. Далее происходит соединение с сервером, где с микроконтроллера отсылаются необходимые данные, такие как, логин и пароль, для входа в учетную запись и база данных, в которую будет микроконтроллер отсылать параметры погодных условий. Если произошла ошибка уже в момент подключения к серверу, будет мигать красный сигнал на микроконтроллере.

Когда есть соединение с базой данных, происходит постоянная отправка данных, содержащих информацию о температуре, давлении и влажности воздуха.

1. Сборка устройства

На рисунке 55 можно увидеть схему подключения датчика BME280 к микроконтроллеру:



Рисунок 55 - Схема подключения

Для начала, чтобы датчик передавал температуру на устройство, его необходимо подключить к нужным портам. На BME280 нужно подключить порты SCL и SDA к портам ввода/вывода на NodeMCU с помощью проводов. Порты ввода вывода на NodeMCU это цифра с приставкой D, например, здесь произведено подключение к D2 и D4. Некоторые порты нельзя использовать, так как они системные, как, например, D3 или D1. Остальные уже, 3.3V и GNB, подключаются аналогично названию, как на устройстве.

Подключается устройство с помощью MicroUSB и используется напряжение от 3.6V до 20V, далее уже преобразуется в 3.3V. С помощью MicroUSB идет передача данных: запись прошивки, закачивание программного кода и настройка устройства.

Так как датчик очень чувствителен к изменениям в окружающей среды, его, желательно, использовать подальше от источников помех, поместив его на как можно больше открытое пространство. Устройство автономно, и требуется лишь доступ к Wi-Fi сети и электрической сети, а второе может обеспечить самый обычный аккумулятор или же батарейки.

1. Настройка устройства

При запуске устройства запускается предустановленная прошивка NodeMCU, которую сразу же можно использовать и программировать на языке Lua, но эта прошивка не имеет предустановленного модуля для BME280, к тому же необходимо было познакомиться с языком программирования Lua. На это устройство существует прошивка MicroPython, которая подходит для моих целей прекрасно.

Установка прошивки представлена на рисунке 56.

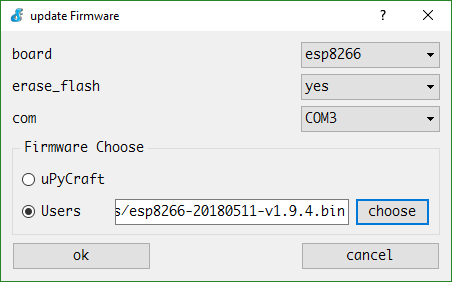


Рисунок 56 - Установка прошивки

После установки, необходимо установить модуль для BME280, который устанавливается посредством закачивания файла bme280.py на устройство.

Чтобы войти в устройство, необходима программа, которая позволяет использовать соединение с COM портом на компьютере, для этого подойдет программ PuTTy, с помощью которой можно подключаться не только к COM, но и по Telnet, SSH, Rlogin. Подключение к микроконтроллеру представлено на рисунке 57.

После подключения, у нас появляется REPL (read-eval-print loop – цикл «чтение-вычисление-вывод») – организация интерактивной среды с помощью интерфейса командной строки. В python приветственная команда для ввода – это «>>>».

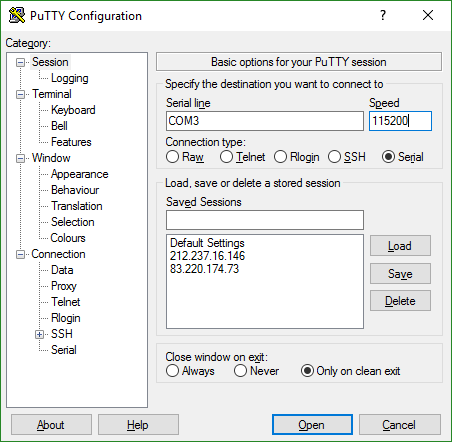


Рисунок 57 - Putty: подключение к NodeMCU

Чтобы проверить работоспособность BME280, необходимо импортировать стандартный модуль machine, в котором находятся инструменты по работе с NodeMCU, а, конкретно, необходимо связаться с датчиком с помощью шины I2C, использующая двунаправленные линии связи SDA и SCL. Далее происходит импорт модуля bme280, в котором находятся инструменты для работы с датчиком BME280. Передаем соединение i2c в конструктор класса BME280 и получаем объект, с которым производится дальнейшая работа для получения данных. Чтобы убедиться, что все работает, необходимо обратится к параметру values, после чего происходит вывод текущих параметров окружающей среды (см. рисунок 58). Первое значение является температурой в градусах Цельсия, второе – это давление, измеряемое в гПа (гектопаскаль), при необходимости можно перевести в миллиметры ртутного столба, так как это международное обозначение единицы измерения для давления при оповещении информации об окружающей среды, а третье – это влажность, измеряемая в процентах.

Для проверки исправности работы датчика можно, например, искусственно нагреть его и увидеть изменения температуры.

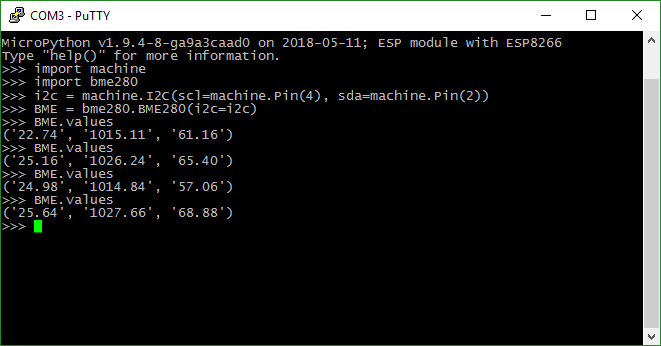


Рисунок 58 - Демонстрация работоспособности датчика BME280

Дальше необходимо написать скрипт, который будет при запуске устройства подключаться к Wi-Fi, инициализировать соединение с сервером и передавать данные о погоде. На микроконтроллере есть в наличии 2 светодиода (красный и синий), с помощью них можно показывать ошибки при работе устройства, например, если нет подключения к Wi-FI, показывать красный сигнал, при невозможности найти датчик BME280, показывать мигающий каждые пол секунды синий сигнал, а при невозможности подключения к серверу мигающий каждые пол секунды красный сигнал. Также можно показывать успешную передачу данных, с помощью синего сигнала, который будет на мгновение включаться при успешной передаче.

Чтобы программный код запускался сразу после загрузки устройства, необходимо, чтобы файл имел расширение main.py, а также бесконечный цикл (см. рисунок 59), где происходит передача данных.

Пример получения данных можно увидеть на рисунке 60, а программный код, используемый в микроконтроллере, с комментариями можно увидеть в приложении А3.1.

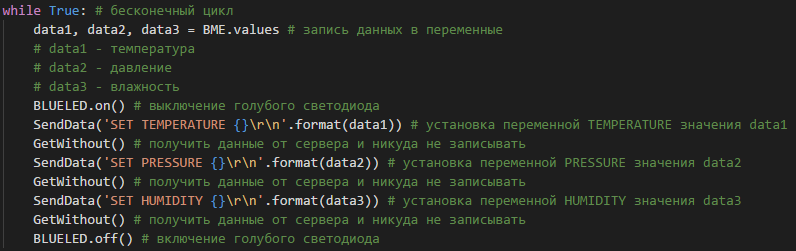


Рисунок 59 - Используемый бесконечный цикл

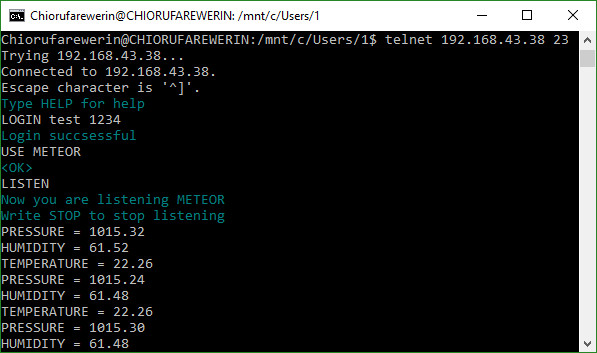


Рисунок 60 - Процесс получения данных

1. Алгоритм работы программы

В этой главе описан принцип работы программы, как взаимодействуют между собой компоненты, а также разбор модулей.

1. Общий алгоритм работы программы

На рисунке 61 или крупном аналоге Б.6 продемонстрирована общая модель работы всех приложений, которые в свою очередь поделены на модули, выполняющие свои функции.

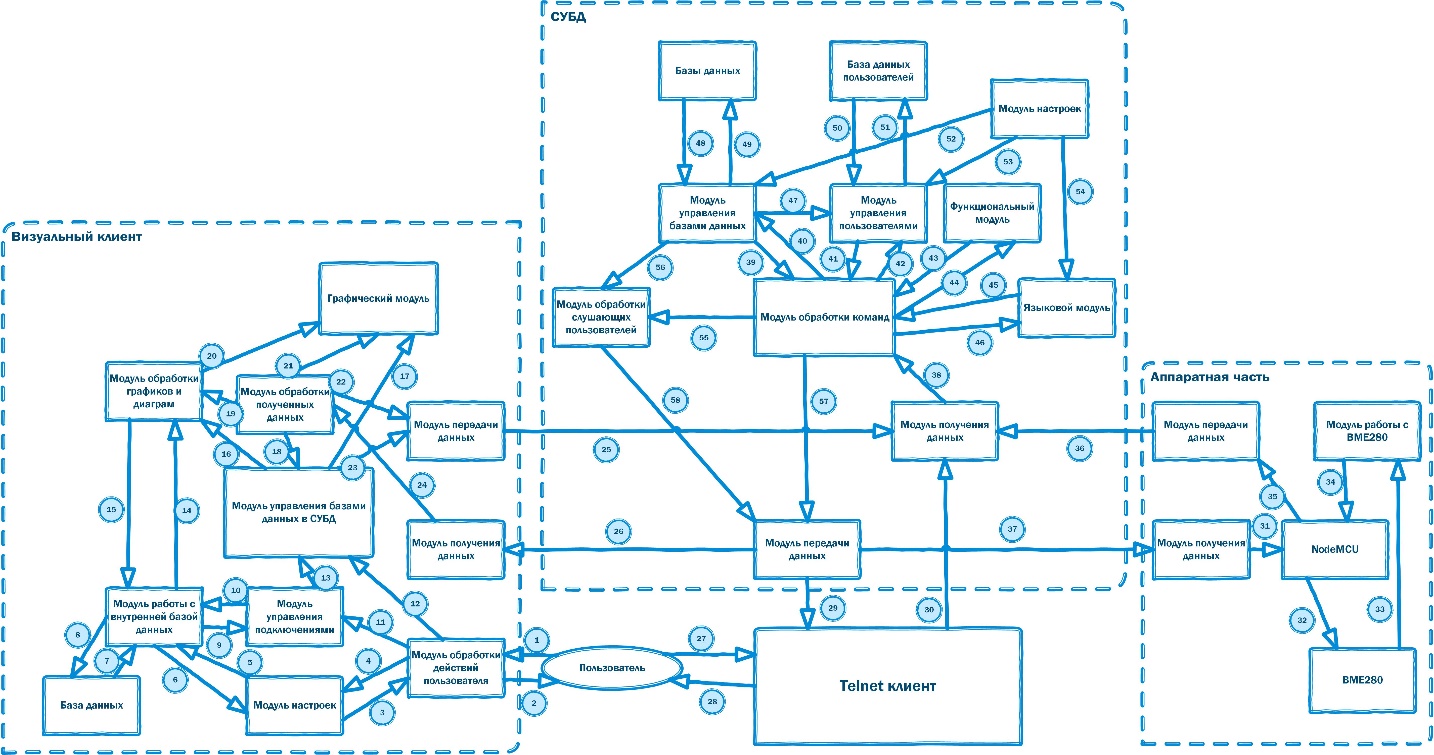


Рисунок 61 - Общая модель работы приложения

Модули взаимодействуют друг с другом, передавая информацию:

1. получение действий и данных от пользователя;
2. передача результирующих данных пользователю;
3. изменения настроек от пользователя;
4. загрузка настроек пользователю;
5. передача настроек в базу данных;
6. загрузка настроек из базы данных;
7. загрузка данных;
8. сохранение данных;
9. загрузка подключений из базы данных;
10. передача подключений в базу данных;
11. передача данных о подключении от пользователя;
12. передача действий и данных от пользователя в режиме управления;
13. информация о подключении;
14. загрузка сохранённых графиков;
15. сохранение графиков;
16. передача данных;
17. информация и данные из баз данных;
18. изменение информации и данных из баз данных;
19. некоторая информация из баз данных;
20. графические объекты;
21. информация о выполнении команды;
22. дополнительные данные;
23. изменение и данных в базах данных от пользователя;
24. полученная информация из СУБД;
25. передача данных пользователю, управляющего визуальным клиентом;
26. получение данных от пользователя, управляющего визуальным клиентом;
27. получение команд от пользователя;
28. передача результирующих данных пользователю;
29. передача данных пользователю, использующего Telnet соединение;
30. получение данных от пользователя, использующего Telnet соединение;
31. сообщения о результатах выполнения команд;
32. поддержка подключения с BME280;
33. сырые данных из BME280;
34. готовые данные из BME280;
35. считанные данные и информация;
36. получение данных от микроконтроллера;
37. передача сообщений микроконтроллеру;
38. полученные данные;
39. получение данных и информации из базы данных;
40. передача данных и информации в базу данных;
41. получение данных и информации о пользователях;
42. передача данных и информации о пользователях;
43. результат обработки функции;
44. данные и аргументы;
45. получение данных на нужном языке;
46. установка языка;
47. информация о базах данных, устанавливаемая пользователям (владельцы, доступ);
48. данные из базы данных;
49. изменение данных в базе данных;
50. получение информации о пользователях;
51. изменение информации о пользователей;
52. информация о хранении баз данных;
53. информация о хранении пользователей;
54. информация о используемых языках по умолчанию;
55. статус прослушивания пользователя и информация об изменениях в СУБД;
56. данные и изменения в базах данных;
57. сообщения и обработанные данные;
58. данные для слушающих пользователей;

Так как это система по управлению базами данных, где используется множество команд, а также единовременно могут использовать базу данных неопределенно количество пользователей, отобразить все связи, то, как будут передаваться данные, очень сложно, в связи с этим будут продемонстрированы основные возможные действия.

1. Инициализация программы

На рисунке 62 показаны вызовы модулей и функций, инициализирующих СУБД. В функции work создаются отдельные потоки для каждого нового подключения, поэтому она работает постоянно. Происходят вызовы инициализации программы, для загрузки данных в память: баз данных и пользователей, проверки необходимых директорий и переданных параметров в базу данных, такие как установка IP – адреса, порта или языка.

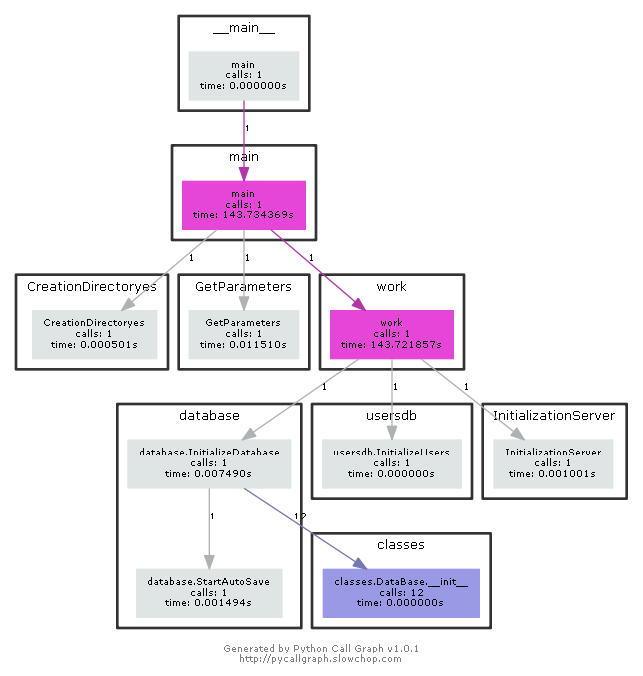


Рисунок 62 - Инициализация СУБД

1. Подключение пользователя по Telnet

На рисунке 63 или крупном аналоге Б.7 продемонстрирована диаграмма вызовов для пользователя, который подключился, зарегистрировался, зашел под своим пользователем, создал базу данных, добавил переменную, получил ее значение и вышел.

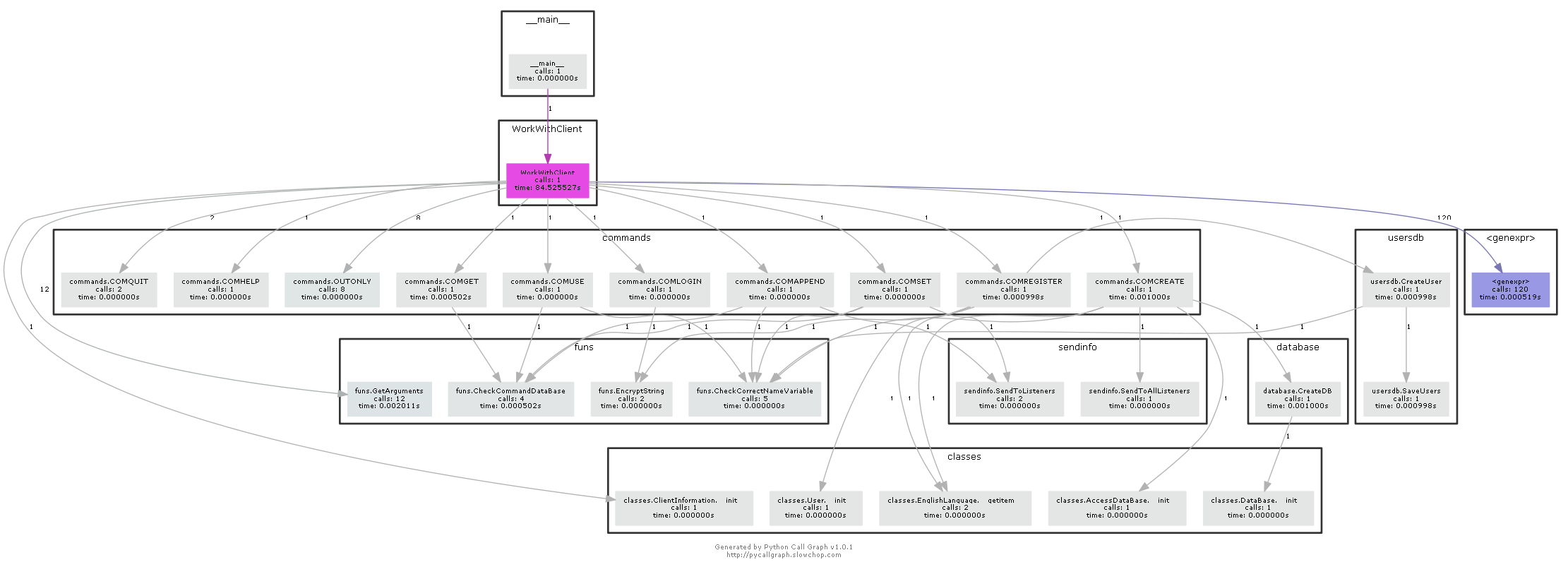


Рисунок 63 - Диаграмма вызовов для пользователя Telnet

Исходя из этой диаграммы, можно видеть, что основная работа с пользователем происходит в функции WorkWithClient, из которой уже идет обращение к модулю commands, где идет обработка нужной команды. При создании пользователя идет обращение к модулю usersdb, при создании базы данных идет обращение к модулю database, к тому же, используются модули funs (вспомогательные функции), sendinfo (отправка данных слушателям) и classes (классы).

1. Подключение пользователя с помощью визуального клиента

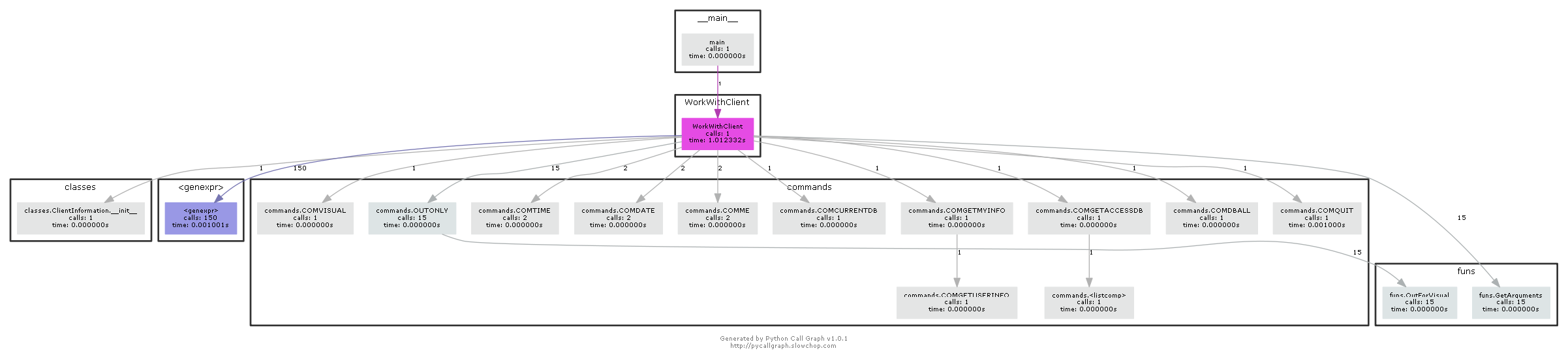


Рисунок 64 - Диаграмма вызовов пользователя визуального клиента

На рисунке 64 или крупном аналоге Б.8 представлена диаграмма вызовов для пользователя использующего визуальный клиент, здесь уже возложена задача отправлять команды на клиент, и, как можно видеть на диаграмме, подключение представляет из себя обращение к множеству функций для получения каких-либо данных с целью отображения их пользователю. Было произведено только подключение и уже задействовано большое количество функций, ведь первое окно пользователю демонстрирует информацию о базе данных.

1. Подключение микроконтроллера

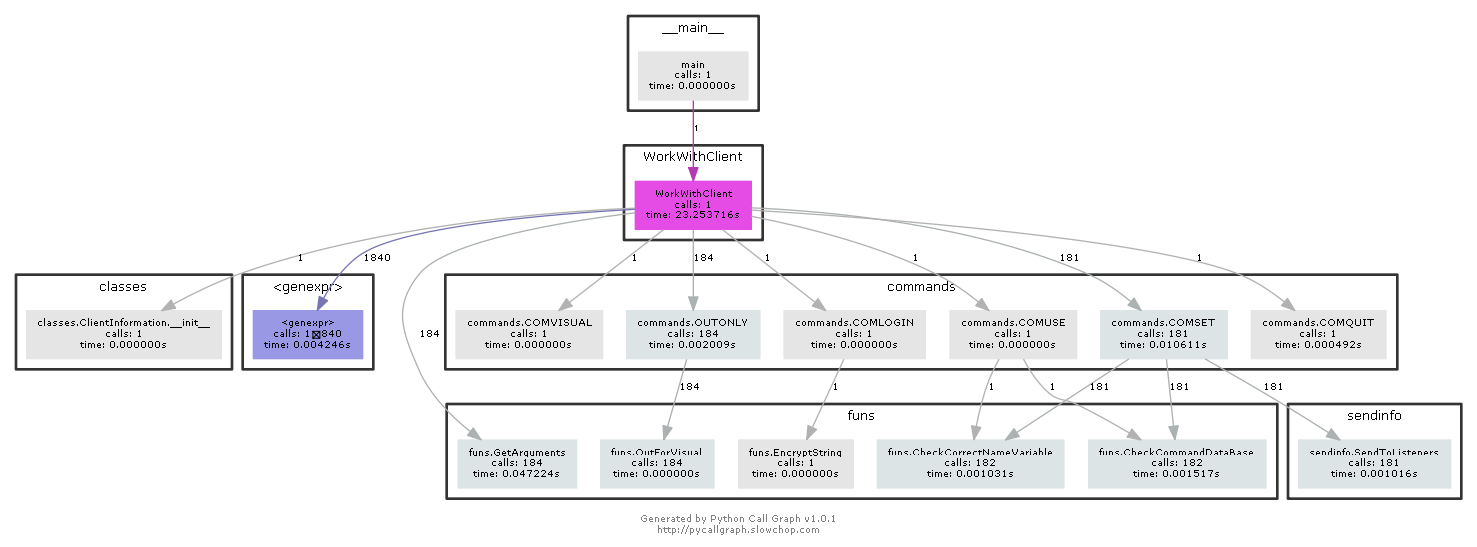


Рисунок 65 - Диаграмма вызовов микроконтроллера

На рисунке 65 представлена диаграмма вызовов микроконтроллера, передающего информацию об окружающей среде. Для удобства обработки команд, после подключения сразу ставится статус визуального клиента, ведь тогда отправляется информация только лишь о том, выполнена ли команда или произошла ошибка. Происходит вход в систему и подключение к заранее созданной базе данных, куда уже устанавливаются нужные значения. Для микроконтроллера, который настроен именно на такую работу, эта диаграмма вызовов будет иметь только такой вид.

1. Алгоритмы отдельных модулей

Для каждой группы методов и данных используются отдельные файлы:

1. файлы сервера:
   1. main.py;
   2. database.py;
   3. usersdb.py;
   4. commands.py;
   5. funs.py;
   6. sendinfo.py;
   7. language.py;
   8. classes.py;
   9. access.py;
   10. variables.py.
2. файлы клиента:
   * 1. Answer.cs;
     2. Connection.cs;
     3. Database.cs;
     4. DataBaseWork.cs;
     5. InfoClass.cs;
     6. Language.cs;
     7. Settings.cs;
     8. User.cs;
     9. UserInfo.cs.
3. файлы микроконтроллера:
   * + 1. main.py.
4. Алгоритм инициализации СУБД

Перед началом работы базы данных, необходимо проверить корректность настроек пользователя, установленных в variables.py. В этом файле устанавливаются некоторые данные, которые будут постоянно использоваться: директории, куда будут сохранятся данные, языки по умолчанию, как для сервера, так и для подключаемых клиентов, настройки ролей и установка некоторых переменных, таких как, необходимость записи логов, возможность регистрации и максимальное количество создания баз данных для пользователя.

Как происходит инициализация:

1. Проверяется наличие директорий по данным, указанных в настройках, а также попытка создать этих директорий. Также создается файл для логирования, если в настройках указана необходимость.
2. Проверяется возможность установки языком по умолчанию сервера использовать язык системы.
3. Получение переданных аргументов при запуске сервера, где может быть передан IP-адрес, порт и язык.
4. Происходит инициализация баз данных: проверяется корректность базы данных, заносятся в память и запускается автоматическое сохранение через определенный интервал, указанный в настройках. Если базу данных корректно загрузить не удается, то об этом сообщается пользователю и предлагается ее пересоздать.
5. Происходит инициализация пользователь, так как для них существует отдельная база данных. Также создаются стандартные учетные записи: «Гость» и «Администратор».
6. Сервер устанавливается на IP-адрес и порт.
7. Ожидание соединений и при подключении, отправлять клиента в другой поток.
8. Алгоритм работы с клиентом

Между сервером и клиентом создается TCP/IP соединение, которое должно поддерживаться постоянно, а при его разрыве пользователь автоматически отключается от сервера. Для передачи данных на сервере используются строки с переносом строки, то есть, конец строки означает конец переданного выражения.

Для связи с клиентом используется основная функция WorkWithClient в модуле main.py.

Связь с клиентом (WorkWithClient)

Назначение: инициализирует клиента и поддерживает передачу данных как от него, так и для него, а также происходят проверки с переданными данными с последующей передачей команды на выполнение. Схему этой функции можно увидеть на рисунке 66.



Рисунок 66 - Схема представления функции WorkWithClient

1. Программная реализация
2. Пользовательский интерфейс

В данном разделе рассмотрены средства, с помощью которых пользователь может взаимодействовать с программой.

1. СУБД

СУБД представляет собой сервер, поэтому взаимодействие происходит через консоль. Возможно указать параметры, при старте сервера, а также получить справку по ним (см. рисунок 67).

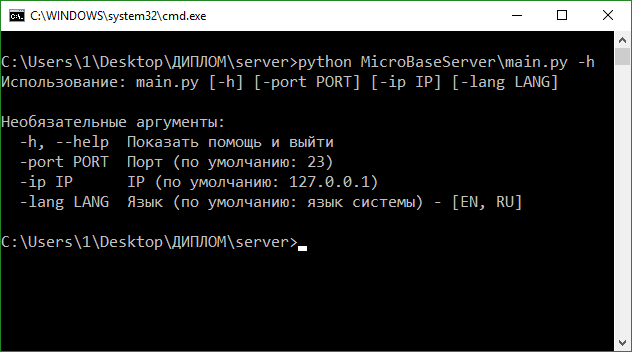


Рисунок 67 - Взаимодействие с СУБД (параметр -h – вызов справки)

Также возможно остановить работу сервера, используя сочетание клавиш CTRL + C, как показано на рисунке 68.

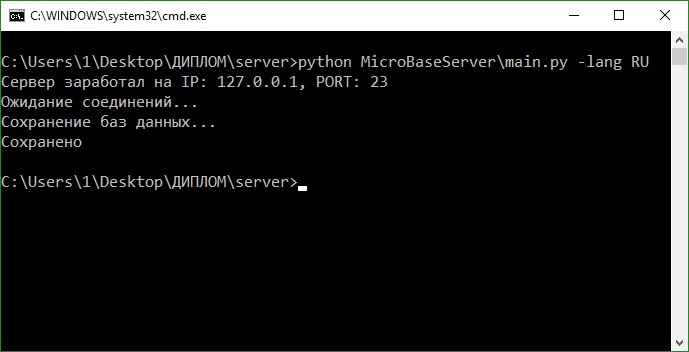


Рисунок 68 - Остановка сервера сочетанием клавиш CTRL + C

У сервера есть проверка на целостность баз данных, так что при отсутствии какой-либо из них, будет задан вопрос, с целью получить Y или N (см. рисунок 69).

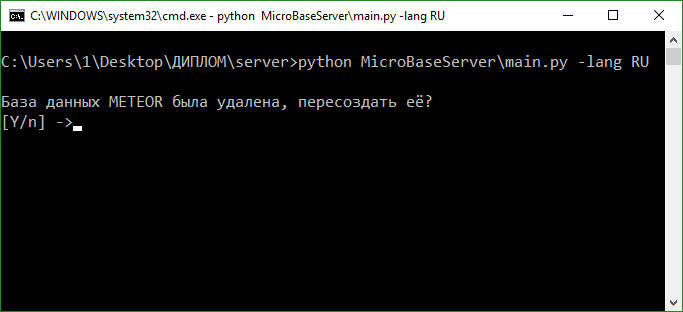


Рисунок 69 - Вопрос пользователю

1. Telnet подключение

Telnet подключение – это двунаправленная передача текстовых данных с использованием сетевого протокола. Интерфейс зависит от того, какой Telnet клиент вы используете, но основной вид – это консоль, куда вводятся команды, а после отправки и обработки полученной команды сервером, возвращается текстовый ответ. Пример интерфейса Telnet подключения с использованием оболочки bash (эмулируемая среда Linux в операционной системе Windows) представлен на рисунке 70.

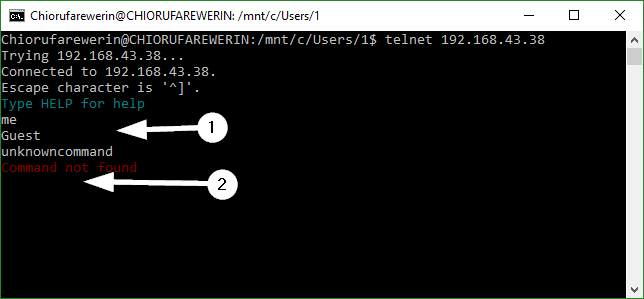


Рисунок 70 - Интерфейс Telnet подключения

Описание интерфейса:

* + - * 1. Получаемый ответ от сервера.
        2. Самая нижняя строчка – ввод команд пользователем.

1. Визуальный клиент

Интерфейс у визуального клиента в общем состоит из приветственной формы, когда необходимо добавить новое подключение, а также выбрать подключение для дальнейшей работы с ним, а также формы работы с сервером, где осуществляется взаимодействие с помощью вкладок. Вкладки добавляют удобства, так как не придется постоянно переходить с одной формы на другую.

1. Форма подключений

При открытии программы пользователю предоставляется возможность добавить подключение, введя наименование подключения, IP адрес и порт. Форма представлена на рисунке 71 и состоит из таких элементов интерфейса:

1. Добавленное пользователем подключение. При нажатии на него происходит переход в форму работы с подключением. Также на элементе есть шестеренка, которая позволяет изменить это подключение.
2. Кнопка «Добавить подключение».
3. Полоска меню. Состоит из элементов:
4. Файл – «Импорт подключений» и «Экспорт подключений», позволяющие все подключения сохранить в отдельный файл, так и выгрузить их из него.
5. Настройки – «Основные» открывает пользователю возможность изменить настройки.
6. Справка – «Открыть справку» открывает форму, где есть описание, как работать с программой и «О программе» открывает стандартную форму сведений о программе.
7. Выход – выход из программы

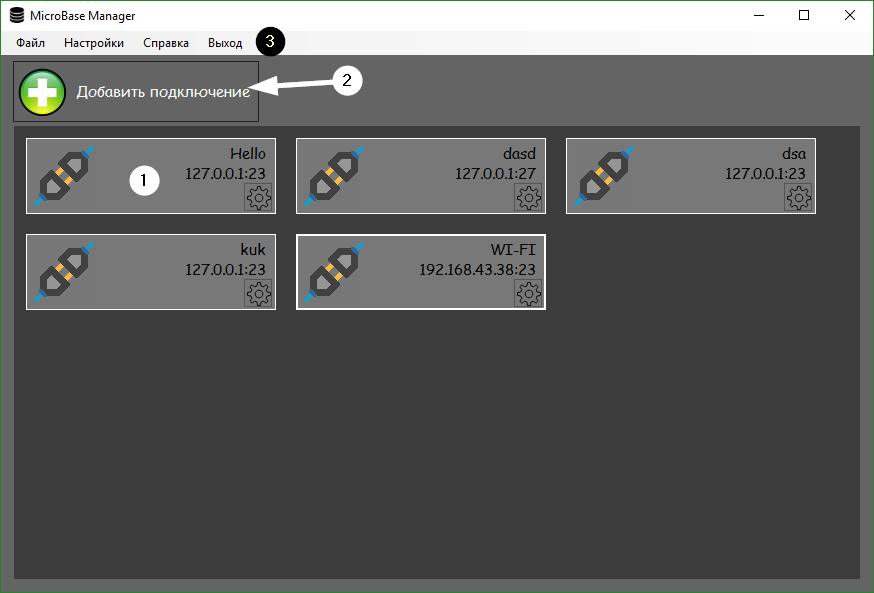


Рисунок 71 - Интерфейс формы подключений

Когда пользователь нажимает на кнопку «Добавить подключение» (2), открывается форма «Добавление подключения», которую можно видеть на рисунке 72.



Рисунок 72 - Интерфейс формы «Добавление подключения»

Описание интерфейса:

1. Поля ввода наименования подключения.
2. Поле ввода адреса подключения.
3. Поле ввода порта подключения.
4. Открытие формы проверки подключения. Открывается форма «Проверка подключения» (см. рисунок 73) с прогресс баром и выводящим сообщение о возможности подключения.
5. Кнопка «ОК», которая добавляет это подключение.
6. Кнопка «Отмена», которая закрывает форму без сохранения подключения.

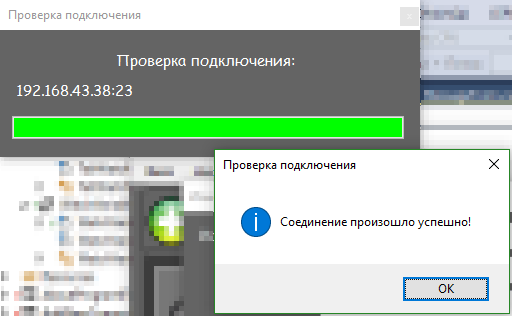


Рисунок 73 - Интерфейс формы «Проверка подключения»

1. Главная форма программы

Когда пользователь подключился, ему открывается главная форма программы, представленная на рисунке 74.

Описание интерфейса:

1. Блок «управление» – позволяет выбрать определенное действие работы с сервером.
2. Блок «Базы данных» – выводит список баз данных, с которыми может работать пользователь. Выделенные более жирным, означает, что пользователь имеет доступ к получению данных из нее.
3. Блок, где открываются вкладки.
4. Полоска меню. Похожа на полоску в окне выбора подключения, за исключением некоторых элементов, которые описаны ниже:
5. Файл – «Открыть менеджер импорта и экспорта баз данных», позволяющего выгрузить данные из базы и загрузить их в базу данных.
6. Обновить – позволяет обновить элементы на форме.

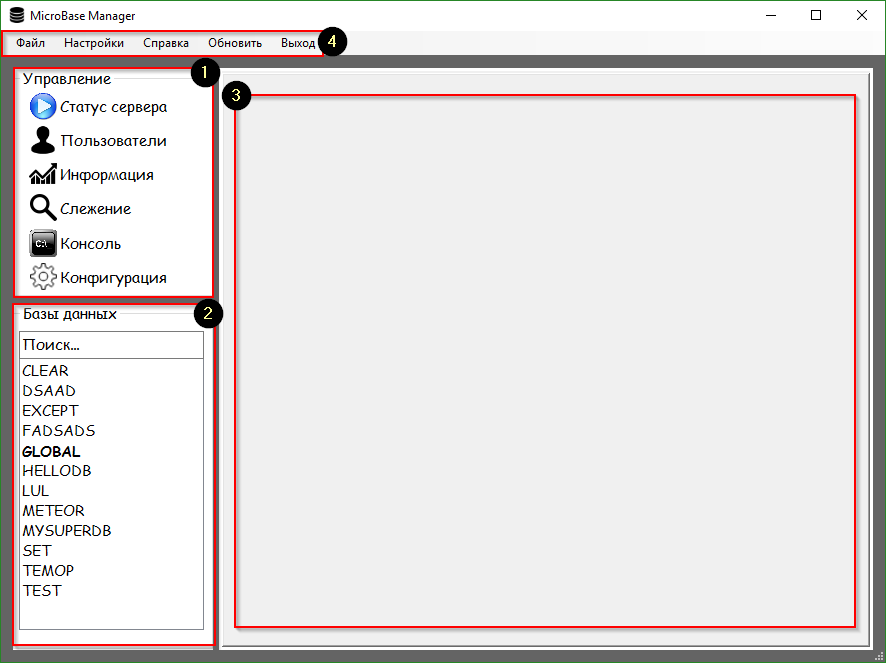


Рисунок 74 - Интерфейс главной формы

Далее будет следовать описание панелей, открываемых в блоке, где открываются вкладки (3).

1. Панель «Статус сервера»

На данной панели выводится информация о подключении и его состоянии. Вид панели представлен на рисунке 75.

Описание интерфейса:

1. Краткие сведения о соединении.
2. Статус сервера, а также задержка между передачей информации.
3. Таблица текущих подключений в базу данных.

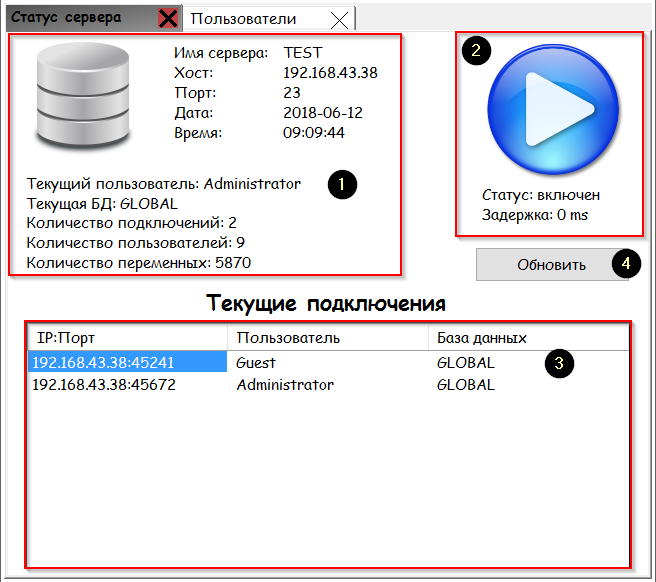


Рисунок 75 - Интерфейс панели «Статус сервера»

1. Панель «Пользователи»

На этой панели представлен интерфейс взаимодействия с пользователями, которая представлена на рисунке 76. Когда вы не вошли на сервер под Администратором, у вас не будет доступа к редактированию пользователей, будут доступны только первые 4 элемента управления.

Описание интерфейса:

1. Текущий пользователь.
2. Открыть форму, которая откроет информацию о текущем пользователе.
3. Откроет форму «Вход в учетную запись», в которую необходимо ввести имя пользователя и пароль (см. рисунок 77).
4. Откроет форму регистрации пользователя, которая похожа на форму «Вход в учетную запись», за исключением того, что добавлено поле, требующее повторного ввода пароля.
5. Таблица со всеми пользователями и их ролями.
6. В блоке отражены логин и роль выбранного пользователя.
7. Список команд, которые может использовать выбранный пользователь.
8. Таблица созданных баз данных, а также максимальное число баз доступных для создания пользователю.
9. Таблица с ролями пользователя в базах данных.
10. Открывает форму «Изменить пользователя» (см. рисунок 78).

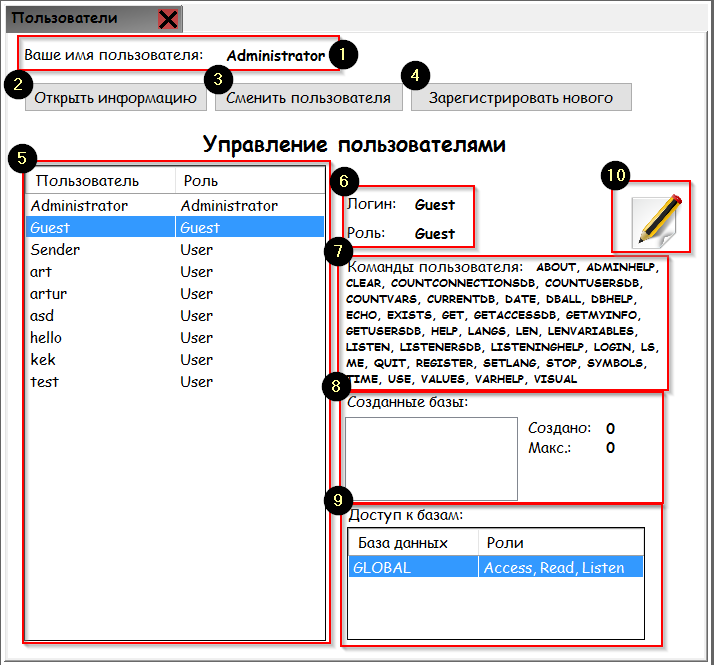


Рисунок 76 - Интерфейс панели «Пользователи»

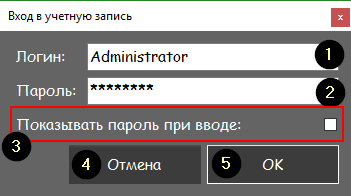


Рисунок 77 - Интерфейс формы «Вход в учетную запись»

Описание интерфейса формы «Вход в учетную запись»:

1. Поле ввода имени пользователя.
2. Поле ввода пароля.
3. Галочка, при включении которой, показывает вводимый пароль.
4. Кнопка «Отмена» – закрывает форму без применения изменений
5. Кнопка «ОК» – при удачном входе в учетную запись, закрывает форму, иначе выводится сообщение об ошибке

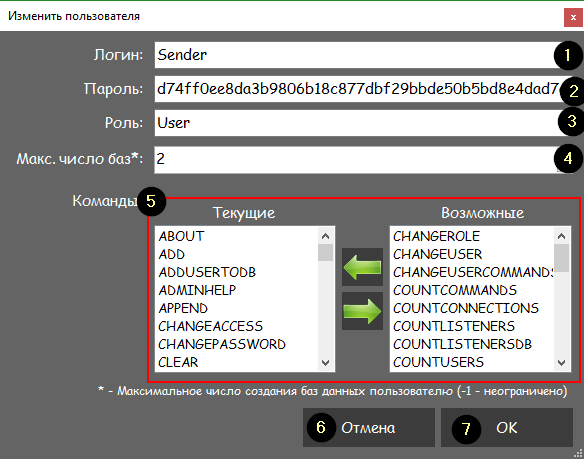


Рисунок 78 - Интерфейс формы Изменить пользователя

Описание интерфейса формы «Изменить пользователя»:

1. Поле, отображающее имя пользователя (при изменении, переименовывает пользователя).
2. Поле пароля пользователя, отображающее пароль закодированный в SHA256. Требуется для проверки, сравниваются в базе данных только закодированные пароли. Если оставить без изменений, то пароль пользователя не будет изменен, иначе будет изменен на новый.
3. Выбор роли пользователя.
4. Указание числа максимального числа баз данных, доступных для создания пользователю.
5. Индивидуальная настройка команд для пользователя. Имеется 2 таблицы: в одной отображаются текущие доступные команды пользователю, а другой – остальные возможные команды. С помощью кнопок, на которых изображены стрелочки можно перемещать команды и давать пользователю больше возможностей управления в СУБД. Команды, которые есть в роли доступны всегда, поэтому их отбирать от пользователя не имеет смысла и будет сообщено об этом.
6. Кнопка «Отмена» – закрывает форму без применения изменений.
7. Кнопка «ОК» – применяет изменения пользователю и закрывает форму. Если произошла ошибка, будет об этом сообщено.
8. Панель «Информация»

В данной панели отражена информация о текущем состоянии на сервере и отражено все в виде графиков (см. рисунок 79).

Описание интерфейса:

1. Дата и время начала работы сервера.
2. Время, которое проработал сервер без выключения.
3. Блок «CPU» – выводит информацию о нагрузке процессора, а также временную шкалу изменений этой информации на машине, где запущен сервер. Информация начинает поступать с момента запуска клиента и входа под тем пользователем, кому доступна эта возможность.
4. Блок «ОЗУ» – Аналогичен блоку «CPU», за исключением, что тут отображается информация об использовании оперативной памяти.
5. Блок «Запросы» – Отображает информацию о количестве поступающих запросов к серверу.

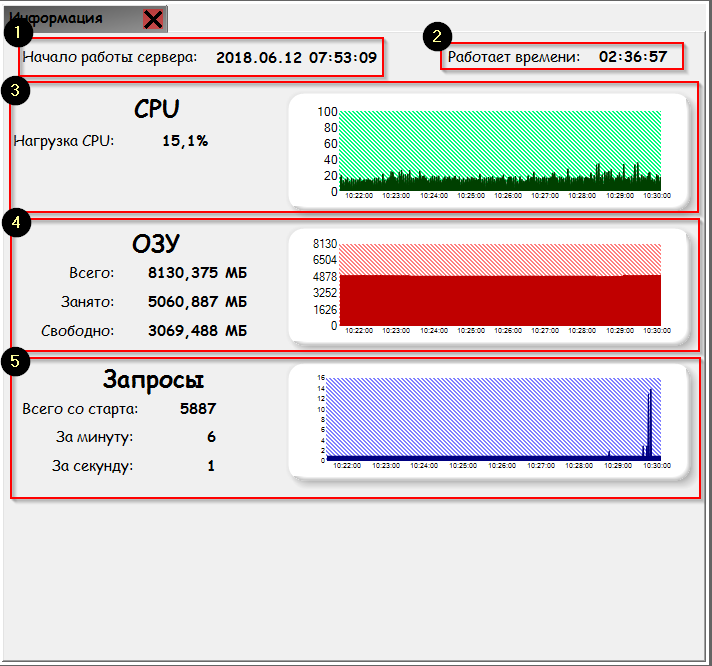


Рисунок 79 - Интерфейс панели «Информация»

1. Панель «Слежение»

В этой панели реализована возможность следить за какой-либо переменной и ее изменениями (см. рисунок 80). С начала слежение происходит заполнения графика, отображающего на временной шкале, как меняла свое состояние переменная. Следить возможно только за числовыми переменными. Если переменная имеет не числовой формат, график заполняться не будет.

Описание интерфейса:

1. Выводится имя переменной и из какой базы данных идет получение информации.
2. Поле установки периода обновления в миллисекундах. Установка значения влияет на то, с какой частотой график будет обновлять значение переменной.
3. Возможность выбора цвета графика нужна, чтобы пользователь выбрал какие цвета будет иметь график для этой переменной.
4. Последнее полученное значение переменной.
5. График, отображающий изменения переменной на временной шкале с начала слежения
6. Кнопка «Перестать следить», нажатие на которую влечет отключение слежения за данной переменной.
7. Можно следить не за одной переменной. Показана еще одна для примера.

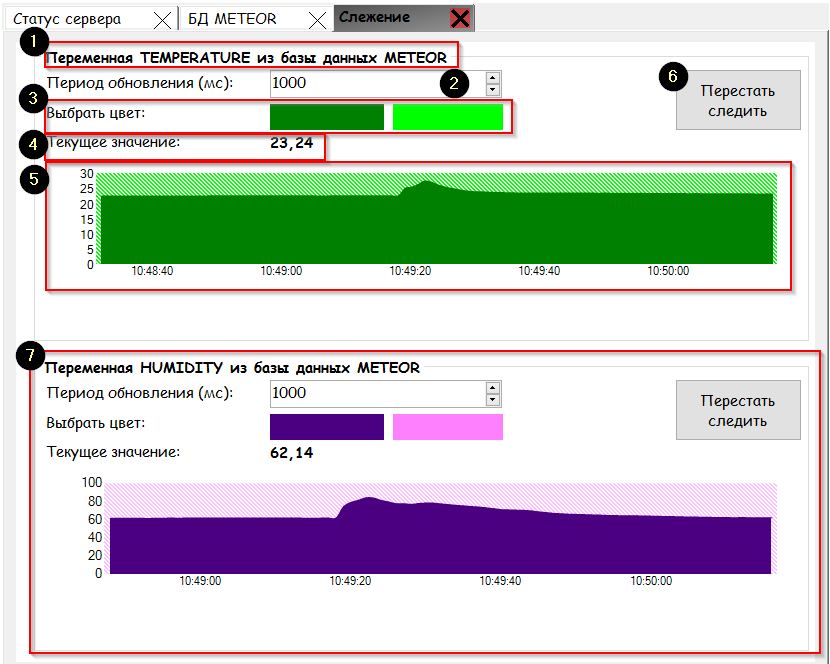


Рисунок 80 - Интерфейс панели «Слежение»

1. Панель «Консоль»

Данная панель позволяет использовать сырое подключение, ничем не отличающегося соединения, как при использовании Telnet.

Описание интерфейса:

1. Поле ввода команд.
2. Поле вывода сообщений сервера.

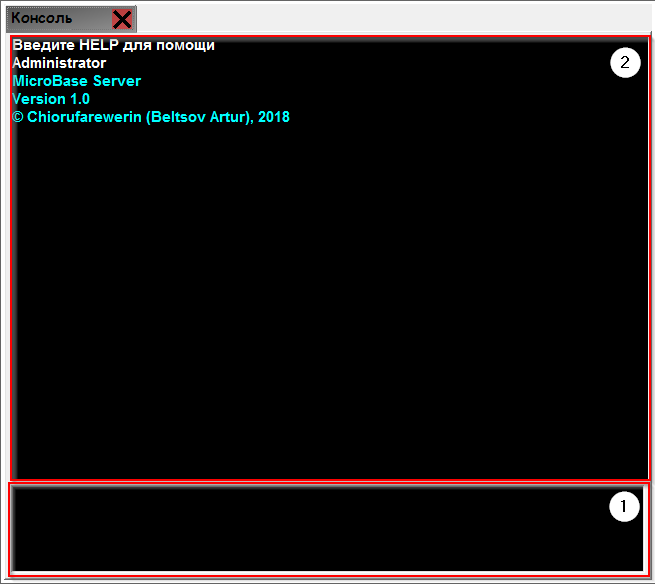


Рисунок 81 - Интерфейс панели «Консоль»

1. Панель «Конфигурация»

С помощью данной панели можно произвести настройки сервера, не прибегая к редактированию конфигов вручную (см. рисунок 82). Недоступна смена пароля администратора. Его возможно поменять исключительно только в файле конфига.

Описание интерфейса:

1. Поля, отображающие текущие конфигурации сервера с возможностью их изменения.
2. Кнопка «Применить изменения» – применяет новые конфигурации на сервере. При возникновении ошибок, возникает сообщение об этом.

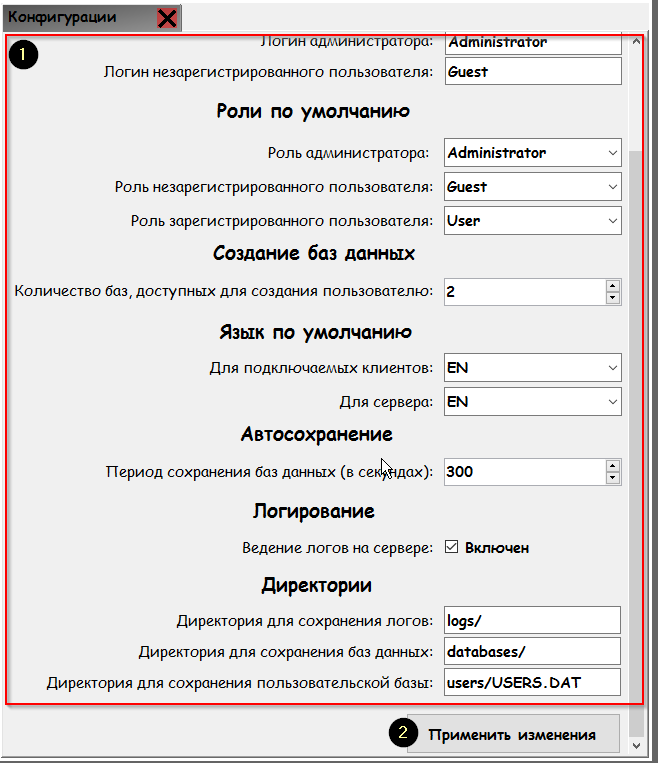


Рисунок 82 - Интерфейс панели «Конфигурации»

1. Панель управления данными в базе данных

Для того, чтобы производить управление переменными в базе данных, необходимо выбрать ее из списка на главной форме программы. Та, что отмечена жирным цветом, означает, что у вас доступ к получению точно есть, а значит есть и доступ к этой панели, интерфейс которой есть на рисунке 83.

Описание интерфейса:

1. Кнопки обновления данных. Позволяют добавлять, изменять и удалять значения в базе данных.
2. Кнопка «Следить» – добавляет переменную в график на панель «Слежение», описание которой было до этого.
3. Выбор метода сортировки.
4. Отбор позволяет вывести переменные с определенным названием. Это поле позволяет вводить маску, где «\*» – означает любые символы с любой длиной (от нуля до бесконечности) и «?» – означает любой один символ.
5. Кнопка «Обновить» обновляет данные в таблице.
6. Таблица переменных и значений. Значения хранятся в базе данных как массивы или списки, поэтому там, где два и более значений, они выводятся через запятую.

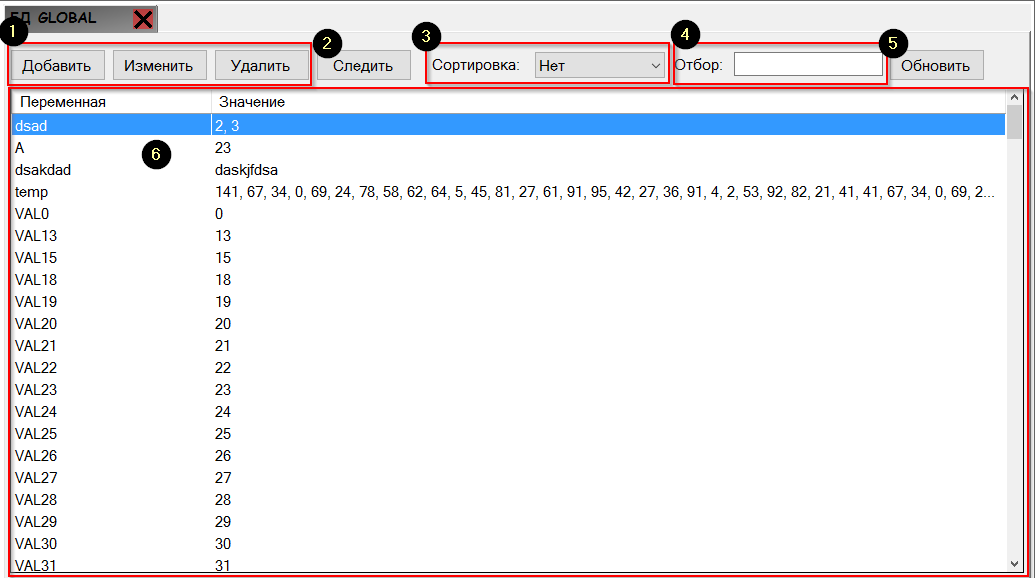


Рисунок 83 - Интерфейс управления данными в безе данных

1. Панель конфигурации базой данных

В данной панели имеется возможность изменять конфигурации базы данных и назначать права пользователям(см. рисунок 84). Данная панель доступна, если у пользователя есть разрешение на полный доступ к базе данных.

Описание интерфейса:

1. Кнопка «Удалить базу данных» – удаляет эту базу данных.
2. Кнопка «Переименовать базу данных» – открывает форму, где указывается новое название базы данных.
3. Некоторые сведения о базе данных.
4. Кнопки управления назначения ролей пользователя в базе данных.
5. Таблица пользователей и их роли в базе данных.



Рисунок 84 - Интерфейс панели конфигурации базы данных

1. Тестирование и отладка программы

Существует несколько основных методов тестирования приложения.

* модульное тестирование;
* интегрированное тестирования.

Модульное тестирования наиболее эффективно и нужно в программах, где идут сложные вычисления, так как под ним подразумевается, что при определенных данных, отданных для теста, мы должны 100% получить какие-то другие данные, если это не так, то тест считается проваленным. С развитием приложения могут возникнут ошибки в тех модулях, где все было нормально, но что-то пошло не так, и оно помогает найти эту ошибку очень быстро.

В СУБД, как и в визуальном клиенте, нет каких-либо сложных вычислений, а все ошибки могут быть связаны исключительно из-за разрыва соединения, поэтому нет необходимости его делать.

А вот интегрированное тестирование необходима, чтобы проверить правильность работы компонентов, поэтому оно было произведено как для сервера, так и для клиента.

Результаты тестирования для сервера представлены в таблице 26, а для клиента в таблице 27.

**Таблица 26 - Тестовые наборы интегрированного тестирования для сервера**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Действие | Реакция системы | Вывод |
| 1) | Введение команды CREATE TESTDB | Ошибка доступа | Успех |
| 2) | Введение команды LOGIN Administrator password | Вход в систему | Успех |
| 3) | Введение команды CREATE TESTDB | Создана база данных TESTDB | Успех |
| 4) | Введение команды USE TESTDB | Вход в базу данных TESTDB | Успех |
| 5) | Введение команды GET A | Получен <NULL> | Успех |
| 6) | Введение команды SET A 1 | Значение А стало 1 | Успех |
| 7) | Введение команды ME | Вывод Administrator | Успех |
| 8) | Введение команды INFO | Вывод данных о состоянии СУБД | Успех |
| 9) | Введение команды USERS | Получен список пользователей | Успех |
| 10) | Введение команды ABCDE | Ошибка команды | Успех |

**Таблица 27 - Тестовые наборы интегрированного тестирования для клиента**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Действие | Реакция системы | Вывод |
| 1) | Нажатие по кнопке «Добавить подключение» | Открывается форма «Добавление подключения» | Успех |
| 2) | Ввод имени подключения, IP адреса и порта и нажатие по кнопке «ОК» | Добавляется подключение | Успех |
| 3) | Нажатие по кнопке с подключением | Открывается главная форма управления | Успех |
| 4) | Нажатие по кнопке «Конфигурация» | Открывается панель «Конфигурация», где надпись «Нет доступа» | Успех |
| 5) | Нажатие по кнопке «Пользователи» | Открытие панели «Пользователи» | Успех |
| 6) | Нажатие по кнопке «Сменить пользователя» | Открывается форма «Вход в систему» | Успех |
| 7) | Ввод имени пользователя и пароля «и нажатие по кнопке «ОК» | Ввод в учетную запись | Успех |
| 8) | Нажатие по кнопке «Управление БД» у GLOBAL | Открытие панели управление БД | Успех |
| 9) | Нажатие по кнопке «изменить» при выбранном пользователе TESTUSER | Открытие изменения прав пользователя TESTUSER | Успех |
| 10) | Добавление TESTUSER возможности редактирования БД и нажатие кнопки «ОК» | Пользователь TESTUSER имеет возможность редактировать БД GLOBAL | Успех |

1. Локализация программного обеспечения

Пользователей в мире огромное количество и кому-нибудь из них может понадобится эта СУБД вместе с визуальным клиентом, поэтому было использовано два основных языка при создании приложений: английский и русский. Английский язык распространен по всему миру, поэтому и был выбран.

Если пользователь откроет помощь для сервера СУБД командой «-h», то увидит, что ему доступно на выбор два языка, как это видно на рисунке 85. Этот язык ставится только для сервера, пользователи имеют возможность менять язык. Подключаясь с помощью Telnet, пользователи могут указать язык, на котором будут выводиться сообщения. Пример вывода сообщений на разных языках можно увидеть на рисунке 86.

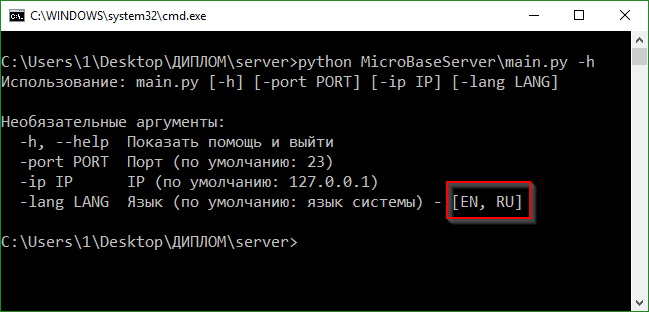


Рисунок 85 - Доступные языки сервера (обведены в красную рамку)

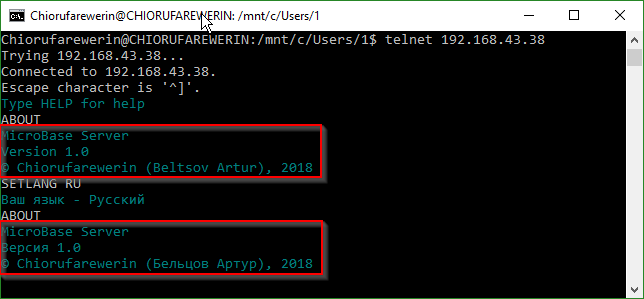


Рисунок 86 - Вывод сообщения до и после смены языка

Аналогично был локализован и пользовательский интерфейс в визуальном клиенте. В настройках можно выбрать доступные языки, как можно увидеть на рисунке 87.

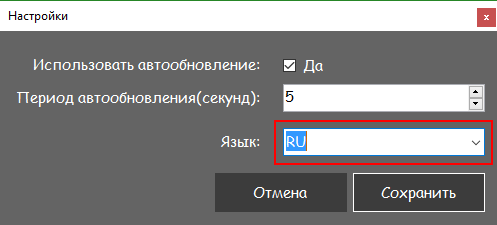


Рисунок 87 - Возможность выбора языка в визуальном клиенте

Разработка приложения велась исходя из того, что пользователи возможно захотят использовать не только встроенные языки, но смогут доработать и добавить новый язык, поэтому на сервере используется класс, от которого могут наследовать другие языки. А для визуального клиента, в связи с тем, что это компилируемое приложение, используются файлы в папке langs, в которых идет перечисление названий переменных и значений. Пользователь может взять, как за основу файл с английским языком, перевести на свой язык, и сохранить под другим наименованием.

1. Инструкция пользователя

В данном разделе представлена инструкция для пользователя, о том, как правильно пользоваться программой. Так как СУБД идет вместе с визуальным клиентом, то, соответственно инструкции представлены не только для сервера и подключения к нему, используя Telnet соединение, а также инструкция для визуального менеджера.

1. Настройка и запуск СУБД

Сервер написан на интерпретированном языке Python, поэтому необходимо установить Python для работы сервера. Для этого нужен Python версии не ниже 3.6. Также для работы необходимы дополнительные модули, такие как psutil, gettext и argparse, устанавливаемые программой pip. Процесс установки представляет из себя введение ее в консоль, а также аргумента install, после которого указывается нужный модуль для установки. Процесс установки представлен на рисунке 88.



Рисунок 88 - Установка пакета Python

1. Настройка конфигураций сервера

Перед запуском сервера необходимо изменить файл конфигурации и настроить переменные. С сервером идет файл config.cfg, в котором находится ряд переменных, который представлен на рисунке 89. Наиболее важно установить пароль администратора по умолчанию, ведь он нужен для настроек баз данных и пользователей в СУБД. Остальные настройки не требуют обязательного изменения, но, при необходимости можно их исправить на нужные вам.



Рисунок 89 - Конфигурации сервера

1. Запуск сервера

В папке с сервером идет файл start.bat, который используется в Windows для выполнения сценариев. В этом файле находится строка для запуска сервера, а также ряд некоторых аргументов, таких как IP адрес и язык запуска. Данный файл представлен на рисунке 90.

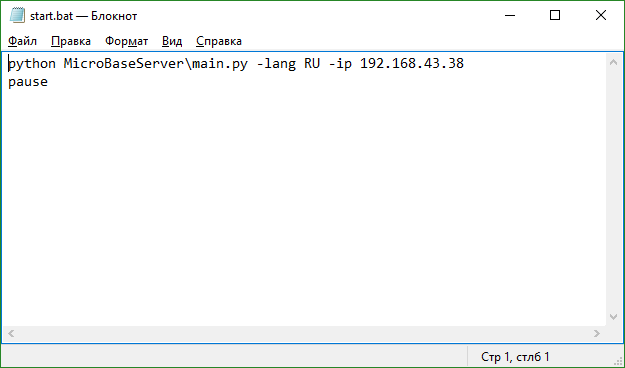


Рисунок 90 - Файл start.bat

После запуска этого сценария следует загрузка сервера и ожидание клиентов для подключения к ней, как можно видеть на рисунке 91.

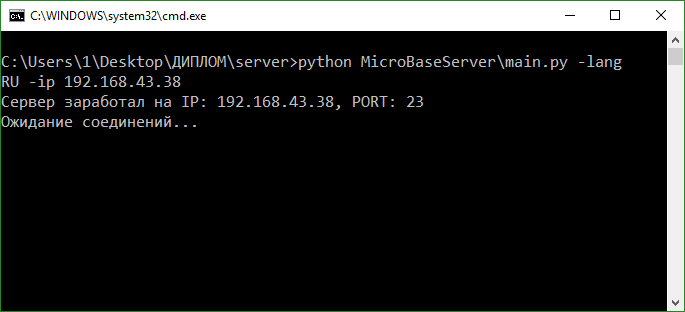


Рисунок 91 - Интерфейс сервера

1. Сценарий работы Telnet подключения

Подключение к базе данных может осуществляться при помощи Telnet соединения, поэтому представлен основной возможный сценарий работы с СУБД, используя Telnet клиент. В данном случае используется встроенный клиент в оболочку bash, которую может эмулировать последняя версия операционной системы Windows.

1. Подключение к серверу

Для подключения, требуется указать IP адрес и порт, если он отличается от 23, но так как, на сервере уже используется именно он, то его не требуется указывать. Процесс подключения представлен на рисунке 92.



Рисунок 92 - Подключение с помощью Telnet

1. Регистрация и вход

Для регистрации необходимо передать команду register с аргументами, такими как логин, пароль и повтор пароля. После подключения, необходимо войти при помощи команды login, которая принимает на вход логин и пароль. Данный процесс представлен на рисунке 93.

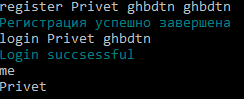


Рисунок 93 - Регистрация и вход с помощью Telnet

1. Создание базы данных

Имеется возможность создать свою базу данных при помощи команды create, которая принимает на вход имя новой базы данных. После создания необходимо войти в нее при помощи команды use. Данный процесс представлен на рисунке 94. В созданной базе данных вы имеете максимальные привилегии, как можно убедиться, передав команду GETACCESSDB.

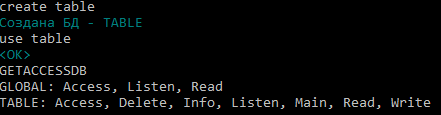


Рисунок 94 - Создание базы и вход в нее с помощью Telnet

1. Добавление и получение данных

В подключенной базе данных вы имеете полное право ей пользоваться, как, например, для создания переменных, при мощи команды set, которая получает на вход имя переменной и значения, которые будут в ней находиться. С помощью команды get можно получить значения переменной, как можно убедиться на рисунке 95.



Рисунок 95 - Добавление и получение переменных с помощью Telnet

1. Команды

СУБД имеет в своем ассортименте множество команд, позволяющих полностью управлять сервером при помощи них. Все команды можно разделить на отдельные подгруппы, описание которых следует ниже.

1. Справочные команды

Данные команды необходимы для получения справочной информации, которая может помочь вам при работе с СУБД.

1. HELP – базовая команда, предлагаемая для ввода при подключении пользователя. В ней описаны все основные возможности взаимодействия с сервером.
2. DBHELP – получить справку о командах для настроек баз данных.
3. VARHELP – получить справку о командах для использования данных.
4. LISTENINGHELP – получить справку о командах для слушателей.
5. OTHERHELP – справка о других возможных командах.
6. LS – получить список доступных команд для пользователя.
7. LSALL – получить список всех команд.
8. SYMBOLS – получить список запрещенных символов.
9. ABOUT – получить справку о программе.
10. Команды работы с пользователями
11. REGISTER – регистрация пользователя. Передается логин, пароль и повтор пароля для заведения нового пользователя.
12. LOGIN – вход на сервер, используя логин и пароль.
13. DELUSER – удалить выбранного пользователя.
14. CHANGEPASSWORD – изменить пароль.
15. USERS – получить список пользователей, зарегистрированных на сервере.
16. ME – получить имя текущего пользователя.
17. CHANGEUSER – изменить настройки пользователя.
18. GETUSERINFO – получить информацию о пользователе.
19. GETMYINFO – получить информацию о текущем пользователе.
20. GETSTANDARTUSERS – получить список стандартных пользователей.
21. Команды работы с базами данных
22. CREATE – создание базы данных.
23. USE – использование базы данных.
24. DBDEL – удалить базу данных.
25. DBALL – получить список всех баз данных.
26. SAVE – сохранить базы данных.
27. Команды работы с данными
28. SET – установить значение переменной.
29. GET – получить значение переменной.
30. DEL – удалить переменную.
31. LEN – получить длину значений переменной. Так как переменная хранит массив данных, то возможно хранение не единственного значения.
32. VALUES – получить все переменные из базы данных. Передавая параметр LIST, выводит все переменные, а при параметре FULL, возможно получить все переменные и их значения. Также команда позволяет сортировать значения при помощи ключей и делать отбор.
33. DBCLEAR – очистить все переменные из базы данных.
34. CURRENTDB – получить текущую базу данных.
35. EXISTS – проверить на существование переменных. Возвращает число существующих переменных.
36. APPEND – добавить значение к переменной.
37. RENAME – переименовать переменную.
38. INCR – инкрементировать, то есть прибавить 1 к переменной, если та имеет числовое значение.
39. DECR – декрементировать, то есть убавить 1 у переменной, если та имеет числовое значение.
40. ADD - прибавить число к переменной, если та имеет числовое значение.
41. SUB – убавить число у переменной, если та имеет числовое значение.
42. Команды прослушивания
43. LISTEN – начать прослушивание базы данных.
44. LISTENALL – слушать всю базу данных.
45. STOP – остановить прослушку.
46. Команды работы с языками
47. LANGS – доступные языки.
48. SETLANGS – установить язык.
49. Команды для разграничения прав
50. GETROLES – получить роли в СУБД.
51. GETROLECOMMANDS – получить команды ролей.
52. GETUSERROLE – получить роль пользователя.
53. USERCOMMANDS – получить команды пользователя.
54. CHANGEUSERCOMMANDS – изменить команды пользователя.
55. GETUSERSDB – получить пользователей базы данных.
56. ADDUSERTODB – добавить пользователя к БД.
57. REMOVEUSERFROMDB – удалить пользователя из БД.
58. GETACCESSDB – получить доступы к базам данных.
59. CHANGEACCESS – изменить доступ к базе данных.
60. Информационные команды
61. COUNTCONNECTIONSDB – количество соединений в базе данных.
62. COUNTVARS – количество переменных в СУБД.
63. COUNTUSERS – количество пользователей в СУБД.
64. COUNTCONNECTIONS – количество соединений в СУБД.
65. COUNTCOMMANDS – количество команд.
66. COUNTLISTENERS – количество слушателей.
67. COUNTLISTENERSDB – количество слушателей в БД.
68. COUNTVARSALL – количество переменных в СУБД.
69. COUNTVARSALLDB – количество переменных в базе данных.
70. GETCONNECTIONS – получить информацию о соединениях.
71. LENVARIABLES – длины переменных в текущей базе данных.
72. LENVAREABLESALL – длины переменных из всей базы данных.
73. DATE – текущая дата на сервере.
74. TIME – текущее время на сервере.
75. INFO – получить информацию об используемом CPU, количестве оперативной памяти и беспрерывной работе сервера.
76. GETSPACE – получить информацию о занимаемом дисковом пространстве базами данных.
77. Команды для управления сервером
78. SHUTDOWN – выключить СУБД.
79. RELOAD – перезагрузить базу данных.
80. GETSETTINGS – получить текущие конфигурации в СУБД.
81. SETSETTINGS – установить конфигурации в СУБД.
82. Другие команды
83. ECHO – вывести полученное сообщение.
84. CLEAR – вывести множество переходов на новую строку.
85. QUIT – отключиться от СУБД.
86. EVAL – выполнить команду Python.
87. EXEC – выполнить команду операционной системы.
88. VISUAL – отправить информацию о том, что используется визуальное подключение.
89. Сценарий работы визуального клиента

Визуальный клиент неявно общается командами, только пользователю нет необходимости их вводить. Но если уж возникает такая необходимость, то в дополнении добавлена консоль, в которую возможен ручной ввод команд, как это представлено на рисунке 96.

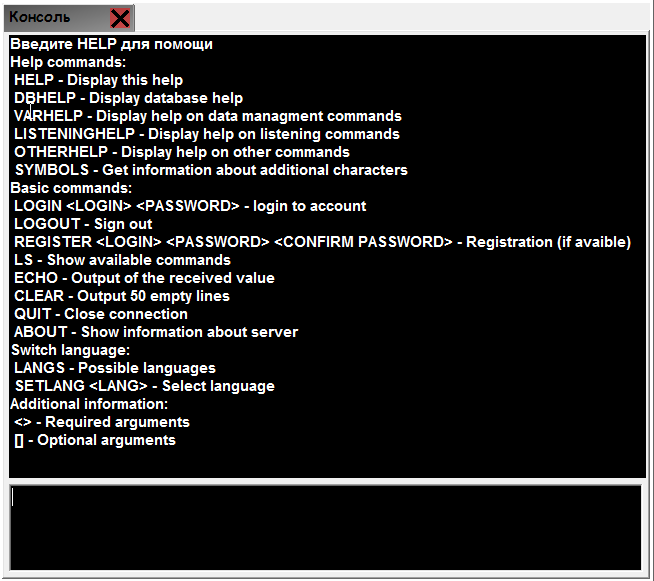


Рисунок 96 - Консоль в визуальном клиенте

1. Открытие справки

С первым входом в программу, если пользователь не разу с подобным программным обеспечением не работал, он сразу может не разобраться, как ею пользоваться, поэтому была добавлена справка, которая открывается из строки меню сверх по вкладке «Справка» – «Открыть справку», как видно на рисунке 97, а вид справки представлен на 98.

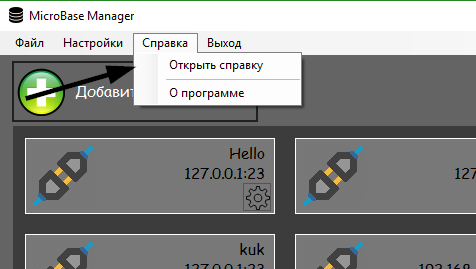


Рисунок 97 - Пункт меню для открытия справки

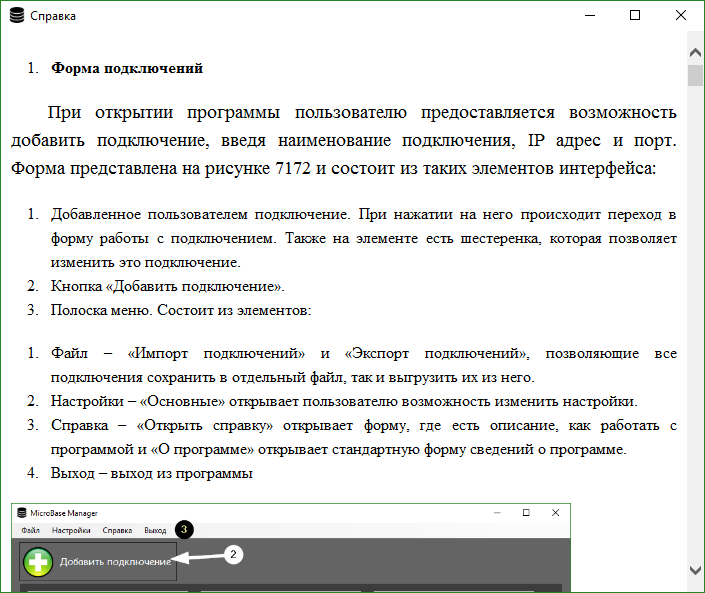


Рисунок 98 - Форма справки

1. Добавление подключения

Перед тем, как подключиться, необходимо на самой первой форме добавить подключение, для этого нажимаете на кнопку «Добавить подключение», как это видно на рисунке 99, вводите «Наименование», которое необходимо только для вас, «Хост», который представляет IP адрес или же адрес сервера, а также порт, на котором работает сервер, после этих процедур нажимаете «ОК». Пример ввода представлен на рисунке 100

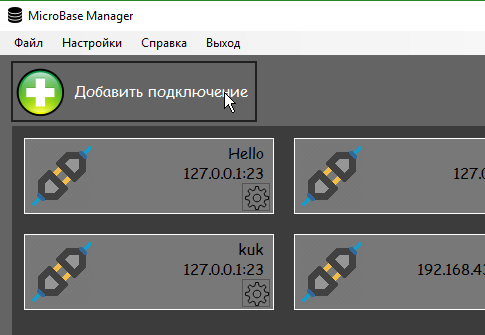


Рисунок 99 - Нажатие на кнопку «Добавить подключение»

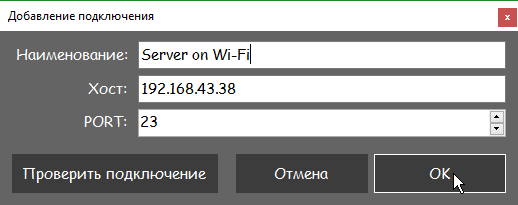


Рисунок 100 - Пример ввода данных

1. Добавление данных в базу данных

Для этого необходимо выбрать базу данных из списка, что у вас имеется на запись и сделать 2 щелчка левой кнопкой мыши по ней, а далее нажать на кнопку «Добавить», как это представлено на рисунке 101.

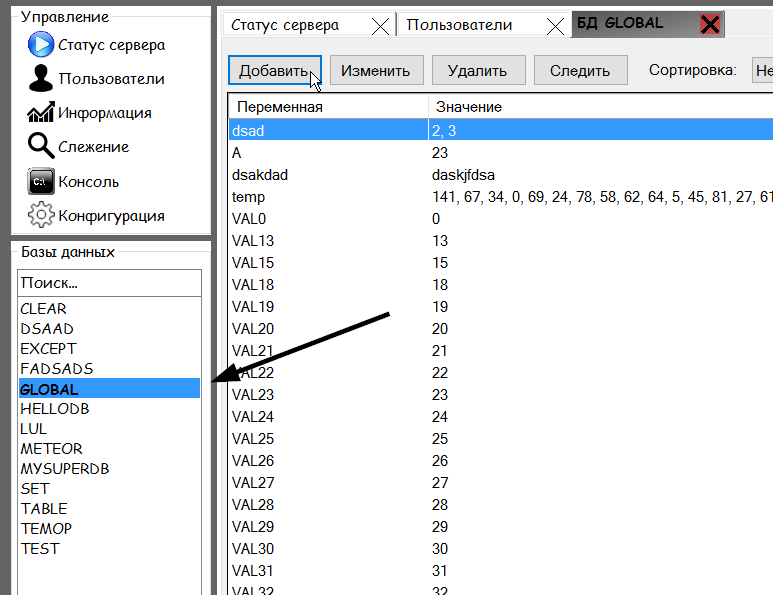


Рисунок 101 - Выбор базы и открытие формы «Добавить переменную»

После этого заполняем поля, где «Наименование» – название переменной, а «Значения» – значения переменной и нажимаем кнопку «ОК». Пример заполнения представлен на рисунке 102.

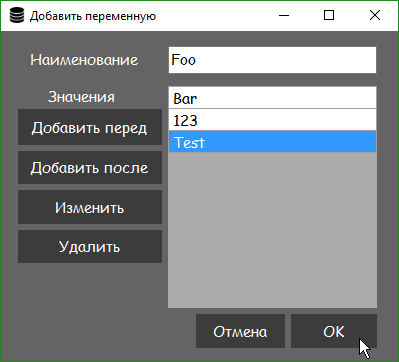


Рисунок 102 - Пример заполнения

1. Анализ результатов работы

В ходе всей работы был создан проект, удовлетворяющий поставленным задачам. Разработка велась исходя из технического задания в начале записки.

- клиенты имеют возможность подключаться к СУБД, используя сетевой протокол;

- клиенты могут использовать подключение Telnet;

- доступна регистрация для клиентов, а также использование пользователей для разграничения ролей;

- возможно создавать базы данных и осуществлять их управление при помощи команд;

- возможно выбрать язык;

- микроконтроллеры способны использовать данную СУБД;

- возможно осуществлять исправления программного кода в СУБД при необходимости коррекции для своей нужды;

- базы данных сохраняются в файлы и, в то же время, находятся в оперативной памяти;

- запись и получение данных;

- доступна справочная информация из программы;

- возможно прослушивать базы данных и следить за изменениями информации;

- ведется запись логов.

Так как на выходе имеется не только СУБД, но и менеджер по управлению базами данных, то для него аналогично было выполнено все, что требовалось, исходя из технического задания.

* может подключаться к СУБД;
* имеет в наличии возможность сохранять подключения, а также может импортировать или экспортировать их;
* осуществляет управление базами данных и данными при помощи визуальных средств;
* отображает графики изменений данных в реальном времени;
* выводит пользователю данные в удобном виде;
* дает возможность импортировать или экспортировать базы данных.

## ЭКОНОМИЧЕСКАЯ ЧАСТЬ

В данном разделе описано экономическое обоснование данного программного продукта, а также имеется расчет прибыли от реализации.

1. Описание продукта

Данное программное решение является СУБД, которое послужит отличным средством для сохранения информации передаваемой микроконтроллерами, и не только ими. Разрабатывая эту СУБД, прежде всего был сделан акцент на ее производительности, поэтому передаваемая информация записывается молниеносно, а выдача происходит с такой же скоростью. Для безопасности информации, существует регистрация пользователей, а также разграничение их прав доступа.

Совместно с СУБД, идет визуальный менеджер по управлению базами данных, с помощью которого осуществляется настройка СУБД, не прибегая к использованию команд, набираемых с помощью клавиатуры, а с помощью визуальных средств. Данный визуальный клиент является неотъемлемой частью, благодаря возможности анализа информации в базе данных, а также вывода общей информации в виде графиков, которые показывают изменения значений в реальном времени.

Эти средства нацелены в первую очередь не на рядового пользователя, а на программиста, у которого возникла проблема сохранения информации и необходимость в выборе СУБД, которая позволит это сделать с большой производительностью. СУБД позволяет сохранять информацию не только с микроконтроллеров, но благодаря своей легкость настройки и быстродействию, послужит отличным решением именно для них.

1. Описание и расчет затрат на выполнение проекта

Разработка данной программы, как и разработка любого другого приложения, требует определенных материальных, временных и трудовых затрат, а значит, должна окупаться. С экономической точки зрения затраты, связанные с выполнением проекта, должны быть покрыты от реализации конечного продукта.

Трудоемкость выполнения программного продукта характеризуется перечнем основных этапов и видов работ, которые должны быть выполнены. Упорядочен данный перечень в соответствии со смысловым содержанием каждого вида работ и взаимосвязями между всеми видами работ.

Перед тем как приступить к разработке программного продукта необходимо сформировать техническое задание, которые может быть выдано заказчиком или сформировано самим производителем основываясь на потребности клиентов.

Выделим основные этапы разработки приложения:

1. Подготовительный этап:

* сбор информации;
* выбор объектного построения программы;
* разработка общей методики создания продукта.

2. Основной этап:

* разработка основного алгоритма;
* создание интерфейса;
* отладка.

3. Завершающий этап:

* подготовка технической документации;
* сдача продукта.

Трудоемкость выполнения работы оценивается в человеко-часах.

Трудоёмкость по видам работ представлена на таблице 28.

**Таблица 28 - Расчет трудоемкости**

|  |  |  |
| --- | --- | --- |
| № п/п | Виды работ | Трудоёмкость |
| 1 | Составление ТЗ | 1 |
| 2 | Сбор информации и ознакомление с предметной областью | 31 |
| 3 | Определение возможных направлений решения | 20 |
| 4 | Выбор объективного построения программы | 8 |
| 5 | Разработка общей методики создания продукта | 7 |
| 6 | Подготовка промежуточного отчета | 10 |
| 7 | Разработка основного алгоритма | 75 |
| 8 | Создание интерфейса | 40 |
| 9 | Отладка | 17 |
| 10 | Подготовка технической документации | 15 |
| 11 | Сдача продукта | 1 |
|  | Итого | 225 |

Суммарная трудоемкость составила 225 чел./час.

1. Расчет расходов на заработную плату

На основе данных о трудоемкости и средней заработной плате по отрасли рассчитываем основную заработную плату. Предположим, что заработная плата программиста без опыта работы составляет 30000 руб. в месяц (21 рабочий день, 8 часовой рабочий день) или 178,57 руб./час.

Таким образом, расходы на заработную плату по нашему проекту составляют:

ЗП = 178,57 \* 225 = 40178,25 руб.

Отчисления на ЗП (Пенсионный Фонд, Фонд социального страхования, Фонд обязательного мед. страхования, территориальные фонды мед. страхования) составляют 30%. В денежном выражении составляют:

40178,25 \* 0,3 = 12053,47 руб.

1. Расчет затрат на материалы

Материалы, затраченные на создание проекта приведены в таблице 29.

**Таблица 29 - Расчет трудоемкости**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Материальные ресурсы | Единицы измерения | Потребляемое количество | Цена за единицу, руб. | Сумма, руб. |
| 1 | Заправка картриджа | шт | 1 | 600 | 600 |
| 2 | Бумага формата А4 | упаковка | 1 | 257 | 257 |
| 3 | Папка | шт | 1 | 199 | 199 |
| 4 | Файлы | упаковка | 2 | 168 | 336 |
| 5 | USB флеш-накопитель | шт | 1 | 250 | 250 |
| 6 | Node MCU | шт | 1 | 236 | 236 |
| 7 | Набор кабелей | упаковка | 1 | 83 | 83 |
| 8 | Датчик BME280 | шт | 1 | 263 | 263 |
|  | Итого |  |  |  | 2224 |

Суммарная стоимость материалов составила 2224 руб.

1. Расчет амортизации

Стоимость компьютера 42000 руб. Используем его в течении 5 лет (60 мес.). Годовая амортизация составит в месяц 700 руб. В час: 4,17 руб. Умножив на трудоемкость, определим:

АО = 4,17 \* 225 = 938,25 руб.

1. Расчет электроэнергии

Персональный компьютер в среднем употребляет 0,3 кВт/ч. 0,3 \* 225 = 67,5 кВт. Стоимость 1 кВт/ч около 4,08 руб.

Итого: 67,5 \* 4,08 = 275,4 руб.

1. Расчет сметы затрат

Все результаты расчета затрат приведены в таблице 30.

**Таблица 30 - Расчет трудоемкости**

|  |  |  |
| --- | --- | --- |
| № п/п | Наименование статей затрат | Сумма, руб. |
| 1 | Расходные материалы | 2224 |
| 2 | Основная заработная плата | 40178,25 |
| 3 | Расходы на электроэнергию | 275,4 |
| 4 | Амортизационные отчисления | 938,25 |
| 5 | Отчисления на ЗП | 12053,47 |
|  | Итого | 55669,37 |

Таким образом, суммарная стоимость затрат на создание данного программного продукта составляет 55669,37 рублей, следовательно, эту программу ниже этой цены продать нельзя так, как в этом случае программа не будет являться экономически эффективной.

1. Расчет предполагаемой прибыли

На рынке программных продуктов аналогичные решения существуют. Многие из них распространяют под свободной лицензией GPL GNU и являются бесплатными. Однако, созданное приложение имеет ряд уникальных особенностей:

* нахождение данных в оперативной памяти;
* простая смена языка;
* возможность следить за изменениями в реальном времени;
* совместно с СУБД идет визуальный клиент-менеджер.

Помимо этих особенностей, в программу включены достоинства аналогичных продуктов:

* разграничение информации с помощью структур, именованных базами данных;
* наличие пользователей на роли и разграничение их прав доступа к базам данных;
* использование коротких команд;
* простая настройка и установка;
* осуществление записей об изменениях.

Основной чертой данного продукта является не только высокое быстродействие, которое обеспечивается нахождением данных в оперативной памяти, но и наличие визуального менеджера по настройке СУБД. Если рассматривать другие аналогичные программные продукты, то можно встретить эти функции, но будут отсутствовать другие, удобные функций.

Визуальный менеджер является отличным дополнением к СУБД. Благодаря этому не придется производить настройку с помощью команд, а также именно он позволяет оценить базу данных с помощью диаграмм и графиков. Пользователю лишь необходимо выбрать за чем следить и будет выводиться график изменений этой переменной.

Реализация создания программного продукта может осуществляться двумя способами.

Первый способ реализации программного продукта полностью, с оформлением договора отчуждения. При средней цене в 80-90 тыс. рублей найти покупателя на СУБД при цене в 65 тыс. рублей не составит труда. Таким образом предполагаемая прибыль составляет:

65000 - 55669 = 9331 руб.

Второй способ реализации программы предполагает предоставление права использования программы с ограниченным функционалом и ограничением по времени. При цене в 10 тыс. рублей, реализовав более 6 экземпляров программы, созданный программный продукт окупает затраты:

60000 - 55669 = 4331 руб.

# ЗАКЛЮЧЕНИЕ

Хранение данных в современном мире не обходится без специализированных программ, именуемых системами управления базами данных. Каждый программист при создании своего проекта, где требуется передача и хранение данных, начинает с выбора СУБД, которая ему в этом поможет.

Данная СУБД будет полезна для начинающих программистов, которые будут использовать микроконтроллеры и передавать данные с них для дальнейшей обработки. В настройках сервера поможет визуальный клиент, который не требует держать в уме множество команд и упрощает получение и изменение информации.

Также существует разграничение прав для пользователей, как для самой СУБД, какие команды сможет выполнять пользователь, так и для отдельных баз данных, что пользователь может с ними делать.

В ходе выполнения проекта были решены следующие задачи:

* выполнить обзор и анализ существующих аналогов;
* рассмотреть способы подключения;
* изучить способы хранения данных;
* внедрить создание пользователей и разграничение прав;
* создать визуальный менеджер.

При создании программного продукта были учтены достоинства и недостатки аналогичных программных продуктов, а также добавлены специальные функциональные возможности для удобства пользователя.

Основной целью дипломного проекта являлась разработка программного продукта, позволяющего хранить и передавать данные с наибольшей скоростью и возможностью использования микроконтроллеров, как основных передатчиков информации. Данная цель в ходе разработки была достигнута.

В дальнейшем в СУБД планируется разграничение команд на отдельные группы, для уменьшения проверок, что позволит передавать данные быстрее, когда это нужно, и при этом оставаться такой же безопасной. А для визуального менеджера добавить больше возможностей анализа данных в СУБД.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. ГОСТ 2.105–95 Единая система конструкторской документации. Общие требования к текстовым документам.
2. ГОСТ 7.1–84 Система стандартов по информации, библиотечному и издательскому делу. Библиографическое описание документа. Общие требования и правила составления.
3. Марк Лутц – Изучаем Python, 4-е издание.
4. Кодд Е.Ф. – Реляционная модель данных для больших совместно используемых банков данных (перевод М.Р. Когаловского).
5. Лапонина О.Р. – «Криптографические основы безопасности» 2004 год выпуска
6. Джон Скит – C# для профессионалов: тонкости программирования, 3-е издание, 2014.
7. http://fb.ru/article/245758/mikrokontroller-esp-podklyuchenie-i-nastroyka – микроконтроллер ESP подключение и настройка.
8. http://[www.bseu.by/it/tohod/lekcii5.htm – лекция/](http://www.bseu.by/it/tohod/lekcii5.htm%20–%20лекция/)понятие СУБД.
9. http://www.mstu.edu.ru/study/materials/zelenkov/ch\_4\_1.html – организация данных в реляционной модели данных.
10. http://db-engines.com/en/ranking – рейтинг баз данных.
11. http://habr.com/post/322532/ - MySQL и MongoDB, когда и что лучше использовать.
12. http://amperka.ru/product/node-mcu – NodeMCU, описание устройства
13. static.cactus.io/docs/sensors/barometric/bme280/BST-BME280\_DS001-10.pdf - настройка устройства на базе BME280.

# Приложение А1. Исходный текст программы СУБД

main.py

import sys

import os

import socket

import threading

import locale

import language

from language import LANGUAGE

import commands

from commands import COMMANDS, SPECSYMBOLS, COMQUIT, Inf, OUTONLY

import classes

import funs

import database

import usersdb

import variables

import access

import variables

import datetime

import time

from importlib import reload

# Главный файл

# Функция, создающая сокет и ставящая сервер на нужный ip и порт

def InitializationServer(port, ip):

serv\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM, proto=0)

serv\_sock.bind((ip, port))

serv\_sock.listen(0)

serv\_sock.settimeout(3)

return serv\_sock

# Функция для работы с подключенными клиентами

def WorkWithClient(client:socket.socket, addr:tuple):

ClientInfo = classes.ClientInformation(client, addr, LANGUAGE[variables.DEFAULTLANGUAGECLIENTS], database.DATABASES['GLOBAL'], usersdb.DEFAULTUSER)

try:

client.send(ClientInfo.language.TypeHelp)

except:

return

database.ALLCLIENTS.append(ClientInfo)

while True:

try:

data = client.recv(1024)

if data in (b'\xff\xf4\xff\xfd\x06', b'\xff\xed\xff\xfd\x06'): # в некоторых Telnet клиентах есть возможность отключиться от сервера при помощи CTRL+Z или CTRL+C, проверка нажатия этиъ комбинаций

COMQUIT(ClientInfo)

except:

COMQUIT(ClientInfo)

return

if not data:

COMQUIT(ClientInfo)

return

try:

udata = data.decode('utf\_8')

except:

continue

if udata == '\r\n':

continue

udata = udata.strip('\r\n ')

timesec = int(time.time())

if timesec in database.QUERYTIME:

database.QUERYTIME[timesec] += 1

else:

database.QUERYTIME.clear()

database.QUERYTIME[timesec] = 1

if timesec//60 in database.QUERYTIMEMIN:

database.QUERYTIMEMIN[timesec//60] += 1

else:

database.QUERYTIMEMIN.clear()

database.QUERYTIMEMIN[timesec//60] = 1

database.QUERYES += 1

if any(symb in udata for symb in SPECSYMBOLS):

client.send(OUTONLY(ClientInfo, Inf.Error, 'SPECSYMB', ClientInfo.language.NotSpecs)[0])

continue

try:

arguments = funs.GetArguments(udata)

except Exception as e:

client.send(OUTONLY(ClientInfo, Inf.Error, 'NOTCORRECTNUMBERQUOTES', ClientInfo.language[e.args[0]])[0])

continue

command, \*args = arguments

command = command.upper()

if command not in COMMANDS:

client.send(OUTONLY(ClientInfo, Inf.Error, 'COMNOTFOUND' ,ClientInfo.language.CommandNotFound)[0])

continue

if command not in access.ACCESSLEVEL[ClientInfo.CurrentUser.UserRole] and command not in ClientInfo.CurrentUser.UserCommands and ClientInfo.CurrentUser.UserRole != variables.ADMINISTRATORROLE:

client.send(OUTONLY(ClientInfo, Inf.PermissionDenied, '', ClientInfo.language.PermissionDenied)[0])

continue

try:

minargs = COMMANDS[command][2]

maxargs = COMMANDS[command][3]

if len(args) < minargs or (len(args) > maxargs and maxargs != -1):

client.send(OUTONLY(ClientInfo, Inf.Error, 'COUNTARGUMENTERROR', ClientInfo.language.NotCorrectNumberArguments)[0])

continue

information, Info = COMMANDS[command][1](ClientInfo, \*COMMANDS[command][0](ClientInfo,\*args))

if information:

client.send(information)

except Exception as e:

client.send((str(e) + '\r\n').encode())

COMQUIT(ClientInfo)

raise e

if variables.WRITELOGS:

date = str(datetime.datetime.fromtimestamp(int(datetime.datetime.now().timestamp())))

FILELOG.write('{}\t{}:{}-{}|{}\r\n'.format(date, \*addr, ClientInfo.CurrentUser.login, command))

if information:

FILELOG.write('{}\t{}:{}-{}|{}\r\n'.format(date, \*addr, ClientInfo.CurrentUser.login, Info))

FILELOG.flush()

# Функция для начала работы сервера

def work(port, ip=''):

usersdb.InitializeUsers()

database.InitializeDatabase()

if database.CLOSE:

return

try:

database.MAINSERVERSOCKET = InitializationServer(port, ip)

except Exception as ex:

print(language.SERVERLANG.CantStartServer.format(ex))

return

print(language.SERVERLANG.ServerWorking.format('IP: {}, '.format(ip) if len(ip) > 0 else '','PORT: {}'.format(port)))

print(language.SERVERLANG.WaitingForConnections)

database.TIMESTART = int(time.time())

try:

while True: # подключение клиентов к серверу

client = None

try:

client, addr = database.MAINSERVERSOCKET.accept()

print(language.SERVERLANG.ClientConnected.format(\*addr))

thr = threading.Thread(target=WorkWithClient, args=(client,addr))

thr.start()

except socket.timeout:

pass

except Exception as ex:

print(ex)

if client:

client.close()

if database.CLOSE:

return

except KeyboardInterrupt:

funs.CloseTheProgramm()

return

# Функция, возвращающая переданные параментры пользователем

def GetParameters():

def Translate(Text): # перевод справки на язык, установленный системой

Text = Text.replace('usage:', language.SERVERLANG.TranslateUsage)

Text = Text.replace('show this help message and exit', language.SERVERLANG.TranslateShowThis)

Text = Text.replace('error:', language.SERVERLANG.TranslateError)

Text = Text.replace('the following arguments are required:', language.SERVERLANG.TranslateRequired)

Text = Text.replace('optional arguments', language.SERVERLANG.TranslateOptional)

Text = Text.replace('unrecognized arguments', language.SERVERLANG.Translateunrecognized)

Text = Text.replace('expected one argument', language.SERVERLANG.TranslateExpected)

Text = Text.replace('argument', language.SERVERLANG.TranslateArgument)

Text = Text.replace('invalid', language.SERVERLANG.TranslateInvalid)

Text = Text.replace('value:', language.SERVERLANG.TranslateValue)

return Text

import gettext

gettext.gettext = Translate

import argparse

parser = argparse.ArgumentParser()

parser.add\_argument('-port', help=language.SERVERLANG.ParserPort, default=23, type=int)

parser.add\_argument('-ip', help=language.SERVERLANG.ParserIP, default='127.0.0.1', type=str)

parser.add\_argument('-lang', help=language.SERVERLANG.ParserLang.format(', '.join(LANGUAGE.keys()))

, default=None)

args = parser.parse\_args()

return {'port': args.port, 'ip': args.ip, 'lang': args.lang}

# Входная функция в программу

def main():

# Если необходимо создавать логи, то создается файл для этого

if variables.WRITELOGS:

global FILELOG

date = datetime.datetime.now().date()

FILELOG = open('{}{}{}.txt'.format(variables.logspath, '/', date.isoformat()),'a+')

message = None

try: # проверка возможности установить языком программы - язык системы

lang = locale.getdefaultlocale()[0][:2].upper()

if lang in LANGUAGE:

language.SERVERLANG = LANGUAGE[lang]

else:

message = language.SERVERLANG.LanguageNotFoundInTranslated

except:

message = language.SERVERLANG.LanguageNotFound

args = GetParameters()

if args['lang'] != None: # проверка возможности установить языком программы - язык, переданный в аргументе

lang = args['lang'].upper()

if lang not in LANGUAGE:

print(language.SERVERLANG.LanguageUserNotFound)

return

else:

language.SERVERLANG = LANGUAGE[lang]

message = None

if message != None:

print(message)

while database.ISRELOADING:

if database.ISRELOADING:

database.CLOSE = False

reload(variables)

reload(commands)

reload(funs)

reload(database)

reload(usersdb)

database.ISRELOADING = False

work(args['port'], args['ip'])

if \_\_name\_\_ == '\_\_main\_\_':

main()

database.py

from os import listdir, remove

from os.path import isfile, join

from threading import Timer

import pickle

import funs

import classes

import language

import sendinfo

import commands

import variables

import usersdb

# Работа с базами данных и клиентами

CLOSE = False # флаг для полного завершения работы сервера

AUTOSAVETIMER = None # переменная для хранения таймера автосохранения баз данных

MAINSERVERSOCKET = None # переменная для хранения сокета сервера

TIMETOSAVE = variables.AUTOSAVETIME # интервал автосохранения баз данных

PATH = variables.DATABASEDIRECTORY # директория для хранения баз данных

DATABASES = dict() # переменная для хранения баз данных

ALLCLIENTS = [] # переменная для хранения подключенных клиентов

ISRELOADING = True # флаг показывающий перезагрузку сервера

QUERYTIME = {0:0} # Количество запросов за последнюю секунду

QUERYTIMEMIN = {0:0} # Количество запросов за последнюю минуту

QUERYES = 0 # Всего запросов

TIMESTART = 0 # Время запуска сервера

# Функция для получения клиентов в определенной базе данных

def GetClientsInDB(DataBase:classes.DataBase):

return [Client for Client in ALLCLIENTS if Client.DataBase == DataBase]

# Функция для создания базы данных

def CreateDB(DB):

if DB not in DATABASES.keys():

path = '{}{}.DAT'.format(PATH, DB)

DATABASES[DB] = classes.DataBase(DB, path, dict())

with open(path, 'wb') as f:

pickle.dump(dict(), f)

return 'CreateBD'

return 'BDExists'

# Функция для удаления базы данных

def DeleteDB(DB):

if DB not in DATABASES.keys():

return 'BDNotExists'

del DATABASES[DB]

try:

remove('{}{}.DAT'.format(PATH, DB))

except:

pass

usersdb.RemoveFromUsersDatabase(DB)

return 'BDDeleted'

# Функция для сохранения баз данных

def SaveALL():

print(language.SERVERLANG.SaveDatabases)

for DB in DATABASES.values():

with open(DB.path,'wb') as f:

pickle.dump(DB.data, f)

sendinfo.SendToAllListeners('ListenerAllDatabasesSaved', [])

print(language.SERVERLANG.SavedDatabases)

# Функция для автосохранение базы данных с интервалом, указанным в variables.AUTOSAVETIME

def AutoSave():

SaveALL()

usersdb.SaveUsers()

global AUTOSAVETIMER

AUTOSAVETIMER = Timer(TIMETOSAVE, AutoSave)

AUTOSAVETIMER.start()

# Функция для запуска возможности автосохранения

def StartAutoSave():

global AUTOSAVETIMER

if not CLOSE:

AUTOSAVETIMER = Timer(TIMETOSAVE, AutoSave)

AUTOSAVETIMER.start()

# Функция для создания основной базы данных - GLOBAL

def CreateGlobal():

with open(PATH + 'GLOBAL.DAT', 'wb') as f:

pickle.dump(dict(), f)

DATABASES['GLOBAL'] = classes.DataBase('GLOBAL', PATH + 'GLOBAL.DAT', dict())

pathes = [f for f in listdir(PATH) if isfile(join(PATH, f))]

pathesUP = [s.upper() for s in pathes]

# Инициализация и загрузка баз данных в память

def InitializeDatabase():

global DATABASES

if 'GLOBAL.DAT' not in pathesUP:

CreateGlobal()

for path in pathes:

try:

if len(path) < 5 or path[-4:].upper() != '.DAT':

continue

name = path[:-4].upper()

with open(PATH + path, 'rb') as f:

DATABASES[name] = classes.DataBase(name, PATH + path, pickle.load(f))

except: # если база данных повреждена, спрашивается возможность пересоздать ее

print(language.SERVERLANG.CantOpenDatabase.format(path.upper()))

if path.upper() == 'GLOBAL.DAT':

print(language.SERVERLANG.DatabaseGlobalCorrupted)

if funs.GetAnswer():

CreateGlobal()

else:

funs.CloseTheProgramm()

break

else:

name = path[:-4].upper()

print(language.SERVERLANG.DatabaseCorrupted.format(name))

if funs.GetAnswer():

CreateDB(name)

dbs = set()

for user in usersdb.USERS.values():

for db in user.DataBases.keys():

if db not in DATABASES:

dbs.add(db)

for Database in dbs:

print(language.SERVERLANG.DatabaseNotFoundInPath.format(Database))

if funs.GetAnswer():

CreateDB(Database)

else:

for user in usersdb.USERS:

if Database in user.DataBases:

del user.DataBases[Database]

StartAutoSave()

commands.py

import re

import subprocess

import datetime

import time

import database

import funs

from classes import ClientInformation

import sendinfo

import language

from language import LANGUAGE

import variables

import usersdb

import classes

import access

# Работа с командами

SPECSYMBOLS = (',', '\\', '[', ']', '{', '}','<','>', '^') # специальные символы, которые не могут быть использованы в командах

# Класс хранения переменных для передачи информации о доступе в режиме визуального клиента

class Inf:

OK = '<OK>' # команда выполнена успешно

Error = '<ERROR>' # произошла ошибка

PermissionDenied = '<PERMISSION DENIED>' # ошибка доступа

Info = '<INFO>' # информация или данные

# Функции для обработки команд от клиентов

# После выполнения команды, отправляются данные в формате (Краткая информация, Данные, Вывод пользователю) на дальнейшую обработку в функции вывода (например: OUTONLY)

# Далее определяется, пользователь если использует Telnet, выводится только "Вывод пользователю", а если в режиме визуального клиента, то "Краткая информация" и "данные"

# Показать справку

def COMHELP(ClientInfo:ClientInformation):

return (Inf.Info, ClientInfo.language.NA, ClientInfo.language.DefaultHelp)

# Показать справку по управлению базами данных

def COMDBHELP(ClientInfo:ClientInformation):

return (Inf.Info, ClientInfo.language.NA, ClientInfo.language.DBHelp)

# Показать справку по управлению данными

def COMVARHELP(ClientInfo:ClientInformation):

return (Inf.Info, ClientInfo.language.NA, ClientInfo.language.VarHelp)

# Показать справку по прослушиванию

def COMLISTENINGHELP(ClientInfo:ClientInformation):

return (Inf.Info, ClientInfo.language.NA, ClientInfo.language.ListeningHelp)

# Показать справку по различным другим командам

def COMOTHERHELP(ClientInfo:ClientInformation):

return (Inf.Info, ClientInfo.language.NA, ClientInfo.language.OtherHelp)

# Показать информацию о дополнительных символах

def COMSYMBOLS(ClientInfo:ClientInformation):

return (Inf.Info, ' '.join(SPECSYMBOLS),ClientInfo.language.HelpSpecSymbols.format(' '.join(SPECSYMBOLS)))

# Установить значение переменной

def COMSET(ClientInfo:ClientInformation, variable, value, \*args):

if not funs.CheckCommandDataBase(ClientInfo, 'SET'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

if not funs.CheckCorrectNameVariable(variable):

return (Inf.Error, ClientInfo.language.NA, ClientInfo.language.NotCorrectNameOfVariable)

data = [value,] + list(args)

ClientInfo.DataBase.data[variable] = data

sendinfo.SendToListeners(ClientInfo.DataBase, 'ListenerSet', [variable, ', '.join(data)])

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.OK)

# Получить значение переменной

def COMGET(ClientInfo:ClientInformation, variable):

if not funs.CheckCommandDataBase(ClientInfo, 'GET'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

Data = ', '.join(ClientInfo.DataBase.data.get(variable, [ClientInfo.language.NULL, ]))

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Получить количество значений в переменной

def COMLEN(ClientInfo:ClientInformation, variable):

if not funs.CheckCommandDataBase(ClientInfo, 'LEN'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

Data = str(len(ClientInfo.DataBase.data.get(variable, [])))

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Удалить переменную

def COMDEL(ClientInfo:ClientInformation, \*args):

if not funs.CheckCommandDataBase(ClientInfo, 'DEL'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

count = 0

for var in args:

if var in ClientInfo.DataBase.data:

del ClientInfo.DataBase.data[var]

sendinfo.SendToListeners(ClientInfo.DataBase, 'ListenerDel', [var, ])

count += 1

Data = str(count)

return (Inf.OK, Data, '{}\r\n'.format(Data))

# Получить переменные и их значения

def COMVALUES(ClientInfo:ClientInformation, argument, filter=''):

if not funs.CheckCommandDataBase(ClientInfo, 'VALUES'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

DataBase = ClientInfo.DataBase

argument = argument.upper()

sorting = None

if filter != '':

if len(filter) >= 2 and filter[-2] == '/':

sorting = filter[-1].upper()

filter = filter[:-2]

if filter == '':

filter = '\*'

filter = filter.replace('\*', '.\*').replace('?', '.')

pattern = '^.\*{}.\*$'.format(filter)

pattern = re.compile(pattern)

if sorting not in [None, 'A', 'D']:

return (Inf.Error, 'SORT', ClientInfo.language.InvalidSortingMethod)

variabless = DataBase.data.keys()

if sorting in ['A', 'D']:

sorting = sorting == 'D'

variabless = [var for var in sorted(variabless, reverse=sorting) if pattern.match(var)]

else:

variabless = [var for var in variabless if pattern.match(var)]

if argument == 'LIST':

Data = ', '.join(variabless)

toret = Data + '\r\n'

if len(toret) == 2:

toret = ClientInfo.language.NoValues

return (Inf.Info, Data, toret)

elif argument == 'FULL':

toret = ''

Data = ''

for var in variabless:

toret += '{} = {}\r\n'.format(var, ', '.join(ClientInfo.DataBase.data.get(var, [ClientInfo.language.NULL, ])))

if len(toret) == 0:

toret = ClientInfo.language.NoValues

else:

Data = toret[:-2]

return (Inf.Info, Data, toret)

else:

return (Inf.Error, 'ARG', ClientInfo.language.NotCorrectArgument)

# Создать базу данных

def COMCREATE(ClientInfo:ClientInformation, Database):

if len(ClientInfo.CurrentUser.OwnerDataBases) >= ClientInfo.CurrentUser.MaxCountCreateDataBases and ClientInfo.CurrentUser.MaxCountCreateDataBases != -1 and ClientInfo.CurrentUser.UserRole != variables.ADMINISTRATORROLE:

return (Inf.Error, 'MAXDB', ClientInfo.language.MaximumOfDataBases.format(ClientInfo.CurrentUser.MaxCountCreateDataBases))

Database = Database.upper()

if not funs.CheckCorrectNameVariable(Database):

return (Inf.Error, 'NAME', ClientInfo.language.NotCorrectNameOfDatabase)

message = database.CreateDB(Database)

if message == 'CreateBD':

sendinfo.SendToAllListeners('ListenerAllCreate', [Database, ])

ClientInfo.CurrentUser.DataBases[Database] = classes.AccessDataBase(True, True, True, True, True, True, True)

ClientInfo.CurrentUser.OwnerDataBases.append(Database)

return (Inf.OK, Database, ClientInfo.language[message].format(Database))

elif message == 'BDExists':

return (Inf.Error, 'EXISTS', ClientInfo.language[message].format(Database))

# Удалить базу данных

def COMDBDEL(ClientInfo:ClientInformation, Database):

Database = Database.upper()

if Database == 'GLOBAL':

return (Inf.Error, 'MAIN', ClientInfo.language.BDCantDeleteMainBase)

if not funs.CheckCorrectNameVariable(Database):

return (Inf.Error, 'NAME', ClientInfo.language.NotCorrectNameOfDatabase)

if Database not in database.DATABASES:

return (Inf.Error, 'NOTEXISTS', ClientInfo.language.BDNotExists.format(Database))

if not funs.CheckCommandDataBase(ClientInfo, 'DBDEL', Database):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

DataBase = database.DATABASES[Database]

if ClientInfo.DataBase == DataBase:

return (Inf.Error, 'CURRENT', ClientInfo.language.BDCantDeleteCurrent)

if len(database.GetClientsInDB(DataBase)) > 0:

return (Inf.Error, 'USERS', ClientInfo.language.BDCantDeleteWhereUsers)

message = database.DeleteDB(Database)

if message == 'BDDeleted':

sendinfo.SendToAllListeners('ListenerAllDelete', [Database, ])

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language[message].format(Database))

elif message == 'BDNotExists':

return (Inf.Error, 'NOTEXISTS', ClientInfo.language[message].format(Database))

# Очистить базу данных

def COMDBCLEAR(ClientInfo:ClientInformation, Database):

Database = Database.upper()

if not funs.CheckCorrectNameVariable(Database):

return (Inf.Error, 'NAME', ClientInfo.language.NotCorrectNameOfDatabase)

if Database not in database.DATABASES:

return (Inf.Error, 'NOTEXISTS', ClientInfo.language.BDNotExists.format(Database))

if not funs.CheckCommandDataBase(ClientInfo, 'DBCLEAR', Database):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

DataBase = database.DATABASES[Database]

DataBase.data = dict()

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.BDCleared.format(Database))

# Показать количество подключений к базе данных

def COMCOUNTCONNECTIONSSDB(ClientInfo:ClientInformation, \*args):

Database = args[0].upper() if len(args) > 0 else ClientInfo.DataBase.name

if Database not in database.DATABASES:

return (Inf.Error, 'NOTEXISTS', ClientInfo.language.BDNotExists.format(Database))

if not funs.CheckCommandDataBase(ClientInfo, 'COUNTUSERSDB', Database):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

Data = len(database.GetClientsInDB(database.DATABASES[Database]))

return (Inf.Info, str(Data), '{}\r\n'.format(Data))

# Показать количество переменных в базе данных

def COMCOUNTVARS(ClientInfo:ClientInformation, \*args):

DataBase = ClientInfo.DataBase

if not funs.CheckCommandDataBase(ClientInfo, 'COUNTVARS'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

Data = str(len(DataBase.data.keys()))

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Показать количество значений в переменных в базе данных

def COMLENVARIABLES(ClientInfo:ClientInformation, \*args):

if not funs.CheckCommandDataBase(ClientInfo, 'LENVARIABLES'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

result = ''

DataBase = ClientInfo.DataBase

LenOfVariables = [len(value) for value in DataBase.data.values()]

DictLenOfVariables = dict()

for count in LenOfVariables:

if count in DictLenOfVariables:

DictLenOfVariables[count] += 1

else:

DictLenOfVariables[count] = 1

for count in sorted(DictLenOfVariables.keys()):

result += '{}: {}\r\n'.format(count, DictLenOfVariables[count])

Data = result[:-2]

return (Inf.Info, Data, result)

# Проверить на существование переменных в базе данных

def COMEXISTS(ClientInfo:ClientInformation, \*args):

if not funs.CheckCommandDataBase(ClientInfo, 'EXISTS'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

DataBase = ClientInfo.DataBase

count = 0

for key in args:

if key in DataBase.data:

count+=1

Data = str(count)

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Добавить значение к переменной в базе данных

def COMAPPEND(ClientInfo:ClientInformation, variable, value, position=None):

if not funs.CheckCommandDataBase(ClientInfo, 'APPEND'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

if not funs.CheckCorrectNameVariable(variable):

return (Inf.Error, 'NAME', ClientInfo.language.NotCorrectNameOfVariable)

DataBase = ClientInfo.DataBase

if variable in DataBase.data:

if position == None:

position = len(DataBase.data[variable])

else:

try:

position = int(position)

except:

return (Inf.Error, 'NOTNUMPOS', ClientInfo.language.NotNumericWherePosition)

if position < 0 or position > len(DataBase.data[variable]):

return (Inf.Error, 'INDEXOUTOFRANGE', ClientInfo.language.IndexOutOfRange)

DataBase.data[variable].insert(position, value)

else:

if position == None:

position = 0

else:

try:

position = int(position)

except:

return (Inf.Error, 'NOTNUMPOS', ClientInfo.language.NotNumericWherePosition)

if position != 0:

return (Inf.Error, 'INDEXOUTOFRANGE', ClientInfo.language.IndexOutOfRange)

DataBase.data[variable] = [value, ]

sendinfo.SendToListeners(DataBase, 'ListenerAppend', [value, variable, position])

Data = str(len(DataBase.data[variable]))

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Переименовать переменную в базе данных

def COMRENAME(ClientInfo:ClientInformation, var1, var2):

if not funs.CheckCommandDataBase(ClientInfo, 'RENAME'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

if not funs.CheckCorrectNameVariable(var1) or not funs.CheckCorrectNameVariable(var2):

return (Inf.Error, 'NAME', ClientInfo.language.NotCorrectNameOfVariable)

DataBase = ClientInfo.DataBase

if var1 in DataBase.data:

DataBase.data[var2] = DataBase.data[var1]

del DataBase.data[var1]

else:

return (Inf.Error, 'VARNOTFOUND', ClientInfo.language.VariableNotFound.format(var1))

sendinfo.SendToListeners(ClientInfo.DataBase, 'ListenerRename', [var1, var2])

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.OK)

# Начать прослушивание базы данных

def COMLISTEN(ClientInfo:ClientInformation):

if not funs.CheckCommandDataBase(ClientInfo, 'LISTEN'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

if not ClientInfo.Listen and not ClientInfo.ListenAll:

ClientInfo.Listen = True

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.StartListening.format(ClientInfo.DataBase.name))

return (Inf.Error, 'CURRENTLISTENING', ClientInfo.language.CurrentListening)

# Получить количество слушателей в базе данных

def COMLISTENERSDB(ClientInfo:ClientInformation):

if not funs.CheckCommandDataBase(ClientInfo, 'LISTENERSDB'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

Total = 0

for Client in database.GetClientsInDB(ClientInfo.DataBase):

if Client.Listen:

Total += 1

continue

Data = str(Total)

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Увеличить на 1 значение в переменной

def COMINCR(ClientInfo:ClientInformation, variable, position=0):

if not funs.CheckCommandDataBase(ClientInfo, 'INCR'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

DataBase = ClientInfo.DataBase

if position != 0:

try:

position = int(position)

except:

return (Inf.Error, 'NOTNUMPOS', ClientInfo.language.NotNumericWherePosition)

if variable in DataBase.data:

if position < 0 or position >= len(DataBase.data[variable]):

return (Inf.Error, 'INDEXOUTOFRANGE', ClientInfo.language.IndexOutOfRange)

try:

DataBase.data[variable][position] = str(int(DataBase.data[variable][position]) + 1)

except:

return (Inf.Error, 'VARNOTNUMBER', ClientInfo.language.VariableNotNumber)

sendinfo.SendToListeners(ClientInfo.DataBase, 'ListenerAdd', [variable, 1, position, DataBase.data[variable][position]])

return (Inf.Info, ClientInfo.language.NA, ClientInfo.language.OK)

else:

return (Inf.Error, 'VARNOTFOUND', ClientInfo.language.NotFound)

# Уменьшить на 1 значение в переменной

def COMDECR(ClientInfo:ClientInformation, variable, position=0):

if not funs.CheckCommandDataBase(ClientInfo, 'DECR'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

DataBase = ClientInfo.DataBase

if position != 0:

try:

position = int(position)

except:

return (Inf.Error, 'NOTNUMPOS', ClientInfo.language.NotNumericWherePosition)

if variable in DataBase.data:

if position < 0 or position >= len(DataBase.data[variable]):

return (Inf.Error, 'INDEXOUTOFRANGE', ClientInfo.language.IndexOutOfRange)

try:

DataBase.data[variable][position] = str(int(DataBase.data[variable][position]) - 1)

except:

return (Inf.Error, 'VARNOTNUMBER', ClientInfo.language.VariableNotNumber)

sendinfo.SendToListeners(ClientInfo.DataBase, 'ListenerSub', [variable, 1, position, DataBase.data[variable][position]])

return (Inf.Info, ClientInfo.language.NA, ClientInfo.language.OK)

else:

return (Inf.Error, 'VARNOTFOUND', ClientInfo.language.NotFound)

# Получить роль пользователя в СУБД

def COMGETUSERROLE(ClientInfo:ClientInformation, login):

if login not in usersdb.USERS:

return (Inf.Error, 'USERNOTFOUND', ClientInfo.language.UserIsNotExists)

return (Inf.OK, usersdb.USERS[login].UserRole, '{}\r\n'.format(usersdb.USERS[login].UserRole))

# Увеличить на value значение в переменной

def COMADD(ClientInfo:ClientInformation, variable, value, position=0):

if not funs.CheckCommandDataBase(ClientInfo, 'ADD'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

DataBase = ClientInfo.DataBase

try:

value = int(value)

except:

return (Inf.Error, 'VALUENOTNUMBER', ClientInfo.language.ValueNotNumber)

if position != 0:

try:

position = int(position)

except:

return (Inf.Error, 'NOTNUMPOS', ClientInfo.language.NotNumericWherePosition)

if variable in DataBase.data:

if position < 0 or position >= len(DataBase.data[variable]):

return (Inf.Error, 'INDEXOUTOFRANGE', ClientInfo.language.IndexOutOfRange)

try:

DataBase.data[variable][position] = str(int(DataBase.data[variable][position]) + value)

except:

return (Inf.Error, 'VARNOTNUMBER', ClientInfo.language.VariableNotNumber)

sendinfo.SendToListeners(ClientInfo.DataBase, 'ListenerAdd', [variable, value, position, DataBase.data[variable][position]])

return (Inf.Info, ClientInfo.language.NA, ClientInfo.language.OK)

else:

return (Inf.Error, 'VARNOTFOUND', ClientInfo.language.NotFound)

# Уменьшить на value значение в переменной

def COMSUB(ClientInfo:ClientInformation, variable, value, position=0):

if not funs.CheckCommandDataBase(ClientInfo, 'SUB'):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

DataBase = ClientInfo.DataBase

try:

value = int(value)

except:

return (Inf.Error, 'VALUENOTNUMBER', ClientInfo.language.ValueNotNumber)

if position != 0:

try:

position = int(position)

except:

return (Inf.Error, 'NOTNUMPOS', ClientInfo.language.NotNumericWherePosition)

if variable in DataBase.data:

if position < 0 or position >= len(DataBase.data[variable]):

return (Inf.Error, 'INDEXOUTOFRANGE', ClientInfo.language.IndexOutOfRange)

try:

DataBase.data[variable][position] = str(int(DataBase.data[variable][position]) - value)

except:

return (Inf.Error, 'VARNOTNUMBER', ClientInfo.language.VariableNotNumber)

sendinfo.SendToListeners(ClientInfo.DataBase, 'ListenerSub', [variable, value, position, DataBase.data[variable][position]])

return (Inf.Info, ClientInfo.language.NA, ClientInfo.language.OK)

else:

return (Inf.Error, 'VARNOTFOUND', ClientInfo.language.NotFound)

# Начать использовать базу данных

def COMUSE(ClientInfo:ClientInformation, Database):

Database = Database.upper()

if ClientInfo.Listen:

return (Inf.Error, 'CANTUSEWHENLISTENING', ClientInfo.language.CantUseWhenListening)

if not funs.CheckCorrectNameVariable(Database):

return (Inf.Error, 'NAME', ClientInfo.language.NotCorrectNameOfDatabase)

if Database in database.DATABASES:

if not funs.CheckCommandDataBase(ClientInfo, 'USE', Database):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

ClientInfo.DataBase = database.DATABASES[Database]

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.OK)

else:

return (Inf.Error, 'NOTEXISTS', ClientInfo.language.BDNotExists.format(Database))

# Получить общее количество слушателей из всех баз данных

def COMLISTENERS(ClientInfo:ClientInformation):

Data = dict()

AllListeners = 0

ListenersFromDatabases = ''

Total = 0

for Client in database.ALLCLIENTS:

if Client.ListenAll:

AllListeners += 1

Total += 1

continue

if Client.Listen:

if Client.DataBase.name not in Data:

Data[Client.DataBase.name] = 0

Data[Client.DataBase.name] = Data[Client.DataBase.name] + 1

Total += 1

continue

for name in sorted(Data.keys()):

ListenersFromDatabases += '{}: {}\r\n'.format(name, Data[name])

Data = '{}\r\n{}\r\n{}'.format(AllListeners, ListenersFromDatabases, Total)

return (Inf.Info, Data, ClientInfo.language.Listeners.format(AllListeners, ListenersFromDatabases, Total))

# Показать общее количество переменных в базах данных

def COMCOUNTVARSALL(ClientInfo:ClientInformation):

count = 0

for Database in sorted(database.DATABASES.keys()):

DataBase = database.DATABASES[Database]

count += len(DataBase.data.keys())

Data = str(count)

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Вывести список баз данных на сервере

def COMDBALL(ClientInfo:ClientInformation):

return (Inf.Info,', '.join(sorted(database.DATABASES.keys())), ', '.join(sorted(database.DATABASES.keys())) + '\r\n')

# Сохранить базы данных на сервере

def COMSAVE(ClientInfo:ClientInformation):

database.SaveALL()

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.Saved)

# Вывести текущую базу данных

def COMCURRENTDB(ClientInfo:ClientInformation):

Data = ClientInfo.DataBase.name

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Вывести количество пользователей в базе данных

def COMCOUNTUSERS(ClientInfo:ClientInformation):

Data = str(len(usersdb.USERS))

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Получить информацию о текущих соединениях

def COMGETCONNECTIONS(ClientInfo:ClientInformation):

Data = []

for con in database.ALLCLIENTS:

Data.append('{}:{}'.format(\*con.addr))

Data.append(', ')

Data.append(con.CurrentUser.login)

Data.append(', ')

Data.append(con.DataBase.name)

Data.append('\r\n')

return (Inf.Info, ''.join(Data[:-1]), ''.join(Data))

# Вывести текущее количество соединений с сервером

def COMCOUNTCONNECTIONS(ClientInfo:ClientInformation):

Data = str(len(database.ALLCLIENTS))

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Вывести количество переменных в базах данных

def COMCOUNTVARSALLDB(ClientInfo:ClientInformation, \*args):

result = ''

for Database in sorted(database.DATABASES.keys()):

DataBase = database.DATABASES[Database]

result += '{}: {}\r\n'.format(Database, len(DataBase.data.keys()))

Data = result[:-2]

return (Inf.Info, Data, result)

# Вывести количество значений в переменных из всех баз данных

def COMLENVARIABLESALL(ClientInfo:ClientInformation, \*args):

result = ''

DictLenOfVariables = dict()

for Database in sorted(database.DATABASES.keys()):

DataBase = database.DATABASES[Database]

LenOfVariables = [len(value) for value in DataBase.data.values()]

for count in LenOfVariables:

if count in DictLenOfVariables:

DictLenOfVariables[count] += 1

else:

DictLenOfVariables[count] = 1

for count in sorted(DictLenOfVariables.keys()):

result += '{}: {}\r\n'.format(count, DictLenOfVariables[count])

Data = result[:-2]

return (Inf.Info, Data, result)

# Вывести полученную строку

def COMECHO(ClientInfo:ClientInformation, value):

Data = value

return (Inf.OK, Data, '{}\r\n'.format(Data))

# Остановить любое прослушивание

def COMSTOP(ClientInfo:ClientInformation):

if ClientInfo.Listen or ClientInfo.ListenAll:

if ClientInfo.Listen:

ClientInfo.Listen = False

return (Inf.OK, ClientInfo.DataBase.name, ClientInfo.language.StopListening.format(ClientInfo.DataBase.name))

if ClientInfo.ListenAll:

ClientInfo.ListenAll = False

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.StopListening.format(''))

return (Inf.Info, 'NOTLISTENING', ClientInfo.language.YouAreNotListening)

# Выдать информацию о памяти и CPU

def COMINFO(ClientInfo:ClientInformation):

try:

import psutil

MemoryUse = psutil.virtual\_memory()[3]/2.\*\*20

TotalMemory = psutil.virtual\_memory()[0]/2.\*\*20

CPUUsed = psutil.cpu\_percent(interval=None)

except:

MemoryUse = 0

TotalMemory = 0

CPUUsed = 0

now = int(time.time())

Sec = database.QUERYTIME.get(now,0)

Min = database.QUERYTIMEMIN.get(now//60,0)

Uptime = now - database.TIMESTART

Data = 'MEMUSE: {}\r\nMEMTOT: {}\r\nCPU: {}\r\nSEC: {}\r\nMIN: {}\r\nTOTAL: {}\r\nTIME: {}'.format(MemoryUse, TotalMemory, CPUUsed, Sec, Min, database.QUERYES, Uptime)

Data2 = ClientInfo.language.InfoCommand.format(MemoryUse, TotalMemory, CPUUsed, Sec, Min, database.QUERYES, Uptime)

return (Inf.Info, Data, Data2)

# Начать прослушивание всех изменений в бд

def COMLISTENALL(ClientInfo:ClientInformation):

if ClientInfo.Listen or ClientInfo.ListenAll:

return (Inf.Info, 'CURRENTLISTENING', ClientInfo.language.CurrentListening)

ClientInfo.ListenAll = True

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.StartListeningAll)

# Произвести отключение от сервера

def COMQUIT(ClientInfo:ClientInformation):

ClientInfo.VisualClient = False

try:

ClientInfo.socket.close()

except:

pass

try:

database.ALLCLIENTS.remove(ClientInfo)

print(language.SERVERLANG.ClientDisconnected.format(\*ClientInfo.addr))

except:

pass

# Вывести список существующих команд на сервере

def COMLSALL(ClientInfo:ClientInformation):

Data = ', '.join(sorted(COMMANDS.keys()))

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Вывести список доступных команд пользователю

def COMLS(ClientInfo:ClientInformation):

if ClientInfo.CurrentUser.UserRole == usersdb.ADMINISTRATOR.UserRole:

Data = ', '.join(sorted(COMMANDS.keys()))

else:

Data = ', '.join(sorted(set(ClientInfo.CurrentUser.UserCommands) | set(access.ACCESSLEVEL[ClientInfo.CurrentUser.UserRole])))

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Вывести общее количество команд на сервере

def COMCOUNTCOMMANDS(ClientInfo:ClientInformation):

Data = str(len(COMMANDS.keys()))

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Выключить сервер удаленно

def COMSHUTDOWN(ClientInfo:ClientInformation):

ClientInfo.VisualClient = False

funs.CloseTheProgramm()

# Перезагрузить сервер удаленно

def COMRELOAD(ClientInfo:ClientInformation):

ClientInfo.VisualClient = False

funs.CloseTheProgramm(Reload=True)

# Очистка экрана для клиента (вывод 50 пустых строк)

def COMCLEAR(ClientInfo:ClientInformation):

return (Inf.Info, ClientInfo.language.NA, '\r\n' \* 50)

# Исполнение команды python

def COMEVAL(ClientInfo:ClientInformation, command):

try:

Data = str(eval(command))

return (Inf.Info, Data, '{}\r\n'.format(Data))

except Exception as ex:

return (Inf.Error,ex.args[0], '{}\r\n'.format(ex.args[0]))

# Исполнение команды операционной системы

def COMEXEC(ClientInfo:ClientInformation, command):

if ClientInfo.VisualClient:

return (Inf.Error, 'NOTAVAIBLEINVISUALMODE', '\r\n')

try:

output = subprocess.check\_output(command, shell=True)

if len(output) == 0:

output = '\r\n'.encode()

ClientInfo.socket.send(output)

except Exception as ex:

ClientInfo.socket.send('{}\r\n'.format(ex.args[0]).encode())

# Вывести список возможных языков на сервере

def COMLANGS(ClientInfo:ClientInformation):

Data = ', '.join(sorted(LANGUAGE.keys()))

return (Inf.OK, Data, '{}\r\n'.format(Data))

# Установка языка для клиента

def COMSETLANG(ClientInfo:ClientInformation, language):

language = language.upper()

if language not in LANGUAGE.keys():

return (Inf.Error, 'INCORRECTLANG', ClientInfo.language.IncorrectLanguage)

ClientInfo.language = LANGUAGE[language]

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.ChooseLanguage)

# Вывод информации о программе

def COMABOUT(ClientInfo:ClientInformation):

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.AboutProgram)

# Регистрация пользователя

def COMREGISTER(ClientInfo:ClientInformation, Login, Password, ConfirmPassword):

if len(Login) > 15:

return (Inf.Error, '15', ClientInfo.language.LoginLength)

if not variables.REGISTRATION and ClientInfo.CurrentUser.UserRole != variables.ADMINISTRATORROLE:

return (Inf.Error, 'REGISTERNOTAVAIBLE', ClientInfo.language.RegistrationIsNotAvaible)

if Password != ConfirmPassword:

return (Inf.Error, 'PASSWORDSDONOTMATCH', ClientInfo.language.PasswordsDoNotMatch)

HashPassword = funs.EncryptString(Password)

RegUser = classes.User(Login, HashPassword, variables.REGISTRATIONROLE, variables.MAXIMUMCREATEDATABASE)

Complete = usersdb.CreateUser(RegUser)

Message = ClientInfo.language[Complete]

if Complete == 'NotCorrectLoginUser':

return (Inf.Error, 'NOTCORRECTLOGINUSER', Message)

if Complete == 'UserIsExists':

return (Inf.Error, 'USEREXISTS', Message)

return (Inf.OK, ClientInfo.language.NA, Message)

# Вход пользователя

def COMLOGIN(ClientInfo:ClientInformation, Login, Password):

if Login not in usersdb.USERS:

return (Inf.Error, 'USERNOTEXISTS', ClientInfo.language.UserIsNotExists)

HashPassword = funs.EncryptString(Password)

if usersdb.USERS[Login].password != HashPassword:

return (Inf.Error, 'PASSWORDSDONOTMATCH', ClientInfo.language.PasswordsDoNotMatchInDataBase)

ClientInfo.CurrentUser = usersdb.USERS[Login]

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.LoginSuccessful)

# Удаление пользователя

def COMDELUSER(ClientInfo:ClientInformation, Login):

Complete = usersdb.DeleteUser(Login)

Message = ClientInfo.language[Complete]

if Complete == 'UserIsNotExists':

return (Inf.Error, 'USERNOTEXISTS', Message)

if Complete == 'ThisUserCannotChange':

return (Inf.Error, 'CANNOTCHANGE', Message)

return (Inf.OK, ClientInfo.language.NA, Message)

# Изменение пароля

def COMCHANGEPASSWORD(ClientInfo:ClientInformation, OldPassword, Password, ConfirmPassword):

if Password != ConfirmPassword:

return (Inf.Error, 'PASSWORDSDONOTMATCH', ClientInfo.language.PasswordsDoNotMatch)

HashOldPassword = funs.EncryptString(OldPassword)

HashPassword = funs.EncryptString(Password)

if ClientInfo.CurrentUser.password != HashOldPassword:

return (Inf.Error, 'PASSWORDSDONOTMATCHINDB', ClientInfo.language.PasswordsDoNotMatchInDataBase)

Complete = usersdb.ChangePassword(ClientInfo.CurrentUser, HashPassword)

Message = ClientInfo.language[Complete]

if Complete == 'UserIsNotExists':

return (Inf.Error, 'USERNOTEXISTS', Message)

if Complete == 'ThisUserCannotChange':

return (Inf.Error, 'CANNOTCHANGE', Message)

return (Inf.OK, ClientInfo.language.NA, Message)

# Посмотреть доступные команды пользователя

def COMUSERCOMMANDS(ClientInfo:ClientInformation, Login):

if Login not in usersdb.USERS:

return (Inf.Error, 'USERNOTEXISTS', ClientInfo.language.UserIsNotExists)

User = usersdb.USERS[Login]

RoleCommands = set(access.ACCESSLEVEL[User.UserRole])

AddedCommands = set(User.UserCommands)

Commands = RoleCommands | AddedCommands

Data = ', '.join(sorted(Commands))

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Посмотреть информацию о пользователе

def COMGETUSERINFO(ClientInfo:ClientInformation, Login):

if Login not in usersdb.USERS:

return (Inf.Error, 'USERNOTEXISTS', ClientInfo.language.UserIsNotExists)

User = usersdb.USERS[Login]

Data = 'Login: {}\r\nPassword: {}\r\nRole: {}\r\nMaxCount: {}\r\nUserComm: {}\r\nDatabases: {}\r\nOwnerDatabases: {}'.format(

User.login, User.password, User.UserRole, User.MaxCountCreateDataBases,

', '.join(sorted(set(access.ACCESSLEVEL[User.UserRole]) | set(User.UserCommands))), ', '.join(User.DataBases.keys()),

', '.join(User.OwnerDataBases))

return (Inf.Info, Data, ClientInfo.language.GetUserInfo.format(User.login, User.password, User.UserRole, User.MaxCountCreateDataBases, ', '.join(User.UserCommands), ', '.join(User.DataBases.keys()), ', '.join(User.OwnerDataBases)))

# Получить информацию о себе

def COMGETMYINFO(ClientInfo:ClientInformation):

return COMGETUSERINFO(ClientInfo, ClientInfo.CurrentUser.login)

# Получить пользователей базы данных с доступами

def COMGETUSERSDB(ClientInfo:ClientInformation, \*args):

Database = ClientInfo.DataBase.name

if len(args) > 0:

Database = args[0].upper()

if Database not in database.DATABASES:

return (Inf.Error, 'DBNOTFOUND', ClientInfo.language.BDNotExists.format(Database))

Database = database.DATABASES[database].name

if not funs.CheckCommandDataBase(ClientInfo, 'GETUSERSDB', Database):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

Data = []

for user in usersdb.USERS.values():

if Database in user.DataBases:

Data.append('{}: {}'.format(user.login, ', '.join([i for i in dir(user.DataBases[Database]) if not i.startswith('\_') and getattr(user.DataBases[Database], i)])))

Data.append('\r\n')

return (Inf.Info, ''.join(Data[:-1]), ''.join(Data))

# Получить информацию о доступах в базы данных

def COMGETACCESSDB(ClientInfo:ClientInformation):

Data = []

for (Database, Access) in ClientInfo.CurrentUser.DataBases.items():

Data.append('{}: {}'.format(Database, ', '.join([i for i in dir(Access) if not i.startswith('\_') and getattr(Access, i)])))

Data.append('\r\n')

return (Inf.Info, ''.join(Data[:-1]), ''.join(Data))

# Получить список ролей

def COMGETROLES(ClientInfo:ClientInformation):

Data = ', '.join(access.ACCESSLEVEL.keys())

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Получить список комманд роли

def COMGETROLECOMMANDS(ClientInfo:ClientInformation, Role):

if Role not in access.ACCESSLEVEL:

return (Inf.Error, 'NOTFOUND', ClientInfo.language.NotFound)

Data = ', '.join(access.ACCESSLEVEL)

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Получить стандартных пользователей

def COMGETSTANDARTUSERS(ClientInfo:ClientInformation):

Data = '{} - {}, {}\r\n{} - {}, {}'.format('ADMINISTRATOR', variables.ADMINISTRATORNAME, variables.ADMINISTRATORPASSWORD, 'DEFAULT', variables.DEFAULTNAME, variables.DEFAULTROLE)

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Изменить комманды пользователя

def COMCHANGEUSERCOMMANDS(ClientInfo:ClientInformation, Login, \*args):

if Login not in usersdb.USERS:

return (Inf.Error, 'USERNOTEXISTS', ClientInfo.language.UserIsNotExists)

User = usersdb.USERS[Login]

Commands = []

for com in args:

com = com.upper()

if com not in COMMANDS:

return (Inf.Error, 'COMMANDNOTFOUND', ClientInfo.language.CommandNotFound)

Commands.append(com)

User.UserCommands = Commands

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.OK)

# Список всех пользователей в базе данных

def COMUSERS(ClientInfo:ClientInformation):

Data = ', '.join(sorted(usersdb.USERS.keys()))

return (Inf.Info, Data, '{}\r\n'.format(Data))

# Изменить роль пользователя

def COMCHANGEROLE(ClientInfo:ClientInformation, Login, Role):

if Login not in usersdb.USERS:

return (Inf.Error, 'USERNOTEXISTS', ClientInfo.language.UserIsNotExists)

User = usersdb.USERS[Login]

if Role not in access.ACCESSLEVEL:

return (Inf.Error, 'ROLENOTFOUND', ClientInfo.language.RoleNotFound)

User.UserRole = Role

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.OK)

# Изменить настройки пользователя

def COMCHANGEUSER(ClientInfo: ClientInformation, Login, NewLogin, NewPassword, NewRole, NewMaxCommands, \*Commands):

if Login not in usersdb.USERS:

return (Inf.Error, 'USERNOTEXISTS', ClientInfo.language.UserIsNotExists)

if Login != NewLogin:

if Login.upper() == ClientInfo.CurrentUser.login.upper():

return (Inf.Error, 'CANNOTYOURSELF', ClientInfo.language.CannotUseOnYourself)

if len(NewLogin) > 15:

return (Inf.Error, 'LOGINLENGTH', ClientInfo.language.LoginLength)

if NewLogin.lower() in (u.lower() for u in usersdb.USERS):

return (Inf.Error, 'NEWLOGINEXISTS', ClientInfo.language.RegistrationIsNotAvaible)

if NewPassword != usersdb.USERS[Login].password:

NewPassword = funs.EncryptString(NewPassword)

if NewRole not in access.ACCESSLEVEL:

return (Inf.Error, 'ROLENOTFOUND', ClientInfo.language.RoleNotFound)

try:

NewMaxCommands = int(NewMaxCommands)

if NewMaxCommands < 0:

NewMaxCommands = -1

except:

return (Inf.Error, 'NOTANUMBER', ClientInfo.language.VariableNotNumber)

Commands = [com.upper() for com in Commands]

for com in Commands:

if com not in COMMANDS:

return (Inf.Error, 'COMMANDNOTFOUND', ClientInfo.language.CommandNotFound)

user = usersdb.USERS[Login]

del usersdb.USERS[Login]

user.password = NewPassword

user.UserRole = NewRole

user.MaxCountCreateDataBases = NewMaxCommands

user.UserCommands = Commands

usersdb.USERS[NewLogin] = user

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.OK)

# Добавить пользователя в базу данных

def COMADDUSERTODB(ClientInfo:ClientInformation, Database, Login):

if Login not in usersdb.USERS:

return (Inf.Error, 'USERNOTEXISTS', ClientInfo.language.UserIsNotExists)

User = usersdb.USERS[Login]

if Login == ClientInfo.CurrentUser.login:

return (Inf.Error, 'CANNOTYOURSELF', ClientInfo.language.CannotUseOnYourself)

Database = Database.upper()

if Database not in database.DATABASES:

return (Inf.Error, 'DBNOTEXISTS', ClientInfo.language.BDNotExists.format(Database))

if Database in User.DataBases:

return (Inf.Info, 'USERCURRENTDB', ClientInfo.language.UserAddedToDB)

if not funs.CheckCommandDataBase(ClientInfo, 'ADDUSERTODB', Database):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

User.DataBases[Database] = classes.AccessDataBase(True, False, True, False, False, True, False)

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.OK)

# Удалить пользователя из базы данных

def COMREMOVEUSERFROMDB(ClientInfo:ClientInformation, Database, Login):

if Login not in usersdb.USERS:

return (Inf.Error, 'USERNOTEXISTS', ClientInfo.language.UserIsNotExists)

User = usersdb.USERS[Login]

if Login == ClientInfo.CurrentUser.login:

return (Inf.Error, 'CANNOTYOURSELF', ClientInfo.language.CannotUseOnYourself)

Database = Database.upper()

if Database not in database.DATABASES:

return (Inf.Error, 'DBNOTEXISTS', ClientInfo.language.BDNotExists.format(Database))

if Database not in User.DataBases:

return (Inf.Info, 'USERNOTINDB', ClientInfo.language.UserNotAddedToDB)

if not funs.CheckCommandDataBase(ClientInfo, 'REMOVEUSERFROMDB', Database):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

del User.DataBases[Database]

if Database in User.OwnerDataBases:

User.OwnerDataBases.remove(Database)

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.OK)

# Изменить у пользователя роль в базе данных

def COMCHANGEACCESS(ClientInfo:ClientInformation, Database, Login, Access, Main, Read, Write, Delete, Listen, Info):

if Login not in usersdb.USERS:

return (Inf.Error, 'USERNOTEXISTS', ClientInfo.language.UserIsNotExists)

User = usersdb.USERS[Login]

if Login == ClientInfo.CurrentUser.login:

return (Inf.Error, 'CANNOTYOURSELF', ClientInfo.language.CannotUseOnYourself)

Database = Database.upper()

if Database not in database.DATABASES:

return (Inf.Error, 'DBNOTEXISTS', ClientInfo.language.BDNotExists.format(Database))

if Database not in User.DataBases:

return (Inf.Error, 'USERNOTINDB', ClientInfo.language.UserNotAddedToDB)

if not funs.CheckCommandDataBase(ClientInfo, 'CHANGEACCESS', Database):

return (Inf.PermissionDenied, ClientInfo.language.NA, ClientInfo.language.PermissionDenied)

Access = funs.StringToBool(Access)

Main = funs.StringToBool(Main)

Read = funs.StringToBool(Read)

Write = funs.StringToBool(Write)

Delete = funs.StringToBool(Delete)

Listen = funs.StringToBool(Listen)

Info = funs.StringToBool(Info)

if None in (Access, Main, Read, Write, Delete, Listen, Info):

return (Inf.Error, 'PARAMNOTBOOL', ClientInfo.language.OneOfParametersIsNotBool)

AccessDB = classes.AccessDataBase(Access, Main, Read, Write, Delete, Listen, Info)

User.DataBases[Database] = AccessDB

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.OK)

# Включение/отключение режима визуального клиента

def COMVISUAL(ClientInfo:ClientInformation):

ClientInfo.VisualClient = not ClientInfo.VisualClient

return (Inf.OK, ClientInfo.language.NA, ClientInfo.language.OK)

# Текущий пользователь

def COMME(ClientInfo:ClientInformation):

return (Inf.Info, ClientInfo.CurrentUser.login, '{}\r\n'.format(ClientInfo.CurrentUser.login))

def COMDATE(ClientInfo:ClientInformation):

Date = str(datetime.date.today())

return (Inf.Info, Date, '{}\r\n'.format(Date))

def COMTIME(ClientInfo:ClientInformation):

Time = '{:02d}:{:02d}:{:02d}'.format(datetime.datetime.now().hour, datetime.datetime.now().minute, datetime.datetime.now().second)

return (Inf.Info, str(Time), '{}\r\n'.format(Time))

# Получить время старта сервера

def COMGETSPACE(ClientInfo:ClientInformation):

return (Inf.OK, '', '\r\n')

# Получить установленные настройки на сервере

def COMGETSETTINGS(ClientInfo:ClientInformation):

return (Inf.OK, '', '\r\n')

# Установить настройки на сервере

def COMSETSETTINGS(ClientInfo:ClientInformation):

return (Inf.OK, '', '\r\n')

# Функции для вывода сообщений в нужном виде

# Отправка сообщения пользователю с кодированием сообщения

def OUTONLY(ClientInfo:ClientInformation, Info, Data, Out):

if not ClientInfo.VisualClient:

if Info == Inf.Info:

return Out.encode(), Info

if Info == Inf.PermissionDenied or Info == Inf.Error:

return ('\033[0;31m' + str(Out) + '\033[0;37m').encode(), Info

if Info == Inf.OK:

return ('\033[0;36m' + str(Out) + '\033[0;37m').encode(), Info

else:

return funs.OutForVisual(Info, Data).encode(), Info

# Отправка сообщения пользователю без кодирования сообщения

def RETURNINFO(ClientInfo:ClientInformation, Info, Data, Out):

if not ClientInfo.VisualClient:

return Out, Info

else:

return funs.OutForVisual(Info, Data).encode(), Info

# Ничего не выводить

def DONOTHING(ClientInfo:ClientInformation, Info, Data, Out):

if ClientInfo.VisualClient:

return funs.OutForVisual(Info, Data).encode(), Info

# Ключ - имя команды, значение - (функция для команды, функция для вывода, минимальное число аргументов, максимальное число аргументов(-1 - неограничено))

COMMANDS = {

# Справочные команды

'HELP' : (COMHELP, OUTONLY, 0, 0),

'DBHELP' : (COMDBHELP, OUTONLY, 0, 0),

'VARHELP' : (COMVARHELP, OUTONLY, 0, 0),

'LISTENINGHELP' : (COMLISTENINGHELP, OUTONLY, 0, 0),

'OTHERHELP' : (COMOTHERHELP, OUTONLY, 0, 0),

'LS' : (COMLS, OUTONLY, 0, 0),

'LSALL' : (COMLSALL, OUTONLY, 0, 0),

'SYMBOLS' : (COMSYMBOLS, OUTONLY, 0, 0),

'ABOUT' : (COMABOUT, OUTONLY, 0, 0),

# Команды работы с пользователями

'REGISTER' : (COMREGISTER, OUTONLY, 3, 3),

'LOGIN' : (COMLOGIN, OUTONLY, 2, 2),

'DELUSER' : (COMDELUSER, OUTONLY, 1, 1),

'CHANGEPASSWORD' : (COMCHANGEPASSWORD, OUTONLY, 3, 3),

'USERS' : (COMUSERS, OUTONLY, 0, 0),

'ME' : (COMME, OUTONLY, 0, 0),

'CHANGEUSER' : (COMCHANGEUSER, OUTONLY, 5, -1),

'GETUSERINFO' : (COMGETUSERINFO, OUTONLY, 1, 1),

'GETMYINFO' : (COMGETMYINFO, OUTONLY, 0, 0),

'GETSTANDARTUSERS' : (COMGETSTANDARTUSERS, OUTONLY, 0, 0),

# Команды работы с базами данных

'CREATE' : (COMCREATE, OUTONLY, 1, 1),

'USE' : (COMUSE, OUTONLY, 1, 1),

'DBDEL': (COMDBDEL, OUTONLY, 1, 1),

'DBALL': (COMDBALL, OUTONLY, 0, 0),

'SAVE' : (COMSAVE, OUTONLY, 0, 0),

# Команды работы с данными

'SET' : (COMSET, OUTONLY, 2, -1),

'GET' : (COMGET, OUTONLY, 1, 1),

'DEL' : (COMDEL, OUTONLY, 1, -1),

'LEN' : (COMLEN, OUTONLY, 1, 1),

'VALUES' : (COMVALUES, OUTONLY, 2, 3),

'DBCLEAR': (COMDBCLEAR, OUTONLY, 1, 1),

'CURRENTDB' : (COMCURRENTDB, OUTONLY, 0, 0),

'EXISTS' : (COMEXISTS, OUTONLY, 1, -1),

'APPEND' : (COMAPPEND, OUTONLY, 2, 3),

'RENAME' : (COMRENAME, OUTONLY, 2, 2),

'INCR' : (COMINCR, OUTONLY, 1, 2),

'DECR' : (COMDECR, OUTONLY, 1, 2),

'ADD' : (COMADD, OUTONLY, 2, 3),

'SUB' : (COMSUB, OUTONLY, 2, 3),

# Команды прослушивания

'LISTEN' : (COMLISTEN, OUTONLY, 0, 0),

'LISTENALL' : (COMLISTENALL, OUTONLY, 0, 0),

'STOP' : (COMSTOP, OUTONLY, 0, 0),

# Команды работы с языками

'LANGS': (COMLANGS, OUTONLY, 0, 0),

'SETLANG' : (COMSETLANG, OUTONLY, 1, 1),

# Команды для разраничения прав

'GETROLES' : (COMGETROLES, OUTONLY, 0, 0),

'GETROLECOMMANDS' : (COMGETROLECOMMANDS, OUTONLY, 1, 1),

'GETUSERROLE' : (COMGETUSERROLE, OUTONLY, 1, 1),

'CHANGEROLE' : (COMCHANGEROLE, OUTONLY, 2, 2),

'USERCOMMANDS' : (COMUSERCOMMANDS, OUTONLY, 1, 1),

'CHANGEUSERCOMMANDS' : (COMCHANGEUSERCOMMANDS, OUTONLY, 1, -1),

'GETUSERSDB' : (COMGETUSERSDB, OUTONLY, 0, 1),

'ADDUSERTODB' : (COMADDUSERTODB, OUTONLY, 2, 2),

'REMOVEUSERFROMDB' : (COMREMOVEUSERFROMDB, OUTONLY, 2, 2),

'GETACCESSDB' : (COMGETACCESSDB, OUTONLY, 0, 0),

'CHANGEACCESS' : (COMCHANGEACCESS, OUTONLY, 9, 9),

# Информационные команды

'COUNTCONNECTIONSDB' : (COMCOUNTCONNECTIONSSDB, OUTONLY, 0, 1),

'COUNTVARS' : (COMCOUNTVARS, OUTONLY, 0, 0),

'COUNTUSERS' : (COMCOUNTUSERS, OUTONLY, 0, 0),

'COUNTCONNECTIONS' : (COMCOUNTCONNECTIONS, OUTONLY, 0, 0),

'COUNTCOMMANDS' : (COMCOUNTCOMMANDS, OUTONLY, 0, 0),

'COUNTLISTENERS' : (COMLISTENERS, OUTONLY, 0, 0),

'COUNTLISTENERSDB' : (COMLISTENERSDB, OUTONLY, 0, 0),

'COUNTVARSALL' : (COMCOUNTVARSALL, OUTONLY, 0, 0),

'COUNTVARSALLDB' : (COMCOUNTVARSALLDB, OUTONLY, 0, 0),

'GETCONNECTIONS' : (COMGETCONNECTIONS, OUTONLY, 0, 0),

'LENVARIABLES' : (COMLENVARIABLES, OUTONLY, 0, 0),

'LENVARIABLESALL' : (COMLENVARIABLESALL, OUTONLY, 0, 0),

'DATE' : (COMDATE, OUTONLY, 0, 0),

'TIME' : (COMTIME, OUTONLY, 0, 0),

'INFO' : (COMINFO, OUTONLY, 0, 0),

'GETSPACE' : (COMGETSPACE, OUTONLY, 0, 0),

# Команды для управления сервером

'SHUTDOWN' : (COMSHUTDOWN, DONOTHING, 0, 0),

'RELOAD' : (COMRELOAD, DONOTHING, 0, 0),

'GETSETTINGS': (COMGETSETTINGS, OUTONLY, 0, 0),

'SETSETTINGS': (COMSETSETTINGS, OUTONLY, 0, 0),

# Другие команды

'ECHO' : (COMECHO, OUTONLY, 1, 1),

'CLEAR' : (COMCLEAR, OUTONLY, 0, 0),

'QUIT' : (COMQUIT, DONOTHING, 0, 0),

'EVAL' : (COMEVAL, OUTONLY, 1, 1),

'EXEC' : (COMEXEC, DONOTHING, 1, 1),

'VISUAL' : (COMVISUAL, OUTONLY, 0, 0),

}

funs.py

import hashlib

import database

import sendinfo

import language

import classes

import variables

# Некоторые вспомогательные функции

# Функция получает все аргументы из посланной пользователем строки

# Пример: 'HELP 123 321 "Hello world!" ' -> ['HELP', '123', '321', 'Hello world!']

def GetArguments(arg:str):

if arg.count('"') % 2 != 0:

raise Exception('NotCorrectNumberQuotes')

temp = ''

arguments = []

inQuot = False

for symb in arg:

if symb == '"':

inQuot = not inQuot

elif symb != ' ' or inQuot:

temp += symb

elif symb == ' ' and temp != '':

arguments.append(temp)

temp = ''

arguments.append(temp)

return arguments

# Функция дает возможность получить ответ "Да" или "Нет"

def GetAnswer():

while True:

answer = input('[Y/n] ->').upper()

if answer == 'Y' or answer == 'N':

break

print(language.SERVERLANG.WrongAnswer)

return answer == 'Y'

# Функция для полного отключения сервера

def CloseTheProgramm(message='', Reload=False):

database.CLOSE = True

database.ISRELOADING = Reload

try:

database.AUTOSAVETIMER.cancel()

try:

sendinfo.SendToAllClients('ServerShutdown', [message])

except:

pass

for client in database.ALLCLIENTS:

try:

client.socket.close()

except:

pass

try:

database.MAINSERVERSOCKET.close()

except:

pass

database.SaveALL()

except Exception as ex:

print(str(ex))

pass

# Функция для проверки валидности переменной

def CheckCorrectNameVariable(variable:str):

return variable.isidentifier()

# Функция для проверки возможности выполнения команды в базе данных

def CheckCommandDataBase(ClientInfo:classes.ClientInformation, command:str, Database=None):

if ClientInfo.CurrentUser.UserRole == variables.ADMINISTRATORROLE:

return True

if Database == None:

Database = ClientInfo.DataBase.name

if Database not in ClientInfo.CurrentUser.DataBases:

return False

Access = ClientInfo.CurrentUser.DataBases[Database]

if Access.Main:

return True

if command in ('USE', ):

return Access.Access

if command in ('DBDEL', 'ADDUSERTODB', 'REMOVEUSERFROMDB', 'CHANGEACCESS'):

return Access.Main

if command in ('GET', 'LEN', 'VALUES', 'COUNTVARS', 'COUNTVARSALLDB', 'LENVARIABLES', 'EXISTS', ):

return Access.Read

if command in ('SET', 'APPEND', 'RENAME', 'INCR', 'DECR', 'ADD', 'SUB', ):

return Access.Write

if command in ('DEL', 'DBCLEAR', ):

return Access.Delete

if command in ('LISTEN', ):

return Access.Listen

if command in ('LISTENERSDB', 'GETUSERSDB'):

return Access.Info

return False

# Получить хэш строки

def EncryptString(HashString:str):

ShaSignature = hashlib.sha256(HashString.encode()).hexdigest()

return ShaSignature

# Перевод из str в bool

def StringToBool(String:str):

String = String.lower()

if String in ('true', 't', '1', 'y', 'yes'):

return True

if String in ('false', 'f', '0', 'n', 'no'):

return False

return None

def OutForVisual(Info:str, Data:str):

return Info + '\r\n' + Data + '\r\n'

# Приложение А2. Исходный текст программы визуального менеджера

EntryForm.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MicroBaseManager

{

public partial class EntryForm : Template

{

public static Dictionary<Button, Connection> ConnectionsButtons = new Dictionary<Button, Connection>();

public EntryForm()

{

InitializeComponent();

this.ConnectionButton.Visible = false;

this.SettingsButton.Visible = false;

LoadListConnections();

}

public void LoadListConnections()

{

try

{

using (StreamReader reader = new StreamReader("connections.dat"))

{

string line;

string[] data;

Connection conn;

while ((line = reader.ReadLine()) != null)

{

data = line.Split(new char[] { ':' });

conn = new Connection(data[0], data[1], int.Parse(data[2]));

AddConnection\_Click(conn, null);

}

}

}

catch (FileNotFoundException)

{

try

{

using (StreamWriter write = new StreamWriter("connections.dat", false))

{

}

}catch

{

}

}

catch (Exception ex)

{

MessageBox.Show("Не удалось загрузить информацию о соединениях\nПричина ошибки:\n" + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

public void SaveListConnections()

{

try

{

using (StreamWriter write = new StreamWriter("connections.dat", false))

{

foreach (Connection conn in ConnectionsButtons.Values)

{

write.WriteLine("{0}:{1}:{2}", conn.GetName(), conn.GetConnect(), conn.GetPort());

}

}

}

catch (Exception ex)

{

MessageBox.Show("Не удалось сохранить информацию о соединениях\nПричина ошибки:\n" + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void ConnectionButton\_Click(object sender, EventArgs e)

{

Button btn = null;

if (sender is ToolStripMenuItem)

{

btn = (Button)ConnectionMenu.SourceControl;

}

else

{

btn = (Button)sender;

}

Connection conn = ConnectionsButtons[btn];

if (!Database.Connect(conn))

return;

try

{

MainForm MF = new MainForm();

this.Visible = false;

MF.WindowState = FormWindowState.Maximized;

MF.ShowDialog();

if (MF.DialogResult == System.Windows.Forms.DialogResult.OK)

{

this.Close();

}

else

{

this.Visible = true;

}

}

catch { }

}

private void AddConnection\_Click(object sender, EventArgs e)

{

Connection conn = null;

bool save = false;

if (sender is Connection)

{

conn = (Connection)sender;

}

else

{

AddNewConnectionForm connectionForm = new AddNewConnectionForm();

connectionForm.ShowDialog();

if (connectionForm.DialogResult != System.Windows.Forms.DialogResult.OK)

return;

conn = connectionForm.connection;

save = true;

}

Button ConnectionButton = new Button();

ConnectionButton.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(120)))), ((int)(((byte)(120)))), ((int)(((byte)(120)))));

ConnectionButton.FlatAppearance.BorderColor = System.Drawing.Color.White;

ConnectionButton.FlatStyle = System.Windows.Forms.FlatStyle.Flat;

ConnectionButton.Image = global::MicroBaseManager.Properties.Resources.ConnectButton;

ConnectionButton.ImageAlign = System.Drawing.ContentAlignment.MiddleLeft;

ConnectionButton.Parent = PanelOfConnections;

ConnectionButton.Name = "ConnectionButton\_" + ConnectionButton.ToString();

ConnectionButton.Size = new System.Drawing.Size(250, 76);

ConnectionButton.Text = String.Format("{0}\n{1}:{2}", conn.GetName(), conn.GetConnect(), conn.GetPort());

ConnectionButton.TextAlign = System.Drawing.ContentAlignment.TopRight;

ConnectionButton.UseVisualStyleBackColor = false;

ConnectionButton.Click += new System.EventHandler(this.ConnectionButton\_Click);

ConnectionButton.MouseEnter += ConnectionButton\_MouseEnter;

ConnectionButton.MouseLeave += ConnectionButton\_MouseLeave;

ConnectionButton.MouseDown += ConnectionButton\_MouseDown;

ConnectionButton.ContextMenuStrip = ConnectionMenu;

Button SettingsButton = new Button();

SettingsButton.Parent = ConnectionButton;

SettingsButton.BackColor = Color.Transparent;

SettingsButton.FlatAppearance.BorderSize = 0;

SettingsButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;

SettingsButton.Image = global::MicroBaseManager.Properties.Resources.settingsOn;

SettingsButton.ImageAlign = System.Drawing.ContentAlignment.MiddleCenter;

SettingsButton.Location = new System.Drawing.Point(218, 45);

SettingsButton.Name = "SettingsButton\_" + SettingsButton.ToString();

SettingsButton.Size = new System.Drawing.Size(28, 28);

SettingsButton.UseVisualStyleBackColor = false;

SettingsButton.MouseEnter += new System.EventHandler(this.SettingsButton\_MouseEnter);

SettingsButton.MouseLeave += new System.EventHandler(this.SettingsButton\_MouseLeave);

SettingsButton.MouseClick += SettingsButton\_Click;

ConnectionsButtons.Add(ConnectionButton, conn);

ButtonsLocationChange(null, null);

if (save)

SaveListConnections();

}

private void ButtonsLocationChange(object sender, EventArgs e)

{

int x = 12;

int y = 12;

int margin = 20;

foreach (Button btn in ConnectionsButtons.Keys)

{

if(x + btn.Width > PanelOfConnections.Width - 30)

{

x = 12;

y += margin + btn.Height;

}

btn.Left = x;

btn.Top = y;

x += margin + btn.Width;

}

}

private void ConnectionButton\_MouseEnter(object sender, EventArgs e)

{

Button btn = (Button)sender;

btn.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(160)))), ((int)(((byte)(160)))), ((int)(((byte)(160)))));

}

private void ConnectionButton\_MouseLeave(object sender, EventArgs e)

{

Button btn = (Button)sender;

btn.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(120)))), ((int)(((byte)(120)))), ((int)(((byte)(120)))));

}

private void SettingsButton\_MouseEnter(object sender, EventArgs e)

{

Button btn = (Button)sender;

btn.Image = global::MicroBaseManager.Properties.Resources.settingsOff;

btn.UseVisualStyleBackColor = false;

btn.BackColor = Color.Transparent;

}

private void SettingsButton\_MouseLeave(object sender, EventArgs e)

{

Button btn = (Button)sender;

btn.Image = global::MicroBaseManager.Properties.Resources.settingsOn;

btn.UseVisualStyleBackColor = false;

btn.BackColor = Color.Transparent;

}

private void SettingsButton\_Click(object sender, EventArgs e)

{

Button parent = null;

if (sender is ToolStripMenuItem)

{

parent = (Button)ConnectionMenu.SourceControl;

}

else

{

Button btn = (Button)sender;

parent = (Button)btn.Parent;

}

Connection conn = ConnectionsButtons[parent];

AddNewConnectionForm form = new AddNewConnectionForm(conn);

form.ShowDialog();

if (form.DialogResult != System.Windows.Forms.DialogResult.OK)

return;

conn = form.connection;

parent.Text = String.Format("{0}\n{1}:{2}", conn.GetName(), conn.GetConnect(), conn.GetPort());

ConnectionsButtons[parent] = conn;

SaveListConnections();

}

private void выходToolStripMenuItem\_Click(object sender, EventArgs e)

{

Environment.Exit(0);

}

private void ConnectionButton\_MouseDown(object sender, MouseEventArgs e)

{

}

private void проверитьСоединениеToolStripMenuItem\_Click(object sender, EventArgs e)

{

Button parent = (Button)ConnectionMenu.SourceControl;

Connection conn = ConnectionsButtons[parent];

new TestConnectionForm(conn).ShowDialog();

}

private void основныеToolStripMenuItem\_Click(object sender, EventArgs e)

{

new SettingsForm().ShowDialog();

}

private void оПрограммеToolStripMenuItem1\_Click(object sender, EventArgs e)

{

new AboutProgramForm().ShowDialog();

}

private void EntryForm\_FormClosing(object sender, FormClosingEventArgs e)

{

Environment.Exit(0);

}

private void открытьСправкуToolStripMenuItem\_Click(object sender, EventArgs e)

{

new HelpForm().ShowDialog();

}

}

}

MainForm.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Imaging;

using System.Globalization;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MicroBaseManager

{

public partial class MainForm : Template

{

public static List<string> DBList = new List<string>();

public static LinkedList<InfoClass> ServerInformation = new LinkedList<InfoClass>();

public static TabControl Tabss;

public static MainForm thisForm;

public MainForm()

{

InitializeComponent();

AddTab(typeof(ClassesTabs.TabStatus), "Статус сервера");

User.CurrentUser = new User();

SearchBox.GotFocus += RemoveText;

SearchBox.LostFocus += AddText;

UpdateDBList();

InfoTimer.Start();

Tabs.CausesValidation=true;

thisForm = this;

Tabss = this.Tabs;

}

public static void UpdateAll()

{

List<TabPage> tabpages = new List<TabPage>();

foreach (UpdateDataInterface tab in Tabss.TabPages)

{

if (tab is ClassesTabs.DB || tab is ClassesTabs.DBSettings)

{

tabpages.Add(tab as TabPage);

continue;

}

tab.UpdateData();

}

foreach (TabPage tab in tabpages)

Tabss.TabPages.Remove(tab);

thisForm.UpdateDBList();

}

private void MainForm\_FormClosing(object sender, FormClosingEventArgs e)

{

Database.Disconnect();

ServerInformation.Clear();

InfoTimer.Stop();

}

public void UpdateDBList()

{

if (SearchBox.Text != "Поиск...")

return;

DBList.Clear();

DataBasesList.Items.Clear();

Answer DB = Database.SendGetAnswer("DBALL");

if (DB.Info != Inf.OK)

return;

string[] data = DB.Data[0].Split(new string[] { ", " }, StringSplitOptions.None);

DBList.AddRange(data);

DataBasesList.Items.AddRange(data);

}

private void Tabs\_DrawItem(object sender, DrawItemEventArgs e)

{

if (e.Index >= Tabs.TabPages.Count)

return;

Font f;

Brush backBrush;

Brush foreBrush;

if (e.Index == Tabs.SelectedIndex)

{

f = new Font(this.Font, FontStyle.Bold);

backBrush = new System.Drawing.Drawing2D.LinearGradientBrush(e.Bounds, Color.FromArgb(80, 80, 80), Color.FromArgb(150, 150, 150), System.Drawing.Drawing2D.LinearGradientMode.ForwardDiagonal);

foreBrush = Brushes.Black;

}

else

{

f = e.Font;

backBrush = new SolidBrush(e.BackColor);

foreBrush = new SolidBrush(e.ForeColor);

}

string tabName = Tabs.TabPages[e.Index].Text;

StringFormat sf = new StringFormat();

e.Graphics.FillRectangle(backBrush, e.Bounds);

Rectangle r = e.Bounds;

r = new Rectangle(r.X+5, r.Y + 3, r.Width, r.Height - 3);

e.Graphics.DrawString(tabName, f, foreBrush, r, sf);

sf.Dispose();

if (e.Index == Tabs.SelectedIndex)

{

r = new Rectangle(e.Bounds.X + e.Bounds.Width - 25, e.Bounds.Y + 5, 20, 20);

Brush br = new SolidBrush(Color.FromArgb(123, 255, 0, 0));

e.Graphics.FillRectangle(br, r);

f.Dispose();

br.Dispose();

backBrush.Dispose();

}

else

{

backBrush.Dispose();

foreBrush.Dispose();

}

r = new Rectangle(e.Bounds.X + e.Bounds.Width - 25, e.Bounds.Y + 5, 20, 20);

Pen p = new Pen(Color.Black, (e.Index == Tabs.SelectedIndex)?3:1);

e.Graphics.DrawLine(p, r.X + 2, r.Y + 2, r.X + r.Width - 2, r.Y + r.Height - 2);

e.Graphics.DrawLine(p, r.X + 2, r.Y + r.Height - 2, r.X + r.Width - 2, r.Y + 2);

p.Dispose();

}

private void Tabs\_MouseDown(object sender, MouseEventArgs e)

{

if (e.Button != System.Windows.Forms.MouseButtons.Left)

return;

for (int i = 0; i < this.Tabs.TabPages.Count; i++)

{

Rectangle r = Tabs.GetTabRect(i);

Rectangle closeButton = new Rectangle(r.Right - 25, r.Top + 5, 20, 20);

if (closeButton.Contains(e.Location))

{

if (Tabs.TabPages.Count > 1 && Tabs.SelectedIndex == i)

{

if (i - 1 >= 0)

Tabs.SelectedIndex = i - 1;

else

Tabs.SelectedIndex = 0;

}

this.Tabs.TabPages.RemoveAt(i);

return;

}

if (r.Contains(e.Location))

{

page = Tabs.SelectedTab;

MouseDowned = true;

coords = Cursor.Position;

TabSize = r;

container.X = r.Right - e.Location.X;

container.Y = r.Top + e.Location.Y;

return;

}

}

}

Rectangle TabSize;

Point container;

Point coords;

bool MouseDowned = false;

OpacityForm MovingForm;

TabPage page;

private void Tabs\_MouseMove(object sender, MouseEventArgs e)

{

if (MouseDowned == false || page == null)

return;

if (Math.Abs(coords.X - Cursor.Position.X) < 10 && Math.Abs(coords.Y - Cursor.Position.Y) < 10)

return;

Bitmap map = new Bitmap(page.Bounds.Width, page.Bounds.Height);

page.Show();

page.DrawToBitmap(map, page.Bounds);

Rectangle r = Tabs.GetTabRect(0);

Bitmap TabMap = new Bitmap(TabSize.Width + TabSize.X, TabSize.Height + TabSize.Y);

Tabs.DrawToBitmap(TabMap, new Rectangle(0,0, TabMap.Width, TabMap.Height));

Graphics.FromImage(map).DrawImage(TabMap.Clone(TabSize, PixelFormat.DontCare), r.X, r.Y);

MovingForm = new OpacityForm(map, Cursor.Position.X, Cursor.Position.Y,TabSize.Width - container.X, container.Y);

int j = Tabs.TabPages.IndexOf(page);

Point pos = Cursor.Position;

if (Tabs.TabPages.Count > 1)

{

if (j - 1 >= 0)

Tabs.SelectedIndex = j - 1;

else

Tabs.SelectedIndex = 0;

}

Tabs.TabPages.Remove(page);

MovingForm.ShowDialog();

pos.X -= Cursor.Position.X;

for (int i = 0; i < this.Tabs.TabPages.Count; i++)

{

r = Tabs.GetTabRect(i);

if (r.X < e.Location.X - pos.X && r.X + r.Width > e.Location.X - pos.X)

{

Tabs.TabPages.Remove(page);

Tabs.TabPages.Insert(i, page);

Tabs.SelectedTab = page;

page = null;

MouseDowned = false;

return;

}

}

if (!Tabs.TabPages.Contains(page))

{

if (Tabs.TabPages.Count == 0)

Tabs.TabPages.Add(page);

else if(e.Location.X - pos.X < Tabs.GetTabRect(0).X)

Tabs.TabPages.Insert(0, page);

else

Tabs.TabPages.Add(page);

}

Tabs.SelectedTab = page;

page = null;

MouseDowned = false;

}

private void Tabs\_MouseUp(object sender, MouseEventArgs e)

{

MouseDowned = false;

}

public void RemoveText(object sender, EventArgs e)

{

SearchBox.Text = "";

}

public void AddText(object sender, EventArgs e)

{

if (String.IsNullOrWhiteSpace(SearchBox.Text))

SearchBox.Text = "Поиск...";

}

private void SearchBox\_TextChanged(object sender, EventArgs e)

{

if (String.IsNullOrWhiteSpace(SearchBox.Text) || SearchBox.Text == Lang.Search)

{

if (DataBasesList.Items.Count != DBList.Count)

{

DataBasesList.Items.Clear();

DataBasesList.Items.AddRange(DBList.ToArray());

}

return;

}

SearchBox.Text = SearchBox.Text.ToUpper();

int length = SearchBox.Text.Length;

List<string> Founded = new List<string>();

foreach (string db in DBList)

{

if (db.StartsWith(SearchBox.Text))

{

Founded.Add(db);

}

}

DataBasesList.Items.Clear();

DataBasesList.Items.AddRange(Founded.ToArray());

if (DataBasesList.Items.Count == 1)

DataBasesList.SelectedIndex = 0;

SearchBox.SelectionStart = length;

SearchBox.SelectionLength = 0;

}

private void SearchBox\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar == 13)

{

if (DataBasesList.SelectedIndex != -1)

{

DataBasesList\_DoubleClick(null, EventArgs.Empty);

SearchBox.Text = "";

}

}

}

private void AddTab(Type NameTab, string CreateName, object[] data=null)

{

if (Tabs.TabPages.ContainsKey(CreateName))

{

Tabs.SelectTab((TabPage)Tabs.Controls[CreateName]);

}

else

{

TabPage page;

if (data == null)

page = Activator.CreateInstance(NameTab) as TabPage;

else

page = Activator.CreateInstance(NameTab, data) as TabPage;

page.Name = CreateName;

page.Text = CreateName + " ";

Tabs.TabPages.Add(page);

Tabs.SelectTab((TabPage)Tabs.Controls[CreateName]);

}

}

private void StatusButton\_MouseEnter(object sender, EventArgs e)

{

((Button)sender).Font = new Font(((Button)sender).Font, FontStyle.Underline);

}

private void StatusButton\_MouseLeave(object sender, EventArgs e)

{

((Button)sender).Font = new Font(((Button)sender).Font, FontStyle.Regular);

}

private void StatusButton\_Click(object sender, EventArgs e)

{

AddTab(typeof(ClassesTabs.TabStatus), "Статус сервера");

}

private void UsersButton\_Click(object sender, EventArgs e)

{

AddTab(typeof(ClassesTabs.TabUsers), "Пользователи");

}

private void InformationButton\_Click(object sender, EventArgs e)

{

AddTab(typeof(ClassesTabs.TabInformation), "Информация");

}

private void Telnet\_Click(object sender, EventArgs e)

{

AddTab(typeof(ClassesTabs.TabTelnet), Lang.Console);

}

private void SettingsButton\_Click(object sender, EventArgs e)

{

AddTab(typeof(ClassesTabs.TabSettings), Lang.Configuration);

}

public void Watch\_Click(object sender, EventArgs e)

{

AddTab(typeof(ClassesTabs.TabWatch), "Слежение");

}

private void DataBasesList\_DoubleClick(object sender, EventArgs e)

{

if (DataBasesList.SelectedItem == null)

return;

string NameDB = DataBasesList.SelectedItem.ToString();

if (User.CurrentUser.Role == "Administrator" || (User.CurrentUser.Databases.ContainsKey(NameDB) && User.CurrentUser.Databases[NameDB].Read))

AddTab(typeof(ClassesTabs.DB), Lang.DB + " " + NameDB, new object[]{NameDB});

}

private void обновитьToolStripMenuItem\_Click(object sender, EventArgs e)

{

UpdateAll();

}

private void выходToolStripMenuItem\_Click(object sender, EventArgs e)

{

Environment.Exit(0);

}

private void Tabs\_ControlRemoved(object sender, ControlEventArgs e)

{

if (e.Control is ClassesTabs.TabTelnet)

((ClassesTabs.TabTelnet)(e.Control)).f.CloseSocket();

}

private void оПрограммеToolStripMenuItem1\_Click(object sender, EventArgs e)

{

new AboutProgramForm().ShowDialog();

}

private void DataBasesList\_DrawItem(object sender, DrawItemEventArgs e)

{

e.DrawBackground();

Rectangle item = e.Bounds;

Graphics g = e.Graphics;

if (e.Index < 0)

return;

string database = DataBasesList.Items[e.Index].ToString();

if (User.CurrentUser.Role == "Administrator")

{

Rectangle rect = new Rectangle(item.X + item.Width - item.Height - 4, item.Y + 2, DataBasesList.ItemHeight - 2, DataBasesList.ItemHeight - 2);

g.DrawImage(Properties.Resources.jmet, rect);

g.DrawString(database, new Font(e.Font, FontStyle.Bold), Brushes.Black, item.X + (item.Height - e.Font.Height), item.Y);

}

else

{

if (User.CurrentUser.Databases.ContainsKey(database))

{

if(User.CurrentUser.Databases[database].Read)

g.DrawString(database, new Font(e.Font,FontStyle.Bold), Brushes.Black, item.X + (item.Height - e.Font.Height), item.Y);

else

g.DrawString(database, e.Font, Brushes.Black, item.X + (item.Height - e.Font.Height), item.Y);

if (User.CurrentUser.Databases[database].Main)

{

Rectangle rect = new Rectangle(item.X + item.Width - item.Height - 4, item.Y + 2, DataBasesList.ItemHeight - 2, DataBasesList.ItemHeight - 2);

g.DrawImage(Properties.Resources.jmet, rect);

}

}

else

{

g.DrawString(database, e.Font, Brushes.Black, item.X + (item.Height - e.Font.Height), item.Y);

}

}

}

private void DataBasesList\_Click(object sender, EventArgs e)

{

if (DataBasesList.SelectedItem == null && !(e is MouseEventArgs))

return;

Point click = ((MouseEventArgs)(e)).Location;

Rectangle key = new Rectangle(DataBasesList.Width - DataBasesList.ItemHeight - 4,

DataBasesList.SelectedIndex \* DataBasesList.ItemHeight + 2,

DataBasesList.ItemHeight - 2, DataBasesList.ItemHeight - 2);

if (!key.Contains(click))

return;

string NameDB = DataBasesList.SelectedItem.ToString();

if (User.CurrentUser.Role == "Administrator" || (User.CurrentUser.Databases.ContainsKey(NameDB) && User.CurrentUser.Databases[NameDB].Main))

AddTab(typeof(ClassesTabs.DBSettings), Lang.DBSettings + " " + NameDB, new object[] { NameDB });

}

private void InfoTimer\_Tick(object sender, EventArgs args)

{

Answer answer = Database.SendGetAnswer("INFO");

if (answer.Info == Inf.OK)

{

AnswerData InfoData = answer.GetSerializedData(new string[] { "D|: ", "V" });

float a = Convert.ToSingle(InfoData["MEMUSE"].This, CultureInfo.InvariantCulture);

float b = Convert.ToSingle(InfoData["MEMTOT"].This, CultureInfo.InvariantCulture);

float c = Convert.ToSingle(InfoData["CPU"].This, CultureInfo.InvariantCulture);

if (a <= 0)

a = 1;

if (b <= 0)

b = 1;

if (c <= 0)

c = 1;

int d = Convert.ToInt32(InfoData["SEC"].This);

int e = Convert.ToInt32(InfoData["MIN"].This);

int f = Convert.ToInt32(InfoData["TOTAL"].This);

int g = Convert.ToInt32(InfoData["TIME"].This);

ServerInformation.AddLast(new InfoClass(a, b, c, d, e, f, g));

if (ServerInformation.Count > 500)

ServerInformation.RemoveFirst();

}

}

private void основныеToolStripMenuItem\_Click(object sender, EventArgs e)

{

new SettingsForm().ShowDialog();

}

private void открытьСправкуToolStripMenuItem\_Click(object sender, EventArgs e)

{

new HelpForm().ShowDialog();

}

}

}

Database.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MicroBaseManager

{

public static class Database

{

public static Socket MainConnection { get; private set; }

public static Connection CurrentConnection { get; private set; }

public static object CheckConnection(Connection conn)

{

string HOST = conn.GetConnect();

int PORT = conn.GetPort();

try

{

IPHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());

IPAddress ipAddress = Dns.GetHostAddresses(HOST)[0];

IPEndPoint remoteEP = new IPEndPoint(ipAddress, PORT);

Socket socket = new Socket(ipAddress.AddressFamily,

SocketType.Stream, ProtocolType.Tcp);

socket.Connect(remoteEP);

GetDataStringWithSocket(socket);

Answer ans = SendGetAnswerWithSocket(socket, "VISUAL");

socket.Close();

if (ans.Info != Inf.OK)

throw new Exception("На порту используется другой сервер");

return null;

}

catch (Exception ex)

{

return ex.Message;

}

}

public static Socket GetSocketFromConnect(Connection conn)

{

string HOST = conn.GetConnect();

int PORT = conn.GetPort();

IPHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());

IPAddress ipAddress = Dns.GetHostAddresses(HOST)[0];

IPEndPoint remoteEP = new IPEndPoint(ipAddress, PORT);

Socket sock;

sock = new Socket(ipAddress.AddressFamily,

SocketType.Stream, ProtocolType.Tcp);

sock.Connect(remoteEP);

GetDataStringWithSocket(sock);

return sock;

}

public static bool Connect(Connection conn)

{

try

{

MainConnection = GetSocketFromConnect(conn);

if (MainConnection == null)

return false;

Answer ans = SendGetAnswerWithSocket(MainConnection, "VISUAL");

if (ans.Info != Inf.OK)

{

MainConnection.Close();

throw new Exception("На порту используется другой сервер");

}

MainConnection.ReceiveTimeout = 5000;

CurrentConnection = conn;

return true;

}

catch (Exception ex)

{

MainConnection = null;

MessageBox.Show(String.Format("Произошла ошибка:\n{0}", ex.Message), "Соединение", MessageBoxButtons.OK, MessageBoxIcon.Error);

return false;

}

}

public static void Disconnect()

{

try

{

CurrentConnection = null;

MainConnection.Close();

MainConnection = null;

}

catch { }

}

public static long CheckConnectionPing()

{

var watch = System.Diagnostics.Stopwatch.StartNew();

SendGetAnswer("ME");

watch.Stop();

return watch.ElapsedMilliseconds;

}

public static void Reconnection()

{

while (MainConnection == null)

{

try

{

if (Database.CurrentConnection == null)

return;

Socket sock = GetSocketFromConnect(Database.CurrentConnection);

if (MainConnection == null)

{

MainConnection = sock;

}

else

{

try

{

sock.Disconnect(false);

}

catch { }

return;

}

Answer ans = SendGetAnswerWithSocket(MainConnection, "VISUAL");

if (ans.Info != Inf.OK)

{

MainConnection.Close();

throw new Exception("На порту используется другой сервер");

}

MainConnection.ReceiveTimeout = 5000;

}

catch

{}

}

}

private static Thread ReconnectionThread;

public static void TryingReconnection()

{

if (ReconnectionThread == null || ReconnectionThread.ThreadState != ThreadState.Running)

{

ReconnectionThread = new Thread(new ThreadStart(Database.Reconnection));

ReconnectionThread.Start();

}

}

public static byte[] buffer = new byte[1048576];

public static String GetDataStringWithSocket(Socket socket)

{

StringBuilder str = new StringBuilder();

int count;

lock (buffer)

{

while (true)

{

try

{

count = socket.Receive(buffer);

str.Append(Encoding.UTF8.GetString(buffer, 0, count));

if (socket.Available <= 0) break;

}

catch (SocketException ex)

{

if (ex.ErrorCode == 10060)

break;

if (ex.ErrorCode == 10054)

{

if (socket == MainConnection)

{

MainConnection = null;

TryingReconnection();

}

}

MessageBox.Show(ex.Message);

break;

}

catch (NullReferenceException)

{

TryingReconnection();

return "<DISCONNECTED>";

}

}

}

return str.ToString().TrimEnd(new char[] { '\n' }).TrimEnd(new char[] { '\r' });

}

public static bool SendDataStringWithSocket(Socket socket, string data)

{

data += "\r\n";

try

{

socket.Send(Encoding.UTF8.GetBytes(data));

return true;

}

catch (SocketException)

{

if (socket == MainConnection)

{

TryingReconnection();

MainConnection = null;

}

return false;

}

catch (NullReferenceException)

{

TryingReconnection();

return false;

}

}

public static bool SendDataString(string data)

{

return SendDataStringWithSocket(MainConnection, data);

}

public static String GetDataString()

{

return GetDataStringWithSocket(MainConnection);

}

public static Answer GetAnswerWithSocket(Socket socket)

{

return new Answer(GetDataStringWithSocket(socket));

}

public static Answer SendGetAnswerWithSocket(Socket socket, params string[] data)

{

if (socket==null)

return new Answer("<DISCONNECTED>");

if (data.Length == 0)

return new Answer("<NULL>");

Answer ans = null;

lock (socket)

{

foreach (string str in data)

{

SendDataStringWithSocket(socket, str);

ans = GetAnswerWithSocket(socket);

}

return ans;

}

}

public static Answer GetAnswer()

{

return GetAnswerWithSocket(MainConnection);

}

public static Answer SendGetAnswer(params string[] data)

{

return SendGetAnswerWithSocket(MainConnection, data);

}

}

}

Answer.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MicroBaseManager

{

public enum Inf{

OK,

Error,

PermissionDenied,

Info,

WrongAnswer

}

public class AnswerData

{

private object data;

public AnswerData(object data)

{

this.data = data;

}

public AnswerData this[object param]

{

get

{

if (data is Dictionary<string, AnswerData>)

{

return ((Dictionary<string, AnswerData>)(data))[(string)param];

}

if (data is AnswerData[])

{

return ((AnswerData[])(data))[(int)param];

}

if (data is string)

{

return this;

}

return (AnswerData)data;

}

}

public string This

{

get

{

return this.ToString();

}

}

public string[] Values

{

get

{

return GetEnumerable().Select(x => x.ToString()).ToArray();

}

}

public object GetData()

{

return data;

}

public IEnumerable<AnswerData> GetRow()

{

if (data is AnswerData)

yield return (AnswerData)(data);

else if (data is string)

yield return this;

else if (data is AnswerData[])

{

foreach (object val in ((AnswerData[])data))

{

yield return (AnswerData)val;

}

}

else if (data is Dictionary<string, AnswerData>)

{

foreach (object val in (((Dictionary<string, AnswerData>)data).Keys))

{

yield return this;

}

}

else

{

foreach (object val in ((IEnumerable)data))

{

yield return this;

}

}

}

public IEnumerable<string> GetEnumerable()

{

if (data is AnswerData)

{

foreach (string d in ((AnswerData)data).GetEnumerable())

{

yield return d;

}

yield break;

}

if (data is IEnumerable)

{

if (data is Dictionary<string, AnswerData>)

{

foreach (object val in (((Dictionary<string, AnswerData>)data).Keys))

{

yield return val.ToString();

}

}

else

{

foreach (object val in ((IEnumerable)data))

{

yield return val.ToString();

}

}

}

yield break;

}

public override string ToString()

{

if (data is AnswerData[])

{

if (((AnswerData[])(data)).Length != 0)

return ((AnswerData[])(data))[0].ToString();

return "";

}

if (data is Dictionary<string, AnswerData>)

return ((Dictionary<string, AnswerData>)(data)).Keys.First();

if (data is KeyValuePair<string, AnswerData>)

return ((KeyValuePair<string, AnswerData>)(data)).Key;

return data.ToString();

}

}

public class Answer

{

private Inf information;

private string[] data;

public Answer(Inf information, string[] data)

{

this.information = information;

this.data = data;

}

public Answer(string RawString)

{

if (RawString == "<DISCONNECTED>")

{

data = new string[] { RawString };

this.information = Inf.Error;

return;

}

string[] strings = RawString.Split(new string[] { "\r\n" }, StringSplitOptions.None);

if (strings.Length < 2)

{

this.information = Inf.WrongAnswer;

this.data = strings;

return;

}

string ans = strings[0];

switch (ans)

{

case "<OK>": this.information = Inf.OK; break;

case "<ERROR>": this.information = Inf.Error; break;

case "<PERMISSION DENIED>": this.information = Inf.PermissionDenied; break;

case "<INFO>": this.information = Inf.Info; break;

}

this.data = strings.Skip(1).ToArray();

}

public Inf Info

{

get

{

if (information == Inf.Info)

return Inf.OK;

return information;

}

}

public string[] Data

{

get

{

return data;

}

}

private AnswerData GetSerializedData(string data, string[] format=null)

{

if (format == null)

return new AnswerData(data);

string[] parser;

if (format != null && format.Length > 0)

parser = format[0].Split(new char[] { '|' });

else

parser = new string[] { "V" };

if (format.Length == 1)

format = null;

if (parser[0] == "V")

{

return new AnswerData(GetSerializedData(data, format));

}

else if (parser[0] == "L")

{

return new AnswerData(data.Split(new string[]{parser[1]}, StringSplitOptions.RemoveEmptyEntries).Select(x => GetSerializedData(x, format)).ToArray());

}

else if (parser[0] == "D")

{

/\* return new AnswerData(data.Select(delegate(string x)

{

string[] spl = x.Split(parser[1].ToCharArray());

return new { k = spl[0], v = GetSerializedData(spl[1].Split(parser[2].ToCharArray(), 2, StringSplitOptions.None), format) };

}).ToDictionary(x => x.k, x => x.v)); \*/

}

return null;

}

public AnswerData GetSerializedData(string[] format)

{

try

{

if (data.Length == 1 && data[0].Length == 0)

return new AnswerData("");

string[] parser;

if (format.Length > 0)

{

parser = format[0].Split(new char[] { '|' });

format = format.Skip(1).ToArray();

}

else

{

parser = new string[] { "V" };

}

if (parser[0] == "V")

{

return new AnswerData(GetSerializedData(data[0], format));

}

else if (parser[0] == "L")

{

return new AnswerData(data.Select(x => GetSerializedData(x, format)).ToArray());

}

else if (parser[0] == "D")

{

return new AnswerData(data.Select(delegate(string x)

{

string[] spl = x.Split(new string[] { parser[1] }, 2, StringSplitOptions.None);

return new

{

k = spl[0],

v = GetSerializedData(spl[1], format)

};

}).ToDictionary(x => x.k, x => x.v));

}

}catch(Exception ex)

{

return new AnswerData(ex.Message);

}

return new AnswerData("");

}

public string Message

{

get {

if (information == Inf.OK || information == Inf.Info)

return data[0];

if (information == Inf.PermissionDenied)

return Lang.PermissionDenied;

if (information == Inf.Error)

{

switch (data[0])

{

case "<DISCONNECTED>": return Lang.ConnectionOff;

case "COMNOTFOUND": return Lang.ComNotFoud;

default: return data[0];

}

}

return Lang.InvalidQuery;

}

}

internal void ShowMessage()

{

MessageBox.Show(this.Message, "");

}

}

}

# Приложение А3. Исходный текст программы микроконтроллера

import machine

import time

import socket

import bme280

import network

WLANSSID = 'VONETS\_1C678C'

WLANPASSWORD = '12345678'

IPADRESS = '192.168.253.'

FIRSTNUMBER = 100

NEWIP = IPADRESS + str(FIRSTNUMBER)

ADD=0

RANGE = 100

PORT = 10000

Connected = False

BLUELED = machine.Pin(2, machine.Pin.OUT)

REDLED = machine.Pin(16, machine.Pin.OUT)

BLUELED.on()

sta\_if = None

def ProblemWithConnect():

time.sleep(0.5)

BLUELED.off()

time.sleep(0.5)

BLUELED.on()

def ProblemWithBME():

while True:

time.sleep(0.5)

BLUELED.off()

time.sleep(0.5)

BLUELED.on()

def BMEConnect():

global BME

try:

i2c = machine.I2C(scl=machine.Pin(4), sda=machine.Pin(2))

BME = bme280.BME280(i2c=i2c)

except:

ProblemWithBME()

def WifiConnect():

global sta\_if

sta\_if = network.WLAN(network.STA\_IF)

sta\_if.active(True)

sta\_if.connect(WLANSSID, WLANPASSWORD)

REDLED.off()

while not sta\_if.isconnected():

pass

REDLED.on()

def GetData():

data = SOCKET.recv(1024)

udata = data.decode('utf\_8')

return udata

def GetWithout():

SOCKET.recv(1024)

def SendData(data):

data = data.encode('utf\_8')

SOCKET.send(data)

def C0nnectToServer(IP=NEWIP):

global SOCKET, Connected

print(IP)

SOCKET = socket.socket()

SOCKET.settimeout(0.3)

SOCKET.connect((IP, PORT))

SOCKET.settimeout(5)

print(GetData())

SendData('VISUAL\r\n')

if GetData()[:4] != '<OK>':

ProblemWithConnect()

print(1)

SendData('LOGIN Sender pass\r\n')

if GetData()[:4] != '<OK>':

ProblemWithConnect()

print(2)

SendData('USE METEOR\r\n')

if GetData()[:4] != '<OK>':

ProblemWithConnect()

print(3)

Connected = True

def Working():

try:

while True:

data1, data2, data3 = BME.values

# data1

# data2

# data3

BLUELED.on()

SendData('SET TEMPERATURE {}\r\n'.format(data1))

GetWithout()

SendData('SET PRESSURE {}\r\n'.format(data2))

GetWithout()

SendData('SET HUMIDITY {}\r\n'.format(data3))

GetWithout()

BLUELED.off()

except Exception as ex:

print(ex)

ProblemWithConnect()

def main():

global NEWIP, ADD, Connected

while True:

BMEConnect()

try:

WifiConnect()

C0nnectToServer(NEWIP)

Working()

except Exception as ex:

print(ex)

if not Connected:

ADD +=1

if ADD > RANGE:

ADD=0

NEWIP = IPADRESS + str(ADD + FIRSTNUMBER)

main()

# Приложение Б. Рисунки

Рисунок Б. - Концептуальная DFD диаграмма

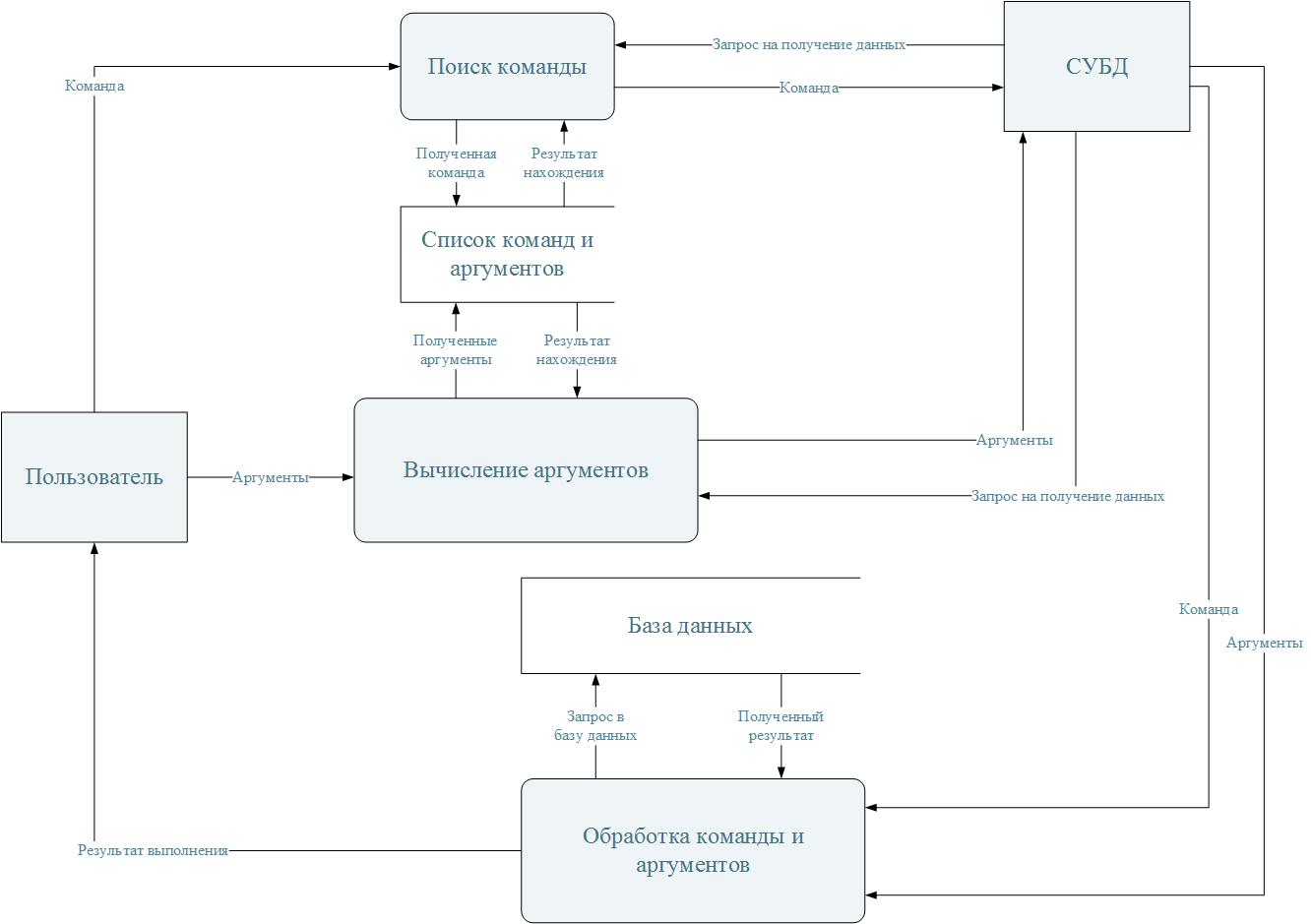


Рисунок Б. - Детализированная DFD диаграмма

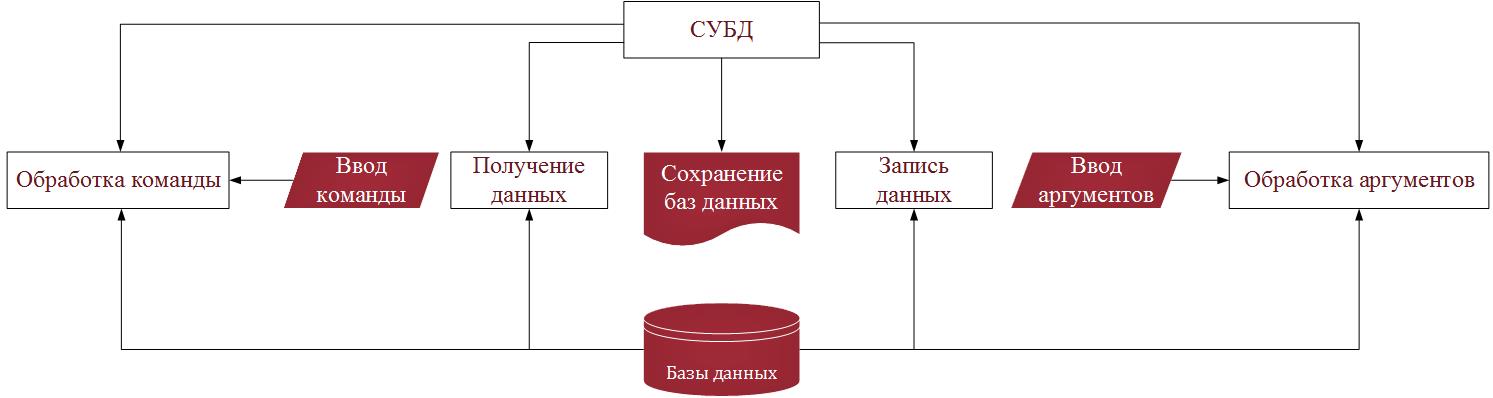


Рисунок Б. - Функциональная схема программы

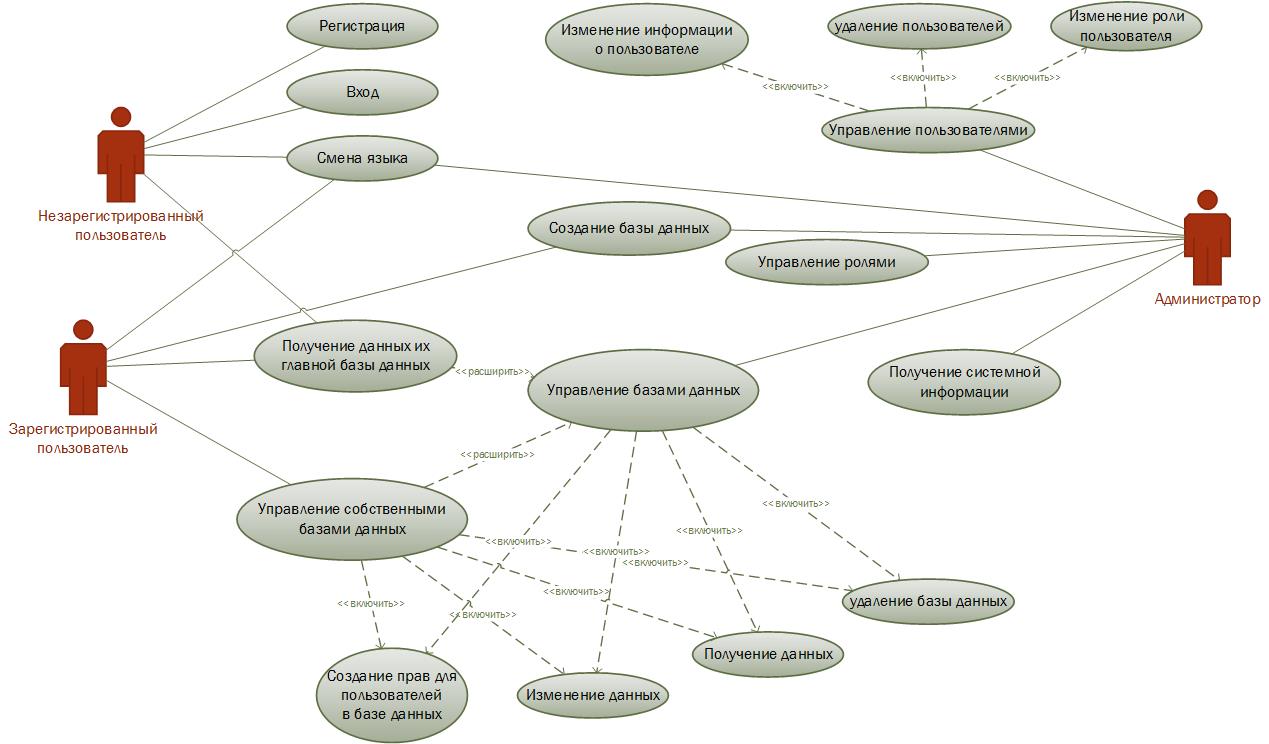
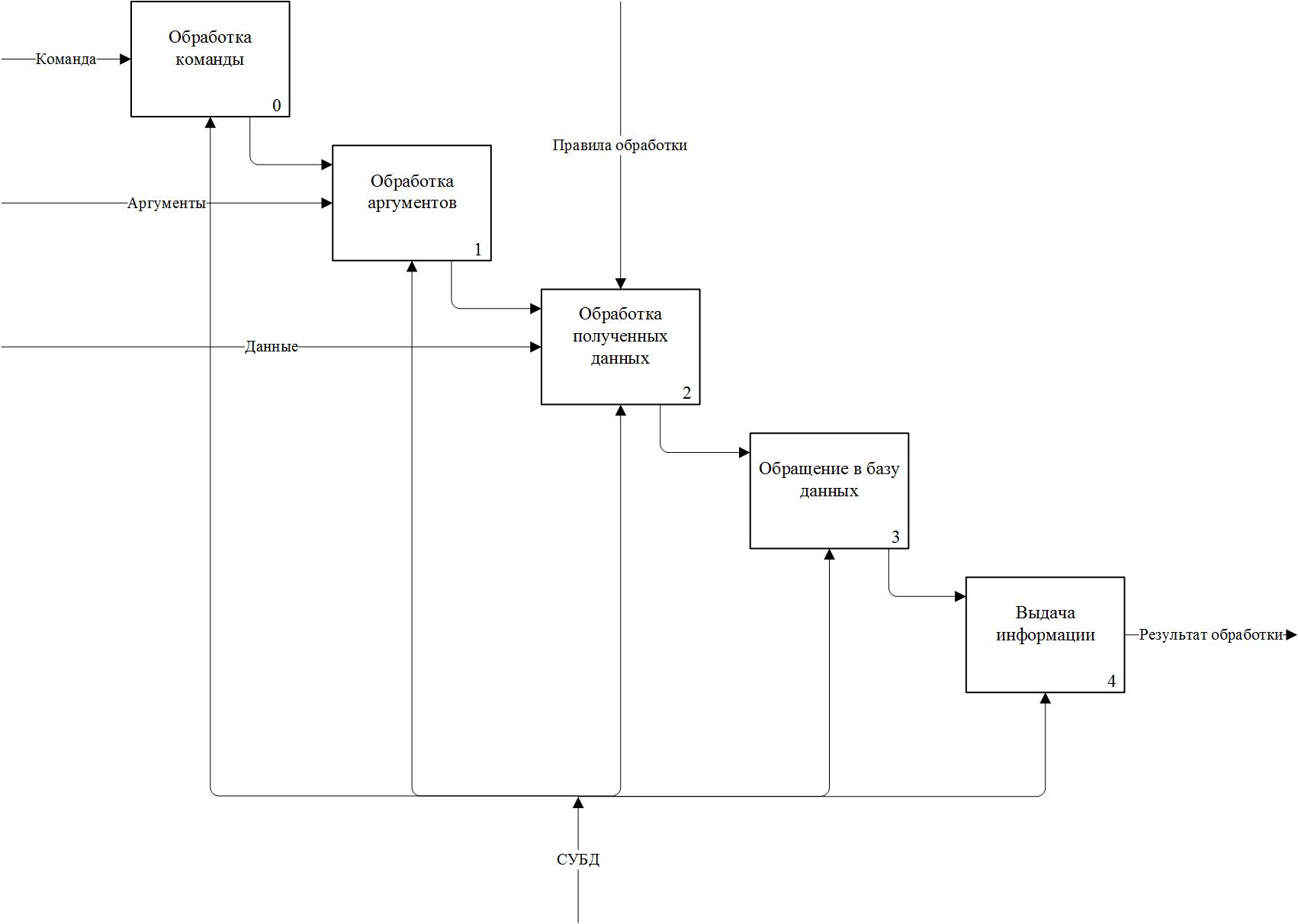


Рисунок Б.4 - Детализированная диаграмма IDEF0

Рисунок Б. - Диаграмма вариантов использования

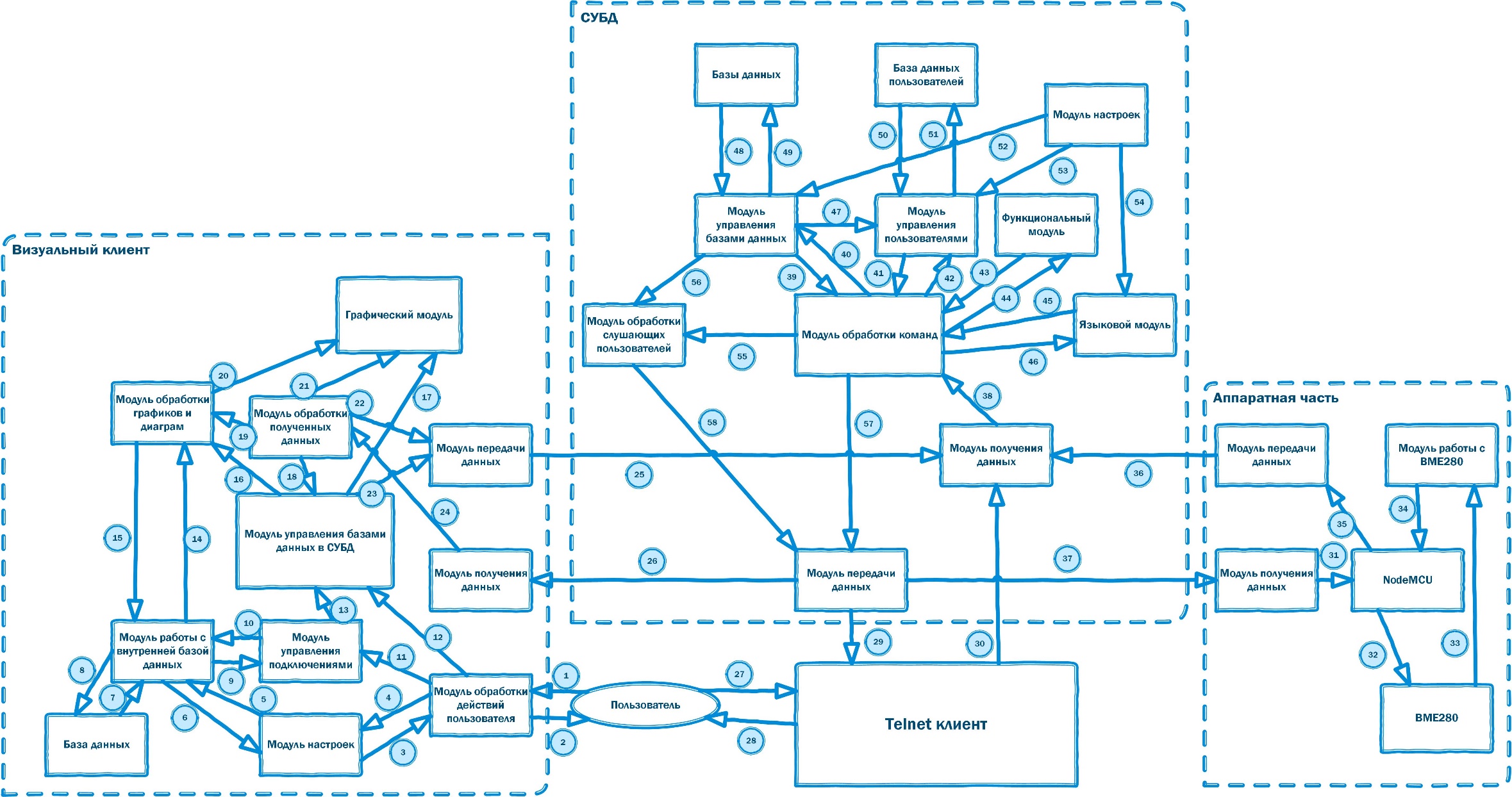


Рисунок Б. - Общий алгоритм работы программы

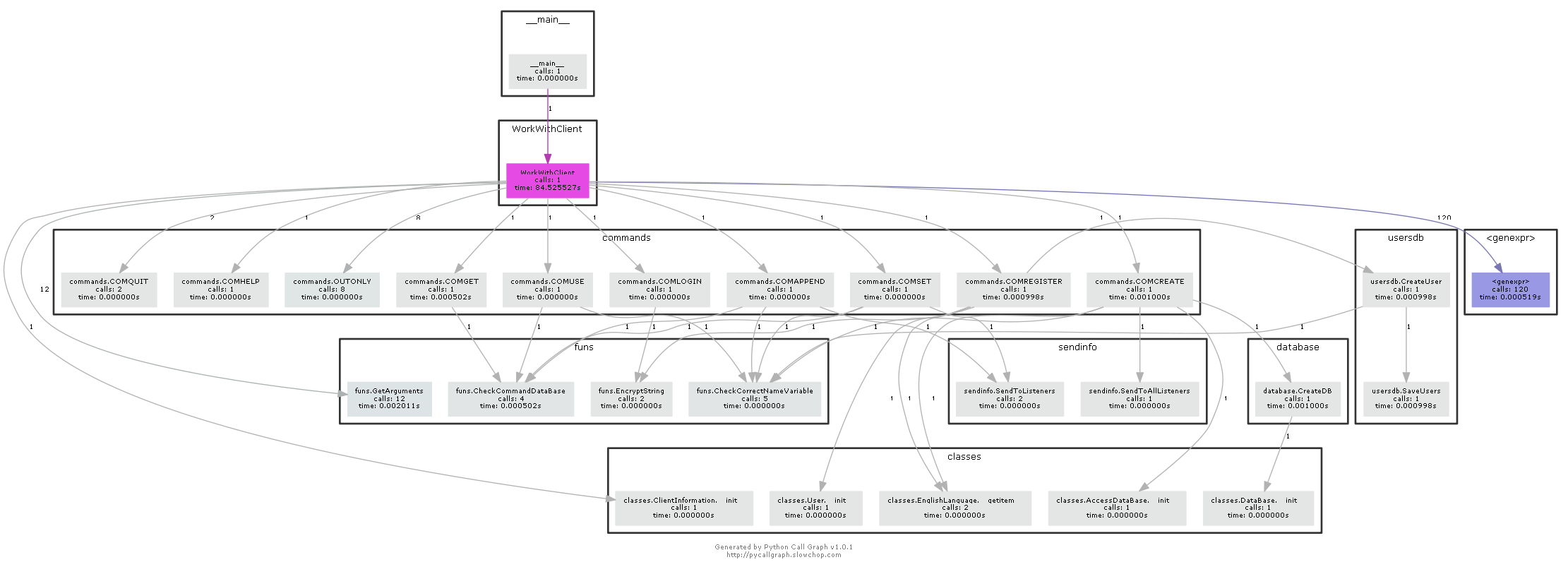


Рисунок Б. - Диаграмма вызово для пользователя, подключенного по Telnet

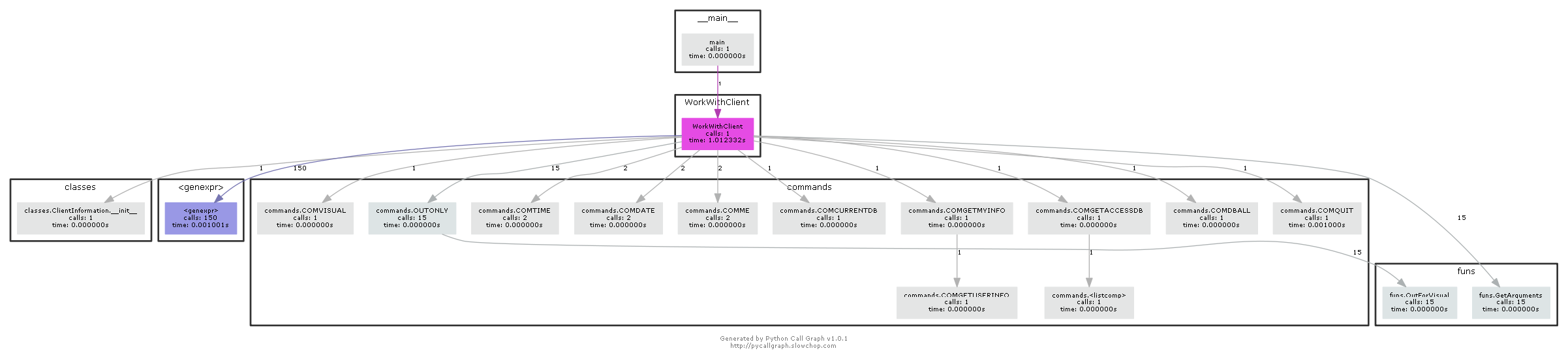


Рисунок Б. - Диаграмма вызовов пользователя визуального клиента