МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«МИРЭА — Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ОТЧЁТ ПО ПРАКТИЧЕСКИМ РАБОТАМ ДИСЦИПЛИНЫ «Инструментальное программное обеспечение разработки и проектирования информационных систем

Выполнил студент группы ИКМО-06-23 Горбунов Р.В.

Преподаватель Куликов А.А.

Москва, 2024

Содержание

Практическая работа №1.1 5
Цель работы 5
Задание 5
Выполнение работы 5
Заключение 6
Практическая работа №1.2 7
Цель работы 7
Задание 7
Выполнение работы 7
Заключение 8
Практическая работа №1.3 9
Цель работы 9
Задание 9
Выполнение работы 9
Заключение 11
Практическая работа №1.4 12
Цель работы 12
Задание 12
Выполнение работы 12
Заключение 13
Практическая работа №1.5 14
Цель работы 14
Задание 14
Выполнение работы 14
Заключение 15
Практическая работа №1.6 16
Цель работы 16
Задание 16
Выполнение работы 16
Заключение 17
Практическая работа №1.7 18

Цель работы 18
Задание 18
Выполнение работы 18
Заключение 20
Практическая работа №1.8 21
Цель работы 21
Задание 21
Выполнение работы 21
Заключение 22
Практическая работа №2.1 23
Цель работы 23
Задание 23
Выполнение работы 23
Заключение 26
Практическая работа №2.2 27
Цель работы 27
Задание 27
Выполнение работы 27
Заключение 31
Практическая работа №2.3 32
Цель работы 32
Задание 32
Выполнение работы 32
Заключение 37
Практическая работа №2.4 38
Цель работы 38
Задание 38
Выполнение работы 38
Заключение 41
Практическая работа №2.5 42
Цель работы 42
Задание 42

Выполнение работы..... 42
Заключение..... 45
Практическая работа №2.6 46
Цель работы 46
Задание 46
Выполнение работы..... 46
Заключение..... 51
Практическая работа №2.7 52
Цель работы 52
Задание 52
Выполнение работы..... 52
Заключение..... 58
Практическая работа №2.8 59
Цель работы 59
Задание 59
Выполнение работы..... 59
Заключение..... 60

Практическая работа №1.1

Цель работы

Научиться работать с классами и их методами

Задание

Создайте класс Soda (для определения типа газированной воды), принимающий 1 аргумент при инициализации (отвечающий за добавку или объем к выбираемому лимонаду). В этом классе реализуйте метод show_my_drink(), выводящий на печать «Газировка и {ДОБАВКА}» в случае наличия добавки, а иначе отобразится следующая фраза: «Обычная газировка».

Выполнение работы

Был создан класс Soda, принимающий 1 аргумент при инициализации – addition, отвечающий отвечающий за добавку или объем к выбираемому лимонаду. В этом классе был реализована функция show_my_drink(), выводящий на печать «Газировка и {ДОБАВКА}» в случае наличия добавки, а иначе отобразится следующая фраза: «Обычная газировка». Исходный код программы представлен на Листинге 1.1.1.

Листинг 1.1.1 – Исходный код программы

```
#include <iostream>
#include <string>

class Soda {
private:
    std::string additive;  // Добавка к газировке

public:
    // Конструктор класса, принимает аргумент additive при инициализации
    Soda(const std::string& additive) : additive(additive) {}

    // Метод для вывода информации о напитке
    void show_my_drink() {
        if (!additive.empty()) {
            std::cout << "Газировка и " << additive << std::endl;
        } else {
            std::cout << "Обычная газировка" << std::endl;
        }
    }
```

```
};

int main() {
    // Пример использования класса Soda
    Soda regularSoda("");  // Создаем объект газировки без добавки
    Soda flavoredSoda("лимон");  // Создаем объект газировки с добавкой

    // Выводим информацию о напитках
    std::cout << "Первый напиток: ";
    regularSoda.show_my_drink();

    std::cout << "Второй напиток: ";
    flavoredSoda.show_my_drink();

    return 0;
}
```

Заключение

В ходе практической работы были закреплены знания языка C++, а также была освоена технология работы с классами и их методами.

Практическая работа №1.2

Цель работы

Научиться работать с классами и их методами

Задание

Николаю требуется проверить, возможно ли из представленных отрезков условной длины сформировать треугольник. Для этого он решил создать класс TriangleChecker, принимающий только положительные числа. С помощью метода is_triangle() возвращаются следующие значения (в зависимости от ситуации):

- Ура, можно построить треугольник!
- С отрицательными числами ничего не выйдет!
- Нужно вводить только числа!
- Жаль, но из этого треугольник не сделать.

Выполнение работы

Был создан класс TriangleChecker, который принимает на вход только положительные числа. С помощью функции is_triangle() возвращается значения, описанные в задании. Исходный код программы представлен на Листинге 1.2.1

Листинг 1.2.1 – Исходный код программы

```
#include <iostream>

class TriangleChecker {
private:
    double side1, side2, side3;

public:
    // Конструктор класса, принимает три положительных числа
    TriangleChecker(double s1, double s2, double s3) {
        // Проверка на положительность входных значений
        if (s1 > 0 && s2 > 0 && s3 > 0) {
```

```
            side1 = s1;
            side2 = s2;
            side3 = s3;
        } else {
            std::cout << "С отрицательными числами ничего не выйдет!" << std::endl;
        }
    }

    // Метод для проверки возможности построения треугольника
    void is_triangle() {
        if (side1 + side2 > side3 && side1 + side3 > side2 && side2 + side3 >
side1) {
            std::cout << "Ура, можно построить треугольник!" << std::endl;
        } else {
            std::cout << "Жаль, но из этого треугольник не сделать." << std::endl;
        }
    }
};

int main() {
    // Пример использования класса TriangleChecker
    TriangleChecker triangle1(3, 4, 5);  // Возможность построения треугольника
    triangle1.is_triangle();

    TriangleChecker triangle2(-1, 2, 3);  // Отрицательное число
    TriangleChecker triangle3(1, 2, 4);   // Невозможность построения треугольника
    triangle3.is_triangle();

    return 0;
}
```

Заключение

В ходе практической работы были закреплены знания языка C++, а также была освоена технология работы с классами и их методами.

Практическая работа №1.3

Цель работы

Научиться работать с классами и их методами

Задание

Создать класс Raboch (Рабочие), у которого имеются 2 атрибута name и vozrast (имя и возраст) и методы init (), display_count(выводит количество рабочих) и display_raboch (выводит данные о рабочем) и класс Deti (Дети), у которого есть: два атрибута name и vozrast (имя и возраст); методы init () и school(), возвращающий номер школы ребенка.

Выполнение работы

Был создан класс Raboch (Рабочие), у которого имеются 2 атрибута name и vozrast (имя и возраст) и методы __init__(), display_count, который выводит количество рабочих и display_raboch, который выводит данные о рабочем, а также класс Deti (Дети), у которого есть: два атрибута name и vozrast (имя и возраст); метод school(), возвращающий номер школы ребенка. Исходный код программы представлен на Листинге 1.3.1.

Листинг 1.3.1 – Исходный код программы

```
#include <iostream>

class Raboch {
private:
    std::string name;
    int vozrast;

public:
    Raboch(const std::string& n, int v) : name(n),
vozrast(v) {
        count++; // Увеличиваем счетчик при создании
объекта
    }
```

```
    void display_raboch() {
        std::cout << "Имя: " << name << ", Возраст: " <<
vozrast << std::endl;
    }

    static int count;

    static void display_count() {
        std::cout << "Количество рабочих: " << count <<
std::endl;
    }
};

int Raboch::count = 0;

class Deti {
private:
    std::string name;
    int vozrast;
    int schoolNum;

public:
    Deti(const std::string& n, int v) : name(n), vozrast(v)
{
        if (vozrast <= 10) {
            schoolNum = 1;  // Младшая школа
        } else if (vozrast >= 11 && vozrast <= 14) {
            schoolNum = 2;  // Средняя школа
        } else {
            schoolNum = 3;  // Старшая школа
        }
    }

    void school() {
        std::cout << name << " учится в школе " <<
schoolNum << " с " << vozrast << " лет." << std::endl;
    }
};

int main() {
    Raboch raboch1("Иван", 35);
    Raboch raboch2("Мария", 28);
```

```
    std::cout << "Данные о рабочем 1:" << std::endl;
    raboch1.display_raboch();

    std::cout << "Данные о рабочем 2:" << std::endl;
    raboch2.display_raboch();

    Raboch::display_count();

    Deti deti1("Анна", 10);
    Deti deti2("Петя", 14);
    Deti deti3("Ольга", 15);

    deti1.school();
    deti2.school();
    deti3.school();

    return 0;
}
```

Заключение

В ходе практической работы были закреплены знания языка C++, а также была освоена технология работы с классами и их методами.

Практическая работа №1.4

Цель работы

Научиться работать с классами и их методами

Задание

Создан класс для перевода из кг в фунты KgToLb с параметром(аргументом) kg. С помощью метода to_lb() переводятся значения в фунты. Чтобы закрыть доступ к атрибуту kg созданы методы set_kg() - для нового значения кг, get_kg() - для вывода текущего значения кг. Но для этого нужно использовать 2 метода для задания и вывода значений. Задание: Нужно переделать класс с использованием функции property() и свойств-декораторов.

Выполнение работы

Был переделан класс для перевода из кг в фунты KgToLb с использованием свойств-декораторов. Исходный код программы представлен на Листинге 1.4.1.

Листинг 1.4.1 – Исходный код программы

```
#include <iostream>

class KgToLb {
private:
    double kg;

public:
    // Конструктор класса
    KgToLb(double initial_kg) : kg(initial_kg) {}

    // Геттер для получения значения в килограммах
    double get_kg() const {
        return kg;
    }

    // Сеттер для установки нового значения в килограммах
    void set_kg(double new_kg) {
        kg = new_kg;
    }

    // Метод для перевода из килограммов в фунты
    double to_lb() const {
        return kg * 2.20462;
    }
```

```
};


int main() {
    // Пример использования класса
    KgToLb weight(10.0);

    // Вывод текущего значения в килограммах
    std::cout << "Текущий вес в килограммах: " << weight.get_kg() << " кг" <<
std::endl;

    // Установка нового значения в килограммах
    weight.set_kg(15.0);

    // Вывод нового значения в килограммах и соответствующего значения в фунтах
    std::cout << "Новый вес в килограммах: " << weight.get_kg() << " кг" <<
std::endl;
    std::cout << "Эквивалент веса в фунтах: " << weight.to_lb() << " фунтов" <<
std::endl;

    return 0;
}
```

Заключение

В ходе практической работы были закреплены знания языка C++, а также была освоена технология работы с классами и их методами.

Практическая работа №1.5

Цель работы

Научиться работать с файлами.

Задание

Создать csv-файл «r_200.csv», который имеет столбцы:

- № - номер п/п (от 1 до 200);
- Секунда – секунда на вашем ПК;
- Микросекунда – миллисекунда на часах.

Приостанавливайте выполнение цикла на 0,02 секунды каждый проход.

Выполнение работы

Был создан csv-файл «r_200.csv», который имеет столбцы, описанные в задании. Для выполнения были использованы следующие библиотеки: fstream, chrono и thread. Исходный код программы представлен на Листинге 1.5.1.

Листинг 1.5.1 – Исходный код программы

```
#include <iostream>
#include <fstream>
#include <chrono>
#include <thread>

int main() {
    const int rowCount = 200;

    // Открываем файл для записи
    std::ofstream outFile("r_200.csv");

    // Проверяем, успешно ли открылся файл
    if (!outFile.is_open()) {
        std::cerr << "Ошибка открытия файла!" << std::endl;
        return 1;
    }

    // Записываем заголовок CSV
    outFile << "№,Секунда,Микросекунда" << std::endl;

    // Цикл для записи данных
    for (int i = 1; i <= rowCount; ++i) {
        // Получаем текущее время
        auto currentTime = std::chrono::system_clock::now();
```

```
        auto seconds =
std::chrono::time_point_cast<std::chrono::seconds>(currentTime);
        auto fraction = currentTime - seconds;

        // Преобразуем время в секунды и миллисекунды
        auto seconds_count =
std::chrono::duration_cast<std::chrono::seconds>(seconds.time_since_epoch()).count();
        auto milliseconds_count =
std::chrono::duration_cast<std::chrono::milliseconds>(fraction).count();

        // Записываем данные в файл
        outFile << i << "," << seconds_count << "," << milliseconds_count <<
std::endl;

        // Приостанавливаем выполнение цикла на 0,02 секунды
        std::this_thread::sleep_for(std::chrono::milliseconds(20));
    }

    // Закрываем файл
    outFile.close();

    std::cout << "CSV-файл успешно создан." << std::endl;

    return 0;
}
```

Заключение

В ходе практической работы были закреплены знания языка C++, а также была освоена технология работы с файлами.

Практическая работа №1.6

Цель работы

Научиться работать с файлами и папками с помощью дополнительных библиотек.

Задание

Выберите любую папку на компьютере, которая имеет вложенные папки. Задание: распечатать содержимое выбранной папки, и всех вложенных папок при помощи функции.

Выполнение работы

Была создана папка в репозитории со вложенными папками и файлами. Для реализации программы была использована библиотеки dirent.h и sys/stat.h. Исходный код программы представлен на Рисунке 1.6.1.

Листинг 1.6.1 – Исходный код программы

```
#include <iostream>
#include <dirent.h>
#include <sys/stat.h>

void print_directory_contents(const std::string& path) {
    DIR* dir = opendir(path.c_str());

    if (dir == nullptr) {
        std::cerr << "Не удалось открыть директорию: " << path << std::endl;
        return;
    }

    struct dirent* entry;
    while ((entry = readdir(dir)) != nullptr) {
        std::string full_path = path + "/" + entry->d_name;

        struct stat file_stat;
        if (stat(full_path.c_str(), &file_stat) == -1) {
            std::cerr << "Ошибка при получении информации о файле: " << full_path
<< std::endl;
            continue;
        }

        if (S_ISDIR(file_stat.st_mode)) {
            if (std::string(entry->d_name) != "." && std::string(entry->d_name) !=
"..") {
                std::cout << "Директория: " << entry->d_name << std::endl;
                print_directory_contents(full_path);
            }
        } else {
```

```
            std::cout << "Файл: " << entry->d_name << std::endl;
        }
    }

    closedir(dir);
}

int main() {
    // Путь к выбранной папке
    std::string folder_path = "/Users/lmistie/Desktop/Практики";

    // Выводим содержимое выбранной папки и всех вложенных папок
    print_directory_contents(folder_path);

    return 0;
}
```

Заключение

В ходе практической работы были закреплены знания языка C++, а также была освоена технология работы с файлами и папками.

Практическая работа №1.7

Цель работы

Научиться работать с файлами.

Задание

Документ «text.txt» содержит следующий текст:

Однажды
крепко спал Амур
любви божок
Отбросив факел свой
для всех сердец опасный
И к месту этому
вдруг подлетел кружок
Нимф, давших клятву
жить в невинности бесстрастной

Необходимо создать функцию, которая принимает на вход файл с указанным текстом (или любой другой текст). Результат работы функции будет вывод слов по максимальной длине

Выполнение работы

Был создан документ «text.txt», в котором содержался текст, представленный в задании. Были написаны функции, find_longest_words_in_file() которая принимает на вход файл, открывает его и разделяет текст на слова и process_file() которая обрабатывает вывод максимальной длины слов. Дальше эти слова сортируются по максимальной длине и выводятся. Исходный код программы представлен на Листинге 1.7.1.

Листинг 1.7.1 – Исходный код программы

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>

// Функция для поиска слов максимальной длины в строке
std::vector<std::string> find_longest_words(const std::string& line) {
    std::istringstream iss(line);
    std::string word;
    std::vector<std::string> longestWords;
    size_t maxLength = 0;

    while (iss >> word) {
        // Удаляем из слова знаки пунктуации
        word.erase(std::remove_if(word.begin(), word.end(), ispunct), word.end());

        // Проверяем длину слова
        if (word.length() > maxLength) {
            maxLength = word.length();
            longestWords.clear();
            longestWords.push_back(word);
        } else if (word.length() == maxLength) {
            longestWords.push_back(word);
        }
    }

    return longestWords;
}

// Функция для обработки файла и вывода слов максимальной длины
void process_file(const std::string& filename) {
    std::ifstream file(filename);

    if (!file.is_open()) {
        std::cerr << "Не удалось открыть файл: " << filename << std::endl;
        return;
    }

    std::string line;
    while (std::getline(file, line)) {
        std::vector<std::string> longestWords = find_longest_words(line);

        // Выводим результат
        if (!longestWords.empty()) {
            std::cout << "Слова максимальной длины (" << longestWords[0].length()
<< " символов): ";
            for (const auto& word : longestWords) {
                std::cout << word << " ";
            }
            std::cout << std::endl;
        }
    }

    file.close();
}

int main() {
```

```
    // Путь к файлу с текстом
    std::string filepath = "text.txt";

    // Обработка файла
    process_file(filepath);

    return 0;
}
```

Заключение

В ходе практической работы были закреплены знания языка C++, а также была освоена технология работы с файлами.

Практическая работа №1.8

Цель работы

Научиться работать с файлами.

Задание

Создать в Excel файл формата .csv, содержащий заголовок и несколько строк. В header прочесть заголовок, а в rows – остальные строки файла.

Выполнение работы

Был создан файл file.csv, в котором есть заголовок и несколько строк. Реализация в main с помощью std::ifstream ,которая получает на вход файл, далее происходит чтение этого файла и первая строчка считывается как заголовок (header), а все последующие как остальные строки файла (rows). Исходный код программы представлен на Листинге 1.8.1.

Листинг 1.8.1 – Исходный код программы

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>

int main() {
    // Открываем файл CSV для чтения
    std::ifstream file("Excel.csv");

    // Проверяем, успешно ли открылся файл
    if (!file.is_open()) {
        std::cerr << "Ошибка открытия файла!" << std::endl;
        return 1;
    }

    // Читаем заголовок
    std::string header;
    if (std::getline(file, header)) {
        std::cout << "Заголовок: " << header << std::endl;

        // Читаем остальные строки
        std::vector<std::string> rows;
        std::string line;
        while (std::getline(file, line)) {
            rows.push_back(line);
        }

        // Выводим остальные строки
        std::cout << "Остальные строки:" << std::endl;
```

```
        for (const auto& row : rows) {
            std::cout << row << std::endl;
        }
    } else {
        std::cerr << "Не удалось прочитать заголовок!" << std::endl;
    }

    // Закрываем файл
    file.close();

    return 0;
}
```

Заключение

В ходе практической работы были закреплены знания языка C++, а также была освоена технология работы с файлами.

Практическая работа №2.1

Цель работы

Создать приложение по отправки email писем из приложения разработанного на Python.

Задание

1. Создать простое приложение по отправки писем.
2. Создать форму в QT Designer.

Выполнение работы

В ходе выполнения данной практической работы была разработана форма для отправки email писем с помощью QT Designer. Результат представлен на Рисунке 2.1.1.

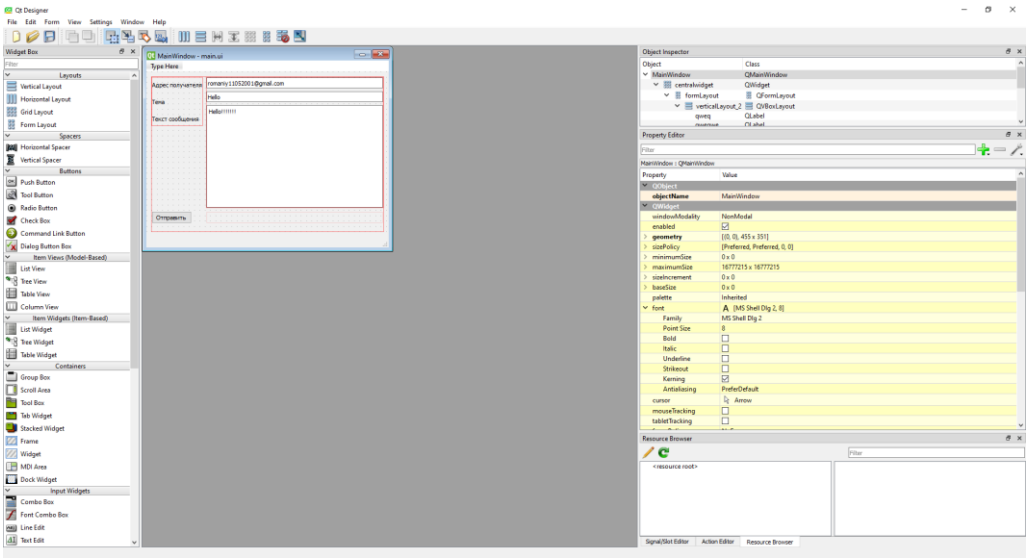


Рисунок 2.1.1 – Форма отправки email писем

Листинг 2.1.1 – Создания класса App

```
class App(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi('C:/Users/Roman/Desktop/Mirea/PR_Qt/PR1/main.ui', self)
        self.btn_send.clicked.connect(self.send_email)

    def send_email(self):
        send_email(addr_to=self.addr_to.text(), msg_subj=self.msg_subj.text(),
msg_text=self.msg_text.toPlainText())
```

В созданном классе в функции __init__ загружаем форму с помощью функции uic.loadUi. Также был сделан обработчик события “click”, который отслеживает, была ли нажата кнопка и при нажатии вызывает функцию send_email. Также была написана функция, которая, непосредственно, отправляет письма. В качестве параметров передаются адрес почты, кому нужно написать, тема письма и текст письма. Функция представлена на Листинге 2.1.2.

Листинг 2.1.2 – Функция отправки кода

```
def send_email(addr_to, msg_subj, msg_text):
    addr_from = "romaniy11052001@mail.ru "
    password = "……"

    msg = MIMEMultipart()
    msg['From'] = addr_from
    msg['To'] = addr_to
    msg['Subject'] = msg_subj

    body = msg_text
    msg.attach(MIMEText(body, 'plain'))

    server = smtplib.SMTP('smtp.mail.ru', 25)
    server.starttls()

    log = False
    if not log:
        try:
            server.login(addr_from, password)
            print('Подключение выполнено успешно.\n')
        except:
            print('Неверные данные отправителя.\n')
            return

    server.send_message(msg)
    print('Отправлено сообщение "{}" на тему "{}" на адрес {}.\n'.format(msg_text,
msg_subj, addr_to))
    server.quit()
```

В результате выполнения получился следующий интерфейс приложения, представленный на Рисунке 2.1.2.

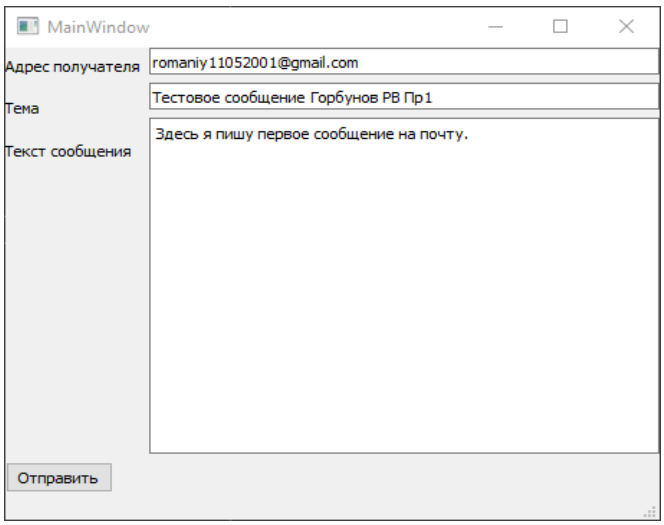


Рисунок 2.1.2 – Интерфейс приложений отправки email писем

При нажатии на кнопку «Отправить» было доставлено письмо. Само письмо представлено на Рисунке 2.1.3.

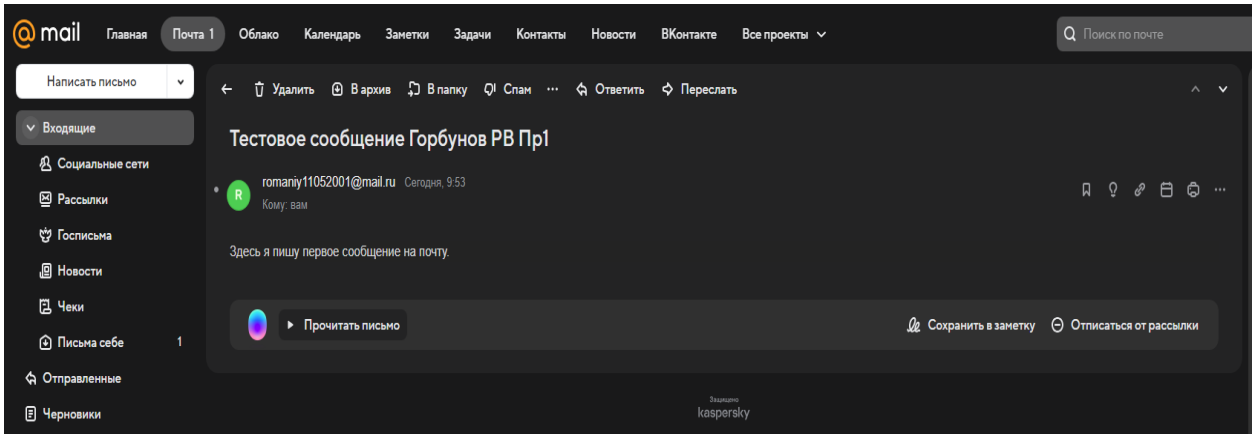


Рисунок 2.1.3 – Полученное письмо, отправленное через разработанное приложение

Листинг 2.1.3 – Исходный код программы

```
import sys
import smtplib
from PyQt5 import uic, QtWidgets
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

class App(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi(' C:/Users/Roman/Desktop/Mirea/PR_Qt/PR1/main.ui', self)
        self.btn_send.clicked.connect(self.send_email)

    def send_email(self):
        send_email(addr_to=self.addr_to.text(), msg_subj=self.msg_subj.text(),
msg_text=self.msg_text.toPlainText())

def send_email(addr_to, msg_subj, msg_text):
```

```
    addr_from = "romaniy11052001@mail.ru "
    password = "……"

    msg = MIMEMultipart()
    msg['From'] = addr_from
    msg['To'] = addr_to
    msg['Subject'] = msg_subj

    body = msg_text
    msg.attach(MIMEText(body, 'plain'))

    server = smtplib.SMTP('smtp.mail.ru', 25)
    server.starttls()

    log = False
    if not log:
        try:
            server.login(addr_from, password)
            print('Подключение выполнено успешно.\n')
        except:
            print('Неверные данные отправителя.\n')
            return

    server.send_message(msg)
    print('Отправлено сообщение "{}" на тему "{}" на адрес {}.\n'.format(msg_text,
msg_subj, addr_to))
    server.quit()

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    ex = App()
    ex.show()
    sys.exit(app.exec_())
```

Заключение

В ходе практической работы были закреплены знания языка Python, а также была освоена технология отправки писем с вложениями. Было написано полностью рабочее приложение по отправки писем.

Практическая работа №2.2

Цель работы

Создать приложение по отправки email писем из приложения разработанного на Python.

Задание

1. Доработать приложение по отправки email писем.
2. Необходимо сделать отправку с любого ящика.
3. Сделать добавление файлов в письмо.

Выполнение работы

В ходе выполнения данной практической работы была модернизирована форма для отправки email писем. Результат представлен на Рисунке 2.2.1.

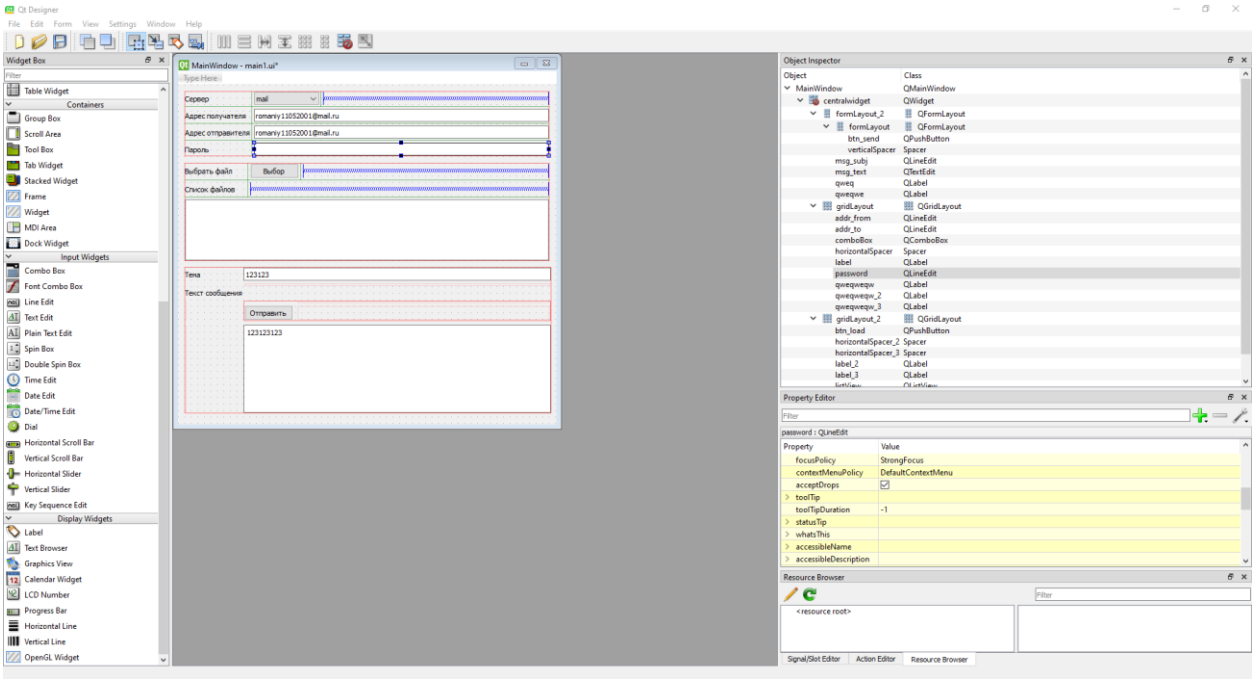


Рисунок 2.2.1 – Модернизированная форма отправки email писем

После модернизации формы отправки писем было проведено изменение в коде. Так, было добавлено несколько функций, а именно: open_file, код которой представлен на Листинге 2.2.1.

Листинг 2.2.1 – Функция open_file

```
    def open_file(self):
        filename = QtWidgets.QFileDialog.getOpenFileName(self, 'Open file', '',
'AllFiles (*)')
        if filename:
            self.path = filename
            item = QtGui.QStandardItem(self.path[0])
            self.model.appendRow(item)
            self.listView.setModel(self.model)
```

Данная функция открывает диалоговое окно для выбора файла, который будет добавлен в список элементов и прикреплен, в дальнейшем, к письму. Также были добавлены две функции из методического указания, которые прикрепляют файл к сообщению. По нажатию кнопки «Отправить» вставляются файлы в письмо. Код данных функций представлен на Листинге 2.2.2.

Листинг 2.2.2 – Функции прикрепления файлов к письму

```
def process_attachment(msg, files):
    for f in files:
        if os.path.isfile(f):
            attach_file(msg, f)
            print(f'File {f} attached')
        elif os.path.exists(f):
            directory = os.listdir(f)
            for file in directory:
                attach_file(msg, f + "/" + file)

def attach_file(msg, filepath):
    part = MIMEBase('application', "octet-stream")
    with open(filepath, 'rb') as file:
        part.set_payload(file.read())
    encoders.encode_base64(part)
    part.add_header('Content-Disposition', 'attachment',
filename=os.path.basename(filepath))
    msg.attach(part)
```

В результате выполнения получился следующий интерфейс приложения, представленный на Рисунке 2.2.2.

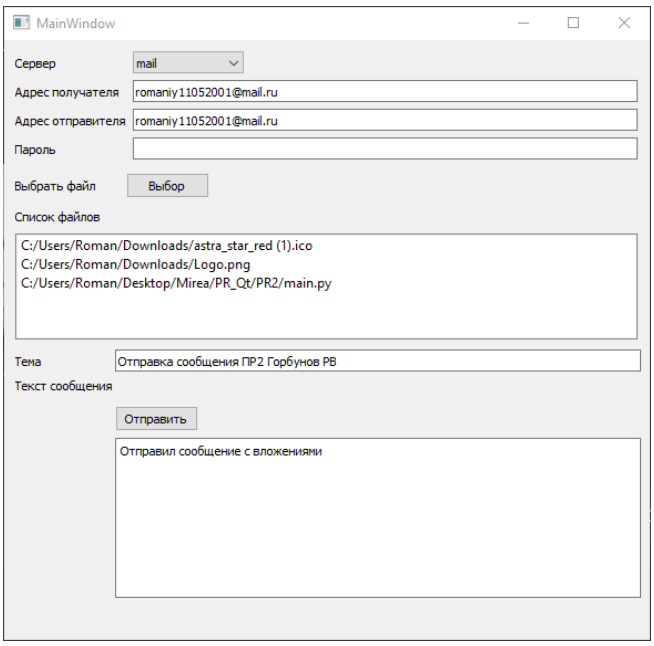


Рисунок 2.2.2 – Интерфейс приложения

При нажатии на кнопку «Отправить» было доставлено письмо. Само письмо представлено на Рисунке 2.2.3.

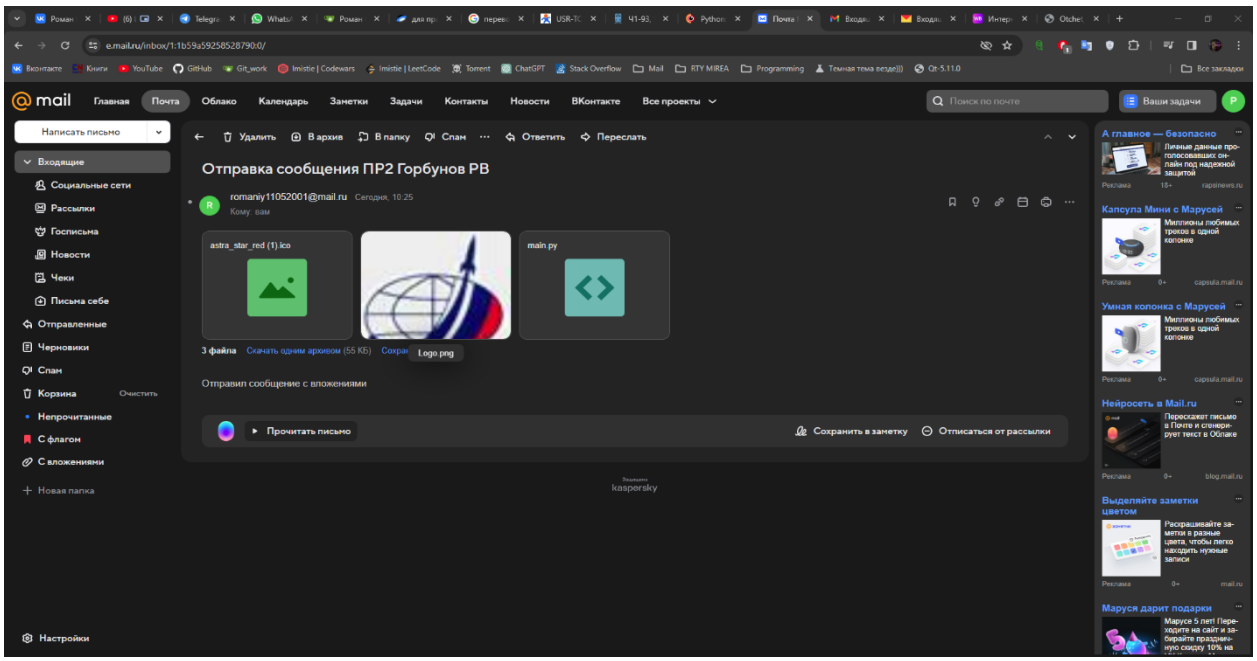


Рисунок 2.2.3 – Полученное письмо, отправленное через разработанное приложение

Весь исходный код представлен на Листинге 2.2.3.

Листинг 2.2.3 – Исходный код приложения

```
import os
import sys
import smtplib
import mimetypes
from email import encoders
from email.mime.multipart import MIMEMultipart
from email.mime.base import MIMEBase
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from email.mime.audio import MIMEAudio
from PyQt5 import uic, QtWidgets, QtGui

def process_attachment(msg, files):
    for f in files:
        if os.path.isfile(f):
            attach_file(msg, f)
            print(f'File {f} attached')
        elif os.path.exists(f):
            directory = os.listdir(f)
            for file in directory:
                attach_file(msg, f + "/" + file)

def attach_file(msg, filepath):
    part = MIMEBase('application', "octet-stream")
    with open(filepath, 'rb') as file:
        part.set_payload(file.read())
    encoders.encode_base64(part)
    part.add_header('Content-Disposition', 'attachment',
filename=os.path.basename(filepath))
    msg.attach(part)

def send_email(server_index, addr_from, password, msg_subj, msg_text, files):
    if server_index == 0:
        server = smtplib.SMTP('smtp.mail.ru', 25)
        server.starttls()
    elif server_index == 1:
        server = smtplib.SMTP_SSL('smtp.yandex.ru', 465)
    elif server_index == 2:
        server = smtplib.SMTP_SSL('smtp.gmail.com', 587)

    msg = MIMEMultipart()
    msg['From'] = addr_from
    msg['To'] = 'romaniy11052001@mail.ru'
    msg['Subject'] = msg_subj
    msg.attach(MIMEText(msg_text, 'plain'))

    log = False
    if not log:
        try:
            server.login(addr_from, password)
            print('Login successful.\n')
        except:
            print('Invalid sender credentials.\n')

    process_attachment(msg, files)
    server.sendmail(addr_from, msg['To'], msg.as_string())
    server.quit()
```

```
class App(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi(' C:/Users/Roman/Desktop/Mirea/PR_Qt/PR2/main1.ui', self)
        self.addr_from.setText('romaniy11052001@mail.ru')
        self.password.setText('GDWYgRSwALRjesx9pGmf')
        self.btn_send.clicked.connect(self.send_email)
        self.btn_load.clicked.connect(self.open_file)
        self.model = QtGui.QStandardItemModel(self.listView)

    def send_email(self):
        send_email(server_index=self.comboBox.currentIndex(),
                   addr_from=self.addr_from.text(),
                   password=self.password.text(),
                   msg_subj=self.msg_subj.text(),
                   msg_text=self.msg_text.toPlainText(),
                   files=[self.model.item(i).text() for i in
range(self.model.rowCount())])

    def open_file(self):
        filename = QtWidgets.QFileDialog.getOpenFileName(self, 'Open file', '',
'AllFiles (*)')
        if filename:
            self.path = filename
            item = QtGui.QStandardItem(self.path[0])
            self.model.appendRow(item)
            self.listView.setModel(self.model)

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    ex = App()
    ex.show()
    sys.exit(app.exec())
```

Заключение

В ходе практической работы были закреплены знания языка Python, а также была освоена технология отправки писем с вложениями. Было написано полностью рабочее приложение по отправки писем.

Практическая работа №2.3

Цель работы

Доработать приложение email писем из приложения разработанного на Python в практической работе №1.

Задание

1. Доработать приложение по отправки email писем.
2. Необходимо сделать массовую отправку адресатам со вложениями.

Выполнение работы

В ходе выполнения данной практической работы была модернизирована форма для отправки email писем. Результат модернизации представлен на Рисунке 2.3.1.

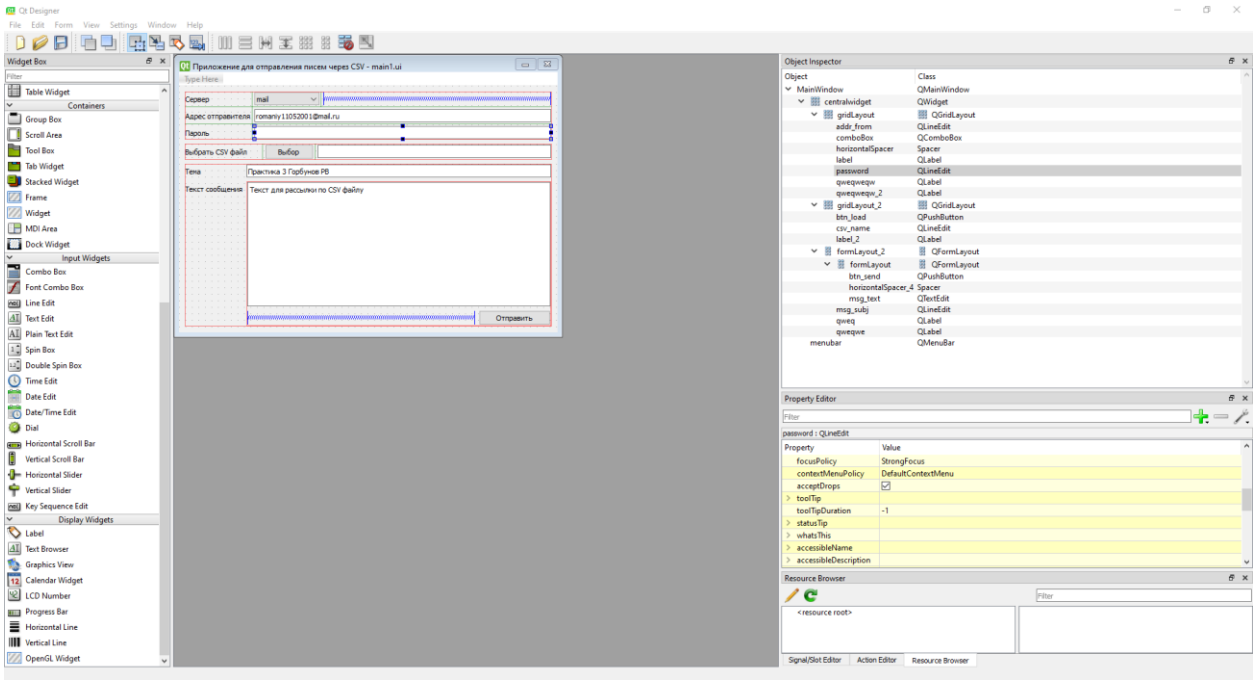


Рисунок 2.3.1 – Форма отправки email писем

После модернизации формы отправки писем было проведено изменение в коде. Для того, чтобы была возможность отправлять массово email письма с вложениями, сделаем файл CSV. В котором первым

значением будет «адресат», а последующие через «;» имена файлов (адрес файла). Содержания такого файла:

```
romaniy11052001@mail.ru;C:\Users\Roman\Desktop\Mirea\PR_Qt\PR3\astra_star_red_1.ico;C
:\Users\Roman\Desktop\Mirea\PR_Qt\PR3\astra_star_white.ico

romaniy11052001@gmail.com;C:\Users\Roman\Desktop\Mirea\PR_Qt\PR3\astra_star_red_1.ico
;C:\Users\Roman\Desktop\Mirea\PR_Qt\PR3\astra_star_white.ico

romaniy11052001@yandex.ru;C:\Users\Roman\Desktop\Mirea\PR_Qt\PR3\astra_star_red_1.ico
;C:\Users\Roman\Desktop\Mirea\PR_Qt\PR3\astra_star_white.ico
```

Обновленная функция, в которой происходит открытие файла рассылки, код которой представлен на Листинге 2.3.1.

Листинг 2.2.2 – Функции send_email

```
    def send_email(self):
        my_file = open("send.txt", "a")
        f = open(self.csv_name.text(), 'r')
        reader = csv.reader(f)
        for row in reader:
            row = row[0].split(';')
            addr_to = row[0]
            files = row[1:]
            send_email(server_index=self.comboBox.currentIndex(),
                       addr_from=self.addr_from.text(),
                       addr_to=addr_to,
                       password=self.password.text(),
                       msg_subj=self.msg_subj.text(),
                       msg_text=self.msg_text.toPlainText(),
                       files=files)
            my_file.write("\n Email {0}; Тема:{1}; Сообщение:{2}".format(addr_to,
self.msg_subj.text(), self.msg_text.toPlainText()))
```

В результате выполнения получился следующий модернизированный интерфейс приложения представлен на Рисунке 2.3.2.

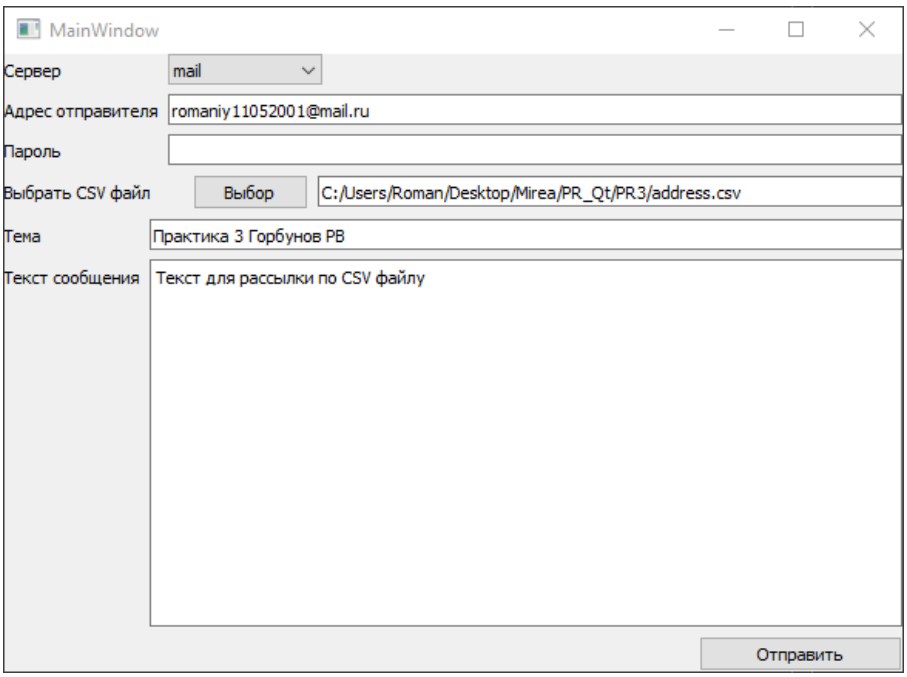
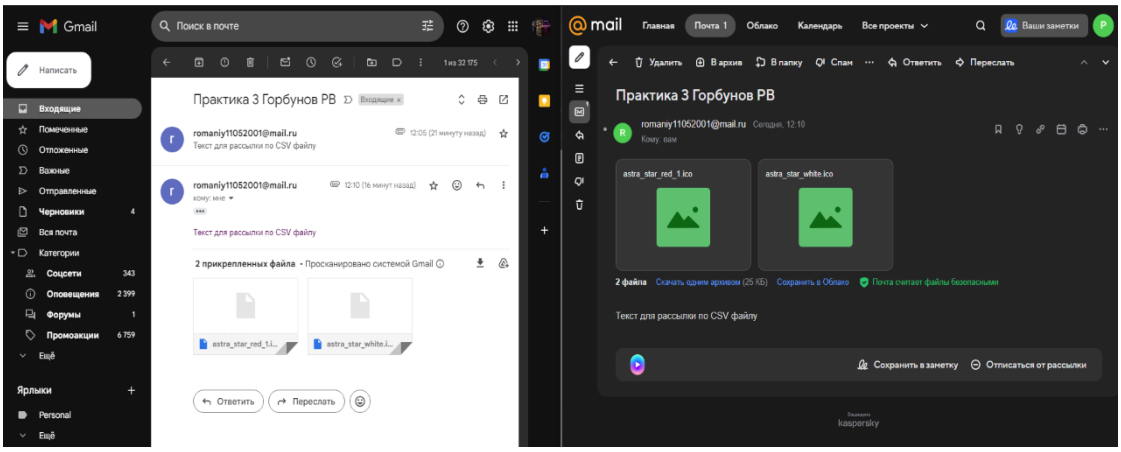


Рисунок 2.3.2 – Интерфейс приложения

При нажатии на кнопку «Отправить» были доставлены 3 письма. Сами письма представлены на Рисунке 2.3.3.



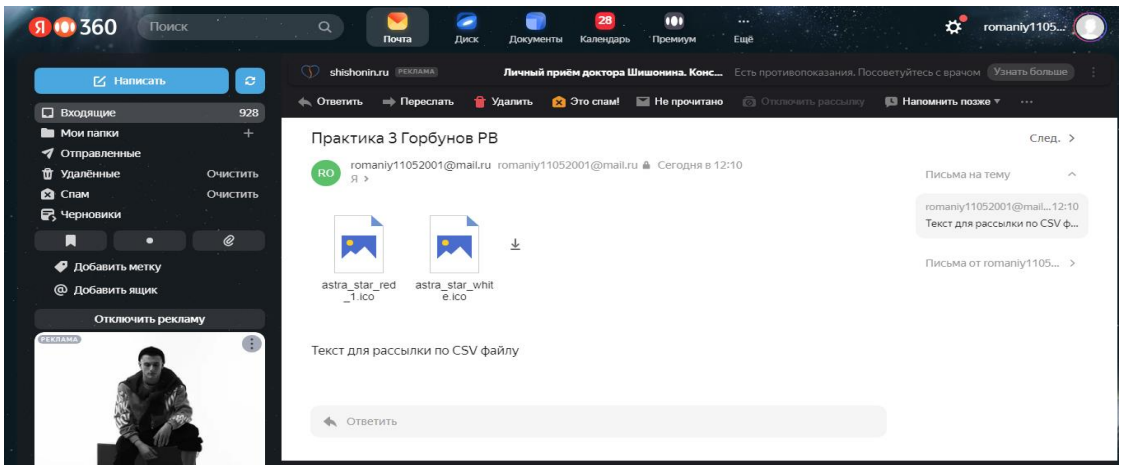


Рисунок 2.3.3 – Полученные письма, полученные через приложение QT

Исходный код представлен на Листинге 2.3.2.

Листинг 2.2.2 – Исходный код программы

```
import os
import sys
import smtplib
import csv
import mimetypes
from email import encoders
from email.mime.multipart import MIMEMultipart
from email.mime.base import MIMEBase
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from email.mime.audio import MIMEAudio
from PyQt5 import uic, QtWidgets

def process_attachment(msg, files):
    for f in files:
        if os.path.isfile(f):
            attach_file(msg, f)
            print(f'File {f} attached')
        elif os.path.exists(f):
            directory = os.listdir(f)
            for file in directory:
                attach_file(msg, f + "/" + file)

def attach_file(msg, filepath):
    filename = os.path.basename(filepath)
    ctype, encoding = mimetypes.guess_type(filepath)
    if ctype is None or encoding is not None:
        ctype = 'application/octet-stream'
    maintype, subtype = ctype.split('/', 1)
    if maintype == 'text':
        with open(filepath) as fp:
            file = MIMEText(fp.read(), _subtype=subtype)
        fp.close()
    elif maintype == 'image':
        with open(filepath, 'rb') as fp:
            file = MIMEImage(fp.read(), _subtype=subtype)
        fp.close()
    elif maintype == 'audio':
```

```
        with open(filepath, 'rb') as fp:
            file = MIMEAudio(fp.read(), _subtype=subtype)
        fp.close()
    else:
        with open(filepath, 'rb') as fp:
            file = MIMEBase(maintype, subtype)
            file.set_payload(fp.read())
        fp.close()
    encoders.encode_base64(file)
    file.add_header('Content-Disposition', 'attachment', filename=filename)
    msg.attach(file)

def send_email(server_index, addr_from, password, addr_to, msg_subj, msg_text, files):
    if server_index == 0:
        server = smtplib.SMTP_SSL('smtp.mail.ru', 465)
    elif server_index == 1:
        server = smtplib.SMTP_SSL('smtp.yandex.ru', 465)
    elif server_index == 2:
        server = smtplib.SMTP_SSL('smtp.gmail.com', 587)

    msg = MIMEMultipart()
    msg['From'] = addr_from
    msg['To'] = addr_to
    msg['Subject'] = msg_subj
    msg.attach(MIMEText(msg_text, 'plain'))

    log = False
    if not log:
        try:
            server.login(addr_from, password)
            print('Подключение выполнено успешно.\n')
        except:
            print('Неверные данные отправителя.\n')

    process_attachment(msg, files)
    server.sendmail(addr_from, [addr_to], msg.as_string())
    server.quit()

class App(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi('C:/Users/Roman/Desktop/Mirea/PR_Qt/PR3/main1.ui', self)
        self.addr_from.setText('romaniy11052001@mail.ru')
        self.password.setText('……')
        self.btn_send.clicked.connect(self.send_email)
        self.btn_load.clicked.connect(self.open_file)

    def send_email(self):
        my_file = open("send.txt", "a")
        f = open(self.csv_name.text(), 'r')
        reader = csv.reader(f)
        for row in reader:
            row = row[0].split(';')
            addr_to = row[0]
            files = row[1:]
            send_email(server_index=self.comboBox.currentIndex(),
                       addr_from=self.addr_from.text(),
                       addr_to=addr_to,
                       password=self.password.text(),
                       msg_subj=self.msg_subj.text(),
                       msg_text=self.msg_text.toPlainText(),
```

```
                        files=files)
            my_file.write("\n Email {0}; Тема:{1}; Сообщение:{2}".format(addr_to,

self.msg_subj.text(),

self.msg_text.toPlainText()))
    def open_file(self):
        filename = QtWidgets.QFileDialog.getOpenFileName(self, 'Открыть файл', '',
                                                         filter='csv (*.csv);;AllFiles
(*)')
        if filename:
            self.csv_name.setText(filename[0])
        else:
            return

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    ex = App()
    ex.show()
    sys.exit(app.exec())
```

Заключение

В ходе практической работы были закреплены знания языка Python, а также была освоена технология отправки писем с вложениями. Чтение CSV файла. Было написано полностью рабочее приложение по отправки писем.

Практическая работа №2.4

Цель работы

Сделать веб сервис для отправки email сообщений.

Задание

1. Сделать веб-сервис, который будет принимать на вход необходимые параметры для отправки email сообщений.

Выполнение работы

Для выполнения практической работы необходимо разработать веб-сервис с использованием библиотеки Flask. Был создан новый проект и назван “WebService”. В качестве интерфейса был разработан HTML код и добавлены стили. Исходный код интерфейса представлен на Листинге 2.4.1.

Листинг 2.4.1 – Исходный код программы

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Веб-сервис отправки писем</title>
    <style>
        .container {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
        }

        form {
            width: 50%;
            border: 1px solid #ccc;
            border-radius: 4px;
            padding: 20px;
            box-sizing: border-box;
            background-color: #f2f2f2;
        }

        .input-group {
            margin-bottom: 15px;
        }

        .input-group label {
            display: block;
            font-weight: bold;
            margin-bottom: 5px;
        }
```

```
        .input-group input[type=text] {
            width: calc(100% - 40px);
            padding: 10px;
            border: 1px solid #ccc;
            border-radius: 4px;
            box-sizing: border-box;
        }

        .input-group input[type=submit] {
            width: 100%;
            background-color: #04AA6D;
            color: white;
            padding: 14px 20px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
        }

        .input-group input[type=submit]:hover {
            background-color: #45a049;
        }
    </style>
</head>

<body>
    <div class="container">
        <form method="post" name="send_email">
            <div class="input-group">
                <label for="addr_from">Адрес отправителя:</label>
                <input type="text" name="addr_from" id="addr_from">
            </div>
            <div class="input-group">
                <label for="password">Пароль:</label>
                <input type="text" name="password" id="password">
            </div>
            <div class="input-group">
                <label for="addr_to">Адрес получателя:</label>
                <input type="text" name="addr_to" id="addr_to">
            </div>
            <div class="input-group">
                <label for="msg_subj">Заголовок письма:</label>
                <input type="text" name="msg_subj" id="msg_subj">
            </div>
            <div class="input-group">
                <label for="msg_text">Текст письма:</label>
                <input type="text" name="msg_text" id="msg_text">
            </div>
            <div class="input-group">
                <input type="submit" value="Отправить">
            </div>
        </form>
    </div>
</body>

</html>
```

Внешний вид интерфейса представлен на Рисунке 2.4.1.

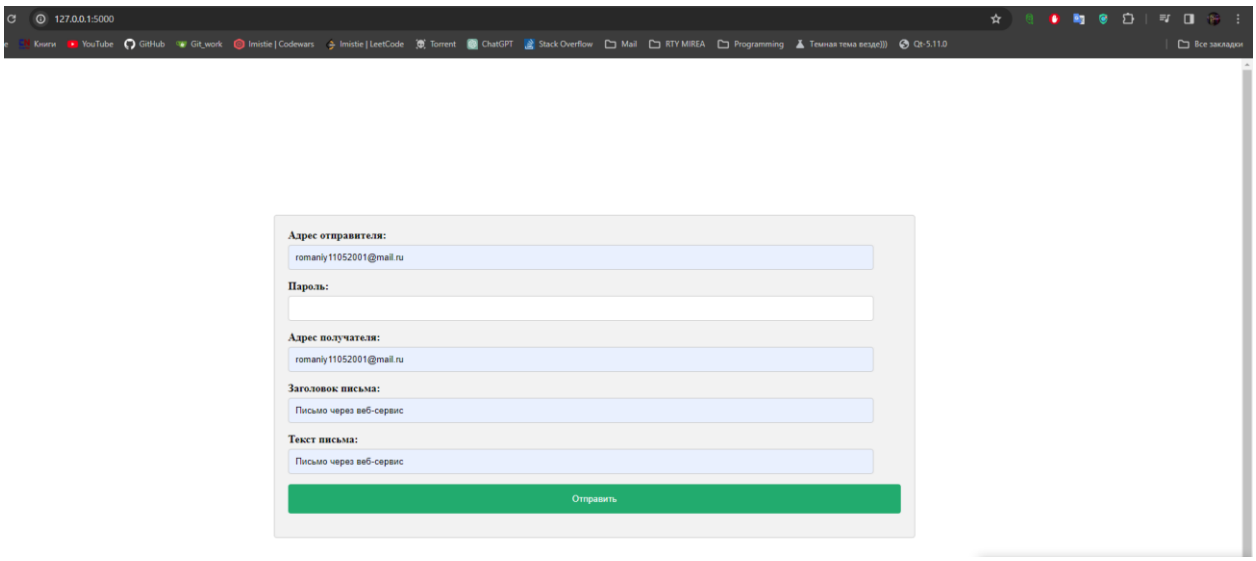


Рисунок 2.4.1. – Внешний вид веб-сервиса

С помощью библиотеки Flask и результатов практической работы №1 была реализована отправка писем с помощью веб-сервиса. При нажатии на кнопку происходит отправка письма. Результат представлен на Рисунке 2.4.2.

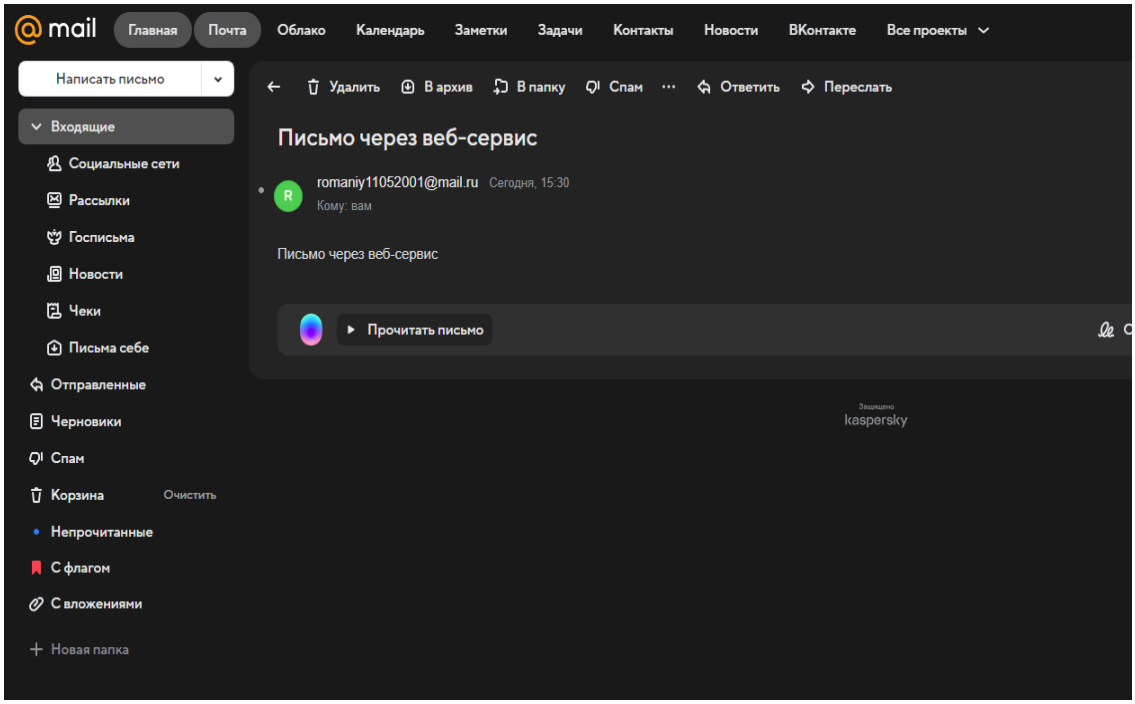


Рисунок 2.4.2 – Полученное письмо, отправленное через веб-сервис

Весь исходный код представлен на Листинге 2.4.2.

Листинг 2.4.2 – Исходный код программы

```
from flask import Flask, request, redirect, render_template
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

app = Flask(__name__, template_folder="")

@app.route("/", methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        server = smtplib.SMTP_SSL('smtp.mail.ru', 465)
        addr_from = request.form['addr_from']
        password = request.form['password']
        addr_to = request.form['addr_to']
        msg_subj = request.form['msg_subj']
        msg_text = request.form['msg_text']

        msg = MIMEMultipart()
        msg['From'] = addr_from
        msg['To'] = addr_to
        msg['Subject'] = msg_subj
        msg.attach(MIMEText(msg_text, 'plain'))

        server.login(addr_from, password)
        server.sendmail(addr_from, [addr_to], msg.as_string())
        server.quit()

        return render_template('index.html')

    if request.method == 'GET':
        return render_template('index.html')

if __name__ == '__main__':
    app.run()
```

Заключение

В ходе практической работы были закреплены знания языка Python, а также была освоена технология отправки писем с веб-сервисом.

Практическая работа №2.5

Цель работы

Создать калькулятор на Python.

Задание

1. Создать умный калькуоятор
2. Создать форму в Qt Designer

Выполнение работы

В ходе выполнения данной практической работы была разработана форма калькулятора с помощью QT Designer. Результат представлен на Рисунке 2.5.1.

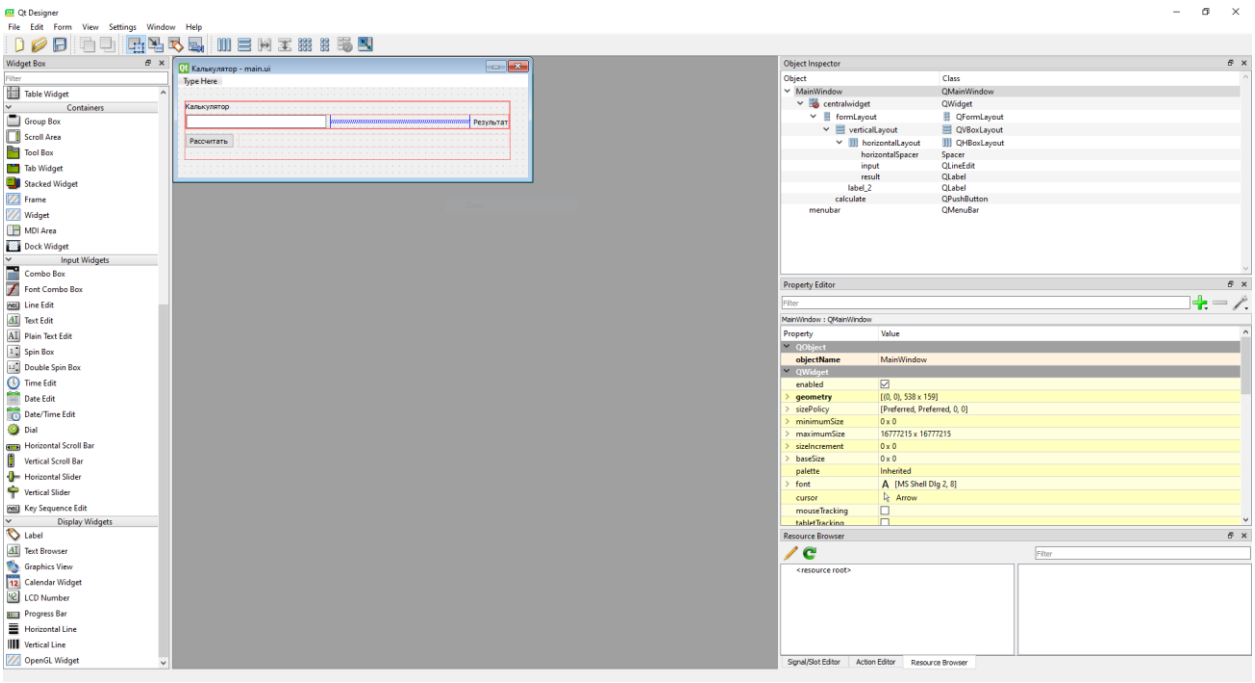


Рисунок 2.5.1 – Форма калькулятора

Затем данный макет был сохранен и загружен в проект. Был создан класс от QtWidgets.QMainWindow, для того чтобы использовать графический 45 интерфейс PyQT. Также была создана функция, которая принимает на вход строку, которую необходимо рассчитать, производит расчет и выводит

результат. Листинг создания класса представлен на Листинге 2.5.1. Весь исходный код представлен на Листинге 2.5.2

Листинг 2.5.1 – Создание класса App

```
class App(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi('C:/Users/Roman/Desktop/Mirea/PR_Qt/PR5/main.ui', self)
        self.calculate.clicked.connect(self.calc)

    def calc(self):
        expression = self.input.text()
        result = my_eval(expression)
        self.result.setText(str(result))
```

В результате выполнения получается следующий интерфейс приложения представленный на Рисунке 2.5.2

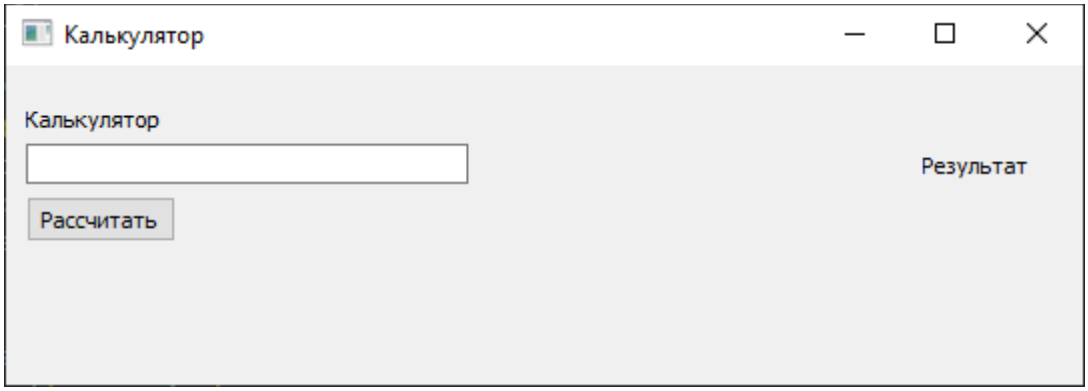


Рисунок 2.5.2 – Интерфейс приложения калькулятора

При нажатии на кнопку «Рассчитать» происходит расчет. Результат работы представлен на Рисунке 2.5.3.

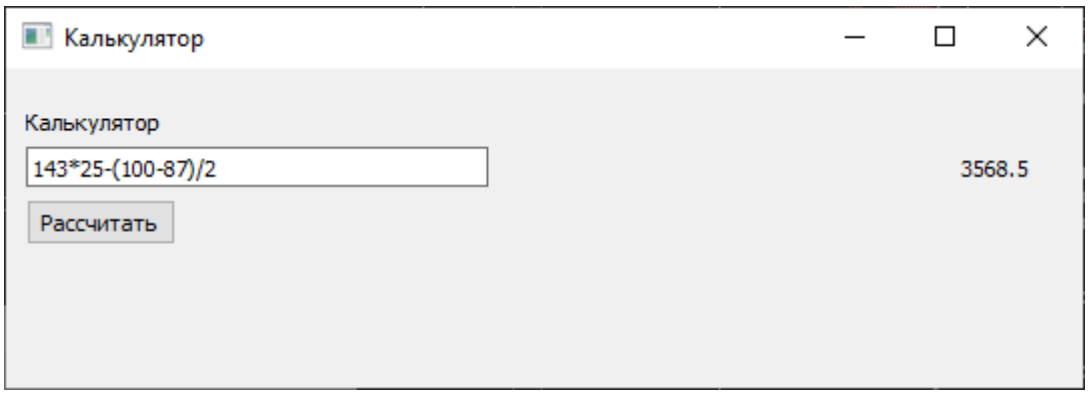


Рисунок 2.5.3 – Полученный результат

Листинг 2.5.2 – Исходный код программы

```
import sys
from PyQt5 import uic, QtWidgets

def my_eval(expression):
    operators = {'+': 1, '-': 1, '*': 2, '/': 2}
    ops_stack = []
    nums_stack = []

    def apply_op():
        op = ops_stack.pop()
        num2 = nums_stack.pop()
        num1 = nums_stack.pop()
        if op == '+':
            nums_stack.append(num1 + num2)
        elif op == '-':
            nums_stack.append(num1 - num2)
        elif op == '*':
            nums_stack.append(num1 * num2)
        elif op == '/':
            nums_stack.append(num1 / num2)

    i = 0
    while i < len(expression):
        if expression[i].isdigit():
            num = int(expression[i])
            i += 1
            while i < len(expression) and expression[i].isdigit():
                num = num * 10 + int(expression[i])
                i += 1
            nums_stack.append(num)
        elif expression[i] in operators:
            while ops_stack and ops_stack[-1] != '(' and operators[ops_stack[-1]]
>= operators[expression[i]]:
                apply_op()
            ops_stack.append(expression[i])
            i += 1
        elif expression[i] == '(':
            ops_stack.append(expression[i])
            i += 1
        elif expression[i] == ')':
            while ops_stack[-1] != '(':
                apply_op()
            ops_stack.pop()
            i += 1
        else:
            i += 1

    while ops_stack:
        apply_op()
    return nums_stack[0]

class App(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi('C:/Users/Roman/Desktop/Mirea/PR_Qt/PR5/main.ui', self)
        self.calculate.clicked.connect(self.calc)

    def calc(self):
```

```
        expression = self.input.text()
        result = my_eval(expression)
        self.result.setText(str(result))

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    ex = App()
    ex.show()
    sys.exit(app.exec())
```

Заключение

В ходе практической работы были закреплены знания языка Python, а также была освоена технология написания умного калькулятора.

Практическая работа №2.6

Цель работы

Создать текстовый редактор на Python.

Задание

1. Создать полноценный редактор на Python
2. Создать форму в Qt Designer

Выполнение работы

В ходе выполнения данной практической работы была разработана форма калькулятора с помощью QT Designer. Результат представлен на Рисунке 2.6.1.

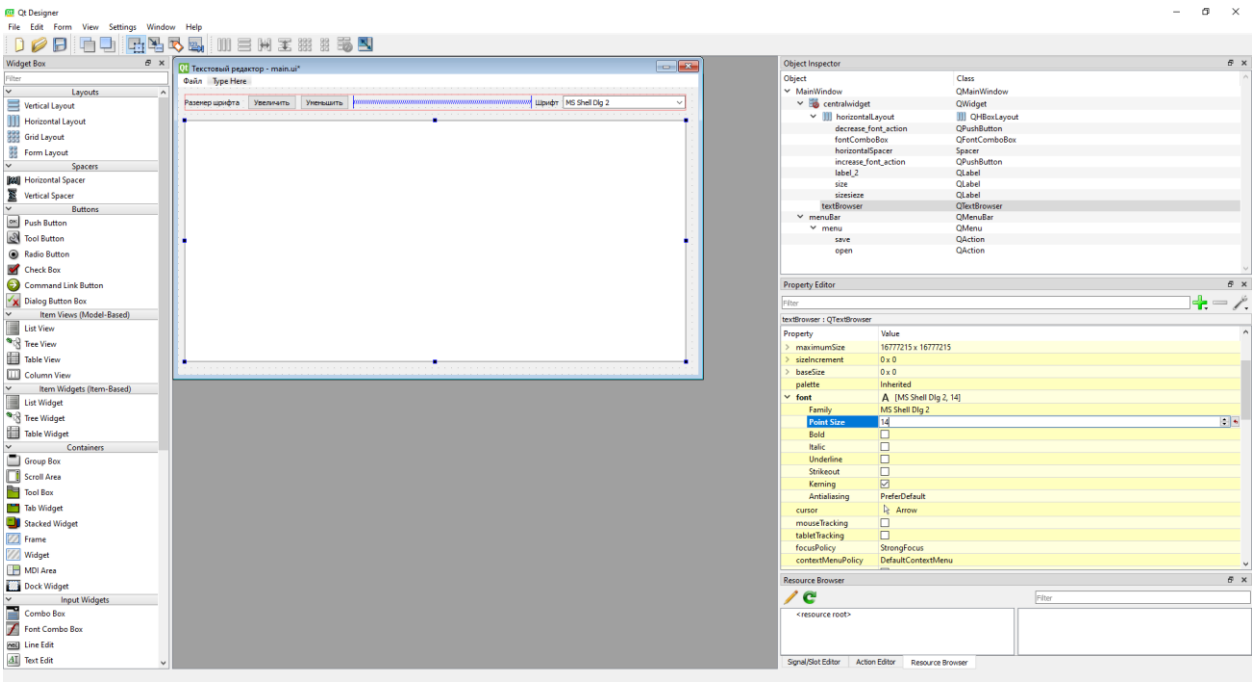


Рисунок 2.6.1 – Форма редактора

Затем данный макет был сохранен и загружен в проект. Был создан класс от QtWidgets.QMainWindow, для того чтобы использовать графический интерфейс PyQT. Листинг создания класса представлен на Листинге 2.6.1. Весь исходный код представлен на Листинге 2.6.2.

Листинг 2.6.1 – Создание класса App

```
class App(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi('C:/Users/Roman/Desktop/Mirea/PR_Qt/PR6/main.ui', self)
        self.font_size = QtWidgets.QLabel()
        self.save.triggered.connect(self.save_file)
        self.open.triggered.connect(self.open_file)
        self.increase_font_action.clicked.connect(self.increase_font_size)
        self.decrease_font_action.clicked.connect(self.decrease_font_size)
        self.fontComboBox.currentFontChanged.connect(self.change_font)
```

В результате выполнения получается следующий интерфейс приложения представленный на Рисунке 2.6.2

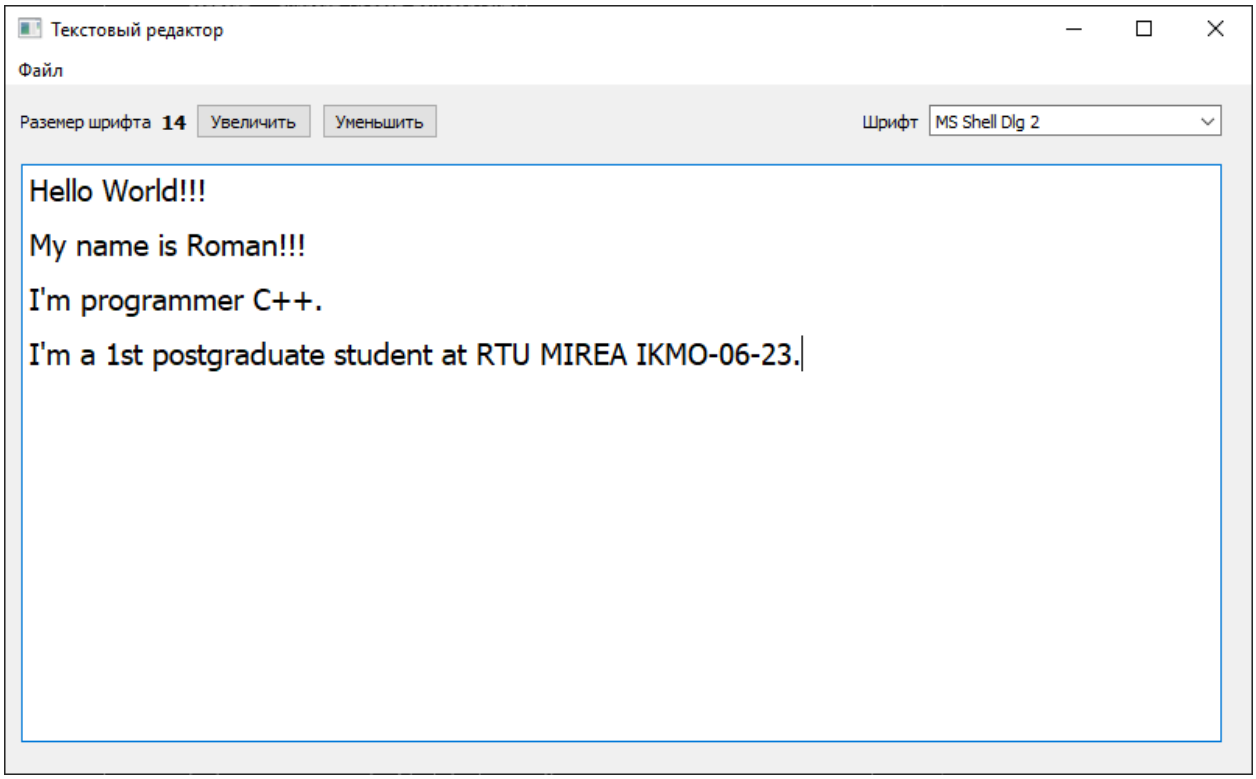


Рисунок 2.6.2 – Интерфейс приложения

При нажатии на кнопку «Файл» раскрывается меню, в котором можно либо сохранить, либо открыть файл, что отображено на Рисунке 2.6.3.

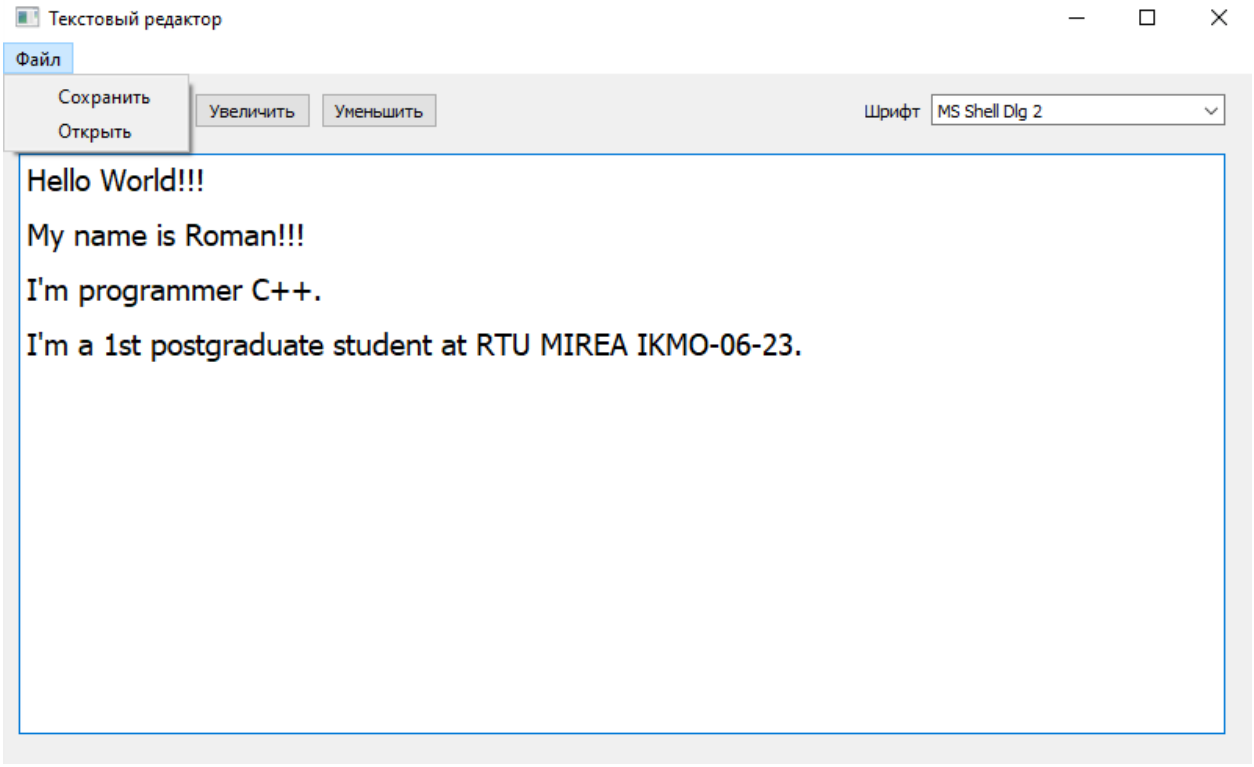


Рисунок 2.6.3 – Полученный результат

При нажатии на кнопку «Открыть файл» открывается диалоговое окно для выбора файла. После выбора файла весь текст открывается в соответствующем окне. Благодаря функционалу приложения можно редактировать файл: сменить шрифт и поменять его размер. Результат работы текстового редактора представлен на Рисунке 2.6.4.

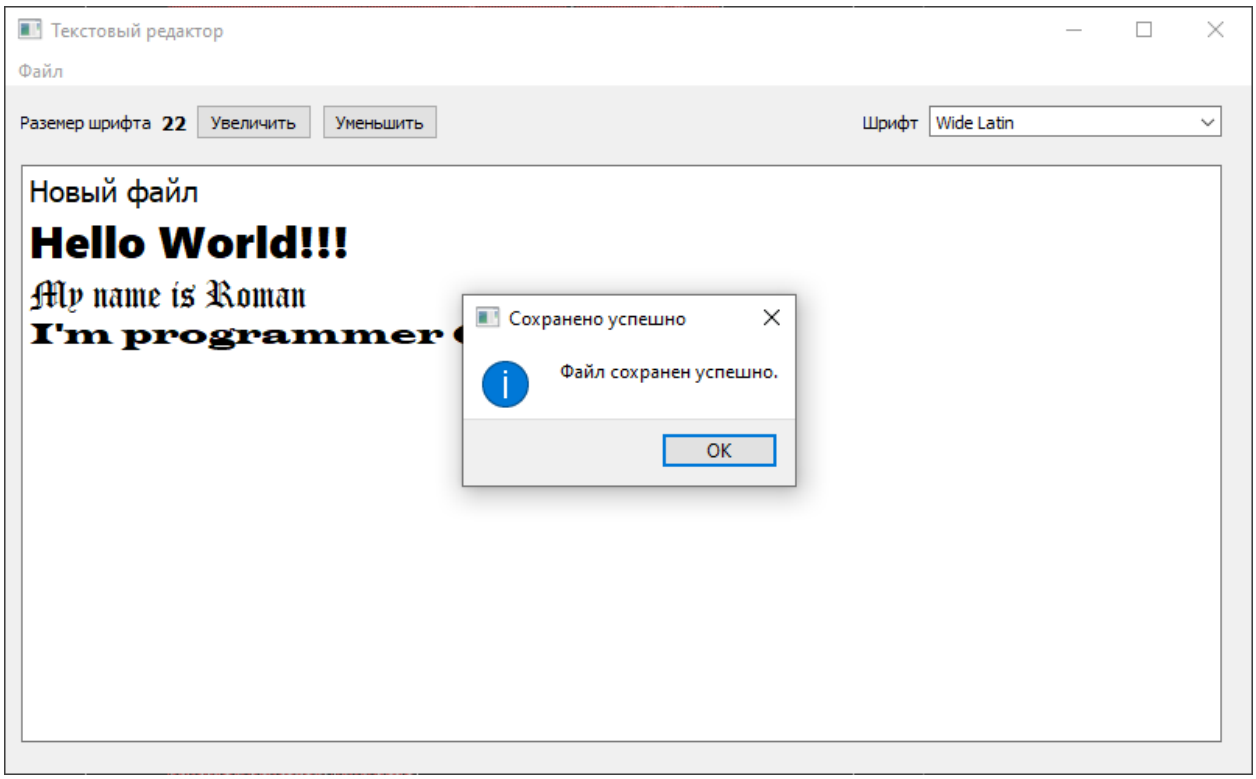


Рисунок 2.6.4 – Результат работы текстового редактора.

Измененный файл представлен на Рисунке 2.6.5. К сожалению, форматирование сохранить не получается, так как функционал текстового редактора сделан для демонстрации работы простого редактора, для полноценного приложения, нужно существенно доработать код.

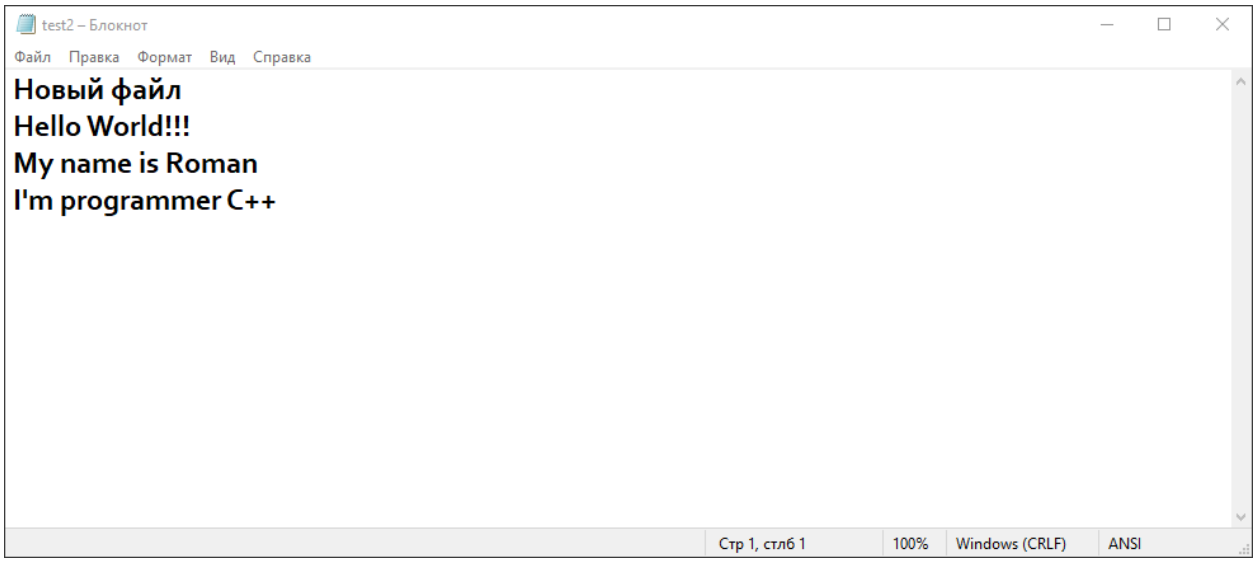


Рисунок 2.6.5 – Сохранённый файл

Листинг 2.6.2 – Исходный код программы

```
import sys
from PyQt5 import uic, QtWidgets

class App(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi('C:/Users/Roman/Desktop/Mirea/PR_Qt/PR6/main.ui', self)
        self.font_size = QtWidgets.QLabel()
        self.save.triggered.connect(self.save_file)
        self.open.triggered.connect(self.open_file)
        self.increase_font_action.clicked.connect(self.increase_font_size)
        self.decrease_font_action.clicked.connect(self.decrease_font_size)
        self.fontComboBox.currentFontChanged.connect(self.change_font)

    def open_file(self):
        file, _ = QtWidgets.QFileDialog.getOpenFileName(self, 'Открыть файл')
        if file:
            encodings = ["utf-8", "latin-1", "cp1252"]
            content = None
            for encoding in encodings:
                try:
                    with open(file, "r", encoding=encoding) as f:
                        content = f.read()
                    break
                except UnicodeDecodeError:
                    continue
            if content is not None:
                self.textBrowser.setPlainText(content)
            else:
                QtWidgets.QMessageBox.warning(self, 'Открыть файл', 'Не получается
открыть файл.')

    def save_file(self):
        current_widget = self.textBrowser
        content = current_widget.toPlainText()
        file_dialog = QtWidgets.QFileDialog()
        file_name, _ = file_dialog.getSaveFileName(self, "Сохранить файл", "", "All
Files (*)")
        if file_name:
            with open(file_name, "w") as f:
                f.write(content)
            QtWidgets.QMessageBox.information(self, "Сохранено успешно", "Файл
сохранен успешно.")

    def increase_font_size(self):
        current_widget = self.textBrowser
        cursor = current_widget.textCursor()
        format = cursor.charFormat()
        font = format.font()
        font_size = font.pointSize()
        font_size += 1
        self.font_size = self.findChild(QtWidgets.QLabel, 'size')
        font.setPointSize(font_size)
        format.setFont(font)
        self.font_size.setText(str(font_size))
        cursor.mergeCharFormat(format)
```

```
        current_widget.setFocus()

    def decrease_font_size(self):
        current_widget = self.textBrowser
        cursor = self.textBrowser.textCursor()
        format = cursor.charFormat()
        font = format.font()
        font_size = font.pointSize()
        font_size -= 1
        self.font_size = self.findChild(QtWidgets.QLabel, 'size')
        font.setPointSize(font_size)
        format.setFont(font)
        self.font_size.setText(str(font_size))
        cursor.mergeCharFormat(format)
        current_widget.setFocus()

    def change_font(self, font):
        current_widget = self.textBrowser
        text_cursor = current_widget.textCursor()
        format = text_cursor.charFormat()
        font_name = font.family()
        format.setFontFamily(font_name)
        text_cursor.mergeCharFormat(format)
        current_widget.setFocus()

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    ex = App()
    ex.show()
    sys.exit(app.exec())
```

Заключение

В ходе практической работы были закреплены знания языка Python.

Практическая работа №2.7

Цель работы

Подключение к БД, выполнение SQL запросов.

Задание

1. Создать приложение к БД, выполнение SQL запросов.
2. Выполнение SQL запросов.

Выполнение работы

В ходе выполнения данной практической работы была разработана форма для взаимодействия с базой данных MySQL с помощью QT Designer. Результат представлен на Рисунке 2.7.1.

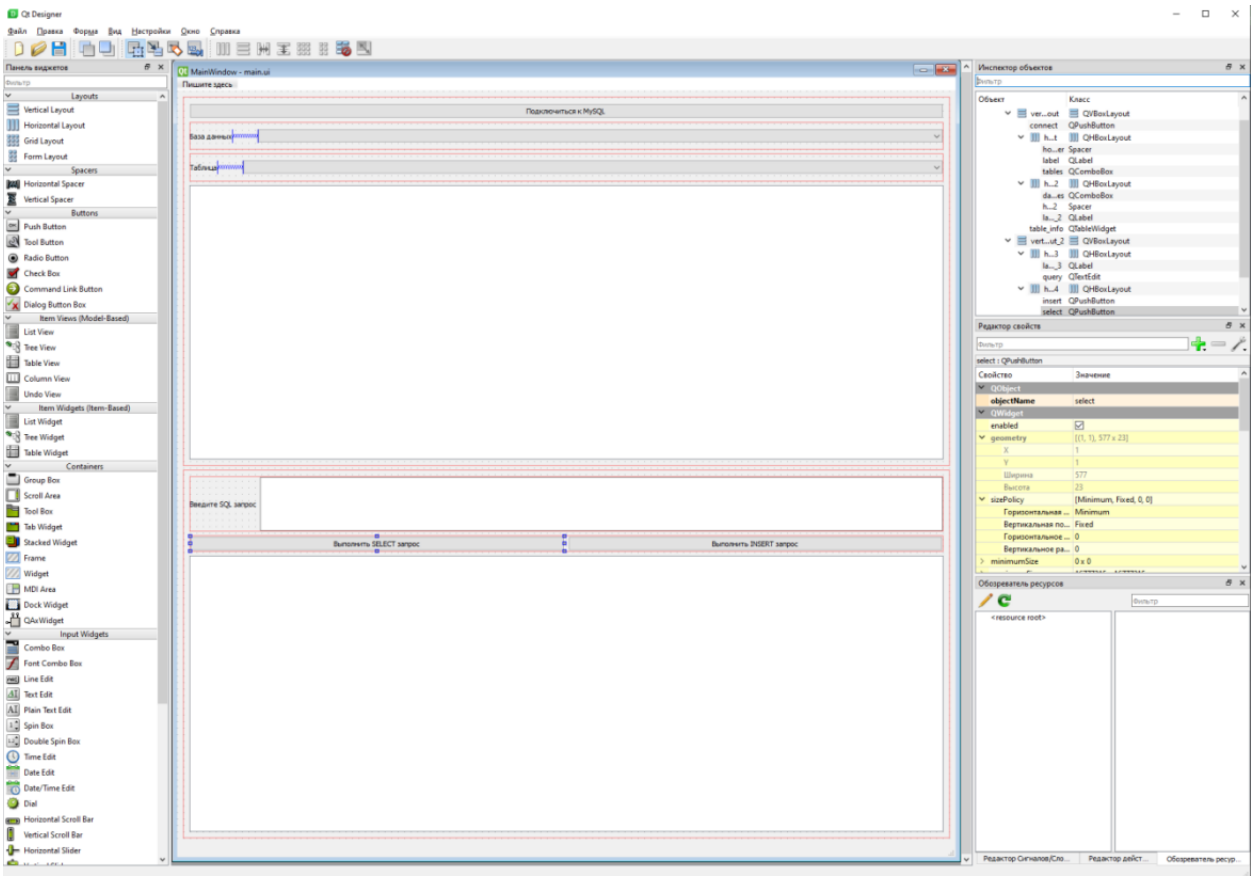


Рисунок 2.7.1 – Форма приложения

Затем данный макет был сохранен и загружен в проект. Был создан класс от QtWidgets.QMainWindow, для того чтобы использовать графический интерфейс PyQT. Листинг создания класса представлен на Листинге 2.7.1.

Листинг 2.7.1 – Создание класса App

```
class App(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi('main.ui', self)
        self.connect.clicked.connect(self.connect_to_db)
        self.databases.currentIndexChanged.connect(self.show_tables)
        self.tables.currentIndexChanged.connect(self.show_info)
        self.select.clicked.connect(self.select_query)
        self.insert.clicked.connect(self.add_value)
```

В созданном классе в функции __init__ загружаем форму с помощью функции uic.loadUi. Также было написано несколько функций, которые, непосредственно, осуществляют взаимодействие с базой данных. Весь исходный код представлен на Листинге 2.7.2. В результате выполнения получился следующий интерфейс приложения, представленный на Рисунке 2.7.2.

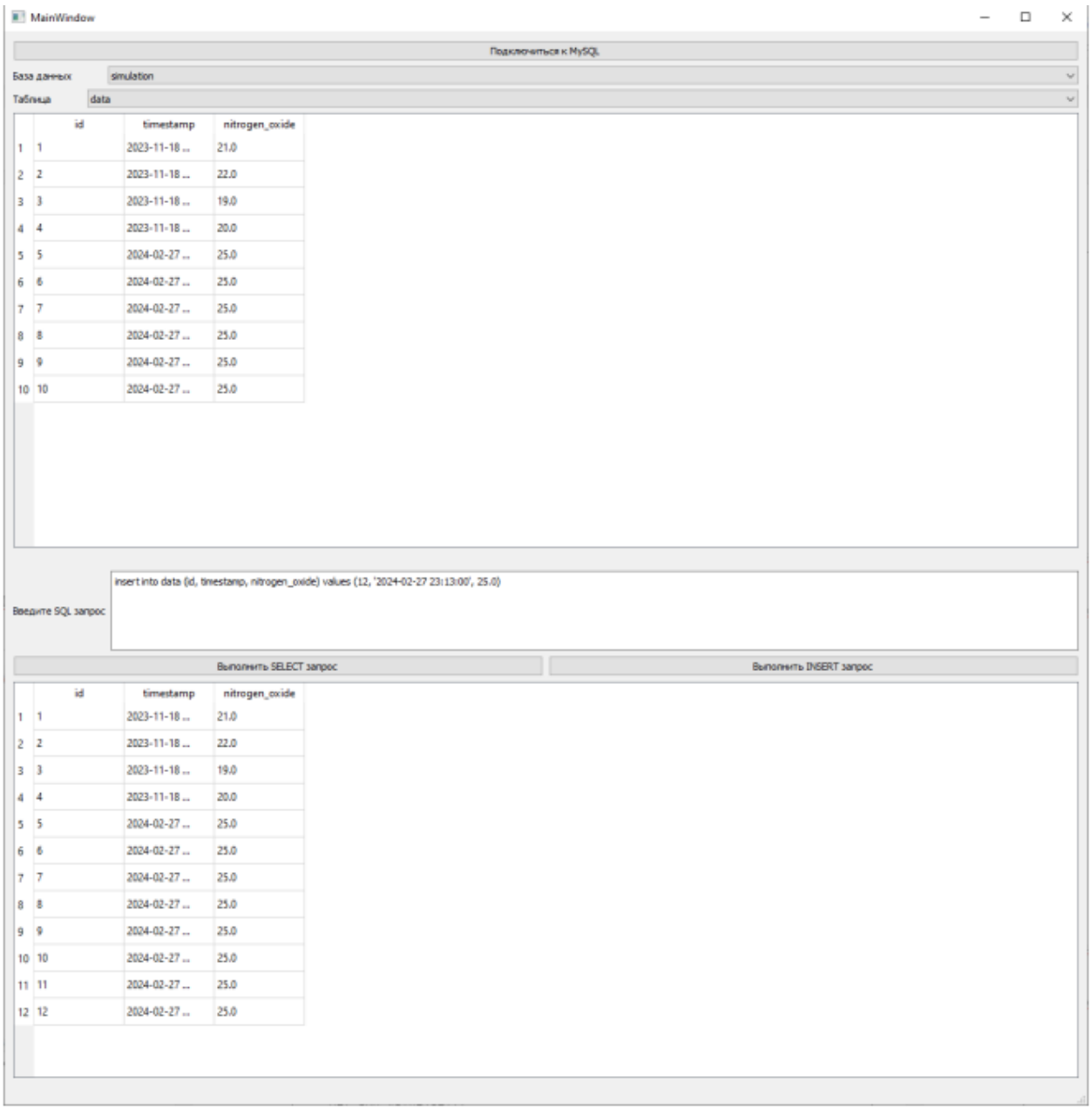


Рисунок 2.7.2 – Интерфейс приложения

При нажатии на кнопку «Подключиться к MySQL» происходит подключение к БД и загружаются все таблицы, которые имеются на сервере. Также подтягиваются и таблицы из каждой базы данных. На Рисунке 2.7.2 представлено выполнение запроса на добавление данных. На Рисунке 2.7.3 представлено выполнение выборки данных, а именно подсчет суммы значений оксида азота. Как можно заметить, результат выводится в виде таблицы и работают агрегированные функции, а также алиасы.

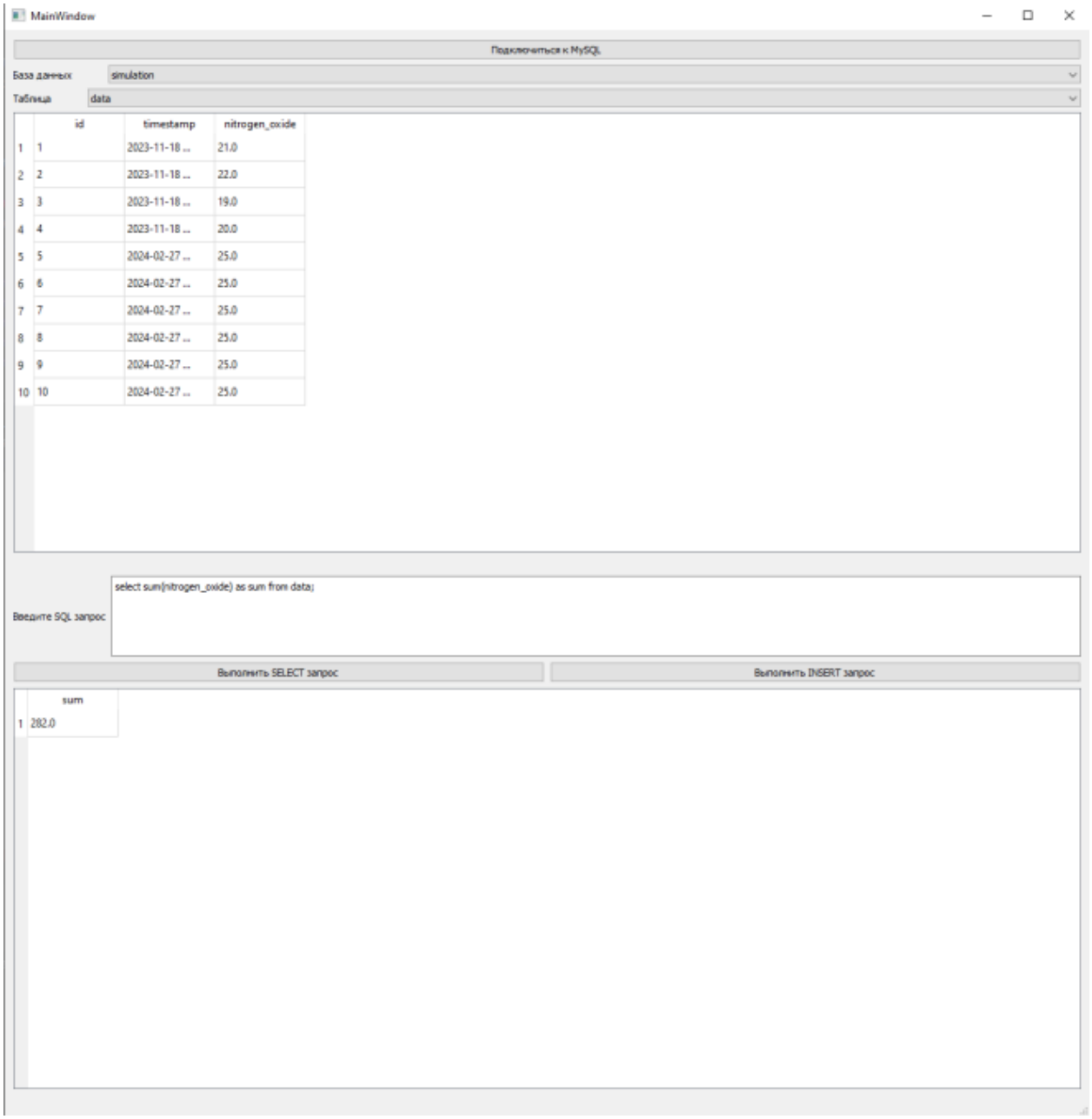


Рисунок 2.7.3 – Полученный результат

Листинг 2.7.2 – Исходный код программы

```
import sys
import mysql.connector as mc
from PyQt5 import uic, QtWidgets

class App(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi('main.ui', self)
        self.connect.clicked.connect(self.connect_to_db)
        self.databases.currentIndexChanged.connect(self.show_tables)
        self.tables.currentIndexChanged.connect(self.show_info)
        self.select.clicked.connect(self.select_query)
        self.insert.clicked.connect(self.add_value)
```

```
    def add_value(self):
        array = []
        try:
            with mc.connect(
                    host="localhost",
                    user="roman",
                    password="...",
                    database=f'{self.databases.currentText()}'
            ) as connection:
                insert_query = str(self.query.toPlainText())
                select_query = f'select * from {self.tables.currentText()}'
                with connection.cursor() as cursor:
                    cursor.execute(insert_query)
                    connection.commit()
                    cursor.execute(select_query)
                    self.result_query.setRowCount(0)
                    for lineIndex, lineData in enumerate(cursor):
                        if lineIndex == 0:
                            self.result_query.setColumnCount(len(lineData))
                            self.result_query.insertRow(lineIndex)
                        for columnIndex, columnData in enumerate(lineData):
                            self.result_query.setItem(lineIndex, columnIndex,

QtWidgets.QTableWidgetItem(str(columnData)))
                            if cursor.description[columnIndex][0] not in array:
                                array.append(cursor.description[columnIndex][0])
                    self.result_query.setHorizontalHeaderLabels(array)
        except mc.Error as e:
            print(e)

    def select_query(self):
        array = []
        try:
            with mc.connect(
                    host="localhost",
                    user="ilya",
                    password="...",
                    database=f'{self.databases.currentText()}'
            ) as connection:
                query = str(self.query.toPlainText())
                with connection.cursor() as cursor:
                    cursor.execute(query)
                    self.result_query.setRowCount(0)
                    for lineIndex, lineData in enumerate(cursor):
                        if lineIndex == 0:
                            self.result_query.setColumnCount(len(lineData))
                            self.result_query.insertRow(lineIndex)
                        for columnIndex, columnData in enumerate(lineData):
                            self.result_query.setItem(lineIndex, columnIndex,

QtWidgets.QTableWidgetItem(str(columnData)))
                            if cursor.description[columnIndex][0] not in array:
                                array.append(cursor.description[columnIndex][0])
                    self.result_query.setHorizontalHeaderLabels(array)
        except mc.Error as e:
            print(e)

    def show_info(self):
        array = []
        try:
```

```
            with mc.connect(
                    host="localhost",
                    user="roman",
                    password="...",
                    database=f'{self.databases.currentText()}'
            ) as connection:
                show_info_query = f"select * from {self.tables.currentText()} limit
10;"
                with connection.cursor() as cursor:
                    cursor.execute(show_info_query)
                    self.table_info.setRowCount(0)
                    for lineIndex, lineData in enumerate(cursor):
                        if lineIndex == 0:
                            self.table_info.setColumnCount(len(lineData))
                            self.table_info.insertRow(lineIndex)
                        for columnIndex, columnData in enumerate(lineData):
                            self.table_info.setItem(lineIndex, columnIndex,

QtWidgets.QTableWidgetItem(str(columnData)))
                            if cursor.description[columnIndex][0] not in array:
                                array.append(cursor.description[columnIndex][0])
                    self.table_info.setHorizontalHeaderLabels(array)
        except mc.Error as e:
            print(e)

    def show_tables(self):
        try:
            with mc.connect(
                    host="localhost",
                    user="ilya",
                    password="...",
                    database=f'{self.databases.currentText()}'
            ) as connection:
                show_tables_query = "show tables;"
                with connection.cursor() as cursor:
                    cursor.execute(show_tables_query)
                    self.tables.clear()
                    for table in cursor:
                        self.tables.addItem(str(table[0]))
        except mc.Error as e:
            print(e)

    def connect_to_db(self):
        try:
            with mc.connect(
                    host="localhost",
                    user="ilya",
                    password="..."
            ) as connection:
                show_db_query = "show databases;"
                with connection.cursor() as cursor:
                    cursor.execute(show_db_query)
                    self.databases.clear()
                    for db in cursor:
                        self.databases.addItem(str(db[0]))
        except mc.Error as e:
            print(e)

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    ex = App()
```

```
    ex.show()
    sys.exit(app.exec())
```

Заключение

В ходе практической работы были закреплены знания языка Python.

Практическая работа №2.8

Цель работы

Научиться работать со списками

Задание

1. Дан список некоторых целых чисел, найдите значение 20 в нем и, если оно присутствует, замените его на 200. Обновите список только при первом вхождении числа 20.
2. Необходимо удалить пустые строки из списка строк.
3. Дан список чисел. Превратите его в список квадратов этих чисел.
4. Дан список чисел, необходимо удалить все вхождения числа 20 из него.

Выполнение работы

Дан список некоторых целых чисел, найдите значение 20 в нем и, если оно присутствует, замените его на 200. Обновите список только при первом вхождении числа 20. В качестве тестового списка возьмем [10, 20, 30, 20, 40]. Код представлен на Листинге 2.8.1. В результате выполнения программы получаем следующий список: [10, 200, 30, 20, 40].

Необходимо удалить пустые строки из списка строк. В качестве примера возьмем следующий список ["hello", None, "world", " ", "python"]. Код представлен на Листинге 2.8.1. В результате выполнения программы получаем следующий список: ['hello', 'world', ' ', 'python'].

Дан список чисел. Превратите его в список квадратов этих чисел. В качестве примера возьмем следующий список [1, 2, 3, 4, 5]. Код представлен на Листинге 2.8.1. В результате выполнения программы получаем следующий список: [1, 4, 9, 16, 25].

Дан список чисел, необходимо удалить все вхождения числа 20 из него. В качестве примера возьмем следующий список [10, 20, 30, 20, 40]. Код

представлен на Листинге 2.8.1. В результате выполнения программы получаем следующий список: [10, 30, 40].

Листинг 2.8.1 – Исходный код программы

```
# Задание 1
numbers = [10, 20, 30, 20, 40]
if 20 in numbers:
 index_20 = numbers.index(20)
 numbers[index_20] = 200
print(numbers)

# Задание 2
strings = ["hello", None, "world", " ", "python"]
filtered_strings = list(filter(lambda item: item is not None, strings))

print(filtered_strings)

# Задание 3
numbers = [1, 2, 3, 4, 5]
squared_numbers = [x**2 for x in numbers]
print(squared_numbers)

# Задание 4
numbers = [10, 20, 30, 20, 40]
numbers_without_20 = [x for x in numbers if x != 20]
print(numbers_without_20)
```

Заключение

В ходе практической работы были закреплены знания языка Python, а также была освоена технология работы со списками.