

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего  
образования

**«Вятский государственный университет»**  
**(ФГБОУ ВО «ВятГУ»)**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Допущено к защите

Руководитель проекта

\_\_\_\_\_/Крутиков А. К./

«\_\_» \_\_\_\_\_ 2024г.

Разработка информационной системы виртуального банка.

Пояснительная записка курсового проекта по дисциплине

«Комплекс знаний бакалавра»

ТПЖА.090302.554 ПЗ

Разработал студент группы ИВТб-3301

\_\_\_\_\_/Абрамович А. В./

Руководитель

\_\_\_\_\_/Крутиков А. К./

Консультант

\_\_\_\_\_/Долженкова М.Л./

Проект защищен с оценкой «\_\_\_\_\_» \_\_\_\_\_  
(оценка) (дата)

Члены комиссии \_\_\_\_\_ / \_\_\_\_\_/  
(подпись) (ФИО)

\_\_\_\_\_ / \_\_\_\_\_/  
(подпись) (ФИО)

\_\_\_\_\_ / \_\_\_\_\_/  
(подпись) (ФИО)

Киров 2024

## Реферат

Абрамович А. В. Разработка информационной системы виртуального банка. ТПЖА.090301.554 ПЗ: Курс. проект / ВятГУ, каф. ЭВМ; рук. Крутиков А. К. - Киров, 2024. – ПЗ 17 стр., 11 рис., 0 прил.

Объект курсового проекта — бизнес-процесса документооборота отчётов по лабораторным работам.

Предмет курсового проекта — информационная система.

Цель курсового проекта — проектирование информационной системы, которая позволит оптимизировать бизнес-процесс документооборота отчётов по лабораторным работам.

Результатом выполнения курсового проекта является информационной системы.

# Содержание

<b>1</b>	<b>Введение</b>	<b>5</b>
1.1	Цель . . . . .	5
1.2	Задача . . . . .	5
<b>2</b>	<b>Обзор предметной области</b>	<b>6</b>
2.1	Обоснование актуальности темы . . . . .	6
2.2	Анализ существующих аналогов. . . . .	6
2.3	Постановка задачи . . . . .	8
<b>3</b>	<b>Декомпозиция</b>	<b>10</b>
3.1	Клиент . . . . .	10
3.2	Сервер . . . . .	11
<b>4</b>	<b>Docker</b>	<b>12</b>
4.1	Что такое Docker? . . . . .	12
4.2	Взаимодействие с Docker . . . . .	13
<b>5</b>	<b>Разработка алгоритма взаимодействия с сервером</b>	<b>14</b>
<b>6</b>	<b>Разработка интерфейса</b>	<b>15</b>
6.1	Чем должен обладать интерфейс? . . . . .	15
<b>7</b>	<b>Реализация</b>	<b>16</b>
7.1	Предоставление доступа к Docker . . . . .	16
7.2	Запуск контейнера на удалённом сервере . . . . .	16
7.3	Реализация обмена сообщениями между клиентом и сервером . . .	17
7.4	Отправка пакета вычислений . . . . .	18
7.5	Запуск вычислений . . . . .	20
7.6	Результаты вычислений . . . . .	21

Подп. и дата		Инв. № дубл.	Взам. инв. №	Подп. и дата	ТПЖА.090301.554 ПЗ				
Инв. № подл.	Изм.	Лист	№ докум.	Подп.	Дата	Разработка системы управления бытовыми приборами	Лит.	Лист	Листов
	Разраб.	Абрамович						3	29
	Пров.	Долженкова							
	Н. контр.								
	УТВ.								
							Кафедра ЭВМ ИВТ6-3301-04-01		

7.7	Реализация интерфейса . . . . .	23
8	Заключение	24
9	Библиографический список	25
10	Приложение	26

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата	<div>ТПЖА.090301.554 ПЗ</div>	Лист
						4
Изм.	Лист	№ докум.	Подп.	Дата		

# 1 Введение

Разработчики часто сталкиваются с ситуациями, когда их персональные устройства не способны справиться с требуемыми вычислительными нагрузками.

Нехватка вычислительных ресурсов ограничивает возможности выполнения сложных задач, таких как обработка больших данных, моделирование и машинное обучение на локальном компьютере. Это приводит к увеличению времени выполнения задач, а значит и к снижению производительности труда.

Один из способов решения этой проблемы - использование вычислительных ресурсов удалённой машины. В таком случае от локального ПК требуется только предоставить способ связи с удалённым сервером. Это позволит не только значительно ускорить процесс обработки данных, но и сократить затраты на обновление локального оборудования.

## 1.1 Цель

Предоставить пользователю возможность использовать ресурсы удалённого ПК для вычислений.

## 1.2 Задача

Разработать программное обеспечение, позволяющего выполнять программы на удалённом ПК.

Это позволит разгрузить локальные ресурсы устройства и задействовать более производительный компьютер вне зависимости от местоположения пользователя. Также благодаря этому ПО станет возможна разработка более требовательных к вычислительным мощностям приложений.

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата	Предоставить пользователю возможность использовать ресурсы удалённо-го ПК для вычислений.
					1.2 Задача
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата	Разработать программное обеспечение, позволяющего выполнять програм-мы на удалённом ПК.
					Это позволит разгрузить локальные ресурсы устройства и задействовать более производительный компьютер вне зависимости от местоположения пользователя. Также благодаря этому ПО станет возможна разработка более требовательных к вычислительным мощностям приложений.
Изм.	Лист	№ докум.	Подп.	Дата	ТПЖА.090301.554 ПЗ
					Лист 5

## 2 Обзор предметной области

Существует множество вариантов как решить данную проблему. В этом разделе они будут рассмотрены.

### 2.1 Обоснование актуальности темы

Для решения некоторых задач необходимо использовать большие вычислительные мощности. Например, обучение нейронных сетей, которое происходит за счёт вычислительных мощностей графического ускорителя. Производительности графического ускорителя локального устройства не достаточно для решения этой задачи за приемлемое время. В таких случаях можно использовать следующие сервисы.

### 2.2 Анализ существующих аналогов.

а) **Удалённый рабочий стол** позволяет пользователям удаленно подключаться к другому компьютеру и управлять им, используя свой локальный компьютер. Он предоставляет возможность выполнения кода на удаленном компьютере, однако требует надежного интернет-соединения. Также удаленный рабочий стол не предоставляет механизма автоматической загрузки и выполнения кода на удаленном ПК, все эти действия приходится выполнять пользователю вручную.

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата					
Изм.	Лист	№ докум.	Подп.	Дата	ТПЖА.090301.554 ПЗ		Лист		
							6		

б) **Google Colab** - это сервис, предоставляемый Google, который позволяет пользователям запускать код Python в облачной среде. Он предоставляет вычислительные ресурсы и возможность совместной работы над кодом. Однако он ограничен только выполнением кода на языке Python и не предоставляет возможности загрузки и выполнения кода на удаленном ПК, работающем на других языках программирования. К тому же вычислительные мощности ограничены, а для их расширения необходимо покупать подписку.

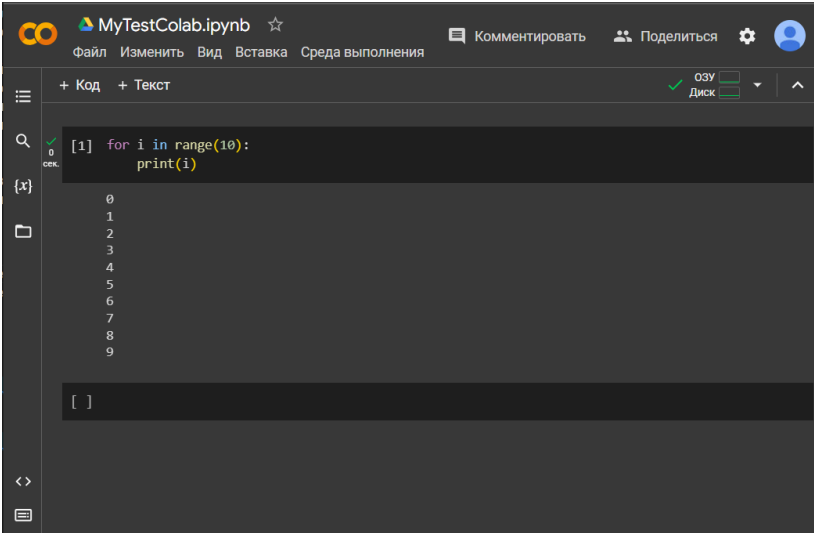


Рисунок 1 – Google Colab

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата					
Изм.	Лист	№ докум.	Подп.	Дата	ТПЖА.090301.554 ПЗ				
					Лист				
					7				

в) **Jupyter Notebook** - это интерактивная среда разработки, которая позволяет пользователям создавать и выполнять код в виде блокнотов. По своей сути это решения аналогично Google Colab лишь с той разницей, что этот вариант предоставляет возможность выполнения кода не только на Python, но и на JavaScript, Lua, SQLite. Однако минусы такого решения аналогичны предыдущему - вычислительные мощности ограничены, запускать код можно только на вышеперечисленных языках.

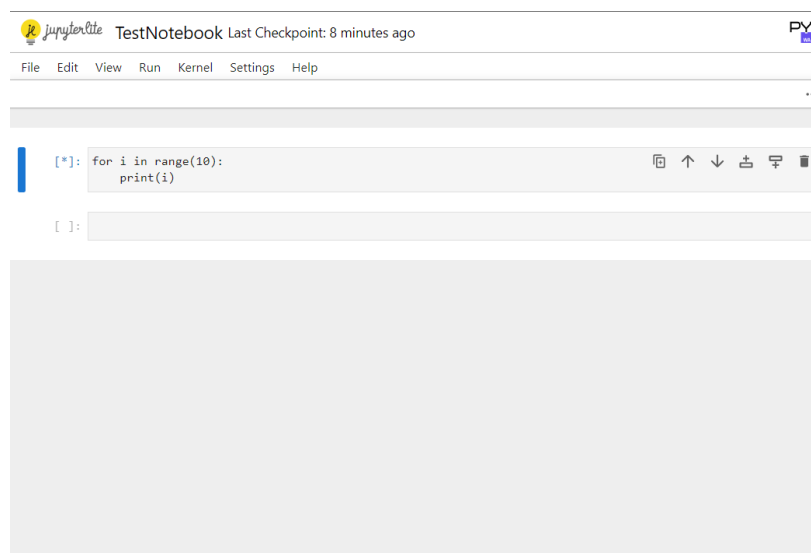


Рисунок 2 – Jupyter Notebook

## 2.3 Постановка задачи

Разработать ПО, которое позволит выполнять программы на удалённом ПК. Программа должна автоматически формировать запросы на выполнение без необходимости доступа к рабочему столу удалённой машины.

При использовании такого способа выполнения кода необходимо учитывать время передачи запроса, т.к., например, в случае обучения нейронных сетей запрос будет включать файлы обучающей выборки. Если их объём будет достаточно большим, выполнить обучение будет быстрее на локальном устройстве. При анализе предметной области были определены функции, которыми должна обладать итоговая программа.

Изм.	Лист	№ докум.	Подп.	Дата	ТПЖА.090301.554 ПЗ	Лист
						8



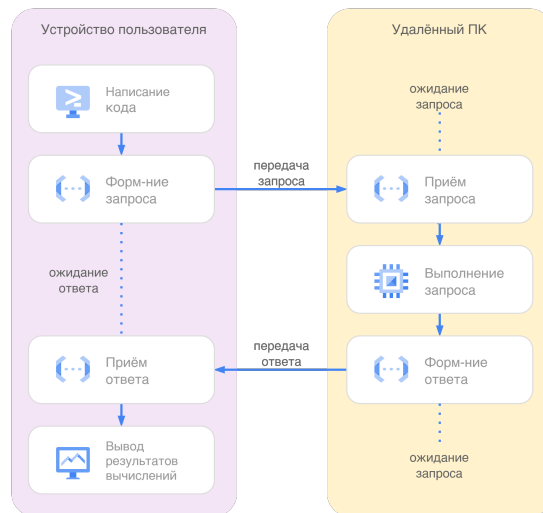


Рисунок 3 – Схема работы программы

- Автоматически формировать запросы к удалённому ПК на выполнение программ;
- Выполнять программы на языке программирования Python;
- Возвращать результат работы программы - новые файлы и вывод консоли.

Инв. № подл.	Изм.	Лист	№ докум.	Подп.	Дата	ТПЖА.090301.554 ПЗ	Лист
							9
Взам. инв. №							
Инв. № дубл.							
Подп. и дата							
Подп. и дата							

### 3 Декомпозиция

Для реализации вышеописанной задачи Была выбрана клиент-серверная архитектура, где в качестве клиента будет выступать устройство пользователя, а сервера - удалённый ПК.

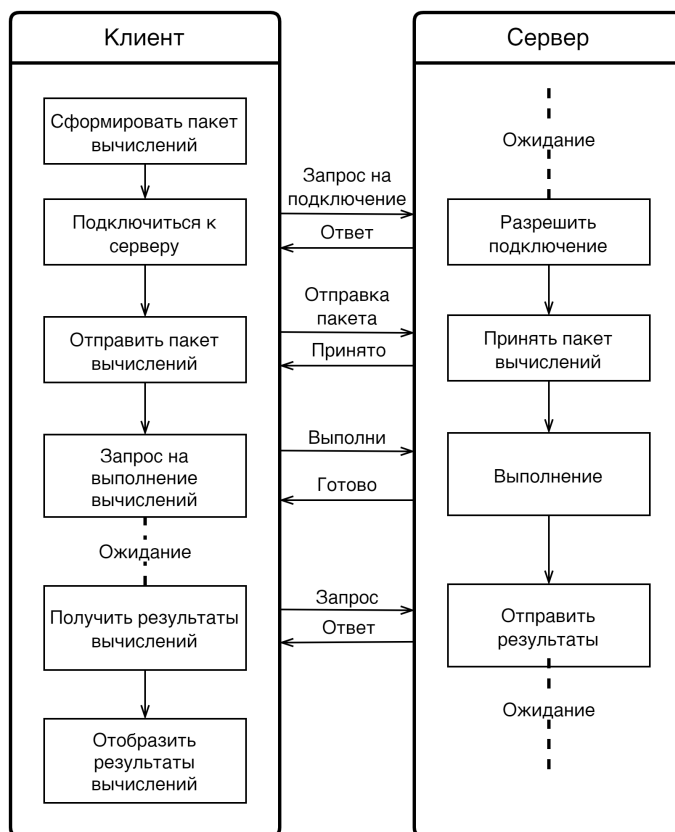
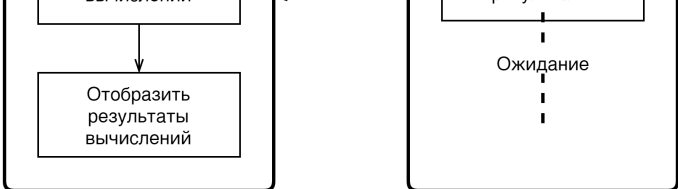


Рисунок 4 – Схема клиент-серверного взаимодействия

В ходе декомпозиции были выделены следующий функциональные блоки:

#### 3.1 Клиент

- Сформировать пакет для вычислений, содержащий все зависимости и рабочие файлы.
- Подключиться к удаленному серверу.
- Передать пакет для вычислений.
- Принять ответ от сервера.

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата	
<p>Рисунок 4 – Схема клиент-серверного взаимодействия</p>					
<p>В ходе декомпозиции были выделены следующий функциональные блоки:</p>					
<h3>3.1 Клиент</h3>					
<p>а) Сформировать пакет для вычислений, содержащий все зависимости и рабочие файлы.</p>					
<p>б) Подключиться к удаленному серверу.</p>					
<p>в) Передать пакет для вычислений.</p>					
<p>г) Принять ответ от сервера.</p>					
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата	<div>ТПЖА.090301.554 ПЗ</div>
Изм.	Лист	№ докум.	Подп.	Дата	Лист
					10

д) Выдать пользователю результат.

### 3.2 Сервер

а) Сформировать соединение с клиентом.

б) Принять пакет для вычислений.

в) Установить необходимые зависимости.

г) Выполнить необходимую программу.

д) Сформировать ответ.

е) Отправить ответ.

Инв. № подл.	Подп. и дата				Инв. № дубл.	Подп. и дата					
Взам. инв. №											
Подп. и дата											
Инв. № подл.											
Изм	Лист	№ докум.	Подп.	Дата	ТПЖА.090301.554 ПЗ					Лист	
										11	

## 4 Docker

Для выполнения скриптов нам нужно передавать их на удалённую машину (сервер). Кроме этого для их выполнения могут потребоваться некоторые дополнительные файлы (входные данные, библиотеки). Таким образом, если все скрипты будут запускаться в одном окружении, могут возникнуть конфликты библиотек разных версий. Кроме этого, в долгосрочной перспективе сервер должен уметь удалять ненужные файлы.

Для решения этих проблем существует *контейнеризация*. Так называется подход организации выполнения программного обеспечения, при котором каждое отдельное приложение со всеми необходимыми дополнительными файлами помещается в изолированную среду - *контейнер*.

Для автоматизации работы с контейнерами существуют специальные программы - *оркестраторы*. Они позволяют автоматически управлять контейнерами.

## 4.1 Что такое Docker?

Docker - это популярная платформа для оркестрации контейнеров с открытым исходным кодом. С его помощью можно управлять контейнерами в том числе и на удалённом ПК, что актуально для реализации данного проекта.

В данном проекте будет уместно использовать Docker в качестве прослойки между глобальным окружением системы и окружением, внутри которого будут запускаться программы. Код для исполнения будет помещаться внутрь контейнера со всеми необходимыми зависимостями и файлами, после чего исполняться средствами контейнера. При окончании внутри контейнера будет формироваться ответ, который будет отправлен клиенту.

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата	<p>Docker - это популярная платформа для оркестрации контейнеров с открытым исходным кодом. С его помощью можно управлять контейнерами в том числе и на удалённом ПК, что актуально для реализации данного проекта.</p> <p>В данном проекте будет уместно использовать Docker в качестве прослойки между глобальным окружением системы и окружением, внутри которого будут запускаться программы. Код для исполнения будет помещаться внутрь контейнера со всеми необходимыми зависимостями и файлами, после чего исполняться средствами контейнера. При окончании внутри контейнера будет формироваться ответ, который будет отправлен клиенту.</p>
Изм.	Лист	№ докум.	Подп.	Дата	<p>ТПЖА.090301.554 ПЗ</p>
					Лист
					12

## 4.2 Взаимодействие с Docker

В силу своей популярности Docker имеет множество вариантов взаимодействия с приложениями посредством API (Application Programming Interface). Подобные программные интерфейсы существуют для многих языков программирования.

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата	<div>ТПЖА.090301.554 ПЗ</div>					Лист				
										13				
										Изм.	Лист	№ докум.	Подп.	Дата

## 5 Разработка алгоритма взаимодействия с сервером

Для реализации вычислений в изолированном контейнере, клиент должен иметь возможность управлять контейнером - запускать, останавливать и получать консольный вывод. Всё это позволяет сделать Docker API, про который было сказано выше.

Кроме этого, нужно иметь возможность передавать файлы непосредственно внутрь контейнера. Для этого было принято решение использовать сокеты, через которые осуществлять передачу файлов и запуск исполнения кода.

Обмена информации клиента с Docker и контейнером будет производиться посредством протокола TCP. Для возможности такого взаимодействия необходимо выделить порты для Docker и контейнера.

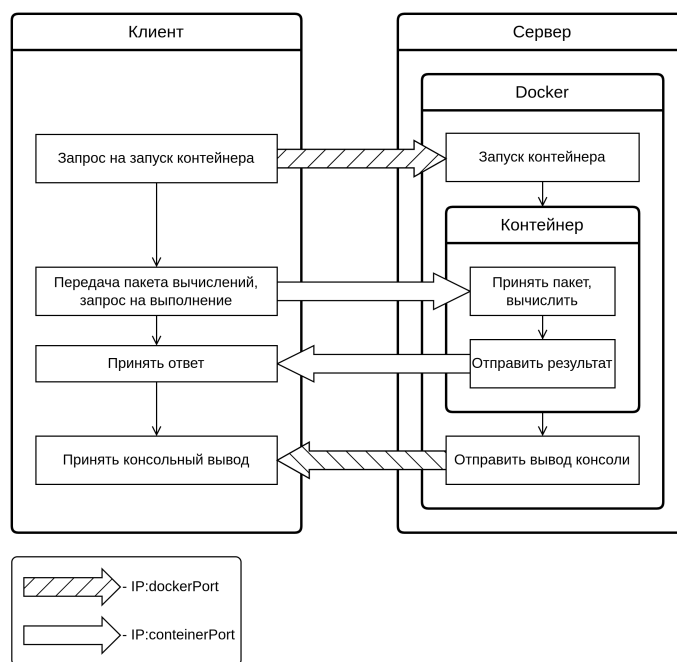


Рисунок 5 – Схема взаимодействия с Docker и контейнером на сервере

На схеме указано какая пара IP адреса и порта будет использоваться при обмене информацией.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм.	Лист	№ докум.	Подп.	Дата

ТПЖА.090301.554 ПЗ

Лист

14

Ввод IP адреса
Выбор файла для исполнения, зависимостей, дополнительных файлов
Результат выполнения
Файлы после выполнения

Рисунок 6 – Макет интерфейса

## 6 Разработка интерфейса

В следствии обзора возможных инструментов реализации интерфейса был выбран Qt. Это кросс-платформенный фреймворк для разработки графическим интерфейсом. Он обладает достаточным функционалом для реализации проекта.

### 6.1 Чем должен обладать интерфейс?

В ходе анализа были выделены следующие функции, которыми должен обладать графический интерфейс:

- а) Выбор сервера для подключения путём ввода IP-адреса и порта
- б) Выбор файлов, отправляемых на сервер для выполнения
- в) Возможность отображение результатов выполнения программы в том числе файлы, которые появились после выполнения
- г) Возможность запросить сервер отправить файл назад

Изм.	Лист	№ докум.	Подп.	Дата	ТПЖА.090301.554 ПЗ	Лист
						15

## 7 Реализация

### 7.1 Предоставление доступа к Docker

Для предоставления доступа к Docker, необходимо дополнить конфигурационный файл `docker.service` следующими строками:

```
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://127.0.0.1:2375
```

Этим действием мы открыли порт 2375 на машине для удалённого управления локальным Docker. Передача данных будет производиться по средствам протокола TCP.

Для подключения к этому порту с удалённой машины, необходимо указать IP и порт удалённой машины в следующем виде:

```
desktop = docker.DockerClient(base_url=f"tcp://*ip*:порт*}")
```

Таким образом переменная `desktop` будет ссылаться на объект `DockerClient`, по средствам которого можно управлять удалённым Docker.

### 7.2 Запуск контейнера на удалённом сервере

Запуск контейнера происходит по средствам следующего кода:

```
container = desktop.containers.run(
    image="rexelserv",
    ports={"10500": ("localhost", ContainerPort)},
    detach=True,
)
```

Изм.	Лист	№ докум.	Подп.	Дата	ТПЖА.090301.554 ПЗ	Лист
						16
Инов. № подл.	Подп. и дата	Взам. инв. №	Инов. № дубл.	Подп. и дата		





```

case rl.MSG.SendFile:
    rl.ReceiveFile(conn, targetFolder)

case rl.MSG.SendConf:
    conf = rl.ReceiveConf(conn)

case rl.MSG.Run:
    exitCodeReq, exitCodeMain, dirDiff = rl.Run(conf)

case rl.MSG.ReturnResult:
    rl.SendResult(conn, exitCodeReq, exitCodeMain, dirDiff)

case rl.MSG.SendFileBack:
    rl.SendFileBack(conn)

case rl.MSG.Break:
    break

case _:
    break

```

## 7.4 Отправка пакета вычислений

После выбора пользователем файла для исполнения, файла зависимостей и дополнительных файлов, необходимо сформировать пакет вычислений и отправить его на сервер. Класс для пакеты следующий

```

class Config:
    def __init__(self, req="", main="", sendFiles=[], sendDirs=[]):
        self.req = req
        self.main = main
        self.sendFiles = sendFiles
        self.sendDirs = sendDirs

```

После формирования пакета необходимо отправить его на сервер. Для этого необходимо сериализовать пакет вычислений, т.е. представить пакет в виде

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата	<div>ТПЖА.090301.554 ПЗ</div>					Лист				
										18				
										Изм.	Лист	№ докум.	Подп.	Дата

байтовой последовательности. Для сериализации будет использоваться встроенный инструмент Python - библиотеку json.

```
def SendConf(conn: socket.socket, conf: Config) -> int:
    if SendMessage(conn, MSG.SendConf) == 0:
        return 0
    data = json.dumps(conf.__dict__())
    conn.sendall(bytes(data, encoding=FORMAT))
    conn.send(MSG.Null)
    conn.recv(ChunkSize)

    return 1
```

На стороне сервера необходимо принять эту последовательность и десериализовать.

```
def ReceiveConf(conn: socket.socket) -> Config:
    data = ""
    while (tempData := conn.recv(ChunkSize)) != MSG.Null:
        data += tempData.decode(FORMAT)

    conn.send(b"1")
    req, main, sendFiles, sendDirs = json.loads(data).values()
    conf = Config(req, main, sendFiles, sendDirs)
    return conf
```

Функция для отправки файла:

```
def SendFile(conn: socket.socket, fileName: str, fileDest="") -> int:
    try:
        file = open(fileName, "rb")
    except OSError:
        return 0

    fileNameToServ = os.path.join(fileDest, os.path.basename(fileName))

    if SendMessage(conn, MSG.SendFile) == 0:
        file.close()
        return 0
```

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подп.	Дата

ТПЖА.090301.554 ПЗ

```

conn.send((bytes(fileNameToServ, encoding=FORMAT)))
conn.recv(ChunkSize)

while data := file.read(ChunkSize):
    conn.send(data)
    conn.recv(1)

conn.send(MSG.Null)
conn.recv(ChunkSize)
file.close()
return 1

```

Приём файла на стороне сервера:

```

def ReceiveFile(conn: socket.socket, targetFolder="./") -> int:
    fileName = os.path.normpath((conn.recv(ChunkSize)).decode(FORMAT))
    fileName = os.path.join(targetFolder, fileName)
    conn.send(b"filename was received")
    try:
        os.makedirs(os.path.split(fileName)[0], exist_ok=True)
        file = open(fileName, "wb")
    except OSError:
        return 0
    while (data := conn.recv(ChunkSize)) != MSG.Null:
        file.write(data)
        conn.send(b"1")
    conn.send(b"done")
    file.close()
    return 1

```

## 7.5 Запуск вычислений

Функция отправки запроса на запуск вычислений на клиентской части:

```

def goRun(conn: socket.socket):

```

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата					
Изм.	Лист	№ докум.	Подп.	Дата	ТПЖА.090301.554 ПЗ				
					Лист				
					20				

```
return SendMessage(conn, MSG.Run)
```

Обработка запроса на серверной части:

```
def Run(conf: Config) -> tuple[int, int, list]:
    if len(conf.req) > 0:
        exitCodeReq = os.system(f"pip install -r {conf.req}")
    else:
        exitCodeReq = 0
    dirBefore = DirectoryList()
    if len(conf.main) > 0:
        exitCodeMain = os.system(f"python3 {conf.main}")
    else:
        exitCodeMain = 1

    dirAfter = DirectoryList()
    dirDiff = DirectoryDiff(dirBefore, dirAfter)

    return exitCodeReq, exitCodeMain, dirDiff
```

## 7.6 Результаты вычислений

После выполнения программы на удалённой машине необходимо вернуть результаты вычислений. Результатом являются не только код возврата, но и консольный вывод программы и файлы, которые появились после выполнения. В функции *Run*, описанной выше, уже используется функция получения списка новых файлов *DirectoryDiff* и *DirectoryList*.

```
def DirectoryList(target=".") -> list:
    dirList = list()
    for root, dirs, files in os.walk(target):
        for file in files:
            dirList.append(os.path.join(root, file))
    return dirList
```

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата	<div>ТПЖА.090301.554 ПЗ</div>					Лист	
Изм.	Лист	№ докум.	Подп.	Дата						21	

```
def DirectoryDiff(before: list, after: list) -> list:
    return list(set(after) - set(before))
```

После этого необходимо сформировать полноценный ответ и отправить его:

```
def SendResult(conn: socket.socket, exitCodeReq, exitCodeMain, dirDiff):

    sendStr = str(exitCodeReq) + " " + str(exitCodeMain) + " " + str(len(dirDiff))

    conn.send(bytes(sendStr, encoding=FORMAT))
    conn.recv(1)

    for fileName in dirDiff:
        conn.send(bytes(fileName, encoding=FORMAT))
        conn.recv(ChunkSize)

    return 0
```

Получение кодов возврата и имён полученных файлов на клиентской части:

```
def ReturnResult(conn: socket.socket) -> tuple[int, int, list[str]]:
    SendMessage(conn, MSG.ReturnResult)

    recvData = map(int, conn.recv(ChunkSize).decode(FORMAT).split())
    exitCodeReq, exitCodeMain, countOfNewFiles = recvData
    conn.send(b"1")

    dirDiff = []
    for i in range(countOfNewFiles):
        fileName = conn.recv(ChunkSize).decode(FORMAT)
        conn.send(b"1")
        dirDiff.append(fileName)

    return exitCodeReq, exitCodeMain, dirDiff
```

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Изм	Лист	№ докум.	Подп.	Дата

ТПЖА.090301.554 ПЗ

Получение непосредственно самих файлов реализовано через функции *SendFile* и *ReceiveFile*, описанные выше.

Получение консольного вывода происходит с помощью встроенных возможностей DockerAPI непосредственно на клиенте.

```
output = str(container.logs(), encoding="utf-8")
```

## 7.7 Реализация интерфейса

Используя встроенные виджеты библиотеки PySide6, был реализован следующий интерфейс программы. Следует отметить, что блоки "Результат выполнения" и "Файлы после выполнения" выделенные при построении макета интерфейса удалось объединить в один блок, используя вкладки. (Рис. 7)

Функционал данной библиотеки позволяет настроить приложение таким образом, чтобы оно изменяло свою цветовую гамму в зависимости от темы операционной системы. (Рис. 8)

При подключении к удалённому ПК будет отображена иконка, показывающая состояние подключения. (Рис. 9-11)

При успешном подключении и выбранном файле на исполнение, кнопка "Запустить" становится активной. (Рис. 12)

Код программы simpleProg.py, выбранной в качестве исполняемой:

```
with open("123.txt", "w") as file:
    file.write("123")
print("привет из докера")
```

После успешного выполнения программы в блоках "Вывод консоли" и "Файлы" отобразятся результаты выполнения программы. (Рис. 13, 14)

При возникновении ошибки в ходе выполнения программы, в "Вывод консоли" также отобразится место, где возникла ошибка. (Рис. 15)

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата	<div>ТПЖА.090301.554 ПЗ</div>					Лист	
Изм.	Лист	№ докум.	Подп.	Дата						23	

## 8 Заключение

В ходе выполнения данной курсовой работы была рассмотрена проблема ограниченности вычислительных ресурсов персональных устройств, с которой сталкиваются разработчики в процессе решения сложных задач. Был проанализирован способ преодоления этой проблемы — использование удалённых вычислительных ресурсов.

Реализация данного подхода позволяет обеспечить значительное ускорение обработки данных и выполнения вычислений, а также минимизировать затраты на обновление и модернизацию локального оборудования. Были рассмотрены различные технологии и инструменты, которые позволяют организовать взаимодействие между локальным устройством и удалённым сервером, а также ряд преимуществ, которые такие решения могут предоставить.

В результате исследования было установлено, что переход на удалённые вычисления не только оптимизирует рабочие процессы, но и открывает новые возможности для разработчиков в области ресурсоёмких задач. Это подтверждает актуальность поставленной проблемы и целесообразность поиска решений в области облачных технологий и распределённых вычислений.

Результатом курсовой работы стало приложение, обеспечивающее удобный доступ к ресурсам удалённого ПК. Оно может быть использовано в различных сценариях, где могут потребоваться большие вычислительные мощности, чем те, которыми обладает локальная машина пользователя.

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата	<div>ТПЖА.090301.554 ПЗ</div>					Лист				
										24				
										Изм.	Лист	№ докум.	Подп.	Дата



## 9 Библиографический список

Изм	Лист	№ докум.	Подп.	Дата	Инов. № подл.	Подп. и дата	Взам. инв. №	Инов. № дубл.	Подп. и дата
ТПЖА.090301.554 ПЗ					Лист 25				

10 Приложение

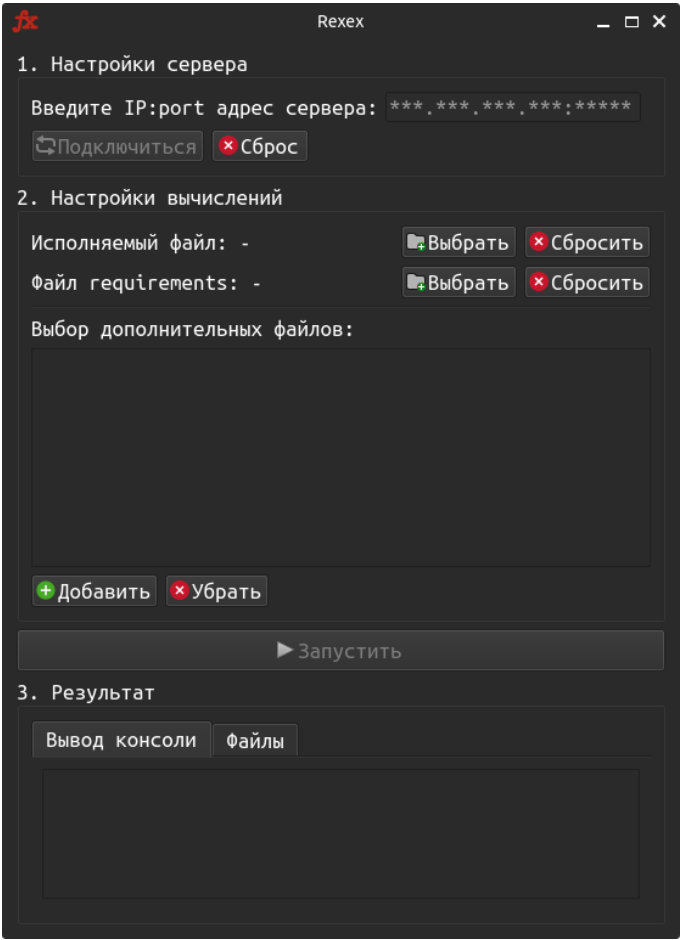


Рисунок 7 – Интерфейс на ОС Linux с тёмной темой

Инов. № подл.	Подп. и дата	Взам. инв. №	Инов. № дубл.	Подп. и дата

Изм.	Лист	№ докум.	Подп.	Дата

ТПЖА.090301.554 ПЗ	

Лист
26

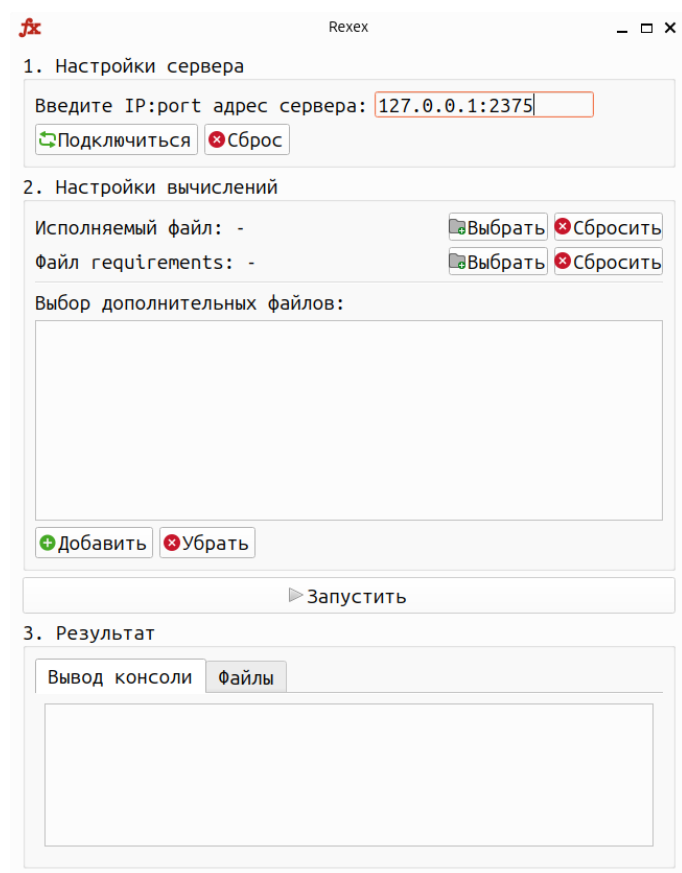


Рисунок 8 – Интерфейс на ОС Linux со светлой темой






Рисунок 9 – Иконка попытки подключения



Рисунок 10 – Иконка успешного подключения



Рисунок 11 – Иконка неудачного подключения

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
<div>Рисунок 8 – Интерфейс на ОС Linux со светлой темой</div>				
<div><div><p>Рисунок 9 – Иконка попытки подключения</p></div><div><p>Рисунок 10 – Иконка успешного подключения</p></div><div><p>Рисунок 11 – Иконка неудачного подключения</p></div></div>				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
<div>ТПЖА.090301.554 ПЗ</div>				
Изм	Лист	№ докум.	Подп.	Дата
				Лист
				27

The screenshot displays the 'Rexex' application window. At the top, the title bar reads 'Rexex'. The main interface is divided into three sections:

- 1. Настройки сервера (Server Settings):** Includes a text input for 'Введите IP:port адрес сервера: 127.0.0.1:2375'. Below it are buttons for 'Подключиться' (Connect), 'Сброс' (Reset), and a green cloud icon.
- 2. Настройки вычислений (Calculation Settings):** Features a text input for 'Исполняемый файл: /home/alex/Документы/Универ/Курсач/prog/files transfer/ex1/simpleProg.py'. Below this are buttons for 'Выбрать' (Select) and 'Сбросить' (Reset). Another row shows 'Файл requirements: -' with similar 'Выбрать' and 'Сбросить' buttons. A large empty box for 'Выбор дополнительных файлов:' (Additional file selection) is present, followed by 'Добавить' (Add) and 'Убрать' (Remove) buttons. At the bottom of this section is a large 'Запустить' (Run) button with a green play icon.
- 3. Результат (Result):** Contains two buttons: 'Вывод консоли' (Console output) and 'Файлы' (Files), positioned above a large empty box for displaying results.

The screenshot displays the Rexex application window, which is titled "Rexex". The interface is organized into several sections:

- 1. Настройки сервера (Server Settings):** This section contains a text input field for the "Введите IP:port адрес сервера:" (Enter IP:port address) with the value "127.0.0.1:2375". Below this are three buttons: "Подключиться" (Connect) with a green plus icon, "Сброс" (Reset) with a red X icon, and a green cloud icon.
- 2. Настройки вычислений (Calculation Settings):** This section includes a text area for the "Исполняемый файл:" (Executable file) with the path "/home/alex/Документы/Универ/Курсач/prog/files transfer/ex1/simpleProg.py". To the right are "Выбрать" (Select) and "Сбросить" (Reset) buttons. Below this is a "Файл requirements: -" (File requirements) section with similar "Выбрать" and "Сбросить" buttons.
- Выбор дополнительных файлов: (Select additional files):** This section has a large empty text area for listing additional files.
- Buttons:** At the bottom of this section are "Добавить" (Add) and "Убрать" (Remove) buttons. A large "Запустить" (Run) button is located at the bottom of the configuration section.
- 3. Результат (Result):** This section features a "Вывод консоли" (Console output) button and a "Файлы" (Files) button. Below these buttons, the text "привет из докера" (hello from docker) is displayed in the console output area.

The screenshot displays the 'Rexex' application window, which is used for running code in a Docker container. The interface is divided into several sections:

- 1. Настройки сервера (Server Settings):** A text input field contains the IP:port address '127.0.0.1:2375'. Below it are three buttons: 'Подключиться' (Connect), 'Сброс' (Reset), and a green circular icon.
- 2. Настройки вычислений (Calculation Settings):**
  - Исполняемый файл:** The path '/home/alex/Документы/Универ/Курсач/prog/files transfer/ex1/simpleProg.py' is entered. To the right are 'Выбрать' (Select) and 'Сбросить' (Reset) buttons.
  - Файл requirements:** The field is empty. To the right are 'Выбрать' (Select) and 'Сбросить' (Reset) buttons.
  - Выбор дополнительных файлов:** An empty text area for specifying additional files.
  - At the bottom of this section are 'Добавить' (Add) and 'Убрать' (Remove) buttons.
- Запустить (Run):** A large button with a play icon and the text 'Запустить'.
- 3. Результат (Result):**
  - Two tabs are present: 'Вывод консоли' (Console Output) and 'Файлы' (Files). 'Вывод консоли' is selected.
  - The console output shows the command being executed: 

```
File "/rexel/simpleProg.py", line 4
print("привет из докера")
```

 with a cursor pointing to the opening quote of the string. Below this, the error message is displayed: 

```
SyntaxError: '(' was never closed
```

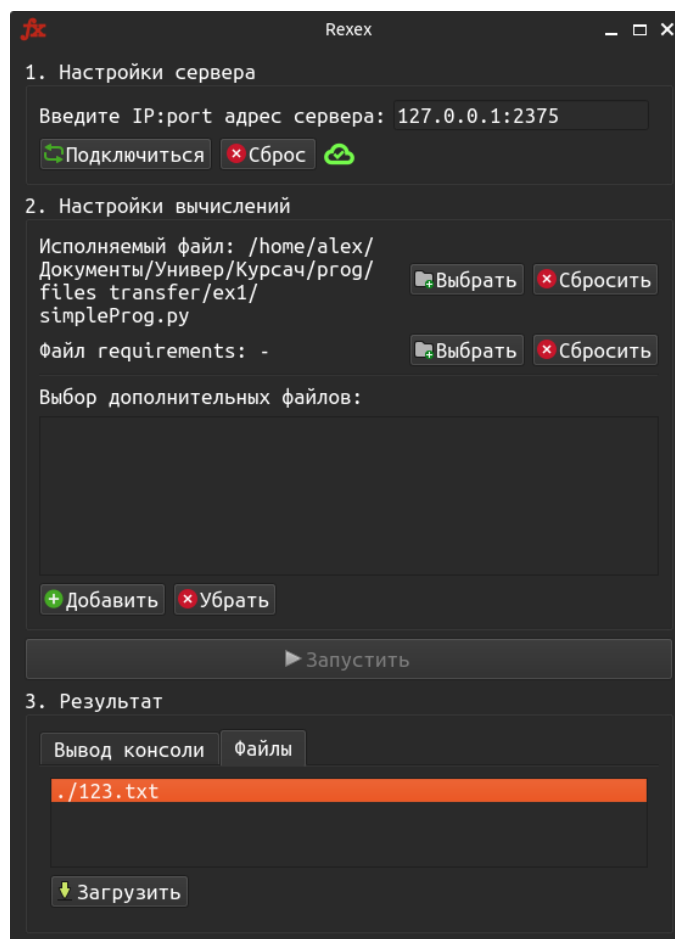


Рисунок 15 – Интерфейс при выполнении программы и ошибкой

Инв. № подл.	Подп. и дата				Изм	Лист	№ докум.	Подп.	Дата	ТПЖА.090301.554 ПЗ		Лист								
	Инв. № дубл.											29								
	Взам. инв. №																			

+ Добавить

✖ Убрать

▶ Запустить

3. Результат

Вывод консоли

Файлы

./123.txt

⬇ Загрузить

Рисунок 15 – Интерфейс при выполнении программы и ошибкой