МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

**КУРСОВОЙ ПРОЕКТ**

по дисциплине:” Компьютерные системы и сети”

на тему:” ***Сетевой словарь терминов и сокращений по технологии Ethernet*”**

Выполнил**:** ст. гр. 10701119

Маканов Д.В

Приняла**:** Белова С.В.

Минск 2021

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

по дисциплине:” Компьютерные системы и сети”

на тему**:” *Сетевой словарь терминов и сокращений по технологии Ethernet* “**

Выполнил**:** ст. гр. 10701119

Маканов Д.В.

Приняла: Белова С.В.

Минск 2021

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 7](#_Toc71612681)

[ГЛАВА I. ОБЗОР СОСТОЯНИЯ ВОПРОСА 8](#_Toc71612682)

[1.1 Протоколы TCP и UDP 8](#_Toc71612683)

[1.2 Сокеты 9](#_Toc71612684)

[1.3 Описание сетевого словаря “Network DICT” 10](#_Toc71612685)

[ГЛАВА II. ПОСТАНОВКА ЗАДАЧИ 11](#_Toc71612686)

[ГЛАВА III. МОДЕЛИРОВАНИЕ И РЕАЛИЗАЦИЯ ПО 12](#_Toc71612687)

[3.1 Архитектура ПО 12](#_Toc71612688)

[3.2 Алгоритмы работы модулей 13](#_Toc71612689)

[3.3 Интерфейс пользователя 13](#_Toc71612690)

[3.4 Реализация клиентской части 15](#_Toc71612691)

[3.2 Реализация серверной части 17](#_Toc71612692)

[ГЛАВА IV. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 20](#_Toc71612693)

[ЗАКЛЮЧЕНИЕ 23](#_Toc71612694)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 24](#_Toc71612695)

[ГРАФИЧЕСКАЯ ЧАСТЬ 25](#_Toc71612696)

[*ПРИЛОЖЕНИЕ A* 28](#_Toc71612697)

# ВВЕДЕНИЕ

Современный прогресс человечества связан в первую очередь с глобальной информатизацией всего мирового сообщества. В современном мире компьютерные сети получили широкое распространение в информационных системах кредитно-финансовой сферы, органов государственного управления и местного самоуправления, предприятий и организаций. Они определяют вектор развития информационных технологий и систем. Именно поэтому сегодня большая часть информационных систем представляет собой Web-ресурс, доступ к которому осуществляется посредством глобальной сети Internet.

Так же компьютерные сети не обошли стороной и игровую индустрию, т.к. они позволили пользователям взаимодействовать друг с другом оставаясь в разных местах. Это привело к тотальному заполнению игровой индустрии сетевыми приложениями.

В данном курсовом проекте будут реализовано сетевое приложение, позволяющее пользователям редактировать сетевой словарь в реальном времени.

Для реализации этой задачи будет использоваться язык программирования высоко уровня Python, ориентированный на повышение производительности разработчика и читаемости кода, PyQt5, набор модулей (библиотек) языка программирования Python, предназначенный для написания компьютерных игр и мультимедиа-приложений, и технология сокетов операционной системы Windows для организации сетевого взаимодействия.

# ГЛАВА I. ОБЗОР СОСТОЯНИЯ ВОПРОСА

## 1.1 Протоколы TCP и UDP

Протокол TCP (Transmission Control Protocol) является протоколом транспортного уровня стека TCP/IP. Он обеспечивает сквозную доставку данных между прикладными процессами, запущенными на узлах, взаимодействующих по сети. Стандартное описание TCP содержится в RFC-793.

Протокол TCP используется в тех случаях, когда требуется надежная доставка сообщений. Он освобождает прикладные процессы от необходимости использовать таймауты и повторные передачи для обеспечения надежности. Наиболее типичными прикладными процессами, использующими TCP, являются FTP (File Transfer Protocol - протокол передачи файлов) и TELNET.

Прикладные процессы взаимодействуют с модулем TCP через порты. Для отдельных приложений выделяются общеизвестные номера портов.

Протокол TCP требует, чтобы все отправленные данные были подтверждены принявшей их стороной. Он использует таймауты и повторные передачи для обеспечения надежной доставки.

Отправителю разрешается передавать некоторое количество данных, не дожидаясь подтверждения приема ранее отправленных данных. Таким образом, между отправленными и подтвержденными данными существует окно уже отправленных, но еще неподтвержденных данных.

Приемники на обеих сторонах виртуального канала выполняют управление потоком передаваемых данных для того, чтобы не допускать переполнения буферов.

Протокол UDP (User Datagram Protocol) предназначен для обмена дейтаграммами между процессами компьютеров, входящих в единую сеть с коммутацией пакетов. Этот протокол не обеспечивает достоверность доставки пакетов, защиты дублирования данных или надежности от сбоев в передаче.

Взаимодействие между прикладными процессами и модулем UDP осуществляется через UDP-порты. Порты нумеруются начиная с нуля. Прикладной процесс, предоставляющий некоторые услуги другим прикладным процессам (сервер), ожидает поступления сообщений в порт, специально выделенный для этих услуг. Сообщения должны содержать запросы на предоставление услуг. Они отправляются процессами-клиентами.

Когда модуль UDP получает дейтаграмму от модуля IP – протокола сетевого уровня стека TCP/IP, он проверяет контрольную сумму, содержащуюся в ее заголовке. Если контрольная сумма равна нулю, то это означает, что отправитель дейтаграммы ее не подсчитывал, и, следовательно, ее нужно игнорировать.

Если контрольная сумма правильная (или равна нулю), то проверяется порт назначения, указанный в заголовке дейтаграммы. Если к этому порту подключен прикладной процесс, то прикладное сообщение, содержащееся в дейтаграмме, становится в очередь для прочтения. В остальных случаях дейтаграмма отбрасывается.

Протокол UDP намного проще, чем TCP и полезен в ситуациях, когда мощные механизмы обеспечения надежности протокола TCP не требуются или будут только помехой для решения определенного рода задач, например, аутентификации пользователей.

## 1.2 Сокеты

Сетевые стеки протоколов представляют возможность программистам напрямую обращаться к службам нижележащих уровней. С помощью специальных API приложения могут реализовывать собственные протоколы взаимодействия.

Сокет – абстрактное программное понятие, используемое для обозначения в прикладной программе конечной точки сетевого взаимодействия. Технология (интерфейс) сокетов – программный интерфейс (API) для обеспечения обмена данными между процессами. Интерфейс сокетов расположен на транспортном уровне стека TCP/IP, но в некоторых случаях может взаимодействовать напрямую с сетевым уровнем в обход транспортному.

Существуют три основных типа сокетов: потоковые, дейтаграммые и сырые. Потоковые сокеты – это сокеты с установлением соединения, состоящие из потока байтов, который может быть двунаправленным. Т.е. через такую конечную точку приложение может и передавать, и получать данные. Потоковый сокет гарантирует обнаружение и исправление ошибок, обрабатывает доставку и сохраняет последовательность данных. Он подходит для передачи больших объемов данных, поскольку в этом случае накладные расходы, связанные с установлением соединения, незначительны по сравнению со временем передачи самого сообщения. Качество передачи достигается за счет использования протокола TCP.

Дейтаграммные сокеты – это сокеты без установления соединения, не обеспечивающие надежность при передаче. Применяются для приложений, когда неприемлемы затраты времени, связанные с установлением явного соединения. Для передачи данных используется протокол UDP.

Сырые сокеты (raw sockets - необрабатываемые, простые) – это сокеты, которые взаимодействуют с протоколами сетевого уровня в обход протоколов транспортного уровня. Используются для непосредственного доступа приложения к IP-пакетам сетевого уровня. Использование сырых сокетов возможно при разработке низкоуровневого системного ПО. Например, сырые сокеты используют различные программы-анализаторы пакетов, сниферы, утилиты TCP/IP ping, tracert и т д.

Семейство адресов – важнейший параметр сокета. Он указывает используемый в настоящее время сетевой протокол и ограничивает применение других параметров сокета.

Адрес сокета при использовании протоколов TCP/IP – это IP-адрес и номер порта прикладной службы. IP-адрес уникален в Internet. Номер порта уникален на отдельном компьютере. Следовательно, адрес сокета будет уникален в Internet. Это позволяет удаленным процессам общаться исключительно на основе адреса сокета.

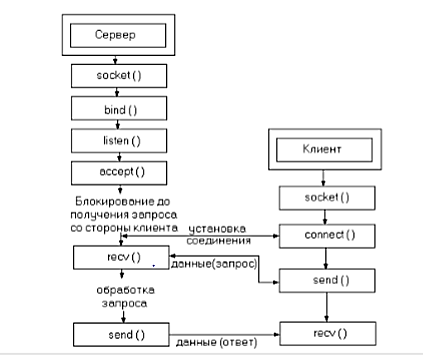
Приложение, использующее сокеты, состоит из распределенной программы, исполняемой на обоих концах канала связи. Программу, инициирующую передачу, называют клиентом. Сервер представляет собой модуль, пассивно ожидающий входящих запросов на установку соединений от удаленных клиентов. Как правило, серверное приложение загружается при запуске системы и прослушивает свой порт, ожидая входящих соединений. Клиентские приложения пытаются установить соединение с сервером, после чего начинается обмен данными. По завершении сеанса связи клиент, как правило, разрывает соединение. На рисунке представлена базовая модель взаимодействия для потоковых сокетов.

Рисунок 1.2.1- Блок-схема взаимодействия потоковых сокетов

## 1.3 Описание сетевого словаря “Network DICT”

Данная программа работает по принципу клиент-сервер. Клиент организует работу со словарем: удаляет или добавляет слова, ищет необходимые слова, редактирует термины. При выполнении всех действий пользователь автоматически посылает запросы серверу, который изменяет список слов в соответствии с требованием пользователя. После выполнения всех операций сервер передает словарь или найденное слово клиенту и тот уже обрабатывает полученную информацию.

# ГЛАВА II. ПОСТАНОВКА ЗАДАЧИ

В рамках данного курсового проекта необходимо реализовать сетевой словарь для взаимодействия клиентов в реальном времени.

Разрабатываемая игра должно иметь архитектуру клиент-сервер, графический интерфейс для пользователя, а также многопользовательский режим.

Клиентская часть должна иметь графический интерфейс. Серверная часть – консольный интерфейс.

Требования к реализации работы:

- операционная система – MS Windows 10;

- среда выполнения – JetBrains PyCharm;

- язык программирования – Python;

- сетевой словарь «Network DICT»;

- графический интерфейс;

- многопользовательский режим.

Взаимодействие клиента и сервера будет осуществляться с помощью технологии сокетов.

# ГЛАВА III. МОДЕЛИРОВАНИЕ И РЕАЛИЗАЦИЯ ПО

## 3.1 Архитектура ПО

Архитектура программного обеспечения – это структура программы или вычислительной системы, которая включает программные компоненты, видимые снаружи свойства этих компонентов, а также отношения между ними.

Программное обеспечение с хорошо продуманной архитектурой будет выполнять поставленные задачи, одновременно обеспечивая максимально высокую производительность, безопасность, надежность и многие другие факторы.

Сетевое приложение представляет собой распределенную программу, т. е. программу, которая состоит из нескольких взаимодействующих частей, каждая из которых выполняется на отдельном компьютере сети (см. рис. 3.1.1).



Рисунок 3.1.1- Взаимодействие частей распределённого приложения

Распределенная программа использует технологию клиент-сервер.

Сетевой словарь, разработанный в ходе данного текущего курсового проекта, является многопользовательским приложением на базе технологии клиент-сервер, реализующим многопоточную модель взаимодействия.

В многопоточном сервере приложение работает в основном потоке, а для каждого подключившегося клиента создается новый поток. Поток существует все время взаимодействия с клиентом. При отключении клиента поток уничтожается.

В случае небольшого сервера и нескольких клиентов такой подход работает хорошо, к тому же его легко реализовать. К сожалению, многопоточный сервер плохо масштабируется. Главный его недостаток — большое число создаваемых и уничтожаемых потоков. Такой сервер может принять около 1000 соединений, после чего начинают генерироваться исключения, извещающие о нехватке памяти (память быстро заканчивается потому, что каждый поток имеет собственный стек, объем которого по умолчанию равен 1 МБ).

Избежать создания отдельного потока для каждого соединения можно создав заранее некоторое количество потоков - пул. При подключении клиента берется поток из пула потоков. В нем происходит взаимодействие с данным клиентом. Задачи выполняются параллельно. Если потоки не имеют общих данных, то не будет накладных расходов на синхронизацию, что делает работу достаточно быстрой. После завершения работы поток не убивается, а лежит в пуле, ожидая следующей задачи. Это убирает накладные расходы на создание и удаление потоков. Этот подход вполне приемлем, если соединения будут недолговечными. Однако соединения с серверами (например, с сервером-чатом) обычно остаются открытыми длительное время, поэтому ограничивать число соединений числом потоков в пуле зачастую неприемлемо.

## 3.2 Алгоритмы работы модулей

Курсовая работа состоит из следующих модулей:

- Server;

- Client.

На рисунке 3.2.1 показано взаимодействие главных модулей.



Рисунок 3.1.1- Взаимодействие главных модулей

Сетевой словарь реализует серверное приложение, которое будет обслуживать клиентов, которые подключаются к этому приложению по многопоточной модели, то есть для нового клиента новый поток.

При подключении клиента к серверу начинается обмен данными между ними на основе технологии сокетов.

Подключившийся клиент сразу отправляет запрос серверу на отображения всех слов в словаре. Каждый клиент нажимает соответствующие кнопки в графическом интерфейсе, результаты нажатия отправляются на сервер. На сервере обрабатываются принятые от каждого клиента команды и, как результат, отправляется клиентам результаты их действий. Каждый клиент принимает эти данные, которые отображаются соответствующим графическим интерфейсом.

Сам словарь представляет собой текстовый файл, в котором каждая новая строка – это новый термин, термин и определение отделены специальным разделяющим символом. Для работы с файлом был написан специальный утилитарный модуль FileDictWorker.py.

Взаимодействие клиента и сервера осуществляется при помощи посылки определенных команд. Команда INSERT\_MODE позволяет понять серверу, что необходимо добавить слово в словарь. Команда FIND\_MODE позволяет понять серверу, что нужно найти термин и отправить его клиенту. Команда REMOVE\_MODE позволяет удалить термин из словаря. Команда SORT\_MODE позволяет отсортировать сервер. Команда EDIT\_MODE позволяет отредактировать заданное слово. Команда ALL\_MODE позволяет отправить весь список слов в словаре клиенту. Команда EXIT\_MODE завершает взаимодействие клиента и сервера.

## 3.3 Интерфейс пользователя

Интерфейс пользователя разрабатывался в среде IDLE PyCharm Community Edition с использованием библиотеки Pygame. Это интегрированная среда разработки для языка программирования Python, которая была написана на Java и Python. Предоставляет средства для анализа кода, графический отладчик, инструмент для запуска юнит-тестов и поддерживает веб-разработку на Django.

При открытии словаря появляется приветственное окно, дающее пользователю подключиться к серверу по DNS-имени.

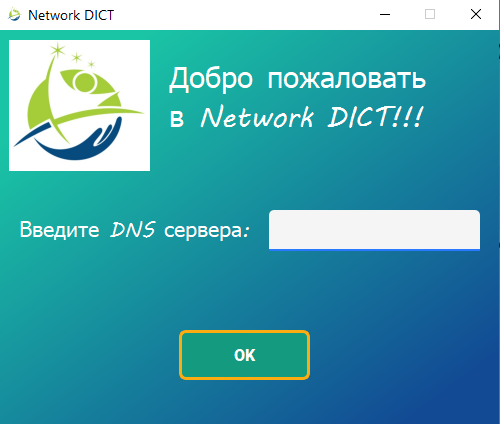


Рисунок 3.3.1- Приветственное окно

При нажатии на кнопку OK при подключении к серверу пользователя встречает окно, изображенное на рисунке 3.3.2. Предварительно в словарь были загружены некоторые термины. Кнопки изменить и Х служат для изменения и удаления нужного слова.

Рисунок 3.3.2- Окно “Все слова”

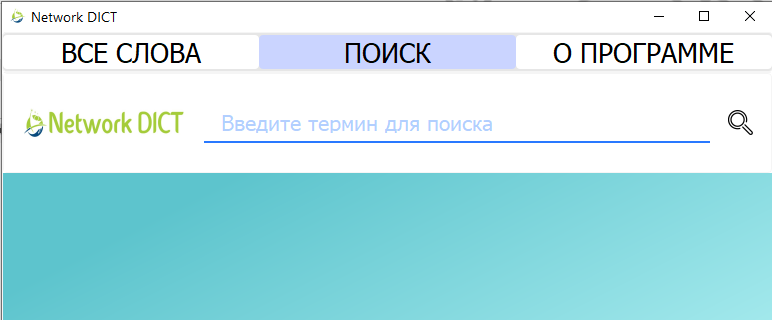
На вкладке поиска пользователь может найти слово или термин в словаре (см. рисунок 3.3.3). В случае отсутствия слова пользователю выскочит сообщение об отсутствии термина.

Рисунок 3.3.3 - Окно “Поиск слова”

## 3.4 Реализация клиентской части

Класс Client является основным классом, обеспечивающим взаимодействие с сервером. Ниже приведён код модуля client\_logic.py с соответствующим классом Client.

SERVER\_ADDRESS = **"127.0.0.1"** *# в курсовой обращение по доменному имени*PORT = 11000  
BYTES\_PER\_PACKAGE = 4000  
  
  
class Client:  
 def \_\_init\_\_(self):  
 self.client = socket.socket(  
 socket.AF\_INET,  
 socket.SOCK\_STREAM  
 )  
 self.app = ClientFormWorker()  
 self.server\_dns\_name = **""** *# HP-PYMAK* self.welcome\_win = WelcomeWin()  
 self.welcome\_win.show()  
 self.welcome\_win.pushButton.clicked.connect(self.welcome\_commands)  
 self.welcome\_win.lineEdit.returnPressed.connect(self.welcome\_commands)  
  
 self.app.searchBtn.clicked.connect(self.find\_mode\_actions)  
 self.app.lineEdit.returnPressed.connect(self.find\_mode\_actions)  
  
 self.app.sortBtn.clicked.connect(self.sort\_mode\_actions)  
  
 self.app.addWordBtn.clicked.connect(self.insert\_mode\_actions)  
  
 def welcome\_commands(self):  
 try:  
 self.server\_dns\_name = self.welcome\_win.lineEdit.text()  
 self.connect()  
 self.welcome\_win.close()  
 self.app.show()  
 self.all\_mode\_actions()  
 except socket.gaierror:  
 self.welcome\_win.warningMsg.show()  
 self.welcome\_win.lineEdit.clear()  
*# сетевые функции* def connect(self):  
 self.client.connect(  
 (self.server\_dns\_name, PORT)  
 )  
  
 def send(self, msg):  
 self.client.send(msg.encode(**"utf-8"**))  
  
 def receive(self):  
 return self.client.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**)  
  
*# Функции для отправки запросов и обработки ответов в главном окне* def all\_mode\_actions(self):  
 self.send(ALL\_MODE)  
 info = self.receive()  
 dictionary = FileDictWorker.convert\_dict\_str\_real\_dict(info)  
 self.app.clear\_cards(self.app.allWordsLay)  
 self.app.add\_cards(self.app.allWordsLay, dictionary, self)  
  
 def find\_mode\_actions(self):  
 if not self.app.lineEdit.text():  
 return  
 self.send(FIND\_MODE)  
 self.send(self.app.lineEdit.text())  
 self.app.clear\_cards(self.app.searchWLay)  
 answers = self.receive()  
 if answers != EMPTY\_RESULT:  
 self.app.add\_cards(self.app.searchWLay,  
 FileDictWorker.convert\_dict\_str\_real\_dict(answers),  
 self)  
 else:  
 self.app.add\_cards(self.app.searchWLay, [Word(  
 self.app.lineEdit.text() + **" не найдено."**, **""** )], self)  
  
 for i in range(self.app.searchWLay.count()):  
 self.app.searchWLay.itemAt(i).widget().changeBtn.hide()  
 self.app.searchWLay.itemAt(i).widget().deleteBtn.hide()  
  
 def sort\_mode\_actions(self):  
 self.send(SORT\_MODE)  
 dictionary = FileDictWorker.convert\_dict\_str\_real\_dict(self.receive())  
 self.app.clear\_cards(self.app.allWordsLay)  
 self.app.add\_cards(self.app.allWordsLay, dictionary, self)  
  
 def insert\_mode\_actions(self):  
 adder = WordWidget(self)  
 adder.changeBtn.hide()  
 adder.deleteBtn.hide()  
 adder.setWindowTitle(**"Добавление слова"**)  
 adder.lineEdit.setReadOnly(False)  
 adder.textEdit.setReadOnly(False)  
 adder.show()  
 adder.lineEdit.returnPressed.connect(  
 lambda: self.add\_word(adder))  
  
 def add\_word(self, adder):  
 self.send(INSERT\_MODE)  
 msg = adder.lineEdit.text() + **"^"** + adder.textEdit.toPlainText()  
 adder.close()  
 self.send(msg)  
 info = self.receive()  
 dictionary = FileDictWorker.convert\_dict\_str\_real\_dict(info)  
 self.app.clear\_cards(self.app.allWordsLay)  
 self.app.add\_cards(self.app.allWordsLay, dictionary, self)  
  
   
 def edit\_word(self, wordCard, adder):  
 self.send(wordCard.lineEdit.text() +  
 **"^"** + wordCard.textEdit.toPlainText())  
 wordCard.lineEdit.setText(adder.lineEdit.text())  
 wordCard.textEdit.setText(adder.textEdit.toPlainText())  
  
 msg = adder.lineEdit.text() + **"^"** + adder.textEdit.toPlainText()  
 adder.close()  
 self.send(msg)  
  
 def remove\_word(self, wordCard):  
 self.send(REMOVE\_MODE)  
 self.send(wordCard.lineEdit.text() +  
 **"^"** + wordCard.textEdit.toPlainText())  
  
 wordCard.hide()

## 3.2 Реализация серверной части

Класс Server является реализации серверной части данной сетевой игры. Он вызывает методы функционального класса DictionaryWorker, обеспечивающий работу со словарем. Приём от клиентов данных и отправку соответствующих ответов осуществляет класс Server. Ниже приведён код модуля server.py с соответствующим классом Server и модуля DictionaryWorker.py.

THREADS\_AMOUNT = 5  
  
DEFAULT\_GATEWAY = **"0.0.0.0"**PORT = 11000  
LENGTH\_QUEUE = 10  
BYTES\_PER\_PACKAGE = 4000  
  
  
def serve\_client(server):  
 while True:  
 user\_socket, address = server.accept()  
  
 print(**'**\n**Ip address and port of the client are '** + str(address) +  
 **"**\n**Client descriptor: "** + str(user\_socket.fileno()) +  
 **"**\n**Time of connection: "** + str(Time.get\_data\_and\_time()) +  
 **"**\n**Thread: "** + threading.current\_thread().name)  
  
 *# получение сообщения клиенту в виде пакета в байтах* mode = user\_socket.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**)  
  
 while mode != EXIT\_MODE:  
 dictionary = FileDictWorker.get\_dictionary\_from\_file()  
 if mode == INSERT\_MODE:  
 word = user\_socket.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**)  
 word = FileDictWorker.parse\_str\_for\_word\_parts(word)  
 DictionaryWorker.insert\_word(Word(word[0], word[1]), dictionary)  
 info = FileDictWorker.convert\_dictionary\_to\_str(dictionary)  
 user\_socket.send(info.encode(**"utf-8"**))  
  
  
 elif mode == FIND\_MODE:  
 word\_name = user\_socket.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**)  
  
 results = DictionaryWorker.find\_word\_in\_dict(word\_name, dictionary)  
 if results:  
 info = FileDictWorker.convert\_dictionary\_to\_str(results)  
 else:  
 info = EMPTY\_RESULT  
 user\_socket.send(info.encode(**"utf-8"**))  
  
 elif mode == REMOVE\_MODE:  
 word\_parts = FileDictWorker.parse\_str\_for\_word\_parts(  
 user\_socket.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**))  
 word = Word(word\_parts[0], word\_parts[1])  
  
 DictionaryWorker.remove\_word(word, dictionary)  
  
 elif mode == EDIT\_MODE:  
 word\_parts = FileDictWorker.parse\_str\_for\_word\_parts(  
 user\_socket.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**))  
 edit\_word\_parts = FileDictWorker.parse\_str\_for\_word\_parts(  
 user\_socket.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**))  
  
 DictionaryWorker.edit\_word(word\_parts, edit\_word\_parts, dictionary)  
  
 elif mode == SORT\_MODE:  
 DictionaryWorker.sort\_dictionary(dictionary)  
 info = FileDictWorker.convert\_dictionary\_to\_str(dictionary)  
 user\_socket.send(info.encode(**"utf-8"**))  
 elif mode == ALL\_MODE:  
 info = FileDictWorker.convert\_dictionary\_to\_str(dictionary)  
 user\_socket.send(info.encode(**"utf-8"**))  
  
 FileDictWorker.put\_dictionary\_to\_file(dictionary)  
  
 mode = user\_socket.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**)  
  
 print(**f"**\n**Client** {str(user\_socket.fileno())} **is out!!!"**)  
  
 user\_socket.shutdown(socket.SHUT\_RDWR)  
 user\_socket.close()

Код модуля DictionaryWorker.py.

*#!/usr/bin/python3  
# -\*- coding: utf-8 -\*-*from util.word\_class import Word  
  
  
class DictionaryWorker:  
 @classmethod  
 def add\_word(cls, word, dictionary):  
 if cls.\_\_check\_parameters(word, dictionary):  
 dictionary.append(word)  
  
 @classmethod  
 def edit\_word(cls, word, edit\_word, dictionary):  
 for i in range(len(dictionary)):  
 item\_key = dictionary[i].get\_key()  
 item\_val = dictionary[i].get\_value()  
 if item\_key == word[0] and word[1] in item\_val:  
 dictionary[i].set\_key(edit\_word[0])  
 dictionary[i].set\_value(edit\_word[1])  
 break  
  
 @classmethod  
 def insert\_word(cls, word, dictionary):  
 cls.add\_word(word, dictionary)  
 cls.sort\_dictionary(dictionary)  
  
 @classmethod  
 def find\_word\_in\_dict(cls, word\_name, dictionary):  
 results = []  
 for item in dictionary:  
 if item.get\_key().lower() == word\_name.lower():  
 results.append(item)  
  
 return results  
 @classmethod  
 def remove\_word(cls, word, dictionary):  
 for item in dictionary:  
 if item.get\_key() == word.get\_key() and item.get\_value() == item.get\_value():  
 dictionary.remove(item)  
 break  
 @staticmethod  
 def sort\_dictionary(dictionary):  
 for i in range(1, len(dictionary)):  
 fixed\_item = dictionary[i]  
 j = i - 1  
 while j >= 0 and fixed\_item.get\_key().lower() < dictionary[j].get\_key().lower():  
 dictionary[j + 1] = dictionary[j]  
 j -= 1  
 dictionary[j + 1] = fixed\_item  
  
 @staticmethod  
 def \_\_check\_parameters(word, dictionary):  
 return (True if isinstance(word, Word) and isinstance(dictionary, list)  
 else False)

# ГЛАВА IV. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска словаря надо сначала запустить сервер (см. рис. 4.1).

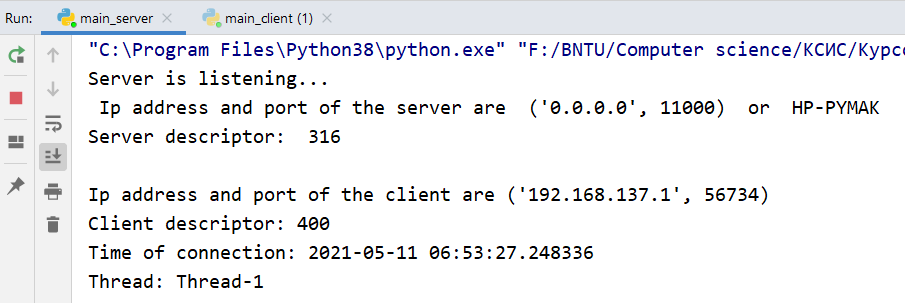


Рисунок 4.1 - Окно сервера программы

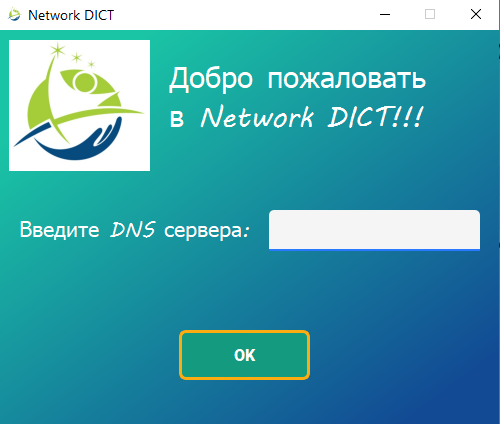
При запуске программы появляется окно приветствия и предлагается ввести DNS сервера.

Рисунок 4.2 – Окно приветствия

В случае отсутствия подключения будет выведено сообщение “Сервер не найден”. Далее пользователя встречает главное окно приложения (см. рис. 4.3), которое имеет три вкладки: все слова, поиск и о программе. Кнопки изменить и Х служат для изменения и удаления нужного слова. Кнопка “Обновить список” служит для обновления списка и его сортировки, в случае если с другого клиента словарь был изменен. Также кнопка для добавления слова служит для добавления нового термина через специальное окно, в котором прописывается термин и пояснение к нему.



Рисунок 4.3 – Главное окно

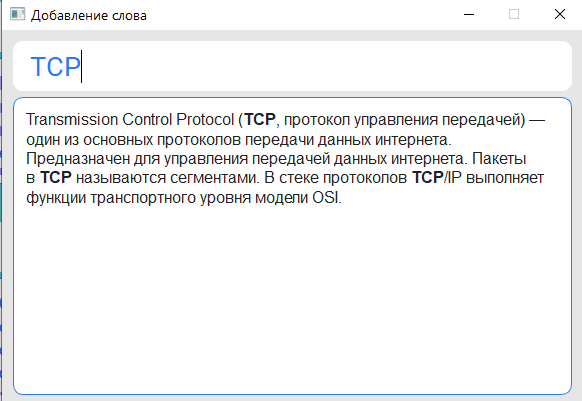


Рисунок 4.4 – Окно добавления слова

Во вкладке поиска можно найти интересующий термин и пояснение к нему (см. рис. 4.5), в случае отсутствия термина пользователю будет выведено соответствующее сообщение. Поиск можно начать либо по нажатию кнопки со значком лупы или по нажатию клавиши Enter.

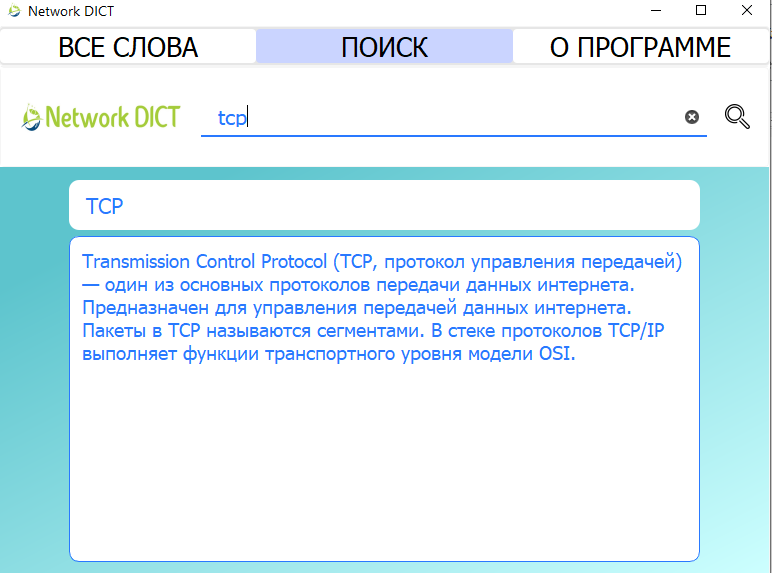


Рисунок 4.5 – Окно поиска термина

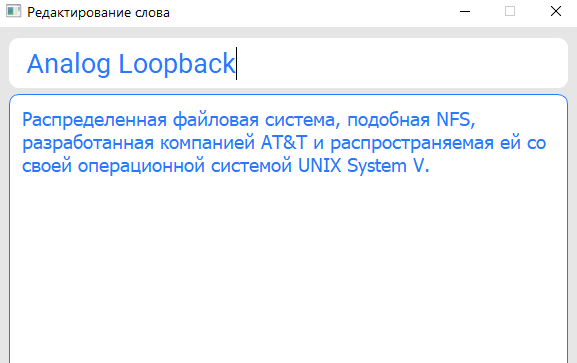
Отредактировать слово можно по нажатию кнопки “Изменить” возле редактируемого слова. В этом случае появиться окно для редактирования (см. рис. 4.6).

Рисунок 4.6 – Окно редактирования

В случае удаления слова при нажатии кнопки “X” карточка с этим словом будет удалена.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данного курсового проекта были закреплены знания: принцип работы клиент-серверного приложения, средства python для разработки клиент-серверного многопоточного приложения на базе сокетов.

Результатом выполнения курсового проекта стал сетевой словарь, позволяющая играть нескольким клиентам обновлять словарь терминов в режиме реального времени.

Серверная часть обладает возможностями получения, отправки и обработки данных.

Клиентская часть может подключатся к серверной, отправлять запросы и получать ответы.

При разработке и тестировании программы были устранены явные ошибки и осуществлена обработка исключительных ситуаций, приводивших к неработоспособности игры.

Данное приложение обладает интуитивно понятным, простым и в то же время эффективным интерфейсом, позволяющий быстро отредактировать словарь терминов по технологии Ethernet.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 2-е изд. / В.Г.Олифер, Н.А.Олифер. — СПб: «Питер», 2003. — 864с.
2. Троелсен Э.У. C# и платформа .NET. Библиотека программиста. / Э.У. Троелсен. – СПб.: Питер, 2004. – 796 с.: ил.
3. Доусон М. Программируем на Python. – СПб.: Питер, 2014. – 416 с.
4. Лутц М. Программирование на Python, том I, 4-е издание. – Пер. с англ. – СПб.: Сим¬вол-Плюс, 2011. – 992 с.
5. Лутц М. Программирование на Python, том II, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 993 с.

# ГРАФИЧЕСКАЯ ЧАСТЬ

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |
|  |  |  |  |  | КП—10701119–ДО–2021 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Маканов |  |  | *Архитектура ПО* | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | Д |  | *1* | *5* |
| Консульт. | | Белова |  |  | 1-40 01 01 БНТУ  г. Минск | | | | |
| Н.контр. | | Белова |  |  |
| Зав.каф. | | Полозков |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
|  | Фамилия | Подпись | Дата | *Алгоритм работы модулей* | Лист | Листов |
| Студент | Маканов |  |  |
| Руководитель | Белова |  |  | *2* | *5* |
|  | | | | | | |
|  | Фамилия | Подпись | Дата | *Интерфейс пользователя* | Лист | Листов |
| Студент | Маканов |  |  |
| Руководитель | Белова |  |  | *3* | *5* |
|  | | | | | | |
|  | Фамилия | Подпись | Дата | *Диаграмма классов клиентской и серверной частей* | Лист | Листов |
| Студент | Маканов |  |  |
| Руководитель | Белова |  |  | *4* | *5* |
|  | | | | | | |
|  | Фамилия | Подпись | Дата | *Формат передаваемых сообщений* | Лист | Листов |
| Студент | Маканов |  |  |
| Руководитель | Белова |  |  | *5* | *5* |

# *ПРИЛОЖЕНИЕ A*

**Листинг программы**

**Файл server.py**

*# -\*- coding: utf-8 -\*-*import socket  
import threading  
  
from util.word\_class import Word  
from util.FileDictWorker import FileDictWorker  
from util.TimeTask import Time  
  
from util.modes import \*  
from server.DictionaryWorker import DictionaryWorker  
  
THREADS\_AMOUNT = 5  
  
DEFAULT\_GATEWAY = **"0.0.0.0"**PORT = 11000  
LENGTH\_QUEUE = 10  
BYTES\_PER\_PACKAGE = 4000  
  
  
def serve\_client(server):  
 while True:  
 user\_socket, address = server.accept()  
  
 print(**'**\n**Ip address and port of the client are '** + str(address) +  
 **"**\n**Client descriptor: "** + str(user\_socket.fileno()) +  
 **"**\n**Time of connection: "** + str(Time.get\_data\_and\_time()) +  
 **"**\n**Thread: "** + threading.current\_thread().name)  
  
 *# получение сообщения клиенту в виде пакета в байтах* mode = user\_socket.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**)  
  
 while mode != EXIT\_MODE:  
 dictionary = FileDictWorker.get\_dictionary\_from\_file()  
 if mode == INSERT\_MODE:  
 word = user\_socket.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**)  
 word = FileDictWorker.parse\_str\_for\_word\_parts(word)  
 DictionaryWorker.insert\_word(Word(word[0], word[1]), dictionary)  
 info = FileDictWorker.convert\_dictionary\_to\_str(dictionary)  
 user\_socket.send(info.encode(**"utf-8"**))  
  
  
 elif mode == FIND\_MODE:  
 word\_name = user\_socket.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**)  
  
 results = DictionaryWorker.find\_word\_in\_dict(word\_name, dictionary)  
 if results:  
 info = FileDictWorker.convert\_dictionary\_to\_str(results)  
 else:  
 info = EMPTY\_RESULT  
 user\_socket.send(info.encode(**"utf-8"**))  
  
 elif mode == REMOVE\_MODE:  
 word\_parts = FileDictWorker.parse\_str\_for\_word\_parts(  
 user\_socket.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**))  
 word = Word(word\_parts[0], word\_parts[1])  
  
 DictionaryWorker.remove\_word(word, dictionary)  
  
 elif mode == EDIT\_MODE:  
 word\_parts = FileDictWorker.parse\_str\_for\_word\_parts(  
 user\_socket.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**))  
 edit\_word\_parts = FileDictWorker.parse\_str\_for\_word\_parts(  
 user\_socket.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**))  
  
 DictionaryWorker.edit\_word(word\_parts, edit\_word\_parts, dictionary)  
  
 elif mode == SORT\_MODE:  
 DictionaryWorker.sort\_dictionary(dictionary)  
 info = FileDictWorker.convert\_dictionary\_to\_str(dictionary)  
 user\_socket.send(info.encode(**"utf-8"**))  
 elif mode == ALL\_MODE:  
 info = FileDictWorker.convert\_dictionary\_to\_str(dictionary)  
 user\_socket.send(info.encode(**"utf-8"**))  
  
 FileDictWorker.put\_dictionary\_to\_file(dictionary)  
  
 mode = user\_socket.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**)  
  
 print(**f"**\n**Client** {str(user\_socket.fileno())} **is out!!!"**)  
  
 user\_socket.shutdown(socket.SHUT\_RDWR)  
 user\_socket.close()

**Файл Clinet.py**

*# -\*- coding: utf-8 -\*-*import socket  
from client.view.action\_wins.ClientFormWorker import ClientFormWorker  
from util.FileDictWorker import FileDictWorker  
from util.word\_class import Word  
from client.view.action\_wins.welcomeWin import WelcomeWin  
from client.view.action\_wins.card\_word\_WGT import WordWidget  
  
from util.modes import \*  
  
SERVER\_ADDRESS = **"127.0.0.1"** *# в курсовой обращение по доменному имени*PORT = 11000  
BYTES\_PER\_PACKAGE = 4000  
  
  
class Client:  
 def \_\_init\_\_(self):  
 self.client = socket.socket(  
 socket.AF\_INET,  
 socket.SOCK\_STREAM  
 )  
 self.app = ClientFormWorker()  
 self.server\_dns\_name = **""** *# HP-PYMAK* self.welcome\_win = WelcomeWin()  
 self.welcome\_win.show()  
 self.welcome\_win.pushButton.clicked.connect(self.welcome\_commands)  
 self.welcome\_win.lineEdit.returnPressed.connect(self.welcome\_commands)  
  
 self.app.searchBtn.clicked.connect(self.find\_mode\_actions)  
 self.app.lineEdit.returnPressed.connect(self.find\_mode\_actions)  
  
 self.app.sortBtn.clicked.connect(self.sort\_mode\_actions)  
  
 self.app.addWordBtn.clicked.connect(self.insert\_mode\_actions)  
  
 def welcome\_commands(self):  
 try:  
 self.server\_dns\_name = self.welcome\_win.lineEdit.text()  
 self.connect()  
 self.welcome\_win.close()  
 self.app.show()  
 self.all\_mode\_actions()  
 except socket.gaierror:  
 self.welcome\_win.warningMsg.show()  
 self.welcome\_win.lineEdit.clear()  
*# сетевые функции* def connect(self):  
 self.client.connect(  
 (self.server\_dns\_name, PORT))  
   
 def send(self, msg):  
 self.client.send(msg.encode(**"utf-8"**))  
  
 def receive(self):  
 return self.client.recv(BYTES\_PER\_PACKAGE).decode(**"utf-8"**)  
  
*# Функции для отправки запросов и обработки ответов в главном окне* def all\_mode\_actions(self):  
 self.send(ALL\_MODE)  
 info = self.receive()  
 dictionary = FileDictWorker.convert\_dict\_str\_real\_dict(info)  
 self.app.clear\_cards(self.app.allWordsLay)  
 self.app.add\_cards(self.app.allWordsLay, dictionary, self)  
  
 def find\_mode\_actions(self):  
 if not self.app.lineEdit.text():  
 return  
 self.send(FIND\_MODE)  
 self.send(self.app.lineEdit.text())  
 self.app.clear\_cards(self.app.searchWLay)  
 answers = self.receive()  
 if answers != EMPTY\_RESULT:  
 self.app.add\_cards(self.app.searchWLay,  
 FileDictWorker.convert\_dict\_str\_real\_dict(answers),  
 self)  
 else:  
 self.app.add\_cards(self.app.searchWLay, [Word(  
 self.app.lineEdit.text() + **" не найдено."**, **""** )], self)  
  
 for i in range(self.app.searchWLay.count()):  
 self.app.searchWLay.itemAt(i).widget().changeBtn.hide()  
 self.app.searchWLay.itemAt(i).widget().deleteBtn.hide()  
  
 def sort\_mode\_actions(self):  
 self.send(SORT\_MODE)  
 dictionary = FileDictWorker.convert\_dict\_str\_real\_dict(self.receive())  
 self.app.clear\_cards(self.app.allWordsLay)  
 self.app.add\_cards(self.app.allWordsLay, dictionary, self)  
  
 def insert\_mode\_actions(self):  
 adder = WordWidget(self)  
 adder.changeBtn.hide()  
 adder.deleteBtn.hide()  
 adder.setWindowTitle(**"Добавление слова"**)  
 adder.lineEdit.setReadOnly(False)  
 adder.textEdit.setReadOnly(False)  
 adder.show()  
 adder.lineEdit.returnPressed.connect(  
 lambda: self.add\_word(adder))  
  
 def add\_word(self, adder):  
 self.send(INSERT\_MODE)  
 msg = adder.lineEdit.text() + **"^"** + adder.textEdit.toPlainText()  
 adder.close()  
 self.send(msg)  
 info = self.receive()  
 dictionary = FileDictWorker.convert\_dict\_str\_real\_dict(info)  
 self.app.clear\_cards(self.app.allWordsLay)  
 self.app.add\_cards(self.app.allWordsLay, dictionary, self)  
  
 def open\_editing\_word(self, wordCard):  
 self.send(EDIT\_MODE)  
 adder = WordWidget(self)  
 adder.changeBtn.hide()  
 adder.deleteBtn.hide()  
 adder.setWindowTitle(**"Редактирование слова"**)  
 adder.show()  
  
 adder.lineEdit.setText(wordCard.lineEdit.text())  
 adder.lineEdit.setReadOnly(False)  
 adder.textEdit.setReadOnly(False)  
 adder.textEdit.setText(wordCard.textEdit.toPlainText())  
 adder.lineEdit.returnPressed.connect(  
 lambda: self.edit\_word(wordCard, adder) )  
 def edit\_word(self, wordCard, adder):  
 self.send(wordCard.lineEdit.text() +  
 **"^"** + wordCard.textEdit.toPlainText())  
 wordCard.lineEdit.setText(adder.lineEdit.text())  
 wordCard.textEdit.setText(adder.textEdit.toPlainText())  
  
 msg = adder.lineEdit.text() + **"^"** + adder.textEdit.toPlainText()  
 adder.close()  
 self.send(msg)  
  
 def remove\_word(self, wordCard):  
 self.send(REMOVE\_MODE)  
 self.send(wordCard.lineEdit.text() +  
 **"^"** + wordCard.textEdit.toPlainText())  
  
 wordCard.hide()

**Файл ClientFormWorker.py**

*# -\*- coding: utf-8 -\*-*from PyQt5.QtWidgets import QWidget  
from client.view.action\_wins.card\_word\_WGT import WordWidget  
  
from client.view.view\_ui.mainUI import Ui\_Form  
class ClientFormWorker(QWidget, Ui\_Form):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
  
 self.allPageBtn.clicked.connect(  
 self.allPageBtn\_actions)  
 self.searchPageBtn.clicked.connect(  
 self.searchPageBtn\_actions)  
 self.aboutPageBtn.clicked.connect(  
 self.aboutPageBtn\_actions  
 )  
 self.lineEdit.setPlaceholderText(**"Введите термин для поиска"**)  
  
 def allPageBtn\_actions(self):  
 self.pagesWGT.setCurrentIndex(0)  
 self.searchPageBtn.setStyleSheet(**"background-color: rgb(255, 255, 255);"**)  
 self.allPageBtn.setStyleSheet(**"background-color: rgb(202, 212, 255);"**)  
 self.aboutPageBtn.setStyleSheet(**"background-color: rgb(255, 255, 255);"**)  
  
 def searchPageBtn\_actions(self):  
 self.pagesWGT.setCurrentIndex(1)  
 self.allPageBtn.setStyleSheet(**"background-color: rgb(255, 255, 255);"**)  
 self.searchPageBtn.setStyleSheet(**"background-color: rgb(202, 212, 255);"**)  
 self.aboutPageBtn.setStyleSheet(**"background-color: rgb(255, 255, 255);"**)  
  
 def aboutPageBtn\_actions(self):  
 self.pagesWGT.setCurrentIndex(2)  
 self.allPageBtn.setStyleSheet(**"background-color: rgb(255, 255, 255);"**)  
 self.searchPageBtn.setStyleSheet(**"background-color: rgb(255, 255, 255);"**)  
 self.aboutPageBtn.setStyleSheet(**"background-color: rgb(202, 212, 255);"**)  
  
 def create\_word\_card(self, word, client):  
 word\_card = WordWidget(client)  
 word\_card.lineEdit.setText(word.get\_key())  
 word\_card.textEdit.setText(word.get\_value())  
 return word\_card  
  
 def add\_cards(self, layout, dictionary, client):  
 for item in dictionary:  
 word\_card = self.create\_word\_card(item, client)  
 layout.addWidget(word\_card)  
 word\_card.show()  
  
 def clear\_cards(self, layout):  
 for i in reversed(range(layout.count())):  
 layout.itemAt(i).widget().setParent(None)