**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

Курсовая работа

По дисциплине «Алгоритмы и структуры данных»

Тема «Информационная система для обмена оповещаниями в корпоративной сети предприятия»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/314-19

Листов 5

**Руководитель разработки**:

доцент каф. ИВК, к.т.н., доцент

*Шишкин Вадим Викторинович*

« » 2023 г.

**Исполнитель**:

студент гр. ИСТбд-21

*Жуклин Никита Дмитриевич*

« » 2023 г.

**2023**

**Содержание**

Аннотация.......................................................................................1

Техническое задание.....................................................................3

Пояснительная записка ................................................................7

Руководство программиста ..........................................................13

Текст программы............................................................................28

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

**на курсовую работу**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема «**Информационная система для обмена оповещаниями в корпоративной сети предприятия»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/314-19 ТЗ-01

Листов 4

**Исполнитель**:

студент гр. ИСТбд-21

*Жуклин Никита Дмитриевич*

« » 2023 г.

**2023**

**Введение**

Информационная система для обмена оповещениями в корпоративной среде предприятия

**Характеристика области применения программы**

Предлагаемая информационная система разработана для облегчения обмена важной информацией и оповещений внутри корпоративной сети предприятия. Её основная задача — обеспечить эффективный и моментальный обмен сообщениями, объявлениями и уведомлениями между сотрудниками в реальном времени.

**Объект использования программы**

Программа предназначена для использования в офисной среде предприятия, где требуется оперативное распространение информации по внутренним деловым процессам. Она может быть применена в любой отрасли и предприятии, где важна оперативная коммуникация и обмен внутрикорпоративными сообщениями между сотрудниками.

Эта информационная система предназначена для повышения эффективности внутренней коммуникации, облегчения координации действий между сотрудниками, и обеспечения оперативного информационного обмена в корпоративной сети.

Начало формы

**1. Основания для разработки**

Учебный план направления 09.03.02 «Информационные системы и технологии» и распоряжение по факультету.

**2. Требования к программе или программному изделию**

**2.1. Функциональное назначение**

Требуется разработать многопользовательское десктопное приложение по обмену оповещениями в сети предприятия с графическим интерфейсом в среде Windows.

Пользователи: сотрудники подключенные к сети предприятия

**2.2 Требования к функциональным характеристикам**

2.2.1 Требования к структуре приложения

Приложение должно быть разработано в виде нескольких модулей, взаимодействующих между собой с использованием дополнительных информационных файлов.

2.2.2 Требования к составу функций приложения

В приложении должны быть реализованы следующие основные функции:

- регистрация/авторизация пользователя;

- отрисовка чата пользователей;

- уведомления для пользователей;

- отправка и получение новых сообщений;

- обработка истории сообщений пользователей;

2.2.2 Требования к организации информационного обеспечения, входных и выходных данных

В приложении должен быть реализован графический интерфейс взаимодействия с пользователем. Отдельно должны быть выделены папки под графические файлы, базы данных, аудио эффектов, а также для самого кода. Логин и пароль пользователя должны вводиться с клавиатуры. Логины и пароли зарегистрированных пользователей должны храниться в базе данных на сервере.

**2.3 Требования к надёжности**

Поддержка непрерывной и стабильной работы сервера и компьютера.

**2.4 Требования к информационной и программной совместимости**

Рекомендуется к использованию на Windows 7 или 10.

При создании программы используются встроенные библиотеки “tkinter”, “json”, “socket“, “threading“, “sqlite3“,

И сторонние библиотеки “notifypy 0.3.42“

Разработка ведётся в “PyCharm community edition 2022.3” на версии языка программирования Python 3.10.

**2.5.Требования к маркировке и упаковке**

Определяются заданием на курсовую работу.

**2.6 Требования к транспортированию и хранению**

2.6.1 Условия транспортирования

Требования к условиям транспортирования не предъявляются

2.6 2 Условия хранения

Обеспечение свободного доступа к проекту в репозитории до окончания срока учебы

2.6 3 Сроки хранения

«Срок хранения – до окончания срока учебы

**3. Требования к программной документации**

Определяются заданием на курсовую работу

**4. Стадии и этапы разработки**

Определяются заданием на курсовую работу.

**5. Порядок контроля и приёмки**

Определяются заданием на курсовую работу.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**Курсовая работа**

**По дисциплине «Алгоритмы и структуры данных»**

**Тема «**Информационная система для обмена оповещаниями в корпоративной сети предприятия»

**Пояснительная записка**

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/314-19 ПЗ-01

Листов 6

**Руководитель разработки**:

доцент каф. ИВК, к.т.н., доцент

*Шишкин Вадим Викторинович*

« » 2023 г.

**Исполнитель**:

студент гр. ИСТбд-21

*Жуклин Никита Дмитриевич*

« » 2023 г.

**2023**

**Введение**

Приложение «Мессенджер» реализует функционал чата для пользователей предприятия. При планировании разработки была выбрана каскадная модель. Ее преимущества заключаются в том, что она проста и понятна, хорошая документированность, каждый этап имеет четкие цели и тд.

Приложение «Мессенджер» представляет собой серверную и клиентскую часть для чата внутри сети предприятия.

**1. Проектная часть**

**1.1 Постановка задачи на разработку приложения**

Определяется заданием на курсовую работу

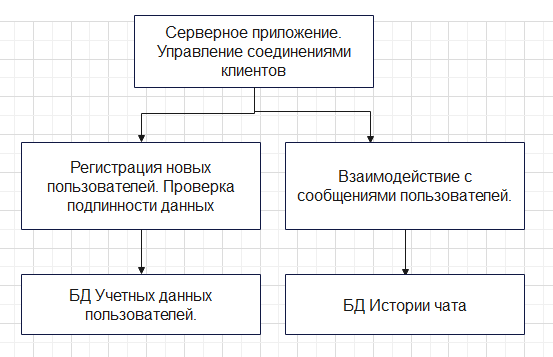
**1.2 Математические методы**

В качестве математической модели для представления и отрисовки поля был выбран двумерный массив, он позволяет легко записать положение всех шашек, выполнять все необходимые проверки и отрисовывать шашки, а также может быть легко изменен, что упрощает вывод хода пользователя. В массиве пустые клетки представлены числом 0, белые шашки 1, белые дамки 2, черные шашки 1, черные дамки 2.

**1.3 Архитектура и алгоритмы**

1.3.1. Архитектура

**Серверная часть**



1. Управление соединениями клиентов

Принятие запроса на подключение с помощью socket для обмена данными между сервером и клиентов

1. Регистрация/Проверка данных

Обработка обращения от клиента и взаимодействие с БД учетных записей.

1. Взаимодействие с сообщениями пользователей

Обработка старых и новых сообщений пользователей с использованием БД истории чата

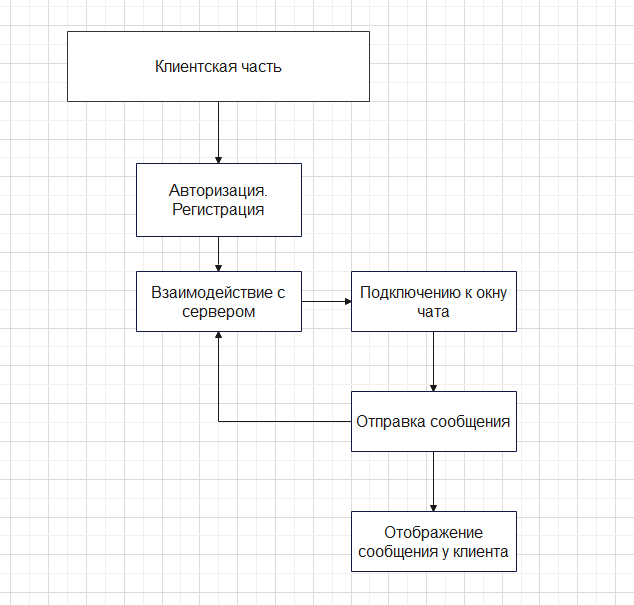
1. БД Учетных данных пользователей

База данных, где хранятся пароли и имена пользователей для дальнейших взаимодействий

1. БД Истории чата

База данных, где хранятся раннее отправленные сообщения пользователей, а также записываются последующие их сообщения

**Клиентская часть**

****

1. Авторизация /Регистрация

Графический интерфейс, созданный с помощью tkinter, позволяющий пользователю отправить запрос на сервер связанный с авторизацией/регистрацией

1. Взаимодействие с сервером

Обработка поступающих запросов от клиента приложения с помощью socket, threading и json

1. Подключение к окну чата

Отображение окна чата, после получения разрешения на подключения от сервера, благодаря библиотеке tkinter

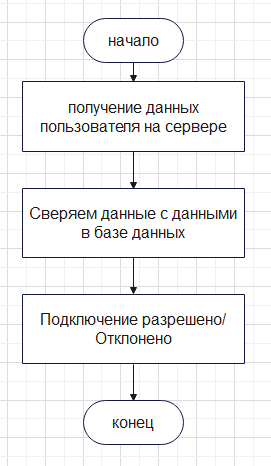
1. Отправка сообщений

Виджет с возможностью отправить сообщение на сервер.

1. Отображение сообщения у клиента

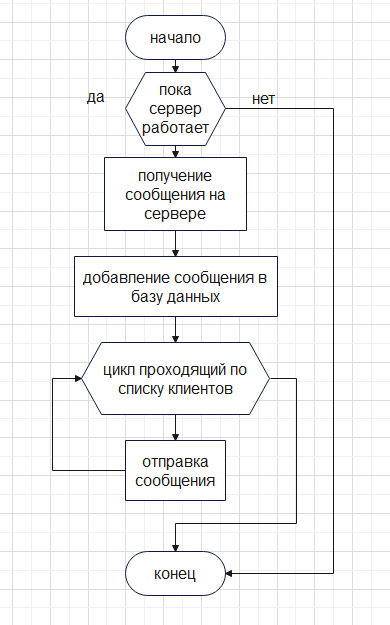
Алгоритм получения отправленного сообщения и его зарисовка в окне чата

1.3.2.1. Алгоритм проверки авторизации



Данный алгоритм необходим для проверки авторизации пользователя. Алгоритм отправляет запрос на сервер и ожидает получения ответа на вопросы «Есть ли пользователь с данным именем» и «Соответствует ли пароль данному имени пользователя?» В случае положительных ответов на оба вопроса сервер позволяет клиенту подключиться к чату.

1.3.2.2. Алгоритм получения сообщения от пользователя и его расспространение остальным клиентам



Данный алгоритм необходим для реализации обмена сообщениями между пользователями в реальном времени, а также сохранения истории переписки. Алгоритм принимает на вход данные сообщение от пользователя.

**1.4 Тестирование**

Весь процесс тестирования проходил вручную, без привлечения специального ПО. На протяжении всего хода разработки, использовался метод белого ящика, так как в любом время имелся доступ ко всем компонентам программы. Всё тестирование выполнялось интуитивным методом, без подготовки специальных тестов.

На протяжении всего хода разработки, по мере добавления новых функций программы, использовалось системное тестирование новых функций, для устранения возникших в ходе написания ошибок. После положительных результатов тестирования функция считалась внедренной.

**2. Источники, использованные при разработке**

Разработка логических компьютерных игр с графическим интерфейсом в среде Питон: учебное пособие для студентов направления 09.03.02 «Информационные системы и технологии» / В.В. Шишкин, Д.С. Афонин. – Ульяновск : УлГТУ, 2023. – 79-88 с.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**Курсовая работа**

**По дисциплине «Алгоритмы и структуры данных»**

**Тема «**Информационная система для обмена оповещаниями в корпоративной сети предприятия»

**Руководство программиста**

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/314-19 РП-01

Листов 15

**Исполнитель**:

студент гр. ИСТбд-21

*Жуклин Никита Дмитриевич*

« » 2023 г.

**2023**

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Приложение «Мессенджер» создано для обмена оповещениями внутри сети предприятия.

**Назначение и функции:**

* Распространение информации. Обеспечивает механизм быстрого и надежного распространения важных сообщений и оповещений среди сотрудников предприятия.
* Своевременное уведомление. Предоставляет средство мгновенного информирования о критических событиях, изменениях в расписании, экстренных уведомлениях и других ситуациях, требующих немедленного внимания.
* Удобство использования. Обеспечивает удобный интерфейс для отправки и приема сообщений, что делает процесс коммуникации интуитивно понятным и доступным для всех пользователей

**Пользователи**. Пользователями приложения являются работники предприятия.

**Требования к устройству.**

* Операционная система: Windows
* Разрешение экрана: 1000x750 и выше
* Интернет-соединение: подключение к сети предприятия

**1.2 Условия, необходимые для использования приложения**

Рекомендуется к использованию на Windows 7.

При создании программы использовались встроенные библиотеки “tkinter”, “json”, “socket“, “threading“, “sqlite3“ и сторонняя библиотека “notifypy“

Разработка велась в “PyCharm community edition 2022.3” на версии языка программирования Python 3.9.

**2. Характеристики программы**

**2.1 Характеристики приложения**

Значимых строк кода 281(client,py), 151(server.py), 432(client.py + server.py)

Использованные библиотеки:

“tkinter“ — это библиотека для разработки графического интерфейса на языке Python. Tkinter позволяет создавать простые GUI-приложения, используя различные виджеты и менеджеры компоновки для управления их расположением на экране.

“threading“ — это библиотека, позволяющая создавать и управлять потоками выполнения в программе. Потоки позволяют выполнять несколько задач одновременно. Модуль «threading» позволяет создавать, запускать и управлять потоками выполнения, что полезно для параллельного выполнения задач и управления ресурсами.

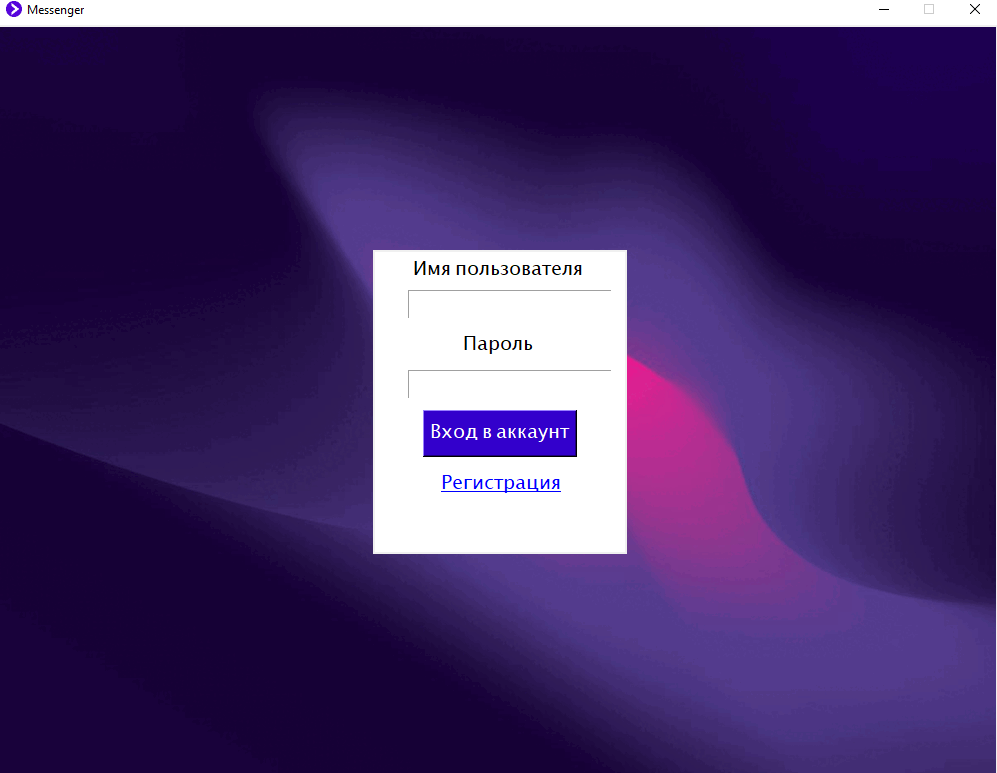
“socket“ — эта библиотека предоставляет функции для создания сетевых приложений, включая создание серверов и клиентов, обмен сообщениями и управление соединениями.

“json“ — эта библиотека позволяет кодировать и декодировать данные в/из формата JSON. Она включает методы для работы с JSON-строками, преобразования Python-объектов в JSON и обратно. JSON - это *lightweight* формат обмена данными, который удобен для чтения и записи как людьми, так и компьютерами. Он используется для передачи структурированных данных между сервером и клиентом.

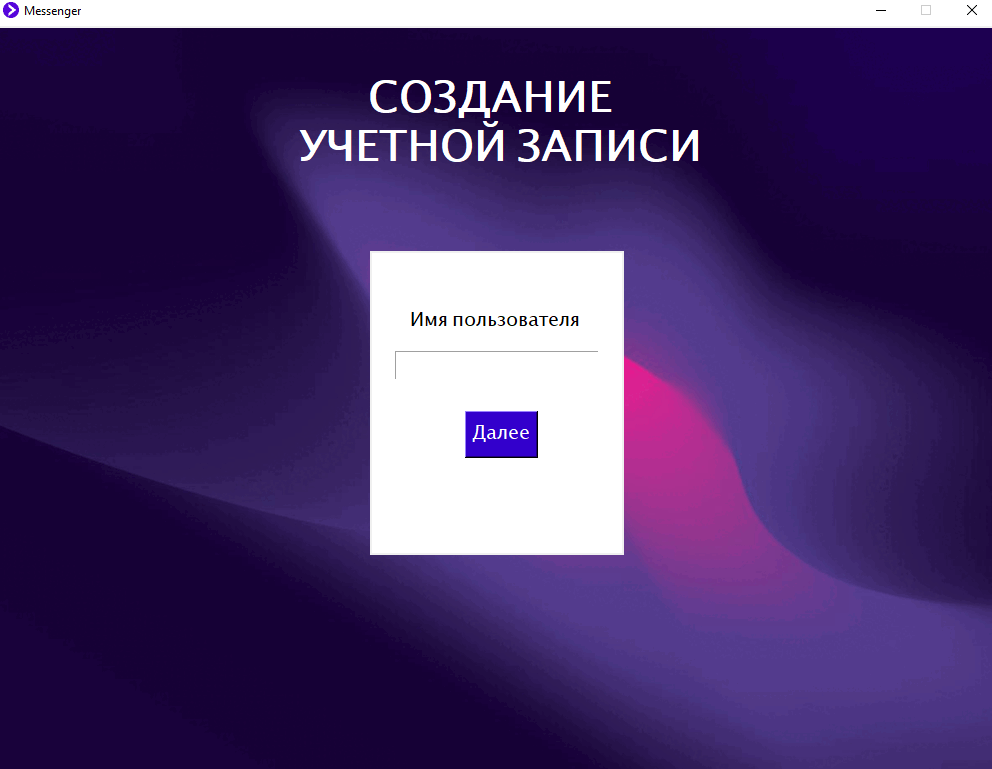
“sqlite3“ — эта библиотека предоставляет возможности работы с базами данных SQLite, включая создание, чтение, запись и выполнение запросов к базе данных. SQLite3 — это легковесная реляционная база данных, которая хранит базы данных в одном файле без необходимости сервера. Она хорошо подходит для небольших приложений или прототипов.

“notifypy“ — это библиотека для уведомлений на рабочем столе. Она позволяет отправлять уведомления на различные платформы. Удобна для создания уведомлений, информирующих пользователя о событиях в приложениях, например, об успешном завершении задачи или получении новых сообщений.

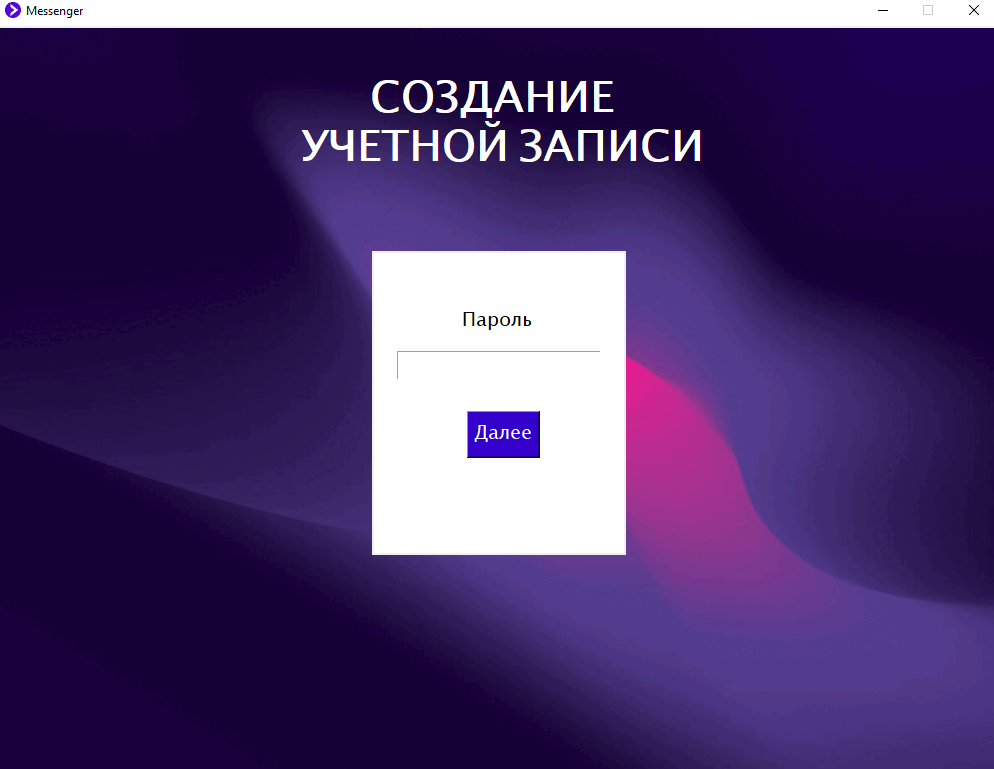
Первым окном, которое видит пользователь является окно авторизации c возможностью перейти к регистрации



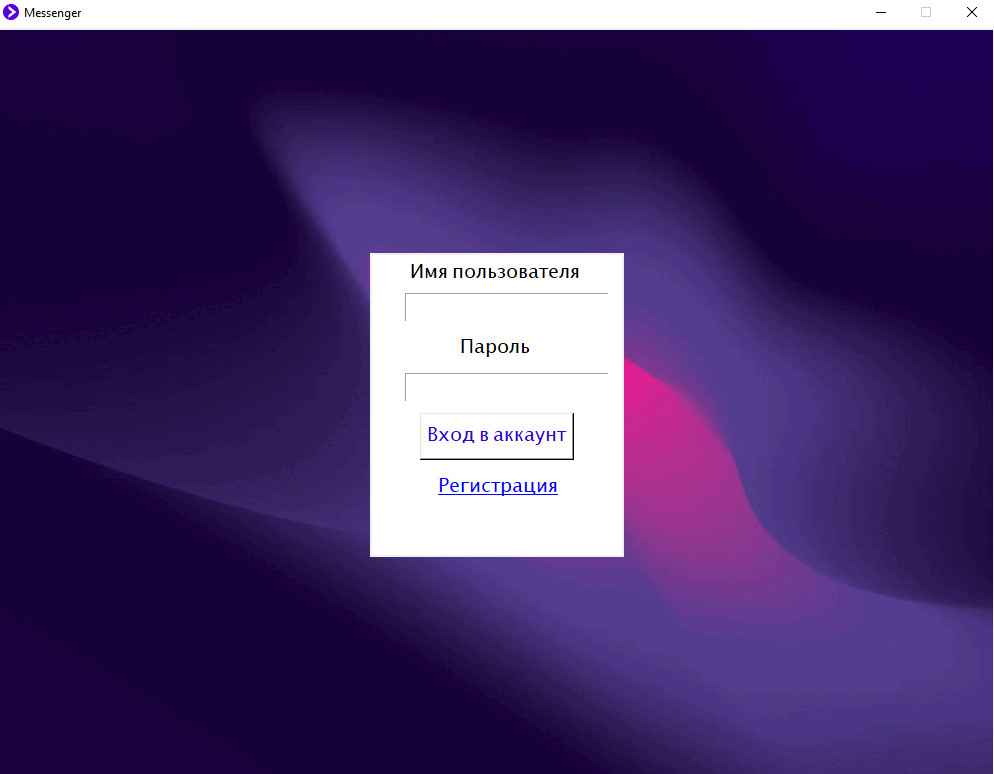
Окно регистрации, ввод имени пользователя

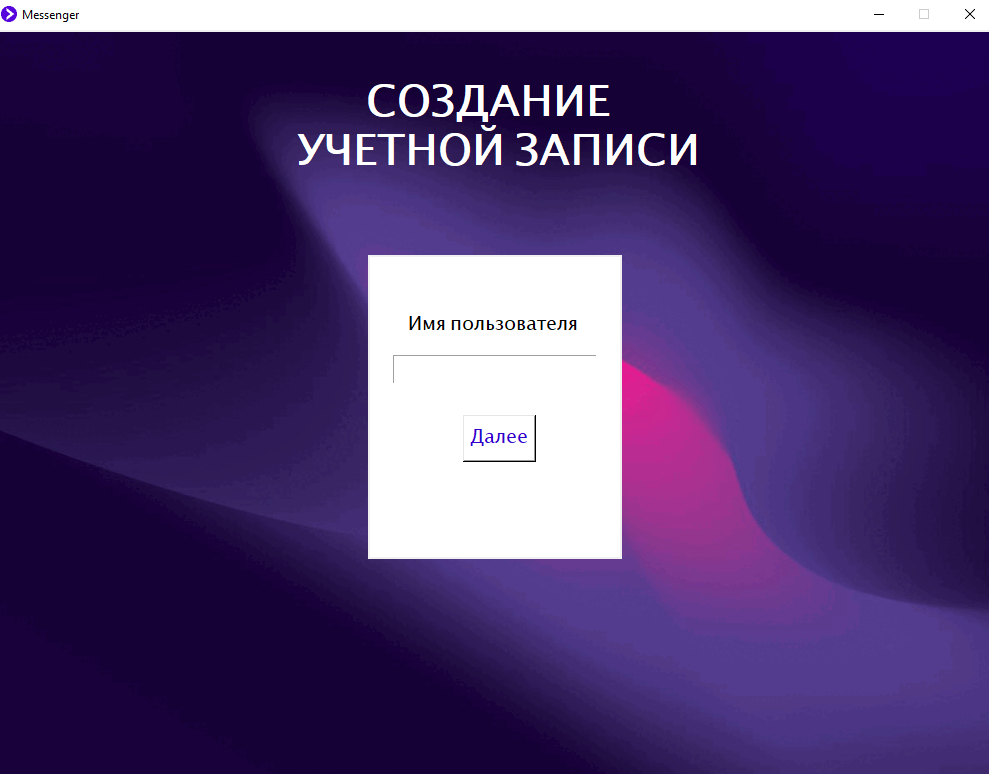


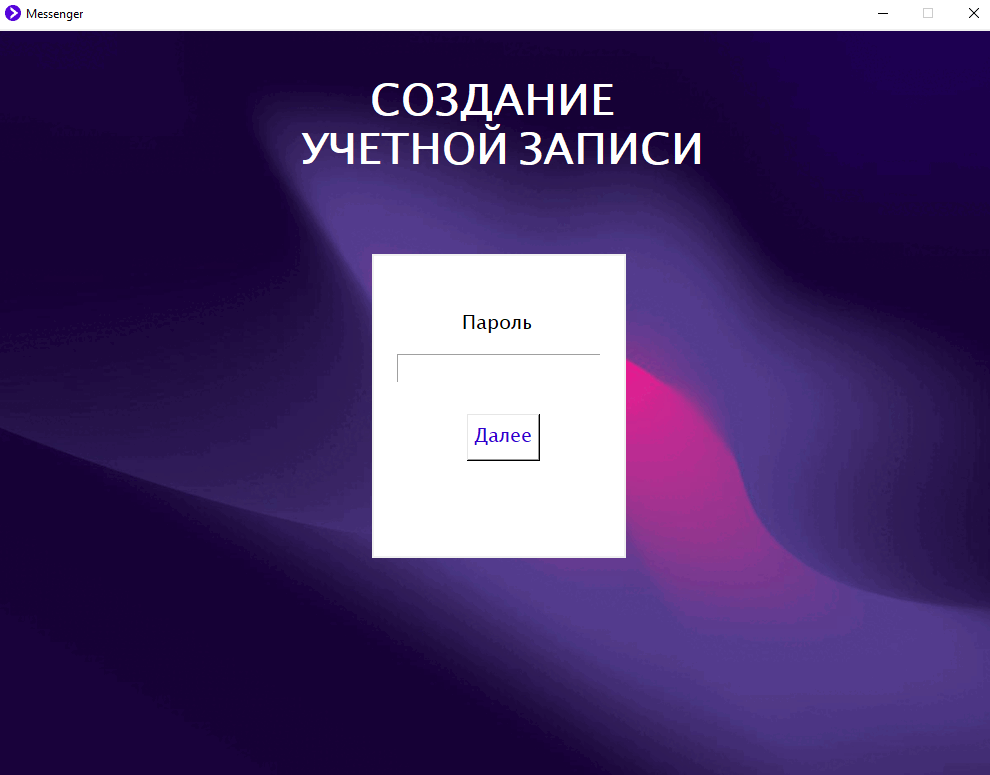
Окно регистрации, ввод пароля



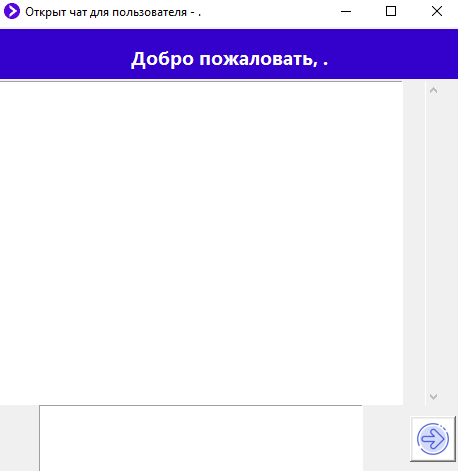
Наведение на кнопки



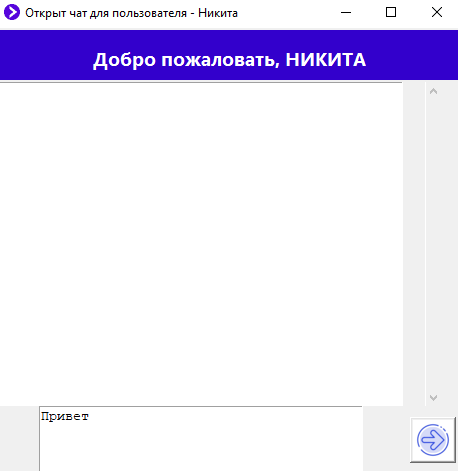


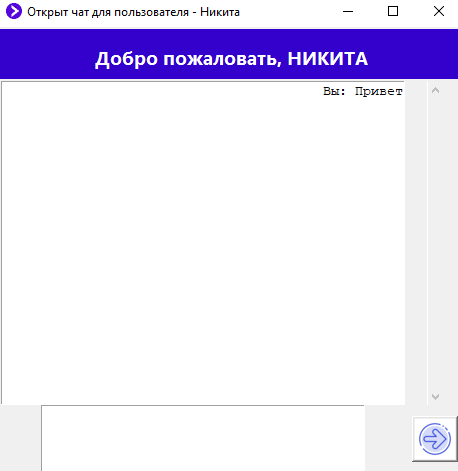


Отрисовка чата пользователей

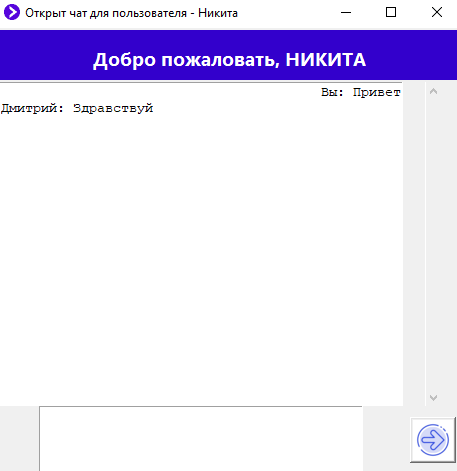


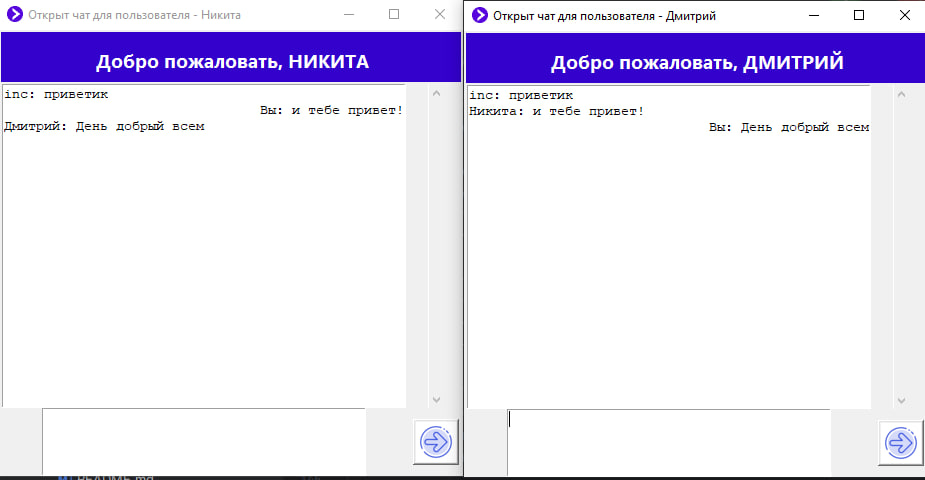
Отправка сообщений в чат





Получение сообщений от пользователей и их отображение





**2.2 Особенности реализации приложения**

В программе используется серверная часть с подключением БД для хранения истории чата, а также паролей и юзернеймов пользователей. Также используется json для надежного обмена данными между сервером и клиентов.

**3. Обращение к программе**

Код клиента содержит в себе следующие алгоритмы:

1. log\_window

Выполняет подключение к серверу с помощью библиотеки socket. Отрисовывает окно авторизации login\_window с возможностями войти в аккаунт или же перейти к окну регистрации.

1.1) Метод focus\_in

При наведении на кнопку для авторизации изменяет ее цвет текста и цвет фона

1.2) Метод focus\_off

При изменении фокуса с кнопки для авторизации на другие элементы, кнопка изменяет свой цвет текста и цвет фона

1. login

Клиент отправляет на сервер информацию о том, что пользователь хочет войти в аккаунт. После пакует username и password в список, преобразуя в json, а затем отправляет эти данные на сервер. Затем клиент ожидает ответа от сервера с разрешением на подключение или же отказом. В случае разрешения подключения осуществляется переход в окно чата. В ином случае выскакивает соответствующая ошибка.

1. reg\_window\_username

Перерисовывает окно login\_window под создание учетной записи. Создает поле для ввода имени пользователя и кнопку для продолжения со следующими методами:

3.1) Метод focus\_in

При наведении на кнопку для авторизации изменяет ее цвет текста и цвет фона

3.2)Метод focus\_off

При изменении фокуса с кнопки для авторизации на другие элементы, кнопка изменяет свой цвет текста и цвет фона

3.3) Метод check\_data

При нажатии на кнопку происходит проверка поля ввода на корректность данных по длине. Если условие выполнено, то осуществляется вызов reg\_window\_password с передачей имени пользователи в этот алгоритм

1. reg\_window\_password

Перерисовывает окно login\_window после алгоритма reg\_window\_username, заменяя поле ввода имени пользователя на поле для ввода пароля. Кнопка для продолжения также перерисовывается и имеет следующие методы:

4.1) Метод focus\_in

При наведении на кнопку для авторизации изменяет ее цвет текста и цвет фона

4.2) Метод focus\_off

При изменении фокуса с кнопки для авторизации на другие элементы, кнопка изменяет свой цвет текста и цвет фона

4.3) Метод check\_data

При нажатии на кнопку происходит проверка поля ввода на корректность данных по длине. Если условие выполнено, то осуществляется вызов алгоритма register с передачей имени пользователя и введённого пароля.

1. register

Клиент отправляет на сервер информацию о том, что пользователь хочет зарегистрироваться. После чего упаковывает username и password в список, преобразуя в json, а затем отправляет эти данные на сервер. Затем клиент ожидает ответа от сервера с информацией об успехе регистрации. В случае успешной регистрации пользователя происходит отключения от сервера, для прохождения повторного выбора опции входа или регистрации, осуществляется вызов log\_window.

1. open\_chat

Отрисовывает окно чата chat\_window, в котором можно прочитать старые сообщения и новые сообщения от пользователей, для этого в окне имеется средство прокрутки. Имеется окно ввода для своего сообщения, а также кнопка для отправки сообщений, которая вызывает алгоритм send\_message. Также отправка сообщений вызывается с помощью нажатия кнопки Enter в окне ввода. И напоследок создается новый поток с помощью threading с запуском алгоритма обработки сообщений handle\_message

1. handle\_message

При подключении пользователя к чату прогружает историю сообщения с базы данных, находящейся на сервере. После загрузки истории переходит в режим принятия сообщений в реальном времени от остальных пользователей. Если окно чата окажется свернутым, то в центр уведомления будут приходить оповещения о новых сообщениях от пользователей с помощью библиотеки notifypy.

1. send\_message

При вызове обрабатывает поле ввода из окна chat\_window, удаляя пустые символы в начале и конце сообщения. После этого отправляет обработанное сообщение на сервер, очищает поле ввода и вызывает алгоритм display\_message для отображения сообщения у пользователя, который отправил данное сообщение.

1. display\_message

Проверяет чтобы сообщение, полученное после обработки в send\_message не оказалось пустым. Если сообщение не пустое выводит сообщение пользователя справа, визуально отделяя его от чужих сообщений. Также имя пользователя сменяется на «Вы»

Код сервера содержит следующие алгоритмы:

1. accept\_clients

Принимает на вход подключение от клиента. После чего ожидает выбора операции от пользователя. Например, вход или регистрация.

* Если выбрана операция входа сервер ожидает на вход данные в формате json, при их получении сервер отправляет данные пользователя на проверку(check\_credentials). Если данные прошли проверку успешно, то сервер отправляет информацию о разрешении подключения к чату на клиент. После запускает поток для прослушивания сообщений от пользователя и отправления ему сообщений от других пользователей(handle\_client). В случае если данные не прошли проверку отправляет информацию об ошибке.
* Если выбрана операция регистрации сервер ожидает на вход данные в формате json, при их получении сервер обращается к алгоритм register\_user для проверки имени пользователя на существование в БД. Если регистрация успешно, то отправляется соответствующее сообщение на клиенте, если неуспешно, то отправляет информацию о неудаче.

1. check\_credentials

Алгоритм подключается к БД users\_credentials и проверяет данные пользователя на соответствие. В случае успеха передает True, в случае неудачи False

1. register\_user

Алгоритм подключается к БД users\_credentials и проверяет имя пользователя на существование в БД. Если такое имя пользователя еще не числится в БД, то возвращает True и записывает данные нового аккаунта такие как username и password. В случае если такое имя пользователя уже существует, то возвращает False

1. handle\_client

При подключении клиента вызывает алгоритм для получения истории сообщений(retrieve\_chat\_history). Начинает прослушивать сообщения от пользователей. При получении сообщения запускает алгоритм save\_message\_to\_db, чтобы сохранить сообщение в БД chat\_history.db. После запускает алгоритм broadcast, чтобы отправить полученное сообщение всем пользователям чата.

1. save\_message\_to\_db

Получает на вход имя пользователя и его сообщения, а затем сохраняет эту информацию в базу данных c историей чата(chat\_history.db)

1. broadcast

Принимает на вход строку вида «username:message». Алгоритм проходит по списку всех клиентов и отправляет каждому полученное на вход сообщение. Если с каким то клиентом возникает проблема, то подключение закрывается, а клиент удаляется из списка.

Алгоритм подключается к БД users\_credentials и проверяет имя пользователя на существование в БД. Если такое имя пользователя еще не числится в БД, то возвращает True и записывает данные нового аккаунта такие как username и password. В случае если такое имя пользователя уже существует, то возвращает False

Библиотеки, используемые во время разработки приложения:

Библиотека “json“ – предоставляет возможность для работы с файлами json. Позволяет кодировать и декодировать данные в/из формата JSON.

Библиотека “socket“ – предоставляет инструменты для работы с сокетами. Позволяет создавать клиент-серверные приложения, обмениваться данными между компьютерами через сеть, взаимодействовать по протоколам TCP/IP и др.

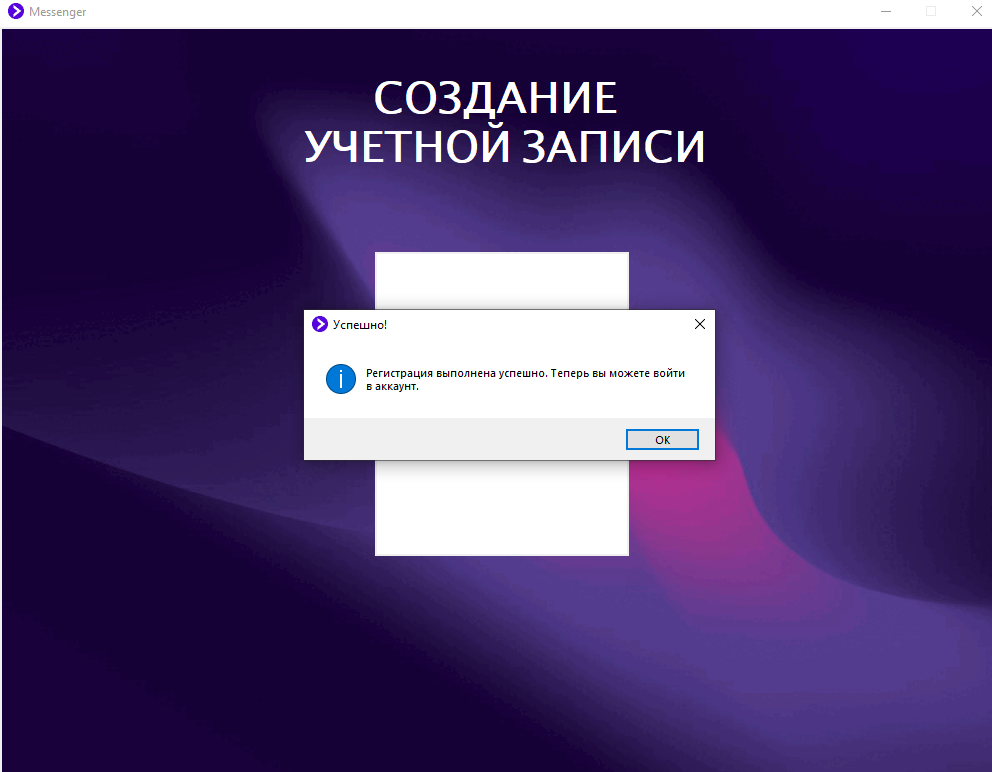
Библиотека “notifypy“ – позволяет создавать различные уведомления на рабочем столе.

Библиотека “threading” – дает возможность создавать и управлять потоками выполнения. Потоки позволяют выполнять параллельные задачи, ускоряя работу программы и позволяя работать с асинхронными задачами.

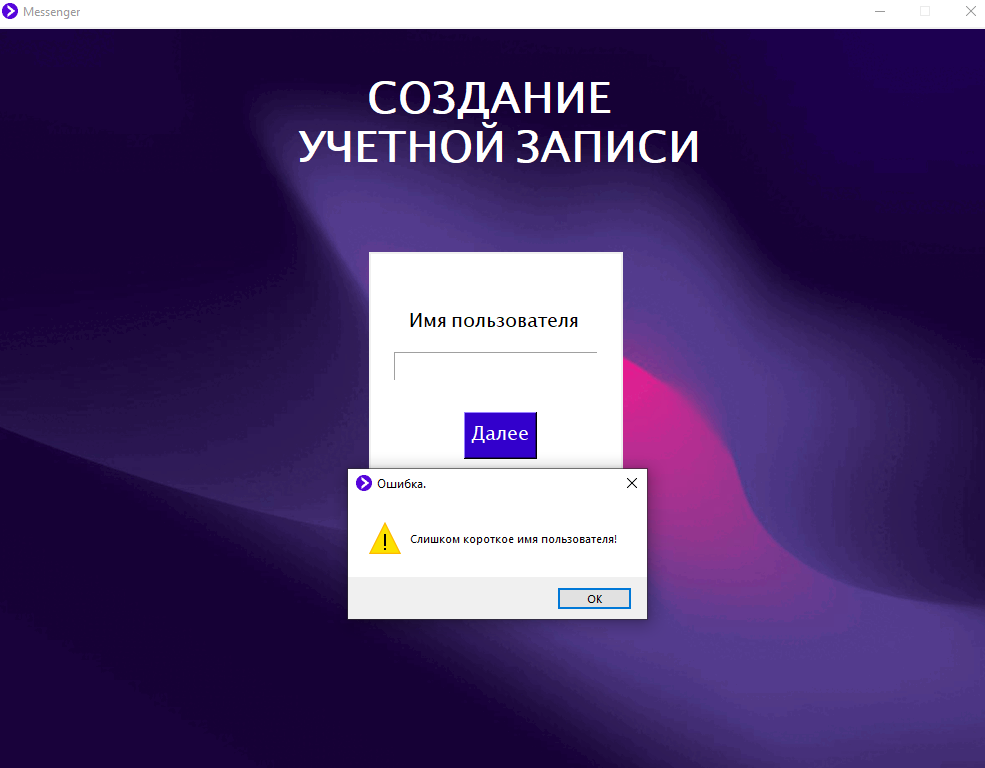
Библиотека “tkinter” – библиотека для создания графического интерфейса. Предоставляет инструменты для создания окон, кнопок, полей ввода и др.

**4. Сообщения**

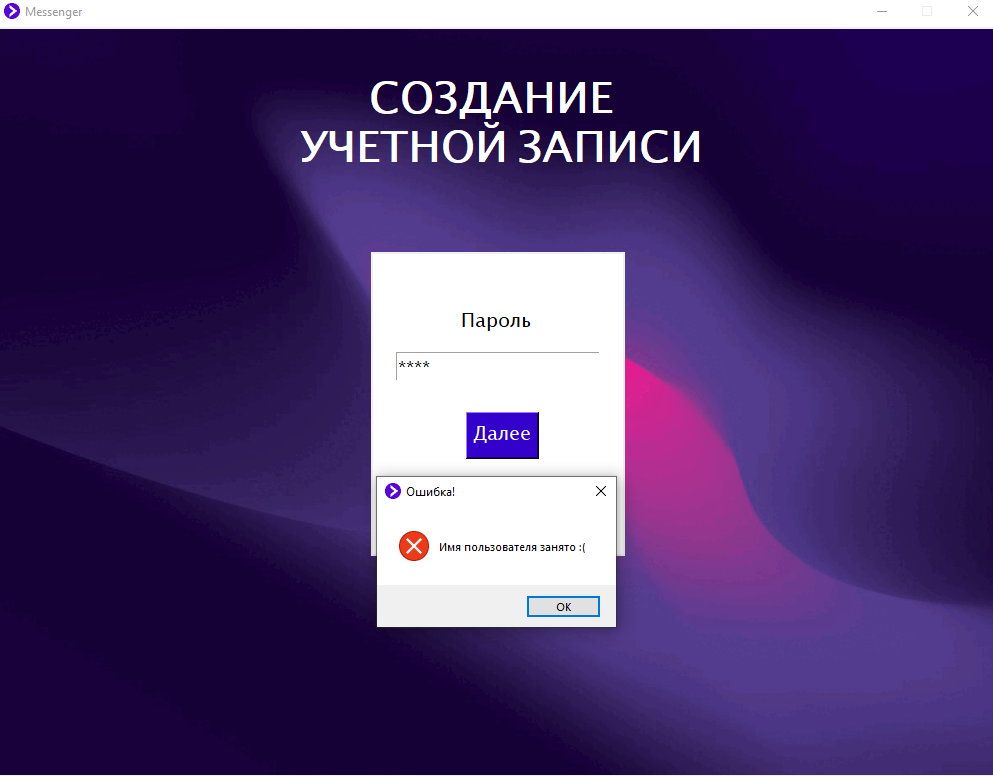
Успешная регистрация



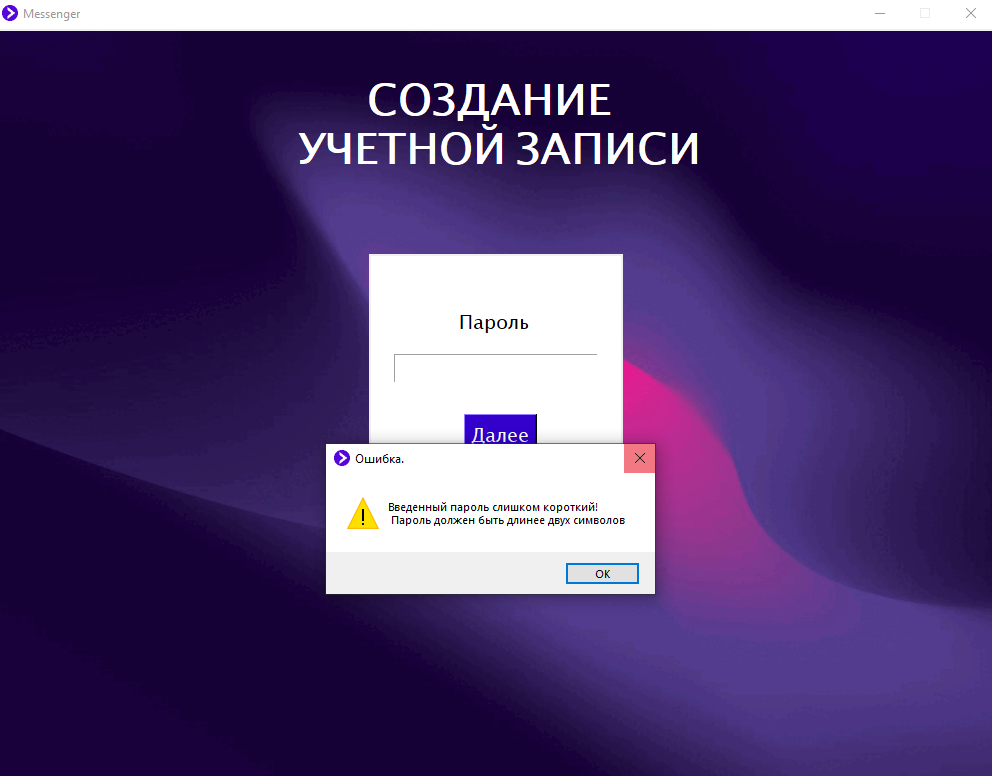
Введенное имя пользователя слишком короткое



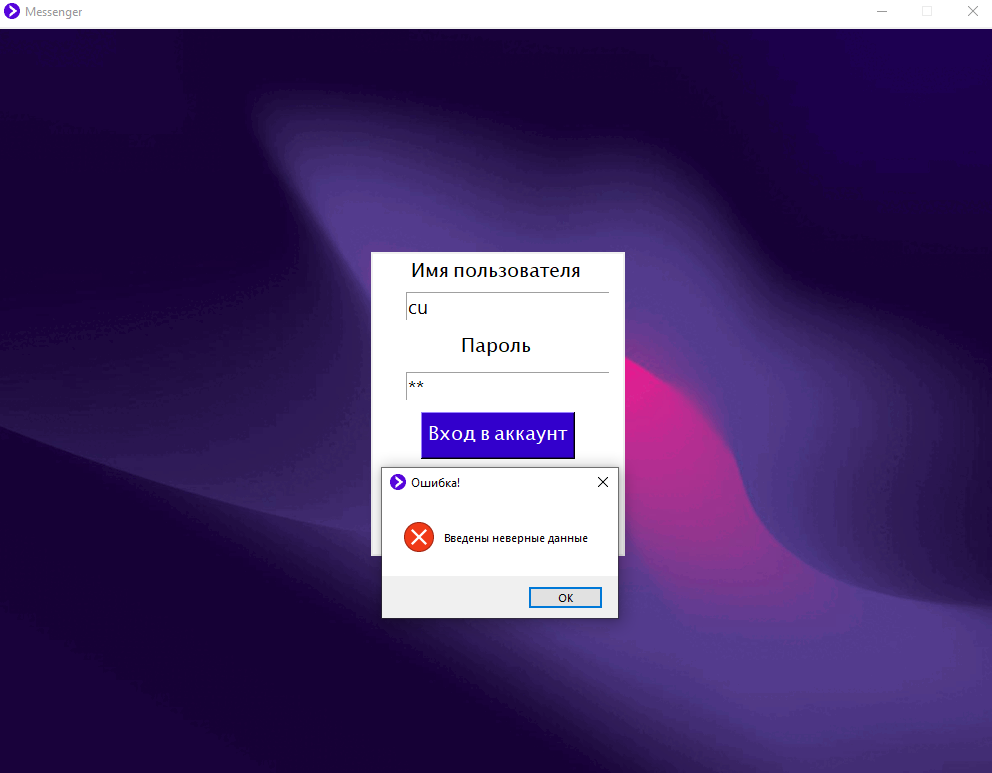
Аккаунт с таким именем пользователя уже существует



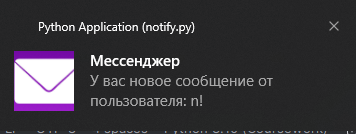
Слишком короткий пароль



Введены неверный пароль или имя пользователя



Звуковое оповещение в Windows



**Текст программы**

server.py

**import** **json**

**import** **socket**

**import** **threading**

**import** **sqlite3**

**def** **register\_user**(username, password):

conn = sqlite3.connect('user\_credentials.db')

cursor = conn.cursor()

**try**:

info = cursor.execute('SELECT \* FROM users WHERE username=?', (username,))

**if** info.fetchone() **is** None:

# Делаем когда нету человека в бд

cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, password))

conn.commit()

conn.close()

**return** True

**else**:

# Делаем когда есть человек в бд

conn.close()

**return** False

**except** sqlite3.IntegrityError:

conn.close()

**return** False

**def** **check\_credentials**(username, password):

conn = sqlite3.connect('data/user\_credentials.db')

cursor = conn.cursor()

cursor.execute("SELECT \* FROM users WHERE username = ? AND password = ?", (username, password))

user = cursor.fetchone()

conn.close()

**return** user **is** **not** None

**def** **save\_message\_to\_db**(username, message):

conn = sqlite3.connect('data/chat\_history.db')

c = conn.cursor()

c.execute("INSERT INTO messages (username, message) VALUES (?, ?)", (username, message))

conn.commit()

conn.close()

**def** **retrieve\_chat\_history**(client\_socket):

**global** chat\_history

conn = sqlite3.connect('data/chat\_history.db')

c = conn.cursor()

c.execute("SELECT \* FROM messages")

history = c.fetchall()

json\_messages = json.dumps(history)

client\_socket.sendall(json\_messages.encode())

chat\_history = history

conn.close()

**def** **handle\_client**(client\_socket, username):

retrieve\_chat\_history(client\_socket)

**while** True:

**try**:

message = client\_socket.recv(**1024**).decode('utf-8')

**if** message:

save\_message\_to\_db(username, message)

chat\_history.append((len(chat\_history) + **1**, username, message))

broadcast(f"{username}: {message}")

**except** **Exception** **as** e:

**print**(f"Ошибка: {e}")

**break**

**def** **broadcast**(message):

**for** client **in** clients:

**try**:

client.send(message.encode('utf-8'))

**except** **Exception** **as** e:

**print**(f"Ошибка при отправке сообщения: {e}")

clients.remove(client)

client.close()

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind(('localhost', **9090**))

server.listen(**100**)

clients = []

**def** **accept\_clients**():

**while** True:

client\_socket, addr = server.accept()

option = client\_socket.recv(**1024**).decode('utf-8')

**if** option.lower() == "login":

data = client\_socket.recv(**4096**).decode('utf-8')

username, password = json.loads(data)

**global** name

name = username

**if** check\_credentials(username, password):

clients.append(client\_socket)

client\_socket.send("Login successful".encode('utf-8'))

**print**(f"{username} connected")

threading.Thread(target=handle\_client, args=(client\_socket, username)).start()

**else**:

client\_socket.send("Введены неверные данные".encode('utf-8'))

**elif** option.lower() == "register":

data = client\_socket.recv(**4096**).decode('utf-8')

username, password = json.loads(data)

**if** register\_user(username, password):

client\_socket.send("Регистрация выполнена успешно".encode('utf-8'))

**else**:

client\_socket.send("Имя пользователя занято :(".encode('utf-8'))

threading.Thread(target=accept\_clients).start()

client.py

**import** **json**

**import** **tkinter** **as** **tk**

**from** **tkinter** **import** messagebox

**from** **notifypy** **import** Notify

**import** **socket**

**import** **threading**

**def** **handle\_messages**():

history\_receive = True

**while** True:

**try**:

**if** history\_receive:

received\_data = client\_socket.recv(**4096**).decode('utf-8')

messages = json.loads(received\_data)

**for** message **in** messages:

**if** message[**1**] != name:

chat\_text.config(state='normal')

chat\_text.insert(tk.END, f'{message[1]}: {message[2]}' + '**\n**')

chat\_text.config(state='disabled')

**else**:

chat\_text.config(state='normal')

chat\_text.insert(tk.END, f'Вы: {message[2]}' + '**\n**', 'right')

chat\_text.config(state='disabled')

chat\_text.see(tk.END)

history\_receive = False

**else**:

message = client\_socket.recv(**1024**).decode('utf-8')

**if** message:

**if** message.split(':')[**0**] != name:

chat\_text.config(state='normal')

chat\_text.insert(tk.END, message + '**\n**')

chat\_text.config(state='disabled')

**if** chat\_window.wm\_state() == 'iconic':

notification.message = f"У вас новое сообщение от пользователя: {message.split(':')[0]}!"

notification.audio = 'resources/notification.wav'

notification.icon = "resources/notif1.jpg"

notification.send()

chat\_text.see(tk.END)

**except**:

**break**

**def** **login**(username, password):

option = "login"

client\_socket.send(option.encode('utf-8'))

data = [username, password]

json\_data = json.dumps(data)

client\_socket.sendall(json\_data.encode())

response = client\_socket.recv(**1024**).decode('utf-8')

**if** response == "Login successful":

login\_window.destroy()

open\_chat(username)

**else**:

messagebox.showerror("Ошибка!", response)

login\_window.destroy()

client\_socket.close()

log\_window()

**def** **register**(username, password):

option = "register"

client\_socket.send(option.encode('utf-8'))

data = [username, password]

json\_data = json.dumps(data)

client\_socket.sendall(json\_data.encode())

response = client\_socket.recv(**1024**).decode('utf-8')

**if** response == "Регистрация выполнена успешно":

messagebox.showinfo("Успешно!", "Регистрация выполнена успешно. Теперь вы можете войти в аккаунт.")

reg\_canvas.destroy()

canvas.delete(register\_text)

login\_window.destroy()

client\_socket.close()

log\_window()

**else**:

messagebox.showerror("Ошибка!", 'Имя пользователя занято :(')

reg\_canvas.destroy()

canvas.delete(register\_text)

login\_window.destroy()

client\_socket.close()

log\_window()

**def** **display\_message**(message, align):

**if** message.split('Вы:')[**1**] != ' ':

chat\_text.config(state=tk.NORMAL) # Enable editing the text widget

chat\_text.insert(tk.END, message + '**\n**', align) # Use the tag to align the message

chat\_text.config(state=tk.DISABLED) # Disable editing the text widget

chat\_text.see(tk.END)

**def** **send\_message**(event=None):

message = message\_entry.get(**1.0**, tk.END).strip()

**if** message:

client\_socket.send(message.encode('utf-8'))

display\_message(f"Вы: {message}", 'right') # Display the user's message on the right

message\_entry.delete(**1.0**, tk.END)

**def** **open\_chat**(username):

**global** message\_entry, chat\_window, notification, chat\_window

notification = Notify()

notification.title = 'Мессенджер'

**def** **on\_closing**():

client\_socket.close()

chat\_window.destroy()

chat\_window = tk.Tk()

chat\_window.title(f"Открыт чат для пользователя - {username}")

chat\_window.protocol("WM\_DELETE\_WINDOW", on\_closing) # Handle window closing event

chat\_window.iconbitmap(r'resources/icon.ico')

**global** name

name = username

chat\_canvas = tk.Canvas(chat\_window, width=**460**, height=**50**, bg='#3300cc')

chat\_canvas.grid(row=**0**, column=**0**, columnspan=**2**)

welcome\_text = chat\_canvas.create\_text(**232.5**, **30**, anchor='center', text=f'Добро пожаловать, {name.upper()}',

fill="#fff",

font=('Segoe UI', **14**, 'bold'))

**global** chat\_text

scrollbar = tk.Scrollbar(chat\_window)

scrollbar.grid(row=**1**, column=**1**, sticky=tk.NS)

chat\_text = tk.Text(chat\_window, height=**20**, width=**50**, wrap="word", yscrollcommand=scrollbar.set)

chat\_text.grid(row=**1**, column=**0**)

scrollbar.config(command=chat\_text.yview)

chat\_text.config(state='disabled')

chat\_text.tag\_configure('left', justify='left')

chat\_text.tag\_configure('right', justify='right')

message\_entry = tk.Text(chat\_window, width=**40**, height=**4**)

message\_entry.grid(row=**2**, column=**0**)

message\_entry.see(**1.0**)

message\_entry.mark\_set('insert', '1.0')

message\_entry.bind('<Return>', send\_message)

but\_photo = tk.PhotoImage(file='resources/next button.gif')

send\_button = tk.Button(chat\_window, image=but\_photo, command=send\_message)

send\_button.grid(row=**2**, column=**1**)

threading.Thread(target=handle\_messages).start()

chat\_window.mainloop()

**def** **reg\_window\_username**(event):

auth\_canvas.destroy()

**global** register\_text

register\_text = canvas.create\_text(**500**, **100**, anchor='center', text=' СОЗДАНИЕ**\n** УЧЕТНОЙ ЗАПИСИ',

fill="White",

font=('Perpetua', **32**, 'bold'))

**def** **focus\_in**(e=None):

next\_button.configure(fg='#3300cc') # Задаём кнопке нужные цвета

next\_button.configure(bg='#fff')

**def** **focus\_out**(e=None):

next\_button.configure(bg='#3300cc')

next\_button.configure(fg='#fff')

**def** **check\_data**():

**if** len(username\_entry.get().rstrip(' ')) >= **1**:

username = username\_entry.get()

reg\_window\_password(username)

**else**:

messagebox.showwarning('Ошибка.', 'Слишком короткое имя пользователя!')

**global** reg\_canvas

reg\_canvas = tk.Canvas(canvas, width=**250**, height=**300**, bg='white')

reg\_canvas.place(x='375', y='225')

reg\_canvas.pack\_propagate(False)

**global** username\_text, username\_entry, next\_button

username\_text = reg\_canvas.create\_text(**125**, **70**, text="Имя пользователя", fill="Black", font=('Perpetua', **14**))

username\_entry = tk.Entry(reg\_canvas, width=**20**, font=('Segoe UI', **14**))

username\_entry.config(highlightbackground="black")

next\_button = tk.Button(reg\_canvas, text="Далее", pady='5',

cursor='hand2', command=check\_data, fg='#fff', bg='#3300cc',

font=('Perpetua', **14**))

next\_button.bind('<Enter>', focus\_in) # При входе курсора в область кнопки выполняем focus\_in

next\_button.bind('<Leave>', focus\_out) # При выходе курсора из области кнопки выполняем focus\_out\_out

username\_entry.pack\_propagate(False)

username\_entry.place(x='25', y='100')

next\_button.place(x='95', y='160')

**def** **reg\_window\_password**(username):

**def** **focus\_in**(e=None):

next\_button2.configure(fg='#3300cc') # Задаём кнопке нужные цвета

next\_button2.configure(bg='#fff')

**def** **focus\_out**(e=None):

next\_button2.configure(bg='#3300cc')

next\_button2.configure(fg='#fff')

**def** **check\_data**():

**if** len(password\_entry.get().rstrip(' ')) >= **3**:

register(username, password\_entry.get())

**else**:

messagebox.showwarning('Ошибка.', 'Введенный пароль слишком короткий!**\n** Пароль должен быть длинее двух '

'символов')

reg\_canvas.delete(username\_text)

username\_entry.destroy()

next\_button.destroy()

password\_text = reg\_canvas.create\_text(**125**, **70**, text="Пароль", fill="Black", font=('Perpetua', **14**))

password\_entry = tk.Entry(reg\_canvas, width=**20**, show="\*", font=('Segoe UI', **14**))

next\_button2 = tk.Button(reg\_canvas, text="Далее", pady='5',

cursor='hand2', command=check\_data, fg='#fff', bg='#3300cc',

font=('Perpetua', **14**))

next\_button2.place(x='95', y='160')

next\_button2.bind('<Enter>', focus\_in) # При входе курсора в область кнопки выполняем focus\_in

next\_button2.bind('<Leave>', focus\_out) # При выходе курсора из области кнопки выполняем focus\_out\_out

password\_entry.place(x='25', y='100')

**def** **log\_window**():

**def** **focus\_in**(e=None):

login\_button.configure(fg='#3300cc') # Задаём кнопке нужные цвета

login\_button.configure(bg='#fff')

**def** **focus\_out**(e=None):

login\_button.configure(bg='#3300cc')

login\_button.configure(fg='#fff')

**global** client\_socket, login\_window

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect(('localhost', **9090**))

login\_window = tk.Tk()

login\_window.geometry('1000x750') # Задаём размер

login\_window.title("Messenger")

login\_window.resizable(width=False, height=False)

login\_window.iconbitmap(r'resources/icon.ico')

**global** canvas

canvas = tk.Canvas(login\_window, width=**1000**, height=**750**)

canvas.pack()

canvas.pack\_propagate(False)

img = tk.PhotoImage(file='resources/back.gif')

canvas.create\_image(**0**, **0**, anchor=tk.NW, image=img)

**global** auth\_canvas

auth\_canvas = tk.Canvas(canvas, width=**250**, height=**300**, bg='white')

auth\_canvas.place(x='375', y='225')

auth\_canvas.pack\_propagate(False)

username\_text = auth\_canvas.create\_text(**125**, **20**, text="Имя пользователя", fill="Black", font=('Perpetua', **14**))

username\_entry = tk.Entry(auth\_canvas, width=**20**, font=('Segoe UI', **14**))

username\_entry.config(highlightbackground="black")

password\_text = auth\_canvas.create\_text(**125**, **95**, text="Пароль", fill="Black", font=('Perpetua', **14**))

password\_entry = tk.Entry(auth\_canvas, width=**20**, show="\*", font=('Segoe UI', **14**))

login\_button = tk.Button(auth\_canvas, text="Вход в аккаунт", pady='5',

command=**lambda**: login(username\_entry.get(), password\_entry.get()),

cursor='hand2', fg='#fff', bg='#3300cc',

font=('Perpetua', **14**))

login\_button.bind('<Enter>', focus\_in) # При входе курсора в область кнопки выполняем focus\_in

login\_button.bind('<Leave>', focus\_out) # При выходе курсора из области кнопки выполняем focus\_out\_out

register\_button = tk.Label(auth\_canvas, text="Регистрация", fg='blue', bg='white', cursor='hand2',

font=('Perpetua', **14**, 'underline'))

register\_button.bind("<Button-1>", reg\_window\_username)

username\_entry.pack\_propagate(False)

username\_entry.place(x='35', y='40')

password\_entry.place(x='35', y='120')

login\_button.place(x='50', y='160')

register\_button.place(x='65', y='220')

login\_window.mainloop()

log\_window()