Создаем приложение мессенджер desktop версия

Выбираем архитектуру для разработки desktop приложения:

Выбрал архитектуру клиент-свервер. Этот шаблон позволяет разделить приложение на две основные компоненты:

клиентскую часть (desktop-приложение) и серверную часть (сервер мессенджера).

Основания:

- Масштабируемость: Архитектура клиент-сервер позволяет легко масштабировать серверную часть для обработки большого количества пользователей и сообщений.

- Централизованное управление: Сервер может обеспечивать централизованное управление пользователями, сообщениями и безопасностью, что важно для мессенджера.

- Синхронизация данных: Сервер может обеспечивать синхронизацию сообщений между разными клиентами, что важно для обеспечения последовательности сообщений.

- Обновления и безопасность: Централизованная архитектура облегчает обновления и обеспечивает высокий уровень безопасности данных.

Описание use case диаграммы:

Use Case диаграмма поможет определить основные функции и взаимодействия с акторами в системе. Определим акторов и их роли:

Пользователь мессенджера (User) - Основной актор, который использует приложение мессенджера для общения.

Администратор (Administrator) - Дополнительный актор, который может управлять пользователями, группами и настройками приложения.

Теперь определим основные Use Case (функциональности) нашего мессенджера:

Отправка сообщения (Send Message) - Пользователь может отправлять сообщения другим пользователям.

Получение сообщения (Receive Message) - Пользователь может получать сообщения от других пользователей.

Создание группы (Create Group) - Пользователь или администратор может создавать группы чата для общения с несколькими пользователями одновременно.

Добавление пользователя в группу (Add User to Group) - Администратор может добавлять пользователей в существующие группы.

Удаление пользователя из группы (Remove User from Group) - Администратор может удалять пользователей из групп.

Управление настройками профиля (Manage Profile Settings) - Пользователь может изменять свои настройки профиля, такие как фотография, статус и прочее.

Управление пользователями (Manage Users) - Администратор может управлять пользователями, блокировать пользователей и т. д.

UML диаграмма:

- Desktop Messenger - это основной класс, представляющий desktop-приложение мессенджера.

UserInterface, LogicLayer и CommunicationLayer представляют разные компоненты внутри desktop-приложения.

- UserUI представляет пользовательский интерфейс с чатовыми окнами, списком контактов и настройками.

- Server - это серверная часть мессенджера с управлением пользователями, хранением сообщений и обеспечением коммуникации.

- User и Group - это классы, представляющие пользователей и группы соответственно.

- Message - класс, представляющий сообщения.

- Administrator - класс, представляющий администратора системы.

- UserRegistration - класс содержит методы RegisterUser() для регистрации новых пользователей и ValidateUser() для проверки данных пользователя при регистрации. Это позволяет учесть важную часть функциональности регистрации пользователей в мессенджере.

Диаграмма ER:

Сущности:

User (Пользователь)

ID: Уникальный идентификатор пользователя (Primary Key, int).

Name: Имя пользователя (string).

Photo: Фотография пользователя (string).

Status: Статус пользователя (string).

Group (Группа)

ID: Уникальный идентификатор группы (Primary Key, int).

Name: Название группы (string).

Message (Сообщение)

ID: Уникальный идентификатор сообщения (Primary Key, int).

Text: Текст сообщения (string).

Date: Дата и время отправки сообщения (datetime).

Administrator (Администратор)

ID: Уникальный идентификатор администратора (Primary Key, int).

Name: Имя администратора (string).

Photo: Фотография администратора (string).

UserRegistration (Регистрация пользователя)

ID: Уникальный идентификатор записи регистрации (Primary Key, int).

Email: Электронная почта пользователя (string).

Password: Пароль пользователя (string).

Связи:

Связь между User и Message:

Один пользователь может отправлять и получать много сообщений, поэтому это отношение 1 к многим.

Связь показывает, что один пользователь может иметь много сообщений, но каждое сообщение принадлежит только одному пользователю.

Связь между User и Group:

Один пользователь может быть членом нескольких групп, поэтому это отношение также 1 к многим.

Связь показывает, что один пользователь может быть членом нескольких групп, но каждая группа имеет много пользователей.

Связь между User и Administrator:

Один пользователь может быть связан с несколькими группами в роли администратора, и каждая группа может иметь нескольких администраторов. Это отношение многих ко многим.

Связь между UserRegistration и User:

UserRegistration связана с User для представления процесса регистрации. Каждая запись UserRegistration может быть связана с одним пользователем (после успешной регистрации), но также может быть не связана с пользователем (если регистрация не завершилась).

Используя метод персон (не менее 3), описать каких функций не хватает.

Пользователь Андрей: ""Мне было бы удобно иметь возможность пересылать сообщения из одного чата или группы в другой. Сейчас я вынужден копировать и вставлять текст вручную, что не всегда удобно."

Пользователь Валерия: "Я не знаю, когда мои сообщения были прочитаны другими пользователями. Мне бы хотелось видеть информацию о прочтении, как в других мессенджерах."

Пользователь Светлана: "Иногда мне нужно быстро найти старые сообщения или информацию в чатах. Но у вас нет функции поиска сообщений."

Пересмотрим нашу IML:

Добавим метод SearchMessages() в LogicLayer который позволяет пользователю искать сообщения в чатах и группах, также добавим метод ForwardMessage() в класс LogicLayer для пересылки сообщений.

Добавим метод Read Receipts() в классы Message и UserUi.

Пересмотрим нашу ER диаграмму:

Добавим атрибут ReadStatus сущности Message: ReadStatus (string)

- который будет указывать статус прочтения сообщения.

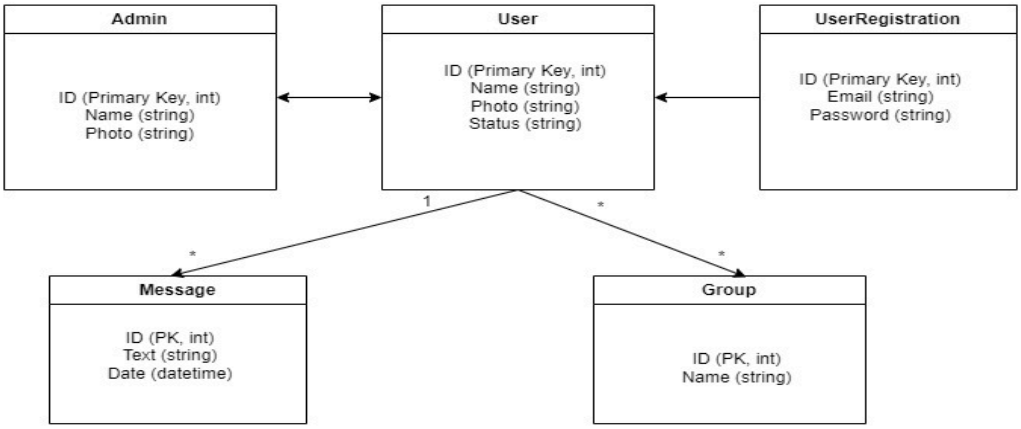
Добавим атрибут ForwardedMessageID к сущности Message:ForwardedMessageID (int, Foreign Key to Message.ID)

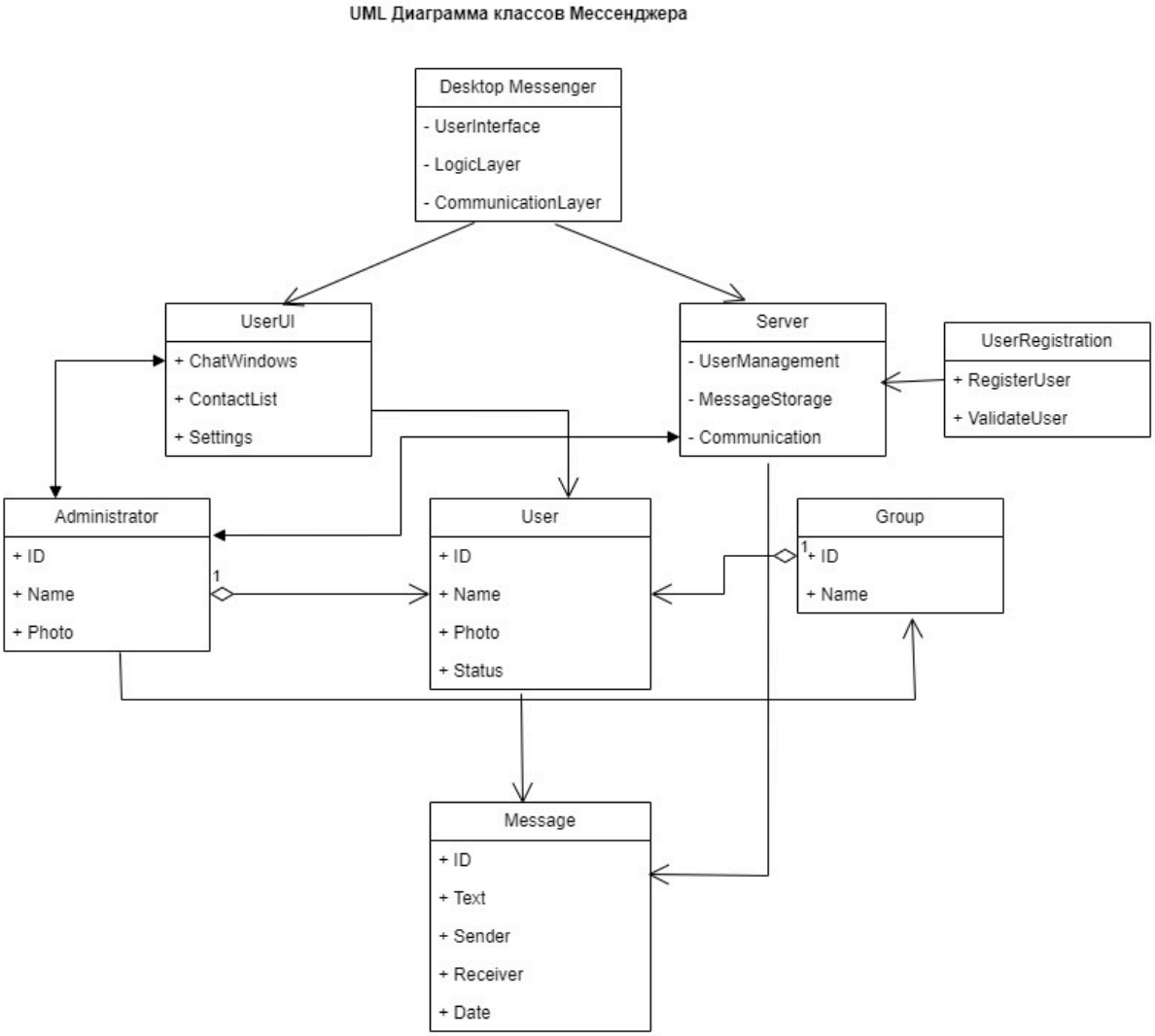
- который будет содержать идентификатор сообщения, которое было переслано.

Дизайн взял как пример того как может выглядеть интерфейс декстоп приложения мессенджера.

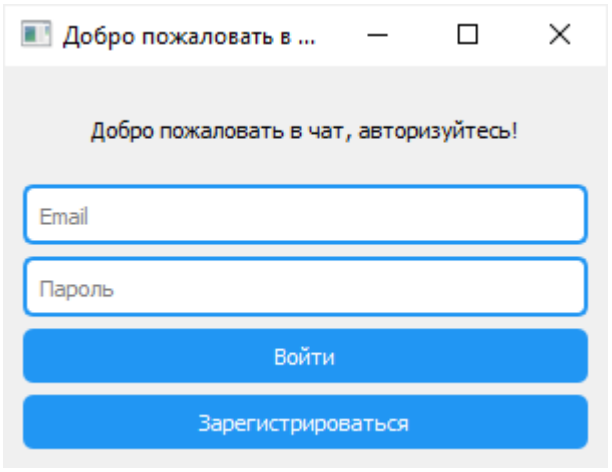
Создана ER и UML диаграммы

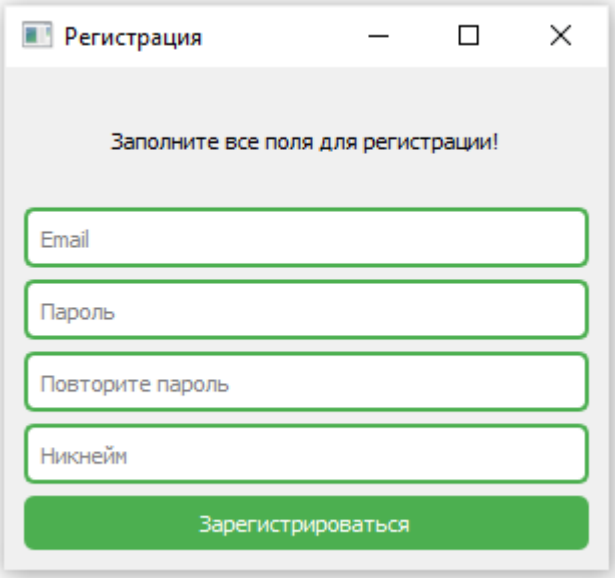
ER диаграмма Мессенджера

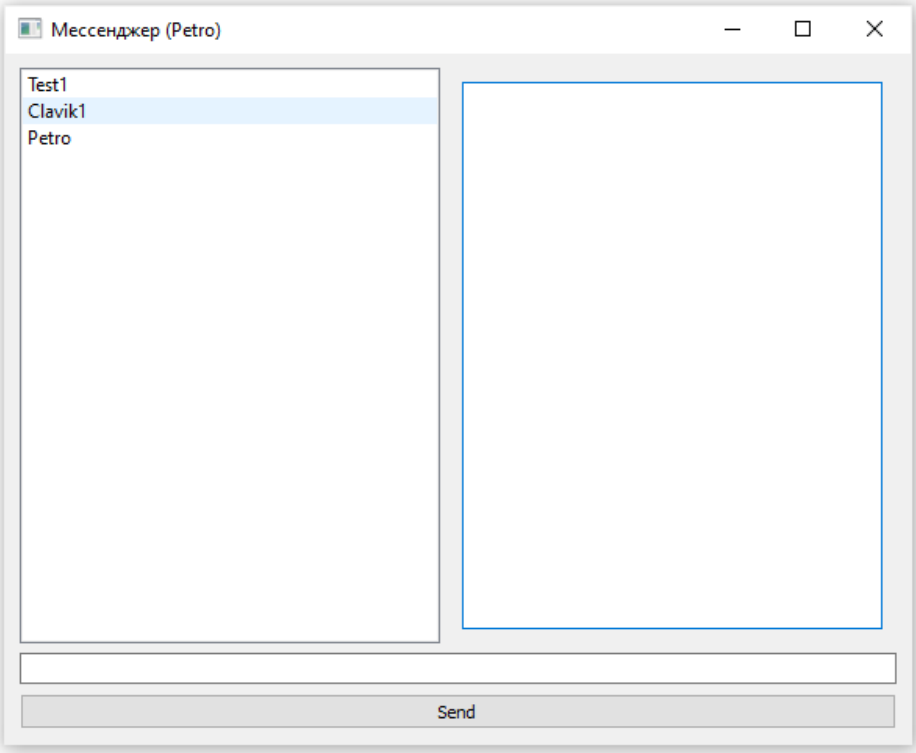




Разработал пока дизайн формы регистрации, аутентификации, главное окно.


Добро пожаловать в ...
Добро пожаловать в чат, авторизуйтесь!
Email
Пароль
Войти
Зарегистрироваться


Регистрация
Заполните все поля для регистрации!
Email
Пароль
Повторите пароль
Никнейм
Зарегистрироваться


Мессенджер (Petro)
Test1
Clavik1
Petro
Send

Далее была разработана логика состоящая из следующих модулей:

database.py

main.py

messenger.py

registration_and_authentication.py

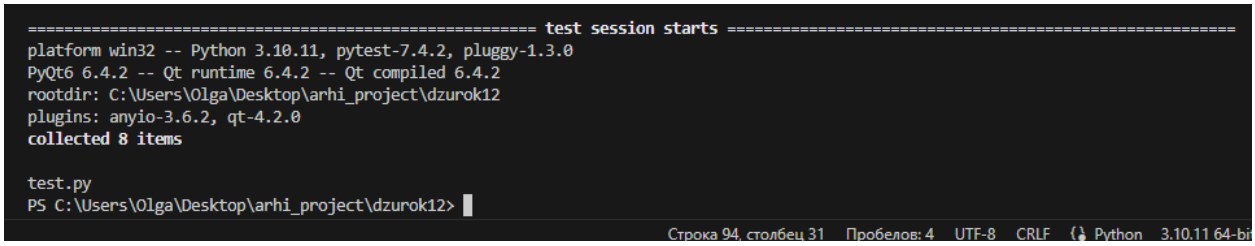
user_auth.py

messenger.db

Далее было запущено тестирование на Python с помощью библиотеке Pytest:

Был создан файл test.py с импортом модулей из файлов программы, далее я написал код для теста разных ситуаций. Тесты проедены успешно, ошибок нет.

Запустил тест с помощью команды Pytest test.py



Но искусственно создал ошибку убрав import модуля os в messenger.py

И получил ошибку в тесте