**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,**

**СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ СЕТЕЙ И СИСТЕМ (ИКСС)**

**КАФЕДРА ЗАЩИЩЕННЫХ СИСТЕМ СВЯЗИ (ЗСС)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Курсовая работа по предмету:

«Разработка защищённых сетевых приложений»

на тему:

«Разработка приложения поиска исполняемых файлов по сигнатуре»

Выполнил студент группы ИБС-23:

Такан Дмитрий Гарипович

Проверил: Цветков Александр Юрьевич

Санкт-Петербург,

2023

**СОДЕРЖАНИЕ**

1. [Введение](file:///C:\Users\Admin\Desktop\dz3.docx#_bookmark0) [3](file:///C:\Users\Admin\Desktop\dz3.docx#_bookmark0)

2. [Описание](file:///C:\Users\Admin\Desktop\dz3.docx#_bookmark7) работы программы [3](file:///C:\Users\Admin\Desktop\dz3.docx#_bookmark7)

3. Описание используемых библиотек [3](file:///C:\Users\Admin\Desktop\dz3.docx#_bookmark8)

4. Пример работы программы [12](file:///C:\Users\Admin\Desktop\dz3.docx#_bookmark8)

5. Листинг программы [19](file:///C:\Users\Admin\Desktop\dz3.docx#_bookmark8)

6. Список литературы [23](file:///C:\Users\Admin\Desktop\dz3.docx#_bookmark8)

**Введение**

Курсовая работа посвящена разработке приложения, которое осуществляет поиск исполняемых файлов по указанной сигнатуре. Основная цель работы заключается в создании полнофункциональной программы с удобным пользовательским интерфейсом, который позволит не только пополнять базу данных сигнатур, указывать путь для поиска файлов, но также детально отображать информацию о найденных файлах и их метаданных. Благодаря этому, пользователь сможет легко и удобно управлять приложением, проводить более глубокий анализ найденных файлов и получать более полную информацию о них. В конечном итоге, разработанное приложение будет предоставлять более широкий функционал и представлять собой полноценный инструмент для работы с исполняемыми файлами и их сигнатурами.

Ниже представлено описание работы программы с шагами

Программа разработана на языке Java и использует следующие библиотеки:

java.nio.file.attribute: Используется для работы с атрибутами файлов, такими как время создания, последнего доступа и модификации.

java.nio.file: Предоставляет возможности для работы с файлами и директориями, включая чтение и запись данных.

java.nio.file.FileSystems: Позволяет создавать объекты, представляющие файловую систему.

javax.swing: Используется для создания графического интерфейса пользователя (GUI).

java.awt: Используется для расположения и управления компонентами GUI.

java.awt.event: Используется для обработки событий в интерфейсе.

java.io: Используется для работы с потоками ввода-вывода.

java.security: Используется для работы с криптографическими алгоритмами и функциями хеширования.

java.sql: Используется для соединения с базой данных и выполнения SQL-запросов.

Программа состоит из класса Main, который наследуется от класса JFrame, представляющего окно приложения. Программа имеет следующие основные компоненты:

JTextArea resultArea: Область текста, используемая для вывода результатов поиска и информации о файлах.

JTextField executableFilePathField: Поле ввода, предназначенное для указания пути к исполняемому файлу.

JTextField hashExistsField: Поле вывода, используемое для отображения информации о наличии хэша в базе данных.

JComboBox<String> hashSelectionComboBox: Комбинированный список, предназначенный для выбора хэша из базы данных.

JButton calculateHashButton: Кнопка, запускающая вычисление и сравнение хэша выбранного файла.

JButton addSignatureButton: Кнопка, позволяющая добавить хэш в базу данных.

Различные панели и компоненты для организации интерфейса.

Программа выполняет следующие шаги:

Устанавливает соединение с базой данных, используя JDBC.

Загружает существующие хэши из базы данных и отображает их в комбинированном списке.

При нажатии на кнопку Calculate and Compare Hash программа осуществляет вычисление хэша выбранного файла и сравнивает его с выбранным хэшем из комбинированного списка. Результаты выводятся в JTextArea resultArea.

Если хэши совпадают, программа выводит информацию о найденном файле, включая его путь и метаданные (время создания, последнего доступа и модификации).

Проверяет, существует ли хэш в базе данных, и выводит соответствующее сообщение в JTextField hashExistsField.

При нажатии на кнопку Add Signature to Database программа добавляет хэш выбранного файла в базу данных, если его нет в базе данных. В противном случае выводится сообщение о том, что хэш уже существует в базе данных.

**Описание используемых библиотек**

**java.io:** библиотека java.io предоставляет обширный набор классов и интерфейсов для эффективной работы с потоками ввода и вывода, а также для манипуляции файлами в языке программирования Java. Она является частью стандартной библиотеки Java и широко используется в различных приложениях для обработки данных, сохранения и загрузки файлов, а также для взаимодействия с различными устройствами ввода/вывода.

Одним из ключевых компонентов библиотеки java.io являются классы `FileInputStream` и `FileOutputStream`. `FileInputStream` предоставляет возможность читать данные из файла, а `FileOutputStream` — записывать данные в файл. Эти классы позволяют эффективно осуществлять операции ввода и вывода с использованием файлового ресурса.

Применение `FileInputStream` позволяет создавать поток для чтения данных из файла. Последовательное чтение байтов из файла обеспечивает программе доступ к содержимому файла в удобном для обработки формате. С другой стороны, `FileOutputStream` предоставляет возможность записи данных в файл. Этот процесс осуществляется последовательной записью байтов в файл, что позволяет сохранять информацию для последующего использования.

Благодаря классам `FileInputStream` и `FileOutputStream`, библиотека java.io обеспечивает мощные инструменты для работы с файлами, что делает ее незаменимой при разработке приложений, где требуется обработка файлов и работа с потоками ввода/вывода.

Кроме FileInputStream и FileOutputStream, о библиотеке java.io следует сказать, что она также предлагает новые и более удобные альтернативы для работы с файлами и потоками данных начиная с Java 7. На верхнем уровне абстракции, появились классы Files и Paths, которые расширяют возможности работы с файлами и директориями, предоставляя удобные методы для копирования, перемещения, удаления файлов и манипуляции путями к файлам.

С классом Files возникает возможность выполнять разнообразные операции над содержимым файлов и директорий без явной работы с потоками ввода/вывода. Он предоставляет удобные методы для чтения и записи файлов, что устраняет необходимость явной работы с потоками, вроде FileInputStream и FileOutputStream. А класс Paths упрощает работу с файловыми путями, предоставляя методы для создания объектов Path — представления пути к файлу или директории в файловой системе.

Интересно заметить, что начиная с Java 7, в библиотеке появились классы, которые автоматически закрывают ресурсы после использования. Классы FileInputStream и FileOutputStream требуют явного закрытия после завершения работы с ними, чтобы избежать утечек ресурсов. Однако в Java 7 появились классы try-with-resources, такие как BufferedReader, BufferedWriter, Scanner и другие, которые автоматически закрывают соответствующие ресурсы. Это улучшает безопасность и удобство работы с потоками ввода/вывода.

Таким образом, библиотека java.io не только предоставляет базовые средства для работы с файлами и потоками, но и постоянно развивается, предлагая удобные и безопасные альтернативы для эффективной работы с файловой системой в языке Java.

**javax.swing**: библиотека `javax.swing` предоставляет набор инструментов для создания графического пользовательского интерфейса (GUI) в языке программирования Java. Она является частью более широкого пакета Java GUI (графического интерфейса пользователя) и предоставляет компоненты, которые облегчают создание интерактивных и интуитивно понятных пользовательских интерфейсов.

Одним из ключевых элементов библиотеки `javax.swing`, использованном в описанной программе, является класс `JFileChooser`. Этот класс предоставляет диалоговое окно для выбора файлов и директорий, что делает взаимодействие с файловой системой более удобным и интуитивным для пользователя. `JFileChooser` поддерживает различные режимы, включая выбор файла, выбор директории и другие, что обеспечивает гибкость в использовании в различных сценариях приложения.

Для работы с `JFileChooser` разработчику необходимо создать экземпляр этого класса и использовать его методы для настройки и взаимодействия с диалоговым окном. После выбора файла или директории пользователем, информация о выбранном пути может быть использована в дальнейших операциях программы, таких как копирование файла.

Библиотека javax.swing представляет собой мощный инструментарий для создания графического пользовательского интерфейса (GUI) в Java. Она включает в себя широкий набор компонентов, начиная от простых кнопок и текстовых полей, и заканчивая сложными компонентами, такими как таблицы, деревья и вкладки.

Когда речь заходит о работе с файловой системой в контексте GUI, класс JFileChooser действительно становится настоящим спасателем. Этот компонент предоставляет удобное и гибкое диалоговое окно для выбора файлов и директорий. Он позволяет пользователям легко выбирать и искать файлы на своем компьютере, тем самым улучшая пользовательский опыт.

Чтобы использовать JFileChooser, разработчику необходимо создать экземпляр этого класса и настроить его поведение, включая определение начальной директории, установку фильтров файлов и многое другое. После того как пользователь выберет файл или директорию, приложение может использовать информацию о выбранном пути для выполнения различных операций, таких как чтение данных из файла, запись в файл, или другие манипуляции с файловой системой.

Таким образом, библиотека `javax.swing` играет ключевую роль в создании удобного и интуитивно понятного пользовательского интерфейса, а `JFileChooser` предоставляет удобные средства для взаимодействия с файловой системой, улучшая опыт пользователя при работе с файлами и директориями в приложении. Эти компоненты совместно обеспечивают эффективное и удобное взаимодействие пользователя с приложением, что является важным аспектом в современной разработке программного обеспечения.

**java.nio:** Библиотека Java NIO (New I/O, новый ввод/вывод) представляет собой пакет для работы с низкоуровневыми операциями ввода и вывода, предоставляя альтернативные средства для работы с файлами, сетями и прочими потоками данных. Она была представлена в Java 1.4 и добавляет новые возможности для работы с вводом/выводом, которые не были доступны в классической библиотеке java.io.

Основными компонентами библиотеки Java NIO являются каналы, буферы и селекторы.

1. \*\*Каналы (Channels):\*\* Каналы представляют собой абстракцию, предназначенную для связи между источником данных (например, файлом) и местом назначения (например, программой). Они могут быть использованы для чтения, записи или дублирования данных.

2. \*\*Буферы (Buffers):\*\* Буферы представляют собой хранилище для данных в процессе их обработки, обеспечивая более эффективную работу с памятью и увеличивая производительность при обмене данными.

3. \*\*Селекторы (Selectors):\*\* Селекторы предоставляют возможность мультиплексирования каналов, т.е. управление несколькими каналами в одном потоке. Это особенно полезно для сетевого ввода/вывода, когда сервер должен обрабатывать одновременно большое количество соединений.

Преимущества Java NIO включают в себя более эффективное использование ресурсов, возможность неблокирующего ввода/вывода (non-blocking I/O), когда поток не блокируется при ожидании завершения операции ввода/вывода, а также возможность работы с каналами и селекторами, позволяющая эффективно обрабатывать множество соединений одновременно.

**java.awt:** awt (Abstract Window Toolkit) представляет собой стандартную библиотеку для создания и управления графическим пользовательским интерфейсом (GUI) в языке Java. Она была одной из первых библиотек GUI в Java и включает в себя широкий набор классов и интерфейсов для создания оконных приложений, рисования на экране, обработки событий, а также создания различных элементов интерфейса, такие как кнопки, текстовые поля, меню, и многое другое.

Основные компоненты java.awt включают:

1. \*\*Компоненты интерфейса (Components):\*\* Классы, такие как Button, TextField, Label, Checkbox и другие, представляют различные элементы для построения пользовательского интерфейса.

2. \*\*Макеты (Layouts):\*\* java.awt также предоставляет различные макеты для управления размещением компонентов в окне, включая FlowLayout, BorderLayout, GridLayout и другие, обеспечивая гибкость в расположении элементов интерфейса.

3. \*\*Графика (Graphics):\*\* Библиотека также включает классы для рисования на экране, такие как Graphics, Graphics2D, что позволяет программистам создавать различные формы, изображения и анимации.

4. \*\*События (Events):\*\* java.awt обеспечивает механизм для обработки пользовательских событий, таких как клики мыши, нажатия клавиш и других действий пользователя.

**java.security**: Библиотека java.security представляет собой одну из ключевых частей Java, она охватывает различные аспекты безопасности виртуальной машины Java (JVM), приложений и сетевых операций. Эта библиотека предоставляет средства для аутентификации, авторизации, шифрования, цифровой подписи, управления ключами, генерации случайных чисел и других компонентов безопасности.

Библиотека java.security представляет собой одну из ключевых частей Java, охватывающую различные аспекты безопасности виртуальной машины Java (JVM), приложений и сетевых операций. Она предоставляет средства для аутентификации, авторизации, шифрования, цифровой подписи, управления ключами, генерации случайных чисел и других компонентов безопасности.

Эта библиотека включает:

- Криптографию: классы и интерфейсы для работы с криптографическими операциями, включая шифрование, дешифрование, подписание и проверку цифровых подписей, а также вычисление хэш-значений.

- Управление ключами: классы для генерации, хранения и управления криптографическими ключами.

- Безопасность среды выполнения: механизмы безопасности JVM, такие как политики безопасности, контроль доступа и аутентификацию.

- Сетевую безопасность: классы для SSL/TLS, аутентификацию и авторизацию в сетевых приложениях.

**java.sql:** Библиотека java.sql предназначена для работы с реляционными базами данных в языке программирования Java. Она содержит классы и интерфейсы, позволяющие установить соединение с базой данных, выполнить запросы к ней, получить и изменить данные.

Основным классом библиотеки является java.sql.DriverManager, который позволяет установить соединение с базой данных, указав URL базы данных, логин и пароль.

Для выполнения запросов к базе данных используется интерфейс java.sql.Statement. Результат запроса сохраняется в объекте ResultSet, который можно использовать для получения данных.

Для выполнения запросов с параметрами можно использовать интерфейс java.sql.PreparedStatement. Значения параметров передаются методом setInt или другими подобными методами.

Для выполнения транзакций можно использовать интерфейс java.sql.Connection. Транзакция состоит из нескольких операций, которые могут быть выполнены успешно или не успешно. Если все операции выполняются успешно, то вызывается метод commit(). В случае ошибки вызывается метод rollback(), который отменяет все изменения.

Библиотека java.sql также содержит множество других классов и интерфейсов для работы с базами данных, таких как java.sql.ResultSetMetaData, java.sql.DatabaseMetaData, java.sql.SQLException и другие. Она позволяет создавать мощные приложения для работы с данными в языке программирования Java.

**Пример работы программы**

При запуске программы пользователь сразу же видит интуитивно понятное диалоговое окно, которое было реализовано с использованием библиотеки javax.swing. Это окно создает комфортное впечатление и позволяет пользователю легко взаимодействовать с программой. Благодаря этому диалоговому окну, пользователь может без труда находить нужные функции и выполнять необходимые действия. Это значительно улучшает пользовательский опыт и повышает эффективность работы с программой. Таким образом, использование библиотеки javax.swing для создания интуитивно-понятного диалогового окна является одним из ключевых аспектов данной программы.

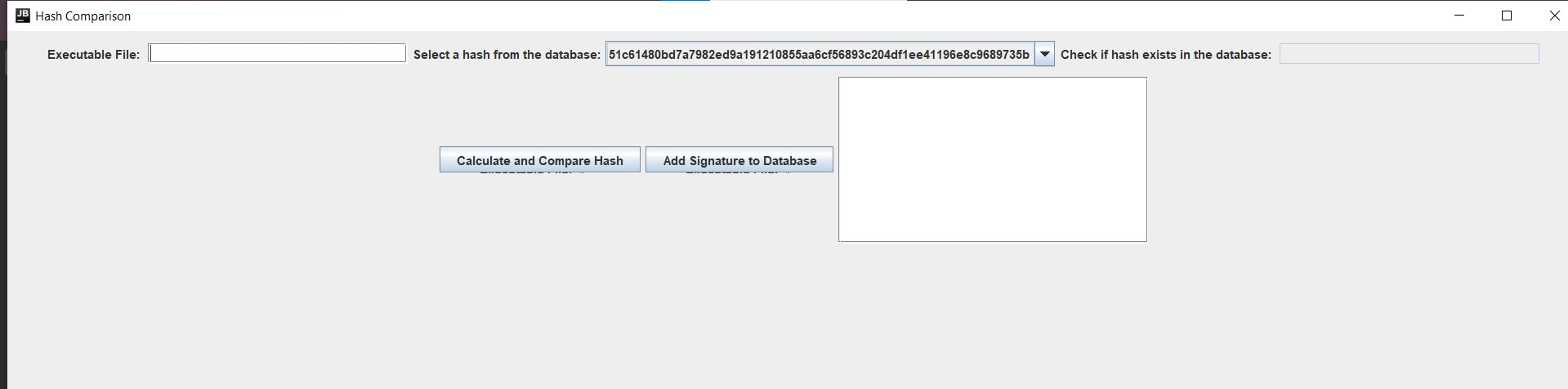


Рис. 1. Начальное окно

Перед нами расположены несколько функциональных кнопок, предназначенных для различных действий, а также удобное поле для ввода пути к исполняемому файлу. Важно отметить, что база данных уже содержит несколько сигнатур в качестве примера, что может быть полезно для ознакомления и понимания функциональности системы.

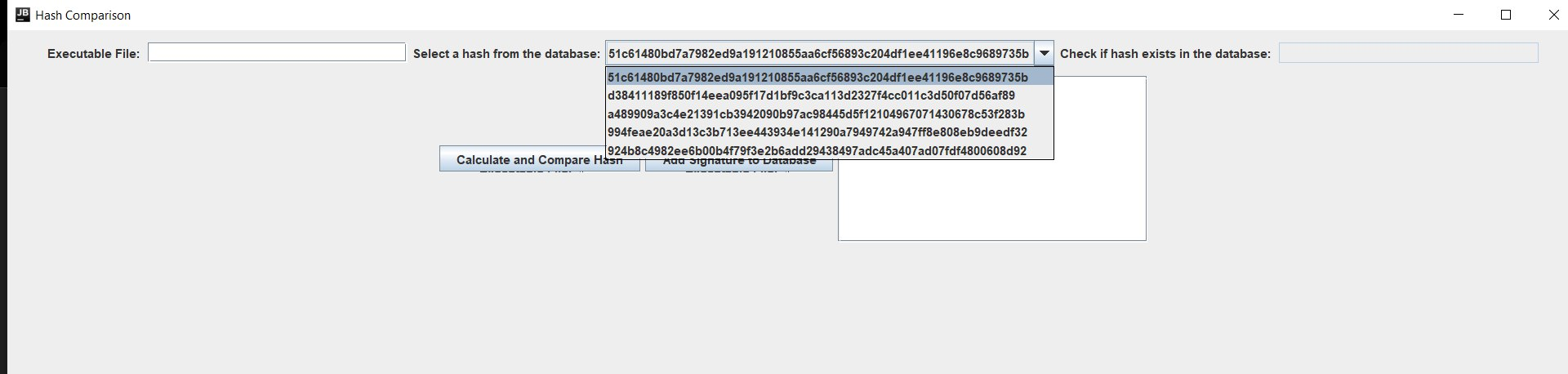
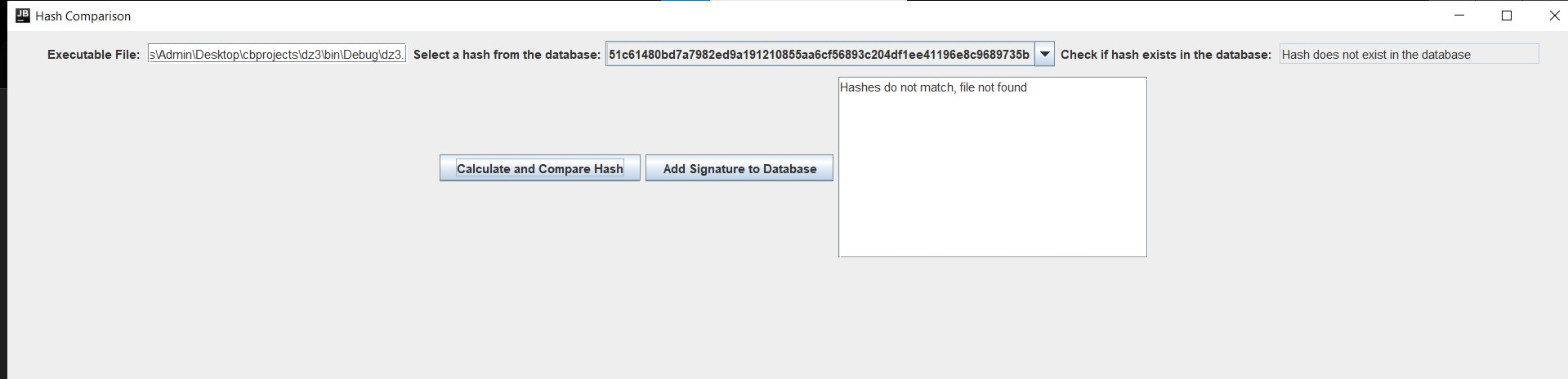


Рис. 2. Сигнатуры

Вводим путь к файлу, выбираем любую сигнатуру из предложенного списка и нажимаем кнопку "Calculate and Compare Hash". Затем, программное обеспечение проводит вычисление и сравнивает полученные хэш-значения для проверки целостности файла. Этот процесс позволяет убедиться, что файл не был изменен или поврежден в процессе передачи или хранения данных.

Рис. 3. Проверка первого файла

В результате работы программы возникает сообщение о том, что сигнатура данного исполняемого файла не соответствует сигнатуре из списка, в результате чего файл не может быть найден. Кроме того, программа сообщает о том, что сигнатура этого файла отсутствует в базе данных. Для решения этой проблемы предлагаю добавить сигнатуру в базу данных, нажав на кнопку "Add Signature to Database".

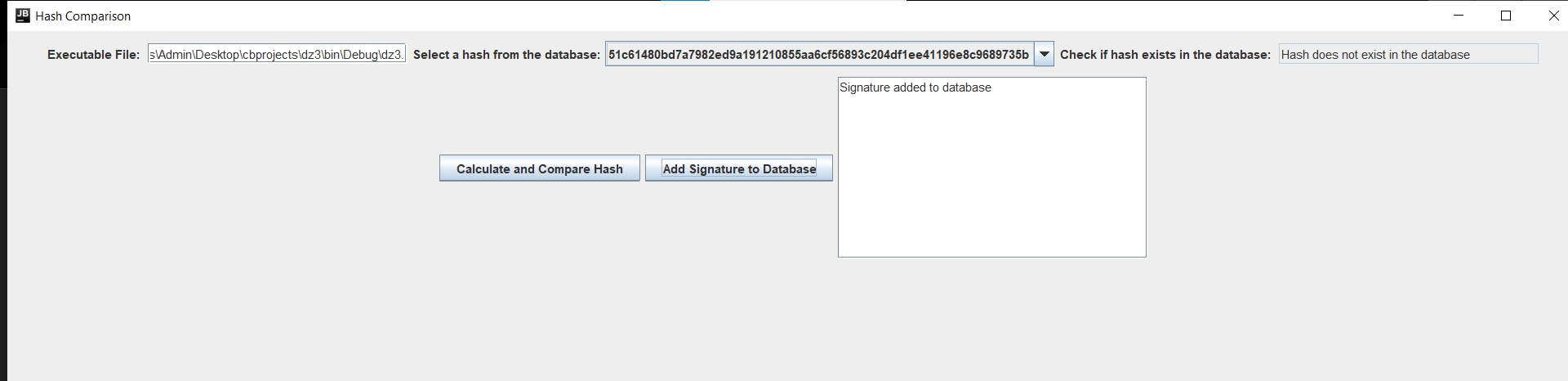
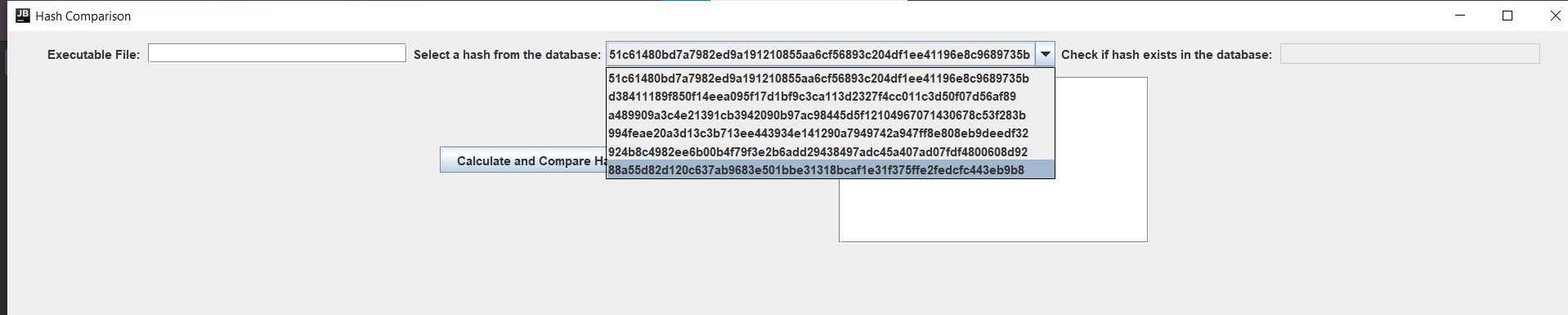


Рис. 4. Добавление сигнатуры в базу данных

 Рис. 5. Отображение новой сигнатуры

Как мы видим, сигнатура успешно добавилась в систему и теперь включена в список допустимых файлов. Теперь давайте проведем повторную проверку, чтобы убедиться, что файл успешно добавлен и доступен для использования.

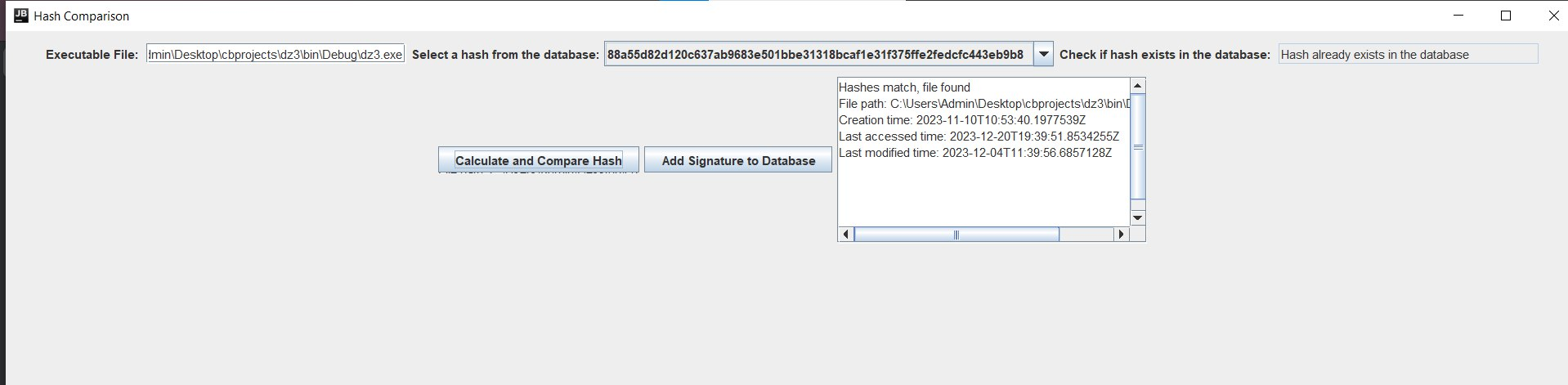


Рис. 6. Успешное выполнение программы

Программа успешно смогла найти полный путь к этому файлу и извлечь его метаданные. Она также предоставила дополнительные сведения о файле, включая его размер, дату создания и последнего изменения. Это позволяет пользователям быстро и легко получать полную информацию о файле, необходимую для его использования.

Тем не менее, иногда программа может столкнуться с трудностями и не сможет правильно выполнить свои задачи. Вот пример неудачной работы программы: она не смогла найти указанный файл, что привело к ошибке и невозможности получить необходимую информацию. В таких случаях важно обратиться к службе поддержки или попробовать другие способы решения проблемы.

Несмотря на то, что программа может иногда сталкиваться с трудностями, ее функциональность и возможности все равно делают ее полезным инструментом для работы с файлами и их метаданными.



Рис. 6. Обработка ошибок (неверная сигнатура)

 Рис. 6. Обработка ошибок (добавление сигнатуры, которая уже есть в базе)



Рис. 6. Обработка ошибок (неверный путь для файла)

Важно отметить, что весь этот процесс взаимодействия осуществляется в рамках удобного и интуитивно-понятного пользовательского интерфейса, который обеспечивает легкость навигации, точность выбора файлов и надежность выполнения операций. Этот дизайн интерфейса направлен на обеспечение комфортного пользовательского опыта, где каждый шаг является четким и понятным взаимодействием между пользователем и программой.

Кроме того, пользовательский интерфейс также предоставляет множество дополнительных функций и возможностей, которые обогащают опыт использования программы. Например, есть функция автоматического сохранения данных, которая позволяет избежать потери информации в случае сбоя или непредвиденных ситуаций. Также есть интегрированный поиск, который помогает быстро находить нужные файлы или документы. Дополнительно, пользователь может настроить интерфейс под свои предпочтения, выбрав цветовую схему или расположение элементов.

В целом, этот пользовательский интерфейс создан с учетом потребностей и ожиданий пользователей, обеспечивая удобство, надежность и функциональность взаимодействия с программой.

Далее перейдем к комментированию кода

import java.nio.file.attribute.BasicFileAttributes; - Импорт класса BasicFileAttributes из пакета java.nio.file.attribute

import java.nio.file.attribute.FileTime; - Импорт класса FileTime из пакета java.nio.file.attribute

import java.nio.file.FileSystems; - Импорт класса FileSystems из пакета java.nio.file

import javax.swing.\*; - Импорт всех классов из пакета javax.swing

import java.awt.\*; - Импорт всех классов из пакета java.awt

import java.awt.event.\*; - Импорт всех классов из пакета java.awt.event

import java.io.\*; - Импорт всех классов из пакета java.io

import java.nio.file.\*; - Импорт всех классов из пакета java.nio.file

import java.security.\*; - Импорт всех классов из пакета java.security

import java.sql.\*; - Импорт всех классов из пакета java.sql

public class Main extends JFrame { - Объявление класса Main, который расширяет класс JFrame

private final JTextArea resultArea; - Объявление переменной resultArea (текстовая область)

private final JTextField executableFilePathField; - Объявление переменной executableFilePathField (поле для ввода пути к исполняемому файлу)

private final JTextField hashExistsField; - Объявление переменной hashExistsField (поле для проверки наличия хэша)

private final JComboBox<String> hashSelectionComboBox; - Объявление переменной hashSelectionComboBox (выпадающий список для выбора хеша)

private Connection connection; - Объявление переменной connection (соединение с базой данных)

public Main() { - Объявление конструктора класса Main

try { - Блок обработки исключений

String url = "jdbc:mysql://localhost:3306/signas"; - Строка подключения к базе данных MySQL

String user = "lucky"; - Имя пользователя для подключения к базе данных

String password = "DEMON28781d"; - Пароль пользователя для подключения к базе данных

connection = DriverManager.getConnection(url, user, password); - Установка соединения с базой данных

catch (SQLException e) { - Обработка исключения SQLException

e.printStackTrace(); - Вывод стека вызовов и информации об исключении

setTitle("Hash Comparison"); - Установка заголовка окна

setDefaultCloseOperation(EXIT\_ON\_CLOSE); - Установка операции по закрытию окна по умолчанию

setSize(400, 400); - Установка размеров окна

setLayout(new FlowLayout()); - Установка менеджера компоновки для окна

JButton calculateHashButton = new JButton("Calculate and Compare Hash"); - Создание кнопки "Calculate and Compare Hash"

JButton addSignatureButton = new JButton("Add Signature to Database"); - Создание кнопки "Add Signature to Database"

resultArea = new JTextArea(10, 30); - Создание текстовой области с размерами 10 строк на 30 столбцов

resultArea.setEditable(false); - Установка возможности редактирования текстовой области в значение "false"

JPanel panel = new JPanel(); - Создание панели

executableFilePathField = new JTextField(25); - Создание поля для ввода пути к исполняемому файлу с размером 25 символов

panel.add(new JLabel("Executable File: ")); - Добавление метки "Executable File" на панель

panel.add(executableFilePathField); - Добавление поля для ввода пути к исполняемому файлу на панель

hashSelectionComboBox = new JComboBox<>(); - Создание выпадающего списка для выбора хеша

try { // Блок обработки исключений

loadHashesFromDatabase(); - Загрузка списка хешей из базы данных

panel.add(new JLabel("Select a hash from the database:")); - Добавление метки "Select a hash from the database" на панель

panel.add(hashSelectionComboBox); - Добавление выпадающего списка на панель

catch (SQLException ex) { - Обработка исключения SQLException

ex.printStackTrace(); - Вывод стека вызовов и информации об исключении

hashExistsField = new JTextField(25); // Создание поля для проверки наличия хэша с размером 25 символов

hashExistsField.setEditable(false); // Установка возможности редактирования поля в значение "false"

panel.add(new JLabel("Check if hash exists in the database: ")); // Добавление метки "Check if hash exists in the database:" на панель

panel.add(hashExistsField); // Добавление поля для проверки наличия хэша на панель

add(panel); // Добавление панели на окно

add(calculateHashButton); // Добавление кнопки "Calculate and Compare Hash" на окно

add(addSignatureButton); // Добавление кнопки "Add Signature to Database" на окно

add(new JScrollPane(resultArea)); // Добавление текстовой области с полосой прокрутки на окно

calculateHashButton.addActionListener(new ActionListener() { // Добавление слушателя для кнопки "Calculate and Compare Hash"

[@Override](https://t.me/Override)

public void actionPerformed(ActionEvent e) { // Переопределение метода actionPerformed для обработки события нажатия кнопки

String executableFilePath = executableFilePathField.getText(); // Получение пути к исполняемому файлу из поля ввода

try { // Блок обработки исключений

String calculatedHash = calculateFileHash(executableFilePath); // Вычисление хэша исполняемого файла

String selectedHash = (String) hashSelectionComboBox.getSelectedItem(); // Получение выбранного хэша из выпадающего списка

boolean hashExists = isHashExistsInDatabase(calculatedHash); // Проверка наличия хэша в базе данных

if (calculatedHash.equals(selectedHash)) { // Проверка совпадения хэшей

resultArea.setText("Hashes match, file found"); // Установка текста в текстовой области

File file = new File(executableFilePath); // Создание объекта File на основе пути к файлу

resultArea.append("\nFile path: " + file.getAbsolutePath()); // Добавление информации о пути к файлу в текстовую область

BasicFileAttributes attr = Files.readAttributes(file.toPath(), BasicFileAttributes.class); // Получение атрибутов файла

resultArea.append("\nCreation time: " + attr.creationTime()); // Добавление информации о времени создания файла в текстовую область

resultArea.append("\nLast accessed time: " + attr.lastAccessTime()); // Добавление информации о последнем доступе к файлу в текстовую область

resultArea.append("\nLast modified time: " + attr.lastModifiedTime()); // Добавление информации о последнем изменении файла в текстовую область

} else { // Если хэши не совпадают

resultArea.setText("Hashes do not match, file not found"); // Установка текста в текстовой области

if (hashExists) { // Если хэш существует в базе данных

hashExistsField.setText("Hash already exists in the database"); // Установка текста в поле для проверки наличия хэша

} else { // Если хэш не существует в базе данных

hashExistsField.setText("Hash does not exist in the database"); // Установка текста в поле для проверки наличия хэша

catch (IOException | NoSuchAlgorithmException | SQLException ex) { // Обработка исключений

resultArea.setText("Error: " + ex.getMessage()); // Установка текста об ошибке в текстовой области

// Метод загрузки хешей из базы данных

private void loadHashesFromDatabase() throws SQLException {

// Начало блока try-with-resources, создание Statement через метод createStatement() объекта connection

try (Statement statement = connection.createStatement();

// Создание ResultSet путем выполнения запроса "select hash from hashes"

ResultSet resultSet = statement.executeQuery("select hash from hashes")) {

// Цикл прохода по результатам

while (resultSet.next()) {

// Получение хеша из результата и добавление его в элемент hashSelectionComboBox

String hash = resultSet.getString("hash");

hashSelectionComboBox.addItem(hash);

// Метод вычисления хеша файла

private String calculateFileHash(String filePath) throws NoSuchAlgorithmException, IOException {

// Создание экземпляра MessageDigest с использованием алгоритма SHA-256

MessageDigest digest = MessageDigest.getInstance("SHA-256");

// Начало блока try-with-resources, создание потока ввода для файла и связывание его с объектом дайджеста

try (InputStream is = Files.newInputStream(Paths.get(filePath)); DigestInputStream dis = new DigestInputStream(is, digest)) {

// Буфер для чтения файла

byte[] buffer = new byte[8192];

// Цикл чтения файла

while (dis.read(buffer) != -1)

// Получение результата вычисления хеша и возврат его в виде строки

byte[] hash = digest.digest();

return bytesToHex(hash);

}

// Метод проверки существования хеша в базе данных

private boolean isHashExistsInDatabase(String hash) throws SQLException {

// Начало блока try-with-resources, создание PreparedStatement для подсчета количества записей с соответствующим хешем

Try (PreparedStatement preparedStatement = connection.prepareStatement("select count(\*) from hashes where hash = ?")) {

// Установка значения хеша в подготовленный запрос

preparedStatement.setString(1, hash);

// Выполнение запроса и получение результата

try (ResultSet resultSet = preparedStatement.executeQuery()) {

// Проверка наличия результата и возврат true, если хеш существует, иначе - false

if(resultSet.next()) {

int count = resultSet.getInt(1);

return count > 0;

// Метод преобразования массива байт в шестнадцатеричное представление

private String bytesToHex(byte[] bytes) {

// Создание строки для хранения шестнадцатеричного представления

StringBuilder hexString = new StringBuilder(2 \* bytes.length);

// Проход по каждому байту и преобразование его в шестнадцатеричную строку

for (byte b : bytes) {

String hex = Integer.toHexString(0xff & b);

// Добавление нуля, если длина шестнадцатеричной строки равна 1

if (hex.length() == 1) {

hexString.append('0');

}

// Добавление шестнадцатеричной строки в общую строку

hexString.append(hex);

}

// Возврат строки с шестнадцатеричным представлением

return hexString.toString();

}

// Точка входа в программу

public static void main(String[] args) {

// Запуск GUI внутри потока обработки событий

SwingUtilities.invokeLater(() -> {

new Main().setVisible(true);

});

}

**Листинг программы**

import java.nio.file.attribute.BasicFileAttributes;  
import java.nio.file.attribute.FileTime;  
import java.nio.file.FileSystems;  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.io.\*;  
import java.nio.file.\*;  
import java.security.\*;  
import java.sql.\*;  
  
public class Main extends JFrame {  
 private final JTextArea resultArea;  
 private final JTextField executableFilePathField;  
 private final JTextField hashExistsField;  
 private final JComboBox<String> hashSelectionComboBox;  
 private Connection connection;  
  
 public Main() {  
 try {  
 String url = "jdbc:mysql://localhost:3306/signas";  
 String user = "lucky";  
 String password = "DEMON28781d";  
 connection = DriverManager.*getConnection*(url, user, password);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 setTitle("Hash Comparison");  
 setDefaultCloseOperation(EXIT\_ON\_CLOSE);  
 setSize(1720, 980);  
 setLayout(new FlowLayout());  
  
 JButton calculateHashButton = new JButton("Calculate and Compare Hash");  
 JButton addSignatureButton = new JButton("Add Signature to Database");  
 resultArea = new JTextArea(10, 30);  
 resultArea.setEditable(false);  
  
 JPanel panel = new JPanel();  
  
 executableFilePathField = new JTextField(25);  
 panel.add(new JLabel("Executable File: "));  
 panel.add(executableFilePathField);  
  
 hashSelectionComboBox = new JComboBox<>();  
 try {  
 loadHashesFromDatabase();  
 panel.add(new JLabel("Select a hash from the database:"));  
 panel.add(hashSelectionComboBox);  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
  
 hashExistsField = new JTextField(25);  
 hashExistsField.setEditable(false);  
 panel.add(new JLabel("Check if hash exists in the database: "));  
 panel.add(hashExistsField);  
  
 add(panel);  
 add(calculateHashButton);  
 add(addSignatureButton);  
 add(new JScrollPane(resultArea));  
  
 calculateHashButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String executableFilePath = executableFilePathField.getText();  
 try {  
 String calculatedHash = calculateFileHash(executableFilePath);  
 String selectedHash = (String) hashSelectionComboBox.getSelectedItem();  
 boolean hashExists = isHashExistsInDatabase(calculatedHash);  
 if (calculatedHash.equals(selectedHash)) {  
 resultArea.setText("Hashes match, file found");  
 File file = new File(executableFilePath);  
 resultArea.append("\nFile path: " + file.getAbsolutePath());  
 BasicFileAttributes attr = Files.readAttributes(file.toPath(), BasicFileAttributes.class);  
 resultArea.append("\nCreation time: " + attr.creationTime());  
 resultArea.append("\nLast accessed time: " + attr.lastAccessTime());  
 resultArea.append("\nLast modified time: " + attr.lastModifiedTime());  
 } else {  
 resultArea.setText("Hashes do not match, file not found");  
 }  
 if (hashExists) {  
 hashExistsField.setText("Hash already exists in the database");  
 } else {  
 hashExistsField.setText("Hash does not exist in the database");  
 }  
 } catch (IOException | NoSuchAlgorithmException | SQLException ex) {  
 resultArea.setText("Error: " + ex.getMessage());  
 }  
 }  
 });  
  
 addSignatureButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String executableFilePath = executableFilePathField.getText();  
 try {  
 String calculatedHash = calculateFileHash(executableFilePath);  
 if (!isHashExistsInDatabase(calculatedHash)) {  
 try (PreparedStatement preparedStatement = connection.prepareStatement("insert into hashes (hash) values (?)")) {  
 preparedStatement.setString(1, calculatedHash);  
 preparedStatement.executeUpdate();  
 resultArea.setText("Signature added to database");  
 }  
 } else {  
 resultArea.setText("Signature already exists in database");  
 }  
 } catch (IOException | NoSuchAlgorithmException | SQLException ex) {  
 resultArea.setText("Error: " + ex.getMessage());  
 }  
 }  
 });  
 }  
  
 private void loadHashesFromDatabase() throws SQLException {  
 try (Statement statement = connection.createStatement();  
 ResultSet resultSet = statement.executeQuery("select hash from hashes")) {  
 while (resultSet.next()) {  
 String hash = resultSet.getString("hash");  
 hashSelectionComboBox.addItem(hash);  
 }  
 }  
 }  
  
 private String calculateFileHash(String filePath) throws NoSuchAlgorithmException, IOException {  
 MessageDigest digest = MessageDigest.*getInstance*("SHA-256");  
 try (InputStream is = Files.*newInputStream*(Paths.get(filePath)); DigestInputStream dis = new DigestInputStream(is, digest)) {  
 byte[] buffer = new byte[8192];  
 while (dis.read(buffer) != -1) {  
 }  
 }  
 byte[] hash = digest.digest();  
 return bytesToHex(hash);  
 }  
  
 private boolean isHashExistsInDatabase(String hash) throws SQLException {  
 try (PreparedStatement preparedStatement = connection.prepareStatement("select count(\*) from hashes where hash = ?")) {  
 preparedStatement.setString(1, hash);  
 try (ResultSet resultSet = preparedStatement.executeQuery()) {  
 if(resultSet.next()) {  
 int count = resultSet.getInt(1);  
 return count > 0;  
 }  
 }  
 }  
 return false;  
 }  
  
 private String bytesToHex(byte[] bytes) {  
 StringBuilder hexString = new StringBuilder(2 \* bytes.length);  
 for (byte b : bytes) {  
 String hex = Integer.*toHexString*(0xff & b);  
 if (hex.length() == 1) {  
 hexString.append('0');  
 }  
 hexString.append(hex);  
 }  
 return hexString.toString();  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(() -> {  
 new Main().setVisible(true);  
 });  
 }  
}

Программа реализована на языке Java в среде Intelij IDEA Ultimate Edition.

**Заключение**

В процессе выполнения данной курсовой работы была разработана программа на языке программирования Java, которая осуществляет поиск исполняемого файла по его сигнатуре. Для удобства пользователей был добавлен графический интерфейс, реализованный с помощью библиотеки Swing. Кроме того, в проекте была использована база данных MySQL для хранения и управления информацией. Этот проект позволил нам более глубоко изучить эти темы и научиться работать с файлами.

**Список источников**

1. Работа с SQL: сайт. – URL: https://timeweb.cloud/tutorials/mysql/kak-ustanovit-mysql-na-windows (дата обращения: 20.12.2023)
2. Создание GUI с библиотекой swing: сайт. – URL: <https://docs.oracle.com/javase/tutorial/uiswing/> ( дата обращения: 20.12.2023)