# TEXT TO HANDWRITING CONVERTER

MINOR PROJECT REPORT

By

**VINEET BIHANI (RA2211003010685)**

**VIJAY MAKKAD (RA2211003010686)**

**AVEECK PANDEY (RA2211003010703)**

Under the guidance of   
Dr. ASHWINI S

*In partial fulfilment for the Course*

of

**21CSC203P – ADVANCED PROGRAMMING PRACTICE**

in **Department of Computing Technologies**



**FACULTY OF ENGINEERING AND TECHNOLOGY**

**SCHOOL OF COMPUTING**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR**

**NOVEMBER 2023**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(Under Section 3 of UGC Act, 1956)**

**BONAFIDE CERTIFICATE**

# Certified that this minor project report for the course 21CSC203P ADVANCED PROGRAMMING PRACTICE entitled in "Text To Handwriting Converter

" is the bonafide work of **VINEET BIHANI (RA2211003010685), VIJAY MAKKAD (RA2211003010686), AVEECK PANDEY (RA2211003010703)** who carried out the work under my supervision.

# SIGNATURE

Dr. Ashwini s

# Assistant Professor

**Department of Computing Technologies**

SRM Institute of Science and Technology

Kattankulathur

# ABSTRACT

The Text-to-Handwriting Converter developed in Java leverages the Swing library to create a user-friendly graphical interface for transforming typed text into a handwritten style. The application comprises two JTextAreas for input and output, respectively, providing users with an intuitive platform for text conversion. A "Convert" button triggers the transformation process, allowing users to witness the immediate change in the displayed output area.

Under the hood, the conversion algorithm manipulates font attributes using the java.awt package, specifically the AttributedString class. By customizing TextAttribute instances, such as FAMILY and SIZE, the converter achieves a handwriting-like appearance. In this example, a HashMap encapsulates these attributes, forming a distinct handwriting style. The font is then applied to the input text, creating an AttributedString object that represents the converted text.

The graphical nature of the Swing library ensures a seamless user experience, with JScrollPane components facilitating easy text input and output navigation. The modular design of the application adheres to Java's object-oriented principles, enhancing maintainability and extensibility. This Text-to-Handwriting Converter serves as an illustrative example of Java's capabilities in building graphical applications and showcases how font manipulation can be employed to transform text, providing a practical and interactive tool for users interested in simulating handwritten text from digital input.

# ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V. Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project.

We extend my gratitude to our **HoD Dr** **Pushpalatha M,** Professor & Head, Department of Computing Technologies.

We wish to express my sincere thanks to **Course Audit Professors Dr. Vadivu. G , Professor, Department of Data Science and Business Systems and Dr. Sasikala. E Professor, Department of Data Science and Business Systems** and **Course Coordinators** for their constant encouragement and support.

We are highly thankful to our my Course project Faculty **Dr. Ashwini S., Assistant Professor , C.Tech ,** for his/herassistance, timely suggestion and guidance throughout the duration of this course project.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAPTER NO** | | **CONTENTS** | **PAGE NO** |
| **1** | **INTRODUCTION** | | **6** |
|  | 1.1 Motivation | | **6** |
|  | 1.2 Objective | | **7** |
|  | 1.3 Problem Statement | | **8** |
|  | 1.4 Challenges | | **8** |
| **2** | **LITERATURE SURVEY** | | **10** |
| **3** | **REQUIREMENT ANALYSIS** | | **11** |
| **4** | **IMPLEMENTATION** | | **12** |
| **5** | **OUTPUT** | | **13** |
|  | **CODE** | | **14-21** |
| **7** | **CONCLUSION** | | **21** |
| **8** | **REFERENCES** | | **22** |

1. **INTRODUCTION**

The introduction sets the stage for the Text-to-Handwriting Converter developed in Java, highlighting its purpose, features, and the underlying technologies employed.

In an era dominated by digital communication, the fusion of technology and creativity has given rise to innovative applications that bridge the gap between the digital and analog worlds. One such application is the Text-to-Handwriting Converter, a Java-based solution designed to transform conventional typed text into a charming, handwritten style. This software leverages the power of the Swing library, a robust framework for building graphical user interfaces in Java.

The Text-to-Handwriting Converter provides users with a user-friendly interface, allowing them to effortlessly convert text with the click of a button. The application features input and output text areas, creating an intuitive environment for users to witness the fascinating transition from typed text to the visually appealing nuances of handwritten script.

This introduction explores the underlying technology that propels this conversion process. By delving into Java's rich ecosystem, specifically the java.awt package and the AttributedString class, the converter achieves its unique handwriting effect. Through an innovative combination of font attributes, the application breathes life into digital text, mimicking the aesthetic charm of handwritten notes and messages.

In essence, the Text-to-Handwriting Converter is not merely a software tool but a testament to the fusion of technology and creativity, demonstrating Java's prowess in crafting engaging graphical applications that add a touch of analog warmth to our digital interactions.

* 1. **Motivation**
* Expressive Communication: The Text-to-Handwriting Converter adds a touch of personalization and warmth to digital messages, mimicking the charm of handwritten notes.
* Bridging Digital and Analog: Motivated by a desire to bridge the gap between digital convenience and the individuality of handwritten communication.
* User-Friendly Interface: Designed with simplicity in mind, allowing users to convert text with a single click through an intuitive graphical interface.
* Java's Creative Capabilities: Showcases Java's versatility in crafting engaging graphical applications using Swing and java.awt.
* Nostalgia and Authenticity: Motivated by the idea that personal expression should not be confined to standard fonts, but rather reflect the unique nuances of one's handwriting.
* Innovation in Digital Interaction: A commitment to exploring creative possibilities within technology, encouraging developers to think innovatively about enhancing user experiences.
  1. **Objective**

The objective of the Text-to-Handwriting Converter project is to develop a user-friendly Java-based application that seamlessly converts typed text into a visually appealing handwritten style. The primary goals include:

Enhanced Personalization: Enable users to infuse a sense of individuality and expressiveness into their digital messages by converting plain text into a charming, handwritten format.

User-Friendly Interface: Provide an intuitive and accessible graphical interface, ensuring a seamless experience for users of varying technical backgrounds to easily input and convert text.

Bridging Digital and Analog Communication: Create a tool that bridges the convenience of digital communication with the nostalgia and authenticity associated with handwritten notes, fostering a more engaging and personalized form of expression.

Java Technology Showcase: Demonstrate the capabilities of Java, specifically utilizing the Swing library and java.awt package, to develop creative and visually appealing graphical applications.

Innovation in Digital Interactions: Encourage innovative thinking in the realm of digital communication tools, exploring new ways to add a touch of creativity and individuality to written messages.

Contribution to Open-Source Community: Consider making the project open-source, fostering collaboration and allowing developers to contribute to its improvement and customization

* 1. **Problem Statement**
* Impersonal Digital Communication: Lack of personalization in digital messages using standard fonts.
* Limited Handwriting-style Conversion Tools: Scarcity of user-friendly tools for converting digital text to a handwritten style.
* Challenges in Font Manipulation: Difficulty for developers in implementing font manipulation for creative expression.
* Limited Access to Handwriting Fonts: Inaccessibility to diverse and aesthetically pleasing handwriting fonts for customization.
* Underutilization of Java in Creative Applications: Java's capabilities not fully exploited for developing visually creative applications.
* Complex Interfaces for Text Customization: Users face difficulties in applying creative elements to text due to complex interfaces.
  1. **Challenges**
* Font Variety and Consistency: In the real world, ensuring a consistent and diverse set of handwriting fonts that cater to various preferences and cultural contexts poses a challenge. Users might have specific expectations regarding the appearance of handwritten text.
* Performance Concerns: As the length of the input text increases, rendering detailed handwriting styles in real-time may impact the performance of the application. Balancing aesthetics and responsiveness is crucial.
* Cross-Platform Compatibility: Achieving consistent performance and appearance across different operating systems and devices can be challenging, given variations in font rendering engines and graphical capabilities.
* Customization Complexity: Allowing users to customize the handwriting style extensively while maintaining a simple and intuitive user interface is a delicate balance. Striking the right balance between flexibility and usability is a significant challenge.
* Integration with Existing Software: Seamlessly integrating the Text-to-Handwriting Converter with other applications or platforms could be challenging. Compatibility issues might arise when users attempt to incorporate handwritten text into existing digital workflows.
* Accessibility Concerns: Ensuring that the converted handwritten text remains accessible to individuals with disabilities is a challenge. This includes considerations for screen readers and other assistive technologies.
* Scalability: The program's scalability concerning a large number of users concurrently accessing the conversion service may pose challenges in terms of server load, response times, and resource utilization.
* Security: Handling sensitive information within the application, especially if it involves online services or user accounts, requires robust security measures to protect user data and privacy.
* Legal and Copyright Considerations: Incorporating diverse handwriting styles may involve copyrighted material. Ensuring compliance with copyright laws and obtaining necessary permissions could be a challenge.
* User Education: Educating users on the limitations and possibilities of the Text-to-Handwriting Converter is crucial. Users might have unrealistic expectations, and managing these expectations through clear communication is a constant challenge

1. **LITERATURE SURVEY**

The literature survey demonstrates viewpoints, methodological solutions and research results related to the area. The existing information is critically analysed so that contradicting and differing research methods are shown. Only material that is relevant and directly related to the research is selected in the survey. A literature survey for the provided "Text to Handwriting converter" code involves examining related work and resources that address similar topics or technologies.

* Handwriting Synthesis:

Researchers might investigate methods for synthesizing realistic handwriting from text. This could involve studying individual writing styles and incorporating elements of variability to make the generated handwriting look more natural.

* Neural Networks and Deep Learning:

Deep learning techniques, such as recurrent neural networks (RNNs) and generative adversarial networks (GANs), may be employed to model and generate handwriting styles. These models can learn complex patterns and variations in handwriting.

* Style Transfer Techniques:

Style transfer involves transferring the style of one image or domain to another. In the context of text-to-handwriting conversion, researchers might explore techniques to transfer the style of existing handwritten samples to generated text.

* User Interface and User Experience (UI/UX) Studies:

Literature may include studies on the user interface and user experience aspects of text-to-handwriting converters. This involves understanding how users interact with such systems and optimizing the design for usability and efficiency.

* Applications in Education and Accessibility:

Researchers might explore the potential applications of text-to-handwriting conversion in educational settings, aiding students in learning to write or assisting individuals with disabilities.

* Natural Language Processing (NLP):

Natural Language Processing techniques could be used to enhance the understanding of the input text, allowing for more context-aware generation of handwritten content.

Integration with Handwriting Recognition:

Some studies may focus on the integration of text-to-handwriting conversion with handwriting recognition systems, creating a seamless interaction between digital text and handwritten content.

* Performance Metrics and Evaluation:

Literature in this area may discuss methodologies for evaluating the performance of text-to-handwriting conversion systems. Metrics might include the visual realism of the generated handwriting, speed of conversion, and user satisfaction.

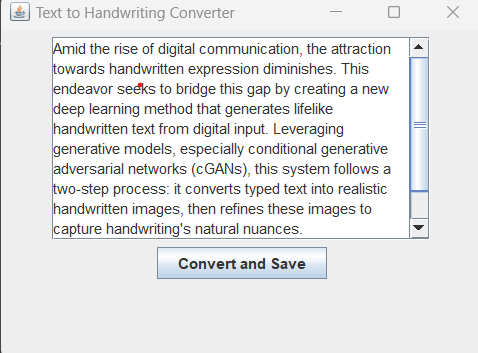
1. **REQUIREMENTS**
   1. **Requirement Analysis**
   2. **Hardware Requirement**

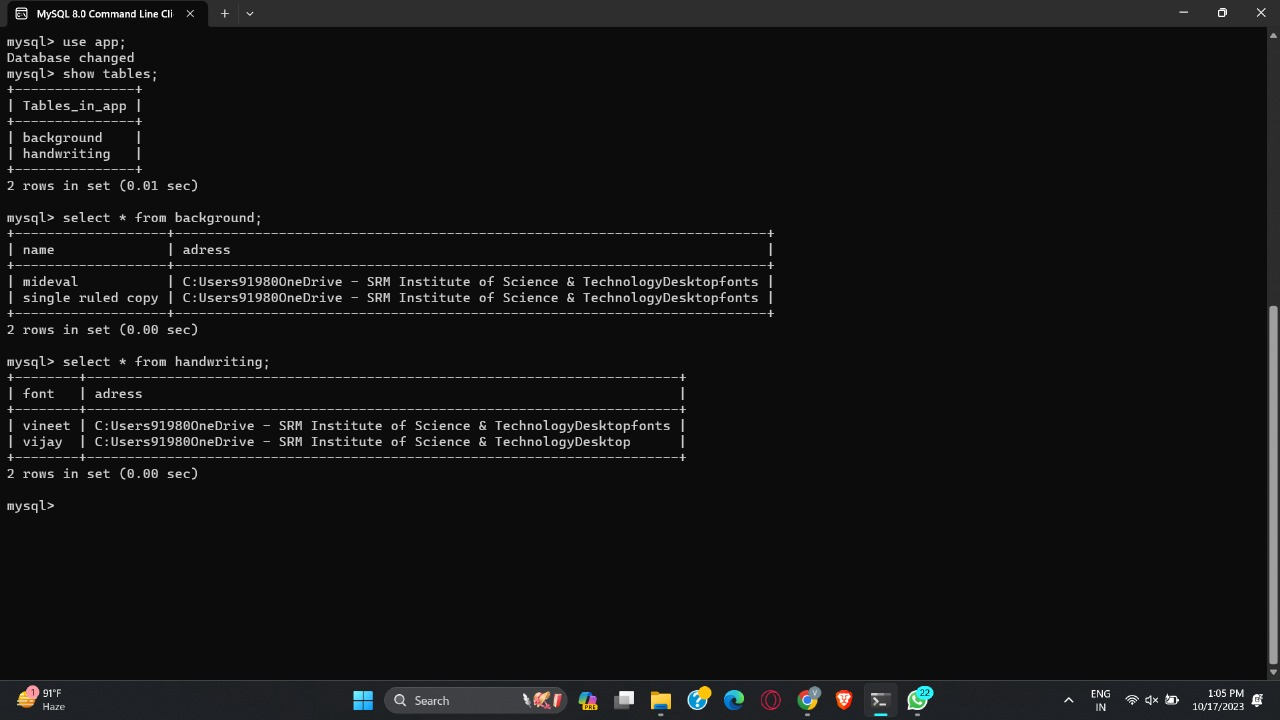
For monitoring network usage, has relatively low hardware requirements. Here are the basic hardware requirements for running the code:

* Computer
* Processor: An Intel Core i3 or equivalent, is sufficient.
* Memory (RAM): 2 GB of RAM or more is recommended for smooth operation
* Required Libraries: The code relies on several Java libraries, including Swings(Java GUI Library), java.awt, JDBC, Imagelo, java.awt.font, and Apache Commons IO.
* Network Connectivity: The code primarily convert text to handwriting , so it doesn't require specific network hardware.

It's important to note that the provided code is not resource-intensive, and the hardware requirements are quite minimal. Most modern computers, even low-end ones, should be able to run the code without any issues. The primary factor influencing performance is the system's ability to handle the Javainterpreter and the GUI components.

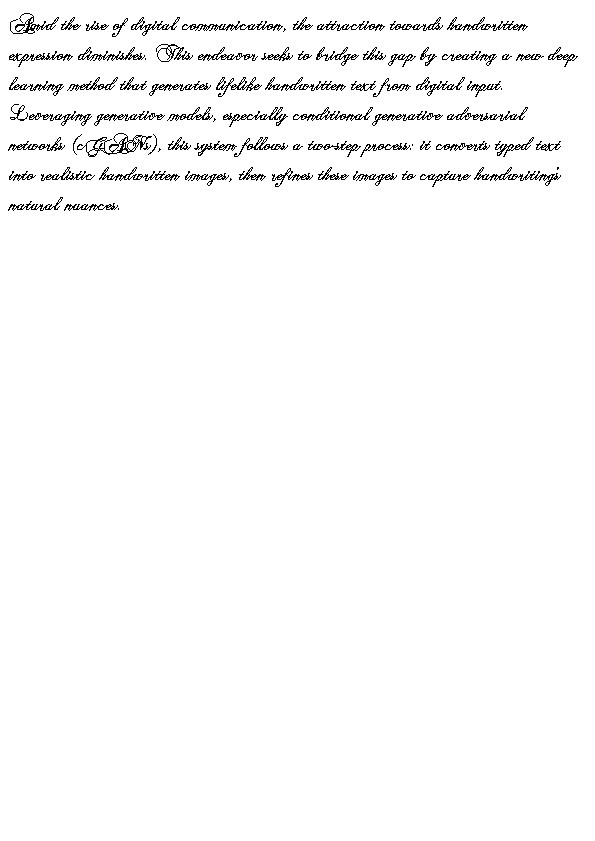
1. **IMPLEMENTATION**



**Fig 4.1 Screenshot of GUI Design**

**Fig 4.2 Screenshot of DATABASE**

1. **OUTPUT**

****

**Fig 5.1 Screenshot of OUTPUT**

1. **CODE**

**package com.example.finalt2h;**

**import javax.swing.\*;**

**import java.awt.\*;**

**import java.awt.image.BufferedImage;**

**import java.io.File;**

**import java.io.IOException;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.PreparedStatement;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**import javax.imageio.ImageIO;**

**public class TextToHandwritingConverter extends JFrame {**

**private String text;**

**private Font handwritingFont;**

**private int fontSize;**

**private int pageWidth;**

**private int pageHeight;**

**private int currentY;**

**private int currentPageIndex;**

**private BufferedImage currentPage;**

**private Graphics2D currentGraphics;**

**private JComboBox<String> handwritingStyleComboBox; // Added JComboBox for handwriting styles**

**public TextToHandwritingConverter() {**

**// Create a JFrame with a title**

**super("Text to Handwriting Converter");**

**// Initialize page size (A4: 210mm x 297mm at 72 DPI)**

**pageWidth = (int) (210 \* 72 / 25.4);**

**pageHeight = (int) (297 \* 72 / 25.4);**

**// Initialize font and font size**

**fontSize = 20; // Start with a default font size**

**// Create a JPanel to hold the text and handwritingStyleComboBox**

**JPanel panel = new JPanel();**

**// Initialize the handwritingStyleComboBox**

**handwritingStyleComboBox = new JComboBox<>();**

**loadHandwritingStyles(); // Load styles into the combo box**

**handwritingStyleComboBox.addItem("Style 1");**

**handwritingStyleComboBox.addItem("Style 2");**

**// Create a JTextArea for text input**

**JTextArea textArea = new JTextArea(10, 30);**

**textArea.setLineWrap(true);**

**textArea.setWrapStyleWord(true);**

**JScrollPane scrollPane = new JScrollPane(textArea);**

**// Create a button to update the text and save as an image**

**JButton convertButton = new JButton("Convert and Save");**

**convertButton.addActionListener(e -> {**

**text = textArea.getText();**

**currentPageIndex = 1;**

**String selectedStyle = (String) handwritingStyleComboBox.getSelectedItem();**

**try (Connection connection = DriverManager.getConnection("jdbc:mysql://localhost/txt", "root", "1234")) {**

**// String selectedStyle = (String) handwritingStyleComboBox.getSelectedItem(); // This line is not needed here**

**loadHandwritingFont(selectedStyle);**

**saveTextToDatabase(text, selectedStyle, connection);**

**currentPage = new BufferedImage(pageWidth, pageHeight, BufferedImage.TYPE\_INT\_ARGB);**

**currentGraphics = currentPage.createGraphics();**

**currentGraphics.setFont(handwritingFont);**

**currentGraphics.setColor(Color.BLACK);**

**currentGraphics.setColor(Color.WHITE);**

**currentGraphics.fillRect(0, 0, pageWidth, pageHeight);**

**currentGraphics.setColor(Color.BLACK);**

**writeTextToPage(text, currentGraphics, fontSize);**

**saveCurrentPage();**

**textArea.setText(text);**

**JOptionPane.showMessageDialog(this, "Images saved as page" + currentPageIndex + ".png and text saved to database.");**

**} catch (SQLException ex) {**

**ex.printStackTrace();**

**JOptionPane.showMessageDialog(this, "Error connecting to the database: " + ex.getMessage());**

**}**

**});**

**panel.add(scrollPane);**

**panel.add(convertButton);**

**// Add the panel to the frame**

**add(panel);**

**// Set frame properties**

**setSize(400, 300);**

**setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);**

**setLocationRelativeTo(null);**

**setVisible(true);**

**}**

**private void loadHandwritingStyles() {**

**try (Connection connection = DriverManager.getConnection("jdbc:mysql://localhost/txt", "root", "1234")) {**

**String query = "SELECT DISTINCT style FROM handwriting\_styles";**

**try (PreparedStatement statement = connection.prepareStatement(query)) {**

**ResultSet resultSet = statement.executeQuery();**

**while (resultSet.next()) {**

**handwritingStyleComboBox.addItem(resultSet.getString("style"));**

**}**

**}**

**} catch (SQLException ex) {**

**ex.printStackTrace();**

**JOptionPane.showMessageDialog(this, "Error loading handwriting styles: " + ex.getMessage());**

**}**

**}**

**private void loadHandwritingFont(String selectedStyle) {**

**try (Connection connection = DriverManager.getConnection("jdbc:mysql://localhost/txt", "root", "1234")) {**

**String query = "SELECT font\_path FROM handwriting\_styles WHERE style = ?";**

**try (PreparedStatement statement = connection.prepareStatement(query)) {**

**statement.setString(1, selectedStyle);**

**ResultSet resultSet = statement.executeQuery();**

**if (resultSet.next()) {**

**String fontPath = resultSet.getString("font\_path");**

**try {**

**handwritingFont = Font.createFont(Font.TRUETYPE\_FONT, new File(fontPath));**

**handwritingFont = handwritingFont.deriveFont(Font.PLAIN, fontSize);**

**} catch (FontFormatException | IOException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**} catch (SQLException ex) {**

**ex.printStackTrace();**

**JOptionPane.showMessageDialog(this, "Error loading handwriting font: " + ex.getMessage());**

**}**

**}**

**private void saveTextToDatabase(String text, String selectedStyle, Connection connection) throws SQLException {**

**String query = "INSERT INTO handwritten\_pages (text, style) VALUES (?, ?)";**

**try (PreparedStatement statement = connection.prepareStatement(query)) {**

**statement.setString(1, text);**

**statement.setString(2, selectedStyle);**

**statement.executeUpdate();**

**}**

**}**

**private void writeTextToPage(String text, Graphics2D g2d, int fontSize) {**

**FontMetrics fm = g2d.getFontMetrics();**

**int lineHeight = fm.getHeight();**

**int maxLines = pageHeight / lineHeight;**

**currentY = lineHeight;**

**for (String line : text.split("\n")) {**

**String[] words = line.split(" ");**

**StringBuilder currentLine = new StringBuilder();**

**for (String word : words) {**

**String lineWithWord = currentLine + (currentLine.length() > 0 ? " " : "") + word;**

**int lineWidth = fm.stringWidth(lineWithWord);**

**if (lineWidth > pageWidth - 20) {**

**if (currentY + lineHeight < maxLines \* lineHeight) {**

**g2d.drawString(currentLine.toString(), 10, currentY);**

**currentY += lineHeight;**

**currentLine = new StringBuilder(word);**

**} else {**

**saveCurrentPage();**

**currentPageIndex++;**

**currentPage = new BufferedImage(pageWidth, pageHeight, BufferedImage.TYPE\_INT\_ARGB);**

**currentGraphics = currentPage.createGraphics();**

**currentGraphics.setFont(handwritingFont);**

**currentGraphics.setColor(Color.BLACK);**

**currentGraphics.setColor(Color.WHITE);**

**currentGraphics.fillRect(0, 0, pageWidth, pageHeight);**

**currentGraphics.setColor(Color.BLACK);**

**currentY = lineHeight;**

**g2d = currentGraphics;**

**g2d.drawString(currentLine.toString(), 10, currentY);**

**currentY += lineHeight;**

**currentLine = new StringBuilder(word);**

**}**

**} else {**

**currentLine = new StringBuilder(lineWithWord);**

**}**

**}**

**if (currentY + lineHeight < maxLines \* lineHeight) {**

**g2d.drawString(currentLine.toString(), 10, currentY);**

**currentY += lineHeight;**

**} else {**

**saveCurrentPage();**

**currentPageIndex++;**

**currentPage = new BufferedImage(pageWidth, pageHeight, BufferedImage.TYPE\_INT\_ARGB);**

**currentGraphics = currentPage.createGraphics();**

**currentGraphics.setFont(handwritingFont);**

**currentGraphics.setColor(Color.BLACK);**

**currentGraphics.setColor(Color.WHITE);**

**currentGraphics.fillRect(0, 0, pageWidth, pageHeight);**

**currentGraphics.setColor(Color.BLACK);**

**currentY = lineHeight;**

**g2d = currentGraphics;**

**g2d.drawString(currentLine.toString(), 10, currentY);**

**currentY += lineHeight;**

**}**

**}**

**}**

**private void saveCurrentPage() {**

**try {**

**String desktopPath = System.getProperty("user.home") + File.separator + "Desktop";**

**String folderName = "t2h";**

**File folder = new File(desktopPath, folderName);**

**if (!folder.exists()) {**

**folder.mkdir();**

**}**

**File outputFile = new File(folder, "page" + currentPageIndex + ".png");**

**ImageIO.write(currentPage, "png", outputFile);**

**} catch (IOException ex) {**

**ex.printStackTrace();**

**}**

**}**

**public static void main(String[] args) {**

**SwingUtilities.invokeLater(() -> new TextToHandwritingConverter());**

**}**

**}**

1. **CONCLUSION**

**The Text-to-Handwriting Converter in Java is a versatile application that utilizes several key libraries and technologies.**

**Swing and AWT libraries form the foundation for creating an intuitive graphical user interface, allowing users to input text and choose handwriting styles effortlessly.**

**JDBC is employed for database connectivity, enabling the storage and retrieval of handwriting styles and fonts, thereby enhancing the system's flexibility and customization.**

**ImageIO, a part of the Java Image I/O API, is used for saving the generated handwritten pages as image files, providing tangible outputs for users.**

**Font handling classes and the MySQL JDBC driver contribute to the dynamic loading and application of selected handwriting styles, ensuring a diverse and customizable user experience.**

**Optionally, the incorporation of deep learning frameworks can enhance handwriting synthesis capabilities, resulting in more realistic and context-aware handwritten content.**

**Attention to non-functional requirements, such as usability, performance, and security, is crucial throughout the development process, with Apache Commons IO being utilized for efficient file-related tasks.**

**The collaboration of these libraries, coupled with a robust requirements analysis, results in a versatile and user-centric application with potential applications in education, accessibility, and other domains.**

**Staying informed about the latest advancements in libraries and frameworks is essential for continually improving the Text-to-Handwriting Converter in response to evolving technology.**

**REFERENCES**

* 1. Java Swing Tutorial (official documentation):

<https://docs.oracle.com/javase/tutorial/uiswing/>

* 1. Java JDBC Tutorial (official documentation)

<https://docs.oracle.com/en/java/javase/14/docs/api/java.sql/module-summary.html>

* 1. Dafont:

<https://www.dafont.com/>

A website offering a wide range of free and premium fonts for personal and commercial use.

* 1. ChatGPT (for content)

<https://opneai.com/>