Technological Institute of the Philippines
Quezon City

College of Information Technology Education

Diffchecker Desktop
Using Text Comparison Algorithm

In partial fulfillment for the course

ITE012- Computer Programming 2

Submitted by:

Jose A. Perez Jr. - BSIT

Submitted to

Ms. Arceli F. Salo

October 2025

Table of Contents

I. INTRODUCTION

Background of the Study

In modern software development, document processing, and version control workflows, identifying changes between two or more versions of a file is a critical task. Developers, writers, editors, and researchers often rely on diff tools to compare text content efficiently. Traditional diff tools, however, are either too technical, lack visual clarity, or are not customizable for specific workflows.

The development of a custom Diffchecker application addresses these limitations by providing a user-friendly environment where users can easily compare content, highlight differences, and analyze changes with accuracy and efficiency. This system is particularly valuable in environments that require precision, collaboration, and documentation of modifications.

Project Objectives

The Diffchecker application is designed with the following objectives:

- Provide a clear visual comparison between two text inputs or files.
- Highlight added, removed, and modified lines to help users quickly identify differences.
- Offer a responsive and easy-to-use interface suitable for both technical and non-technical users.
- Support future expandability, such as saving results, version integration, and merge capabilities.
- Increase productivity and accuracy in proofreading, code review, and document editing processes.

Significance of the Study

The Diffchecker application contributes value across multiple fields and use cases:

- **For Programmers** – simplifies code review and debugging by clearly showing differences between file versions.
- **For Writers and Editors** – enables precise proofreading and revision tracking.
- **For Students and Researchers** – assists in comparing documentation drafts and academic content.
- **For Teams and Organizations** – supports collaboration, transparency, and version accountability.

By providing a custom-built solution, the project bridges the gap between basic diff tools and more complex version control systems, making comparison tasks more accessible and efficient.

Scope and Delimitations

Scope:

The system focuses on comparing two blocks of text or files and visually displaying their differences. Core features include side-by-side comparison, line change detection, word change detection, and intuitive highlighting. It uses SQLite database for doing CRUD operations like creating a tab data, reading a saved tab, updating the saved tab's data, deleting a saved tab's data. It also includes the ability to toggle between syntax highlighting and a normal text editor. It also features custom Java frame for a more modern interface.

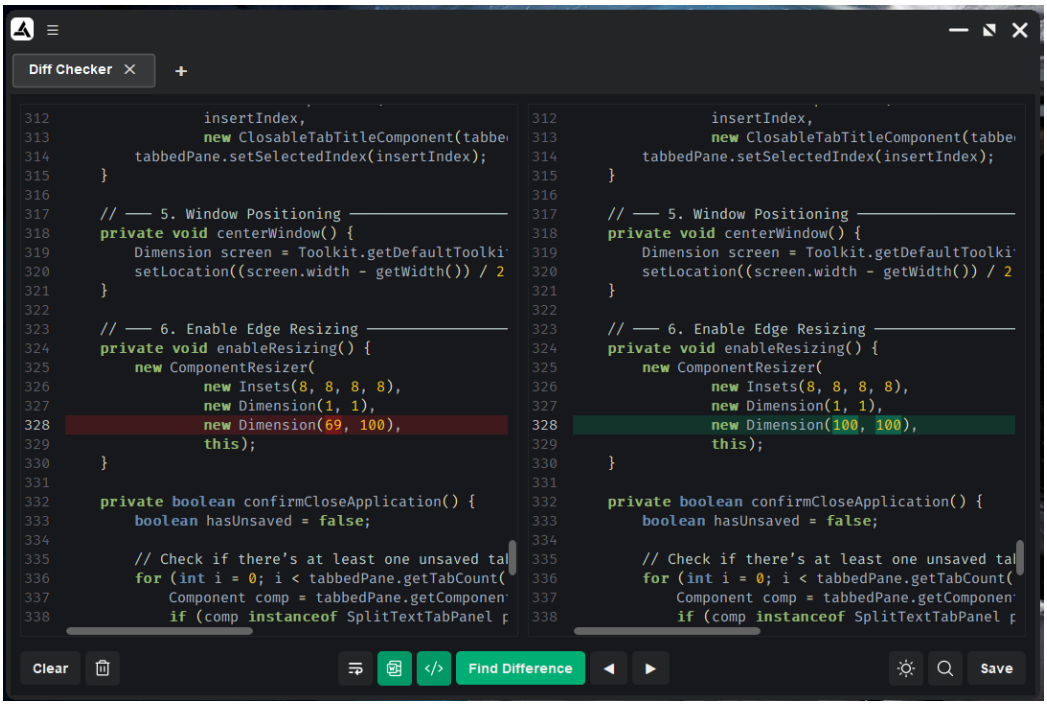
Delimitations:

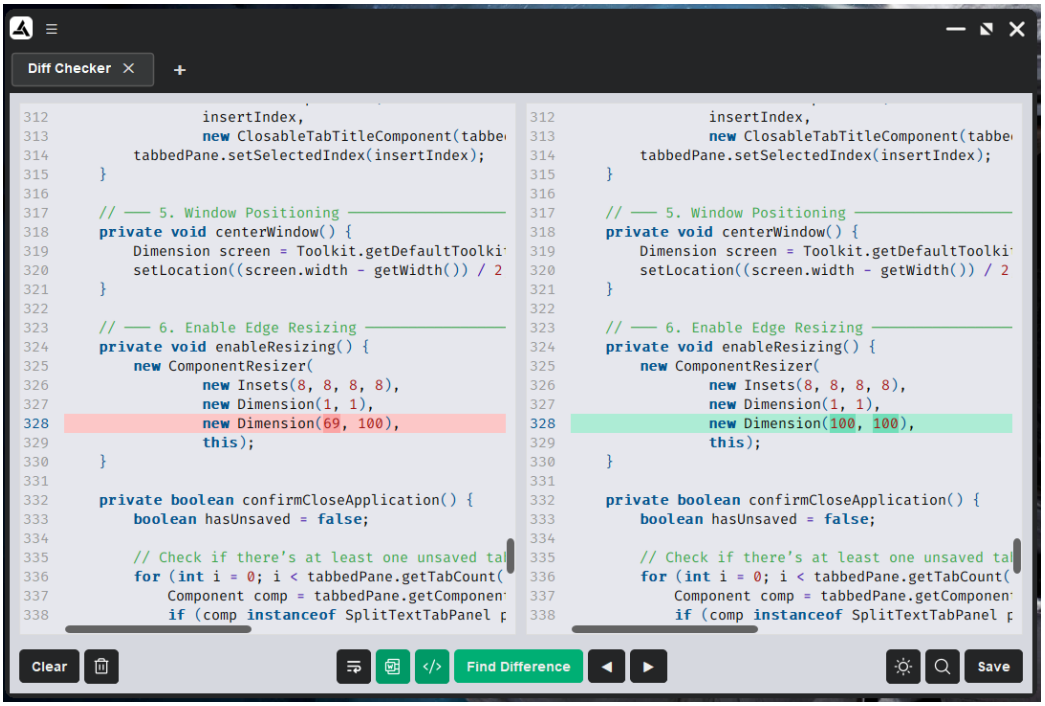
- The current version is limited to pasted text on the built in code editor
- File merge functionality and multi-file comparison are outside the initial implementation.
- It does not support text file importation
- It does not support Image Comparison
- It is does support saving data to the cloud and multi user collaborations

II. USER'S MANUAL

Screen shot of output with description

Dark Mode / Light Mode Themes





Instructions on how to use:

Figure 1 *To start, paste your two differing texts individually on each text editors / code editors*

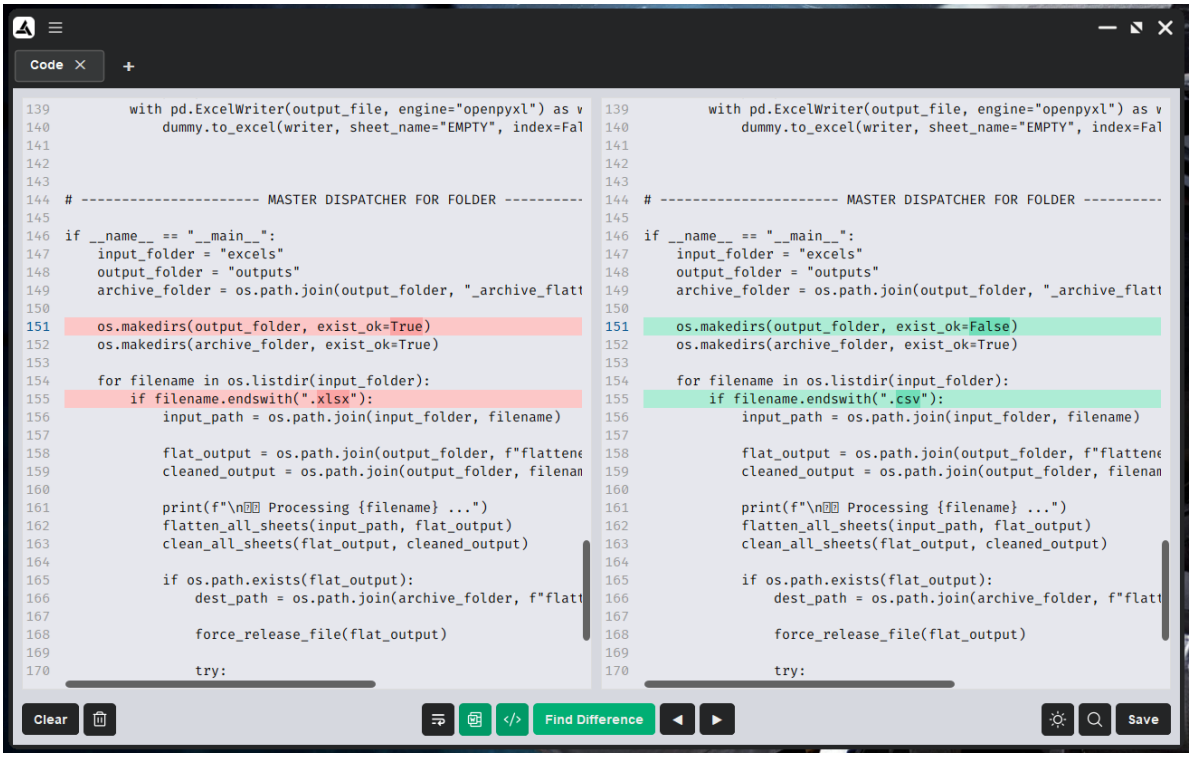


Figure 2 *After pasting your two differing texts into the text editors, click the "Find Difference" button to highlight texts that are different from each other. You can also cycle through the next or previous other differences using the "Arrow" buttons*

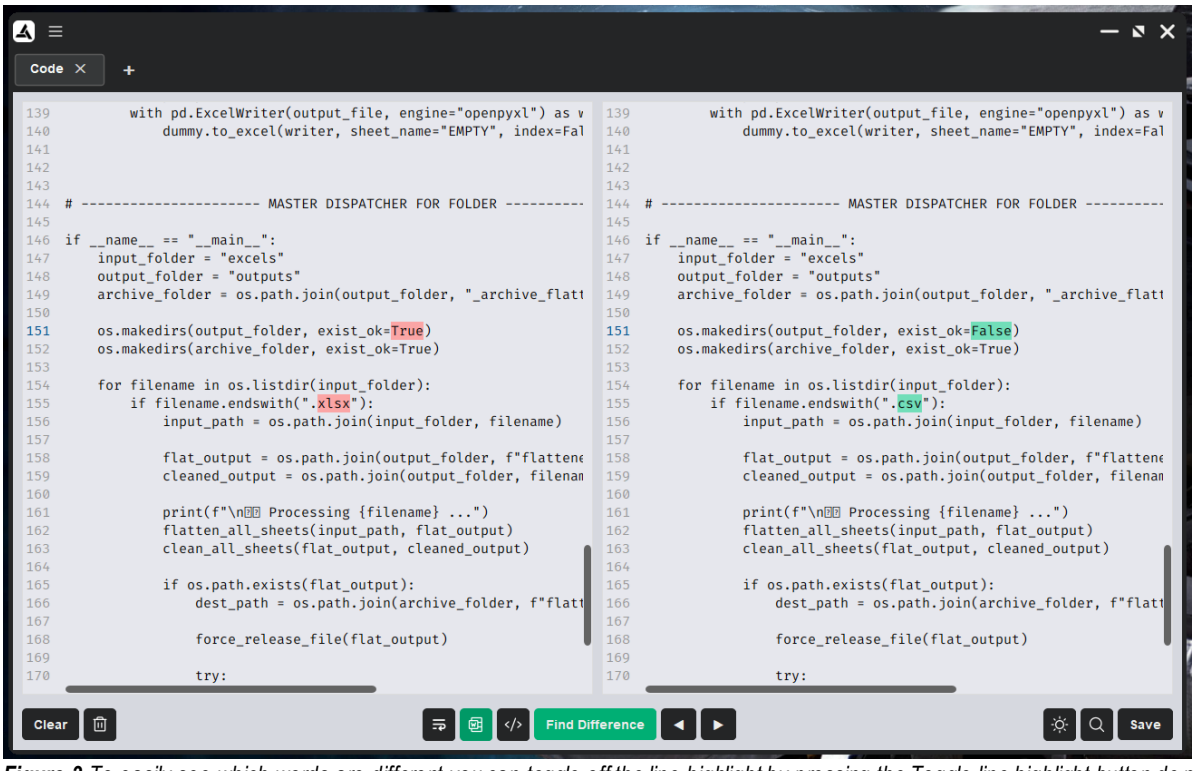


Figure 3 *To easily see which words are different you can toggle off the line highlight by pressing the Toggle line highlight button de noted by these characters "</>"*

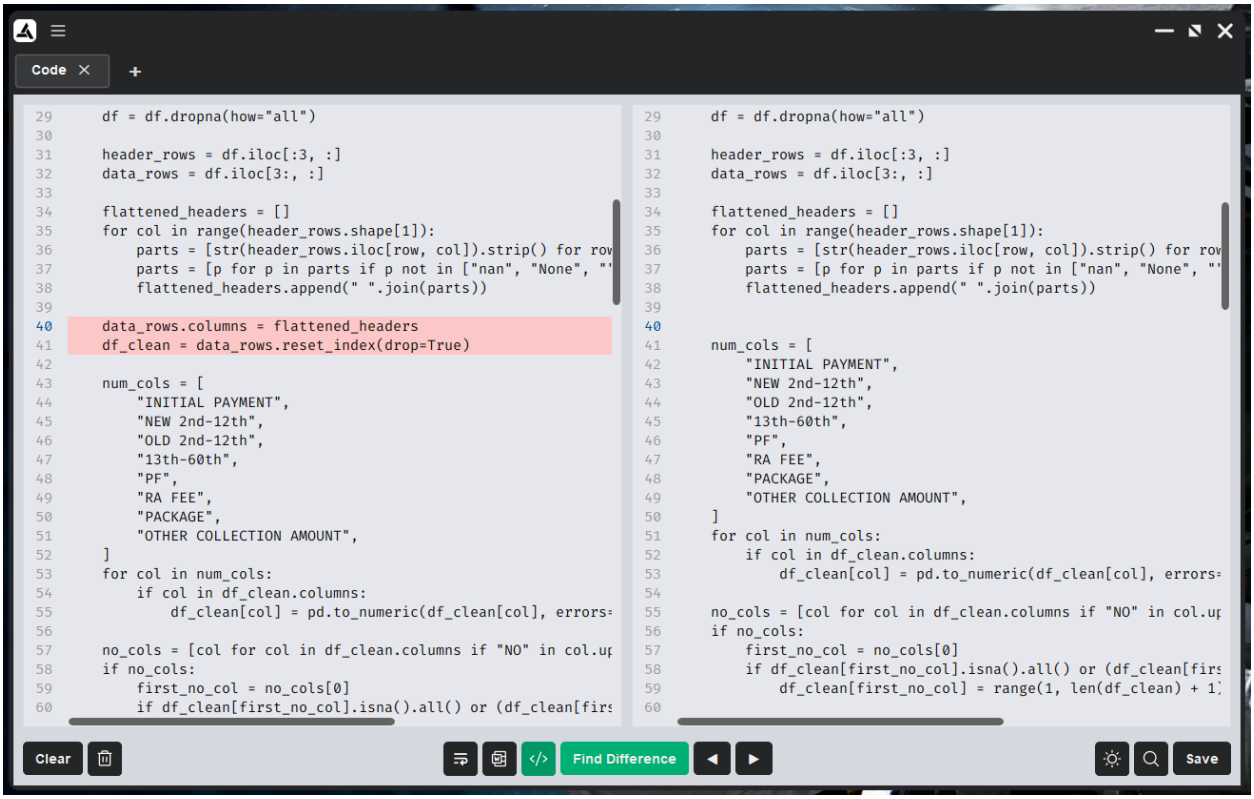


Figure 4 *In some cases there are texts or codes that are completely absent from original text that you have pasted. You can easily spot those by toggling Line Highlight on.*

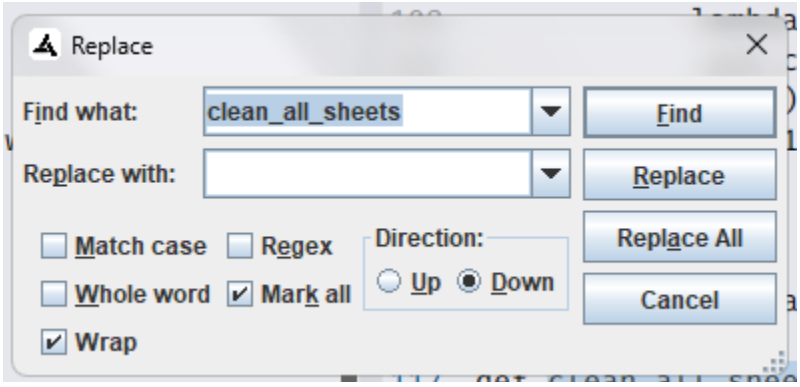


Figure 5 *Sometimes, there are cases wherein you would need to find and replace some texts. You can easily do that by pressing the Magnifying glass button or by hitting Ctrl + F on your keyboard.*

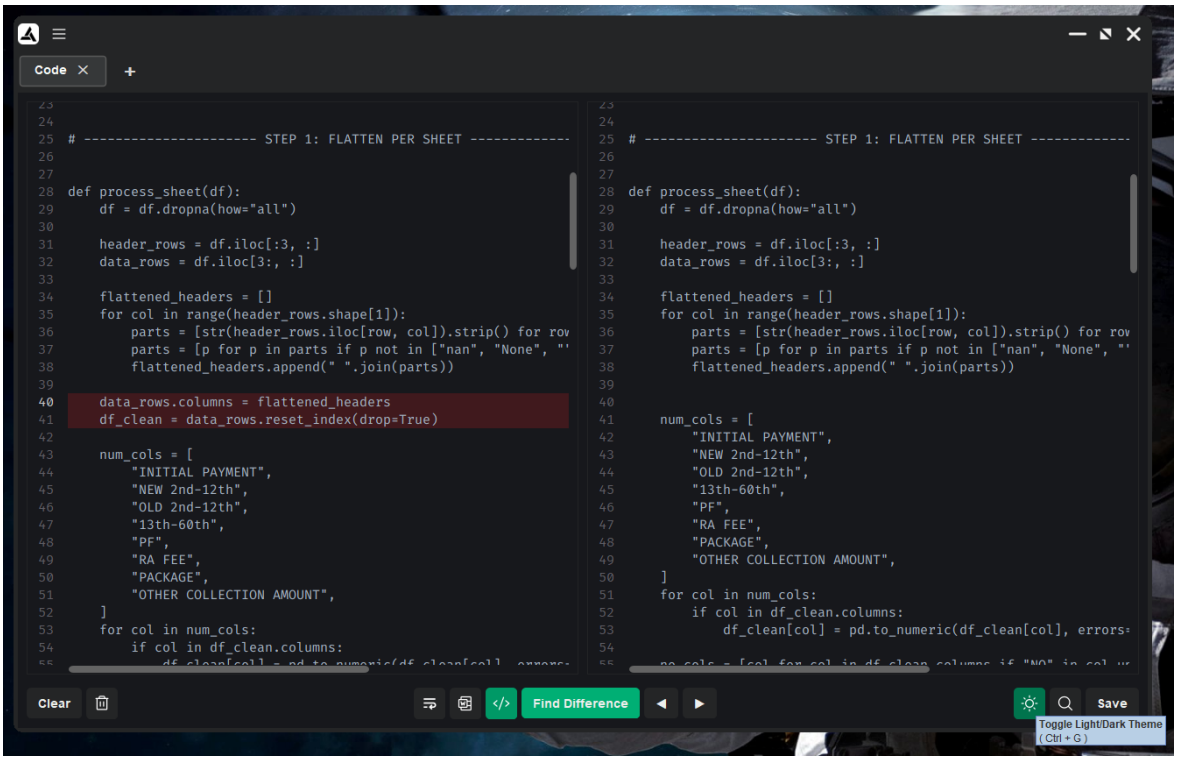


Figure 6 *Sometimes, there are also cases wherein you want to not strain your eyes especially on long hours of coding. You can turn on Dark mode hitting Ctrl + G or clicking the sun button.*

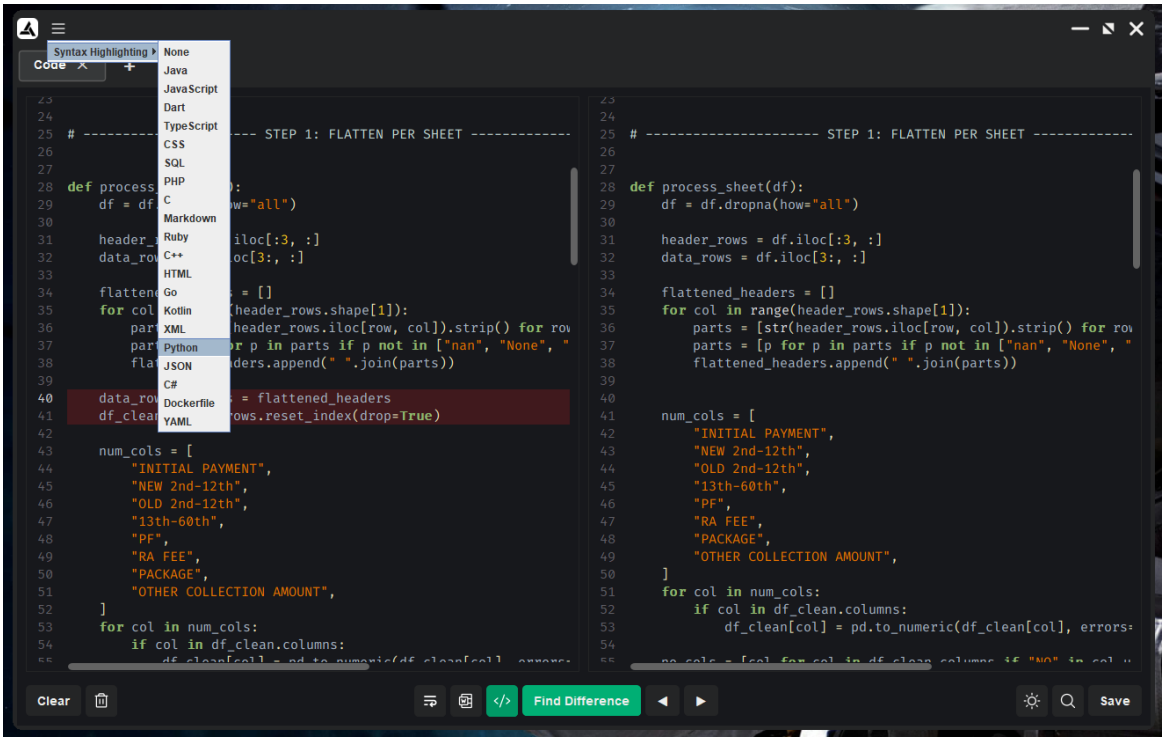


Figure 7 You can also turn on the Code Editor feature by turning on the Syntax highlighting via menu button up top on the title bar.

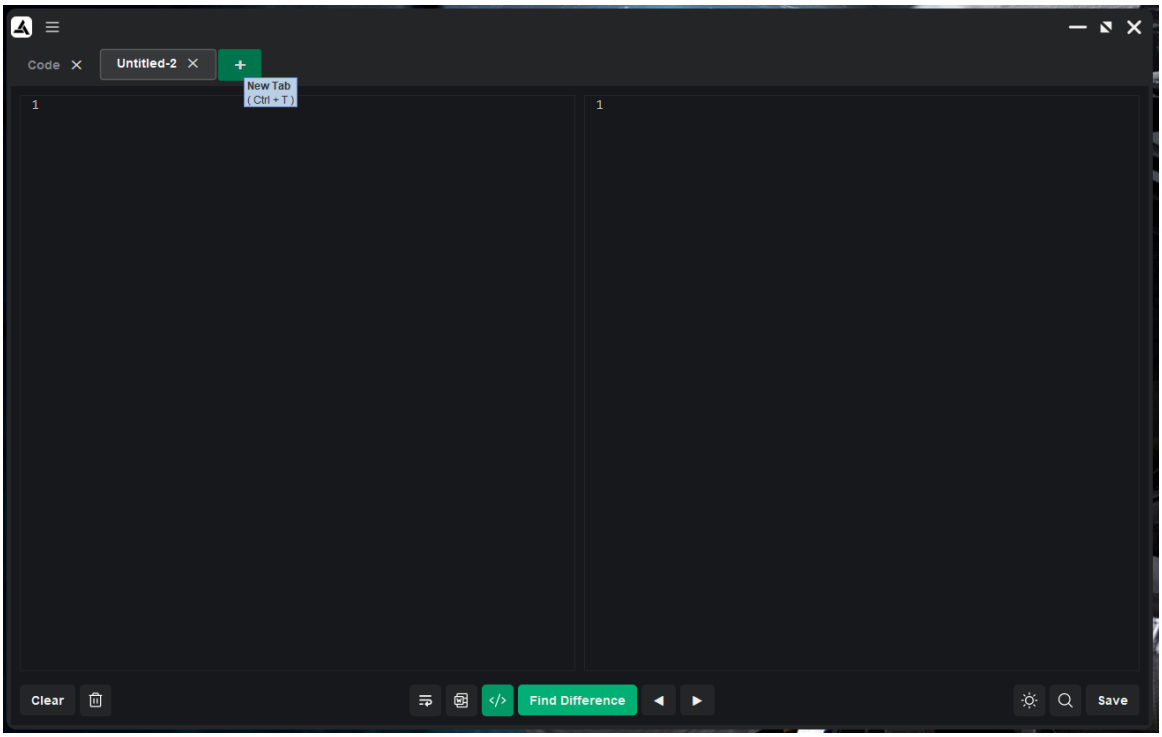


Figure 8 *You can open a "New Tab" by pressing Ctrl + T or by click the Add New Tab "+" button*

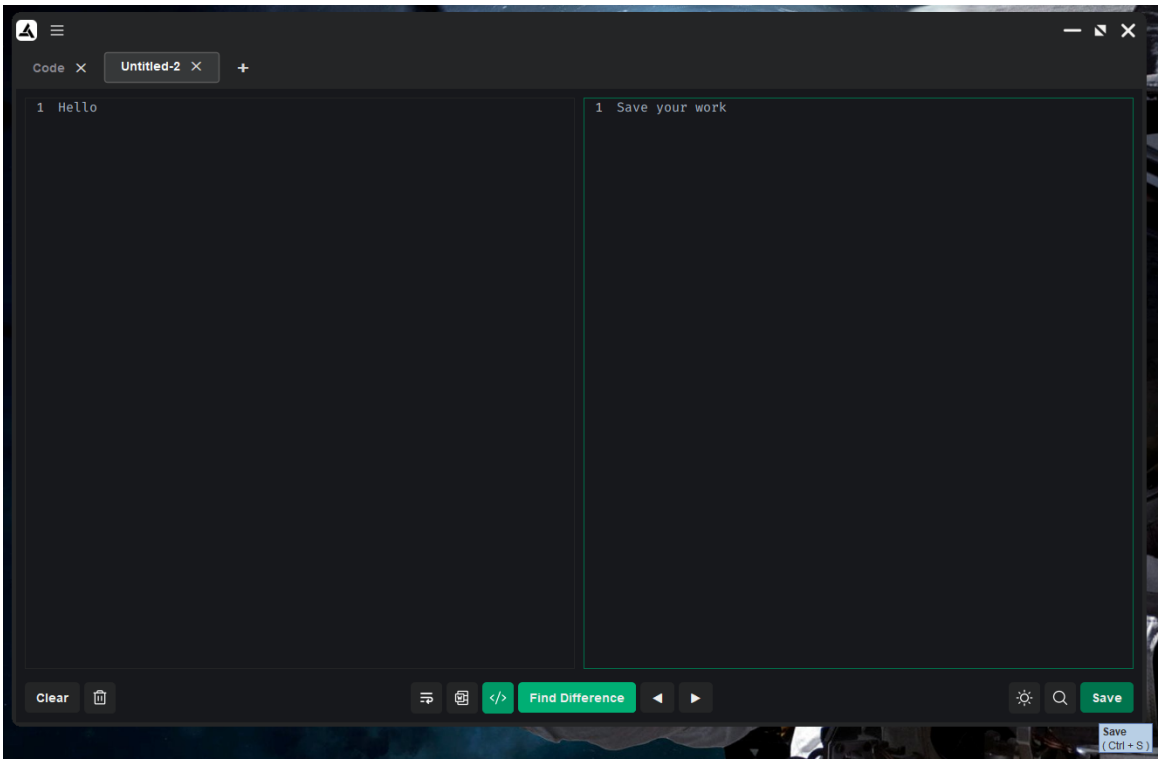


Figure 9 *You can save your work into the SQLite database by either clicking the Save button or pressing Ctrl + S*

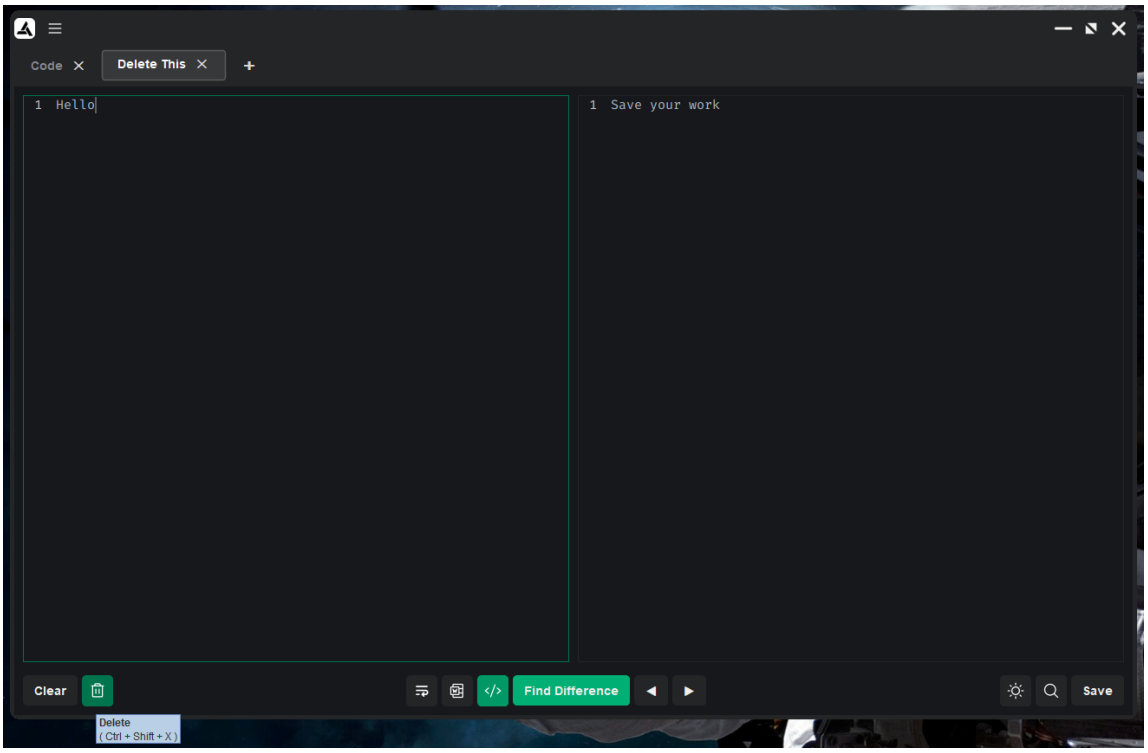


Figure 10 *You can delete your work by pressing Ctrl + Shift + X or clicking the trash icon*

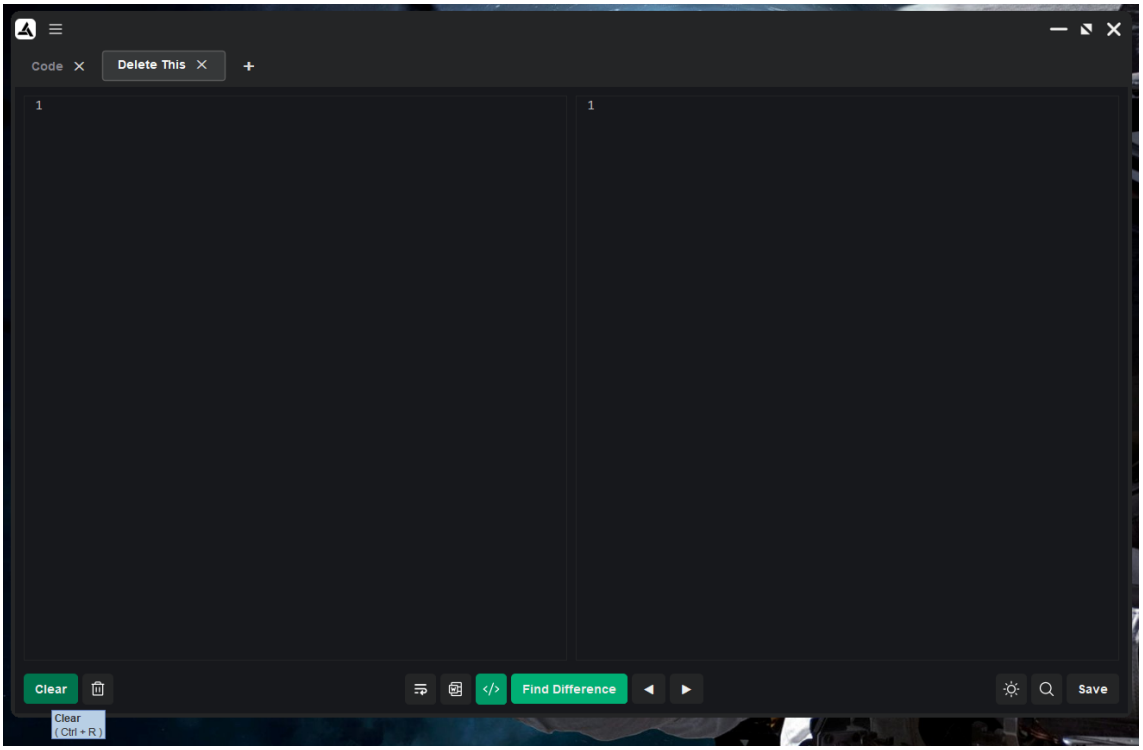
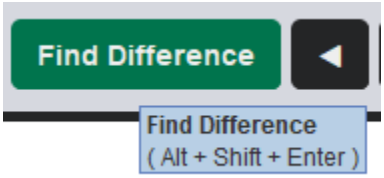


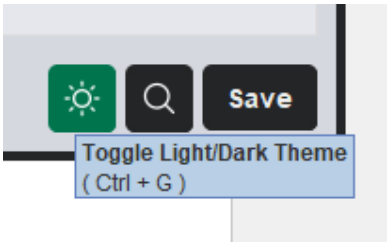
Figure 11 *Finally, to clear everything on both of the editors at the same time you can press Ctrl + R or hit the Clear button*

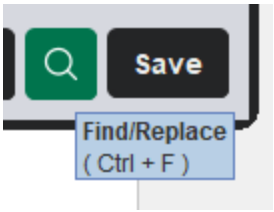
Keyboard Shortcuts

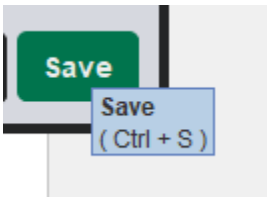
- **Alt + Shift + Enter:** Find difference between two texts
- **Alt + E:** Toggle line highlight
- **Alt + W:** Toggle word highlight
- **Alt + Q:** Toggle wrap
- **Ctrl + G:** Toggle light theme/dark theme
- **Ctrl + S:** Save
- **Ctrl + F:** Find/Replace
- **Ctrl + Shift + X:** Delete tab from database
- **Ctrl + R:** Clear texts from both Text Editors
- **Ctrl + W:** Close current tab

Sample Tooltips







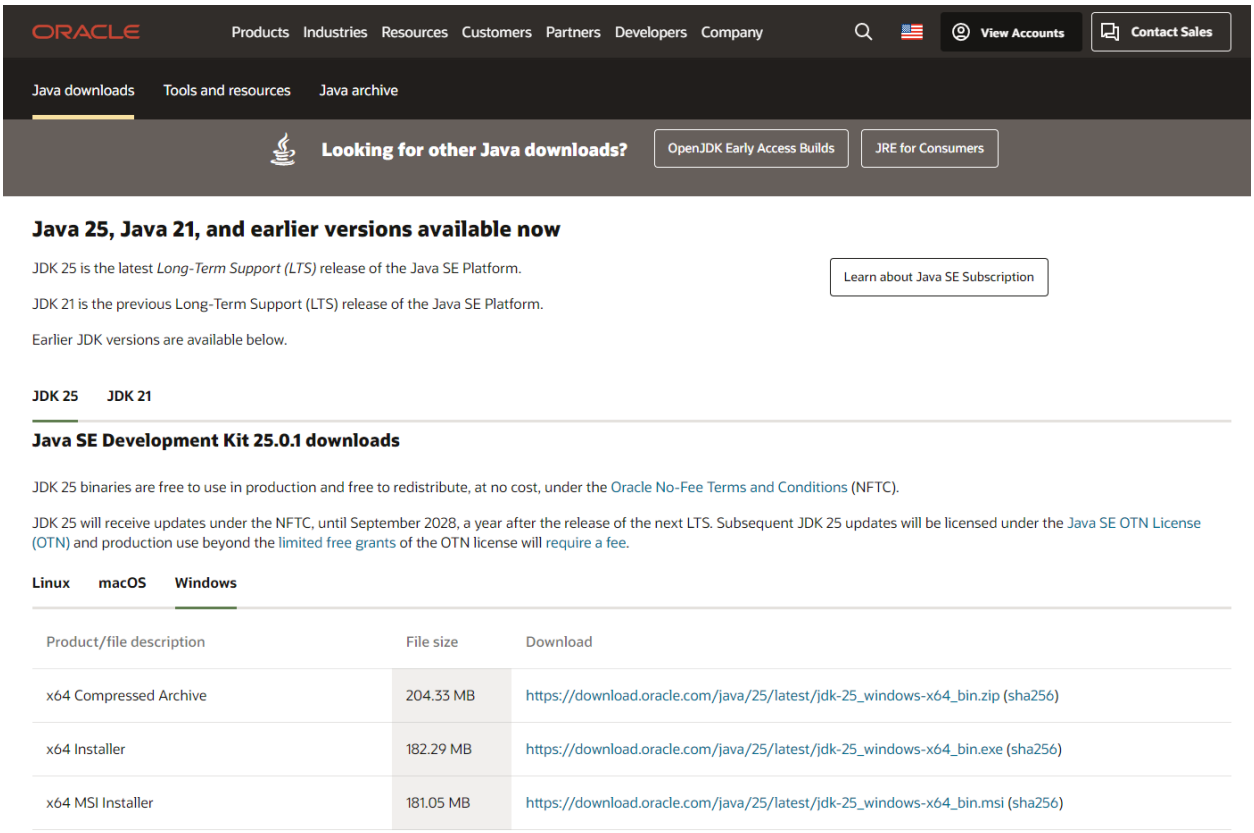


Installation instructions

- about the **"Run anyway"** screen, in order for me to remove ***that*** warning for this desktop app i would need to pay a trusted Certificate Authority for a **"Code Signing Certificate" (around $70 – $500 PER YEAR)** from Sectigo (Comodo), GoDaddy, SSL.com etc. in order for Windows to not flag me as an **"Unknown Publisher."**
- this is a ***free*** tool
- Java Runtime Environment is required to run this program

Step 1: Download JDK first

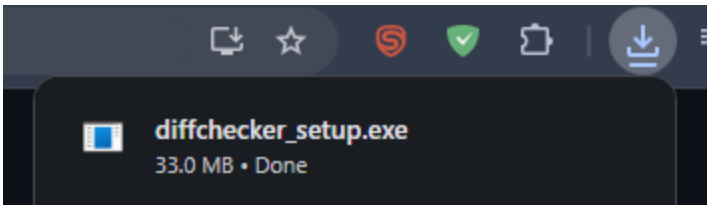
https://www.oracle.com/java/technologies/downloads/



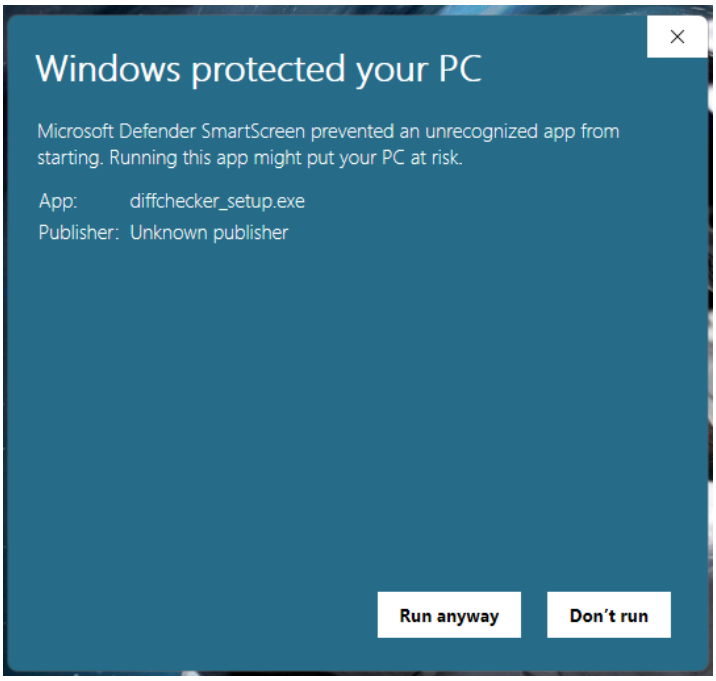
- Download the latest version of java JDK 21 or higher. **x64 Windows Installer** is applicable for this desktop application

Step 2: Download the installer

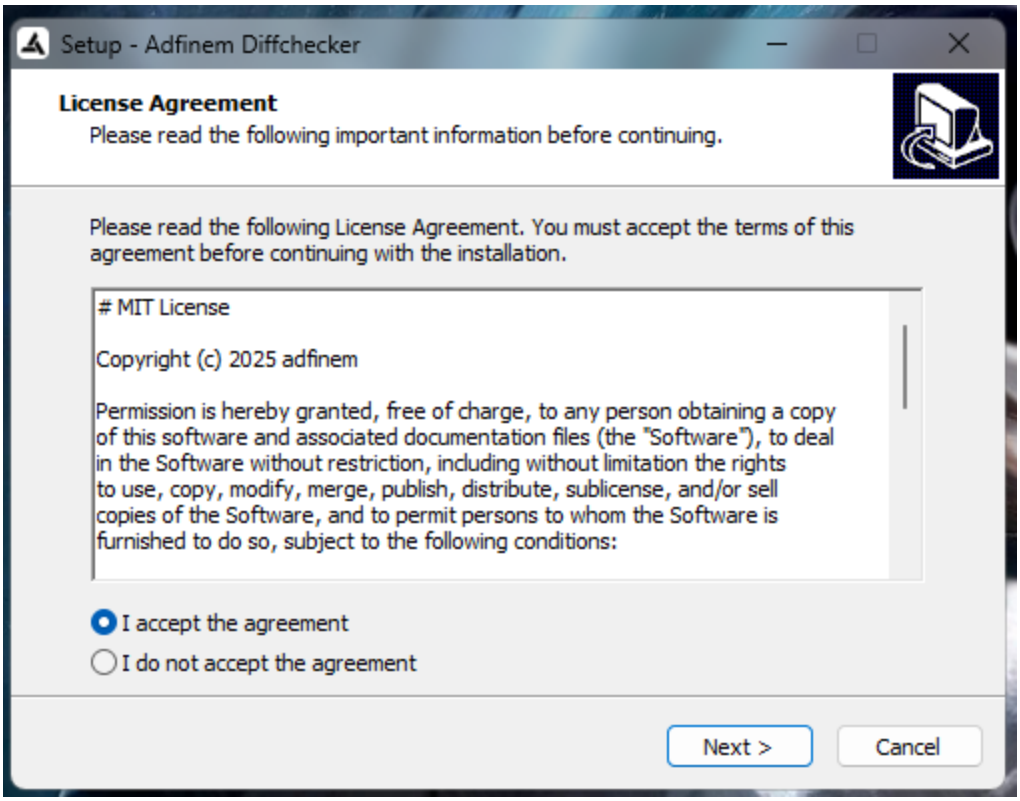
Download latest installer here:
https://github.com/rising-dancho/diffchecker_sqlite/releases

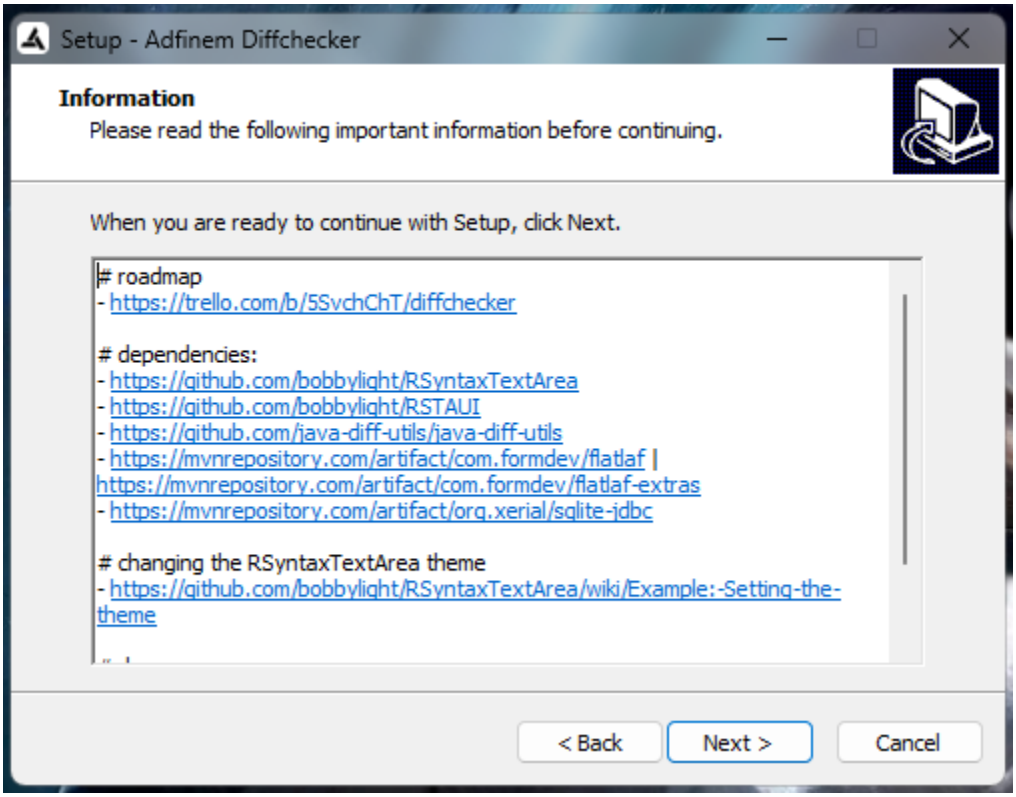


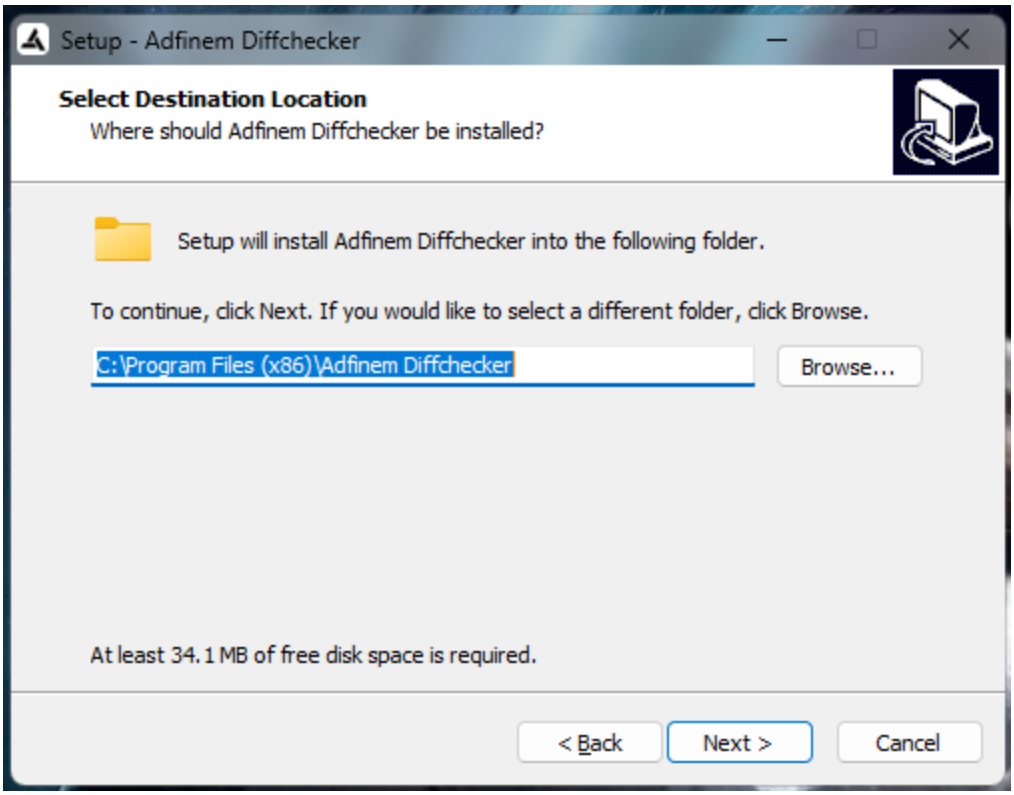
Click "More Info" > "Run anyway"

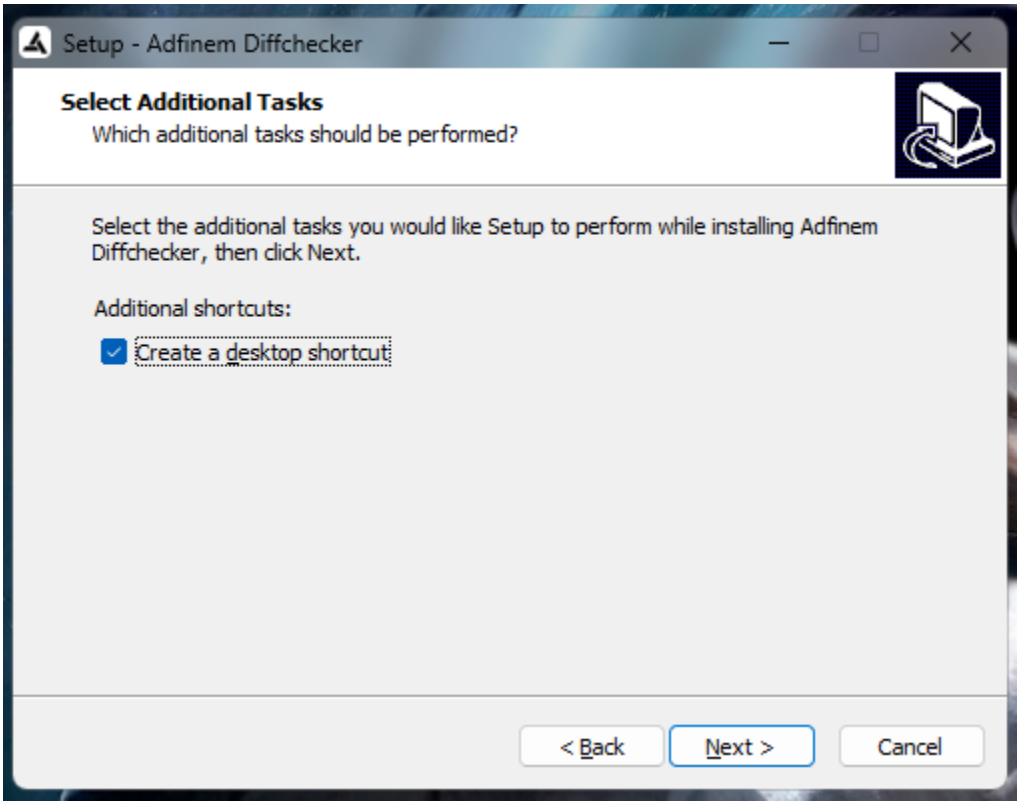


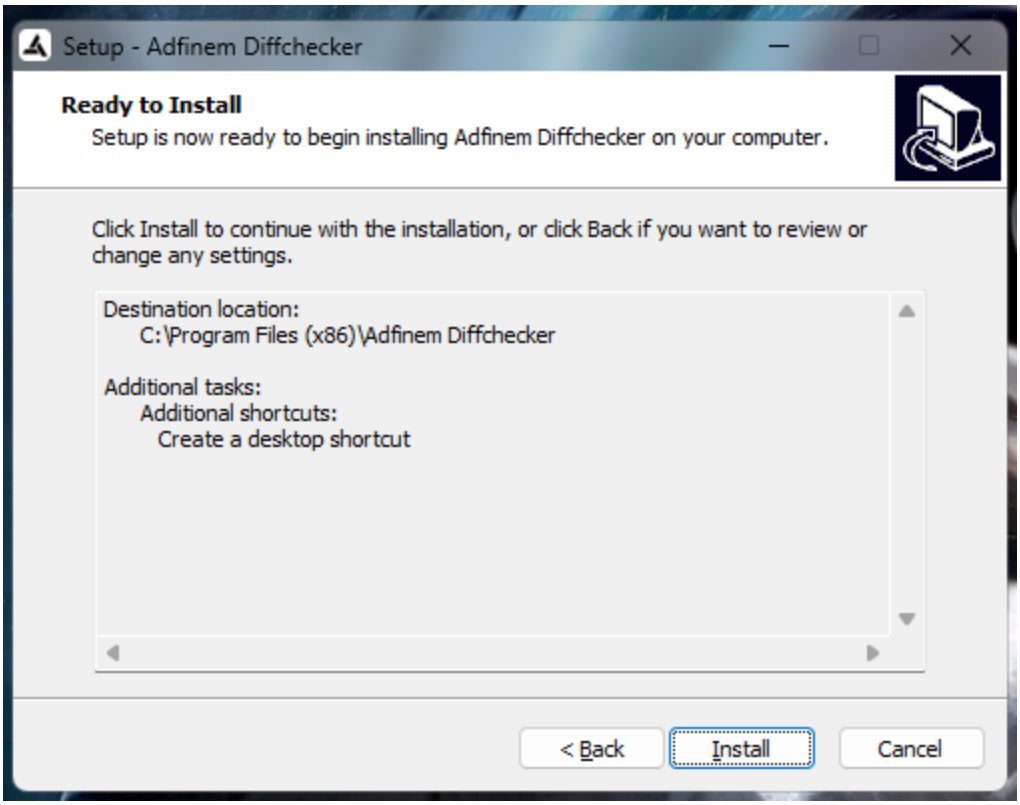
After that just proceed to your normal installation, hit "Next" until you see the "Finish" button

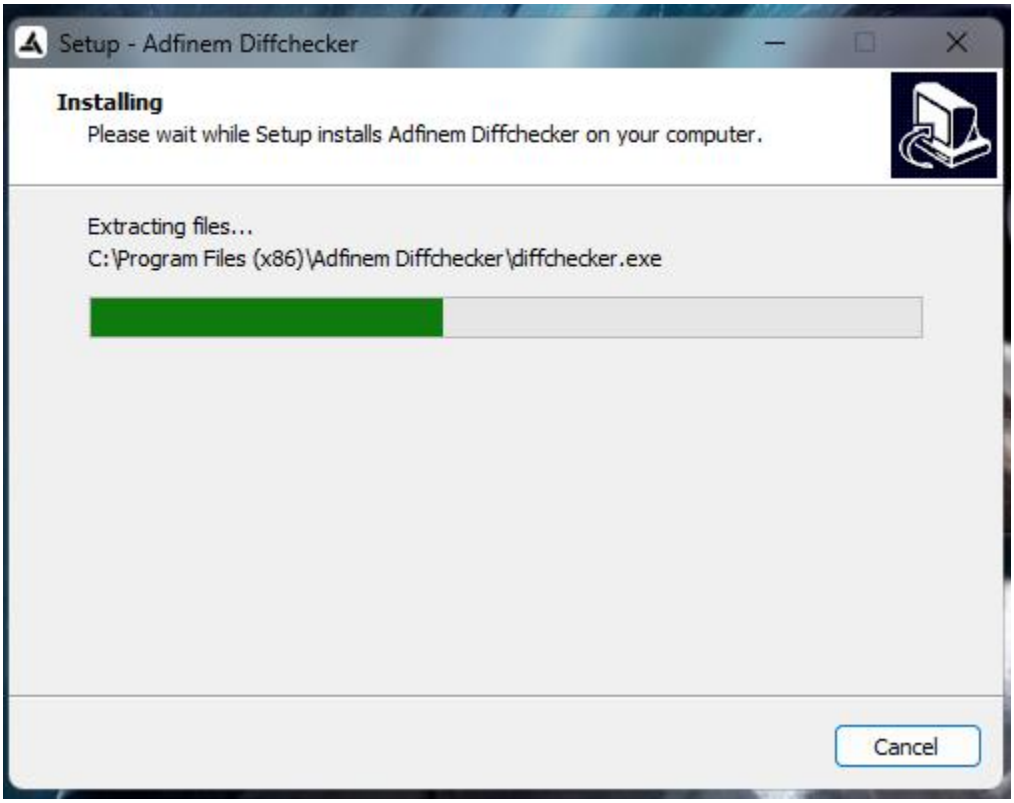

Setup - Adfinem Diffchecker
License Agreement
Please read the following important information before continuing.
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.
MIT License
Copyright (c) 2025 adfinem
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
I accept the agreement
I do not accept the agreement
Next >
Cancel

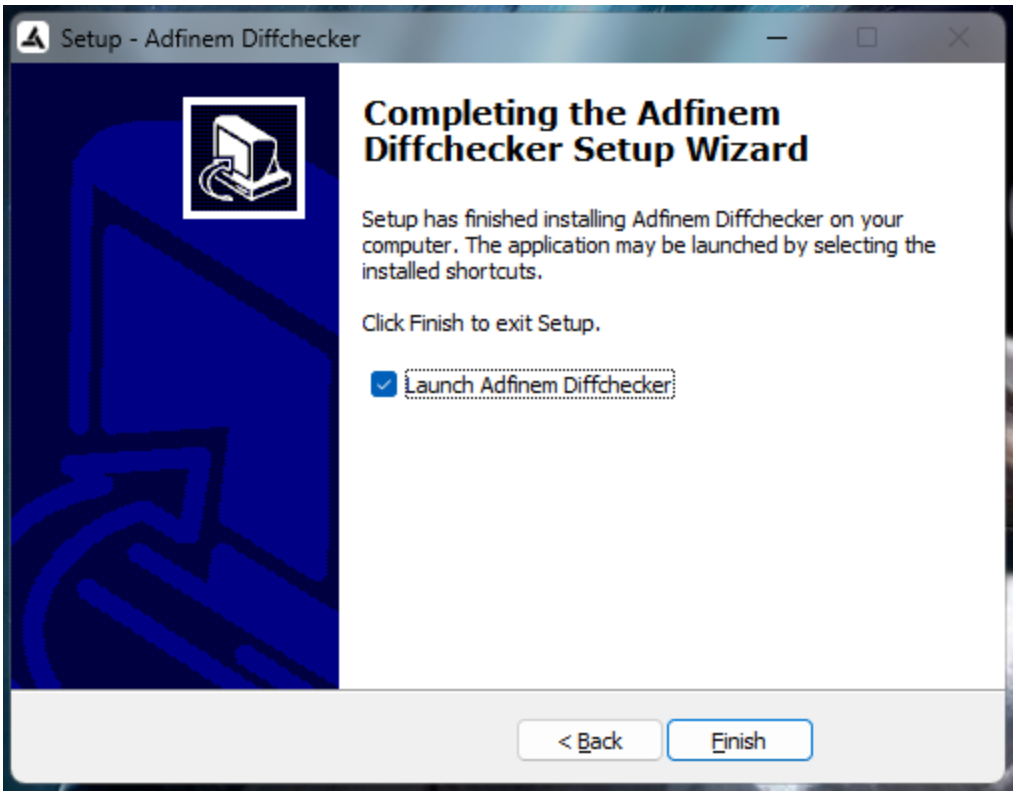

Setup - Adfinem Diffchecker
Information
Please read the following important information before continuing.
When you are ready to continue with Setup, click Next.
roadmap
- https://trello.com/b/5SvchChT/diffchecker
dependencies:
- https://github.com/bobbylight/RSyntaxTextArea
- https://github.com/bobbylight/RSTAUI
- https://github.com/java-diff-utils/java-diff-utils
- https://mvnrepository.com/artifact/com.formdev/flatlaf | https://mvnrepository.com/artifact/com.formdev/flatlaf-extras
- https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc
changing the RSyntaxTextArea theme
- https://github.com/bobbylight/RSyntaxTextArea/wiki/Example:-Setting-the-theme
< Back
Next >
Cancel


Setup - Adfinem Diffchecker
Select Destination Location
Where should Adfinem Diffchecker be installed?
Setup will install Adfinem Diffchecker into the following folder.
To continue, click Next. If you would like to select a different folder, click Browse.
C:\Program Files (x86)\Adfinem Diffchecker
Browse...
At least 34.1 MB of free disk space is required.
< Back
Next >
Cancel


Setup - Adfinem Diffchecker
Select Additional Tasks
Which additional tasks should be performed?
Select the additional tasks you would like Setup to perform while installing Adfinem Diffchecker, then click Next.
Additional shortcuts:
Create a desktop shortcut
< Back
Next >
Cancel


Setup - Adfinem Diffchecker
Ready to Install
Setup is now ready to begin installing Adfinem Diffchecker on your computer.
Click Install to continue with the installation, or click Back if you want to review or change any settings.
Destination location:
C:\Program Files (x86)\Adfinem Diffchecker
Additional tasks:
Additional shortcuts:
Create a desktop shortcut
< Back
Install
Cancel


Setup - Adfinem Diffchecker
Installing
Please wait while Setup installs Adfinem Diffchecker on your computer.
Extracting files...
C:\Program Files (x86)\Adfinem Diffchecker\diffchecker.exe
Cancel



Download version 1.0 release:
https://github.com/rising-dancho/diffchecker_sqlite/releases/download/v1.0.0/diffchecker_setup.exe

Video Demo:
https://www.youtube.com/watch?v=f4yTybuBwkg

Source Codes:
https://github.com/rising-dancho/diffchecker_sqlite

Main.java

```
package com.diffchecker;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;

import javax.swing.*;

import com.diffchecker.components.ClosableTabTitleComponent;
import com.diffchecker.components.ComponentResizer;
import com.diffchecker.components.CustomTitleBar;
// IMPORT COMPONENTS
import com.diffchecker.components.RoundedTabbedPaneUI;
import com.diffchecker.components.SplitTextTabPanel;
import com.diffchecker.components.Database.DB;
import com.diffchecker.components.Database.DiffData;
import com.diffchecker.components.Database.DiffRepository;

import java.util.List;

public class Main extends JFrame {
```

```
    // ——— Static Resources ———————————————————————————————————————————————
    private static final String PACKAGE_NAME = "diffchecker";
    private static final ImageIcon LOGO = new ImageIcon(
            Main.class.getResource("/" + PACKAGE_NAME +
"/images/logo/logo.png"));

    // ——— Instance Fields ————————————————————————————————————————————————
    private final JPanel container = new JPanel();
    private final Color FONT_COLOR = new Color(0xd6d6d6);
    public SplitTextTabPanel splitArea;

    // FOR CLOSING TABS WITH CTRL+W
    JTabbedPane tabbedPane;
    private final Runnable onTabEmptyFallback = () ->
addNewTab(tabbedPane);

    // hold a reference so we can inject the tabbedPane after it's created
    private CustomTitleBar titleBarComponent;

    public static void main(String[] args) {
        SwingUtilities.invokeLater(Main::new);
    }

    public Main() {
        initFrame(); // 1. Frame setup
        // Initialize first tab panel
        splitArea = new SplitTextTabPanel(() -> addNewTab(tabbedPane));
        JPanel wrapper = initWrapper(); // 2. Background wrapper
        JPanel titleBar = buildTitleBar(); // 3. Custom title bar (no
tabbedPane yet)
        // JPanel menuPanel = buildMenuPanel(); // 4. Menu bar
        JPanel content = buildMainContent(); // 5. Main tabbed pane area

        // Inject the tabbedPane into the title bar now that it's created
        if (titleBarComponent != null && tabbedPane != null) {
            titleBarComponent.setTabs(tabbedPane);
        }

        // FOR DEBUGGING PURPOSES ONLY
        //
menuPanel.setBorder(BorderFactory.createLineBorder(Color.GREEN));

        // —— Compose Layout ——————————————————————————————————————————————
        JPanel centerContent = new JPanel();
        centerContent.setLayout(new BoxLayout(centerContent,
BoxLayout.Y_AXIS));
        centerContent.setBackground(new Color(0x242526));
        // centerContent.add(menuPanel);
        centerContent.add(content);

        wrapper.add(titleBar, BorderLayout.NORTH);
        wrapper.add(centerContent, BorderLayout.CENTER);
        setContentPane(wrapper);

        // —— Final Steps ——————————————————————————————————————————————————
        centerWindow();
        enableResizing();

        // Hook into window closing
        addWindowListener(new java.awt.event.WindowAdapter() {
            @Override
            public void windowClosing(java.awt.event.WindowEvent e) {
                if (!confirmCloseApplication()) {
                    // cancel close
                    setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLO
SE);
                } else {
                    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                }
            }
        });

        setVisible(true);
    }

    // ——— 1. Initialize Frame ————————————————————————————————————————————
    private void initFrame() {
        setTitle("Diffchecker");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setIconImage(LOGO.getImage());
        setUndecorated(true);
        setBackground(new Color(0x1F1F1F));
        setSize(1080, 720);

        // Rounded corners
        setShape(new java.awt.geom.RoundRectangle2D.Double(0, 0,
getWidth(),
                getHeight(), 20, 20));
        addComponentListener(new java.awt.event.ComponentAdapter() {
            @Override
            public void componentResized(java.awt.event.ComponentEvent e)
{
                setShape(new java.awt.geom.RoundRectangle2D.Double(0, 0,
getWidth(),
                        getHeight(), 20, 20));
            }
        });
    }

    // ——— 2. Wrapper Panel ———————————————————————————————————————————————
    private JPanel initWrapper() {
        JPanel wrapper = new JPanel(new BorderLayout());
        wrapper.setBackground(new Color(0x242526));
        wrapper.setOpaque(true);
        // ALLOWING THE CORNERS TO HAVE ENOUGH SPACE TO
DETECT RESIZING
        wrapper.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
        return wrapper;
    }

    // ——— 3. Title Bar Panel ——————————————————————————————————————————————
    // note: no longer accepts a SplitTextTabPanel
    private JPanel buildTitleBar() {

        titleBarComponent = new CustomTitleBar(
                this,
                "",
                PACKAGE_NAME,
                "/" + PACKAGE_NAME + "/images/logo/logo_24x24.png",
                new Color(0x242526),
                33);

        // FOR DEBUGGING PURPOSES ONLY
        //
titleBarComponent.setBorder(BorderFactory.createLineBorder(Color.RED
));
```

```
        JPanel titleWrapper = new JPanel(new BorderLayout());
        titleWrapper.setOpaque(false);
        // titleWrapper.setBorder(BorderFactory.createEmptyBorder(3, 3, 3,
3));
        titleWrapper.add(titleBarComponent, BorderLayout.NORTH);

        return titleWrapper;
    }

    // ——— 4. Main Content Panel with Tabs
————————————————————————————————————————————————————————————————————
—
    private JPanel buildMainContent() {
        container.setBackground(new Color(0x242526));
        container.setLayout(new BorderLayout());
        container.setBorder(null);

        tabbedPane = new JTabbedPane();

        InputMap inputMap =
getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDO
W);
        ActionMap actionMap = getRootPane().getActionMap();

        // HOTKEY FOR CLOSING TABS (CTRL + W)
        // Ctrl+W → Close tab
        inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_W,
InputEvent.CTRL_DOWN_MASK), "closeTab");
        actionMap.put("closeTab", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                closeTabAt();
            }
        });

        // HOTKEY FOR OPENING TABS (CTRL + T)
        // Ctrl+T → New tab
        inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_T,
InputEvent.CTRL_DOWN_MASK), "newTab");
        actionMap.put("newTab", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                addNewTab(tabbedPane);
            }
        });

        RoundedTabbedPaneUI ui = new RoundedTabbedPaneUI();
        tabbedPane.setUI(ui); // Set only once

        tabbedPane.setFont(new Font("SansSerif", Font.BOLD, 13));
        tabbedPane.setFocusable(false);

        // Hover behavior
        ui.setHoverListener(index -> {
            for (int i = 0; i < tabbedPane.getTabCount(); i++) {
                Component c = tabbedPane.getTabComponentAt(i);
                if (c instanceof ClosableTabTitleComponent) {
                    ((ClosableTabTitleComponent) c).setHovered(i == index);
                }
            }
        });

        // ADD TABS
        JButton addButton = new JButton("+");
        addButton.setBorder(null);
        addButton.setFocusPainted(false);
        addButton.setContentAreaFilled(false);
        addButton.setPreferredSize(new Dimension(26, 26));
        addButton.addActionListener(e -> addNewTab(tabbedPane));
        addButton.setForeground(FONT_COLOR);
        addButton.setFont(addButton.getFont().deriveFont(13.8f));
        addButton.setToolTipText("<html><strong>New Tab</strong> <br> (
Ctrl + T )</html>");

        // ---- RESTORE LAST SESSION (load all diffs from DB) ----
        DB db = new DB();
        DiffRepository repo = new DiffRepository(db);

        List<DiffData> diffs = repo.getAllDiffs();
        for (DiffData data : diffs) {
            SplitTextTabPanel panel = new SplitTextTabPanel(() ->
addNewTab(tabbedPane));
            panel.loadFromDatabase(data); // populate the text areas
            int index = tabbedPane.getTabCount();
            tabbedPane.insertTab(data.title, null, panel, null, index);
            tabbedPane.setTabComponentAt(index,
                    new ClosableTabTitleComponent(
                            tabbedPane, data.title, onTabEmptyFallback, tabIndex -
> closeTabAt(tabIndex)));
        }

        // Always add one Untitled tab AFTER loading saved tabs
        addNewTab(tabbedPane);

        // Add the + button as the last tab
        tabbedPane.addTab("", null);
        tabbedPane.setTabComponentAt(tabbedPane.getTabCount() - 1,
addButton);

        container.add(tabbedPane, BorderLayout.CENTER);

        return container;
    }

    public void closeTabAt() {
        int index = tabbedPane.getSelectedIndex();
        closeTabAt(index);
    }

    public void closeTabAt(int index) {
        if (index == -1)
            return;

        Component comp = tabbedPane.getComponentAt(index);

        // don't close the "+" tab
        if (comp instanceof JButton)
            return;

        // Ask to save unsaved changes
        if (comp instanceof SplitTextTabPanel panel &&
panel.hasUnsavedChanges()) {
            int option = JOptionPane.showConfirmDialog(
                    this,
                    "You have unsaved changes. Save before closing?",
                    "Unsaved Changes",
                    JOptionPane.YES_NO_CANCEL_OPTION,
                    JOptionPane.WARNING_MESSAGE);

            if (option == JOptionPane.CANCEL_OPTION || option ==
JOptionPane.CLOSED_OPTION) {
                // Do nothing if cancelled or dialog closed
                return;
            }
            if (option == JOptionPane.YES_OPTION) {
                panel.saveToDatabase();
            }
        }
```

```
        tabbedPane.remove(index);

        // If only the "+" tab remains, create and select a new real tab
        if (tabbedPane.getTabCount() == 1) {
            addNewTab(tabbedPane);
            return;
        }

        // Move selection to previous real tab, skipping "+"
        int newIndex = Math.max(0, index - 1);
        Component newComp =
tabbedPane.getTabComponentAt(newIndex);
        if (newComp instanceof JButton) {
            newIndex = Math.max(0, newIndex - 1);
        }
        tabbedPane.setSelectedIndex(newIndex);
    }

    private int untitledCounter = 1;

    private void addNewTab(JTabbedPane tabbedPane) {
        // Decide where to insert:
        // - If the last tab is the "+" button, insert before it
        // - Otherwise, append at the end (e.g., on first run before "+" exists)
        int insertIndex = tabbedPane.getTabCount();
        if (insertIndex > 0) {
            Component lastTabComponent =
tabbedPane.getTabComponentAt(insertIndex - 1);
            if (lastTabComponent instanceof JButton) {
                insertIndex--; // keep new tab before the "+" tab
            }
        }

        String title = "Untitled-" + untitledCounter++;
        splitArea = new SplitTextTabPanel(() -> addNewTab(tabbedPane));

        tabbedPane.insertTab(title, null, splitArea, null, insertIndex);
        tabbedPane.setTabComponentAt(
                insertIndex,
                new ClosableTabTitleComponent(tabbedPane, title,
onTabEmptyFallback, tabIndex -> closeTabAt(tabIndex)));
        tabbedPane.setSelectedIndex(insertIndex);
    }

    // ——— 5. Window Positioning
————————————————————————————————————————————————————————————————————
    private void centerWindow() {
        Dimension screen = Toolkit.getDefaultToolkit().getScreenSize();
        setLocation((screen.width - getWidth()) / 2, (screen.height -
getHeight()) / 2);
    }

    // ——— 6. Enable Edge Resizing
————————————————————————————————————————————————————————————————————
    private void enableResizing() {
        new ComponentResizer(
                new Insets(8, 8, 8, 8),
                new Dimension(1, 1),
                new Dimension(100, 100),
                this);
    }

    private boolean confirmCloseApplication() {
        boolean hasUnsaved = false;

        // Check if there's at least one unsaved tab
        for (int i = 0; i < tabbedPane.getTabCount(); i++) {
            Component comp = tabbedPane.getComponentAt(i);
            if (comp instanceof SplitTextTabPanel panel &&
panel.hasUnsavedChanges()) {
                hasUnsaved = true;
                break;
            }
        }

        if (!hasUnsaved) {
            return true; // no unsaved changes → exit freely
        }

        // Ask once for all tabs
        int option = JOptionPane.showConfirmDialog(
                this,
                "You have unsaved changes in one or more tabs. Save before
exiting?",
                "Unsaved Changes",
                JOptionPane.YES_NO_CANCEL_OPTION,
                JOptionPane.WARNING_MESSAGE);

        if (option == JOptionPane.CANCEL_OPTION || option ==
JOptionPane.CLOSED_OPTION) {
            return false; // cancel exit
        }

        if (option == JOptionPane.YES_OPTION) {
            // Save ALL unsaved tabs
            for (int i = 0; i < tabbedPane.getTabCount(); i++) {
                Component comp = tabbedPane.getComponentAt(i);
                if (comp instanceof SplitTextTabPanel panel &&
panel.hasUnsavedChanges()) {
                    panel.saveToDatabase();
                }
            }
        }

        return true; // allow exit (either saved or discarded)
    }

}
```

DB.java

```
package com.diffchecker.components.Database;


import java.io.File;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;


public class DB {
```

```
  private static final String URL;

  static {
    // Get %APPDATA% (Roaming) → e.g. C:\Users\YourUser\AppData\Roaming
    String appData = System.getenv("APPDATA");
    if (appData == null) {
      // Fallback if not found
      appData = System.getProperty("user.home");
    }

    // Create a subfolder for your app
    File dbDir = new File(appData, "DiffcheckerAdfinem");
    if (!dbDir.exists()) {
      dbDir.mkdirs();
    }

    // Final DB file path
    File dbFile = new File(dbDir, "diffchecker.db");
    URL = "jdbc:sqlite:" + dbFile.getAbsolutePath();
  }

  public DB() {
    // Auto-create schema if database is new
    try (Connection conn = getConnection(); Statement stmt = conn.createStatement()) {
      String createTable = """
          CREATE TABLE IF NOT EXISTS diff_tabs (
              id INTEGER PRIMARY KEY AUTOINCREMENT,
              title TEXT NOT NULL,
              left_text TEXT,
              right_text TEXT
          );
          """;
      stmt.execute(createTable);
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }

  public Connection getConnection() throws SQLException {
    return DriverManager.getConnection(URL);
  }

  public static void close(AutoCloseable... resources) {
    for (AutoCloseable res : resources) {
      if (res != null) {
        try {
          res.close();
        } catch (Exception ignored) {
        }
      }
    }
  }
}
```

DiffData.java

```
package com.diffchecker.components.Database;

public class DiffData {
    public int id;
    public String title;
    public String leftText;
    public String rightText;

    public DiffData(int id, String title, String leftText, String rightText) {
        this.id = id;
        this.title = title;
        this.leftText = leftText;
```

```
        this.rightText = rightText;
    }

    // For new diffs without an id yet
    public DiffData(String title, String leftText, String rightText) {
        this(-1, title, leftText, rightText);
    }
}
```

DiffRepository.java

```
package com.diffchecker.components.Database;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class DiffRepository {
  private final DB db;

  public DiffRepository(DB db) {
    this.db = db;
  }

  public List<DiffData> getAllDiffs() {
    List<DiffData> list = new ArrayList<>();
    String sql = "SELECT id, title, left_text, right_text FROM diff_tabs";

    try (Connection conn = db.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql);
        ResultSet rs = stmt.executeQuery()) {
      while (rs.next()) {
        int id = rs.getInt("id");
        String title = rs.getString("title");
        String left = rs.getString("left_text");
        String right = rs.getString("right_text");
        list.add(new DiffData(id, title, left, right));
      }
    } catch (SQLException e) {
      e.printStackTrace();
    }
    return list;
  }

  public boolean updateDiff(DiffData data) {
    String sql = "UPDATE diff_tabs SET title = ?, left_text = ?, right_text = ?
WHERE id = ?";
    try (Connection conn = db.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql)) {
      stmt.setString(1, data.title);
      stmt.setString(2, data.leftText);
      stmt.setString(3, data.rightText);
      stmt.setInt(4, data.id);
      return stmt.executeUpdate() > 0;
    } catch (SQLException e) {
      e.printStackTrace();
    }
    return false;
  }

  public boolean deleteDiff(int id) {
    String sql = "DELETE FROM diff_tabs WHERE id = ?";
    try (Connection conn = db.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql)) {
      stmt.setInt(1, id);
      return stmt.executeUpdate() > 0;
    } catch (SQLException e) {
      e.printStackTrace();
    }
    return false;
  }

  public boolean saveDiff(DiffData data) {
    String sql = "INSERT INTO diff_tabs (title, left_text, right_text) VALUES
(?, ?, ?)";
    try (Connection conn = db.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {
      stmt.setString(1, data.title);
      stmt.setString(2, data.leftText);
      stmt.setString(3, data.rightText);
      int affected = stmt.executeUpdate();

      if (affected > 0) {
        try (ResultSet keys = stmt.getGeneratedKeys()) {
          if (keys.next()) {
            data.id = keys.getInt(1); // SQLite supports getGeneratedKeys
          }
        }
        return true;
      }
    } catch (SQLException e) {
      e.printStackTrace();
    }
    return false;
  }
}
```

EditorUtils.java

```
package com.diffchecker.components.Helper;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.Window;
import java.awt.event.ActionEvent;
```

```
import java.awt.event.KeyEvent;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import javax.swing.AbstractAction;
import javax.swing.Action;
import javax.swing.ActionMap;
import javax.swing.BorderFactory;
import javax.swing.InputMap;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JWindow;
import javax.swing.KeyStroke;
import javax.swing.text.BadLocationException;
import javax.swing.text.Caret;
import javax.swing.text.DefaultHighlighter;
import javax.swing.text.JTextComponent;
import javax.swing.Timer;

import org.fife.ui.rsyntaxtextarea.RSyntaxTextArea;
import org.fife.ui.rsyntaxtextarea.RSyntaxTextAreaEditorKit;
import org.fife.ui.rsyntaxtextarea.SyntaxConstants;

import com.github.difflib.DiffUtils;
import com.github.difflib.patch.Patch;

public class EditorUtils {

  private static final Color EDITOR_BACKGROUND = new
Color(0x17181C);
  private static final Color EDITOR_FONT_COLOR = new
Color(0xa9b7c6);
  public static final java.util.List<HighlightInfo> highlightPositions = new
ArrayList<>();

  public static class HighlightInfo {
    public int startOffset;
    public int endOffset;
    public RSyntaxTextArea area;

    public HighlightInfo(RSyntaxTextArea area, int start, int end) {
      this.area = area;
      this.startOffset = start;
      this.endOffset = end;
    }
  }

  public static RSyntaxTextArea createRSyntaxArea() {
    RSyntaxTextArea localArea = new RSyntaxTextArea();
    localArea.setSyntaxEditingStyle(SyntaxConstants.SYNTAX_STYLE_N
ONE);
    localArea.setAntiAliasingEnabled(true);
    localArea.setEditable(true); // Allow editing if you still want to diff edited
text
    localArea.setBackground(EDITOR_BACKGROUND);
    localArea.setForeground(EDITOR_FONT_COLOR);
    localArea.setCaretColor(EDITOR_FONT_COLOR);
    localArea.setBorder(BorderFactory.createEmptyBorder());
    // localArea.setCodeFoldingEnabled(true);

    // COMMENT AND UNCOMMENT HOTKEY (Ctrl+/ on Win/Linux,
Cmd+/ on macOS)
    int menuMask =
Toolkit.getDefaultToolkit().getMenuShortcutKeyMaskEx();
    InputMap im = localArea.getInputMap();
    ActionMap am = localArea.getActionMap();
    im.put(KeyStroke.getKeyStroke(KeyEvent.VK_SLASH, menuMask),
"toggleComment");
    am.put("toggleComment", new ToggleCommentWrapper(localArea));

    // LINE WRAPPING FOR LONG LINES
    // localArea.setLineWrap(true);
    // localArea.setWrapStyleWord(true); // optional, wraps at word
boundaries
    return localArea;
  }

  static class ToggleCommentWrapper extends AbstractAction {
    private final RSyntaxTextArea area;

    public ToggleCommentWrapper(RSyntaxTextArea area) {
      this.area = area;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
      Action toggleComment =
area.getActionMap().get(RSyntaxTextAreaEditorKit.rstaToggleCommentA
ction);
      if (toggleComment != null) {
        toggleComment.actionPerformed(e);
      }
    }
  }

  public static String capitalizeTitle(String input) {
    String[] words = input.trim().toLowerCase().split("\\s+");
    StringBuilder result = new StringBuilder();
    for (String word : words) {
      if (word.isEmpty())
        continue;
      result.append(Character.toUpperCase(word.charAt(0)))
          .append(word.substring(1))
          .append(" ");
    }
    return result.toString().trim();
  }

  public static void highlightFullLines(RSyntaxTextArea area, int startLine,
int count, Color color) {
    for (int i = 0; i < count; i++) {
      try {
        area.addLineHighlight(startLine + i, color);
      } catch (BadLocationException e) {
        e.printStackTrace();
      }
    }
  }

  // store highlight positions to clear later
  public static void highlightWordDiffs(RSyntaxTextArea area, int lineIndex,
String oldLine, String newLine,
      Color color,
      boolean isLeft) {
    List<String> tokens1 = Arrays.asList(oldLine.split("\\b"));
    List<String> tokens2 = Arrays.asList(newLine.split("\\b"));

    Patch<String> wordPatch = DiffUtils.diff(tokens1, tokens2);

    try {
      int pos = area.getLineStartOffset(lineIndex);
      List<String> tokens = isLeft ? tokens1 : tokens2;

      for (String token : tokens) {
        boolean changed = wordPatch.getDeltas().stream()
```

```
          .anyMatch(delta -> (isLeft ? delta.getSource().getLines() :
delta.getTarget().getLines())
              .contains(token));

      if (changed && !token.isBlank()) {
        int tokenStart = pos;
        int tokenEnd = pos + token.length();
        area.getHighlighter().addHighlight(tokenStart, tokenEnd,
            new DefaultHighlighter.DefaultHighlightPainter(color));
        highlightPositions.add(new HighlightInfo(area, tokenStart,
tokenEnd));
      }
      pos += token.length();
    }
   } catch (BadLocationException e) {
    e.printStackTrace();
   }
  }

  // simple toast popup
  public static void showToast(JButton button, String message) {
    // Create a lightweight popup
    JWindow toast = new JWindow();
    toast.setBackground(new Color(0, 0, 0, 0));

    // Style the label
    JLabel label = new JLabel(message);
    label.setOpaque(true);
    label.setBackground(new Color(50, 50, 50, 220)); // semi-transparent
dark bg
    label.setForeground(Color.WHITE);
    label.setBorder(BorderFactory.createEmptyBorder(8, 15, 8, 15));
    label.setFont(label.getFont().deriveFont(Font.PLAIN, 14f));

    toast.add(label);
    toast.pack();

    // Get button location on screen
    Point btnLoc = button.getLocationOnScreen();

    // Position toast directly above the button, centered horizontally
    int x = btnLoc.x + (button.getWidth() - toast.getWidth()) / 2;
    int y = btnLoc.y - toast.getHeight() - 8; // 8px gap above button

    toast.setLocation(x, y);

    // Show and auto-hide
    toast.setVisible(true);

    new Timer(3000, (ActionEvent e) -> toast.dispose()).start(); // disappear
after 3s
  }

  // toast popup centered on screen
  public static void showCenteredToast(String message, Window parent) {
    JWindow toast = (parent != null) ? new JWindow(parent) : new
JWindow();
    toast.setBackground(new Color(0, 0, 0, 0));
    toast.setBackground(new Color(0, 0, 0, 0));
    JLabel label = new JLabel(message);
    label.setOpaque(true);
    label.setBackground(new Color(50, 50, 50, 220));
    label.setForeground(Color.WHITE);
    label.setBorder(BorderFactory.createEmptyBorder(8, 15, 8, 15));
    label.setFont(label.getFont().deriveFont(Font.PLAIN, 14f));

    toast.add(label);
    toast.pack();

    if (parent != null) {
      int x = parent.getX() + (parent.getWidth() - toast.getWidth()) / 2;
      int y = parent.getY() + (parent.getHeight() - toast.getHeight()) / 2;
      toast.setLocation(x, y);
    } else {
      // fallback: center on screen
      Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
      int x = (screenSize.width - toast.getWidth()) / 2;
      int y = (screenSize.height - toast.getHeight()) / 2;
      toast.setLocation(x, y);
    }

    toast.setAlwaysOnTop(true); // 👈 keep above parent
    toast.setVisible(true);

    new Timer(3000, (ActionEvent e) -> toast.dispose()).start();
  }

  public static void scrollToOffset(JTextComponent area, int offset) {
    try {
      // hide caret so theme doesn't trigger
      Caret caret = area.getCaret();
      boolean wasVisible = caret.isVisible();
      caret.setVisible(false);

      area.setCaretPosition(offset); // moves view
      Rectangle2D viewRect2D = area.modelToView2D(offset);
      if (viewRect2D != null) {
        Rectangle viewRect = viewRect2D.getBounds(); // convert to
Rectangle for scrollRectToVisible
        area.scrollRectToVisible(viewRect);
      }

      caret.setVisible(wasVisible); // optional
    } catch (BadLocationException e) {
      e.printStackTrace();
    }
  }

}
```

ClosableTabTitleComponent.java

```
package com.diffchecker.components;



import javax.swing.*;

import java.awt.*;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;
```

```
import java.util.function.IntConsumer;


public class ClosableTabTitleComponent extends JPanel {

    private final Color ACTIVE_COLOR = new Color(0xF9FAFA); // Active tab color

    private final Color INACTIVE_COLOR = new Color(0x888690); // Inactive tab color

    private final Color FONT_COLOR = new Color(0xd6d6d6);


    private final JLabel titleLabel;

    private final Color HOVER_COLOR = new Color(0xd6d6d6); // your desired hover text color

    private boolean isHovered = false;

    private final JTabbedPane tabbedPane;


    public void setHovered(boolean hovered) {

        if (hovered != isHovered) {

            isHovered = hovered;

            updateColor();

        }

    }


    // FOR UPDATING THE TAB TITLE DYNAMICALLY

    public void setTitle(String title){

        titleLabel.setText(title);

        revalidate();

        repaint();

    }


    public String getTitle() {

        return titleLabel.getText();

    }


    private void updateColor(){

        int index = tabbedPane.indexOfTabComponent(this);

        if (index == tabbedPane.getSelectedIndex()) {

            titleLabel.setForeground(ACTIVE_COLOR);

        } else if (isHovered) {

            titleLabel.setForeground(HOVER_COLOR);

        } else {

            titleLabel.setForeground(INACTIVE_COLOR);

        }

    }


    /**

     * @param tabbedPane        The JTabbedPane this title belongs to

     * @param title             The text to display

     * @param onTabEmptyFallback Runnable to add a new tab if all tabs are closed

     * @param onCloseTabAtIndex  IntConsumer to handle closing the tab at a specific

     *                          index

     */

    public ClosableTabTitleComponent(JTabbedPane tabbedPane, String title, Runnable onTabEmptyFallback,

            IntConsumer onCloseTabAtIndex) {


        super(new BorderLayout(10, 0)); // add horizontal gap between label and button

        this.tabbedPane = tabbedPane;

        setOpaque(false);


        titleLabel = new JLabel(title);

        titleLabel.setHorizontalAlignment(SwingConstants.CENTER);

        titleLabel.setVerticalAlignment(SwingConstants.BOTTOM);


        // CHANGE FONT COLOR OF TAB TITLE

        // Add a listener to repaint text color when tab selection changes

        tabbedPane.addChangeListener(e -> {

            int index = tabbedPane.indexOfTabComponent(this);

            if (index != -1) {

                if (index == tabbedPane.getSelectedIndex()) {

                    titleLabel.setForeground(ACTIVE_COLOR);

                } else {

                    titleLabel.setForeground(INACTIVE_COLOR);

                }
```

```
        }
    });
    // Initial color setup (delayed to later via invokeLater to ensure index
is
    // valid)
    SwingUtilities.invokeLater(() -> {
        int index = tabbedPane.indexOfTabComponent(this);
        if (index == tabbedPane.getSelectedIndex()) {
            titleLabel.setForeground(ACTIVE_COLOR);
        } else {
            titleLabel.setForeground(INACTIVE_COLOR);
        }
    });
    // TAB TITLE FONT WEIGHT AND SIZE
    Font base = new Font("SansSerif", Font.BOLD,
titleLabel.getFont().getSize());
    titleLabel.setFont(base.deriveFont(14f));

    SwingUtilities.invokeLater(this::updateColor);
    tabbedPane.addChangeListener(e -> updateColor());

    // CLOSE BUTTON ACTION
    JButton closeButton = new JButton("×") {
        private boolean hover = false;

        {
            setBorder(BorderFactory.createEmptyBorder(0, 2, 0, 4));
            setFocusPainted(false);
            setContentAreaFilled(false);
            setOpaque(false);
            setForeground(FONT_COLOR);

            addMouseListener(new MouseAdapter() {
                @Override
                public void mouseEntered(MouseEvent e) {
                    hover = true;
                    repaint();
                }

                @Override
                public void mouseExited(MouseEvent e) {
                    hover = false;
                    repaint();
                }
            });

            setToolTipText("<html><strong>Close Tab</strong> <br> ( Ctrl
+ W )</html>");
        }

        @Override
        protected void paintComponent(Graphics g) {
            if (hover) {
                Graphics2D g2 = (Graphics2D) g.create();
                g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
                g2.setColor(new Color(80, 80, 80, 180)); // hover color
                g2.fillRoundRect(0, 0, getWidth(), getHeight(), 6, 6); //
rounded background
                g2.dispose();
            }
            super.paintComponent(g);
        }
    };
    closeButton.setFont(closeButton.getFont().deriveFont(14f));

    closeButton.addActionListener(e -> {
        int index = tabbedPane.indexOfTabComponent(this);
        if (index != -1) {
            onCloseTabAtIndex.accept(index); // close the correct tab
        }
    });

    add(titleLabel, BorderLayout.CENTER);
    add(closeButton, BorderLayout.EAST);
```

```
        setBorder(BorderFactory.createEmptyBorder(0, 5, 0, 5)); // optional: horizontal padding

    }
```

ComponentResizer.java

```
 package com.diffchecker.components;

import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.HashMap;
import java.util.Map;
import javax.swing.JComponent;
import javax.swing.SwingUtilities;

public class ComponentResizer extends MouseAdapter {

  private final static Dimension DEFAULT_MINIMUM_SIZE = new Dimension(10, 10);
  private final static Dimension DEFAULT_MAXIMUM_SIZE = new Dimension(Integer.MAX_VALUE, Integer.MAX_VALUE);
  private static final Map<Integer, Integer> cursors = new HashMap<>();

  static {
    cursors.put(1, Cursor.N_RESIZE_CURSOR);
    cursors.put(2, Cursor.W_RESIZE_CURSOR);
    cursors.put(4, Cursor.S_RESIZE_CURSOR);
    cursors.put(8, Cursor.E_RESIZE_CURSOR);
    cursors.put(3, Cursor.NW_RESIZE_CURSOR);
    cursors.put(9, Cursor.NE_RESIZE_CURSOR);
    cursors.put(6, Cursor.SW_RESIZE_CURSOR);
    cursors.put(12, Cursor.SE_RESIZE_CURSOR);
  }

  private Insets dragInsets;
  private Dimension snapSize;
  private int direction;
  private Cursor sourceCursor;
  private boolean resizing;
  private Rectangle bounds;
  private Point pressed;
  private boolean autoscrolls;

  private Dimension minimumSize = DEFAULT_MINIMUM_SIZE;
  private Dimension maximumSize = DEFAULT_MAXIMUM_SIZE;

  public ComponentResizer() {
    this(new Insets(5, 5, 5, 5), new Dimension(1, 1));
  }

  public ComponentResizer(Component... components) {
    this(new Insets(5, 5, 5, 5), new Dimension(1, 1), components);
  }

  public ComponentResizer(Insets dragInsets, Component... components)
  {
    this(dragInsets, new Dimension(1, 1), components);
  }

  public ComponentResizer(Insets dragInsets, Dimension snapSize, Component... components) {
    setDragInsets(dragInsets);
    setSnapSize(snapSize);
    registerComponent(components);
  }

  //  SAFE constructor to avoid dragInsets < minSize crash
  public ComponentResizer(Insets dragInsets, Dimension snapSize, Dimension minimumSize, Component... components) {
    setMinimumSize(minimumSize); // first
    setDragInsets(dragInsets); // then safe to call
    setSnapSize(snapSize);
    registerComponent(components);
  }

  public Insets getDragInsets() {
    return dragInsets;
  }

  public void setDragInsets(Insets dragInsets) {
    validateMinimumAndInsets(minimumSize, dragInsets);
    this.dragInsets = dragInsets;
  }

  public Dimension getMaximumSize() {
    return maximumSize;
  }

  public void setMaximumSize(Dimension maximumSize) {
    this.maximumSize = maximumSize;
  }

  public Dimension getMinimumSize(){
    return minimumSize;
  }

  public void setMinimumSize(Dimension minimumSize) {
    validateMinimumAndInsets(minimumSize, dragInsets);
    this.minimumSize = minimumSize;
  }

  public void deregisterComponent(Component... components) {
    for (Component component : components) {
      component.removeMouseListener(this);
      component.removeMouseMotionListener(this);
    }
  }

  public void registerComponent(Component... components) {
    for (Component component : components) {
      component.addMouseListener(this);
      component.addMouseMotionListener(this);
    }
  }

  public Dimension getSnapSize() {
    return snapSize;
  }

  public void setSnapSize(Dimension snapSize) {
    this.snapSize = snapSize;
  }

  private void validateMinimumAndInsets(Dimension minimum, Insets drag) {
    if (drag == null || minimum == null)
```

```
      return;
    int minW = drag.left + drag.right;
    int minH = drag.top + drag.bottom;
    if (minimum.width < minW || minimum.height < minH) {
      throw new IllegalArgumentException("Minimum size cannot be less
than drag insets");
    }
  }

  @Override
  public void mouseMoved(MouseEvent e) {
    Component source = e.getComponent();
    Point location = e.getPoint();
    direction = 0;

    if (location.x < dragInsets.left)
      direction += WEST;
    if (location.x >= source.getWidth() - dragInsets.right)
      direction += EAST;
    if (location.y < dragInsets.top)
      direction += NORTH;
    if (location.y >= source.getHeight() - dragInsets.bottom)
      direction += SOUTH;

    if (direction == 0) {
      source.setCursor(sourceCursor);
    } else {
      Integer cursorType = cursors.get(direction);
      if (cursorType != null) {
        source.setCursor(Cursor.getPredefinedCursor(cursorType));
      } else {
        source.setCursor(Cursor.getDefaultCursor());
      }
    }
  }

  @Override
  public void mouseEntered(MouseEvent e) {
    if (!resizing) {
      Component source = e.getComponent();
      sourceCursor = source.getCursor();
    }
  }

  @Override
  public void mouseExited(MouseEvent e) {
    if (!resizing) {
      Component source = e.getComponent();
      source.setCursor(sourceCursor);
    }
  }

  @Override
  public void mousePressed(MouseEvent e) {
    if (direction == 0)
      return;

    resizing = true;
    Component source = e.getComponent();
    pressed = e.getPoint();
    SwingUtilities.convertPointToScreen(pressed, source);
    bounds = source.getBounds();

    if (source instanceof JComponent jc) {
      autoscrolls = jc.getAutoscrolls();
      jc.setAutoscrolls(false);
    }
  }

  @Override
  public void mouseReleased(MouseEvent e) {
    resizing = false;
    Component source = e.getComponent();
    source.setCursor(sourceCursor);

    if (source instanceof JComponent jc) {
      jc.setAutoscrolls(autoscrolls);
    }
  }

  @Override
  public void mouseDragged(MouseEvent e) {
    if (!resizing)
      return;

    Component source = e.getComponent();
    Point dragged = e.getPoint();
    SwingUtilities.convertPointToScreen(dragged, source);
    changeBounds(source, direction, bounds, pressed, dragged);
  }

  protected void changeBounds(Component source, int direction,
Rectangle bounds, Point pressed, Point current) {
    int x = bounds.x;
    int y = bounds.y;
    int width = bounds.width;
    int height = bounds.height;

    if ((direction & WEST) != 0) {
      int drag = getDragDistance(pressed.x, current.x, snapSize.width);
      int max = Math.min(width + x, maximumSize.width);
      drag = getDragBounded(drag, snapSize.width, width,
minimumSize.width, max);
      x -= drag;
      width += drag;
    }

    if ((direction & NORTH) != 0) {
      int drag = getDragDistance(pressed.y, current.y, snapSize.height);
      int max = Math.min(height + y, maximumSize.height);
      drag = getDragBounded(drag, snapSize.height, height,
minimumSize.height, max);
      y -= drag;
      height += drag;
    }

    if ((direction & EAST) != 0) {
      int drag = getDragDistance(current.x, pressed.x, snapSize.width);
      int max = Math.min(getBoundingSize(source).width - x,
maximumSize.width);
      drag = getDragBounded(drag, snapSize.width, width,
minimumSize.width, max);
      width += drag;
    }

    if ((direction & SOUTH) != 0) {
      int drag = getDragDistance(current.y, pressed.y, snapSize.height);
      int max = Math.min(getBoundingSize(source).height - y,
maximumSize.height);
      drag = getDragBounded(drag, snapSize.height, height,
minimumSize.height, max);
      height += drag;
    }

    source.setBounds(x, y, width, height);
    source.validate();
  }

  private int getDragDistance(int larger, int smaller, int snapSize) {
    int halfway = snapSize / 2;
```

```
    int drag = larger - smaller;
    drag += (drag < 0) ? -halfway : halfway;
    drag = (drag / snapSize) * snapSize;

    return drag;
  }

  private int getDragBounded(int drag, int snapSize, int dim, int min, int
max) {
    while (dim + drag < min)
      drag += snapSize;
    while (dim + drag > max)
      drag -= snapSize;
    return drag;
  }

  private Dimension getBoundingSize(Component source) {
    if (source instanceof Window) {
      Rectangle bounds =
GraphicsEnvironment.getLocalGraphicsEnvironment().getMaximumWindo
wBounds();
      return new Dimension(bounds.width, bounds.height);
    } else {
      return source.getParent().getSize();
    }
  }

  protected static final int NORTH = 1;
  protected static final int WEST = 2;
  protected static final int SOUTH = 4;
  protected static final int EAST = 8;
}
```

CustomScrollBarUI.java

```
package com.diffchecker.components;


import javax.swing.*;

import javax.swing.plaf.basic.BasicScrollBarUI;

import java.awt.*;


public class CustomScrollBarUI extends BasicScrollBarUI {


  private Color trackColor = new Color(0x17181C); // default dark

  private Color thumbHoverColor = new Color(0x8B8B8B); // Darker on
hover

  private Color thumbColor = new Color(0x636363); // Light gray


  public void setTrackColor(Color c) {

    this.trackColor = c;

  }


  // THINNER SCROLLBAR

  @Override

  protected Dimension getMinimumThumbSize() {

    // Ensure thumb is still visible and usable

    return new Dimension(30, 30); // Width/Height depending on orientation

  }


  @Override

  public Dimension getPreferredSize(JComponent c) {

    if (scrollbar.getOrientation() == JScrollBar.VERTICAL) {

      return new Dimension(10, super.getPreferredSize(c).height); // 🔧
thinner vertical bar

    } else {

      return new Dimension(super.getPreferredSize(c).width, 10); // 🔧
thinner horizontal bar

    }

  }


  @Override

  protected void installDefaults() {

    super.installDefaults();

    scrollbar.setBorder(BorderFactory.createEmptyBorder());

  }


  // SPACE AROUND THE SCROLLBAR

  @Override

  protected void paintThumb(Graphics g, JComponent c, Rectangle
thumbBounds) {

    Graphics2D g2 = (Graphics2D) g.create();

    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);


    g2.setColor(isThumbRollover() ? thumbHoverColor : thumbColor);
```

```
    // Add some inset (shrink the thumb a bit)
    int inset = 1;
    int arc = 10;
    g2.fillRoundRect(thumbBounds.x + inset, thumbBounds.y + inset,
        thumbBounds.width - 2 * inset, thumbBounds.height - 2 * inset, arc,
arc);
    g2.dispose();
  }

  @Override
  protected void paintTrack(Graphics g, JComponent c, Rectangle
trackBounds) {
    Graphics2D g2 = (Graphics2D) g.create();
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

    g2.setColor(trackColor); // <- use current theme
    g2.fillRect(trackBounds.x, trackBounds.y, trackBounds.width,
trackBounds.height);

    g2.dispose();
  }

  @Override
  protected JButton createDecreaseButton(int orientation) {
    return createZeroButton();
  }

  @Override
  protected JButton createIncreaseButton(int orientation) {
    return createZeroButton();
  }

  private JButton createZeroButton() {
    JButton button = new JButton();
    button.setPreferredSize(new Dimension(0, 0));
    button.setMinimumSize(new Dimension(0, 0));
    button.setMaximumSize(new Dimension(0, 0));
    button.setVisible(false); // <- Important: Make it invisible
    button.setOpaque(false); // <- Optional: remove background
    button.setContentAreaFilled(false); // <- Optional: remove fill
    button.setBorderPainted(false); // <- Optional: remove border
    return button;
  }
}
```

CustomTitleBar.java

```
package com.diffchecker.components;

import javax.swing.*;

import org.fife.ui.rsyntaxtextarea.SyntaxConstants;

import com.formdev.flatlaf.extras.FlatSVGIcon;

import java.awt.*;
import java.awt.event.*;
import java.util.Map;
import java.util.Set;

public class CustomTitleBar extends JPanel {

  private final JFrame frame;
  private final JLabel titleLabel;
  private final JButton closeButton;
  private final JButton minimizeButton;
  private final JButton maximizeButton;
  private final JPanel controlPanel;

  private final Color FONT_COLOR = new Color(0xd6d6d6);

  private Dimension previousSize;
```

```
  // SYNTAX STYLES

  private static final Map<String, String> SYNTAX_STYLES =
Map.ofEntries(

      Map.entry("None", SyntaxConstants.SYNTAX_STYLE_NONE),

      Map.entry("Java", SyntaxConstants.SYNTAX_STYLE_JAVA),

      Map.entry("JavaScript",
SyntaxConstants.SYNTAX_STYLE_JAVASCRIPT),

      Map.entry("Dart", SyntaxConstants.SYNTAX_STYLE_DART),

      Map.entry("TypeScript",
SyntaxConstants.SYNTAX_STYLE_TYPESCRIPT),

      Map.entry("CSS", SyntaxConstants.SYNTAX_STYLE_CSS),

      Map.entry("SQL", SyntaxConstants.SYNTAX_STYLE_SQL),

      Map.entry("Python", SyntaxConstants.SYNTAX_STYLE_PYTHON),

      Map.entry("C", SyntaxConstants.SYNTAX_STYLE_C),

      Map.entry("C++", SyntaxConstants.SYNTAX_STYLE_CPLUSPLUS),

      Map.entry("C#", SyntaxConstants.SYNTAX_STYLE_CSHARP),

      Map.entry("HTML", SyntaxConstants.SYNTAX_STYLE_HTML),

      Map.entry("XML", SyntaxConstants.SYNTAX_STYLE_XML),

      Map.entry("JSON", SyntaxConstants.SYNTAX_STYLE_JSON),

      Map.entry("YAML", SyntaxConstants.SYNTAX_STYLE_YAML),

      Map.entry("PHP", SyntaxConstants.SYNTAX_STYLE_PHP),

      Map.entry("Ruby", SyntaxConstants.SYNTAX_STYLE_RUBY),

      Map.entry("Kotlin", SyntaxConstants.SYNTAX_STYLE_KOTLIN),

      Map.entry("Dockerfile",
SyntaxConstants.SYNTAX_STYLE_DOCKERFILE),

      Map.entry("Go", SyntaxConstants.SYNTAX_STYLE_GO),

      Map.entry("Markdown",
SyntaxConstants.SYNTAX_STYLE_MARKDOWN));


  // BUTTON COLOR AND HOVER COLOR

  private static final Color BTN_COLOR_DARKER = new
Color(0x00744d);

  private static final Color BTN_COLOR_BLACK = new Color(0x242526);


  // package-level config

  private static String PACKAGE_NAME;


  // Lazy-injected tabs reference (set by Main after tabs are created)

  private JTabbedPane tabs;

  public CustomTitleBar(JFrame frame,

      String title,

      String packageName,

      String iconPath,

      Color background,

      int height) {


    initUI();


    // —— store static config so createButton() can access it

    PACKAGE_NAME = packageName;


    this.frame = frame;

    this.previousSize = frame.getSize();


    setLayout(new BorderLayout());

    setBackground(background);


    // fixed height; allow width to stretch but avoid Integer.MAX_VALUE
weirdness

    Dimension fixedSize = new Dimension(new
Dimension(Integer.MAX_VALUE, 33));

    setPreferredSize(fixedSize);

    setMaximumSize(fixedSize);

    setMinimumSize(new Dimension(0, height));


    // —— Title label (optional icon) ----------------------------------------

    titleLabel = new JLabel(title);

    titleLabel.setFont(new Font("SansSerif", Font.BOLD, 16));

    titleLabel.setForeground(FONT_COLOR);

    titleLabel.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));

    titleLabel.setVerticalAlignment(SwingConstants.CENTER);


    if (iconPath != null) {

      ImageIcon icon = new ImageIcon(getClass().getResource(iconPath));

      titleLabel.setIcon(icon);
```

```
    titleLabel.setIconTextGap(10);
  }

  // ── Control buttons ──────────────────────────────────────────
  controlPanel = new JPanel();
  controlPanel.setLayout(new BoxLayout(controlPanel,
BoxLayout.X_AXIS));
  controlPanel.setOpaque(false);

  minimizeButton = createButton("minimize_def.png",
"minimize_hover.png",
      e -> frame.setState(JFrame.ICONIFIED));

  maximizeButton = createButton("maximize_def.png",
"maximize_hover.png",
      e -> toggleMaximize());

  closeButton = createButton("close_def.png", "close_hover.png",
      e -> frame.dispatchEvent(new java.awt.event.WindowEvent(
          frame, java.awt.event.WindowEvent.WINDOW_CLOSING)));

  RoundedButton menuButton = new RoundedButton();
  FlatSVGIcon menuIcon = new
FlatSVGIcon("diffchecker/images/icons/menu.svg", 20, 20);
  menuIcon.setColorFilter(new FlatSVGIcon.ColorFilter(c ->
Color.WHITE));
  menuButton.setIcon(menuIcon);
  menuButton.setHorizontalTextPosition(SwingConstants.LEFT); // text
after icon
  menuButton.setIconTextGap(4);
  menuButton.setHorizontalAlignment(SwingConstants.CENTER);
  menuButton.setVerticalAlignment(SwingConstants.CENTER);
  menuButton.setVerticalTextPosition(SwingConstants.NORTH);
  menuButton.setBackgroundColor(BTN_COLOR_BLACK);
  menuButton.setHoverBackgroundColor(BTN_COLOR_DARKER);
  menuButton.setBorderColor(BTN_COLOR_BLACK);
  menuButton.setHoverBorderColor(BTN_COLOR_DARKER);
  menuButton.setBorderThickness(2);
  menuButton.setCornerRadius(10);
  menuButton.setMargin(new Insets(0, 0, 0, 0));
  menuButton.setFont(menuButton.getFont().deriveFont(14f));

  // Example popup menu
  JPopupMenu popup = new JPopupMenu();
  popup.add(createSyntaxMenu());

  menuButton.addActionListener(e -> popup.show(menuButton, 0,
menuButton.getHeight()));

  controlPanel.add(minimizeButton);
  controlPanel.add(maximizeButton);
  controlPanel.add(closeButton);

  JPanel rightPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT,
0, 0));
  rightPanel.setOpaque(false);
  rightPanel.add(controlPanel); // add minimize/maximize/close

  JPanel centerPanel = new JPanel();
  centerPanel.setOpaque(false);
  centerPanel.setLayout(new BoxLayout(centerPanel,
BoxLayout.X_AXIS));

  centerPanel.add(titleLabel);
  centerPanel.add(Box.createRigidArea(new Dimension(10, 0)));
  centerPanel.add(menuButton);

  add(centerPanel, BorderLayout.CENTER);
  add(rightPanel, BorderLayout.EAST);

  // ── Enable dragging ──────────────────────────────────────────
  new TitlebarMover(
      frame,
      this,
      this::toggleMaximize,
      () -> {
        updateMaximizeIcon();
```

```
        previousSize = frame.getSize(); // <-- capture restored size again
      });
}

private void initUI() {
  // existing UI setup if needed
}

// allow Main to inject tabs after they are created
public void setTabs(JTabbedPane tabs) {
  this.tabs = tabs;
}

// resolve the currently selected SplitTextTabPanel (handles wrappers)
private SplitTextTabPanel activeSplitPanel() {
  if (tabs == null)
    return null;
  Component c = tabs.getSelectedComponent();
  if (c instanceof SplitTextTabPanel s1)
    return s1;

  // If you wrap panels in scroll panes or other containers:
  if (c instanceof JScrollPane sp) {
    Component view = sp.getViewport().getView();
    if (view instanceof SplitTextTabPanel s2)
      return s2;
  }
  return null;
}

// ---------------------------------------------------------------------- utilities
private JButton createButton(String defIcon,
    String hoverIcon,
    ActionListener action) {

  JButton templateButton = new JButton(new ImageIcon(
      getClass().getResource("/" + PACKAGE_NAME + "/images/" +
defIcon)));
    int size = 32; // icon size
    templateButton.setPreferredSize(new Dimension(size, size));
    templateButton.setBorderPainted(false);
    templateButton.setFocusPainted(false);
    templateButton.setContentAreaFilled(false);
    templateButton.setToolTipText(defIcon.split("_")[0]); // quick tooltip

    templateButton.addActionListener(action);
    templateButton.addMouseListener(new MouseAdapter() {
      @Override
      public void mouseEntered(MouseEvent e) {
        templateButton.setIcon(new ImageIcon(getClass().getResource(
            "/" + PACKAGE_NAME + "/images/" + hoverIcon)));
      }

      @Override
      public void mouseExited(MouseEvent e) {
        templateButton.setIcon(new ImageIcon(getClass().getResource(
            "/" + PACKAGE_NAME + "/images/" + defIcon)));
      }
    });
    return templateButton;
  }

  // ———————————————————————————————————————— SYNTAX
MENU ————————————————————————————————————————
  // Dynamically create menu items from SYNTAX_STYLES map
  private JMenu createSyntaxMenu() {
    JMenu syntaxHighlighting = new JMenu("Syntax Highlighting");

    // Helper to add an item that applies to the active tab at click time
    java.util.function.BiConsumer<String, String> addItem = (label,
styleConst) -> {
      JMenuItem item = new JMenuItem(label);
      item.addActionListener(e -> {
        SyntaxManager.setSyntax(styleConst);
```

```
    });
    syntaxHighlighting.add(item);
  };

  // Favorites first
  addItem.accept("None", SYNTAX_STYLES.get("None"));
  addItem.accept("Java", SYNTAX_STYLES.get("Java"));
  addItem.accept("JavaScript", SYNTAX_STYLES.get("JavaScript"));
  addItem.accept("Dart", SYNTAX_STYLES.get("Dart"));
  addItem.accept("TypeScript", SYNTAX_STYLES.get("TypeScript"));
  addItem.accept("CSS", SYNTAX_STYLES.get("CSS"));
  addItem.accept("SQL", SYNTAX_STYLES.get("SQL"));

  Set<String> favorites = Set.of("None", "Java", "JavaScript", "Dart",
"TypeScript", "CSS", "SQL");

  // 2. Add the rest (skip "None" and favorites already added)
  for (Map.Entry<String, String> entry : SYNTAX_STYLES.entrySet()) {
    if (favorites.contains(entry.getKey()))
      continue;
    addItem.accept(entry.getKey(), entry.getValue());
  }

  return syntaxHighlighting;
 }

 // ---------------------------------------------------------------- maximize
 // logic
 private void toggleMaximize() {
  if ((frame.getExtendedState() & JFrame.MAXIMIZED_BOTH) ==
JFrame.MAXIMIZED_BOTH) {
    frame.setExtendedState(JFrame.NORMAL);
    frame.setSize(previousSize);
  } else {
    previousSize = frame.getSize();
    frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
  }
  updateMaximizeIcon();
 }

 private void updateMaximizeIcon() {
  boolean maximized = (frame.getExtendedState() &
JFrame.MAXIMIZED_BOTH) == JFrame.MAXIMIZED_BOTH;

  String def = maximized ? "collapse_def.png" : "maximize_def.png";
  String hover = maximized ? "collapse_hover.png" :
"maximize_hover.png";
  maximizeButton.setIcon(new ImageIcon(
    getClass().getResource("/" + PACKAGE_NAME + "/images/" + def)));

  // refresh hover behaviour
  for (MouseListener ml : maximizeButton.getMouseListeners()) {
   if (ml instanceof MouseAdapter)
    maximizeButton.removeMouseListener(ml);
  }
  maximizeButton.addMouseListener(new MouseAdapter() {
   @Override
   public void mouseEntered(MouseEvent e) {
    maximizeButton.setIcon(new ImageIcon(getClass().getResource(
      "/" + PACKAGE_NAME + "/images/" + hover)));
   }

   @Override
   public void mouseExited(MouseEvent e) {
    maximizeButton.setIcon(new ImageIcon(getClass().getResource(
      "/" + PACKAGE_NAME + "/images/" + def)));
   }
  });
 }
}
```

FindReplaceSupport.java

```
package com.diffchecker.components;

import org.fife.ui.rsyntaxtextarea.RSyntaxTextArea;
import org.fife.ui.rtextarea.SearchContext;
import org.fife.ui.rtextarea.SearchEngine;
import org.fife.ui.rtextarea.SearchResult;
import org.fife.rsta.ui.search.ReplaceDialog;
import org.fife.rsta.ui.search.SearchEvent;
import org.fife.rsta.ui.search.SearchListener;

import javax.swing.*;

public class FindReplaceSupport {
    private final ReplaceDialog replaceDialog;
    private final SearchContext searchContext;

    public FindReplaceSupport(JFrame parentFrame, RSyntaxTextArea
textArea) {
        this.searchContext = new SearchContext();
        this.searchContext.setSearchWrap(true);

        SearchListener listener = new SearchListener() {
            @Override
            public void searchEvent(SearchEvent e) {
                SearchResult result = switch (e.getType()) {
                    case FIND -> SearchEngine.find(textArea,
e.getSearchContext());
                    case REPLACE -> SearchEngine.replace(textArea,
e.getSearchContext());
                    case REPLACE_ALL -> SearchEngine.replaceAll(textArea,
e.getSearchContext());
                    case MARK_ALL -> SearchEngine.markAll(textArea,
e.getSearchContext());
                };
                if (result != null && !result.wasFound()) {
                    UIManager.getLookAndFeel().provideErrorFeedback(textAre
a);
                }
            }

            @Override
            public String getSelectedText() {
                return textArea.getSelectedText();
            }
        };

        this.replaceDialog = new ReplaceDialog(parentFrame, listener);
        this.replaceDialog.setSearchContext(searchContext);

        // Register shortcuts
        InputMap im = textArea.getInputMap();
        ActionMap am = textArea.getActionMap();

        im.put(KeyStroke.getKeyStroke("control F"), "Replace");
        im.put(KeyStroke.getKeyStroke("control H"), "Replace");

        am.put("Replace", getReplaceAction());
    }

    public void showReplaceDialog() {
        replaceDialog.setVisible(true);
    }

    /**  Extracted Action so both keyboard and button can use the same
behavior */
    public Action getReplaceAction() {
        return new AbstractAction(){
            @Override
            public void actionPerformed(java.awt.event.ActionEvent e) {
                showReplaceDialog();
            }
        };
    }
}
```

RoundedButton.java

```
package com.diffchecker.components;

import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class RoundedButton extends JButton {
    private Color backgroundColor = new Color(0x00C281);
    private Color hoverBackgroundColor = new Color(0x009966);
    private Color textColor = new Color(0xeeeeee); // FONT COLOR
    private Color borderColor = new Color(0x00C281);
    private Color hoverBorderColor = new Color(0x007a4f); // New: hover
border
    private int cornerRadius = 20;
    private int borderThickness = 2;
    private boolean hovered = false;

    // SELECTED STATE (FOR TOGGLE BUTTONS)
    private boolean selected = false;
    private Color activeBackgroundColor = new Color(0x009966); // when
ON
    private Color activeBorderColor = new Color(0x009966); // optional
border for ON state

    public RoundedButton() {
        this("");
    }

    public RoundedButton(String text) {
        super(text);
        setFont(new Font("SansSerif", Font.BOLD, 14));
        setFocusPainted(false);
        setBorderPainted(false);
        setContentAreaFilled(false);
        setOpaque(false);
        setForeground(textColor);
        setMargin(new Insets(5, 10, 5, 10));
        setCursor(Cursor.getDefaultCursor());

        // CENTER THE TEXT
        setHorizontalAlignment(SwingConstants.CENTER);
        setVerticalAlignment(SwingConstants.CENTER);
        setHorizontalTextPosition(SwingConstants.CENTER);
        setVerticalTextPosition(SwingConstants.CENTER);

        // Hover detection
        addMouseListener(new MouseAdapter() {
```

```
        @Override
        public void mouseEntered(MouseEvent e) {
            hovered = true;
            setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSO
R));
            repaint();
        }

        @Override
        public void mouseExited(MouseEvent e) {
            hovered = false;
            setCursor(Cursor.getDefaultCursor());
            repaint();
        }
    });
  }

  // ——— Setters
——————————————————————————————————————————

  public void setCornerRadius(int radius) {
    this.cornerRadius = radius;
    repaint();
  }

  public void setBackgroundColor(Color color) {
    this.backgroundColor = color;
    repaint();
  }

  public void setHoverBackgroundColor(Color color) {
    this.hoverBackgroundColor = color;
    repaint();
  }

  public void setBorderColor(Color color) {
    this.borderColor = color;
    repaint();
  }

  public void setHoverBorderColor(Color color) {
    this.hoverBorderColor = color;
    repaint();
  }

  public void setBorderThickness(int thickness) {
    this.borderThickness = thickness;
    repaint();
  }

  public void setTextColor(Color color) {
    this.textColor = color;
    setForeground(color);
    repaint();
  }

  // ——— Paint
——————————————————————————————————————————

  @Override
  protected void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g.create();
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

    if (selected) {
      g2.setColor(hovered ? activeBackgroundColor.darker() :
activeBackgroundColor);
    } else {
      g2.setColor(hovered ? hoverBackgroundColor : backgroundColor);
    }

    g2.fillRoundRect(0, 0, getWidth(), getHeight(), cornerRadius,
cornerRadius);
    super.paintComponent(g);
    g2.dispose();
  }

  public void setSelectedState(boolean selected) {
    this.selected = selected;
    repaint();
  }

  public boolean isSelectedState() {
    return selected;
  }

  public void setActiveBackgroundColor(Color color) {
    this.activeBackgroundColor = color;
    repaint();
  }

  public void setActiveBorderColor(Color color) {
    this.activeBorderColor = color;
    repaint();
  }

  @Override
  protected void paintBorder(Graphics g) {
    Graphics2D g2 = (Graphics2D) g.create();
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

    if (selected) {
      g2.setColor(hovered ? activeBorderColor.darker() :
activeBorderColor);
    } else {
      g2.setColor(hovered ? hoverBorderColor : borderColor);
    }

    g2.setStroke(new BasicStroke(borderThickness));
    g2.drawRoundRect(borderThickness / 2, borderThickness / 2,
        getWidth() - borderThickness, getHeight() - borderThickness,
        cornerRadius, cornerRadius);

    g2.dispose();
  }
}
```

RoundedTabbedPaneUI.java

```
package com.diffchecker.components;

import javax.swing.JComponent;
import javax.swing.plaf.basic.BasicTabbedPaneUI;
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;

public class RoundedTabbedPaneUI extends BasicTabbedPaneUI {

  private final Color selectedColor = new Color(0x363839);
  private final Color unselectedColor = new Color(0x242526);
  private final int arc = 6; // Increase for more roundness
```

```
  private final Color hoverColor = new Color(0x00744d);
  private int hoveredTabIndex = -1;

  private TabHoverListener hoverListener;

  public void setHoverListener(TabHoverListener listener) {
    this.hoverListener = listener;
  }

  public interface TabHoverListener {
    void onTabHoverChanged(int hoveredIndex);
  }

  // This controls space around the tabs:
  @Override
  protected void installDefaults() {
    super.installDefaults();
    tabInsets = new Insets(6, 6, 6, 6); // top, left, bottom, right (more
padding)
    tabAreaInsets = new Insets(5, 0, 5, 0); // space around tab area
    contentBorderInsets = new Insets(0, 0, 0, 0);
  }

  @Override
  protected Insets getTabInsets(int tabPlacement, int tabIndex) {
    return new Insets(6, 12, 6, 12); // Adjust horizontal space (left/right)
  }

  @Override
  public void installUI(JComponent c) {
    super.installUI(c); // Initializes tabPane
    tabPane.addMouseMotionListener(new MouseMotionAdapter() {
      @Override
      public void mouseMoved(MouseEvent e) {
        int tab = getTabAtLocation(e.getX(), e.getY());
        if (tab != hoveredTabIndex) {
          hoveredTabIndex = tab;
          if (hoverListener != null) {
            hoverListener.onTabHoverChanged(hoveredTabIndex);
          }
          tabPane.repaint();
        }
      }

      @Override
      public void mouseDragged(MouseEvent e) {
        if (hoveredTabIndex != -1) {
          hoveredTabIndex = -1;
          tabPane.repaint();
        }
      }
    });
  }

  private int getTabAtLocation(int x, int y) {
    for (int i = 0; i < tabPane.getTabCount(); i++) {
      Rectangle rect = getTabBounds(tabPane, i);
      if (rect.contains(x, y)) {
        return i;
      }
    }
    return -1;
  }

  @Override
  protected void paintTabBackground(Graphics g, int tabPlacement,
      int tabIndex, int x, int y, int w, int h, boolean isSelected) {

    Graphics2D g2 = (Graphics2D) g.create();
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

    Color bg;
    int bgY = y;
    int bgH = h;

    if (isSelected) {
      bg = selectedColor;
      bgY = y + 5;
      bgH = h - 10;
    } else if (tabIndex == hoveredTabIndex) {
      bg = hoverColor;
      bgY = y + 3;
      bgH = h - 4;
    } else {
      bg = unselectedColor;
      bgY = y + 5;
      bgH = h - 10;
    }

    g2.setColor(bg);
    g2.fillRoundRect(x + 2, bgY, w - 4, bgH, arc, arc);
    g2.dispose();
  }

  @Override
  protected void paintContentBorder(Graphics g, int tabPlacement,
      int selectedIndex) {
    // Optional: don't paint a border
  }

  @Override
  protected void paintFocusIndicator(Graphics g, int tabPlacement,
      Rectangle[] rects, int tabIndex,
      Rectangle iconRect, Rectangle textRect, boolean isSelected) {
    // Do not paint focus indicator
  }

  @Override
  protected void paintTabBorder(Graphics g, int tabPlacement,
      int tabIndex, int x, int y, int w, int h, boolean isSelected) {
    if (!isSelected)
      return;

    Graphics2D g2 = (Graphics2D) g.create();
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

    g2.setColor(new Color(0x5A5A5A)); // border color
    int arc = 6;
    g2.setStroke(new BasicStroke(1.5f));
    g2.drawRoundRect(x + 2, y + 5, w - 4, h - 10, arc, arc);

    g2.dispose();
  }

}
```

SplitTextTabPanel.java

```
package com.diffchecker.components;

import javax.swing.*;

import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.text.BadLocationException;
```

```
import com.diffchecker.components.Database.DB;
import com.diffchecker.components.Database.DiffData;
import com.diffchecker.components.Database.DiffRepository;
import com.diffchecker.components.Helper.EditorUtils;
import com.github.difflib.DiffUtils;
import com.github.difflib.patch.AbstractDelta;
import com.github.difflib.patch.Patch;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.FocusAdapter;
import java.awt.event.FocusEvent;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;
import java.awt.event.MouseAdapter;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

// RSyntaxTextArea dependencies
import org.fife.ui.rsyntaxtextarea.*;
import org.fife.ui.rtextarea.RTextScrollPane;

// RSyntaxTextArea search and replace dependencies
import com.formdev.flatlaf.extras.FlatSVGIcon;

public class SplitTextTabPanel extends JPanel implements
ThemedComponent, SyntaxHighlightable {
    // Keep the XML's size
    private static final int sizeFromXML = 15; // match your <baseFont
size="15"/>

    // FOR CREATING NEW TABS
    private final Runnable newTabCallback; // ✔ store callback

    // Active highlight colors (switch based on theme)
    private Color lineRemovedColor;
    private Color lineAddedColor;
    private Color wordRemovedColor;
    private Color wordAddedColor;

    // WORD HIGHLIGHT
    private static final Color LINE_REMOVED_DARK = new
Color(0x40191D);
    private static final Color LINE_ADDED_DARK = new Color(0x12342B);
    private static final Color WORD_REMOVED_DARK = new
Color(0x6f1817);
    private static final Color WORD_ADDED_DARK = new
Color(0x0f6248);

    private static final Color LINE_REMOVED_LIGHT = new
Color(0xFCC7C7);
    private static final Color LINE_ADDED_LIGHT = new
Color(0xADECD5);
    private static final Color WORD_REMOVED_LIGHT = new
Color(0xFBA5A5);
    private static final Color WORD_ADDED_LIGHT = new
Color(0x71DEB9);

    // BORDER COLORS
    private static final Color EDITOR_BORDER_COLOR_DARK = new
Color(0x242526);
    private static final Color ACTIVE_BORDER_COLOR_DARK = new
Color(0x00744d);

    private static final Color EDITOR_BORDER_COLOR_LIGHT = new
Color(0xdddddd);
    private static final Color ACTIVE_BORDER_COLOR_LIGHT = new
Color(0x00af74);

    // DARK AND LIGHT MODE BACKGROUND COLORS
    private final Color BACKGROUND_DARK = new Color(0x17181C);
    private final Color BACKGROUND_LIGHT = new Color(0xD6D8DF);
    // private final Color BACKGROUND_TEST = new Color(0x04395E);

    // SCROLLBAR CORNER COLORS
    private final Color SCROLLBAR_CORNER_DARK = new
Color(0x17181C);
    private final Color SCROLLBAR_CORNER_LIGHT = new
Color(0xE6E7ED);

    // EDITOR LIGHT THEME SCROLLBAR TRACK COLOR
    private final Color SCROLLBAR_TRACK_DARK = new
Color(0x17181C);
    private final Color SCROLLBAR_TRACK_LIGHT = new
Color(0xE6E7ED);

    // BUTTON COLOR AND HOVER COLOR
    private static final Color BTN_COLOR = new Color(0x00af74);
    private static final Color BTN_COLOR_DARKER = new
Color(0x00744d);
    private static final Color BTN_COLOR_BLACK = new Color(0x242526);

    // DEFAULT DECLARATIONS
    private RSyntaxTextArea jt1;
    private RSyntaxTextArea jt2;

    private final RTextScrollPane scroll1;
    private final RTextScrollPane scroll2;

    private FindReplaceSupport findReplace1;
    private FindReplaceSupport findReplace2;

    // TOGGLES
    private RoundedButton wordHighlightToggleBtn;
    private RoundedButton wordWrapToggleBtn;
    private RoundedButton lineHighlightToggleBtn;
    private RoundedButton themeToggleBtn;
    private RoundedButton diffcheckBtn;

    // SCROLLBAR CORNER PANELS TO REMOVE WHITE SQUARES
    JPanel scroll1CornerLeft;
    JPanel scroll2CornerLeft;
    JPanel scroll1CornerRight;
    JPanel scroll2CornerRight;

    // EDITOR AND BUTTON BACKGROUND PANELS
    JPanel leftButtonPanel;
    JPanel centerButtonPanel;
    JPanel rightButtonPanel;

    // PANELS DIRECTLY SURROUNDING THE TEXT EDITORS (LIKE
MARGINS)
    JPanel sideBySidePanel;
    JPanel contentPanel;
    JPanel bottomPanel;

    // CHECKING IF GREEN BORDER IS ACTIVE OR NOT
    private boolean jt1IsActive = false;
    private boolean jt2IsActive = false;

    // THEME MANAGEMENT
    private boolean darkThemeEnabled = true;

    // FOR NAVIGATING DIFFS
    private final List<DiffGroup> diffGroups = new ArrayList<>();
    private int currentGroupIndex = -1;
```

```
    private DiffData currentDiff; // to track the saved diff record

    private static class DiffGroup {
        EditorUtils.HighlightInfo left;
        EditorUtils.HighlightInfo right;
    }

    @Override
    public void applySyntaxStyle(String syntaxStyle) {
        jt1.setSyntaxEditingStyle(syntaxStyle);
        jt2.setSyntaxEditingStyle(syntaxStyle);
    }

    // TOGGLE WORD HIGHLIGHT
    private boolean wordHighlightEnabled = false;

    // TOGGLE LINE WRAP
    private boolean lineHighlightEnabled = true;

    // TOGGLE WORD WRAP
    private boolean wordWrapEnabled = false;

    private RSyntaxTextArea lastFocusedEditor;

    private final FocusAdapter trackFocus = new FocusAdapter() {
        @Override
        public void focusGained(FocusEvent e) {
            if (e.getComponent() instanceof RSyntaxTextArea) {
                lastFocusedEditor = (RSyntaxTextArea) e.getComponent();
            }
        }
    };

    // TRACKING UNSAVED CHANGES
    private boolean isDirty = false;

    // DOCUMENT LISTENERS TO TRACK CHANGES
    private void markDirty() {
        if (!isDirty) {
            isDirty = true;
            // Append * to tab title
            Container parent = getParent();
            while (parent != null && !(parent instanceof JTabbedPane)) {
                parent = parent.getParent();
            }
            if (parent instanceof JTabbedPane) {
                JTabbedPane tabbedPane = (JTabbedPane) parent;
                int index = tabbedPane.indexOfComponent(this);
                if (index != -1) {
                    String title = tabbedPane.getTitleAt(index);
                    if (!title.endsWith("*")) {
                        tabbedPane.setTitleAt(index, title + "*");
                    }
                }
            }
        }
    }

    private void markSaved() {
        isDirty = false;
    }

    public boolean hasUnsavedChanges() {
        return isDirty;
    }

    private final DocumentListener dirtyListener1 = new
DocumentListener() {
        public void insertUpdate(DocumentEvent e) {
            markDirty();
        }

        public void removeUpdate(DocumentEvent e) {
            markDirty();
        }

        public void changedUpdate(DocumentEvent e) {
            markDirty();
        }
    };

    private final DocumentListener dirtyListener2 = new
DocumentListener() {
        public void insertUpdate(DocumentEvent e) {
            markDirty();
        }

        public void removeUpdate(DocumentEvent e) {
            markDirty();
        }

        public void changedUpdate(DocumentEvent e) {
            markDirty();
        }
    };

    @Override
    public void addNotify() {
        super.addNotify();
        if (findReplace1 == null || findReplace2 == null) {
            JFrame frame = (JFrame) SwingUtilities.getWindowAncestor(this);
            if (frame != null) {
                findReplace1 = new FindReplaceSupport(frame, jt1);
                findReplace2 = new FindReplaceSupport(frame, jt2);
            }
        }
    }

    public SplitTextTabPanel(Runnable newTabCallback) {
        setLayout(new BorderLayout());
        setKeyboardShortcuts();

        // FOR ADDING NEW TAB
        this.newTabCallback = newTabCallback; // or your existing setup
code

        // DECLARE TEXT AREAS
        jt1 = EditorUtils.createRSyntaxArea();
        jt2 = EditorUtils.createRSyntaxArea();

        jt1.setCodeFoldingEnabled(true);
        jt2.setCodeFoldingEnabled(true);

        // TRACK EDITOR CHANGES
        jt1.getDocument().addDocumentListener(dirtyListener1);
        jt2.getDocument().addDocumentListener(dirtyListener2);

        // TRACK FOCUS
        jt1.addFocusListener(trackFocus);
        jt2.addFocusListener(trackFocus);

        // DISABLE CURRENT LINE HIGHLIGHTING SINCE IT CLASHES
WITH DIFF HIGHLIGHT
        jt1.setHighlightCurrentLine(false);
        jt2.setHighlightCurrentLine(false);

        // Disable mark occurrences (highlights and tooltips for matching
tokens)
        jt1.setMarkOccurrences(false);
        jt2.setMarkOccurrences(false);
```

```
        // Disable matched bracket popup tooltips
        jt1.setBracketMatchingEnabled(false);
        jt2.setBracketMatchingEnabled(false);

        scroll1 = new RTextScrollPane(jt1);
        scroll2 = new RTextScrollPane(jt2);

        // REMOVING THE WHITE SQUARES AT THE INTERSCTION OF
THE SCROLLBARS
        scroll1CornerLeft = new JPanel();
        scroll2CornerLeft = new JPanel();
        scroll1CornerRight = new JPanel();
        scroll2CornerRight = new JPanel();
        // Set corners for scroll1
        scroll1.setCorner(JScrollPane.LOWER_LEFT_CORNER,
scroll1CornerLeft);
        scroll1.setCorner(JScrollPane.LOWER_RIGHT_CORNER,
scroll1CornerRight);
        // Set corners for scroll2
        scroll2.setCorner(JScrollPane.LOWER_LEFT_CORNER,
scroll2CornerLeft);
        scroll2.setCorner(JScrollPane.LOWER_RIGHT_CORNER,
scroll2CornerRight);

        // CUSTOM SCROLLBARS
        scroll1.getVerticalScrollBar().setUI(new CustomScrollBarUI());
        scroll2.getVerticalScrollBar().setUI(new CustomScrollBarUI());
        scroll1.getHorizontalScrollBar().setUI(new CustomScrollBarUI());
        scroll2.getHorizontalScrollBar().setUI(new CustomScrollBarUI());

        scroll1.getHorizontalScrollBar().setOpaque(true);
        scroll2.getHorizontalScrollBar().setOpaque(true);
        scroll1.getVerticalScrollBar().setOpaque(true);
        scroll2.getVerticalScrollBar().setOpaque(true);

        scroll1.setOpaque(false);
        scroll1.getViewport().setOpaque(false);

        scroll2.setOpaque(false);
        scroll2.getViewport().setOpaque(false);

        // REMOVE DEFAULT BORDERS
        jt1.setBorder(BorderFactory.createEmptyBorder());
        jt2.setBorder(BorderFactory.createEmptyBorder());

        // SYNCHRONIZED VERTICAL SCROLLING
        JScrollBar vBar1 = scroll1.getVerticalScrollBar();
        JScrollBar vBar2 = scroll2.getVerticalScrollBar();
        vBar1.addAdjustmentListener(e -> {
            if (vBar2.getValue() != vBar1.getValue()) {
                vBar2.setValue(vBar1.getValue());
            }
        });

        vBar2.addAdjustmentListener(e -> {
            if (vBar1.getValue() != vBar2.getValue()) {
                vBar1.setValue(vBar2.getValue());
            }
        });

        scroll1.setBorder(null);
        scroll2.setBorder(null);

        JPanel p1 = new JPanel(new BorderLayout());
        p1.add(scroll1, BorderLayout.CENTER);

        JPanel p2 = new JPanel(new BorderLayout());
        p2.add(scroll2, BorderLayout.CENTER);

        // SIDE BY SIDE TEXT AREAS
        sideBySidePanel = new JPanel(new GridLayout(1, 2, 10, 0)); // 10px
gap between areas
        sideBySidePanel.add(p1);
        sideBySidePanel.add(p2);

        // add(splitPane, BorderLayout.CENTER);
        contentPanel = new JPanel(new BorderLayout());
        contentPanel.setBorder(BorderFactory.createEmptyBorder(10, 10,
10, 10)); // top, left, bottom, right
        contentPanel.add(sideBySidePanel, BorderLayout.CENTER);

        add(contentPanel, BorderLayout.CENTER);

        // CUSTOM BUTTON
        diffcheckBtn = new RoundedButton("Find Difference");
        diffcheckBtn.setBackgroundColor(BTN_COLOR); // <- normal color
        diffcheckBtn.setHoverBackgroundColor(BTN_COLOR_DARKER); //
<- hover color
        diffcheckBtn.setBorderColor(BTN_COLOR);// <- normal color
        diffcheckBtn.setHoverBorderColor(BTN_COLOR_DARKER); // <-
hover color
        diffcheckBtn.setBorderThickness(2);
        diffcheckBtn.setCornerRadius(10);
        diffcheckBtn.addActionListener(e -> {
            try {
                highlightDiffs(true);
            } catch (BadLocationException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
        });

        RoundedButton findBtn = new RoundedButton();
        findBtn.setText(null);
        // Load local SVG (supports recoloring and scaling)
        FlatSVGIcon findIcon = new
FlatSVGIcon("diffchecker/images/icons/find.svg", 20, 20);
        // turn the icon monochrome white
        findIcon.setColorFilter(new FlatSVGIcon.ColorFilter(c ->
Color.WHITE));
        findBtn.setIcon(findIcon);
        findBtn.setBackgroundColor(BTN_COLOR_BLACK); // <- normal
color
        findBtn.setHoverBackgroundColor(BTN_COLOR_DARKER); // <-
hover color
        findBtn.setBorderColor(BTN_COLOR_BLACK);// <- normal color
        findBtn.setHoverBorderColor(BTN_COLOR_DARKER); // <- hover
color
        findBtn.setBorderThickness(2);
        findBtn.setCornerRadius(10);
        findBtn.setMargin(new Insets(5, 5, 5, 5));
        // Hook the 🔍 button
        findBtn.addActionListener(e -> {
            RSyntaxTextArea target = lastFocusedEditor;
            if (target == jt1 && findReplace1 != null) {
                findReplace1.getReplaceAction().actionPerformed(
                        new java.awt.event.ActionEvent(jt1,
ActionEvent.ACTION_PERFORMED, "Replace"));
            } else if (target == jt2 && findReplace2 != null) {
                findReplace2.getReplaceAction().actionPerformed(
                        new java.awt.event.ActionEvent(jt2,
ActionEvent.ACTION_PERFORMED, "Replace"));
            } else {
                EditorUtils.showToast(findBtn,
                        "<html>Click inside one of the <strong>text
editors</strong> first, <br> then press the  \"   🔍  
\"  button to use <strong>Find/Replace</strong></html>");
            }
        });
```

```
        themeToggleBtn = new RoundedButton();
        themeToggleBtn.setText(null);
        // Load local SVG (supports recoloring and scaling)
        FlatSVGIcon themeIcon = new
FlatSVGIcon("diffchecker/images/icons/sun.svg", 20, 20);
        // turn the icon monochrome white
        themeIcon.setColorFilter(new FlatSVGIcon.ColorFilter(c ->
Color.WHITE));
        themeToggleBtn.setIcon(themeIcon);
        themeToggleBtn.setBackgroundColor(BTN_COLOR_BLACK); // <-
normal color
        themeToggleBtn.setHoverBackgroundColor(BTN_COLOR_DARKER
); // <- hover color
        themeToggleBtn.setBorderColor(BTN_COLOR_BLACK);// <- normal
color
        themeToggleBtn.setHoverBorderColor(BTN_COLOR_DARKER); //
<- hover color
        themeToggleBtn.setBorderThickness(2);
        themeToggleBtn.setCornerRadius(10);
        themeToggleBtn.setMargin(new Insets(5, 5, 5, 5));
        themeToggleBtn.addActionListener(e ->
ThemeManager.toggleTheme());

        lineHighlightToggleBtn = new RoundedButton();
        lineHighlightToggleBtn.setText(null);
        // Load local SVG (supports recoloring and scaling)
        FlatSVGIcon codeIcon = new
FlatSVGIcon("diffchecker/images/icons/code.svg", 20, 20);
        // turn the icon monochrome white
        codeIcon.setColorFilter(new FlatSVGIcon.ColorFilter(c ->
Color.WHITE));
        lineHighlightToggleBtn.setIcon(codeIcon);
        lineHighlightToggleBtn.setBackgroundColor(BTN_COLOR_BLACK);
// <- normal color
        lineHighlightToggleBtn.setHoverBackgroundColor(BTN_COLOR_DA
RKER); // <- hover color
        lineHighlightToggleBtn.setBorderColor(BTN_COLOR_BLACK);// <-
normal color
        lineHighlightToggleBtn.setHoverBorderColor(BTN_COLOR_DARKE
R); // <- hover color
        lineHighlightToggleBtn.setBorderThickness(2);
        lineHighlightToggleBtn.setCornerRadius(10);
        lineHighlightToggleBtn.setMargin(new Insets(5, 5, 5, 5));
        // Initial state sync (startup) STARTS ENABLED
        lineHighlightToggleBtn.setSelectedState(lineHighlightEnabled);
        lineHighlightToggleBtn.addActionListener(e -> {
            lineHighlightEnabled = !lineHighlightEnabled; // toggle state
            lineHighlightToggleBtn.setSelectedState(lineHighlightEnabled);

            try {
                // clear old highlights
                jt1.getHighlighter().removeAllHighlights();
                jt2.getHighlighter().removeAllHighlights();
                EditorUtils.highlightPositions.clear();
                highlightDiffs(false);
            } catch (BadLocationException ex) {
                ex.printStackTrace();
            }
        });

        wordHighlightToggleBtn = new RoundedButton();
        wordHighlightToggleBtn.setText(null);
        // Load local SVG (supports recoloring and scaling)
        FlatSVGIcon wordIcon = new
FlatSVGIcon("diffchecker/images/icons/word.svg", 20, 20);
        // turn the icon monochrome white
        wordIcon.setColorFilter(new FlatSVGIcon.ColorFilter(c ->
Color.WHITE));
        wordHighlightToggleBtn.setIcon(wordIcon);
        wordHighlightToggleBtn.setBackgroundColor(BTN_COLOR_BLACK)
; // <- normal color
        wordHighlightToggleBtn.setHoverBackgroundColor(BTN_COLOR_D
ARKER); // <- hover color
        wordHighlightToggleBtn.setBorderColor(BTN_COLOR_BLACK);// <-
normal color
        wordHighlightToggleBtn.setHoverBorderColor(BTN_COLOR_DARK
ER); // <- hover color
        wordHighlightToggleBtn.setBorderThickness(2);
        wordHighlightToggleBtn.setCornerRadius(10);
        wordHighlightToggleBtn.setMargin(new Insets(5, 5, 5, 5));
        wordHighlightToggleBtn.addActionListener(e -> {
            wordHighlightEnabled = !wordHighlightEnabled; // toggle state
            wordHighlightToggleBtn.setSelectedState(wordHighlightEnabled);

            try {
                // clear old highlights
                jt1.getHighlighter().removeAllHighlights();
                jt2.getHighlighter().removeAllHighlights();
                EditorUtils.highlightPositions.clear();
                highlightDiffs(false);
            } catch (BadLocationException ex) {
                ex.printStackTrace();
            }
        });

        wordWrapToggleBtn = new RoundedButton();
        wordWrapToggleBtn.setText(null);
        // Load local SVG (supports recoloring and scaling)
        FlatSVGIcon wordWrapIcon = new
FlatSVGIcon("diffchecker/images/icons/wrap_text.svg", 20, 20);
        // turn the icon monochrome white
        wordWrapIcon.setColorFilter(new FlatSVGIcon.ColorFilter(c ->
Color.WHITE));
        wordWrapToggleBtn.setIcon(wordWrapIcon);
        wordWrapToggleBtn.setBackgroundColor(BTN_COLOR_BLACK); //
<- normal color
        wordWrapToggleBtn.setHoverBackgroundColor(BTN_COLOR_DAR
KER); // <- hover color
        wordWrapToggleBtn.setBorderColor(BTN_COLOR_BLACK);// <-
normal color
        wordWrapToggleBtn.setHoverBorderColor(BTN_COLOR_DARKER);
// <- hover color
        wordWrapToggleBtn.setBorderThickness(2);
        wordWrapToggleBtn.setCornerRadius(10);
        wordWrapToggleBtn.setMargin(new Insets(5, 5, 5, 5));
        wordWrapToggleBtn.addActionListener(e -> {
            wordWrapToggle();
        });
        RoundedButton previousBtn = new RoundedButton("◀");
        previousBtn.setBackgroundColor(BTN_COLOR_BLACK); // <-
normal color
        previousBtn.setHoverBackgroundColor(BTN_COLOR_DARKER); //
<- hover color
        previousBtn.setBorderColor(BTN_COLOR_BLACK);// <- normal
color
        previousBtn.setHoverBorderColor(BTN_COLOR_DARKER); // <-
hover color
        previousBtn.setBorderThickness(2);
        previousBtn.setCornerRadius(10);
        previousBtn.setMargin(new Insets(5, 10, 5, 0));
        previousBtn.addActionListener(e -> {
            previousDiff();
        });

        RoundedButton nextBtn = new RoundedButton("▶");
        nextBtn.setBackgroundColor(BTN_COLOR_BLACK); // <- normal
color
        nextBtn.setHoverBackgroundColor(BTN_COLOR_DARKER); // <-
hover color
```

```
        nextBtn.setBorderColor(BTN_COLOR_BLACK);// <- normal color
        nextBtn.setHoverBorderColor(BTN_COLOR_DARKER); // <- hover
color
        nextBtn.setBorderThickness(2);
        nextBtn.setCornerRadius(10);
        nextBtn.setMargin(new Insets(5, 10, 5, 0));
        nextBtn.addActionListener(e -> {
            nextDiff();
        });

        // LEFT: Clear Button
        RoundedButton clearBtn = new RoundedButton("Clear");
        clearBtn.setBackgroundColor(BTN_COLOR_BLACK);
        clearBtn.setHoverBackgroundColor(BTN_COLOR_DARKER);
        clearBtn.setBorderColor(BTN_COLOR_BLACK);
        clearBtn.setHoverBorderColor(BTN_COLOR_DARKER);
        clearBtn.setBorderThickness(2);
        clearBtn.setCornerRadius(10);
        clearBtn.addActionListener(e -> {
            jt1.setText("");
            jt2.setText("");

            // Clear old highlights and reapply with new theme colors
            jt1.getHighlighter().removeAllHighlights();
            jt2.getHighlighter().removeAllHighlights();
            jt1.removeAllLineHighlights();
            jt2.removeAllLineHighlights();
            EditorUtils.highlightPositions.clear();

            revalidate();
            repaint();
        });

        RoundedButton deleteBtn = new RoundedButton();
        deleteBtn.setText(null);
        // Load local SVG (supports recoloring and scaling)
        FlatSVGIcon deleteIcon = new
FlatSVGIcon("diffchecker/images/icons/trash.svg", 20, 20);
        // turn the icon monochrome white
        deleteIcon.setColorFilter(new FlatSVGIcon.ColorFilter(c ->
Color.WHITE));
        deleteBtn.setIcon(deleteIcon);
        deleteBtn.setBackgroundColor(BTN_COLOR_BLACK); // <- normal
color
        deleteBtn.setHoverBackgroundColor(BTN_COLOR_DARKER); // <-
hover color
        deleteBtn.setBorderColor(BTN_COLOR_BLACK);// <- normal color
        deleteBtn.setHoverBorderColor(BTN_COLOR_DARKER); // <- hover
color
        deleteBtn.setBorderThickness(2);
        deleteBtn.setCornerRadius(10);
        deleteBtn.setMargin(new Insets(5, 5, 5, 5));
        deleteBtn.addActionListener(e -> {
            deleteDiff();
        });

        // RIGHT: Save Button
        RoundedButton saveBtn = new RoundedButton("Save");
        saveBtn.setBackgroundColor(BTN_COLOR_BLACK);
        saveBtn.setHoverBackgroundColor(BTN_COLOR_DARKER);
        saveBtn.setBorderColor(BTN_COLOR_BLACK);
        saveBtn.setHoverBorderColor(BTN_COLOR_DARKER);
        saveBtn.setBorderThickness(2);
        saveBtn.setCornerRadius(10);
        saveBtn.addActionListener(e -> saveToDatabase());

        // TOOLTIPS
        diffcheckBtn.setToolTipText("<html><strong>Find
Difference</strong> <br> ( Alt + Shift + Enter )</html>");
        previousBtn.setToolTipText("<html><strong>Previous Diff</strong>
<br> ( Alt + Left Arrow )</html>");
        nextBtn.setToolTipText("<html><strong>Next Diff</strong> <br> ( Alt
+ Right Arrow )</html>");
        clearBtn.setToolTipText("<html><strong>Clear</strong> <br> ( Ctrl +
R )</html>");
        deleteBtn.setToolTipText("<html><strong>Delete</strong> <br> (
Ctrl + Shift + X )</html>");
        findBtn.setToolTipText("<html><strong>Find/Replace</strong> <br>
( Ctrl + F )</html>");
        themeToggleBtn.setToolTipText("<html><strong>Toggle Light/Dark
Theme</strong> <br> ( Ctrl + G )</html>");
        saveBtn.setToolTipText("<html><strong>Save</strong> <br> ( Ctrl +
S )</html>");
        wordHighlightToggleBtn.setToolTipText("<html><strong>Toggle
Word Highlight</strong> <br> ( Alt + W )</html>");
        lineHighlightToggleBtn.setToolTipText("<html><strong>Toggle Line
Highlight</strong> <br> ( Alt + E )</html>");
        wordWrapToggleBtn.setToolTipText("<html><strong>Toggle Word
Wrap</strong> <br> ( Alt + Q )</html>");

        bottomPanel = new JPanel(new BorderLayout());
        bottomPanel.setBorder(BorderFactory.createEmptyBorder(0, 5, 5,
5));

        leftButtonPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        leftButtonPanel.add(clearBtn);
        leftButtonPanel.add(deleteBtn);
        bottomPanel.add(leftButtonPanel, BorderLayout.WEST);

        // CENTER: diffcheckBtn, previousBtn, nextBtn
        centerButtonPanel = new JPanel(new
FlowLayout(FlowLayout.CENTER));
        centerButtonPanel.add(wordWrapToggleBtn);
        centerButtonPanel.add(wordHighlightToggleBtn);
        centerButtonPanel.add(lineHighlightToggleBtn);
        centerButtonPanel.add(diffcheckBtn);
        centerButtonPanel.add(previousBtn);
        centerButtonPanel.add(nextBtn);
        bottomPanel.add(centerButtonPanel, BorderLayout.CENTER);

        rightButtonPanel = new JPanel(new
FlowLayout(FlowLayout.RIGHT));
        rightButtonPanel.add(themeToggleBtn);
        rightButtonPanel.add(findBtn);
        rightButtonPanel.add(saveBtn);

        // SYNTAX HIGHLIGHT TEST BUTTON
        // RoundedButton syntaxBtn = new RoundedButton("Java Syntax");
        // syntaxBtn.addActionListener(e -> {
        // SyntaxManager.setSyntax("text/java");
        // });
        // rightButtonPanel.add(syntaxBtn);

        bottomPanel.add(rightButtonPanel, BorderLayout.EAST);

        add(bottomPanel, BorderLayout.SOUTH);

        // APPLY ACTIVATED BORDER STYLE
        activatedEditorBorderStyle();

        // PRE APPLY THEME TO RUN DEFAULT
        ThemeManager.register(this);
        SyntaxManager.register(this);
    }

    // KEYBOARD SHORTCUTS
    public void setKeyboardShortcuts(){
        // CTRL + S HOTKEY FOR SAVING
```

```
        getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_CO
MPONENT)
                .put(KeyStroke.getKeyStroke("control S"), "saveDiff");

        getActionMap().put("saveDiff", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                saveToDatabase();
            }
        });

        // ALT + Q HOTKEY FOR TOGGLING WORD WRAP
        getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_CO
MPONENT)
                .put(KeyStroke.getKeyStroke("alt Q"), "toggleWordWrap");

        getActionMap().put("toggleWordWrap", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                wordWrapToggle();
            }
        });

        // ALT + W HOTKEY FOR TOGGLING WORD HIGHLIGHT
        getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_CO
MPONENT)
                .put(KeyStroke.getKeyStroke("alt W"), "toggleHighlightWord");

        getActionMap().put("toggleHighlightWord", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                highlightedWordToggle();
            }
        });

        // ALT + E HOTKEY FOR TOGGLING LINE HIGHLIGHT
        getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_CO
MPONENT)
                .put(KeyStroke.getKeyStroke("alt E"), "toggleHighlightLine");

        getActionMap().put("toggleHighlightLine", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                highlightedLineToggle();
            }
        });

        // ALT + SHIFT + ENTER hotkey for diff checking
        getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_CO
MPONENT)
                .put(KeyStroke.getKeyStroke(KeyEvent.VK_ENTER,
InputEvent.ALT_DOWN_MASK | InputEvent.SHIFT_DOWN_MASK),
                        "highlightDiffs");

        getActionMap().put("highlightDiffs", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    highlightDiffs(true);
                } catch (BadLocationException ex) {
                    ex.printStackTrace();
                }
            }
        });

        // CTRL + SHIFT + X hotkey for Deleting from database
        getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_CO
MPONENT)
                .put(KeyStroke.getKeyStroke(KeyEvent.VK_X,
InputEvent.CTRL_DOWN_MASK | InputEvent.SHIFT_DOWN_MASK),
                        "deleteDiff");

        getActionMap().put("deleteDiff", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                deleteDiff();
            }
        });

        // ALT + LEFT = Previous diff
        getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_CO
MPONENT)
                .put(KeyStroke.getKeyStroke(KeyEvent.VK_LEFT,
InputEvent.ALT_DOWN_MASK), "previousDiff");

        getActionMap().put("previousDiff", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                previousDiff();
            }
        });

        // ALT + RIGHT = Next diff
        getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_CO
MPONENT)
                .put(KeyStroke.getKeyStroke(KeyEvent.VK_RIGHT,
InputEvent.ALT_DOWN_MASK), "nextDiff");

        getActionMap().put("nextDiff", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                nextDiff();
            }
        });

        // CTRL + R HOTKEY FOR CLEARING
        getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_CO
MPONENT)
                .put(KeyStroke.getKeyStroke("control R"), "clearTextAreas");

        getActionMap().put("clearTextAreas", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                jt1.setText("");
                jt2.setText("");

                // Clear old highlights and reapply with new theme colors
                jt1.getHighlighter().removeAllHighlights();
                jt2.getHighlighter().removeAllHighlights();
                jt1.removeAllLineHighlights();
                jt2.removeAllLineHighlights();
                EditorUtils.highlightPositions.clear();

                revalidate();
                repaint();
            }
        });

        // CTRL + G HOTKEY FOR TOGGLING THEME
        getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_CO
MPONENT)
                .put(KeyStroke.getKeyStroke("control G"), "toggleTheme");

        getActionMap().put("toggleTheme", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                ThemeManager.toggleTheme();
            }
        });
    }
```

```
    public void activatedEditorBorderStyle() {
        // STEAL FOCUS FROM TEXTAREAS WHEN CLICKING OUTSIDE
        addMouseListener(new MouseAdapter() {

            public void mousePressed(java.awt.event.MouseEvent e) {
                Component clicked =
SwingUtilities.getDeepestComponentAt(SplitTextTabPanel.this, e.getX(),
e.getY());
                if (!(SwingUtilities.isDescendingFrom(clicked, jt1) ||
SwingUtilities.isDescendingFrom(clicked, jt2))) {
                    requestFocusInWindow(); // steal focus from textareas
                    repaint(); // helps caret disappear properly
                }
            }
        });
    }

    private void highlightDiffs(boolean diffCheckerBtnInitiated) throws
BadLocationException {
        // always start clean
        jt1.getHighlighter().removeAllHighlights();
        jt2.getHighlighter().removeAllHighlights();
        jt1.removeAllLineHighlights();
        jt2.removeAllLineHighlights();
        EditorUtils.highlightPositions.clear();

        diffGroups.clear();
        currentGroupIndex = -1;

        String leftText = jt1.getText();
        String rightText = jt2.getText();

        // Check if both are exactly the same
        JFrame frame = (JFrame) SwingUtilities.getWindowAncestor(this);
        if (leftText.equals(rightText)) {
            if (diffCheckerBtnInitiated) {
                EditorUtils.showCenteredToast(
                        "No differences found — both text editors are either empty
or identical!",
                        frame);
            }
            return; // nothing else to do
        }

        List<String> leftLines = Arrays.asList(leftText.split("\n"));
        List<String> rightLines = Arrays.asList(rightText.split("\n"));

        Patch<String> patch = DiffUtils.diff(leftLines, rightLines);

        // FOR DEBUGGING LINE HIGHLIGHT
        // System.out.println("Line highlight: " + lineHighlightEnabled);
        for (AbstractDelta<String> delta : patch.getDeltas()) {
            int origPos = delta.getSource().getPosition();
            int revPos = delta.getTarget().getPosition();

            DiffGroup group = new DiffGroup();

            switch (delta.getType()) {
                case DELETE:
                    if (lineHighlightEnabled) {
                        EditorUtils.highlightFullLines(jt1, origPos,
delta.getSource().size(), lineRemovedColor);
                    }
                    int startOffsetLeft = jt1.getLineStartOffset(origPos);
                    group.left = new EditorUtils.HighlightInfo(jt1, startOffsetLeft,
startOffsetLeft);
                    break;

                case INSERT:
                    if (lineHighlightEnabled) {
                        EditorUtils.highlightFullLines(jt2, revPos,
delta.getTarget().size(), lineAddedColor);
                    }
                    int startOffsetRight = jt2.getLineStartOffset(revPos);
                    group.right = new EditorUtils.HighlightInfo(jt2,
startOffsetRight, startOffsetRight);
                    break;

                case CHANGE:
                    if (lineHighlightEnabled) {
                        EditorUtils.highlightFullLines(jt1, origPos,
delta.getSource().size(), lineRemovedColor);
                        EditorUtils.highlightFullLines(jt2, revPos,
delta.getTarget().size(), lineAddedColor);
                    }

                    int lOff = jt1.getLineStartOffset(origPos);
                    int rOff = jt2.getLineStartOffset(revPos);
                    group.left = new EditorUtils.HighlightInfo(jt1, lOff, lOff);
                    group.right = new EditorUtils.HighlightInfo(jt2, rOff, rOff);

                    // Word-level highlighting
                    if (wordHighlightEnabled){
                        for (int i = 0; i < Math.min(delta.getSource().size(),
delta.getTarget().size()); i++) {
                            EditorUtils.highlightWordDiffs(
                                    jt1, origPos + i,
                                    delta.getSource().getLines().get(i),
                                    delta.getTarget().getLines().get(i),
                                    wordRemovedColor, true);
                            EditorUtils.highlightWordDiffs(
                                    jt2, revPos + i,
                                    delta.getSource().getLines().get(i),
                                    delta.getTarget().getLines().get(i),
                                    wordAddedColor, false);
                        }
                    }
                    break;
                default:
                    break;
            }
            diffGroups.add(group);
        }

        // 👉 Jump to first difference if available
        if (!diffGroups.isEmpty()) {
            currentGroupIndex = 0;
            focusDiffGroup(diffGroups.get(0));
        }
    }

    @Override
    public void applyTheme(boolean dark) {
        Color scrollColor, scrollCornerColor, panelColor,
editorMarginBackgroundColor, trackColor, defaultBorderColor,
                activeBorderColor;

        if (dark) {
            // DARK THEME
            scrollColor = BACKGROUND_DARK;
            scrollCornerColor = SCROLLBAR_CORNER_DARK;
            panelColor = BACKGROUND_DARK;
            editorMarginBackgroundColor = BACKGROUND_DARK;
            trackColor = SCROLLBAR_TRACK_DARK;
            defaultBorderColor = EDITOR_BORDER_COLOR_DARK;
            activeBorderColor = ACTIVE_BORDER_COLOR_DARK;

            // Active highlight colors (switch based on theme)
            lineRemovedColor = LINE_REMOVED_DARK;
```

```
            lineAddedColor = LINE_ADDED_DARK;
            wordRemovedColor = WORD_REMOVED_DARK;
            wordAddedColor = WORD_ADDED_DARK;
        } else {
            // LIGHT THEME
            scrollColor = BACKGROUND_LIGHT;
            scrollCornerColor = SCROLLBAR_CORNER_LIGHT;
            panelColor = BACKGROUND_LIGHT;
            editorMarginBackgroundColor = BACKGROUND_LIGHT;
            trackColor = SCROLLBAR_TRACK_LIGHT;
            defaultBorderColor = EDITOR_BORDER_COLOR_LIGHT;
            activeBorderColor = ACTIVE_BORDER_COLOR_LIGHT;

            // Active highlight colors (switch based on theme)
            lineRemovedColor = LINE_REMOVED_LIGHT;
            lineAddedColor = LINE_ADDED_LIGHT;
            wordRemovedColor = WORD_REMOVED_LIGHT;
            wordAddedColor = WORD_ADDED_LIGHT;
        }

        // FOR scroll1 TRACK BAR COLOR: value is passed down to each
CustomScrollBarUI
        if (scroll1.getVerticalScrollBar().getUI() instanceof CustomScrollBarUI
ui1) {
            ui1.setTrackColor(trackColor);
        }
        if (scroll1.getHorizontalScrollBar().getUI() instanceof
CustomScrollBarUI ui2) {
            ui2.setTrackColor(trackColor);
        }

        // For scroll2 TRACK BAR COLOR
        if (scroll2.getVerticalScrollBar().getUI() instanceof CustomScrollBarUI
ui3) {
            ui3.setTrackColor(trackColor);
        }
        if (scroll2.getHorizontalScrollBar().getUI() instanceof
CustomScrollBarUI ui4) {
            ui4.setTrackColor(trackColor);
        }
        scroll1.getHorizontalScrollBar().setBackground(scrollColor);
        scroll2.getHorizontalScrollBar().setBackground(scrollColor);
        scroll1.getVerticalScrollBar().setBackground(scrollColor);
        scroll2.getVerticalScrollBar().setBackground(scrollColor);

        // CHANGE BORDER COLOR UPON ACTIVATING EDITORS
        jt1.addFocusListener(new FocusAdapter() {
            @Override
            public void focusGained(FocusEvent e) {
                jt1IsActive = true;
                jt2IsActive = false;
                scroll1.setBorder(BorderFactory.createLineBorder(activeBorder
Color));
                scroll2.setBorder(BorderFactory.createLineBorder(defaultBorder
Color));
            }

            @Override
            public void focusLost(FocusEvent e) {
                jt1IsActive = false;
                scroll1.setBorder(BorderFactory.createLineBorder(defaultBorder
Color));
            }
        });

        jt2.addFocusListener(new FocusAdapter() {
            @Override
            public void focusGained(FocusEvent e) {
                jt2IsActive = true;
                jt1IsActive = false;
                scroll2.setBorder(BorderFactory.createLineBorder(activeBorder
Color));
                scroll1.setBorder(BorderFactory.createLineBorder(defaultBorder
Color));
            }

            @Override
            public void focusLost(FocusEvent e) {
                jt2IsActive = false;
                scroll2.setBorder(BorderFactory.createLineBorder(defaultBorder
Color));
            }
        });

        // DEFAULT BORDER COLOR FOR EDITORS
        scroll1.setBorder(BorderFactory.createLineBorder(defaultBorderColo
r));
        scroll2.setBorder(BorderFactory.createLineBorder(defaultBorderColo
r));

        // REMOVING THE WHITE SQUARES AT THE INTERSCTION OF
THE SCROLLBARS
        scroll1CornerLeft.setBackground(scrollCornerColor);
        scroll1CornerRight.setBackground(scrollCornerColor);
        scroll2CornerLeft.setBackground(scrollCornerColor);
        scroll2CornerRight.setBackground(scrollCornerColor);

        // BUTTONS AND TEXT EDITORS PANELS
        leftButtonPanel.setBackground(panelColor);
        centerButtonPanel.setBackground(panelColor);
        rightButtonPanel.setBackground(panelColor);

        sideBySidePanel.setBackground(editorMarginBackgroundColor);
        contentPanel.setBackground(editorMarginBackgroundColor);
        bottomPanel.setBackground(editorMarginBackgroundColor);

        revalidate();
        repaint();

        String themePath = dark ? "/diffchecker/themes/dark.xml"
                : "/diffchecker/themes/light.xml";

        try (InputStream in = getClass().getResourceAsStream(themePath))
{
            if (in != null) {
                Theme theme = Theme.load(in);
                theme.apply(jt1);
                theme.apply(jt2);

                // If you also want scroll pane borders/background consistent:
                Color bg = jt1.getBackground();
                scroll1.getViewport().setBackground(bg);
                scroll2.getViewport().setBackground(bg);

                // Load embedded Fira Code
                InputStream fontStream =
getClass().getResourceAsStream("/diffchecker/fonts/FiraCode-
Regular.ttf");
                Font firaCode = Font.createFont(Font.TRUETYPE_FONT,
fontStream);

                firaCode = firaCode.deriveFont(Font.PLAIN, sizeFromXML);
                jt1.setFont(firaCode);
                jt2.setFont(firaCode);
            }
        } catch (IOException | FontFormatException e) {
            e.printStackTrace();
        }

        // Clear old highlights and reapply with new theme colors
```

```
        jt1.getHighlighter().removeAllHighlights();
        jt2.getHighlighter().removeAllHighlights();
        jt1.removeAllLineHighlights();
        jt2.removeAllLineHighlights();
        EditorUtils.highlightPositions.clear();

        try {
            highlightDiffs(false);
        } catch (BadLocationException e) {
            e.printStackTrace();
        }
    }

    // APPLY SYNTAX HIGHLIGHTING: CONNECTED TO CUSTOM
TITLE BAR
    public void setSyntaxStyleBoth(String style) {
        if (jt1 != null)
            jt1.setSyntaxEditingStyle(style);
        if (jt2 != null)
            jt2.setSyntaxEditingStyle(style);
    }

    public void setSyntaxStyleLeft(String style) {
        if (jt1 != null)
            jt1.setSyntaxEditingStyle(style);
    }

    public void setSyntaxStyleRight(String style) {
        if (jt2 != null)
            jt2.setSyntaxEditingStyle(style);
    }

    private void highlightedWordToggle() {
        wordHighlightEnabled = !wordHighlightEnabled; // toggle state
        wordHighlightToggleBtn.setSelectedState(wordHighlightEnabled);

        try {
            // clear old highlights
            jt1.getHighlighter().removeAllHighlights();
            jt2.getHighlighter().removeAllHighlights();
            EditorUtils.highlightPositions.clear();
            highlightDiffs(false);
        } catch (BadLocationException ex) {
            ex.printStackTrace();
        }
    }

    private void highlightedLineToggle() {
        lineHighlightEnabled = !lineHighlightEnabled; // toggle state
        lineHighlightToggleBtn.setSelectedState(lineHighlightEnabled);

        try {
            // clear old highlights
            jt1.removeAllLineHighlights();
            jt2.removeAllLineHighlights();
            EditorUtils.highlightPositions.clear();
            highlightDiffs(false);
        } catch (BadLocationException ex) {
            ex.printStackTrace();
        }
    }

    private void wordWrapToggle() {
        wordWrapEnabled = !wordWrapEnabled; // toggle state
        wordWrapToggleBtn.setSelectedState(wordWrapEnabled);

        jt1.setLineWrap(wordWrapEnabled);
        jt1.setWrapStyleWord(wordWrapEnabled);

        jt2.setLineWrap(wordWrapEnabled);
        jt2.setWrapStyleWord(wordWrapEnabled);

        // Force UI refresh
        jt1.revalidate();
        jt1.repaint();
        jt2.revalidate();
        jt2.repaint();
    }

    private void focusDiffGroup(DiffGroup group) {
        if (group == null)
            return;

        // // Steal focus from text areas temporarily
        requestFocusInWindow();

        if (group.left != null) {
            EditorUtils.scrollToOffset(jt1, group.left.startOffset);
        }

        if (group.right != null) {
            EditorUtils.scrollToOffset(jt2, group.right.startOffset);
        }

    }

    private void previousDiff() {
        if (diffGroups.isEmpty())
            return;
        currentGroupIndex--;
        if (currentGroupIndex < 0)
            currentGroupIndex = diffGroups.size() - 1;
        focusDiffGroup(diffGroups.get(currentGroupIndex));
    }

    private void nextDiff(){
        if (diffGroups.isEmpty())
            return;
        currentGroupIndex++;
        if (currentGroupIndex >= diffGroups.size())
            currentGroupIndex = 0;
        focusDiffGroup(diffGroups.get(currentGroupIndex));
    }

    private void deleteDiff() {
        if (currentDiff == null || currentDiff.id == -1) {
            JOptionPane.showMessageDialog(this, "No saved record to
delete.");
            return;
        }

        int confirm = JOptionPane.showConfirmDialog(
                this,
                "Are you sure you want to delete \"" + currentDiff.title + "\"?",
                "Confirm Delete",
                JOptionPane.YES_NO_OPTION);

        if (confirm == JOptionPane.YES_OPTION) {
            DB db = new DB();
            DiffRepository repo = new DiffRepository(db);
            boolean success = repo.deleteDiff(currentDiff.id);

            if (success){
                JOptionPane.showMessageDialog(this, "Deleted successfully!");

                // Remove this tab from the JTabbedPane
                Container parent = getParent();
                while (parent != null && !(parent instanceof JTabbedPane)) {
                    parent = parent.getParent();
                }
```

```
            if (parent instanceof JTabbedPane) {
                JTabbedPane tabbedPane = (JTabbedPane) parent;
                int index = tabbedPane.indexOfComponent(this);

                if (index != -1) {
                    tabbedPane.remove(index);

                    // After removing, check tab count:
                    int totalTabs = tabbedPane.getTabCount();

                    // If we just deleted the last *content* tab (now only the '+'
tab remains)
                    if (totalTabs == 1) {
                        // Create a new blank tab instead of leaving only the "+"
tab
                        if (newTabCallback != null) {
                            newTabCallback.run();
                        }
                        return;
                    } else {
                        // Otherwise, select the previous tab if available
                        if (index > 0) {
                            tabbedPane.setSelectedIndex(index - 1);
                        } else {
                            tabbedPane.setSelectedIndex(0);
                        }
                    }
                }
            } else {
                JOptionPane.showMessageDialog(this, "Delete failed.");
            }
        }
    }

    // LOAD/STORE FROM DATABASE
    public void loadFromDatabase(DiffData data) {
        currentDiff = data;

        // temporarily detach listeners
        jt1.getDocument().removeDocumentListener(dirtyListener1);
        jt2.getDocument().removeDocumentListener(dirtyListener2);

        jt1.setText(data.leftText);
        jt2.setText(data.rightText);

        // reattach listeners
        jt1.getDocument().addDocumentListener(dirtyListener1);
        jt2.getDocument().addDocumentListener(dirtyListener2);

        markSaved(); // reset dirty flag
    }

    public void saveToDatabase() {
        String title = JOptionPane.showInputDialog(
                this,
                "What's the title of this diff?",
                currentDiff != null ? currentDiff.title : "");

        // User pressed Cancel or closed the dialog → do nothing
        if (title == null) {
            return;
        }

        // User pressed OK but left input blank → show error and stop
        if (title.trim().isEmpty()) {
            JOptionPane.showMessageDialog(
                    this,
                    "Title cannot be empty.",
                    "Invalid Title",
                    JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Capitalize the title
        title = EditorUtils.capitalizeTitle(title);

        String leftText = jt1.getText();
        String rightText = jt2.getText();

        DB db = new DB();
        DiffRepository repo = new DiffRepository(db);

        boolean success;
        if (currentDiff != null && currentDiff.id != -1) {
            // Update existing record
            currentDiff.title = title;
            currentDiff.leftText = leftText;
            currentDiff.rightText = rightText;
            success = repo.updateDiff(currentDiff);
        } else {
            // Insert new record
            DiffData newData = new DiffData(title, leftText, rightText);
            success = repo.saveDiff(newData);
            currentDiff = newData; // track this record now
        }

        currentDiff.leftText = jt1.getText();
        currentDiff.rightText = jt2.getText();

        JOptionPane.showMessageDialog(this, success ? "Saved
successfully!" : "Save failed.");

        if (success) {
            // 🔷 Update the tab title in the JTabbedPane
            Container parent = getParent();
            while (parent != null && !(parent instanceof JTabbedPane)) {
                parent = parent.getParent();
            }
            if (parent instanceof JTabbedPane) {
                JTabbedPane tabbedPane = (JTabbedPane) parent;
                int index = tabbedPane.indexOfComponent(this);
                if (index != -1) {
                    // If you use ClosableTabTitleComponent, update its label too
                    Component tabComponent =
tabbedPane.getTabComponentAt(index);
                    if (tabComponent instanceof ClosableTabTitleComponent) {
                        ((ClosableTabTitleComponent)
tabComponent).setTitle(title);
                    } else {
                        tabbedPane.setTitleAt(index, title);
                    }
                }
            }
            markSaved();
        }
    }
}
```

SyntaxHighlightable.java

```
package com.diffchecker.components;
```

```
public interface SyntaxHighlightable {
    void applySyntaxStyle(String syntaxStyle);
}
```

SyntaxManager.java

```
package com.diffchecker.components;

import java.util.ArrayList;
import java.util.List;
import java.util.prefs.Preferences;

public class SyntaxManager {

    private static final String PREF_NODE = "com.diffchecker.syntax";
    private static final String PREF_KEY  = "currentSyntaxStyle";

    private static final List<SyntaxHighlightable> components = new
ArrayList<>();

    // Load last-used syntax (default to "text/plain")
    private static String currentSyntax;

    static {
        Preferences prefs = Preferences.userRoot().node(PREF_NODE); //
SAVE THE SYNTAX IN THE OS PREFERENCES
        currentSyntax = prefs.get(PREF_KEY, "text/plain");
    }

    public static void register(SyntaxHighlightable comp) {
        components.add(comp);
        comp.applySyntaxStyle(currentSyntax);
    }

    public static void setSyntax(String syntaxStyle) {
        currentSyntax = syntaxStyle;

        // Persist the new style
        Preferences.userRoot()
            .node(PREF_NODE)
            .put(PREF_KEY, syntaxStyle);

        // Apply to all registered components
        for (SyntaxHighlightable comp : components) {
            comp.applySyntaxStyle(syntaxStyle);
        }
    }

    public static String getCurrentSyntax() {
        return currentSyntax;
    }
}
```

ThemedComponent.java

```
package com.diffchecker.components;

public interface ThemedComponent {
    void applyTheme(boolean dark);
}
```

ThemeManager.java

```
package com.diffchecker.components;

import java.util.*;
import java.util.prefs.Preferences;

public class ThemeManager{
```

```
    private static final String PREF_NODE = "com.diffchecker";
    private static final String PREF_KEY = "darkThemeEnabled";
    private static boolean darkThemeEnabled;

    private static final List<ThemedComponent> components = new
ArrayList<>();

    static {
        Preferences prefs = Preferences.userRoot().node(PREF_NODE); //
THE THEME TOGGLE IS BEING SAVED IN THE PREFERENCES
        darkThemeEnabled = prefs.getBoolean(PREF_KEY, true); // default
dark
    }

    public static boolean isDarkTheme() { return darkThemeEnabled; }

    public static void toggleTheme() { setDarkTheme(!darkThemeEnabled);
}
    public static void setDarkTheme(boolean dark) {
        darkThemeEnabled = dark;
        Preferences.userRoot().node(PREF_NODE).putBoolean(PREF_KEY
, dark);
        applyThemeToAll();
    }

    public static void register(ThemedComponent comp) {
        components.add(comp);
        comp.applyTheme(darkThemeEnabled);
    }

    public static void applyThemeToAll() {
        for (ThemedComponent comp : components) {
            comp.applyTheme(darkThemeEnabled);
        }
    }
}
```

TitlebarMover.java

```
package com.diffchecker.components;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TitlebarMover extends MouseAdapter {
  private final JFrame frame;
  private final Runnable onDoubleClick;
  private final Runnable onRestoreFromDrag;
  private Point mousePressedLocation = null;
  private Point mouseScreenLocation = null;
  private Dimension previousSize = null;
  private boolean isDragging = false;
  private boolean restoredOnDrag = false;

  public TitlebarMover(JFrame frame, Component draggableArea,
      Runnable onDoubleClick,
      Runnable onRestoreFromDrag) {
    this.frame = frame;
    this.onDoubleClick = onDoubleClick;
    this.onRestoreFromDrag = onRestoreFromDrag;

    draggableArea.addMouseListener(this);
    draggableArea.addMouseMotionListener(this);
  }

  @Override
  public void mousePressed(MouseEvent e) {
    // Save current mouse position and window size
```

```
    mousePressedLocation = e.getPoint();
    mouseScreenLocation = e.getLocationOnScreen();

    isDragging = true;
    restoredOnDrag = false;
  }

  @Override
  public void mouseDragged(MouseEvent e) {
    if (!isDragging || mousePressedLocation == null ||
mouseScreenLocation == null)
      return;

    if ((frame.getExtendedState() & JFrame.MAXIMIZED_BOTH) ==
JFrame.MAXIMIZED_BOTH && !restoredOnDrag) {

      Dimension restoreSize = previousSize != null ? previousSize : new
Dimension(1024, 768);

      // Restore frame
      frame.setExtendedState(JFrame.NORMAL);
      frame.setSize(restoreSize);

      // Get cursor location on screen
      int cursorX = e.getXOnScreen();
      int cursorY = e.getYOnScreen();

      // Place window so the cursor lands on the same Y as it did in the title
bar
      int newX = cursorX - (restoreSize.width / 2);
      int newY = cursorY - mousePressedLocation.y;

      frame.setLocation(newX, newY);

      // Update mousePressedLocation to continue dragging smoothly
      mousePressedLocation = new Point(restoreSize.width / 2,
mousePressedLocation.y);

      restoredOnDrag = true;
      if (onRestoreFromDrag != null)
        onRestoreFromDrag.run();
      return;
    }

    // Move the window with the cursor
    Point current = e.getLocationOnScreen();
    frame.setLocation(
        current.x - mousePressedLocation.x,
        current.y - mousePressedLocation.y);
  }

  @Override
  public void mouseReleased(MouseEvent e) {
    isDragging = false;
    mousePressedLocation = null;
    mouseScreenLocation = null;
  }

  @Override
  public void mouseClicked(MouseEvent e) {
    if (e.getClickCount() == 2 && SwingUtilities.isLeftMouseButton(e)) {
      if (onDoubleClick != null)
        onDoubleClick.run();
    }
  }

}
```

III. MEMBER'S SUMMARY OF A-SSIGNED TASK

Name (LN, FN, MI)	Picture:	Detailed Contributions/Assigned Tasks:
Member Name: Perez Jr., Jose, A.		- Planning of interface and features - Research on what algorithm to use for difference checking between two texts - Project design and features implementation - Documentation

IV. REFERENCES

On real-world usage of Git diff and merge tools
Spinellis, D. (2021). Modern practices in version-controlled text comparison. *IEEE Software, 38*(2), 45–51. https://doi.org/10.1109/MS.2020.3045813

On visual comparison tools and developer workflows
Han, S., Zhang, L., & Wang, X. (2021). A study on the effectiveness of visual differencing tools in code review. *Journal of Software Engineering and Applications, 14*(3), 125–138. https://doi.org/10.4236/jsea.2021.143008

On text comparison for documents and collaboration
Kato, Y., & Oda, T. (2020). Improving document revision accuracy using automated diff tools. *International Journal of Advanced Computer Science and Applications, 11*(9), 586–593. https://thesai.org/Downloads/Volume11No9/Paper_77-Improving_Document_Revision_Accuracy.pdf

On practical applications of online diff utilities
Diffchecker. (2024). About Diffchecker – Online text comparison tool. https://www.diffchecker.com

On modern diff algorithms and optimization
Kim, H., & Lee, D. (2023). Optimizing line-based differencing for large-scale codebases. *ACM Transactions on Software Engineering and Methodology, 32*(4), 1–25. https://doi.org/10.1145/3591234

On real-world usage of Git diff and merge tools
Spinellis, D. (2021). Modern practices in version-controlled text comparison. *IEEE Software, 38*(2), 45–51. https://doi.org/10.1109/MS.2020.3045813

On version control, collaboration, and diff usage
Loeliger, J., & McCullough, M. (2022). *Version control with Git* (3rd ed.). O'Reilly Media. https://www.oreilly.com/library/view/version-control-with/9781492091179/