Міністерство освіти і науки України
Національний технічний університет
України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №9

Технологія розроблення програмного забезпечення

“РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ:
CLIENT-SERVER, PEER-TO-PEER,
SERVICE-ORIENTED ARCHITECTURE”

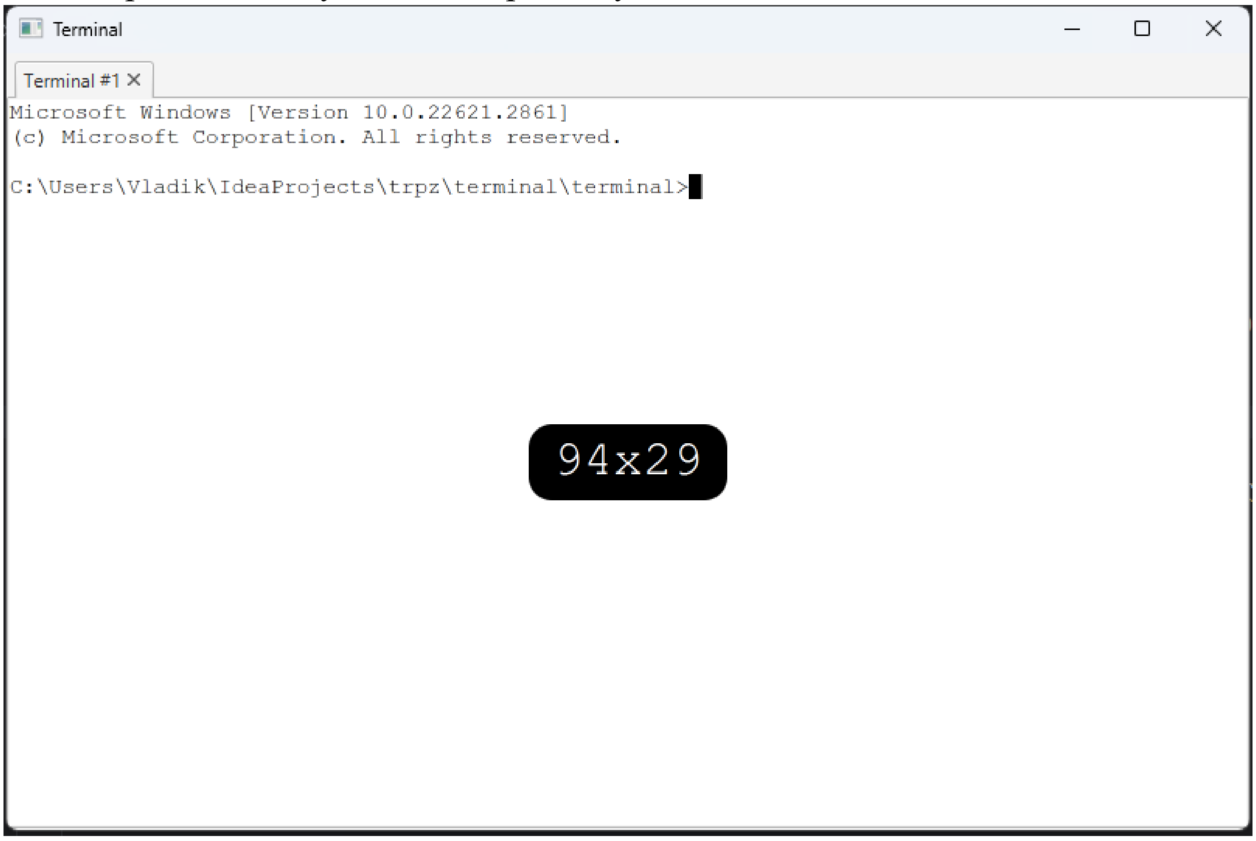
Варіант 8

Виконав
студент групи ІА-13
Крутиус Владислав Віталійович

Київ 2023р.

Хід роботи

Застосунок має Client-Server Architecture. В даній архітектурі сторона клієнта реалізована у вигляді терміналу



```
Microsoft Windows [Version 10.0.22621.2861]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Vladik\IdeaProjects\trpz\terminal\terminal>
```

Серверна частина ж, являє собою застосунок, попередньо завантажений на девайс користувача:

WebkitCall annotation:

```
public @interface WebkitCall {
    String from() default "";
}
```

TabNameGenerator interface:

```
public interface TabNameGenerator {
    String next();
}
```

TerminalConfig class:

```
@JsonInclude(JsonInclude.Include.NON_NULL)
public class TerminalConfig {
```

```java
@JsonProperty("use-default-window-copy")
private boolean useDefaultWindowCopy = true;

@JsonProperty("clear-selection-after-copy")
private boolean clearSelectionAfterCopy = true;

@JsonProperty("copy-on-select")
private boolean copyOnSelect = false;

@JsonProperty("ctrl-c-copy")
private boolean ctrlCCopy = true;

@JsonProperty("ctrl-v-paste")
private boolean ctrlVPaste = true;

@JsonProperty("cursor-color")
private String cursorColor = "black";

@JsonProperty(value = "background-color")
private String backgroundColor = "white";

@JsonProperty("font-size")
private int fontSize = 14;

@JsonProperty(value = "foreground-color")
private String foregroundColor = "black";

@JsonProperty("cursor-blink")
private boolean cursorBlink = false;

@JsonProperty("scrollbar-visible")
private boolean scrollbarVisible = true;

@JsonProperty("enable-clipboard-notice")
private boolean enableClipboardNotice = true;

@JsonProperty("scroll-wheel-move-multiplier")
private double scrollWhellMoveMultiplier = 0.1;
```

```java
    @JsonProperty("font-family")
    private String fontFamily = "\"DejaVu Sans Mono\",
\"Everson Mono\", FreeMono, \"Menlo\", \"Terminal\",
monospace";

    @JsonProperty(value = "user-css")
    private String userCss = "data:text/plain;base64," +
"eC1zY3JlZW4geyBjdXJzb3I6IGF1dG87IH0=";

    @JsonIgnore
    private String windowsTerminalStarter = "cmd.exe";

    @JsonIgnore
    private String unixTerminalStarter = "/bin/bash -i";

    public boolean isUseDefaultWindowCopy() {
        return useDefaultWindowCopy;
    }

    public void setUseDefaultWindowCopy(boolean
useDefaultWindowCopy) {
        this.useDefaultWindowCopy = useDefaultWindowCopy;
    }

    public boolean isClearSelectionAfterCopy() {
        return clearSelectionAfterCopy;
    }

    public void setClearSelectionAfterCopy(boolean
clearSelectionAfterCopy) {
        this.clearSelectionAfterCopy =
clearSelectionAfterCopy;
    }

    public boolean isCopyOnSelect() {
        return copyOnSelect;
    }

    public void setCopyOnSelect(boolean copyOnSelect) {
        this.copyOnSelect = copyOnSelect;
    }
```

```java
public boolean isCtrlCCopy() {
    return ctrlCCopy;
}

public void setCtrlCCopy(boolean ctrlCCopy) {
    this.ctrlCCopy = ctrlCCopy;
}

public boolean isCtrlVPaste() {
    return ctrlVPaste;
}

public void setCtrlVPaste(boolean ctrlVPaste) {
    this.ctrlVPaste = ctrlVPaste;
}

public String getCursorColor() {
    return cursorColor;
}

public void setCursorColor(String cursorColor) {
    this.cursorColor = cursorColor;
}

public String getBackgroundColor() {
    return backgroundColor;
}

public void setBackgroundColor(String backgroundColor) {
    this.backgroundColor = backgroundColor;
}

public int getFontSize() {
    return fontSize;
}

public void setFontSize(int fontSize) {
    this.fontSize = fontSize;
}
```

```java
    public String getForegroundColor() {
        return foregroundColor;
    }

    public void setForegroundColor(String foregroundColor) {
        this.foregroundColor = foregroundColor;
    }

    public boolean isCursorBlink() {
        return cursorBlink;
    }

    public void setCursorBlink(boolean cursorBlink) {
        this.cursorBlink = cursorBlink;
    }

    public boolean isScrollbarVisible() {
        return scrollbarVisible;
    }

    public void setScrollbarVisible(boolean scrollbarVisible)
{
        this.scrollbarVisible = scrollbarVisible;
    }

    public double getScrollWhellMoveMultiplier() {
        return scrollWhellMoveMultiplier;
    }

    public void setScrollWhellMoveMultiplier(double
scrollWhellMoveMultiplier) {
        this.scrollWhellMoveMultiplier =
scrollWhellMoveMultiplier;
    }

    public String getUserCss() {
        return userCss;
    }

    public void setUserCss(String userCss) {
        this.userCss = userCss;
    }

    public String getWindowsTerminalStarter() {
        return windowsTerminalStarter;
    }

    public void setWindowsTerminalStarter(String
windowsTerminalStarter) {
        this.windowsTerminalStarter = windowsTerminalStarter;
    }

    public String getUnixTerminalStarter() {
        return unixTerminalStarter;
    }

    public void setUnixTerminalStarter(String
unixTerminalStarter) {
        this.unixTerminalStarter = unixTerminalStarter;
    }

    public void setBackgroundColor(Color color) {
        setBackgroundColor(FxHelper.colorToHex(color));
    }

    public void setForegroundColor(Color color) {
        setForegroundColor(FxHelper.colorToHex(color));
    }

    public void setCursorColor(Color color) {
        setCursorColor(FxHelper.colorToHex(color));
    }

    public String getFontFamily() {
        return fontFamily;
    }

    public void setFontFamily(String fontFamily) {
        this.fontFamily = fontFamily;
    }

    public boolean isEnableClipboardNotice() {
```

```java
        return enableClipboardNotice;
    }

    public void setEnableClipboardNotice(boolean
enableClipboardNotice) {
        this.enableClipboardNotice = enableClipboardNotice;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return
false;
        TerminalConfig that = (TerminalConfig) o;
        return useDefaultWindowCopy ==
that.useDefaultWindowCopy &&
                clearSelectionAfterCopy ==
that.clearSelectionAfterCopy &&
                copyOnSelect == that.copyOnSelect &&
                ctrlCCopy == that.ctrlCCopy &&
                ctrlVPaste == that.ctrlVPaste &&
                fontSize == that.fontSize &&
                cursorBlink == that.cursorBlink &&
                scrollbarVisible == that.scrollbarVisible &&
                enableClipboardNotice ==
that.enableClipboardNotice &&
                Double.compare(that.scrollWhellMoveMultiplier,
scrollWhellMoveMultiplier) == 0 &&
                Objects.equals(cursorColor, that.cursorColor)
&&
                Objects.equals(backgroundColor,
that.backgroundColor) &&
                Objects.equals(foregroundColor,
that.foregroundColor) &&
                Objects.equals(fontFamily, that.fontFamily) &&
                Objects.equals(userCss, that.userCss) &&
                Objects.equals(windowsTerminalStarter,
that.windowsTerminalStarter) &&
                Objects.equals(unixTerminalStarter,
that.unixTerminalStarter);
    }

    @Override
    public int hashCode() {
        return Objects.hash(useDefaultWindowCopy,
clearSelectionAfterCopy, copyOnSelect, ctrlCCopy, ctrlVPaste,
cursorColor, backgroundColor, fontSize, foregroundColor,
cursorBlink, scrollbarVisible, enableClipboardNotice,
scrollWhellMoveMultiplier, fontFamily, userCss,
windowsTerminalStarter, unixTerminalStarter);
    }
}
```

DefaultTabGenerator class:

```
public class DefaultTabNameGenerator implements
TabNameGenerator {

    private AtomicInteger counter = new AtomicInteger();
    private String prefix = "Terminal ";

    @Override
    public String next() {
        return prefix + "#" + counter.incrementAndGet();
    }

    public AtomicInteger getCounter() {
        return counter;
    }

    public void setCounter(AtomicInteger counter) {
        this.counter = counter;
    }

    public String getPrefix() {
        return prefix;
    }

    public void setPrefix(String prefix) {
        this.prefix = prefix;
    }
}
```

FxHelper class:

```
public class FxHelper {
   public static String colorToHex(Color color) {
       return String.format("#%02X%02X%02X",
               (int) (color.getRed() * 255),
               (int) (color.getGreen() * 255),
               (int) (color.getBlue() * 255));
   }

   public static boolean askQuestion(String message) {
       CompletableFuture<Boolean> completableFuture = new
CompletableFuture<>();
       CompletableFuture.runAsync(() ->
ThreadHelper.runActionLater(() -> {
           Alert alert = new
Alert(Alert.AlertType.INFORMATION, message, ButtonType.YES,
ButtonType.NO);
           ButtonType buttonType =
alert.showAndWait().orElse(ButtonType.NO);
           completableFuture.complete(buttonType ==
ButtonType.YES);
       }));

       return completableFuture.join();
   }

   public static String askInput(String message) {
       CompletableFuture<String> completableFuture = new
CompletableFuture<>();
       CompletableFuture.runAsync(() -> {
           ThreadHelper.runActionLater(() -> {
               TextInputDialog inputDialog = new
TextInputDialog();
               inputDialog.setContentText(message);
               Optional<String> optional =
inputDialog.showAndWait();

completableFuture.complete(optional.orElse(null));
           });
       });

       return completableFuture.join();
   }
}
```

IOHelper class:

```
public class IOHelper {

    public static void close(Closeable... closables) {
        for (Closeable closable : closables) {
            try {
                closable.close();
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
        }
    }

    public static void copyLibPty(Path dataDir) throws
IOException {

        Path donePath = dataDir.resolve(".DONE");

        if (Files.exists(donePath)) {
            return;
        }

        Set<String> nativeFiles = getNativeFiles();

        for (String nativeFile : nativeFiles) {
            Path nativePath = dataDir.resolve(nativeFile);

            if (Files.notExists(nativePath)) {

Files.createDirectories(nativePath.getParent());
                InputStream inputStream =
IOHelper.class.getResourceAsStream("/" + nativeFile);
                Files.copy(inputStream, nativePath);
                close(inputStream);
            }

        }

        Files.createFile(donePath);

    }
```

```java
   private static Set<String> getNativeFiles() {

       final Set<String> nativeFiles = new HashSet<>();

       List<String> freebsd =
Arrays.asList("libpty/freebsd/x86/libpty.so",
"libpty/freebsd/x86_64/libpty.so");
       List<String> linux =
Arrays.asList("libpty/linux/x86/libpty.so",
"libpty/linux/x86_64/libpty.so");
       List<String> macosx =
Arrays.asList("libpty/macosx/x86/libpty.dylib",
"libpty/macosx/x86_64/libpty.dylib");
       List<String> win_x86 =
Arrays.asList("libpty/win/x86/winpty.dll",
"libpty/win/x86/winpty-agent.exe");
       List<String> win_x86_64 =
Arrays.asList("libpty/win/x86_64/winpty.dll",
"libpty/win/x86_64/winpty-agent.exe",
"libpty/win/x86_64/cyglaunch.exe");
       List<String> win_xp =
Arrays.asList("libpty/win/xp/winpty.dll",
"libpty/win/xp/winpty-agent.exe");


       nativeFiles.addAll(freebsd);
       nativeFiles.addAll(linux);
       nativeFiles.addAll(macosx);
       nativeFiles.addAll(win_x86);
       nativeFiles.addAll(win_x86_64);
       nativeFiles.addAll(win_xp);

       return nativeFiles;
   }
}
```

ThreadHelper:

```java
public class ThreadHelper {

   private static final Semaphore uiSemaphore = new
Semaphore(1);
   private static final ExecutorService singleExecutorService
= Executors.newSingleThreadExecutor();

   public static void runActionLater(final Runnable runnable)
```

```java
{

       if (Platform.isFxApplicationThread()) {
           runnable.run();
       } else {
           try {
               uiSemaphore.acquire();
               Platform.runLater(() -> {
                   try {
                       runnable.run();
                       releaseUiSemaphor();
                   } catch (Exception e) {
                       releaseUiSemaphor();
                       throw new RuntimeException(e);
                   }
               });
           } catch (Exception e) {
               releaseUiSemaphor();
               throw new RuntimeException(e);
           }
       }

   }

   private static void releaseUiSemaphor() {
       singleExecutorService.submit(() -> {
           uiSemaphore.release();
       });
   }

   public static void runActionLater(Runnable runnable,
boolean force) {
       if (force) {
           Platform.runLater(runnable);
       } else {
           runActionLater(runnable);
       }
   }

   public static void start(Runnable runnable) {
       Thread thread = new Thread(runnable);
       thread.start();
   }

   public static void sleep(int millis) {
       try {
           Thread.sleep(millis);
       } catch (InterruptedException e) {
           throw new RuntimeException(e);
       }
   }

   public static void awaitLatch(CountDownLatch
countDownLatch) {
       try {
           countDownLatch.await();
       } catch (InterruptedException e) {
           throw new RuntimeException(e);
       }
   }

   public static void stopExecutorService() {
       if (!singleExecutorService.isShutdown()) {
           singleExecutorService.shutdown();
       }
   }

}
```

FXMLController class:

```java
public class FXMLController implements Initializable {

    public TabPane tabPane;

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        TerminalConfig darkConfig = new TerminalConfig();
        darkConfig.setBackgroundColor(Color.rgb(16, 16, 16));
        darkConfig.setForegroundColor(Color.rgb(240, 240,
240));
        darkConfig.setCursorColor(Color.rgb(255, 0, 0, 0.5));
        TerminalConfig cygwinConfig = new TerminalConfig();

cygwinConfig.setWindowsTerminalStarter("C:\\cygwin64\\bin\\ba
sh -i");
        cygwinConfig.setFontSize(14);

        TerminalConfig defaultConfig = new TerminalConfig();

        TerminalBuilder terminalBuilder = new
TerminalBuilder(defaultConfig);
        TerminalTab terminal = terminalBuilder.newTerminal();

        tabPane.getTabs().add(terminal);

    }


}
```

Terminal:

```
public class Terminal extends TerminalView {

    private PtyProcess process;
    private final ObjectProperty<Writer> outputWriterProperty;
    private final Path terminalPath;
    private String[] termCommand;
    private final LinkedBlockingQueue<String> commandQueue;

    public Terminal() {
        this(null, null);
    }

    public Terminal(TerminalConfig terminalConfig, Path
terminalPath) {
        setTerminalConfig(terminalConfig);
        this.terminalPath = terminalPath;
        outputWriterProperty = new SimpleObjectProperty<>();
        commandQueue = new LinkedBlockingQueue<>();
    }
```

```
   @WebkitCall
   public void command(String command) {
       try {
           commandQueue.put(command);
       } catch (final InterruptedException e) {
           throw new RuntimeException(e);
       }
       ThreadHelper.start(() -> {
           try {
               final String commandToExecute =
commandQueue.poll();
               getOutputWriter().write(commandToExecute);
               getOutputWriter().flush();
           } catch (final IOException e) {
               throw new RuntimeException(e);
           }
       });
   }

   @Override
   public void onTerminalReady() {
       ThreadHelper.start(() -> {
           try {
               initializeProcess();
           } catch (final Exception e) {
               throw new RuntimeException(e);
           }
       });
   }

   private void initializeProcess() throws Exception {
       final Path dataDir = getDataDir();
       if (SystemUtils.IS_OS_WINDOWS) {
           this.termCommand =
getTerminalConfig().getWindowsTerminalStarter().split("\\s+")
;
       } else {
           this.termCommand =
getTerminalConfig().getUnixTerminalStarter().split("\\s+");
       }
```

```
        final Map<String, String> envs = new
HashMap<>(System.getenv());
        envs.put("TERM", "xterm");

        System.setProperty("PTY_LIB_FOLDER",
dataDir.resolve("libpty").toString());

        if (Objects.nonNull(terminalPath) &&
Files.exists(terminalPath)) {
            this.process = PtyProcess.exec(termCommand, envs,
terminalPath.toString());
        } else {
            this.process = PtyProcess.exec(termCommand, envs);
        }

        columnsProperty().addListener(evt -> updateWinSize());
        rowsProperty().addListener(evt -> updateWinSize());
        updateWinSize();
        String
defaultCharEncoding=System.getProperty("file.encoding");
        setInputReader(new BufferedReader(new
InputStreamReader(process.getInputStream(),
defaultCharEncoding)));
        setErrorReader(new BufferedReader(new
InputStreamReader(process.getErrorStream(),
defaultCharEncoding)));
        setOutputWriter(new BufferedWriter(new
OutputStreamWriter(process.getOutputStream(),
defaultCharEncoding)));
        focusCursor();

        countDownLatch.countDown();

        process.waitFor();
    }

    private Path getDataDir() {
        final String userHome =
System.getProperty("user.home");
        final Path dataDir =
Paths.get(userHome).resolve(".terminalfx");
        return dataDir;
    }
```

```java
    public Path getTerminalPath() {
        return terminalPath;
    }

    private void updateWinSize() {
        try {
            process.setWinSize(new WinSize(getColumns(),
getRows()));
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    public ObjectProperty<Writer> outputWriterProperty() {
        return outputWriterProperty;
    }

    public Writer getOutputWriter() {
        return outputWriterProperty.get();
    }

    public void setOutputWriter(Writer writer) {
        outputWriterProperty.set(writer);
    }

    public PtyProcess getProcess() {
        return process;
    }

}
```

TermianlView:

```java
public class TerminalView extends Pane {

    private final WebView webView;
    private final ReadOnlyIntegerWrapper columnsProperty;
    private final ReadOnlyIntegerWrapper rowsProperty;
    private final ObjectProperty<Reader> inputReaderProperty;
    private final ObjectProperty<Reader> errorReaderProperty;
```

```
   private TerminalConfig terminalConfig = new
TerminalConfig();
   protected final CountDownLatch countDownLatch = new
CountDownLatch(1);
   private static Path tempDirectory;

   static {
       Runtime.getRuntime().addShutdownHook(new Thread() {
           @Override
           public void run() {
               try {
                   if (Objects.nonNull(tempDirectory) &&
Files.exists(tempDirectory)) {

FileUtils.deleteDirectory(tempDirectory.toFile());
                   }
               } catch (IOException ex) {
                   ex.printStackTrace();
               }
           }
       });
   }

   public TerminalView() {
       initializeResources();
       webView = new WebView();
       columnsProperty = new ReadOnlyIntegerWrapper(150);
       rowsProperty = new ReadOnlyIntegerWrapper(10);
       inputReaderProperty = new SimpleObjectProperty<>();
       errorReaderProperty = new SimpleObjectProperty<>();

       inputReaderProperty.addListener((observable, oldValue,
newValue) -> {
           ThreadHelper.start(() -> {
               printReader(newValue);
           });
       });

       errorReaderProperty.addListener((observable, oldValue,
newValue) -> {
           ThreadHelper.start(() -> {
               printReader(newValue);
           });
       });


webView.getEngine().getLoadWorker().stateProperty().addListen
er((observable, oldValue, newValue) -> {
           getWindow().setMember("app", this);
       });
       webView.prefHeightProperty().bind(heightProperty());
       webView.prefWidthProperty().bind(widthProperty());

       Path htmlPath = tempDirectory.resolve("hterm.html");
       webEngine().load(htmlPath.toUri().toString());
   }

   private void initializeResources() {
       try {
           if (Objects.isNull(tempDirectory) ||
Files.notExists(tempDirectory)) {
               tempDirectory =
Files.createTempDirectory("TerminalFX_Temp");
           }
       } catch (IOException e) {
           throw new RuntimeException(e);
       }
       Path htmlPath = tempDirectory.resolve("hterm.html");
       if (Files.notExists(htmlPath)) {
           try (InputStream html =
TerminalView.class.getResourceAsStream("/hterm.html");) {
               Files.copy(html, htmlPath,
StandardCopyOption.REPLACE_EXISTING);
           } catch (IOException e) {
               throw new RuntimeException(e);
           }
       }
       Path htermJsPath =
tempDirectory.resolve("hterm_all.js");
       if (Files.notExists(htermJsPath)) {
           try (InputStream html =
TerminalView.class.getResourceAsStream("/hterm_all.js");) {
               Files.copy(html, htermJsPath,
StandardCopyOption.REPLACE_EXISTING);
```

```
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
    }

    @WebkitCall(from = "hterm")
    public String getPrefs() {
        try {
            return new
ObjectMapper().writeValueAsString(getTerminalConfig());
        } catch (final Exception e) {
            throw new RuntimeException(e);
        }
    }

    public void updatePrefs(TerminalConfig terminalConfig) {
        if (getTerminalConfig().equals(terminalConfig)) {
            return;
        }

        setTerminalConfig(terminalConfig);
        final String prefs = getPrefs();

        ThreadHelper.runActionLater(() -> {
            try {
                getWindow().call("updatePrefs", prefs);
            } catch (final Exception e) {
                throw new RuntimeException(e);
            }
        }, true);
    }

    @WebkitCall(from = "hterm")
    public void resizeTerminal(int columns, int rows) {
        columnsProperty.set(columns);
        rowsProperty.set(rows);
    }

    @WebkitCall
    public void onTerminalInit() {
        ThreadHelper.runActionLater(() -> {
            getChildren().add(webView);
        }, true);
    }

    @WebkitCall
    public void onTerminalReady() {
        ThreadHelper.start(() -> {
            try {
                focusCursor();
                countDownLatch.countDown();
            } catch (final Exception e) {
            }
        });
    }

    private void printReader(Reader bufferedReader) {
        try {
            int nRead;
            final char[] data = new char[1024];

            while ((nRead = bufferedReader.read(data, 0,
data.length)) != -1) {
                print(String.valueOf(data, 0, nRead));
            }

        } catch (final Exception e) {
            throw new RuntimeException(e);
        }
    }

    @WebkitCall(from = "hterm")
    public void copy(String text) {
        final Clipboard clipboard =
Clipboard.getSystemClipboard();
        final ClipboardContent clipboardContent = new
ClipboardContent();
        clipboardContent.putString(text);
        clipboard.setContent(clipboardContent);
    }
```

```java
public void onTerminalFxReady(Runnable onReadyAction) {
    ThreadHelper.start(() -> {
        ThreadHelper.awaitLatch(countDownLatch);

        if (Objects.nonNull(onReadyAction)) {
            ThreadHelper.start(onReadyAction);
        }
    });
}

protected void print(String text) {
    ThreadHelper.awaitLatch(countDownLatch);
    ThreadHelper.runActionLater(() -> {
        getTerminalIO().call("print", text);
    });

}

public void focusCursor() {
    ThreadHelper.runActionLater(() -> {
        webView.requestFocus();
        getTerminal().call("focus");
    }, true);
}

private JSObject getTerminal() {
    return (JSObject) webEngine().executeScript("t");
}

private JSObject getTerminalIO() {
    return (JSObject) webEngine().executeScript("t.io");
}

public JSObject getWindow() {
    return (JSObject) webEngine().executeScript("window");
}

private WebEngine webEngine() {
    return webView.getEngine();
    }

    public TerminalConfig getTerminalConfig() {
        if (Objects.isNull(terminalConfig)) {
            terminalConfig = new TerminalConfig();
        }
        return terminalConfig;
    }

    public void setTerminalConfig(TerminalConfig
terminalConfig) {
        this.terminalConfig = terminalConfig;
    }

    public ReadOnlyIntegerProperty columnsProperty() {
        return columnsProperty.getReadOnlyProperty();
    }

    public int getColumns() {
        return columnsProperty.get();
    }

    public ReadOnlyIntegerProperty rowsProperty() {
        return rowsProperty.getReadOnlyProperty();
    }

    public int getRows() {
        return rowsProperty.get();
    }

    public ObjectProperty<Reader> inputReaderProperty() {
        return inputReaderProperty;
    }

    public Reader getInputReader() {
        return inputReaderProperty.get();
    }

    public void setInputReader(Reader reader) {
        inputReaderProperty.set(reader);
    }

    public ObjectProperty<Reader> errorReaderProperty() {
        return errorReaderProperty;
    }

    public Reader getErrorReader() {
        return errorReaderProperty.get();
    }

    public void setErrorReader(Reader reader) {
        errorReaderProperty.set(reader);
    }

}
```

TerminalTab:

```
public class TerminalTab extends Tab {

    private final Terminal terminal;
    private final TabNameGenerator tabNameGenerator;
    private static final String NEW_TAB_KEY = "T";

    public TerminalTab(TerminalConfig terminalConfig,
TabNameGenerator tabNameGenerator, Path terminalPath) {
        this(new Terminal(terminalConfig, terminalPath),
tabNameGenerator);
    }

    public TerminalTab(Terminal terminal, TabNameGenerator
tabNameGenerator) {
        this.terminal = terminal;
        this.tabNameGenerator = tabNameGenerator;

        this.terminal.addEventFilter(KeyEvent.KEY_PRESSED,
event -> {

            if (event.isShortcutDown() &&
NEW_TAB_KEY.equalsIgnoreCase(event.getText())) {
                newTerminal();
            }
        });

        this.setOnCloseRequest(event -> {
            event.consume();
            closeTerminal();
        });

        final String tabName = getTabNameGenerator().next();
        setText(tabName);

        final ContextMenu contextMenu = new ContextMenu();
        final MenuItem newTab = new MenuItem("New Tab");
        final MenuItem closeTab = new MenuItem("Close");
        final MenuItem closeOthers = new MenuItem("Close
Others");
        final MenuItem closeAll = new MenuItem("Close All");

        newTab.setOnAction(this::newTerminal);
        closeTab.setOnAction(this::closeTerminal);
        closeAll.setOnAction(this::closeAllTerminal);
        closeOthers.setOnAction(this::closeOtherTerminals);

        contextMenu.getItems().addAll(newTab, closeTab,
closeOthers, closeAll);
        this.setContextMenu(contextMenu);

        setContent(terminal);
    }

    private void closeOtherTerminals(ActionEvent actionEvent)
{
        final ObservableList<Tab> tabs =
FXCollections.observableArrayList(this.getTabPane().getTabs()
);
        for (final Tab tab : tabs) {
            if (tab instanceof TerminalTab) {
                if (tab != this) {
                    ((TerminalTab) tab).closeTerminal();
                }
            }
        }
```

```java
    }

    private void closeAllTerminal(ActionEvent actionEvent) {
        final ObservableList<Tab> tabs =
FXCollections.observableArrayList(this.getTabPane().getTabs()
);
        for (final Tab tab : tabs) {
            if (tab instanceof TerminalTab) {
                ((TerminalTab) tab).closeTerminal();
            }
        }
    }

    public void newTerminal(ActionEvent... actionEvent) {
        final TerminalTab terminalTab = new
TerminalTab(getTerminalConfig(), getTabNameGenerator(),
getTerminalPath());
        getTabPane().getTabs().add(terminalTab);
        getTabPane().getSelectionModel().select(terminalTab);
    }

    public void closeTerminal(ActionEvent... actionEvent) {
        ThreadHelper.runActionLater(() -> {
            final ObservableList<Tab> tabs =
this.getTabPane().getTabs();
            if (tabs.size() == 1) {
                newTerminal(actionEvent);
            }
            tabs.remove(this);

            destroy();
        });
    }

    public void destroy() {
        ThreadHelper.start(() -> {
            while (Objects.isNull(getProcess())) {
                ThreadHelper.sleep(250);
            }
            getProcess().destroy();
            IOHelper.close(getInputReader(), getErrorReader(),
getOutputWriter());
```

```
        });
    }

    public void onTerminalFxReady(Runnable onReadyAction) {
        terminal.onTerminalFxReady(onReadyAction);
    }

    public TabNameGenerator getTabNameGenerator() {
        return tabNameGenerator;
    }

    public Path getTerminalPath() {
        return terminal.getTerminalPath();
    }

    public TerminalConfig getTerminalConfig() {
        return terminal.getTerminalConfig();
    }

    public PtyProcess getProcess() {
        return terminal.getProcess();
    }

    public Reader getInputReader() {
        return terminal.getInputReader();
    }

    public Reader getErrorReader() {
        return terminal.getErrorReader();
    }

    public Writer getOutputWriter() {
        return terminal.getOutputWriter();
    }

    public Terminal getTerminal() {
        return terminal;
    }
}
```

TerminalBuilder:

```
public class TerminalBuilder {

   private Path terminalPath;
   private TerminalConfig terminalConfig;
   private TabNameGenerator nameGenerator;

   public TerminalBuilder() {
   }

   public TerminalBuilder(TerminalConfig terminalConfig) {
       this.terminalConfig = terminalConfig;
   }

   public TerminalConfig getTerminalConfig() {
       if (Objects.isNull(terminalConfig)) {
           terminalConfig = new TerminalConfig();
       }
       return terminalConfig;
   }

   public void setTerminalConfig(TerminalConfig
terminalConfig) {
       this.terminalConfig = terminalConfig;
   }

   public TabNameGenerator getNameGenerator() {
       if (Objects.isNull(nameGenerator)) {
           nameGenerator = new DefaultTabNameGenerator();
       }
       return nameGenerator;
   }

   public void setNameGenerator(TabNameGenerator
nameGenerator) {
       this.nameGenerator = nameGenerator;
   }

   public Path getTerminalPath() {
       return terminalPath;
```

```
    }

    public void setTerminalPath(Path terminalPath) {
        this.terminalPath = terminalPath;
    }

    public TerminalTab newTerminal() {
        return new TerminalTab(getTerminalConfig(),
getNameGenerator(), getTerminalPath());
    }
}
```

TerminalAppStarter class:

```
public class TerminalAppStarter extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        InputStream sceneStream =
TerminalAppStarter.class.getResourceAsStream("/fxml/Terminal_
Scene.fxml");
        FXMLLoader loader = new FXMLLoader();
        Parent root = loader.load(sceneStream);

        Scene scene = new Scene(root);

scene.getStylesheets().add(TerminalAppStarter.class.getResour
ce("/styles/Styles.css").toExternalForm());

        stage.setTitle("Terminal");
        stage.setScene(scene);
        stage.show();
    }

    @Override
    public void stop() throws Exception {
        ThreadHelper.stopExecutorService();
        Platform.exit();
        System.exit(0);
    }
    public static void main(String[] args) {
        launch(args);
    }

}
```