Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 3

з дисципліни «Теорія розробки прогрманого забезпечення»

Тема: «Основи проектування розгортання.»

Варіант «Powershell terminal»

Виконав:

студент групи ІА-32
Бечке Олексій Ігорович

Перевірив:

Мягкий Михайло Юрійович

Київ 2025

Зміст

Вступ

Тема: Основи проектування розгортання.

Мета: Навчитися проєктувати діаграми розгортання та компонентів для системи що проєктується, а також розробляти діаграми взаємодії, а саме діаграми послідовностей, на основі сценаріїв зроблених в попередній лабораторній роботі.

Тема проєкту: Powershell terminal (strategy, command, abstract factory, bridge, interpreter, client-server). Термінал для powershell повинен нагадувати типовий термінал з можливістю налаштування кольорів синтаксичних конструкцій, розміру вікна, фону вікна, а також виконання команд powershell і виконуваних файлів, а також працювати в декількох вікнах терміналу (у вкладках або одночасно шляхом розділення вікна).

Теоретичні відомості

Діаграми розгортання представляють фізичне розташування системи, показуючи, на якому фізичному обладнанні запускається та чи інша складова програмного забезпечення

Вузол (node) – це те, що може містити програмне забезпечення. Вузли бувають двох типів. Пристрій (device) – це фізичне обладнання: комп'ютер або пристрій, пов'язаний із системою. Середовище виконання (execution environment) – це програмне забезпечення, яке саме може включати інше програмне забезпечення, наприклад операційну систему або процес-контейнер (наприклад, вебсервер).

Між вузлами можуть стояти зв'язки, які зазвичай зображують у вигляді прямої лінії. Як і на інших діаграмах, у зв'язків можуть бути атрибути множинності (для показання, наприклад, підключення 2х і більше клієнтів до одного сервера) і назва. У назві, як правило, міститься спосіб зв'язку між двома 40 вузлами – це може бути назва протоколу (HTTP, IPC) або технологія, що використовується для забезпечення взаємодії вузлів (.NET Remoting, WCF).

Діаграма компонентів UML є представленням проєктованої системи, розбитої на окремі модулі [3]. Залежно від способу поділу на модулі розрізняють три види діаграм компонентів:

- логічні;

- фізичні;
- виконувані

Діаграма послідовностей (Sequence Diagram) – це один із типів діаграм у моделюванні UML (Unified Modeling Language), який використовується для моделювання взаємодії між об'єктами системи у певній послідовності часу. Вона відображає, як об'єкти обмінюються повідомленнями, показуючи порядок і логіку виконання операцій.

Діаграма складається з таких основних елементів:

- Актори
- Об'єкти або класи
- Повідомлення
- Активності
- Контрольні структури

Основні кроки створення діаграми послідовностей: • визначити акторів і об'єкти, які беруть участь у сценарії; • побудувати їхні лінії життя; • розробити послідовність передачі повідомлень між об'єктами; • додати умовні блоки або цикли за необхідності.

Хід роботи

Завдання

- Ознайомитись з короткими теоретичними відомостями.
- Проаналізувати діаграми створені в попередній лабораторній роботі а також тему системи та спроєктувати діаграму розгортання використання відповідно до обраної теми лабораторного циклу.
- Розробити діаграму компонентів для проєктованої системи.
- Розробити діаграму розгортання для проєктованої системи.
- Розробити як мінімум дві діаграми послідовностей для сценаріїв прописаних в попередній лабораторній роботі.
- На основі спроєктованих діаграм розгортання та компонентів доопрацювати програмну частину системи. Реалізація системи, додатково до попередньої реалізації, повинна містити як мінімум дві візуальні форми. В системі вже повинен бути повністю реалізована архітектура (повний цикл роботи з даними від вводу на формі до збереження їх в БД і подальшій виборці з БД та відображенням на UI).
- Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму розгортання з описом, діаграму компонентів системи з описом, діаграми послідовностей, а також вихідний код системи, який було додано в цій лабораторній роботі.

Діаграма розгортання системи

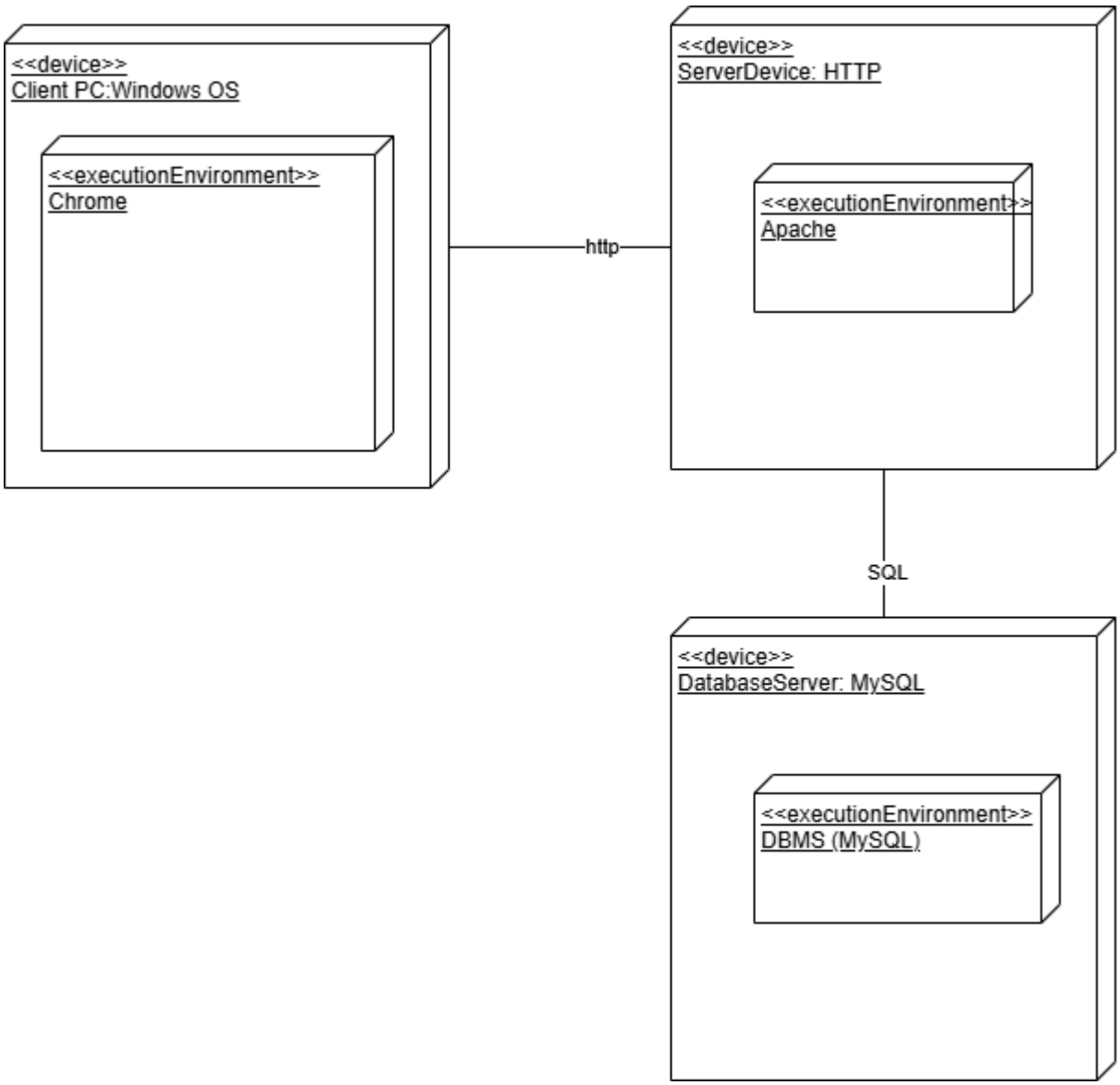

Рис. 1 – Діаграма розгортання системи (Deployment diagram)

Опис діаграми розгортання

Ця діаграма ілюструє фізичну архітектуру та розміщення програмних компонентів системи «Powershell terminal». Оскільки система є настільним додатком, вся її інфраструктура розгорнута на одній клієнтській машині.

Вузли (Nodes)

На діаграмі представлено два типи вузлів, вкладених один в одного:

1. **<<device>> Робоча станція користувача**
    - Це головний апаратний вузол, що представляє фізичний комп'ютер користувача.
    - **<<executionEnvironment>> Chrome**

- Це вузол середовища виконання, який працює у веб браузері користувача.

2. **<<device>> ServerDevice**
   - Це вузол, що представляє сервер, до якого звертатиметься коистувач та від якого отримуватиме відповіді.
   - **<<executionEnvironment>> Apache**
   - Це вузлол середовища виконання, який працює на сервері, використовуючи його апаратні ресурси
3. **<<device>> DatabaseSever**
   - Це вузол, що представляє сервер бази даних, який зберігатиме базу даних.
   - **<<executionEnvironment>> DBMS MySQL**
   - Це вузол середовища, який відповідає за СУБД MySQL.

Діаграма компонентів

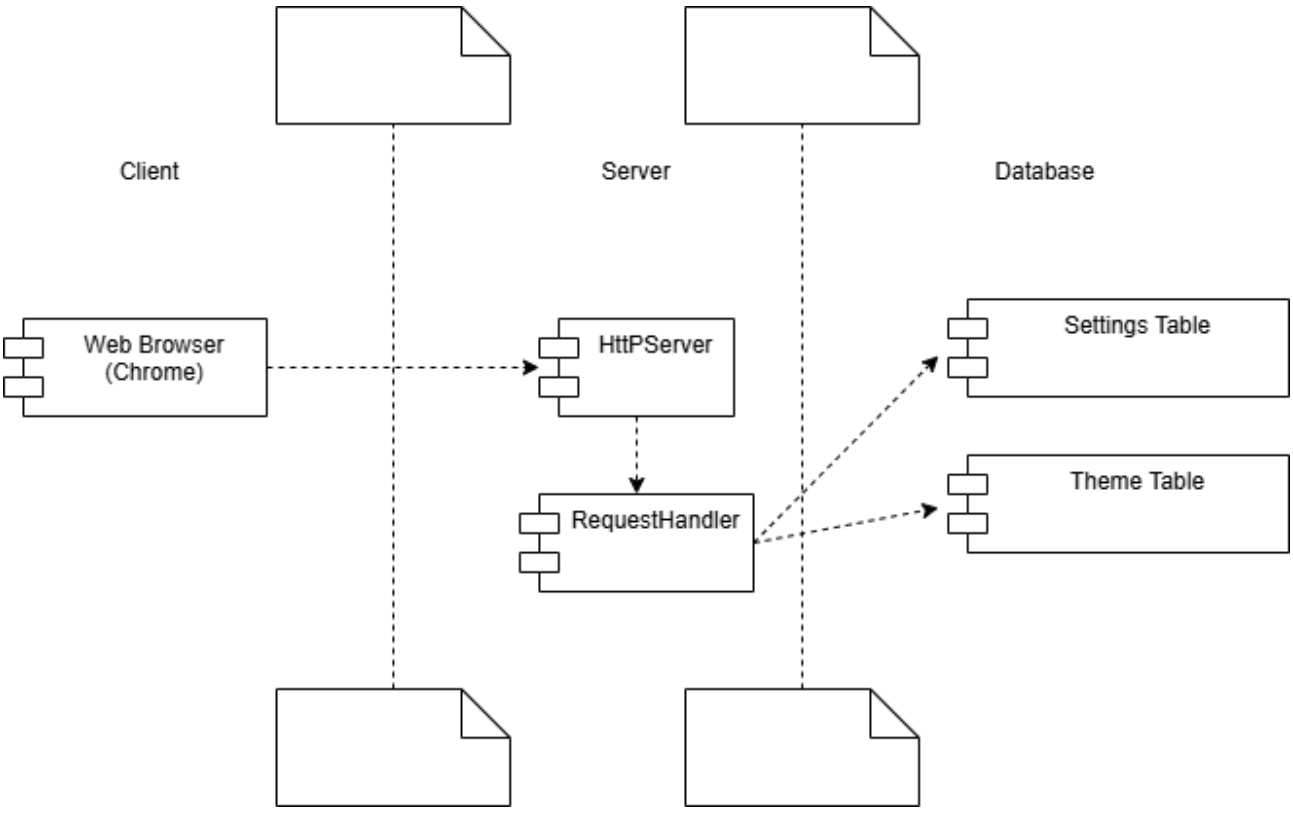

Рис. 2 – Діаграма компонентів (Component diagram)

Ця діаграма компонентів представляє **фізичну** архітектуру системи «Powershell terminal». Вона візуалізує основні програмні файли, з яких складається система, їхні технологічні ролі та залежності між ними.

Опис діаграми компонентів

Ця діаграма компонентів представляє **виконувану/фізичну** архітектуру системи «Powershell terminal». Вона візуалізує основні програмні файли (артефакти), з яких складається система, їхні технологічні ролі (стереотипи) та залежності між ними.

Для кращого розуміння архітектури, всі компоненти логічно згруповані у три стовпці: Client Application (клієнтська частина), Execution Services (виконавчі сервіси) та Database Services (сервіси бази даних).

1. Структура системи (Рівні)

Система поділена вертикальними пунктирними лініями на три основні зони:

- **Client (Клієнт):** Сторона користувача.
- **Server (Сервер):** Сторона обробки бізнес-логіки.
- **Database (База даних):** Сторона зберігання даних.

2. Компоненти системи

На діаграмі зображено 5 ключових компонентів (позначені прямокутниками з двома маленькими виступами зліва):

- Web Browser (Chrome): Знаходиться на стороні клієнта. Це точка входу в систему, через яку користувач взаємодіє з програмою.
- HttpServer: Знаходиться на стороні сервера. Цей компонент відповідає за отримання HTTP-запитів від клієнта.
- RequestHandler (Обробник запитів): Також на сервері. Це компонент, який виконує логіку обробки отриманого запиту.
- Settings Table (Таблиця налаштувань): Компонент бази даних, що відповідає за зберігання налаштувань.
- Theme Table (Таблиця тем): Компонент бази даних, що відповідає за зберігання інформації про теми оформлення.

3. Залежності (Зв'язки)

Зв'язки на діаграмі показані пунктирними стрілками (-->), які позначають залежність "використовує" або "залежить від".

- Web Browser -> HttpServer: Браузер ініціює з'єднання, надсилаючи запит до веб-сервера.
- HttpServer -> RequestHandler: Веб-сервер передає отриманий запит до спеціального обробника (RequestHandler) для виконання конкретних дій.

- RequestHandler -> Database Tables: Обробник запитів звертається до бази даних для читання або запису інформації. На діаграмі показано, що він взаємодіє конкретно з:
    - Settings Table (для завантаження конфігурації користувача).
    - Theme Table (для визначення візуальної теми інтерфейсу).

Діаграма послідовностей

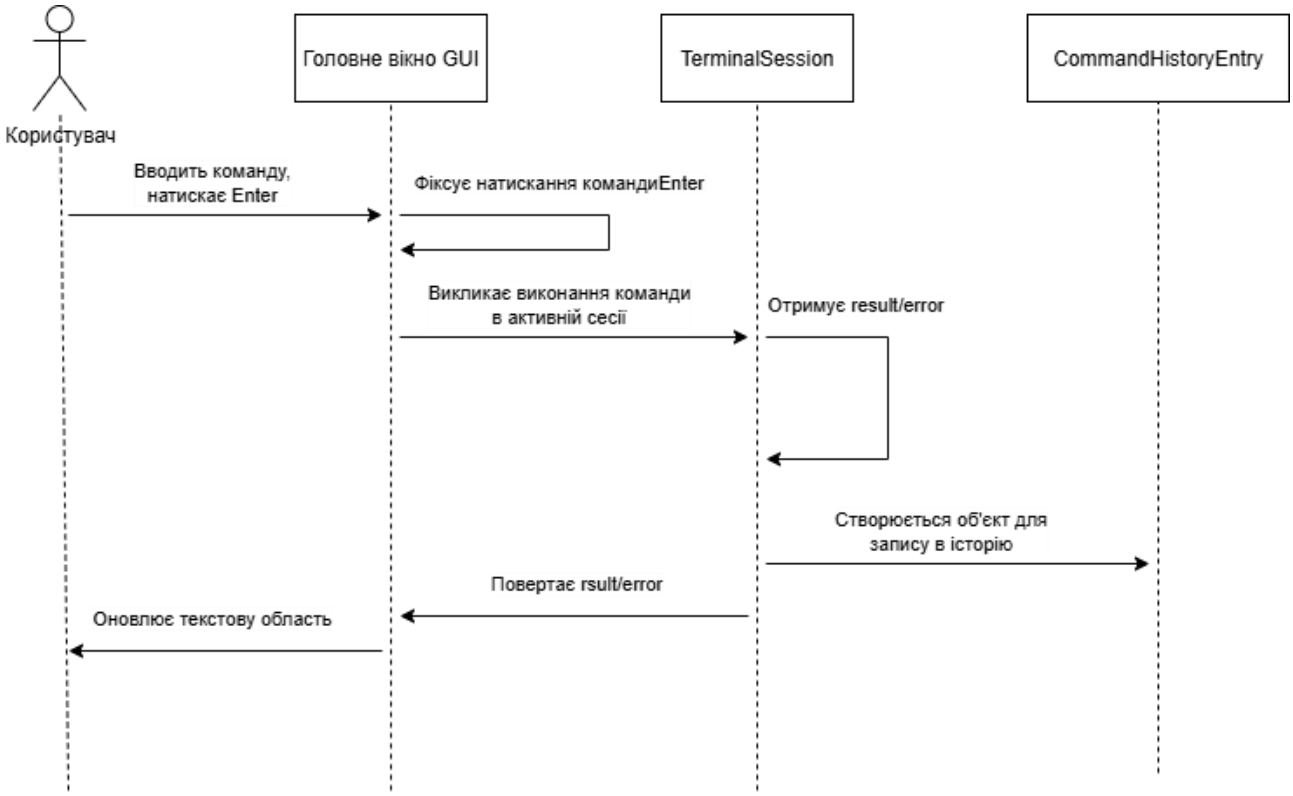

Рис. 3 – діаграма послідовності (Sequence diagram) – Виконання команди PowerShell

Перебіг подій:

1. Користувач вводить текстову команду з клавіатури (напр., Get-ChildItem -Path C:\).
2. Система відображає символи, що вводяться, в активній текстовій області.
3. Користувач натискає клавішу Enter для підтвердження виконання.
4. Система отримує введений текстовий рядок.
5. Система надсилає рядок команди у стандартний потік вводу (stdin) відповідного процесу powershell.exe.
6. Система очікує та зчитує дані зі стандартного потоку виводу (stdout) процесу powershell.exe.
7. Система відображає отримані дані у текстовій області, додаючи їх після введеної команди.
8. Система повертається в режим очікування вводу від користувача.

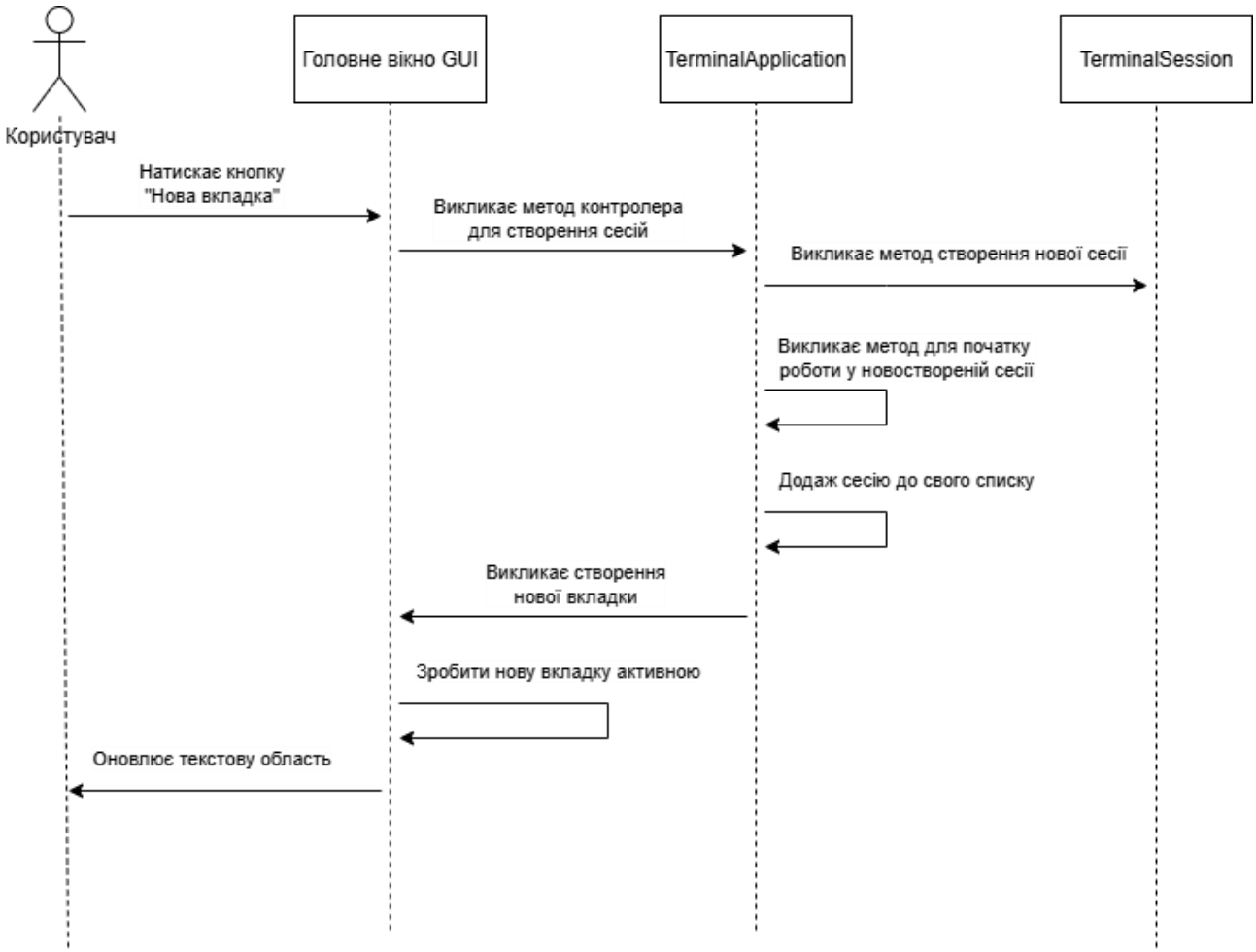

Рис. 4 – діаграма послідовності (Sequence diagram) – Відкриття нової вкладки

Перебіг подій:

1. Користувач натискає на елемент керування "Нова вкладка" (зазвичай кнопка +).
2. Система створює новий об'єкт TerminalSession.
3. Система запускає новий процес powershell.exe в операційній системі.
4. Система пов'язує потік вводу/виводу нового процесу з об'єктом TerminalSession.
5. Система додає новий віджет вкладки до TabPanel у головному вікні.
6. Система робить новостворену вкладку активною (перемикає фокус на неї).
7. Система відображає початкове запрошення до вводу від PowerShell у текстовій області нової вкладки.

Вихідні коди класів системи

```
import java.sql.*;


// Цей клас відповідає за повний цикл роботи з БД
```

```
public class SettingsRepository {

    private static final String DB_URL = "jdbc:sqlite:TerminalApp.db";

    // Метод для ініціалізації БД при першому запуску
    public void initializeDatabase() {
        String sqlTheme = "CREATE TABLE IF NOT EXISTS Theme (" +
                "name TEXT PRIMARY KEY, " +
                "backgroundColor TEXT, " +
                "textColor TEXT, " +
                "keywordColor TEXT, " +
                "parameterColor TEXT, " +
                "stringColor TEXT, " +
                "variableColor TEXT" +
                ");";

        String sqlSettings = "CREATE TABLE IF NOT EXISTS Settings (" +
                "id INTEGER PRIMARY KEY, " +
                "fontName TEXT, " +
                "fontSize INTEGER, " +
                "activeThemeName TEXT, " +
                "FOREIGN KEY (activeThemeName) REFERENCES Theme(name)" +
                ");";

        String sqlInsertDefaultTheme = "INSERT OR IGNORE INTO Theme VALUES
" +
                "('Default Dark', '#2B2B2B', '#A9B7C6', '#CC7832', '#BDB76B', '#6A8759',
'#9876AA');";

        String sqlInsertDefaultSettings = "INSERT OR IGNORE INTO Settings
VALUES (1, 'Consolas', 14, 'Default Dark');";
```

```
    try (Connection conn = DriverManager.getConnection(DB_URL);
         Statement stmt = conn.createStatement()) {

        stmt.execute(sqlTheme);
        stmt.execute(sqlSettings);
        stmt.execute(sqlInsertDefaultTheme);
        stmt.execute(sqlInsertDefaultSettings);

        System.out.println("Базу даних ініціалізовано.");

    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}

// Завантаження налаштувань з БД
public Settings loadSettings() {
    String sql = "SELECT s.fontName, s.fontSize, t.* FROM Settings s " +
                 "JOIN Theme t ON s.activeThemeName = t.name WHERE s.id = 1";

    try (Connection conn = DriverManager.getConnection(DB_URL);
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {

        if (rs.next()) {
            Theme theme = new Theme(
                    rs.getString("name"),
                    rs.getString("backgroundColor"),
                    rs.getString("textColor"),
```

```
                rs.getString("keywordColor"),
                rs.getString("parameterColor"),
                rs.getString("stringColor"),
                rs.getString("variableColor")
            );
            return new Settings(theme, rs.getString("fontName"), rs.getInt("fontSize"));
        }
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
    // Повертаємо дефолтні, якщо завантаження не вдалося
    return new Settings(Theme.createDefaultTheme(), "Consolas", 14);
}

// Збереження налаштувань в БД (повний цикл)
public void saveSettings(Settings settings) {
    // Ми припускаємо, що теми зберігаються окремо,
    // а налаштування лише посилаються на них.
    // Для простоти, оновимо лише шрифт та розмір.

    String sql = "UPDATE Settings SET fontName = ?, fontSize = ?, activeThemeName = ? WHERE id = 1";

    try (Connection conn = DriverManager.getConnection(DB_URL);
         PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(1, settings.getFontName());
        pstmt.setInt(2, settings.getFontSize());
        pstmt.setString(3, settings.getActiveTheme().getName());
        pstmt.executeUpdate();
```

```
            System.out.println("Налаштування збережено в БД.");

        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

```
import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;
import java.util.function.Consumer;

public class TerminalSession {
    private final UUID id;
    private final List<CommandHistoryEntry> history;
    private Process process; // Реальний процес powershell.exe
    private BufferedWriter processInput;

    // Callbacks для відправки даних в GUI асинхронно
    private Consumer<String> onOutput;
    private Consumer<String> onError;

    public TerminalSession(Consumer<String> onOutput, Consumer<String> onError)
{
        this.id = UUID.randomUUID();
```

```
        this.history = new ArrayList<>();
        this.onOutput = onOutput;
        this.onError = onError;
    }

    // Метод start() тепер реально запускає процес
    public void start() {
        System.out.println("Запуск сесії " + id);
        try {
            ProcessBuilder pb = new ProcessBuilder("powershell.exe", "-NoLogo", "-NoExit", "-Command", "-");
            this.process = pb.start();

            this.processInput = new BufferedWriter(new OutputStreamWriter(process.getOutputStream()));

            // Асинхронне читання stdout (в окремому потоці, щоб не блокувати GUI)
            new Thread(() -> {
                try (BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()))) {
                    String line;
                    while ((line = reader.readLine()) != null) {
                        onOutput.accept(line);
                    }
                } catch (IOException e) {
                    onError.accept("Помилка читання stdout: " + e.getMessage());
                }
            }).start();

            // Асинхронне читання stderr
             new Thread(() -> {
```

```
            try (BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getErrorStream()))) {
                String line;
                while ((line = reader.readLine()) != null) {
                    onError.accept(line);
                }
            } catch (IOException e) {
                onError.accept("Помилка читання stderr: " + e.getMessage());
            }
        }).start();

    } catch (IOException e) {
        onError.accept("Не вдалося запустити PowerShell: " + e.getMessage());
    }
}

// Метод executeCommand() тепер реально надсилає команду
public void executeCommand(String commandText) {
    if (processInput == null || !process.isAlive()) {
        onError.accept("Сесія не активна.");
        return;
    }

    try {
        processInput.write(commandText);
        processInput.newLine();
        processInput.flush();

        // Імітуємо додавання в історію (в реальному житті це було б складніше)
        // Ми не маємо прямого "результату", оскільки він асинхронний
```

```
            addHistoryEntry(new CommandHistoryEntry(commandText, "...", false));

        } catch (IOException e) {
            onError.accept("Помилка відправки команди: " + e.getMessage());
        }
    }

    // ... (решта геттерів та addHistoryEntry залишаються як у ЛР2) ... [cite: 264, 274, 277]

    public void close() {
        System.out.println("Закриття сесії " + id);
        if (this.process != null) {
            this.process.destroy();
        }
    }

    public UUID getId() { return id; }
    public List<CommandHistoryEntry> getHistory() { return history; }
    public void addHistoryEntry(CommandHistoryEntry entry) { this.history.add(entry); }
}
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;

public class TerminalApplication {
```

```
    private List<TerminalSession> sessions;
    private Settings settings;
    private SettingsRepository repository; // Залежність від Репозиторію

    public TerminalApplication() {
        this.sessions = new ArrayList<>();
        this.repository = new SettingsRepository();

        // Ініціалізуємо БД при першому запуску
        this.repository.initializeDatabase();

        loadSettings(); // Завантажуємо реальні налаштування
    }

    // Метод loadSettings() тепер працює з БД
    public void loadSettings() {
        System.out.println("Завантаження налаштувань з БД...");
        this.settings = repository.loadSettings();
    }

    // Метод saveSettings() тепер працює з БД
    public void saveSettings() {
        System.out.println("Збереження налаштувань в БД...");
        repository.saveSettings(this.settings);
    }

    // Метод createSession() тепер приймає "callback" з GUI
    public TerminalSession createSession(Consumer<String> onOutput, Consumer<String> onError) {
        TerminalSession newSession = new TerminalSession(onOutput, onError);
```

```
        newSession.start();
        this.sessions.add(newSession);
        System.out.println("Створено нову сесію. Всього сесій: " + sessions.size());
        return newSession;
    }

    public void closeSession(TerminalSession session) {
        if (session != null) {
            session.close();
            this.sessions.remove(session);
            System.out.println("Закрито сесію. Залишилось сесій: " + sessions.size());
        }
    }

    public Settings getSettings() {
        return this.settings;
    }
}

// Класи Theme, Settings, CommandHistoryEntry залишаються з ЛР2



import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Main extends Application {
```

```
    @Override
    public void start(Stage primaryStage) throws Exception{
        // Створюємо екземпляр нашої логіки
        TerminalApplication terminalApp = new TerminalApplication();

        // Завантажуємо FXML для головного вікна
        FXMLLoader loader = new FXMLLoader(getClass().getResource("MainWindow.fxml"));

        // Передаємо логіку в контролер GUI
        loader.setControllerFactory(c -> new MainWindowController(terminalApp));

        Parent root = loader.load();

        primaryStage.setTitle("PowerShell Terminal");
        primaryStage.setScene(new Scene(root, 800, 600));
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}



import javafx.application.Platform;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
```

```
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.input.KeyCode;
import javafx.scene.layout.VBox;
import javafx.stage.Modality;
import javafx.stage.Stage;
import java.io.IOException;

public class MainWindowController {

    @FXML
    private TabPane tabPane;

    private TerminalApplication terminalApp;

    public MainWindowController(TerminalApplication terminalApp) {
        this.terminalApp = terminalApp;
    }

    @FXML
    private void initialize() {
        // Створюємо першу вкладку при запуску
        addNewTab();

        // Одразу застосовуємо стилі з БД
        applySettings();
    }

    @FXML
```

```
    private void handleNewTab() {
        addNewTab();
    }

    @FXML
    private void handleOpenSettings() {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("SettingsWindow.fxml"));

            // Передаємо ті ж самі налаштування у вікно налаштувань
            loader.setControllerFactory(c -> new SettingsWindowController(terminalApp));

            Parent page = loader.load();
            Stage dialogStage = new Stage();
            dialogStage.setTitle("Налаштування");
            dialogStage.initModality(Modality.WINDOW_MODAL);
            dialogStage.setScene(new Scene(page));

            // Показуємо і чекаємо, поки користувач його закриє
            dialogStage.showAndWait();

            // Після закриття вікна налаштувань, оновлюємо вигляд
            applySettings();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    private void addNewTab() {
        TextArea outputArea = new TextArea();
        outputArea.setEditable(false);

        TextField inputField = new TextField();

        VBox terminalView = new VBox(outputArea, inputField);

        Tab tab = new Tab("PowerShell", terminalView);

        // Створюємо сесію і "прив'язуємо" її до нашого GUI
        TerminalSession session = terminalApp.createSession(
            // onOutput
            (line) -> Platform.runLater(() -> outputArea.appendText(line + "\n")),
            // onError
            (line) -> Platform.runLater(() -> outputArea.appendText("ПОМИЛКА: " +
line + "\n"))
        );

        // Обробник вводу команди
        inputField.setOnKeyPressed(event -> {
            if (event.getCode() == KeyCode.ENTER) {
                String command = inputField.getText();
                outputArea.appendText("PS> " + command + "\n");
                session.executeCommand(command);
                inputField.clear();
            }
        });
```

```
        // Зберігаємо сесію, щоб мати змогу її закрити
        tab.setUserData(session);

        tab.setOnClosed(e -> terminalApp.closeSession((TerminalSession) tab.getUserData()));

        tabPane.getTabs().add(tab);
        tabPane.getSelectionModel().select(tab);
    }

    // Застосовуємо налаштування з БД до GUI
    private void applySettings() {
        Settings settings = terminalApp.getSettings();
        Theme theme = settings.getActiveTheme();

        String style = String.format(
            "-fx-font-family: '%s'; -fx-font-size: %dpx;",
            settings.getFontName(),
            settings.getFontSize()
        );

        // Цей стиль можна застосувати до outputArea та inputField
        // (в реальному коді це робиться через CSS, але для лаби так простіше)

        // Приклад зміни фону (потрібно застосувати до outputArea)
        // outputArea.setStyle("-fx-control-inner-background: " + theme.getBackgroundColor() + ";");

        System.out.println("Налаштування застосовано до GUI. Фон: " + theme.getBackgroundColor());
    }
```

```
}



import javafx.fxml.FXML;
import javafx.scene.control.ComboBox;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

public class SettingsWindowController {

    @FXML
    private ComboBox<String> fontComboBox; // Має бути заповнений шрифтами
    @FXML
    private TextField fontSizeField;
    // ... тут мають бути ColorPicker для кольорів теми ...

    private TerminalApplication terminalApp;

    public SettingsWindowController(TerminalApplication terminalApp) {
        this.terminalApp = terminalApp;
    }

    @FXML
    private void initialize() {
        // Крок "виборки з БД та відображенням на UI"
        Settings settings = terminalApp.getSettings();

        // fontComboBox.getItems().addAll("Consolas", "Courier New", "Arial"); // Приклад
```

```
    fontComboBox.setValue(settings.getFontName());
    fontSizeField.setText(String.valueOf(settings.getFontSize()));

    //... заповнити ColorPicker'ї з settings.getActiveTheme() ...
}

@FXML
private void handleSave() {
    // Крок "вводу на формі"
    String fontName = fontComboBox.getValue();
    int fontSize = Integer.parseInt(fontSizeField.getText());

    // Оновлюємо об'єкт Settings
    Settings settings = terminalApp.getSettings();
    settings.setFontName(fontName);
    settings.setFontSize(fontSize);

    // ... оновити тему ...

    // Крок "збереження їх в БД"
    terminalApp.saveSettings();

    closeWindow();
}

@FXML
private void handleCancel() {
    closeWindow();
}
```

```
    private void closeWindow() {
        Stage stage = (Stage) fontSizeField.getScene().getWindow();
        stage.close();
    }
}
```

Питання до лабораторної роботи

1. Що собою становить діаграма розгортання?

Діаграми розгортання представляють фізичне розташування системи, показуючи, на якому фізичному обладнанні запускається та чи інша складова програмного забезпечення

2. Які бувають види вузлів на діаграмі розгортання?

Вузол (node) – це те, що може містити програмне забезпечення. Вузли бувають двох типів. Пристрій (device) – це фізичне обладнання: комп'ютер або пристрій, пов'язаний із системою. Середовище виконання (execution environment) – це програмне забезпечення, яке саме може включати інше програмне забезпечення, наприклад операційну систему або процес-контейнер (наприклад, вебсервер).

3. Які бувають зв’язки на діаграмі розгортання?

Між вузлами можуть стояти зв'язки, які зазвичай зображують у вигляді прямої лінії. Як і на інших діаграмах, у зв'язків можуть бути атрибути множинності (для показання, наприклад, підключення 2x і більше клієнтів до одного сервера) і назва. У назві, як правило, міститься спосіб зв'язку між двома 40 вузлами – це може бути назва протоколу (HTTP, IPC) або технологія, що використовується для забезпечення взаємодії вузлів (.NET Remoting, WCF).

4. Які елементи присутні на діаграмі компонентів?

- Компонент (Component): Модульна, замінна частина системи, що інкапсулює свою поведінку та дані.
- Інтерфейс (Interface): Визначає набір операцій, які компонент надає або вимагає.
- Порт (Port): є точкою взаємодії та групує логічно пов'язані інтерфейси.
- Пакети (Package): групування компонентів.
- Зв’язки (Dependency, Association): взаємозв’язки між компонентами.

5. Що становлять собою зв’язки на діаграмі компонентів?

Пунктирна стрілка, що показує, що один компонент (клієнт) залежить від іншого (постачальника).

6. Які бувають види діаграм взаємодії?

Діаграми взаємодії (Interaction Diagrams) показують динамічну поведінку системи. До цієї групи належать:

- Діаграма послідовностей (Sequence Diagram)
- Діаграма комунікації (Communication Diagram) (раніше – діаграма кооперації)
- Діаграма огляду взаємодії (Interaction Overview Diagram)
- Діаграма часових залежностей (Timing Diagram)

7. Для чого призначена діаграма послідовностей?

Це один із типів діаграм у моделюванні UML, який використовується для моделювання взаємодії між об'єктами системи у певній послідовності часу.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

- Актори
- Об'єкти або класи
- Повідомлення
- Активності
- Контрольні структури

9. Як діаграми послідовностей пов’язані з діаграмами варіантів використання?

Діаграма послідовностей є деталізацією одного сценарію (scenario) варіанту використання (Use Case). Варіант використання (напр., "Виконати команду") описує що система робить для актора , а діаграма послідовностей показує як саме об'єкти всередині системи взаємодіють між собою, щоб реалізувати цей сценарій

10. Як діаграми послідовностей пов’язані з діаграмами класів?

- Учасники (лінії життя) на діаграмі послідовностей – це, як правило, екземпляри класів, визначених на діаграмі класів.
- Повідомлення, що надсилаються між лініями життя, відповідають операціям (методам), які визначені у цих класах.

Висновок

Під час виконання лабораторної роботи я навчився проєктувати діаграми розгортання та компонентів для системи що проєктується, а також розробляти діаграми взаємодії, а саме діаграми послідовностей, на основі сценаріїв зроблених в попередній лабораторній роботі.