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 №3

«Основи проектування»

Виконав
студент групи ІА-34:
Сльота Максим

Перевірив:
Мягкий М. Ю.

Київ - 2025

Тема: Основи проектування розгортання.

Мета роботи: 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.

Теоретичні відомості:

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

Головними елементами діаграми є вузли, пов'язані інформаційними шляхами. **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).

Вузли можуть містити артефакти (artifacts), які є фізичним уособленням програмного забезпечення; зазвичай це файли.

Артефакти часто є реалізацією компонентів. Це можна показати, задавши значення-мітки всередині прямокутників артефактів. Основні види артефактів: вихідні файли, виконувані файли, сценарії, таблиці баз даних, документи, результати процесу розробки, UML-моделі.

Діаграма компонентів UML є представленням проєктованої системи, розбитої на окремі модулі. 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 (Actors): Зазвичай позначаються піктограмами або назвами. Це користувачі чи інші системи, які взаємодіють із системою. Актори можуть бути зовнішніми стосовно моделювання системи

Об'єкти або класи: Розміщуються горизонтально на діаграмі. Вони позначаються прямокутниками з іменем об'єкта або класу під прямокутником. Кожен об'єкт має «життєвий цикл», який представлений вертикальною пунктирною лінією (лінія життя).

Повідомлення: Це лінії зі стрілками, які з'єднують об'єкти. Вони показують передачу повідомлень чи виклик методів. Стрілка може бути синхронною (звичайна стрілка) або асинхронною (лінія з відкритим трикутником) та з пунктирною лінією, що показує повернення результату.

Активності: Вказують періоди, протягом яких об'єкт виконує певну дію. На діаграмі це позначається прямокутником, накладеним на лінію життя.

Контрольні структури: Використовуються для відображення умов, циклів або альтернативних сценаріїв.

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.

Обрана тема: System activity monitor

Монітор активності системи повинен зберігати і запам'ятовувати статистику використовуваних компонентів системи, включаючи навантаження на процесор, обсяг займаної оперативної пам'яті, натискання клавіш на клавіатурі, дії миші (переміщення, натискання), відкриття вікон і зміна вікон; будувати звіти про використання комп'ютера за різними критеріями (% часу перебування у веб-браузері, середнє навантаження на процесор по годинах, середній час роботи комп'ютера по днях і т.д.); правильно поводитися з «простоюванням» системи – відсутністю користувача.

Хід роботи:

1. Діаграма розгортання

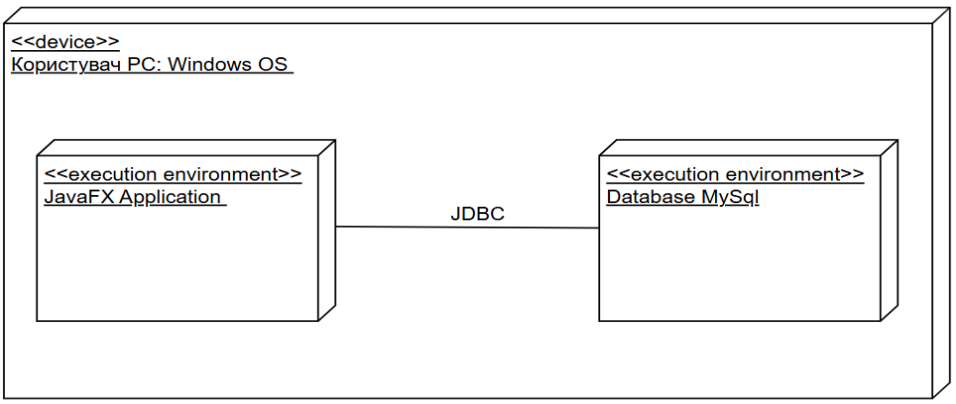

Рис 1. Діаграма розгортання

2. Діаграма компонентів

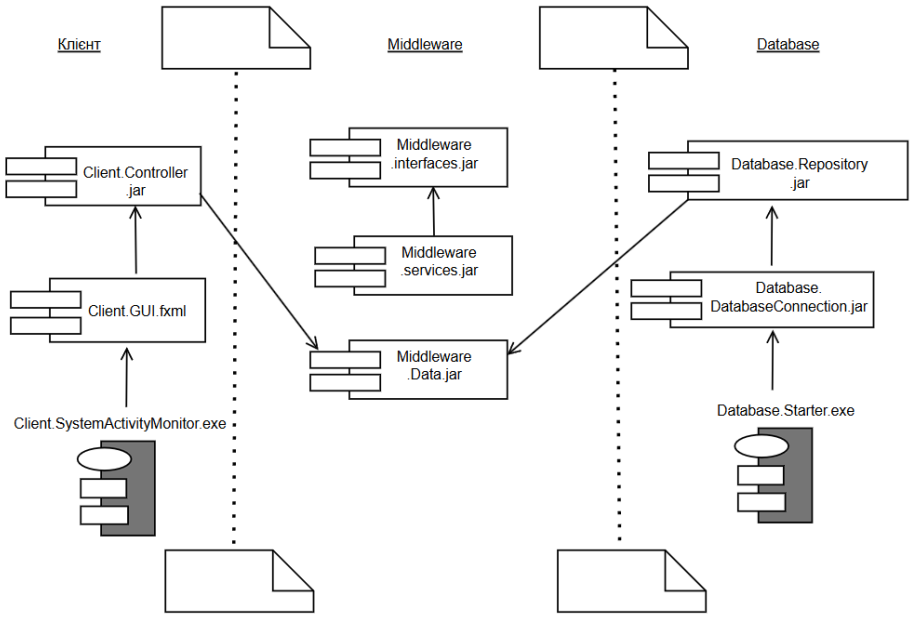

Рис 2. Діаграма компонентів

3. Діаграми послідовностей

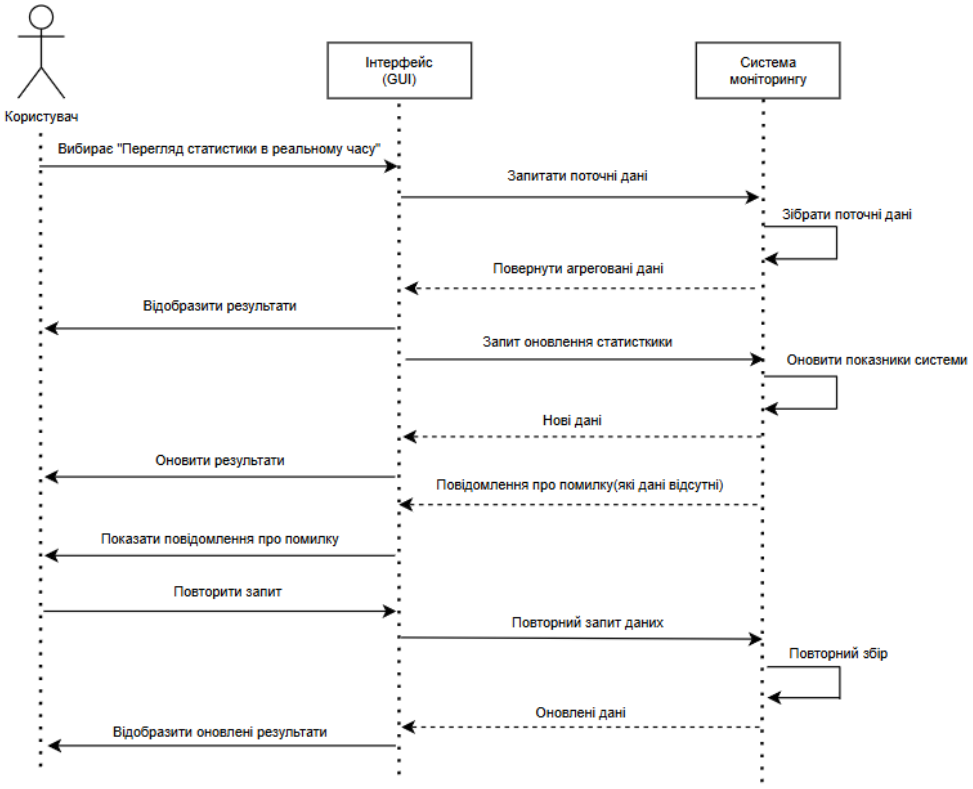

Рис 3. Діаграма послідовностей. Перегляд статистики в реальному часі

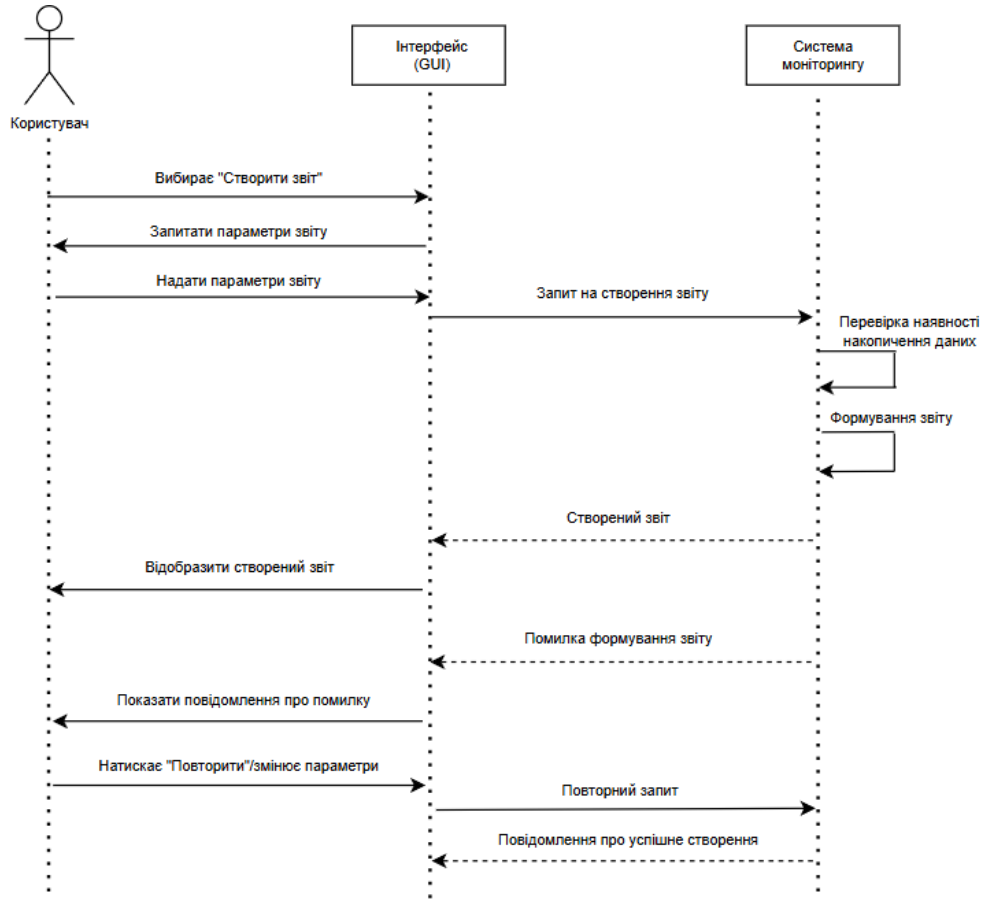

Рис 4. Діаграма послідовностей. Створити звіт про використання

4. Доданий вихідний код системи

MonitoringController.java:

```
@FXML
private void updateStats() {
    if (Session.getCurrentUser() == null) {
        cpuLabel.setText("Спочатку увійдіть у систему!");
        return;
    }
    activeUser = Session.getCurrentUser();
    if (!monitoringActive) {
        monitoringActive = true;
        startAutoMonitoring();
    } else {
        cpuLabel.setText("Моніторинг уже запущено...");
    }
}
private void startAutoMonitoring() {
    scheduler = Executors.newSingleThreadScheduledExecutor();
    scheduler.scheduleAtFixedRate(() -> Platform.runLater(() -> {
        try {
            monitoringService.recordSystemStats(activeUser);
```

```
            BigDecimal cpu = systemCollector.getCpuLoad();
            BigDecimal ram = systemCollector.getMemoryUsageMb();
            cpuLabel.setText("CPU: " + cpu + "%");
            ramLabel.setText("RAM: " + ram + " MB");
            osLabel.setText("OS: " + systemCollector.getOsName());
            windowLabel.setText("Window: " + ActiveWindowTracker.getActiveWindowTitle());
        } catch (Exception e) {
            cpuLabel.setText("Помилка моніторингу");
            e.printStackTrace();
        }
    }), 0, 5, TimeUnit.SECONDS);
    cpuLabel.setText("Моніторинг запущено (оновлення кожні 5 сек)");
}
```

MonitoringServiceImpl.java:

```
@Override
public void recordSystemStats(User user) {
    double cpu = systemCollector.getCpuLoad().doubleValue();
    double ram = systemCollector.getMemoryUsageMb().doubleValue();
    String window = ActiveWindowTracker.getActiveWindowTitle();
    SystemStats stats = new SystemStats(
            user,
            BigDecimal.valueOf(cpu),
            BigDecimal.valueOf(ram),
            window,
            0, 0);
    statsRepository.save(stats);
}
```

StatsRepositoryImpl.java:

```
@Override
public void save(SystemStats stats) {
    String sql = "INSERT INTO system_stats (user_id, cpu_load, memory_usage_mb, active_window, recorded_at)
VALUES (?, ?, ?, ?, ?)";
    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setInt(1, stats.getUser().getId());
        stmt.setBigDecimal(2, stats.getCpuLoad());
        stmt.setBigDecimal(3, stats.getMemoryUsageMb());
        stmt.setString(4, stats.getActiveWindow());
        stmt.setTimestamp(5, Timestamp.valueOf(LocalDateTime.now()));
        stmt.executeUpdate();
```

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

ReportsController.java

```
@FXML
private void createReport() {
    if (Session.getCurrentUser() == null) {
        messageLabel.setText("Спочатку увійдіть у систему!");
        return;
    }
    try {
        User user = Session.getCurrentUser();
        String reportName = reportNameField.getText();
        LocalDate start = startDatePicker.getValue();
        LocalDate end = endDatePicker.getValue();
        Report report = reportService.generateReport(user, reportName, start, end);
        messageLabel.setText("Звіт створено: " + report.getReportName());
        loadReportsTable();
    } catch (Exception e) {
        messageLabel.setText("Помилка створення звіту: " + e.getMessage());
    }
}
```

ReportServiceImpl.java:

```
@Override
public Report generateReport(User user, String reportName, LocalDate startDate, LocalDate endDate) {
    List<SystemStats> stats = statsRepository.findByRecordedAtBetween(
            startDate.atStartOfDay(),
            endDate.atTime(23, 59, 59)
    );
    List<IdleTime> idleTimes = idleRepository.findByStartTimeBetween(
            startDate.atStartOfDay(),
            endDate.atTime(23, 59, 59)
    );
    double avgCpu = stats.stream().mapToDouble(s -> s.getCpuLoad().doubleValue()).average().orElse(0);
    double avgRam = stats.stream().mapToDouble(s -> s.getMemoryUsageMb().doubleValue()).average().orElse(0);
    double totalIdleSeconds = idleTimes.stream().mapToDouble(IdleTime::getDurationSeconds).sum();
    Report report = new Report();
    report.setUser(user);
    report.setReportName(reportName);
    report.setPeriodStart(startDate);
```

```
    report.setPeriodEnd(endDate);
    report.setCpuAvg(BigDecimal.valueOf(avgCpu));
    report.setRamAvg(BigDecimal.valueOf(avgRam));
    report.setIdleTimeTotalSeconds(BigDecimal.valueOf(totalIdleSeconds));
    report.setBrowserUsagePercent(BigDecimal.ZERO);
    report.setCreatedAt(LocalDateTime.now());
    return reportRepository.save(report);
}
```

ReportRepositoryImpl.java:

```
@Override
public Report save(Report report) {
    String sql = "INSERT INTO reports (user_id, report_name, period_start, period_end, cpu_avg, ram_avg,
idle_time_total_seconds, browser_usage_percent, created_at) " +
                 "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";
    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
        stmt.setInt(1, report.getUser().getId());
        stmt.setString(2, report.getReportName());
        stmt.setDate(3, Date.valueOf(report.getPeriodStart()));
        stmt.setDate(4, Date.valueOf(report.getPeriodEnd()));
        stmt.setBigDecimal(5, report.getCpuAvg());
        stmt.setBigDecimal(6, report.getRamAvg());
        stmt.setBigDecimal(7, report.getIdleTimeTotalSeconds());
        stmt.setBigDecimal(8, report.getBrowserUsagePercent());
        stmt.setTimestamp(9, Timestamp.valueOf(report.getCreatedAt()));
        stmt.executeUpdate();
        try (ResultSet rs = stmt.getGeneratedKeys()) {
            if (rs.next()) {
                report.setId(rs.getInt(1));
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return report;
}
```

Висновок: Під час виконання лабораторної роботи було розроблено діаграму компонентів, діаграму розгортання і дві діаграми послідовностей для проєктованої системи system activity monitor. На основі спроєктованих діаграм розгортання та компонентів було доопрацьовано програмну частину системи.

Відповідь на контрольні питання:

1. Що собою становить діаграма розгортання?

Діаграма розгортання (Deployment Diagram) - це діаграма UML, яка показує фізичне розміщення програмних компонентів на апаратних вузлах (сервери, пристрої, клієнти тощо). Вона відображає архітектуру системи на рівні розгортання.

2. Які бувають види вузлів на діаграмі розгортання?

Апаратні вузли (hardware nodes) - фізичні пристрої: сервер, комп'ютер, телефон, маршрутизатор.

Програмні вузли (execution environment nodes) - середовища виконання, наприклад, JVM, контейнер Docker, ОС.

Віртуальні вузли — розміщені всередині інших вузлів (наприклад, віртуальна машина на фізичному сервері).

3. Які бувають зв'язки на діаграмі розгортання?

З'єднання (communication paths) - показують канали зв'язку між вузлами.

Відношення розміщення (deployment relationships) - показують, який компонент або артефакт розгортається на якому вузлі.

4. Які елементи присутні на діаграмі компонентів?

Компоненти (components) - логічно відокремлені частини системи (модулі, бібліотеки, підсистеми).

Інтерфейси (interfaces) - точки взаємодії між компонентами.

Артефакти (artifacts) - файли, програми, бібліотеки.

Зв'язки (dependencies) - залежності між компонентами.

Пакети (packages) - для групування компонентів.

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

Зв'язки (dependencies) показують, що один компонент залежить від іншого тобто використовує його інтерфейси, викликає його сервіси або потребує його для своєї роботи.

6. Які бувають види діаграм взаємодії?

Діаграми взаємодії (Interaction Diagrams) бувають двох основних видів: діаграма послідовностей (Sequence Diagram) і діаграма комунікації (Communication Diagram)

7. Для чого призначена діаграма послідовностей?

Вона показує послідовність обміну повідомленнями між об'єктами у певному сценарії (наприклад, виконання конкретного варіанта використання).

Мета - показати часовий порядок викликів методів і взаємодій.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

Актор (Actor) - зовнішній користувач або система.

Об’єкти (Objects) - учасники взаємодії.

Життєві лінії (Lifelines) - вертикальні лінії, що показують існування об’єкта в часі.

Повідомлення (Messages) - виклики методів або обмін даними.

Активізації (Activations) - періоди, коли об’єкт виконує дію.

Фрагменти (Combined Fragments) - умовні або циклічні частини (alt, loop, opt тощо).

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

Діаграми послідовностей деталізують сценарії варіантів використання. Тобто кожен варіант використання (use case) може бути представлений однією або кількома діаграмами послідовностей, які показують як саме система виконує цей сценарій крок за кроком.

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

Діаграма послідовностей показує динамічну поведінку об’єктів, створених на основі класів, а діаграма класів — статичну структуру цих класів. Повідомлення на діаграмі послідовностей відповідають викликам методів, визначених у класах на діаграмі класів.