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

з дисципліни "Технології розроблення програмного забезпечення"

Виконав

студент групи ІА–33:

Китченко Д.В.

Перевірив:

Мягкий М.Ю.

Київ 2025

Зміст

Вступ

Мета роботи

Метою даної лабораторної роботи є набуття практичних навичок проектування діаграм розгортання та компонентів для системи, що розробляється, а також розробка діаграм взаємодії (діаграм послідовностей) на основі визначених сценаріїв використання. Додатково, необхідно доопрацювати програмну частину системи з реалізацією повного циклу роботи з даними від введення користувачем до збереження в базі даних та відображення результатів в графічному інтерфейсі.

Завдання роботи

Для досягнення поставленої мети необхідно виконати наступні завдання:

1. Ознайомитися з теоретичними відомостями щодо діаграм розгортання, компонентів та послідовностей в нотації UML.
2. Проаналізувати діаграми, створені в попередніх лабораторних роботах, та розробити діаграму розгортання для системи "Архіватор".
3. Розробити діаграму компонентів, що відображає модульну структуру системи та залежності між компонентами.
4. Створити мінімум дві діаграми послідовностей для основних сценаріїв використання системи.
5. На основі спроектованих діаграм доопрацювати програмну частину з реалізацією:
    - мінімум двох візуальних форм (GUI)
    - повного циклу роботи з даними (введення → обробка → збереження в БД → виборка → відображення)
6. Підготувати звіт з описом всіх створених діаграм та вихідним кодом доданого функціоналу.

Теоретичні відомості

Діаграма розгортання (Deployment Diagram)

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, веб-сервер).

Між вузлами можуть стояти зв'язки, які зазвичай зображують у вигляді прямої лінії. У зв'язків можуть бути атрибути множинності (для показання, наприклад, підключення двох і більше клієнтів до одного сервера) і назва. У назві, як правило, міститься спосіб зв'язку між двома вузлами – це може бути назва протоколу (HTTP, JDBC, TCP/IP) або технологія, що використовується для забезпечення взаємодії вузлів.

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

- виконувані файли (файли .exe, .jar, .dll)
- конфігураційні файли
- файли даних
- бази даних
- HTML-документи

Діаграми розгортання розрізняють двох видів: описові та екземплярні. На діаграмах описової форми вказуються вузли, артефакти і зв'язки між вузлами

без вказівки конкретного обладнання. Діаграми екземплярної форми несуть у собі екземпляри обладнання з конкретними характеристиками.

Діаграма компонентів

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

- Логічні – система уявляється як набір самостійних, автономних модулів, що взаємодіють між собою.
- Фізичні – показують компоненти та залежності між ними. Залежності показують, що класи з одного компонента використовують класи з іншого компонента.
- Виконувані – кожен компонент являє собою файл (виконувані файли, файли вихідних кодів, сторінки HTML, бази даних).

Діаграма компонентів розробляється для таких цілей:

- візуалізації загальної структури вихідного коду програмної системи
- специфікації виконуваного варіанта програмної системи
- забезпечення багаторазового використання окремих фрагментів програмного коду
- представлення концептуальної та фізичної схем баз даних

Діаграми послідовностей

Діаграма послідовностей (Sequence Diagram) – це один із типів діаграм у моделюванні UML, 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) – користувачі чи інші системи, які взаємодіють із системою.
- Об'єкти або класи – розміщуються горизонтально на діаграмі. Кожен об'єкт має життєвий цикл, який представлений вертикальною пунктирною лінією (лінія життя).

- Повідомлення – лінії зі стрілками, які з'єднують об'єкти. Стрілка може бути синхронною (звичайна стрілка) або асинхронною (лінія з відкритим трикутником).
- Активності – вказують періоди, протягом яких об'єкт виконує певну дію. Позначається прямокутником, накладеним на лінію життя.
- Контрольні структури – блоки "alt" (альтернатива), "loop" (цикл), "opt" (опційно) для відображення умов та циклів.

Основні кроки створення діаграми послідовностей:

1. Визначити акторів і об'єкти, які беруть участь у сценарії
2. Побудувати їхні лінії життя
3. Розробити послідовність передачі повідомлень між об'єктами
4. Додати умовні блоки або цикли за необхідності

Хід роботи

Діаграма розгортання системи "Архіватор"

На основі аналізу вимог до системи "Архіватор" була спроектована діаграма розгортання за клієнт-серверною моделлю (рис. 3.1).

Опис архітектури:

Система складається з двох основних фізичних вузлів, що взаємодіють через мережу TCP/IP.

Клієнтський вузол (<<device>> Клієнтський ПК):

Представлений комп'ютером користувача з операційною системою Windows, Linux або macOS (<<execution environment>> ОС: Windows / Linux / macOS). На клієнтській стороні функціонує середовище виконання JVM (<<execution environment>> JVM), в якому запускається основний виконуваний артефакт Archiver.jar.

Додатково на клієнті присутні допоміжні артефакти:

- postgresql-driver.jar – JDBC драйвер для з'єднання з базою даних PostgreSQL
- config.properties – конфігураційний файл з параметрами підключення до БД

Клієнт має доступ до локальної файлової системи (позначено як "Локальні архіви (*.zip)") для створення, читання та модифікації архівних файлів. Взаємодія з файловою системою здійснюється через механізми File I/O операційної системи.

Серверний вузол (<<device>> Сервер баз даних):

Представлений виділеним сервером з операційною системою Linux (<<execution environment>> ОС: Linux Server). На серверній стороні розгорнуто середовище виконання PostgreSQL Server (<<execution environment>> PostgreSQL Server), яке управляє базою даних ArchiverDB.

База даних містить чотири основні таблиці:

- archive_info – інформація про архіви
- operation_log – журнал операцій

- operationdetail – деталі операцій
- archive_bookmark – закладки в архівах

Мережева взаємодія:

З'єднання між клієнтським та серверним вузлами здійснюється через мережу TCP/IP за протоколом JDBC (порт 5432). Клієнт ініціює з'єднання з базою даних для виконання SQL-запитів (SELECT, INSERT, UPDATE, DELETE). Зв'язок є односпрямованим від клієнта до сервера, що відповідає архітектурі клієнт-серверних додатків.

Переваги такої архітектури:

- Централізоване зберігання даних на сервері
- Можливість роботи декількох клієнтів одночасно
- Незалежність клієнтської та серверної частин
- Легке масштабування системи

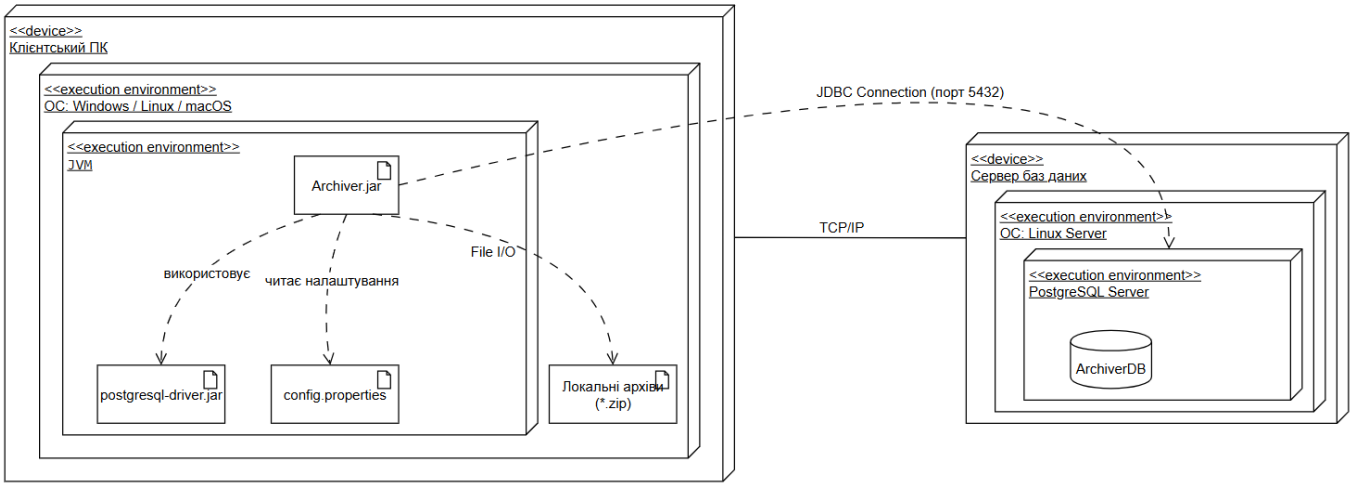


Рисунок 3.1 – Діаграма розгортання системи "Архіватор"

Діаграма компонентів системи

Діаграма компонентів відображає логічну структуру системи "Архіватор" та залежності між модулями (рис. 3.2). Система розділена на три основні частини згідно з багатошаровою архітектурою.

Client Component (Клієнтські компоненти):

Містить три основні модулі:

1. UI Module – модуль графічного інтерфейсу користувача. Реалізований з використанням бібліотеки Swing та включає:
    - SimpleMainWindow – головне вікно зі списком архівів
    - OperationHistoryWindow – вікно перегляду історії операцій
    - Діалогові вікна для введення даних
2. Archiver Core Module – модуль основної бізнес-логіки системи. Включає:
    - ArchiverFacade – фасад для спрощення взаємодії з підсистемами
    - Strategy паттерн (IArchiverStrategy, ZipStrategy)
    - Factory Method (ArchiverStrategyFactory)
    - Координацію роботи між UI та Repository шарами
3. Domain Entities – модуль доменних моделей. Містить класи предметної області:
    - Archive, ArchiveInfo, ArchiveItem
    - ArchivedFile, ArchivedFolder
    - OperationLog, OperationDetail, ArchiveBookmark
    - Енумерації: ArchiveFormat, OperationType, OperationStatus

Data Contract (Контракт даних):

Визначає загальні інтерфейси та схему даних для взаємодії між шарами:

1. DTO Classes – класи передачі даних між шарами. Використовуються для серіалізації та передачі інформації.
2. Database Schema – опис структури таблиць бази даних. Визначає структуру даних для збереження в PostgreSQL.

Server Component (Серверні компоненти):

Містить два модулі:

1. Repository Module – модуль доступу до даних. Включає:
    - Інтерфейси репозиторіїв (IArchiveInfoRepository, IOperationLogRepository, IOperationDetailRepository, IArchiveBookmarkRepository)
    - PostgreSQL реалізації (PostgresArchiveInfoRepository,

PostgresOperationLogRepository, PostgresOperationDetailRepository, PostgresArchiveBookmarkRepository)

2. Database Service – сервіс управління з'єднанням з БД:
    - DatabaseConnector – клас для створення та управління з'єднаннями
    - Завантаження конфігурації з config.properties
    - Пул з'єднань

Залежності між компонентами:

- UI Module залежить від Archiver Core Module (пунктирна стрілка)
- Archiver Core Module використовує Domain Entities
- Archiver Core Module залежить від DTO Classes для передачі даних
- Repository Module залежить від DTO Classes
- Database Service використовує Database Schema для роботи з таблицями
- Repository Module залежить від Database Service для отримання з'єднань

Така архітектура забезпечує:

- Слабку зв'язаність між компонентами
- Можливість незалежного тестування модулів
- Легку заміну реалізацій (наприклад, зміна БД з PostgreSQL на MySQL)
- Дотримання принципу Single Responsibility

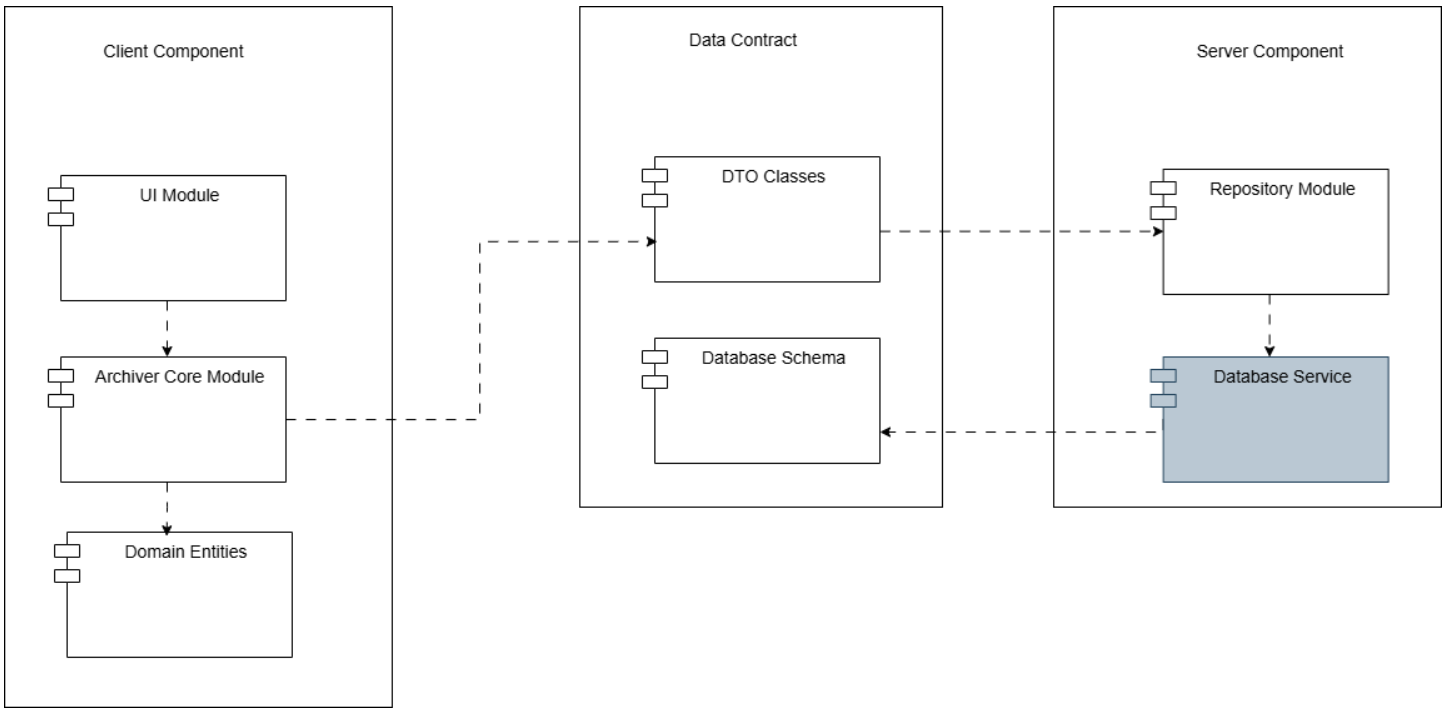


Рисунок 3.2 – Діаграма компонентів системи "Архіватор"

Діаграма послідовності "Створення архіву"

Діаграма описує процес створення нового архіву користувачем (рис. 3.3).

Учасники взаємодії:

- User – користувач системи
- UI – графічний інтерфейс (SimpleMainWindow)
- BusinessLogic – бізнес-логіка (ArchiverFacade + ZipStrategy)
- DataBase – база даних PostgreSQL

Опис основного сценарію:

1. Користувач натискає кнопку "Створити" в головному вікні.
2. UI відкриває діалогове вікно для вибору файлів та папок.
3. Користувач обирає елементи для архівування та натискає "Далі".
4. UI показує діалог збереження, де користувач вказує ім'я архіву, формат (ZIP/TAR.GZ/RAR/ACE), місце збереження та рівень стиснення.
5. Користувач натискає "Зберегти".
6. UI передає команду створення архіву до BusinessLogic з усіма параметрами.
7. BusinessLogic виконує валідацію (перевірка шляху, прав доступу, вільного місця на диску).
8. При успішній валідації BusinessLogic створює архів та виконує транзакцію в базі даних:
    - Відкриває транзакцію (BEGIN)
    - Додає запис про архів в таблицю archive_info
    - Додає запис про операцію в таблицю operation_log
    - Фіксує транзакцію (COMMIT)
9. BusinessLogic повертає UI повідомлення про успішне завершення.
10. UI показує користувачу повідомлення "Архів успішно створено".

Альтернативні сценарії:

- Скасування операції: Користувач може натиснути "Скасувати" на етапах вибору файлів або збереження. Операція перерветься без змін у файловій системі.

- Помилка валідації: Якщо система виявляє проблеми (недостатньо місця, немає прав доступу, файл існує, формат не підтримується), показується відповідне повідомлення про помилку та операція переривається.
- Помилка транзакції: При виникненні помилки під час роботи з базою даних автоматично виконується ROLLBACK для збереження цілісності даних.

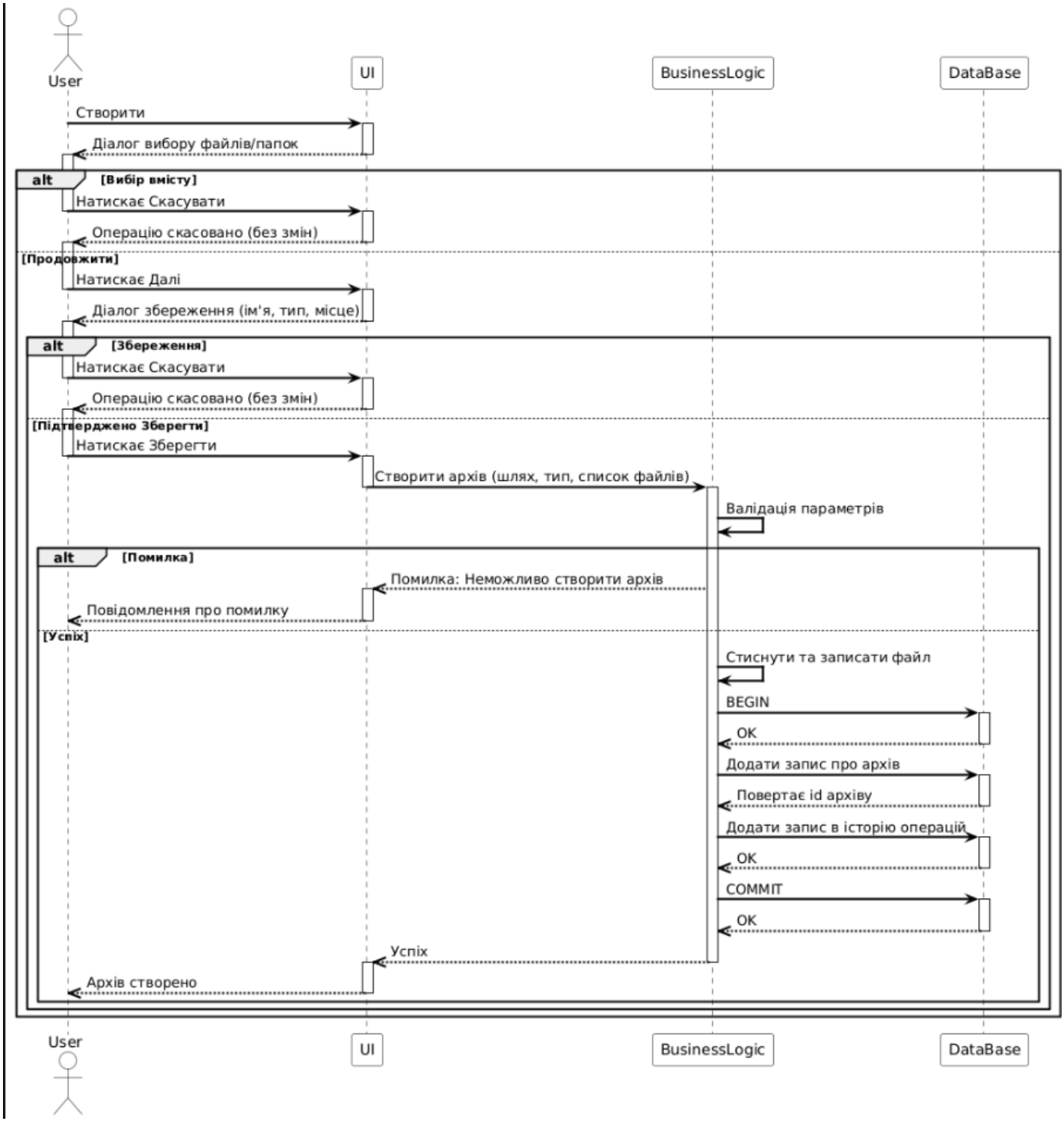


Рисунок 3.3 – Діаграма послідовності "Створення архіву"

Діаграма послідовності "Перевірка контрольної суми"

Діаграма описує процес перевірки цілісності файлів в архіві через обчислення та порівняння контрольних сум (рис. 3.4).

Учасники взаємодії:

- User – користувач системи
- UI – графічний інтерфейс
- BusinessLogic – бізнес-логіка
- DataBase – база даних

Опис основного сценарію:

1. Користувач виділяє один або кілька файлів у вмісті відкритого архіву.
2. Натискає кнопку "Перевірити контрольну суму".
3. UI передає запит до BusinessLogic та показує індикатор виконання операції.
4. BusinessLogic циклічно обробляє кожен вибраний файл:
    - Зчитує дані файлу з архіву через DataBase
    - Обчислює контрольну суму (MD5/SHA-256)
    - Запитує еталонне значення checksum з БД
    - Порівнює обчислену та еталонну суму (якщо еталон є)
    - Повертає результат для файлу: OK, Невідповідність, або тільки обчислену суму
5. Після обробки всіх файлів BusinessLogic повертає загальний результат.
6. UI показує підсумковий список файлів зі статусами та закриває індикатор.

Альтернативні сценарії:

- Скасування операції: Користувач може натиснути "Скасувати" під час перевірки. Операція переривається, результати не зберігаються.
- Помилка читання файлу: Якщо файл пошкоджений або немає прав доступу, він пропускається з відповідним повідомленням. Обробка інших файлів продовжується.

- Еталон відсутній: Якщо в БД немає еталонного значення, система показує тільки обчислену контрольну суму без порівняння.
- Еталон невірний: Якщо еталонне значення некоректне, система пропонує вказати інший еталон або продовжити без порівняння.

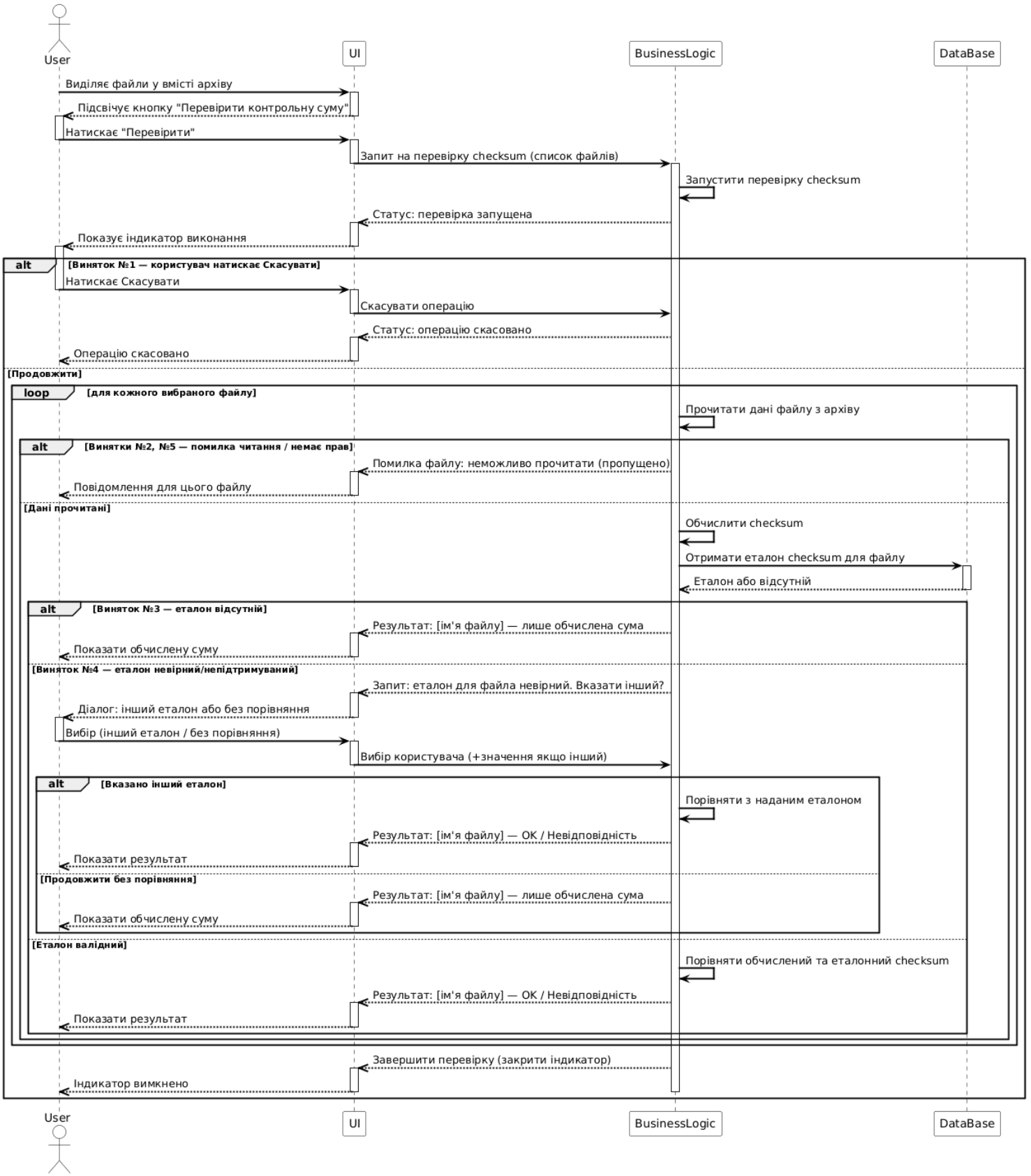


Рисунок 3.4 – Діаграма послідовності "Перевірка контрольної суми"

Реалізація програмної частини

На основі спроектованих діаграм розгортання та компонентів було доопрацьовано програмну частину системи "Архіватор". Реалізація включає повний цикл роботи з даними від введення на формі до збереження в базі даних та відображення результатів.

Система реалізована за багатошаровою архітектурою з чітким розподілом відповідальності:

Presentation Layer (Шар представлення):

- SimpleMainWindow.java – головне вікно програми з таблицею архівів
- OperationHistoryWindow.java – вікно перегляду історії операцій

Business Logic Layer (Шар бізнес-логіки):

- ArchiverFacade.java – фасад для координації всіх операцій системи
- IArchiverStrategy.java – інтерфейс стратегії архівування
- ZipStrategy.java – реалізація стратегії для ZIP формату
- ArchiverStrategyFactory.java – фабрика для створення стратегій

Data Access Layer (Шар доступу до даних):

- Інтерфейси репозиторіїв: IArchiveInfoRepository, IOperationLogRepository, IOperationDetailRepository, IArchiveBookmarkRepository
- PostgreSQL реалізації: PostgresArchiveInfoRepository, PostgresOperationLogRepository, PostgresOperationDetailRepository, PostgresArchiveBookmarkRepository
- DatabaseConnector.java – управління з'єднаннями з БД

Domain Layer (Доменний шар):

- Model класи: Archive, ArchiveInfo, OperationLog, OperationDetail, ArchiveBookmark
- Енумерації: ArchiveFormat, OperationType, OperationStatus

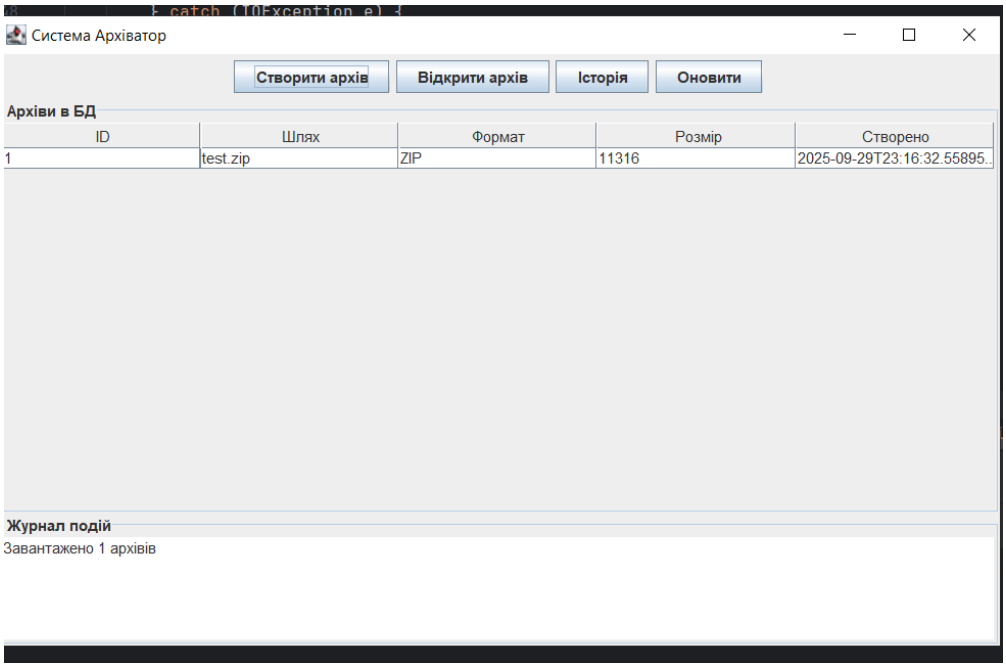
Форма №1: SimpleMainWindow (Головне вікно)

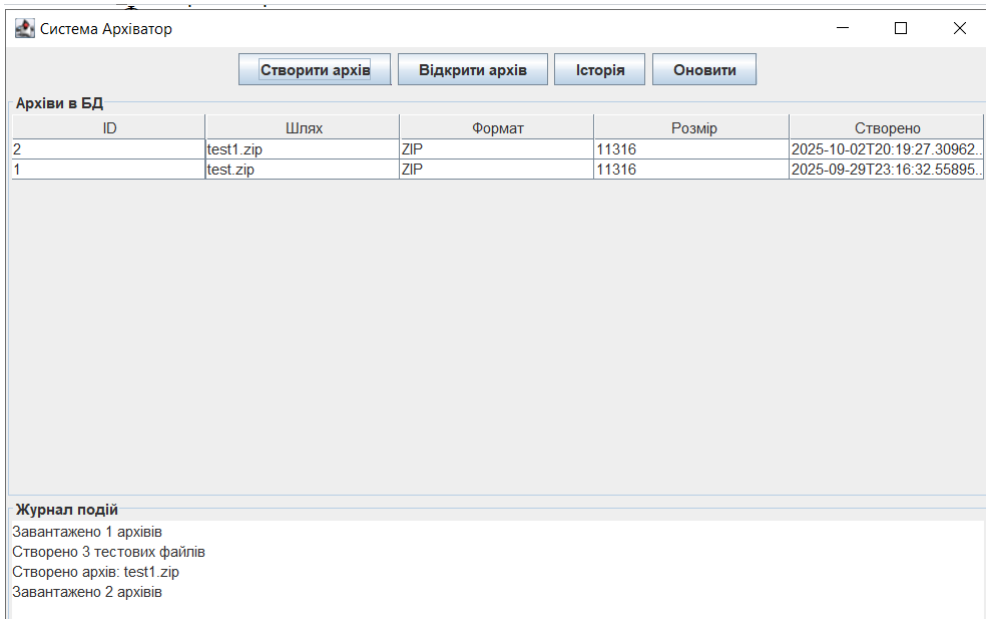
Головне вікно системи містить наступні елементи:

- Панель кнопок управління: "Створити архів", "Відкрити архів", "Історія", "Оновити"
- Таблицю JTable з колонками: ID, Шлях, Формат, Розмір, Створено
- Текстову область JTextArea для відображення журналу подій

Функціональність:

- Відображення списку архівів з бази даних
- Створення нових архівів через діалогові вікна
- Відкриття існуючих архівів
- Перехід до вікна історії операцій
- Оновлення даних з БД





	id [PK] integer	file_path character varying (500)	format character varying (50)	total_size bigint	created_at timestamp without time zone	last_accessed_at timestamp without time zone
1	1	test.zip	ZIP	11316	2025-09-29 23:16:32.55895	2025-09-29 23:41:03.782564
2	2	test1.zip	ZIP	11316	2025-10-02 20:19:27.309625	2025-10-02 20:20:06.968761

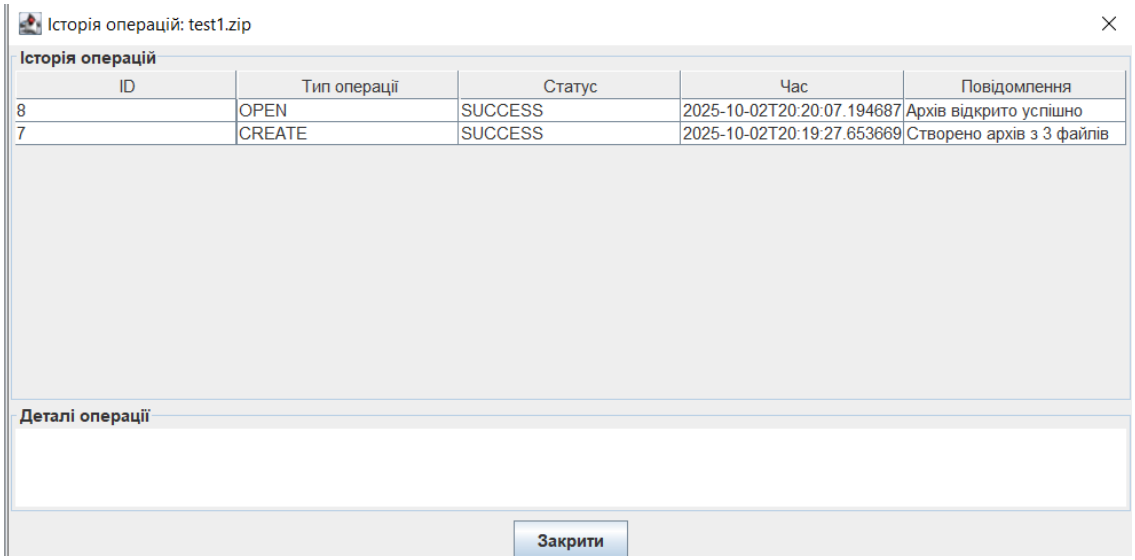
Форма №2: OperationHistoryWindow (Вікно історії операцій)

Вікно історії операцій містить:

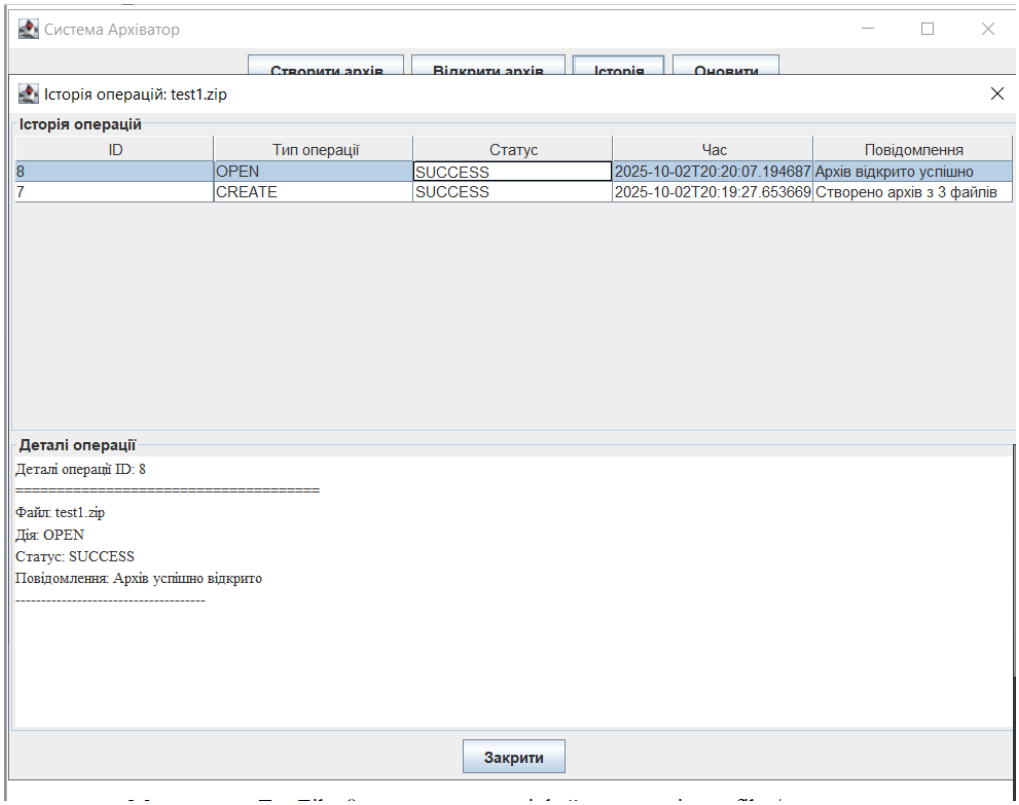
- Таблицю JTable з історією: ID, Тип операції, Статус, Час, Повідомлення
- Текстову область JTextArea для відображення деталей операції
- Кнопку "Закрити"

Функціональність:

- Завантаження історії операцій для вибраного архіву
- Відображення деталей операції при виборі запису
- Перегляд інформації про кожен оброблений файл



	id [PK] integer	archive_id bigint	operation_type character varying (50)	status character varying (50)	timestamp timestamp without time zone	details text
1	1	1	CREATE	SUCCESS	2025-09-29 23:16:32.693951	Створено архів з 3 файлів
2	2	1	OPEN	SUCCESS	2025-09-29 23:16:36.03237	Архів відкрито успішно
3	3	1	OPEN	SUCCESS	2025-09-29 23:16:37.938679	Архів відкрито успішно
4	4	1	CREATE	SUCCESS	2025-09-29 23:25:32.550249	Створено архів з 3 файлів
5	5	1	OPEN	SUCCESS	2025-09-29 23:25:38.053861	Архів відкрито успішно
6	6	1	OPEN	SUCCESS	2025-09-29 23:41:04.023565	Архів відкрито успішно
7	7	2	CREATE	SUCCESS	2025-10-02 20:19:27.653669	Створено архів з 3 файлів
8	8	2	OPEN	SUCCESS	2025-10-02 20:20:07.194687	Архів відкрито успішно



	detailid [PK] integer	logid bigint	itempath character varying (500)	action character varying (50)	status character varying (50)	message text
1	1	1	file1.txt	CREATE	SUCCESS	Файл додано до архіву
2	2	1	file2.txt	CREATE	SUCCESS	Файл додано до архіву
3	3	1	file3.txt	CREATE	SUCCESS	Файл додано до архіву
4	4	2	test.zip	OPEN	SUCCESS	Архів успішно відкрито
5	5	3	test.zip	OPEN	SUCCESS	Архів успішно відкрито
6	6	4	D:\IdeaProjects\untitled11\test_files\test_file1.t...	CREATE	SUCCESS	Файл додано до архіву
7	7	4	D:\IdeaProjects\untitled11\test_files\test_file2.t...	CREATE	SUCCESS	Файл додано до архіву
8	8	4	D:\IdeaProjects\untitled11\test_files\test_file3.t...	CREATE	SUCCESS	Файл додано до архіву
9	9	5	test.zip	OPEN	SUCCESS	Архів успішно відкрито
10	10	6	test.zip	OPEN	SUCCESS	Архів успішно відкрито
11	11	7	D:\IdeaProjects\untitled11\test_files\test_file1.t...	CREATE	SUCCESS	Файл додано до архіву
12	12	7	D:\IdeaProjects\untitled11\test_files\test_file2.t...	CREATE	SUCCESS	Файл додано до архіву
13	13	7	D:\IdeaProjects\untitled11\test_files\test_file3.t...	CREATE	SUCCESS	Файл додано до архіву
14	14	8	test1.zip	OPEN	SUCCESS	Архів успішно відкрито

Система забезпечує наступний повний цикл роботи з даними:

1. Введення даних користувачем:

String archiveName = JOptionPane.showInputDialog("Введіть назву архіву:");

Користувач вводить назву архіву через стандартний діалог Swing.

2. Створення тестових файлів:

```
private List<String> createTestFiles() {
    File testDir = new File("test_files");
    testDir.mkdirs();
```

```
    // Створення 3 реальних файлів з текстовим вмістом
}
```

Метод createTestFiles() створює реальні файли в папці test_files/ для подальшого архівування.

3. Архівування файлів:

```
Archive archive = facade.create(archiveName, ArchiveFormat.ZIP, files);
```

ZipStrategy створює реальний ZIP архів, стискаючи файли за допомогою ZipOutputStream.

4. Збереження в базі даних:

```
// Збереження інформації про архів
archiveRepo.upsert(newInfo);
// Збереження операції
long logId = logRepo.add(logEntry);
// Збереження деталей для кожного файлу
for (String filePath : files) {
    OperationDetail detail = new OperationDetail(...);
    detailRepo.add(detail);
}
```

Дані зберігаються в трьох таблицях PostgreSQL: archive_info, operation_log, operationdetail.

5. Виборка даних з БД:

```
List<ArchiveInfo> archives = facade.listRecentArchives(20);
```

Метод завантажує список останніх 20 архівів з бази даних.

6. Відображення на UI:

```
java
for (ArchiveInfo info : archives) {
    Object[] row = {info.getId(), info.getFilePath(),
                    info.getFormat(), info.getTotalSize(),
                    info.getCreatedAt()};
    tableModel.addRow(row);
```

}

Дані відображаються в JTable головного вікна.\

Вихідний код системи

Нижче наведено ключові фрагменти вихідного коду, доданого в цій лабораторній роботі.

ArchiverFacade

```
public class ArchiverFacade {
   private final IArchiveInfoRepository archiveRepo;
   private final IOperationLogRepository logRepo;

   private Archive currentOpenArchive;
   private IArchiverStrategy currentStrategy;

   private final IArchiveBookmarkRepository bookmarkRepo;
   private final IOperationDetailRepository detailRepo;
   public ArchiverFacade(IArchiveInfoRepository archiveRepo, IOperationLogRepository logRepo,
                         IArchiveBookmarkRepository bookmarkRepo, IOperationDetailRepository
detailRepo) {
      this.archiveRepo = archiveRepo;
      this.logRepo = logRepo;
      this.bookmarkRepo = bookmarkRepo;
      this.detailRepo = detailRepo;
   }

   public Archive create(String path, ArchiveFormat format, List<String> files) {
      OperationLog logEntry = null;
      try {
         currentStrategy = ArchiverStrategyFactory.createStrategy(format);
         currentOpenArchive = currentStrategy.create(path, files);
         ArchiveInfo newInfo = new ArchiveInfo(
               0L, path, format.name(),
               currentOpenArchive.getTotalSize(),
               LocalDateTime.now(), LocalDateTime.now());
         archiveRepo.upsert(newInfo);
         long archiveId = archiveRepo.findByPath(path)
               .map(ArchiveInfo::getId).orElse(0L);

         logEntry = new OperationLog(
               archiveId, OperationType.CREATE,
               OperationStatus.SUCCESS, "Створено архів з " + files.size() + " файлів");

         long logId = logRepo.add(logEntry);
         for (String filePath : files) {
            OperationDetail detail = new OperationDetail(
                  0,
                  logId,
                  filePath,
                  OperationType.CREATE,
```

```
                OperationStatus.SUCCESS,
                "Файл додано до архіву"
            );
            detailRepo.add(detail);
        }
        return currentOpenArchive;
    }catch (Exception e) {
            long logId = logRepo.add(new OperationLog(
                0L, OperationType.CREATE,
                OperationStatus.FAILURE, "Помилка створення архіву"
            ));
            OperationDetail detail = new OperationDetail(
                0, logId,
                path,
                OperationType.CREATE,
                OperationStatus.FAILURE,
                "Помилка: " + e.getMessage()
            );
            detailRepo.add(detail);
        throw new RuntimeException("Помилка створення архіву", e);
    }
}

public Archive open(String path) {
    OperationLog logEntry = null;
    try {
        currentStrategy = ArchiverStrategyFactory.createForPath(path);
        currentOpenArchive = currentStrategy.open(path);
        ArchiveInfo info = archiveRepo.findByPath(path)
                .orElse(new ArchiveInfo(
                    0L, path, currentOpenArchive.getFormat().name(),
                    currentOpenArchive.getTotalSize(),
                    LocalDateTime.now(), LocalDateTime.now()
                ));
        info.setLastAccessedAt(LocalDateTime.now());
        archiveRepo.upsert(info);
        long archiveId = archiveRepo.findByPath(path)
                .map(ArchiveInfo::getId).orElse(0L);
        logEntry = new OperationLog(
                archiveId, OperationType.OPEN,
                OperationStatus.SUCCESS, "Архів відкрито успішно"
        );
        long logId = logRepo.add(logEntry);

        OperationDetail detail = new OperationDetail(
                0,
                logId,
                path,
                OperationType.OPEN,
                OperationStatus.SUCCESS,
                "Архів успішно відкрито"
        );
        detailRepo.add(detail);
```

```
            return currentOpenArchive;
        } catch (Exception e) {
            long logId = logRepo.add(new OperationLog(
                    0L, OperationType.OPEN,
                    OperationStatus.FAILURE, "Помилка відкриття архіву"
            ));

            OperationDetail detail = new OperationDetail(
                    0, logId, path,
                    OperationType.OPEN,
                    OperationStatus.FAILURE,
                    "Помилка: " + e.getMessage()
            );
            detailRepo.add(detail);
            throw new RuntimeException("Помилка відкриття архіву", e);
        }
    }
```

SimpleMainWindow

```
package ui;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import facade.ArchiverFacade;
import model.*;
import repository.*;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class SimpleMainWindow extends JFrame {
    private ArchiverFacade facade;
    private JTable table;
    private DefaultTableModel tableModel;
    private JTextArea logArea;

    public SimpleMainWindow() {

        IArchiveInfoRepository archiveRepo = new PostgresArchiveInfoRepository();
        IOperationLogRepository logRepo = new PostgresOperationLogRepository();
        IOperationDetailRepository detailRepo = new PostgresOperationDetailRepository();
        IArchiveBookmarkRepository bookmarkRepo = new PostgresArchiveBookmarkRepository();
        this.facade = new ArchiverFacade(archiveRepo, logRepo, bookmarkRepo, detailRepo);

        init();
        loadData();
    }
```

```
private void init() {
    setTitle("Система Архіватор");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new BorderLayout());

    JPanel topPanel = new JPanel();
    JButton createBtn = new JButton("Створити архів");
    JButton refreshBtn = new JButton("Оновити");
    JButton openBtn = new JButton("Відкрити архів");
    JButton historyBtn = new JButton("Історія");
    createBtn.addActionListener(e -> createArchive());
    refreshBtn.addActionListener(e -> loadData());
    openBtn.addActionListener(e -> openArchive());
    historyBtn.addActionListener(e -> showHistory());

    topPanel.add(createBtn);
    topPanel.add(openBtn);
    topPanel.add(historyBtn);
    topPanel.add(refreshBtn);

    String[] columns = {"ID", "Шлях", "Формат", "Розмір", "Створено"};
    tableModel = new DefaultTableModel(columns, 0);
    table = new JTable(tableModel);
    JScrollPane scrollPane = new JScrollPane(table);
    scrollPane.setBorder(BorderFactory.createTitledBorder("Архіви в БД"));


    logArea = new JTextArea(5, 50);
    logArea.setEditable(false);
    JScrollPane logScroll = new JScrollPane(logArea);
    logScroll.setBorder(BorderFactory.createTitledBorder("Журнал подій"));

    add(topPanel, BorderLayout.NORTH);
    add(scrollPane, BorderLayout.CENTER);
    add(logScroll, BorderLayout.SOUTH);

    setSize(800, 500);
    setLocationRelativeTo(null);
}

private void loadData() {
    tableModel.setRowCount(0);
    try {
        List<ArchiveInfo> archives = facade.listRecentArchives(20);

        for (ArchiveInfo info : archives) {
            Object[] row = {
                    info.getId(),
                    info.getFilePath(),
                    info.getFormat(),
                    info.getTotalSize(),
                    info.getCreatedAt()
```

```
            };
            tableModel.addRow(row);
        }
        logArea.append("Завантажено " + archives.size() + " архівів\n");
    } catch (Exception e) {
        logArea.append("Помилка: " + e.getMessage() + "\n");
    }
}

private void createArchive() {

    String archiveName = JOptionPane.showInputDialog(null, "Введіть назву архіву:",
"test.zip");

    if (archiveName != null && !archiveName.isEmpty()) {
        try {
            List<String> files = createTestFiles();
            Archive archive = facade.create(archiveName, ArchiveFormat.ZIP, files);

            logArea.append("Створено архів: " + archive.getFilePath() + "\n");
            loadData();

        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Помилка: " + e.getMessage());
            logArea.append("Помилка: " + e.getMessage() + "\n");
        }
    }
}
private List<String> createTestFiles() {
    List<String> filePaths = new ArrayList<>();
    try {
        File testDir = new File("test_files");
        if (!testDir.exists()) {
            testDir.mkdirs();
        }
        File file1 = new File(testDir, "test_file1.txt");
        try (FileWriter writer = new FileWriter(file1)) {
            writer.write("Це тестовий файл номер 1\n");
            writer.write("Створено для перевірки архіватора\n");
            writer.write("Дата: " + LocalDateTime.now() + "\n");
        }
        filePaths.add(file1.getAbsolutePath());
        File file2 = new File(testDir, "test_file2.txt");
        try (FileWriter writer = new FileWriter(file2)) {
            writer.write("Другий тестовий файл\n");
            writer.write("ТУТА БІЛЬШЕ ТЕКСТУ\n");
            writer.write("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\n");
            writer.write("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB\n");
        }
        filePaths.add(file2.getAbsolutePath());

        File file3 = new File(testDir, "test_file3.txt");
        try (FileWriter writer = new FileWriter(file3)) {
```

```
            writer.write("Третій файл\n");
            for (int i = 0; i < 100; i++) {
                writer.write("Рядок номер " + i + " з довгим текстом для збільшення розміру
файлу\n");
            }
        }
        filePaths.add(file3.getAbsolutePath());

        logArea.append("Створено " + filePaths.size() + " тестових файлів\n");

    } catch (IOException e) {
        logArea.append("Помилка створення тестових файлів: " + e.getMessage() + "\n");
    }
    return filePaths;
  }
  private void showHistory() {
    int selectedRow = table.getSelectedRow();

    if (selectedRow >= 0) {
        Long archiveId = (Long) tableModel.getValueAt(selectedRow, 0);
        String archivePath = (String) tableModel.getValueAt(selectedRow, 1);

        try {
            facade.open(archivePath);
            OperationHistoryWindow historyWindow = new OperationHistoryWindow(this, facade,
archiveId, archivePath);
            historyWindow.setVisible(true);

        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Помилка: " + e.getMessage());
        }
    } else {
        JOptionPane.showMessageDialog(null, "Виберіть архів для перегляду історії");
    }
  }
  private void openArchive() {
    int selectedRow = table.getSelectedRow();
    if (selectedRow >= 0) {
        String path = (String) tableModel.getValueAt(selectedRow, 1);
        try {
            Archive archive = facade.open(path);
            logArea.append("Відкрито архів: " + archive.getFilePath() + "\n");

            StringBuilder content = new StringBuilder("Вміст архіву:\n");
            for (ArchiveItem item : archive.getRootFolder().getChildren()) {
                content.append("- ").append(item.getName()).append("\n");
            }
            JOptionPane.showMessageDialog(null, content.toString());

        } catch (Exception e) {
            logArea.append("Помилка відкриття: " + e.getMessage() + "\n");
        }
    } else {
```

```
            JOptionPane.showMessageDialog(null, "Виберіть архів для відкриття");
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            SimpleMainWindow window = new SimpleMainWindow();
            window.setVisible(true);
        });
    }
}
```

ZipStrategy

```
package strategy;

import model.*;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.file.Files;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Map;
import java.util.HashMap;
import java.util.ArrayList;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;

public class ZipStrategy implements IArchiverStrategy {

    @Override
    public Archive create(String archivePath, List<String> files) {
        System.out.println("Створення ZIP архіву: " + archivePath);
        Archive archive = new Archive(archivePath, ArchiveFormat.ZIP);
        long totalCompressedSize = 0;

        try (FileOutputStream fos = new FileOutputStream(archivePath);
             ZipOutputStream zos = new ZipOutputStream(fos)) {


            for (String filePath : files) {
                File file = new File(filePath);

                if (!file.exists()) {
                    System.err.println("Файл не знайдено: " + filePath);
                    continue;
                }

                ZipEntry entry = new ZipEntry(file.getName());
                entry.setTime(file.lastModified());
                entry.setSize(file.length());
                zos.putNextEntry(entry);

                Files.copy(file.toPath(), zos);
```

```
            zos.closeEntry();

            ArchivedFile archivedFile = new ArchivedFile(
                    file.getName(),
                    LocalDateTime.now(),
                    file.length(),
                    entry.getCompressedSize(),
                    file.getAbsolutePath()
            );

            archive.getRootFolder().addChild(archivedFile);
            totalCompressedSize += file.length();
        }

    } catch (IOException e) {
        throw new RuntimeException("Помилка створення ZIP: " + e.getMessage(), e);
    }

    archive.setTotalSize(totalCompressedSize);
    return archive;
  }
}
```

OperationHistoryWindow

```
package ui;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import facade.ArchiverFacade;
import model.*;
import java.util.List;

public class OperationHistoryWindow extends JDialog {
  private ArchiverFacade facade;
  private long archiveId;
  private JTable historyTable;
  private DefaultTableModel historyModel;
  private JTextArea detailsArea;

  public OperationHistoryWindow(JFrame parent, ArchiverFacade facade, long archiveId, String
archiveName) {
    super(parent, "Історія операцій: " + archiveName, true);


    this.facade = facade;
    this.archiveId = archiveId;

    init();
    loadHistory();
  }

  private void init() {
    setLayout(new BorderLayout());
```

```
    setSize(800, 400);


    String[] columns = {"ID", "Тип операції", "Статус", "Час", "Повідомлення"};
    historyModel = new DefaultTableModel(columns, 0);
    historyTable = new JTable(historyModel);


    historyTable.getSelectionModel().addListSelectionListener(e -> {
        if (!e.getValueIsAdjusting()) {
            showDetails();
        }
    });

    JScrollPane historyScroll = new JScrollPane(historyTable);
    historyScroll.setBorder(BorderFactory.createTitledBorder("Історія операцій"));
    historyScroll.setPreferredSize(new Dimension(0, 250));


    detailsArea = new JTextArea();
    detailsArea.setEditable(false);
    detailsArea.setFont(new Font(Font.SERIF, Font.PLAIN, 12));
    JScrollPane detailsScroll = new JScrollPane(detailsArea);
    detailsScroll.setBorder(BorderFactory.createTitledBorder("Деталі операції"));

    JButton closeBtn = new JButton("Закрити");
    closeBtn.addActionListener(e -> dispose());
    JPanel buttonPanel = new JPanel();
    buttonPanel.add(closeBtn);

    add(historyScroll, BorderLayout.PAGE_START);
    add(detailsScroll, BorderLayout.CENTER);
    add(buttonPanel, BorderLayout.PAGE_END);

    setLocationRelativeTo(getParent());
}

private void loadHistory() {
    historyModel.setRowCount(0);

    try {
        List<OperationLog> logs = facade.getOperationLogsForCurrentArchive(50);

        for (OperationLog log : logs) {
            Object[] row = {
                    log.getId(),
                    log.getOperationType().name(),
                    log.getStatus().name(),
                    log.getTimestamp(),
                    log.getMessage()
            };
            historyModel.addRow(row);
        }
```

```
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Помилка: " + e.getMessage());
        }
    }

    private void showDetails() {
        int selectedRow = historyTable.getSelectedRow();
        if (selectedRow >= 0) {
            long operationId = (Long) historyModel.getValueAt(selectedRow, 0);
            try {
                List<OperationDetail> details = facade.getDetailsForOperation(operationId);

                StringBuilder sb = new StringBuilder();
                sb.append("Деталі операції ID: ").append(operationId).append("\n");
                sb.append("==================================\n");

                for (OperationDetail detail : details) {
                    sb.append("Файл: ").append(detail.getItemPath()).append("\n");
                    sb.append("Дія: ").append(detail.getType()).append("\n");
                    sb.append("Статус: ").append(detail.getStatus()).append("\n");
                    sb.append("Повідомлення: ").append(detail.getMessage()).append("\n");
                    sb.append("----------------------------------\n");
                }

                detailsArea.setText(sb.toString());

            } catch (Exception e) {
                detailsArea.setText("Помилка завантаження деталей: " + e.getMessage());
            }
        }
    }
}
```

PostgresArchiveBookmarkRepository

```
package repository;

import model.ArchiveBookmark;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class PostgresArchiveBookmarkRepository implements IArchiveBookmarkRepository{
    @Override
    public void add(ArchiveBookmark bookmark) {
        String sql = "INSERT INTO archivebookmark(archiveid, itempath, displayname, isfolder,
note, created_at) VALUES(?, ?, ?, ?, ?, ?)";
        try (Connection conn = DatabaseConnector.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setLong(1, bookmark.getArchiveID());
```

```
            pstmt.setString(2, bookmark.getItemPath());
            pstmt.setString(3, bookmark.getDisplayName());
            pstmt.setBoolean(4, bookmark.isFolder());
            pstmt.setString(5, bookmark.getNote());
            pstmt.setObject(6, bookmark.getCreatedAt());
            pstmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void delete(long bookmarkId) {
        String sql = "DELETE FROM archivebookmark WHERE id = ?";
        try (Connection conn = DatabaseConnector.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setLong(1, bookmarkId);
            pstmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    @Override
    public List<ArchiveBookmark> listByArchive(long archiveId) {
        String sql = "SELECT * FROM archivebookmark WHERE archiveId = ?";
        List<ArchiveBookmark> result = new ArrayList<>();
        try(Connection conn = DatabaseConnector.getConnection()) {
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setLong(1,archiveId);
            ResultSet rs = pstmt.executeQuery();
            while (rs.next()){
                ArchiveBookmark archiveBookmark = new ArchiveBookmark(
                        rs.getLong("bookmarkid"),
                        rs.getLong("archiveid"),
                        rs.getString("itempath"),
                        rs.getString("displayname"),
                        rs.getBoolean("isfolder"),
                        rs.getString("note"),
                        rs.getObject("created_at", LocalDateTime.class)
                );
                result.add(archiveBookmark);
            }

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

PostgresArchiveInfoRepository

```
package repository;

import model.ArchiveInfo;
```

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

public class PostgresArchiveInfoRepository implements IArchiveInfoRepository {
   @Override
   public Optional<ArchiveInfo> findByPath(String path) {
      String sql = "SELECT id, file_path, format, total_size, created_at, last_accessed_at FROM
archive_info WHERE file_path = ?";

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

         pstmt.setString(1, path);
         ResultSet rs = pstmt.executeQuery();

         if (rs.next()) {

            ArchiveInfo info = new ArchiveInfo(
                  rs.getLong("id"),
                  rs.getString("file_path"),
                  rs.getString("format"),
                  rs.getLong("total_size"),
                  rs.getObject("created_at", LocalDateTime.class),
                  rs.getObject("last_accessed_at", LocalDateTime.class)
            );
            return Optional.of(info);
         }

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

      return Optional.empty();
   }


   public void update(ArchiveInfo info) {
      String sql = "UPDATE archive_info SET format = ?, total_size = ?, last_accessed_at = ?
WHERE id = ?";

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

         pstmt.setString(1, info.getFormat());
         pstmt.setLong(2, info.getTotalSize());
         pstmt.setObject(3, info.getLastAccessedAt());
         pstmt.setLong(4, info.getId());
```

```
            pstmt.executeUpdate();

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

    public void add(ArchiveInfo info) {
        String sql = "INSERT INTO archive_info(file_path, format, total_size, created_at,
last_accessed_at) VALUES(?, ?, ?, ?, ?)";
        try (Connection conn = DatabaseConnector.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, info.getFilePath());
            pstmt.setString(2, info.getFormat());
            pstmt.setLong(3, info.getTotalSize());
            pstmt.setObject(4, info.getCreatedAt());
            pstmt.setObject(5, info.getLastAccessedAt());
            pstmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void upsert(ArchiveInfo info) {
        Optional<ArchiveInfo> existing = findByPath(info.getFilePath());
        if (existing.isPresent()) {
            info.setId(existing.get().getId());
            update(info);
        } else {
            add(info);
        }
    }

    @Override
    public void delete(long id) {
        String sql = "DELETE FROM archive_info WHERE id = ?";

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

            pstmt.setLong(1, id);
            pstmt.executeUpdate();

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

    @Override
    public List<ArchiveInfo> listRecent(int limit) {
        String sql = "SELECT id, file_path, format, total_size, created_at, last_accessed_at FROM
```

```
archive_info ORDER BY last_accessed_at DESC LIMIT ?";
        List<ArchiveInfo> result = new ArrayList<>();

        try (Connection conn = DatabaseConnector.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, limit);
            ResultSet rs = pstmt.executeQuery();
            while (rs.next()) {
                ArchiveInfo info = new ArchiveInfo(
                        rs.getLong("id"),
                        rs.getString("file_path"),
                        rs.getString("format"),
                        rs.getLong("total_size"),
                        rs.getObject("created_at", LocalDateTime.class),
                        rs.getObject("last_accessed_at", LocalDateTime.class)
                );
                result.add(info);
            }

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

PostgresOperationDetailRepository

```
package repository;

import model.OperationDetail;
import model.OperationStatus;
import model.OperationType;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class PostgresOperationDetailRepository implements IOperationDetailRepository {
    @Override
    public void add(OperationDetail detail) {
        String sql = "INSERT INTO operationdetail(logid, itempath, action, status, message)
VALUES(?,?,?,?,?)";
        try(Connection conn = DatabaseConnector.getConnection();
            PreparedStatement pstmt = conn.prepareStatement(sql)){
            pstmt.setLong(1,detail.getLogID());
            pstmt.setString(2,detail.getItemPath());
            pstmt.setString(3,detail.getType().name());
            pstmt.setString(4,detail.getStatus().name());
            pstmt.setString(5,detail.getMessage());
            pstmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
```

```
        }


    }

    @Override
    public void addAll(List<OperationDetail> details) {
        for (OperationDetail detail : details){
            add(detail);
        }
    }

    @Override
    public List<OperationDetail> listByOperation(long operationId) {
        String sql = "SELECT detailid, logid, itempath, \"action\", status, message FROM
operationdetail WHERE logid = ?";
        List<OperationDetail> result = new ArrayList<>();
        try(Connection conn = DatabaseConnector.getConnection()) {
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setLong(1,operationId);
            ResultSet rs = pstmt.executeQuery();
            while (rs.next()){
                OperationDetail operationDetail = new OperationDetail(
                        rs.getLong("detailid"),
                        rs.getLong("logid"),
                        rs.getString("itempath"),
                        OperationType.valueOf(rs.getString("action")),
                        OperationStatus.valueOf(rs.getString("status")),
                        rs.getString("message")
                );
                result.add(operationDetail);
            }

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

PostgresOperationLogRepository

```
package repository;

import model.OperationLog;
import model.OperationStatus;
import model.OperationType;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
```

```
public class PostgresOperationLogRepository implements IOperationLogRepository {

    @Override
    public long add(OperationLog log) {
        String sql = "INSERT INTO operation_log(archive_id, operation_type, status, \"timestamp\",
details) VALUES(?, ?, ?, ?, ?) RETURNING id";
        try (Connection conn = DatabaseConnector.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setLong(1, log.getArchiveInfoId());
            pstmt.setString(2, log.getOperationType().name());
            pstmt.setString(3, log.getStatus().name());
            pstmt.setObject(4, log.getTimestamp());
            pstmt.setString(5, log.getMessage());

            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                return rs.getLong("id");
            }
            throw new SQLException("Не вдалось отримати ID");

        } catch (SQLException e) {
            e.printStackTrace();
            return 0;
        }
    }

    @Override
    public List<OperationLog> listByArchive(long archiveId, int limit) {

        String sql = "SELECT id, archive_id, operation_type, status, \"timestamp\", details FROM
operation_log WHERE archive_id = ? ORDER BY \"timestamp\" DESC limit ?";
        List<OperationLog> result = new ArrayList<>();
        try(Connection conn = DatabaseConnector.getConnection();
            PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setLong(1,archiveId);
            pstmt.setInt(2,limit);
            ResultSet rs = pstmt.executeQuery();
            while (rs.next()){
                OperationLog operationLog = new OperationLog(
                        rs.getLong("id"),
                        rs.getLong("archive_id"),
                        OperationType.valueOf(rs.getString("operation_type")),
                        OperationStatus.valueOf(rs.getString("status")),
                        rs.getObject("timestamp", LocalDateTime.class),
                        rs.getString("details")
                );
                result.add(operationLog);
            }

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

```
        }
        return result;
    }
}
```

DatabaseConnector

```
package repository;

import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class DatabaseConnector {

    private static final Properties properties = new Properties();

    static {
        try (InputStream input =
DatabaseConnector.class.getClassLoader().getResourceAsStream("resources/config.properties")) {
            if (input == null) {
                System.out.println("Не вдалося знайти config.properties");
            } else {
                properties.load(input);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }


    public static Connection getConnection() throws SQLException {
        try {
            Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException e) {
            System.err.println("PostgreSQL JDBC Driver не знайдено");
            e.printStackTrace();
            throw new SQLException("JDBC Driver not found", e);
        }

        return DriverManager.getConnection(
                properties.getProperty("db.url"),
                properties.getProperty("db.user"),
                properties.getProperty("db.password")
        );
    }
}
```

Висновки

Під час виконання лабораторної роботи №3 я спроектував та реалізував систему "Архіватор" з використанням UML діаграм. Було створено діаграму розгортання, яка показує фізичну архітектуру з клієнтським комп'ютером (Java програма) та сервером бази даних PostgreSQL, з'єднаними через мережу. Діаграма компонентів демонструє структуру програми з розподілом на модулі інтерфейсу, бізнес-логіки та роботи з даними. Розроблено дві діаграми послідовностей для сценаріїв створення архіву та перевірки контрольної суми, які показують покрокову взаємодію між користувачем, інтерфейсом, логікою та базою даних. Програмна частина включає два вікна (головне з таблицею архівів та вікно історії операцій), реалізацію створення реальних ZIP архівів, збереження всієї інформації в PostgreSQL та відображення даних користувачу. Застосовано патерни проектування Strategy, Factory Method та Facade для гнучкої архітектури. Система має повний цикл роботи з даними від введення через діалоги до збереження в базі та показу в таблиці. Використання UML діаграм на етапі проектування допомогло спланувати архітектуру перед написанням коду, що спростило розробку та забезпечило якісну реалізацію системи.