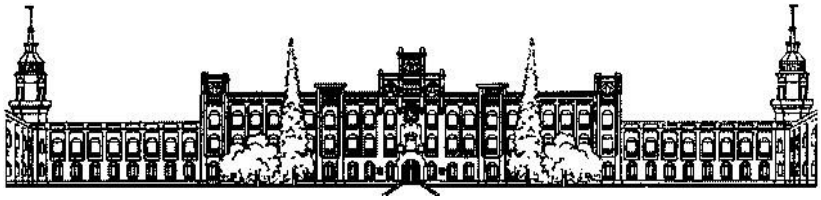

Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Лабораторна робота №3

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Основи проектування розгортання»

Виконав:

студент групи ІА-32

Нагорний Максим

Перевірив:

Мягкий М. Ю.

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.

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.

14. **Архіватор** (strategy, adapter, factory method, facade, visitor, p2p)

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) – додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

Теоретичні відомості

Діаграма розгортання(Deployment Diagram)

Діаграма розгортання — це структурна діаграма UML, яка моделює фізичне розміщення програмних артефактів на апаратних вузлах системи. Її основна мета — візуалізувати фізичну архітектуру системи та показати, як і де саме буде розгорнуто програмне забезпечення. Вона відповідає на питання: "Де будуть виконуватися програмні компоненти?". Основними елементами діаграми розгортання є вузол (Node), що представляє обчислювальний ресурс (фізичний сервер, віртуальна машина), та артефакт (Artifact) — фізичний результат процесу розробки (наприклад, виконуваний файл, бібліотека, база даних), який розгортається на вузлі. Вузли з'єднуються зв'язками (Communication Path), що показують шляхи комунікації між ними, зазвичай із зазначенням мережевого протоколу. Ці діаграми є важливими для планування інфраструктури, оцінки продуктивності та забезпечення надійності системи.

Діаграма компонентів (Component Diagram)

Діаграма компонентів є структурною діаграмою UML, що демонструє організацію та залежності між програмними компонентами системи. Вона розбиває систему на логічні модулі з чітко визначеними інтерфейсами, що дозволяє зрозуміти архітектуру на високому рівні, не заглиблюючись у деталі реалізації класів. Головним елементом є компонент (Component) — модульна частина системи, що інкапсулює свою поведінку та надає сервіси через інтерфейси. Інтерфейс (Interface) визначає набір операцій для взаємодії; він може бути наданим (provided), показуючи, що компонент реалізує цей сервіс, або необхідним (required), вказуючи, що компонент потребує сервісу від іншого компонента. Залежність (Dependency) показує, що один компонент використовує функціональність іншого. Діаграми компонентів корисні для проектування сервіс-орієнтованих архітектур, управління залежностями та планування повторного використання коду.

Діаграма послідовностей (Sequence Diagram)

Діаграма послідовностей — це діаграма взаємодії, яка показує, як об'єкти співпрацюють один з одним у часовій послідовності. Вона детально ілюструє логіку виконання конкретного сценарію або прецеденту (Use Case), фокусуючись на послідовності надісланих повідомлень. Час на діаграмі рухається зверху вниз. Ключовими елементами є лінія життя (Lifeline), що представляє існування об'єкта протягом певного часу, та повідомлення (Message) — стрілка між лініями життя, що позначає комунікацію, наприклад, виклик методу. Повідомлення можуть бути синхронними, коли відправник очікує на відповідь, або асинхронними, коли відправник

продовжує роботу, не чекаючи відповіді. Період, протягом якого об'єкт активний і виконує дію, позначається фокусом контролю (Activation Box) на його лінії життя. Діаграми послідовностей є незамінним інструментом для деталізації вимог, проектування логіки методів та аналізу взаємодії між об'єктами.

Хід виконання

Діаграма розгортання

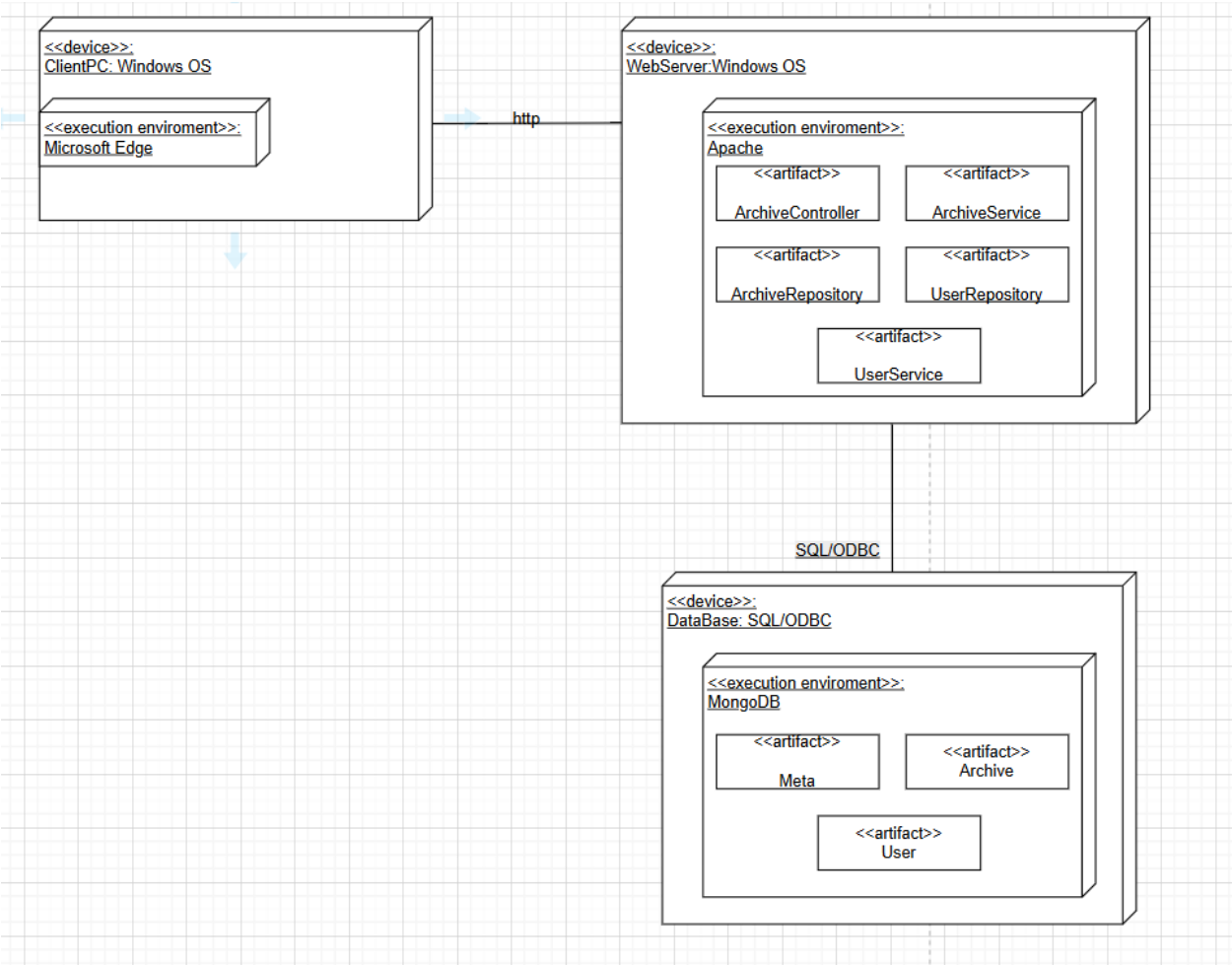

Рис.1. Діаграма розгортання системи

1) ClientPC – це вузо, що описує фізичний пристрій користувача, який працює на ОС Windows, всередині вузла знаходиться середовище виконання Microsoft Edge, пов’язаний з вузлом WebServer за допомогою протоколу HTTP;
2) WebServer – це вузол, що відповідає за обробку бізнес-логіки з середовищем виконання Apache, який містить в собі такі артефакти:
    1. ArchiveController – відповідає за прийом HTTP запитів від браузера та викликає сервіс для обробки;

2. UserService – відповідає за бізнес-логіку пов’язану з користувачем, таку як: авторизація, реєстрація.
3. ArchiveService - відповідає за бізнес-логіку пов’язану з створенням та редагуванням архіву.
4. UserRepository – відповідає за встановлення зв’язку сервісу та бази даних.
5. ArchiveRepository– відповідає за встановлення зв’язку сервісу та бази даних.

3) DataBase – цей вузол відповідає за зберігання та надання інформації, пов’язаний з WebServer через SQL, складається з виконавчого середовища MongoDB, що містить у собі такі артефакти:
   1. User – відповідає а користувачів;
   2. Archive – відповідає за архіви;
   3. Meta – відповідає за файли;

Діаграма компонентів

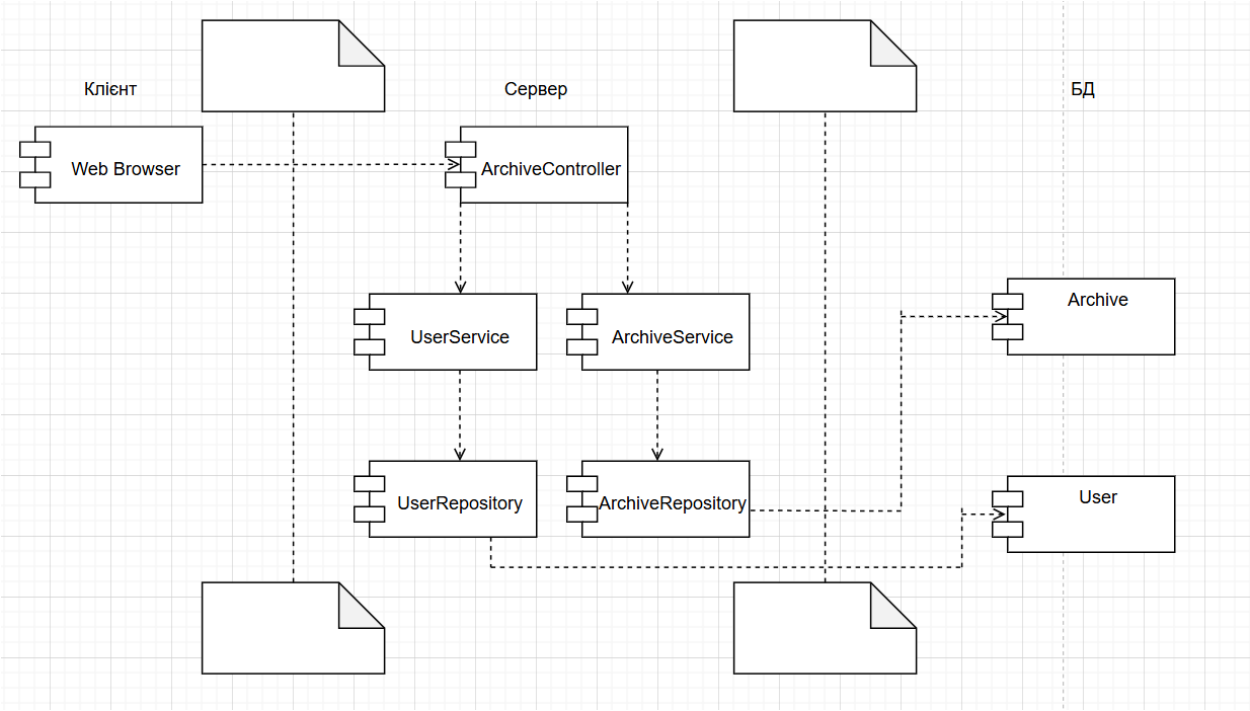

Рис.2. Діаграма компонентів

Система архіватора складається з трьох модулів:

1. Клієнт – складається з одного компоненту Web Browser, що відповідає за інтерфейс та відправку запитів на сервер, є простим браузером, з яким взаємодіє користувач;
2. Сервер – складається з п’яти компонентів: ArchiveController приймає HTTP запити від користувача, обробляє їх та передає сервісам, UserService та ArchiveService відповідають за бізнес логіку, UserService

відповідає за операції, пов'язані з користувачами, а ArchiveService містить логіку для роботи з архівами, компоненти UserRepository та ArchiveRepository відповідають за зв’язок сервісу та бази даних.

3. БД – складається з двох компонентів Archive та User, що являють собою сутності бази даних.

Діаграми послідовностей

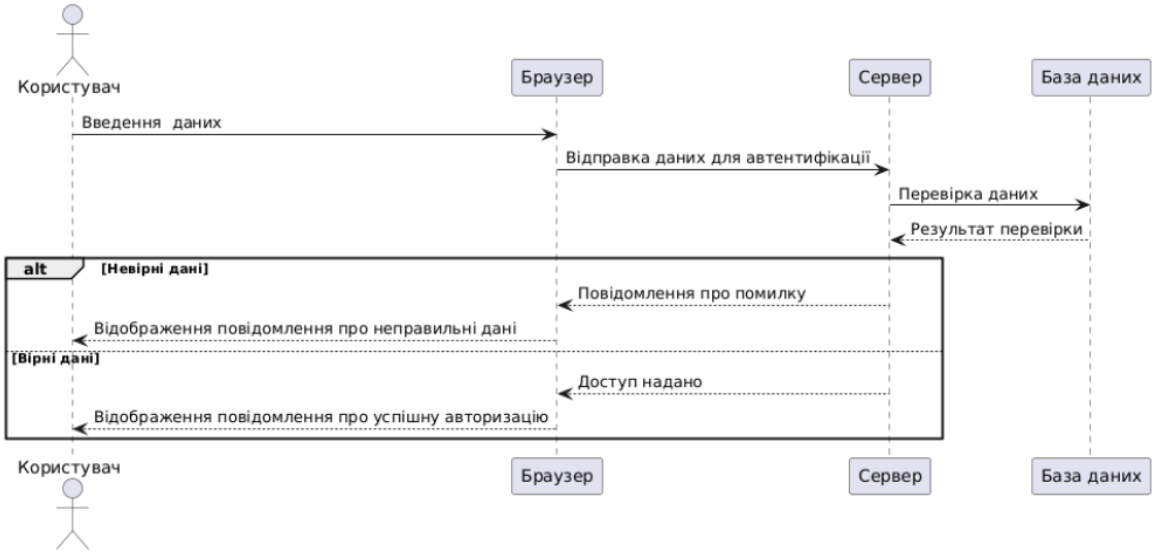

Рис.3. Діаграма послідовностей (авторизація)

Послідовність виконання:

1) Користувач вводить дані для авторизації;
2) Дані відправляються до сервера для перевірки;
3) Сервер звіряє отримані дані з БД;
4) Якщо користувача з такими даними знайдено, то сервер відправляє повідомлення про те, що авторизація пройшла успішно, в іншому випадку сервер відправляє помилку про те, що введені неправильні дані;

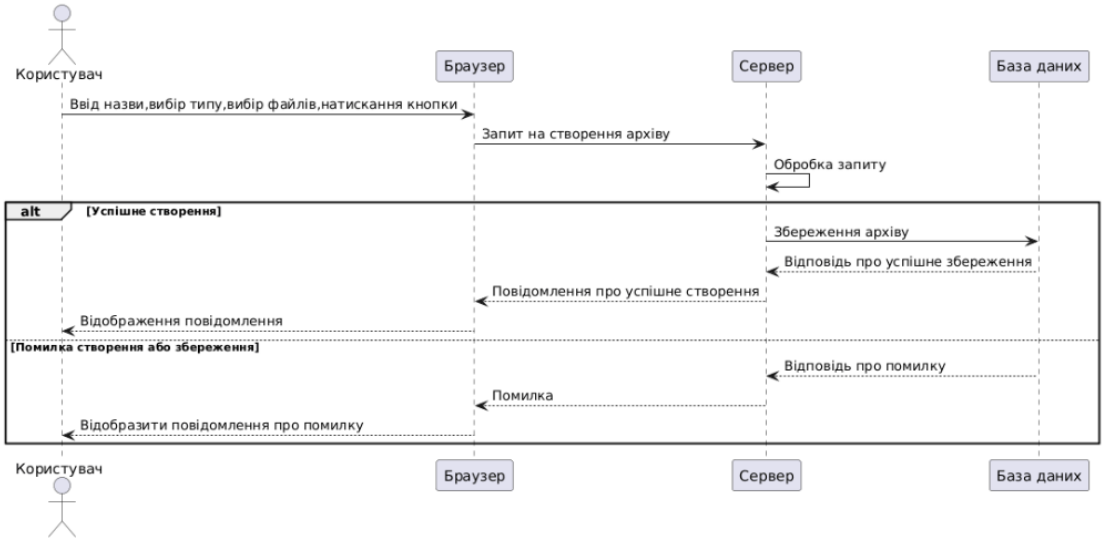

Рис.4. Діаграма послідовностей (створення архіву)

1) Користувач вводить назву, обирає тип архіву, обирає файли та натискає кнопку створити;
2) На сервер відправляється запит на створення нового архіву;
3) Сервер обробляє запит та надсилає новий архіву в БД;
4) Якщо новий архів зберігся в БД, то сервер відправляє повідомлення про успішне збереження, в іншому випадку повідомлення про те, що виникла помилка;

Код програми

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class LabApplication {
	public static void main(String[] args) {
		SpringApplication.run(LabApplication.class, args);
		System.out.println("Hello!");
	}
}
package com.example.demo.model;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import java.util.Set;
```

```
@Document(collection = "archives")
public class Archive {
    @Id
    private String id;
    private String name;
    private String checksum;
    private String archiveTypeId;
    private String userId;
    private Set<Meta> meta;
    private Set<String> breakArchiveIds;
    public Archive() {}
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getChecksum() {
        return checksum;
    }
    public void setChecksum(String checksum) {
        this.checksum = checksum;
    }
    public String getArchiveTypeId() {
        return archiveTypeId;
    }

    public void setArchiveTypeId(String archiveTypeId) {
```

```
        this.archiveTypeId = archiveTypeId;
    }
    public String getUserId() {
        return userId;
    }
    public void setUserId(String userId) {
        this.userId = userId;
    }
    public Set<Meta> getMeta() {
        return meta;
    }
    public void setMeta(Set<Meta> meta) {
        this.meta = meta;
    }
    public Set<String> getBreakArchiveIds() {
        return breakArchiveIds;
    }
    public void setBreakArchiveIds(Set<String> breakArchiveIds) {
        this.breakArchiveIds = breakArchiveIds;
    }
}
package com.example.demo.model;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
@Document(collection = "archive_types")
public class ArchiveType {
    @Id
    private String id;
    private String name;
    public ArchiveType() {}

    public String getId() {
        return id;
    }
```

```
    public void setId(String id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
package com.example.demo.model;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
@Document(collection = "broken_archives")
public class BreakArchive {
    @Id
    private String id;
    private String name;
    private long size;
    private String archiveId;
    public BreakArchive() {}
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
    public long getSize() {
        return size;
    }
    public void setSize(long size) {
        this.size = size;
    }
    public String getArchiveId() {
        return archiveId;
    }
    public void setArchiveId(String archiveId) {
        this.archiveId = archiveId;
    }
}
package com.example.demo.model;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
@Document(collection = "meta")
public class Meta {
    @Id
    private String id;
    private String fileName;
    private long size;
    public Meta() {}
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }

    public String getFileName() {
        return fileName;
    }
    public void setFileName(String fileName) {
```

```
        this.fileName = fileName;
    }
    public long getSize() {
        return size;
    }
    public void setSize(long size) {
        this.size = size;
    }
}
package com.example.demo.model;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import java.util.Set;
@Document(collection = "users")
public class User {
    @Id
    private String id;
    private String email;
    private String password;
    private String role;
    private Set<String> archiveIds;
    public User() {}
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }

    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
```

```
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getRole() {
        return role;
    }
    public void setRole(String role) {
        this.role = role;
    }
    public Set<String> getArchiveIds() {
        return archiveIds;
    }
    public void setArchiveIds(Set<String> archiveIds) {
        this.archiveIds = archiveIds;
    }
}
package com.example.demo.repository;
import org.springframework.data.mongodb.repository.MongoRepository;
import com.example.demo.model.Archive;
public interface ArchiveRepository extends MongoRepository<Archive, String> {}
package com.example.demo.repository;
import org.springframework.data.mongodb.repository.MongoRepository;

import com.example.demo.model.ArchiveType;
public interface ArchiveTypeRepository extends MongoRepository<ArchiveType, String> {}
package com.example.demo.repository;
import org.springframework.data.mongodb.repository.MongoRepository;
import com.example.demo.model.BreakArchive;
public interface BreakArchiveRepository extends MongoRepository<BreakArchive, String> {}
package com.example.demo.repository;
```

```
import org.springframework.data.mongodb.repository.MongoRepository;
import com.example.demo.model.Meta;
public interface MetaRepository extends MongoRepository<Meta, String> {}
package com.example.demo.repository;
import org.springframework.data.mongodb.repository.MongoRepository;
import com.example.demo.model.User;
public interface UserRepository extends MongoRepository<User, String> {}
package com.example.demo.controller;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import com.example.demo.model.Archive;
import com.example.demo.model.Meta;
import com.example.demo.repository.ArchiveRepository;
import java.io.IOException;
import java.security.MessageDigest;
import java.util.HashSet;
import java.util.Set;
@Controller
@RequestMapping("/archive")
public class ArchiveController {
    private final ArchiveRepository archiveRepository;
    public ArchiveController(ArchiveRepository archiveRepository) {
        this.archiveRepository = archiveRepository;
    }

    @GetMapping("/archives")
    public String showArchives(Model model) {
        model.addAttribute("archives", archiveRepository.findAll());
        return "archives";
    }
    @GetMapping("/create")
    public String createArchiveForm(Model model) {
```

```
        model.addAttribute("archive", new Archive());
        model.addAttribute("archiveTypes", new String[]{"zip", "rar", "7z"});
        return "create_archive";
    }
    @PostMapping("/create")
    public String createArchiveSubmit(
            @ModelAttribute("archive") Archive archive,
            @RequestParam("files") MultipartFile[] files) throws IOException {
        Set<Meta> metaSet = new HashSet<>();
        for (MultipartFile file : files) {
            if (!file.isEmpty()) {
                Meta meta = new Meta();
                meta.setFileName(file.getOriginalFilename());
                meta.setSize(file.getSize());
                metaSet.add(meta);
            }
        }
        archive.setMeta(metaSet);
        archive.setChecksum(generateChecksum(archive));
        Set<String> parts = new HashSet<>();
        parts.add("part1");
        parts.add("part2");
        archive.setBreakArchiveIds(parts);
        archiveRepository.save(archive);


        return "redirect:/archive/archives";
    }
    private String generateChecksum(Archive archive) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            String data = archive.getName() + archive.getArchiveTypeId() +
System.currentTimeMillis();
            byte[] hash = digest.digest(data.getBytes());
            StringBuilder hexString = new StringBuilder();
```

```
            for (byte b : hash) {
                hexString.append(String.format("%02x", b));
            }
            return hexString.toString();
        } catch (Exception e) {
            return "error";
        }
    }
}
package com.example.demo.controller;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import com.example.demo.model.User;
import com.example.demo.repository.UserRepository;
import java.util.List;
@Controller
@RequestMapping("/user")
public class UserController {
    private final UserRepository userRepository;
    public UserController(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
    @GetMapping("users")
    public String showUsers(Model model) {
        model.addAttribute("users", userRepository.findAll());
        return "users";
    }
    @GetMapping("/create")
    public String createUserForm(Model model) {
        model.addAttribute("user", new User());
        return "create";
    @PostMapping("/create")
    public String createUserSubmit(@ModelAttribute("user") User user) {
```

```
        userRepository.save(user);
        return "redirect:/user/users";
    }
}
```

Висновок

Під час виконання лабораторної роботи ми дізналися, що таке діаграма розгортання системи, з яких компонентів вона складається, як використовується діаграма компонентів та на які модулі вона розбивається та навчилися розробляти діаграми послідовності, для побудови діаграм обрали систему draw.io для побудування наших uml-діаграм. Почали ми з створення діаграми розгортання системи, визначивши пристрої, які будуть взаємодіяти та артефакти, потім ми створили діаграму компонентів розбивши її на модулі та створили дві діаграми послідовностей для сценарію авторизації та сценарії створення архіву.

Питання до лабораторної роботи

1. **Що собою становить діаграма розгортання?**

   Діаграма розгортання — це структурна діаграма UML, яка показує фізичну архітектуру системи. Вона візуалізує, як артефакти розміщуються на вузлах і як ці вузли з'єднані між собою.

2. **Які бувають види вузлів на діаграмі розгортання?**

   Існує два види вузлів:

   1) Пристрій - редставляє фізичний обчислювальний ресурс з процесором і пам'яттю, наприклад, сервер, персональний комп'ютер
   2) Середовище виконання – це програмне середовище, яке працює всередині пристрою

3. **Які бувають зв’язки на діаграмі розгортання?**
   1) Залежність вказується ,коли один вузол залежить від іншого.
   2) Асоціації між вузлами – вказують, який компонент на якому вузлі розміщено.
4. **Які елементи присутні на діаграмі компонентів?**
   1) Компоненти — самостійні модулі або пакети програми.
   2) Інтерфейси (interfaces) — точки взаємодії компонентів.
   3) Зв’язки — вказують, які компоненти залежать один від одного або використовують сервіси інших.
5. **Що становлять собою зв’язки на діаграмі компонентів?**

Зв’язки між компонентами показують залежність, який компонент залежить від якого.

6. **Які бувають види діаграм взаємодії?**
   1) Діаграми послідовностей — показують послідовність обміну повідомленнями між об’єктами.
   2) Діаграми комунікацій — фокусуються на структурі взаємодії об’єктів.
   3) Діаграми часу — відображають зміну станів об’єктів у часі.
   4) Діаграми взаємодії — комбінують різні сценарії взаємодії.
7. **Для чого призначена діаграма послідовностей?**
   Діаграма послідовностей призначена для опису динаміки системи — показує, які об’єкти взаємодіють, які повідомлення надсилаються і в якій послідовності для виконання певного сценарію.
8. **Які ключові елементи можуть бути на діаграмі послідовностей?**
   1) Актори;
   2) Активності;
   3) Повідомлення;
9. **Як діаграми послідовностей пов’язані з діаграмами варіантів використання?**
   Діаграма варіантів використання показує, що система робить, а діаграма послідовностей показує, як система це робить, деталізуючи один сенарій.
10. **Як діаграми послідовностей пов’язані з діаграмами класів?**
    Діаграма класів описує статичну структуру системи,а діаграма послідовностей описує, як об’єкти класів взаємодіють між собою.