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

Лабораторна робота №**9**
з предмету «Технології розроблення програмного забезпечення»
на тему «**Різні види взаємодії додатків: client-server, peer-to-peer, service-oriented architecture**»

Виконав
студент групи ІА-23:
Воронюк Є. В.

Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Мета: реалізація частини функціоналу робочої програми у вигляді класів та їх взаємодій для досягнення конкретних функціональних можливостей та реалізація взаємодії із програмою клієнт-серверною архітектурою.

Завдання 1: ознайомитися з короткими теоретичними відомостями.
Завдання 2: реалізувати частину функціоналу робочої програми у вигляді класів і їх взаємодій для досягнення конкретних функціональних можливостей.
Завдання 3: реалізувати взаємодію програми в одній з архітектур відповідно до обраної теми.

Тема для виконання:
22 - FTP-server (state, builder, memento, template method, visitor, client- server)
FTP-сервер повинен вміти коректно обробляти і відправляти відповіді по протоколу FTP, з можливістю створення користувачів (з паролями) і доступних їм папок, розподілу прав за стандартною схемою (rwe), ведення статистики з'єднань, обмеження максимальної кількості підключень і максимальної швидкості поширення глобально і окремо для кожного облікового запису.

Короткі теоретичні відомості:

Клієнт-серверні додатки.

Клієнт-серверні додатки являють собою найпростіший варіант розподілених додатків, де виділяється два види додатків: клієнти (представляють додаток користувачеві) і сервери (використовується для зберігання і обробки даних). Розрізняють тонкі клієнти і товсті клієнти.101 Технології розроблення ПЗ

Тонкий клієнт - клієнт, який повністю всі операції (або більшість, пов'язаних з логікою роботи програми) передає для обробки на сервер, а сам зберігає лише візуальне уявлення одержуваних від сервера відповідей. Грубо кажучи, тонкий клієнт - набір форм відображення і канал зв'язку з сервером.

У такому варіанті використання дуже велике навантаження лягає на сервер. Раніше такий варіант був доступний, оскільки обчислень було небагато, термінали були дорогими і як пра-вило йшли з урізаним обчислювальним пристроєм. У нинішній час зі зростанням обчислювальних можливостей клієнтських машин має сенс частину обчислень перекладати на клієнтські комп'ютери.

Однак дана модель має сенс ще в захищених сценаріях - коли зайві дані не можна показувати клієнтським комп'ютерам в зв'язку з небезпекою взлому. Ще однією проблемою може бути множинний доступ - щоб уникнути конфліктності даних має сенс централізація обчислень в одному місці (на сервері) для визначення послідовності операцій та уникнення пошкодження даних.

Товстий клієнт - антипод тонкого клієнта, більшість логіки обробки даних містить на стороні клієнта. Це сильно розвантажує сервер. Сервер в таких випадках зазвичай працює лише як точка доступу до деякого іншого ресурсу (наприклад, бази даних) або сполучна ланка з іншими клієнтськими комп'ютерами.

У моделі клієнт-серверної взаємодії також важливе місце займає модель «підписки / видачі». Комп'ютери клієнтів можуть «підписуватися» серверу на певний набір подій (поява нових користувачів, зміна даних тощо), а сервер в свою чергу сповіщає клієнтські комп'ютери про походження цих подій. Даний спосіб взаємодії реалізується за допомогою шаблону «оглядач»; він несе велике навантаження на сервер (необхідно відстежувати хто на що підписався) і на канали зв'язку (багато повторюваних даних відправляються на різні точки в мережі). З іншого боку, це дуже зручно для клієнтських машин для організації оновлення даних без їх повторного перезапросу. Природно, інші механізми включають ліниву ініціалізацію, перезапрос, запит за таймером, запит за потребою (після натискання кнопки «оновити» наприклад).

Клієнт-серверна взаємодія, як правило, організовується за допомогою 3-х рівневої структури: клієнтська частина, загальна частина, серверна частина.

Оскільки велика частина даних загальна (класи, використовувані системою), їх прийнято виносити в загальну частину (middleware) системи.

Клієнтська частина містить візуальне відображення і логіку обробки дії користувача; код для встановлення сеансу зв'язку з сервером і виконання відповідних викликів.

Серверна частина містить основну логіку роботи програми (бізнес-логіку) або ту її частину, яка відповідає зберіганню або обміну даними між клієнтом і сервером або клієнтами.

Додатки типу «peer-to-peer».

Модель взаємодії додатків типу peer-to-peer має на увазі рівноправ'я клієнтських програм і відсутність серверної програми. Усі клієнтські програми контактують між собою для виконання спільних цілей. У таких мережах

найчастіше виникають наступні проблеми: синхронізація даних, пошук клієнтських застосувань.

Для пошуку клієнтських застосувань може використовуватися одна загальна адреса, що містить набір адрес підключених клієнтів. Це в деякому роді нагадує сервер. Проте сервер є лише центром сходження безлічі клієнтів, а адреси можуть зберігатися в самих клієнтських застосуваннях. Це називається структуровані одноранговi мережі.

Для вирішення проблеми синхронізації даних використовуються традиційні підходи: можливий варіант договору про набір для синхронізації (negotiation), або визначення по деякому алгоритму порівняння даних (hash-алгоритми і інші). Як правило, peer-to-peer застосування є звичайними застосуваннями з продуманою інфраструктурою обміну повідомленнями між клієнтами. Для таких випадків як правило розробляються спеціальні формати і протоколи обміну для структуризації діалогу між клієнтами.

Сервіс-орієнтована архітектура.

Сервіс-орієнтована архітектура (SOA, англ. service-oriented architecture) —- модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабо пов’язаних (англ. Loose coupling) замінних компонентів, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами.

Програмні комплекси, розроблені відповідно до сервіс-орієнтованою архітектурою, зазвичай реалізуються як набір веб-служб, взаємодіючих за протоколом SOAP, але існують і інші реалізації (наприклад, на базі jini, CORBA, на основі REST).

Інтерфейси компонентів в сервіс-орієнтованій архітектурі інкапсулюють деталі реалізації (операційну систему, платформу, мову програмування) від інших компонентів, таким чином забезпечуючи комбінування і багаторазове використання компонентів для побудови складних розподілених програмних комплексів, забезпечуючи незалежність від використовуваних платформ та інструментів розробки, сприяючи масштабованості і керованості створюваних систем. SaaS (англ. Software as a service - програмне забезпечення як послуга; також англ. Software on demand - програмне забезпечення на вимогу) - бізнес-модель продажу та використання програмного забезпечення, при якій постачальник розробляє веб-додаток і самостійно керує ним, надаючи замовнику доступ до

програмного забезпечення через Інтернет. Основна перевага моделі SaaS для споживача послуги полягає у відсутності витрат, пов'язаних з установкою, оновленням і підтримкою працездатності обладнання і працюючого на ньому програмного забезпечення.

В рамках моделі SaaS замовники платять не за володіння програмним забезпеченням як таким, а за його оренду (тобто за його використання через веб-інтерфейс). Таким чином, на відміну від класичної схеми ліцензування ПЗ, замовник несе порівняно невеликі періодичні витрати, і йому не потрібно інвестувати значні кошти в придбання ПЗ та апаратної платформи для його розгортання, а потім підтримувати його працездатність. Схема періодичної оплати передбачає, що якщо необхідність в програмному забезпеченні тимчасово відсутня, то замовник може призупинити його використання і заморозити виплати розробнику.

З точки зору розробника пропрієтарного програмного забезпечення модель SaaS дозволяє ефективно боротися з неліцензійним програмним забезпеченням, оскільки ПО як таке не потрапляє до кінцевих замовників. Крім того, концепція SaaS часто дозволяє зменшити витрати на розгортання і впровадження систем технічної і консультаційної підтримки продукту, хоча і не виключає їх повністю..

Реалізація SaaS ґрунтується на моделі сервіс-орієнтованої архітектури. Як правило, розробляється сервіс без стану (stateless) і викладається в хмару (cloud). Оскільки сервіс не має стану, для зберігання даних авторизації користувачів необхідно використовувати спеціальний маркер доступу (token), що видається при успішній авторизації.

Мікро-сервісна архітектура.

Сама назва дає зрозуміти, що архітектура мікрослужб є підходом до створення серверного додатку як набору малих служб. Це означає, що архітектура мікрослужб головним чином орієнтована на серверну частину, не дивлячись на те, що цей підхід так само використовується для зовнішнього інтерфейсу, де кожна служба виконується в своєму процесі і взаємодіє з іншими службами за такими протоколами, як HTTP/HTTPS, WebSockets чи AMQP. Кожна мікрослужба реалізує специфічні можливості в предметній області і свою бізнес- логіку в рамках конкретного обмеженого контексту, повинна розроблятись автономно і розвертатись незалежно.

Чому варто використовувати архітектуру мікрослужб? Якщо говорити коротко, то це пружність в довгостроковій перспективі. Мікрослужби забезпечують чудові можливості супроводження в величезних комплексних системах з високою масштабуємістю за рахунок створення додатків, заснованих на множині незалежно розгортуючих служб з автономними життєвими циклами.

Хід роботи::

Структура проекту ftp-сервера:

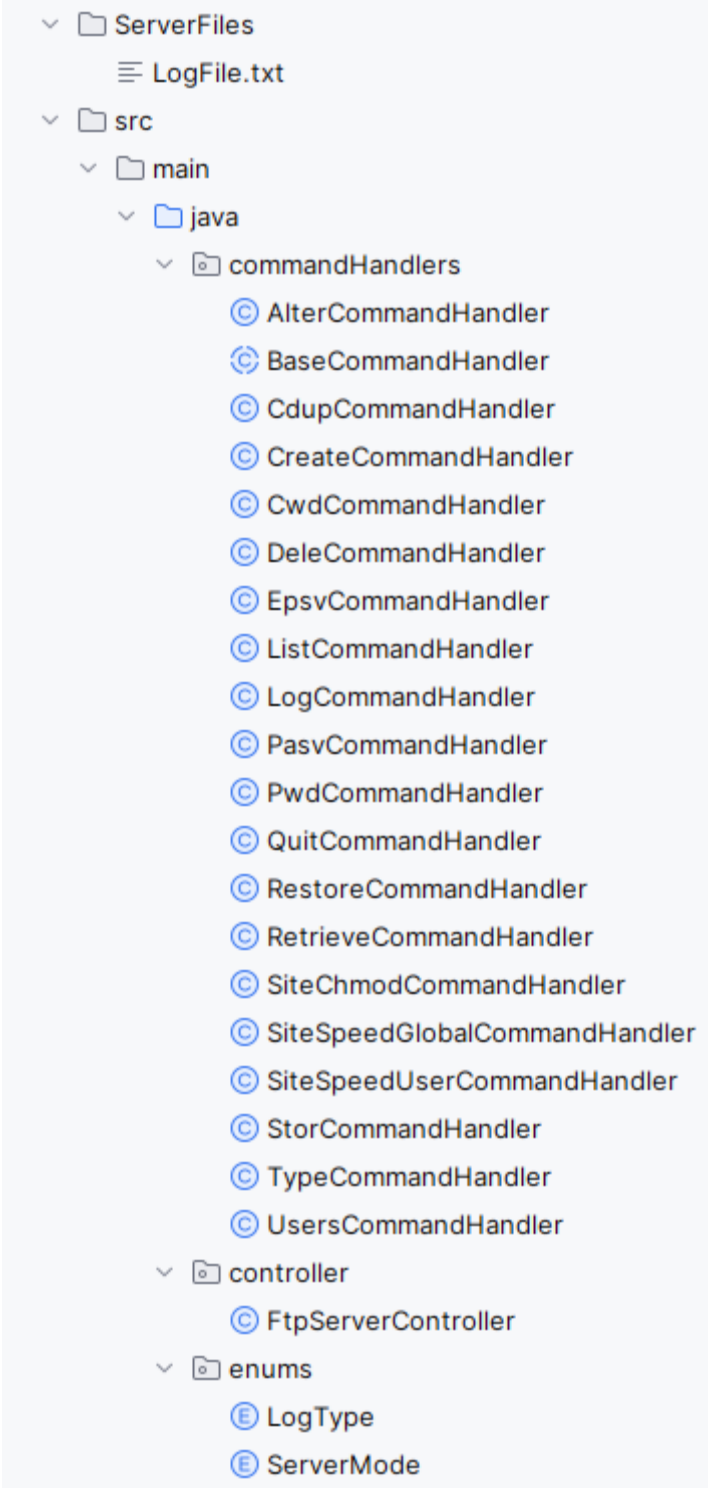

Рисунок 1.1 – структура проекту ftp-сервера

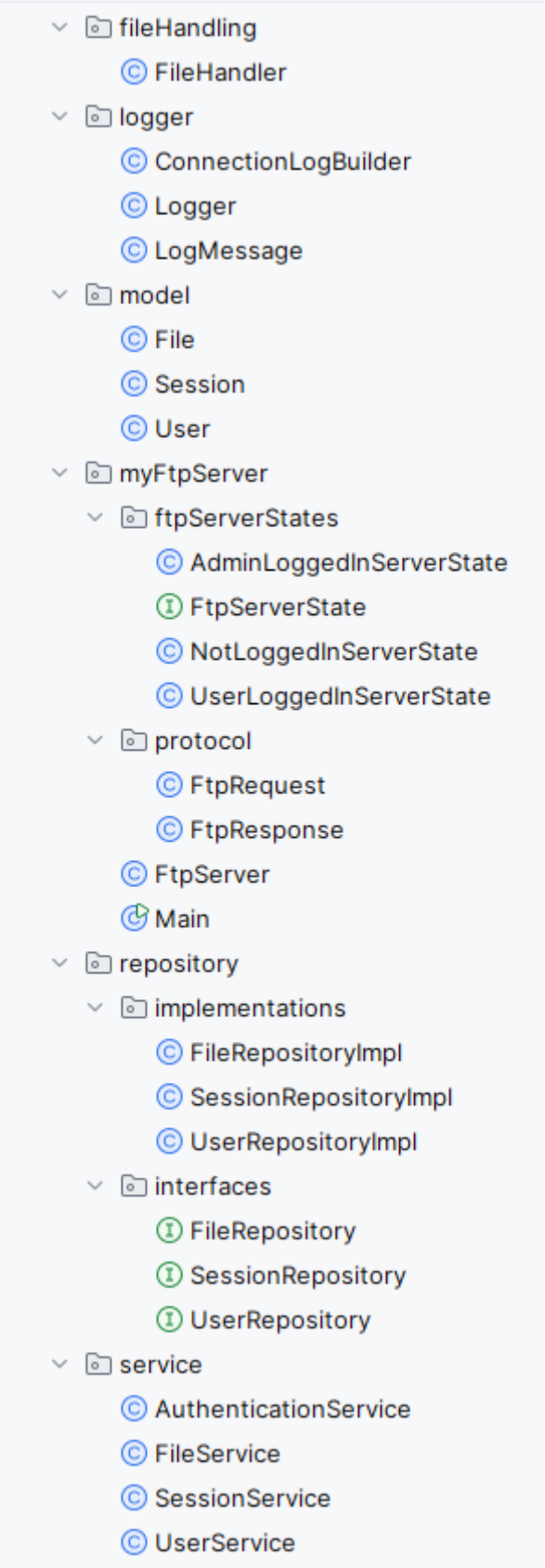

Рисунок 1.2 – структура проекту ftp-сервера

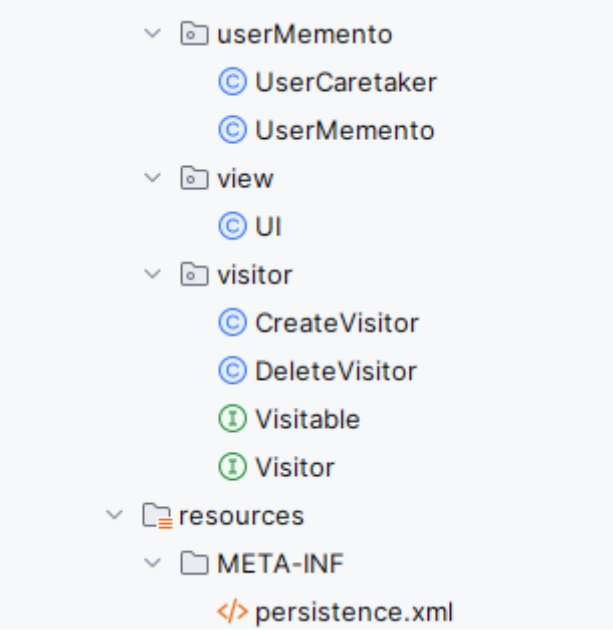

Рисунок 1.3 – структура проекту ftp-сервера

Опис структури ftp-сервера:

Директорія **ServerFiles** містить один файл у розширенні txt для запису логів серверу – а саме входу користувачів, виходу користувачів; відбувається реєстрація помилок на сервері.

Директорія **commandHandlers** включає класи, які містять логіку обробки команд, надісланих на сервер. Тут використано шаблон Template Method, який визначає кістяк алгоритмів роботи сервера та дозволяє підкласам BaseCommandHandler'а реалізовувати логіку виконання команд.

Директорія **controller** містить один клас контролеру сервера, який керує найважливішими процесами взаємодії сервера з користувачами та сесіями.

Директорія **enums** ініціалізовує два enum – для виду логів, які записуються в LogFile.txt (LOG_IN_TYPE, LOG_OUT_TYPE, ERROR_TYPE) та режиму сервера (PASSIVE).

Директорія **fileHandling** містить один клас fileHandler, на який покладено задачу готувати вивід логів, коли адміністратор використовує команду LOG.

Директорія **logger** містить класи: ConnectionLogBuilder – будівник логів, Logger – основний клас для логування, LogMessage - структура повідомлень логів. Тут використовується шаблон Builder для формування логів, які записуються у LogFile.txt.

Директорія **model** містить моделі File – файл, Session – сесія та User – користувач. Тут визначаються поля основних сутностей на сервері.

Піддиректорія **ftpServerStates** містить три стани серверу: для адміністратора, для юзера, для незалогіненого користувача. Містить інтерфейс FtpServerState для всіх станів серверу. Саме тут ініціалізовано команди, які можуть виконувати користувачі сервером (admin, user, not logged). Для реалізації даної було застосовано шаблон State.

Піддиректорія **protocol**: FtpRequest (обробка FTP-запитів), FtpResponse (формування відповідей)

Директорія **myFtpServer** включає піддиректорії ftpServerStates, protocol та класи: FtpServer (основний клас сервера), Main (точка входу).

Директорія **repository** включає дві піддиректорії: **interfaces** - FileRepository, SessionRepository, UserRepository (інтерфейси для доступу до даних) та **implementations** - FileRepositoryImpl, SessionRepositoryImpl, UserRepositoryImpl (реалізації інтерфейсів).

Директорія **service** містить наступні класи: AuthenticationService (логіка авторизації), FileService (управління файлами), SessionService (управління сесіями), UserService (управління користувачами).

Директорія **userMemento**: містить класи UserCaretaker та UserMemento (реалізація патерну "Memento" для збереження стану користувача).

Директорія **view**: клас UI відповідає за взаємодію користувача з сервером шляхом введення та виведення даних. Цей клас реалізує інтерфейс вводу/виводу для FTP-сервера і використовується для відображення даних клієнта та прийому команд.

Директорія **visitor** містить класи та інтерфейси для реалізації шаблону Visitor.

persistence.xml - використовується для налаштування JPA (Java Persistence API) та Hibernate як постачальника реалізації. Він визначає конфігурацію підключення до бази даних та керування сутностями на FTP-сервері.

FTP-клієнт:
Для роботи із сервером необхідне певне середовище – FTP-клієнт. Це може бути спеціалізована програма для роботи по FTP, або звичайний командний рядок, із якого встановлено з’єднання із сервером.
Я використовую дві спеціалізовані програми: FileZilla та WinSCP, проте надаю перевагу FileZilla. Для підключення до серверу необхідно: хост (127.0.0.1 / localhost),

логін – ім’я користувача, пароль та порт (стандартно 21, але порт можна змінити у класі FtpServer).

Тепер ми можемо робити вхідні запити до ftp-серверу за протоколом FTP за допомогою різних ftp-команд.

Практичне використання ftp-сервера:
Команди виконуються в Windows PowerShell.

```
ftp> quote ALTER -password 3333
200 User successfully updated.
ftp> quote ALTER -username user33
200 User successfully updated.
ftp> quote ACCT
230 username: user33; has admin rights: false.
```

Рисунок 2 – використання команди ALTER для зміни паролю або імені користувача. Команда виконується юзером (user).

```
ftp> quote PWD
257 D:/ftp-server/DataStorage/\ is the current directory.
ftp> quote CWD user2
250 Directory successfully changed.
ftp> quote PWD
257 D:\ftp-server\DataStorage\user2\ is the current directory.
ftp> quote CDUP
200 Changed to parent directory.
ftp> quote PWD
257 D:\ftp-server\DataStorage\ is the current directory.
ftp> quote ACCT
230 username: admin; has admin rights: true.
ftp>
```

Рисунок 3 – використання команд PWD для перевірки поточної директорії, CWD для зміни директорії (вказується або абсолютний шлях, або відносний), CDUP для повернення в батьківську директорію, яка закріплена за будь-яким користувачем.
Команда виконується адміністратором.

```
ftp> quote CREATE user3 1111 admin+
501 Syntax error in parameters or arguments. Argument 3 must be true or false.
ftp> quote CREATE user3 1111 false
230 User successfully created.
ftp> quote CREATE user3 1112 false
501 Syntax error in parameters or arguments. Username is already occupied.
ftp>
```

Рисунок 4 – створення нового користувача командою CREATE.
Створення може виконувати тільки адмін, треба вказати наступні параметри: юзернейм, пароль, права адміністратора (true / false).
Бачимо обробку винятків від ftp-сервера.

```
ftp> quote CWD D:\Tools\
550 Access denied: Cannot go above home directory..
```

Рисунок 5 – будь-хто: адміністратор чи користувач не може вийти за межі своєї робочої зони.

```
ftp> quote PWD
257 D:/ftp-server/DataStorage/\ is the current directory.
ftp> quote DELE test11.txt
204 File deleted successfully.
ftp> quote DELE txt.test11.txt
404 File not found.
```

Рисунок 6 – використання команди DELE для видалення файлів. Команда є доступною як юзерам, так і адмінам.

```
ftp> quote EPSV
229 Entering Extended Passive Mode (|||51814|).
ftp> quote LIST
226 Transfer complete.
```

Рисунок 7 – отримання списку файлів у поточній директорії командою LIST.

```
12-18-2024 17:19                0 testuser3.bmp
```

Рисунок 8 – відповідь на команду LIST – бачимо таблицю із датою завантаження, розмір (тут файл пустий – 0 байт), та назву файлу.

```
ftp> quote USER user3
331 Username okay, need password.
ftp> quote PASS 1111
230 User successfully logged in.
```

Рисунок 9 – використання команд USER та PASS для аутентифікації на сервері.

```
ftp> quote LOG
Reading log file...
LOG IN: Client 0:0:0:0:0:0:0:1 logged in as user3 at 2024-12-18T17:18.
LOG OUT: Client 0:0:0:0:0:0:0:1 logged out as user3 at 2024-12-18T17:19.
ERROR: At 2024-12-18T17:19 client 0:0:0:0:0:0:0:1 logged as user3 faced this error: Socket closed
LOG OUT: Client 0:0:0:0:0:0:0:1 logged out as user3 at 2024-12-18T17:19.
LOG OUT: Client 0:0:0:0:0:0:0:1 logged out as user3 at 2024-12-18T17:19.
LOG IN: Client 0:0:0:0:0:0:0:1 logged in as user3 at 2024-12-18T17:19.
LOG OUT: Client 0:0:0:0:0:0:0:1 logged out as user3 at 2024-12-18T17:20.
LOG IN: Client 127.0.0.1 logged in as user3 at 2024-12-18T17:21.
LOG OUT: Client 127.0.0.1 logged out as user3 at 2024-12-18T17:22.
LOG IN: Client 127.0.0.1 logged in as admin at 2024-12-18T17:27.
LOG OUT: Client 127.0.0.1 logged out as admin at 2024-12-18T17:27.
LOG IN: Client 127.0.0.1 logged in as admin at 2024-12-18T17:29.
212 Transfer complete. Log file content successfully displayed.
ftp>
```

Рисунок 10 – використання команди LOG. Бачимо записи у файлі, кількість виведених рядків можна змінювати.

```
ftp> quote PASV
227 Entering Passive Mode (127,0,0,1,203,55).
ftp> quote EPSV
229 Entering Extended Passive Mode (|||52026|).
ftp>
```

Рисунок 11 – вхід у пасивний режим командою PASV. Вхід у розширений пасивний режим командою EPSV.

```
Received input: PASV
PASV Command issued. Server IP: 127,0,0,1, Port: 52023
Sending response: 227 Entering Passive Mode (127,0,0,1,203,55).
Received input: EPSV
Sending response: 229 Entering Extended Passive Mode (|||52026|).
```

Рисунок 12 – відповіді сервера на команди PASV та EPSV.

```
ftp> quote QUIT
221 Service closing control connection.
```

Рисунок 13 – вихід користувача та закриття з’єднання.

```
ftp> quote ACCT
230 username: user3; has admin rights: false.
ftp> quote RESTORE
202 No action taken, user state is current.
ftp> quote ALTER -username user333
200 User successfully updated.
ftp> quote ACCT
230 username: user333; has admin rights: false.
ftp> quote RESTORE
250 User state restored to the latest one.
ftp> quote ACCT
230 username: user3; has admin rights: false.
```

Рисунок 14 – приклад використання команди RESTORE – реалізовано з використанням шаблону Memento.

```
Received input: RETR testuser3.bmp
RETR command received. Sending file: D:\ftp-server\DataStorage\user3\testuser3.bmp
File sent successfully: D:\ftp-server\DataStorage\user3\testuser3.bmp
Sending response: 226 File transfer complete..
```

Рисунок 15 – робота із файлом – виконання команди RETR, вивантаження файлу із сервера.

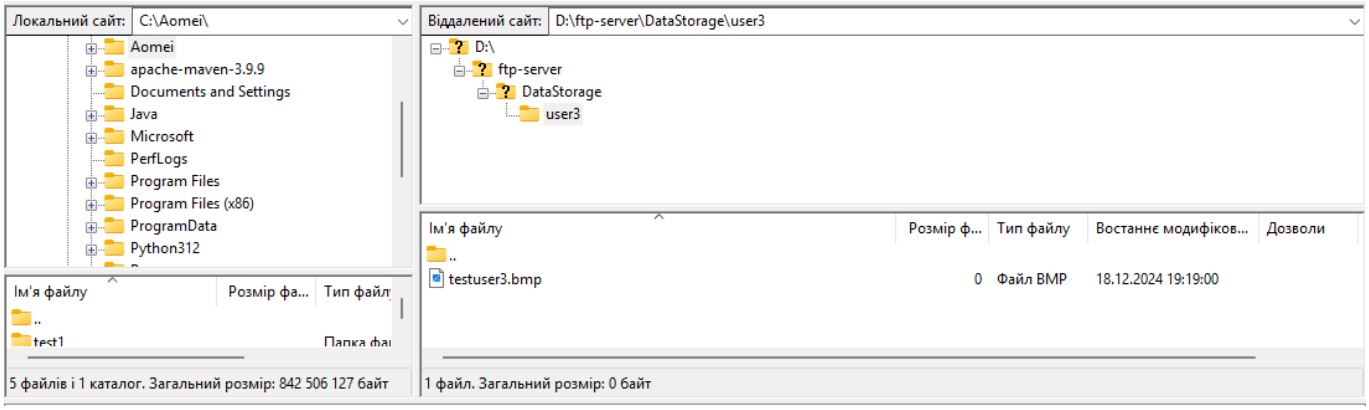

Рисунок 16 – інтерфейс програми FileZilla.

```
Received input: TYPE I
Sending response: 200 Type set to I (Binary mode).
Received input: EPSV
Sending response: 229 Entering Extended Passive Mode (|||51743|).
Received input: STOR testuser3.bmp
File received successfully: D:\ftp-server\DataStorage\user3\testuser3.bmp
File record created in database: fileId - 8; name - testuser3.bmp; location -D:\ftp-server\DataStorage\user3; owner - User{ userId=4, username=user3, isAdmin=false, homeDirectory=D:\ftp-server\DataStorage\user3}
Sending response: 226 File transfer complete..
```

Рисунок 17 – встановлення типу передачі файлу, та завантаження файлу на сервер командою STOR, логи сервера про збереження запису про файл у базі даних. Після вдалого завантаження ми можемо бачити файл у інтерфейсі програми FileZilla (рисунок 16).

```
ftp> quote CWD user3
250 Directory successfully changed.
ftp> quote SITE CHMOD 644
501 Syntax error in SITE CHMOD command arguments..
ftp> quote SITE CHMOD 644 testuser3.bmp
200 Permissions for testuser3.bmp set to 644.
ftp> quote ACCT
230 username: admin; has admin rights: true.
```

Рисунок 18 – задання прав доступу на файл командою SITE CHMOD

```
230 User successfully logged in.
ftp> quote SITE SPEED USER user3 50
200 Speed limit for user user3 set to 50 KB/s..
ftp> quote SITE SPEED GLOBAL 100
200 Global speed limit set to 100 KB/s..
```

Рисунок 19 – встановлення швидкості поширення для юзера та глобально командою SITE SPEED.

```
ftp> quote USERS
User{ userId=1, username=admin, isAdmin=true, homeDirectory=D:/ftp-server/DataStorage/}
User{ userId=2, username=user1, isAdmin=false, homeDirectory=D:\ftp-server\DataStorage\user1}
User{ userId=3, username=user33, isAdmin=false, homeDirectory=D:\ftp-server\DataStorage\user2}
User{ userId=4, username=user3, isAdmin=false, homeDirectory=D:\ftp-server\DataStorage\user3}
212 List of users successfully retrieved.
ftp>
```

Рисунок 20 – використання команди USERS для отримання інформації про користувачів та їх директорії.

Вихідний код системи:

FtpServer.java: центральний клас усього серверу.

```
package myFtpServer;

import controller.FtpServerController;
import logger.Logger;
import model.User;
import myFtpServer.ftpServerStates.FtpServerState;
import myFtpServer.ftpServerStates.NotLoggedInServerState;
import myFtpServer.protocol.FtpRequest;
import myFtpServer.protocol.FtpResponse;
import view.UI;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;

public class FtpServer {
    private static final int PORT = 21;
    private static final int MAX_CONNECTION_NUM = 5;
    private final FtpServerController controller = new FtpServerController();
    private Socket clientSocket;
    private UI ui;
    private final Logger logger;
    private User currentUser;
    private FtpServerState serverState;

    public FtpServer() throws IOException {
        logger = Logger.getLogger();
    }

    public void start() {
        try (ServerSocket commandServerSocket = new ServerSocket(PORT)) {
            System.out.println("FTP server started on port " + PORT);

            while (true) {
                try {
                    clientSocket = commandServerSocket.accept();
                    clientSocket.setSoTimeout(300000);
                    System.out.println("Client connected: " +
clientSocket.getInetAddress().getHostAddress());

                    ui = new UI(clientSocket.getInputStream(),
clientSocket.getOutputStream());
                    serverState = new NotLoggedInServerState(this);

                    Thread clientThread = new Thread(() ->
handleClient(clientSocket));
                    clientThread.start();
                } catch (IOException e) {
                    System.err.println("Error accepting client connection: " +
e.getMessage());
                }
            }
        } catch (IOException e) {
            System.err.println("Error starting FTP server: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private void handleClient(Socket clientSocket) {
        try {
```

```
            if (!controller.newUserCanConnect(MAX_CONNECTION_NUM)) {
                ui.displayFtpResponse(new FtpResponse(421, "Connection limit reached.
Please try again later."));
                Thread.sleep(30000);
                return;
            }

            ui.displayFtpResponse(new FtpResponse(220, "Successfully connected"));

            currentUser = new User();
            while (true) {
                String userInput = ui.acceptUserInput();
                if (userInput == null) {
                    System.out.println("Client disconnected.");
                    break;
                }
                if (userInput.isEmpty()) {
                    continue;
                }

                System.out.println("Received input: " + userInput);

                FtpRequest ftpRequest = new FtpRequest(userInput);
                FtpResponse ftpResponse = handleCommands(currentUser, ftpRequest);

                System.out.println("Sending response: " + ftpResponse.toString());
                ui.displayFtpResponse(ftpResponse);

                if (ftpResponse.getStatusCode() == 221) {
                    System.out.println("221 Service closing control connection.");
                    break;
                }
            }
        } catch (IOException e) {
            System.err.println("I/O Error handling client: " + e.getMessage());

logger.writeErrorEventToFile(clientSocket.getInetAddress().getHostAddress(),
currentUser.getUsername(), e.getMessage());
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            System.err.println("Thread interrupted: " + e.getMessage());
        } finally {
            cleanUpResources();
        }
    }

    private void cleanUpResources() {
        try {
            if (currentUser != null) {
                controller.deactivateSessionIfActive(currentUser.getUserId());

logger.writeLogOutEventToFile(clientSocket.getInetAddress().getHostAddress(),
currentUser.getUsername());
            }

            if (clientSocket != null && !clientSocket.isClosed()) {
                clientSocket.close();
                System.out.println("Client socket closed.");
            }

            if (ui != null) {
                ui.closeStreams();
                System.out.println("UI streams closed.");
            }
        } catch (IOException e) {
            System.err.println("Error cleaning up resources: " + e.getMessage());
```

```
        }
    }

    public FtpResponse handleCommands(User currentUser, FtpRequest ftpRequest) throws
IOException {
        return serverState.handleCommand(currentUser, ftpRequest);
    }

    public void setState(FtpServerState ftpServerState) {
        this.serverState = ftpServerState;
    }

    public void setUser(User loggedInUser) {
        currentUser.setUserId(loggedInUser.getUserId());
        currentUser.setUsername(loggedInUser.getUsername());
        currentUser.setPassword(loggedInUser.getPassword());
        currentUser.setHomeDirectory(loggedInUser.getHomeDirectory());
        currentUser.setAdmin(loggedInUser.getIsAdmin());
    }

    private String transferType = "I";

    public String getTransferType() {
        return transferType;
    }

    public void setTransferType(String transferType) {
        this.transferType = transferType;
    }

    public Socket getClientSocket() {
        return clientSocket;
    }

    public FtpServerController getController() {
        return controller;
    }

    public UI getUi() {
        return ui;
    }
}
```

Вихідний код усієї системи розміщено на GitHub: https://github.com/dobrogo-dnia/SDT-labs.

Висновок: під час виконання даної лабораторної роботи було реалізовано частину функціоналу робочої програми у вигляді класів та їх взаємодій для досягнення конкретних функціональних можливостей та реалізовано взаємодію із програмою за допомогою клієнт-серверної архітектури.