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

Технології розроблення програмного забезпечення

«РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE»

FTP-server

Виконав
студент групи ІА–22:
Білокур Євгеній
Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Зміст

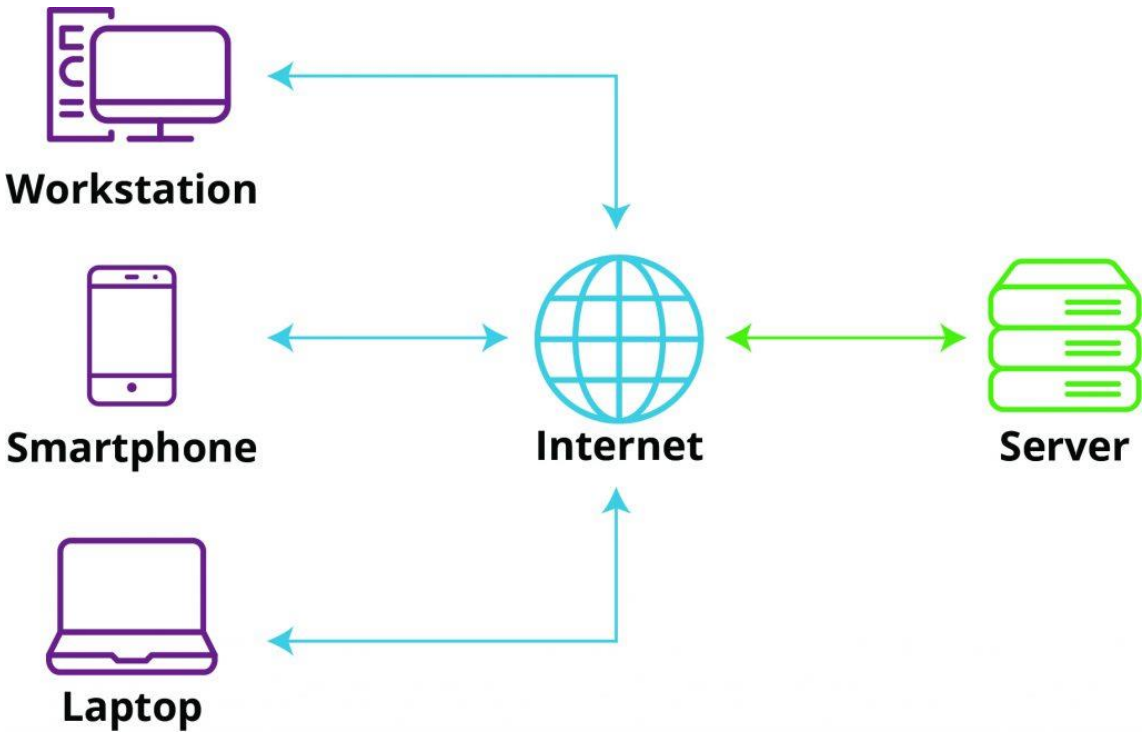
Тема: «РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE»

Мета: ознайомитися з різними видами взаємодії додатків: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE та набути практичних навичок їх застосування. Реалізувати частину функціоналу програми за допомогою одного з розглянутих шаблонів для досягнення конкретних функціональних можливостей та забезпечення ефективної взаємодії між класами.

Хід роботи

1. Короткі теоритичні відомості

Клієнт-серверні додатки



Клієнт-серверні додатки — це тип розподілених систем, які складаються з двох компонентів: клієнта (для взаємодії з користувачем) і сервера (для обробки та зберігання даних).

Типи клієнтів:

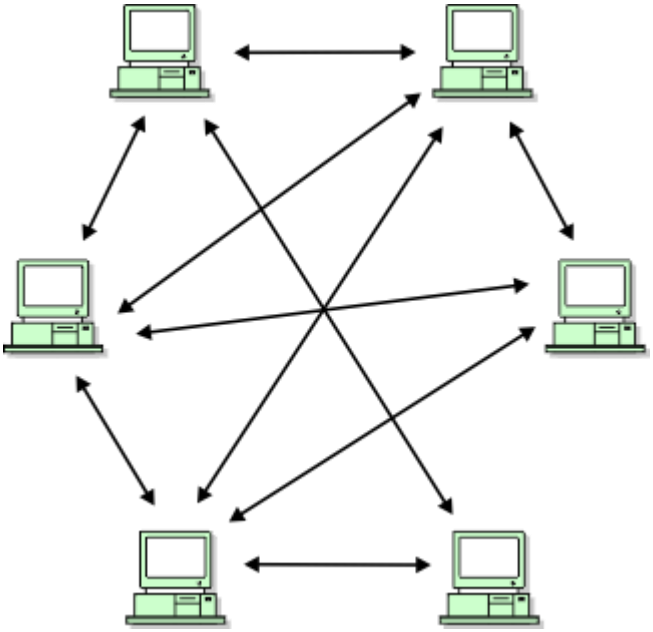
1. **Тонкий клієнт** — передає обробку логіки додатка на сервер і лише відображає результати. Недоліки: високе навантаження на сервер, переваги — захист даних.
2. **Товстий клієнт** — виконує більшість обчислень на клієнтській стороні, зменшуючи навантаження на сервер.

Модель "підписки/видачі": Клієнти підписуються на події, а сервер повідомляє про їх настання. Реалізується через шаблон "спостерігач", що спрощує оновлення даних, але збільшує навантаження на сервер.

3-рівнева структура:

1. **Клієнтська частина** — відповідає за інтерфейс і взаємодію з користувачем.
2. **Загальна частина (middleware)** — містить спільні класи та логіку.
3. **Серверна частина** — реалізує бізнес-логіку, управління даними та їх збереження.

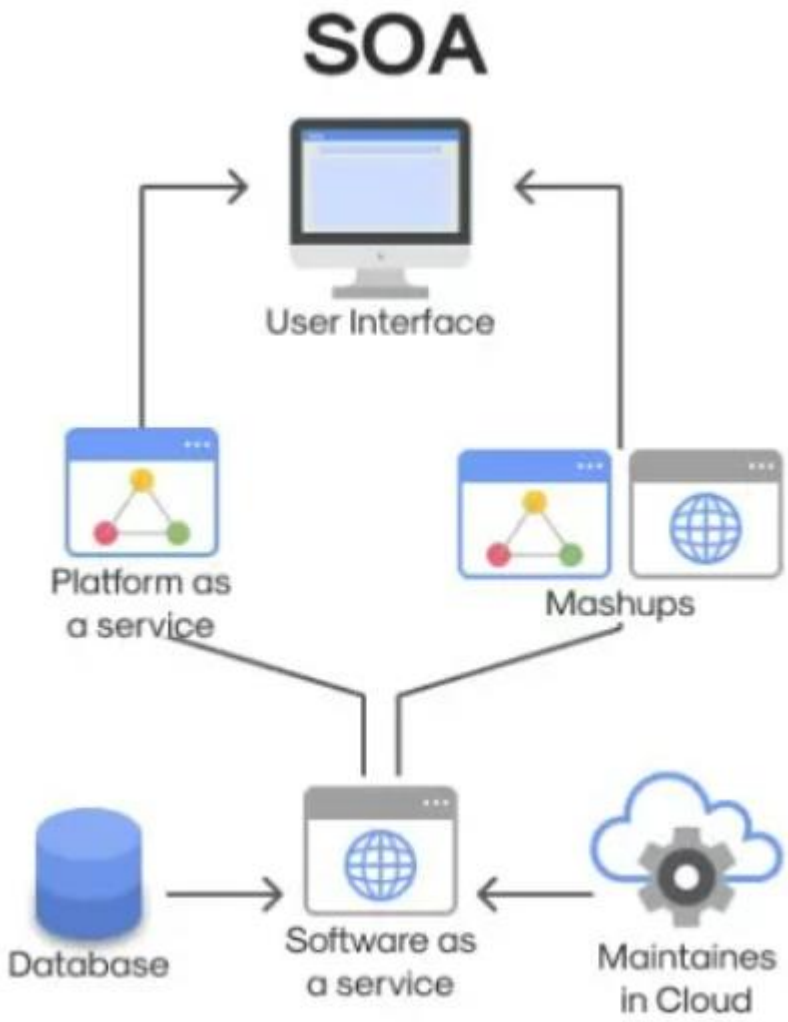
Додатки типу «peer-to-peer»



Peer-to-peer (P2P) додатки організовують рівноправну взаємодію між клієнтами без використання серверів. Всі клієнти обмінюються даними напряму для досягнення спільних цілей. Основними викликами є синхронізація даних та пошук інших клієнтів.

Для пошуку використовується або загальна адреса з переліком підключених клієнтів, або адреси зберігаються у самих клієнтів (структуровані однорангові мережі). Синхронізація даних може здійснюватися через алгоритми порівняння (наприклад, hash) чи узгодження набору для синхронізації. P2P-додатки базуються на спеціальних форматах і протоколах обміну для забезпечення ефективної взаємодії між клієнтами.

Сервіс-орієнтована архітектура (SOA)



Сервіс-орієнтована архітектура (SOA) — це модульний підхід до розробки програмного забезпечення, що базується на використанні розподілених і слабо пов'язаних компонентів зі стандартизованими інтерфейсами.

Системи SOA реалізуються через набір веб-служб, які взаємодіють за протоколами SOAP, REST або іншими. Компоненти SOA ізолюють деталі реалізації, що забезпечує масштабованість, незалежність від платформ і повторне використання.

Модель SaaS є прикладом SOA, де додатки надаються користувачам через Інтернет як послуга. Ключові переваги SaaS: відсутність витрат на обслуговування, оновлення програм, гнучка оплата, і прозорість модернізації.

Мікро-сервісна архітектура

Мікро-сервісна архітектура — це підхід до створення серверних додатків як набору незалежних малих служб, кожна з яких виконується у власному

процесі та взаємодіє з іншими через протоколи, такі як HTTP, WebSockets, або AMQP.

Кожна мікрослужба реалізує конкретну бізнес-логіку в обмеженому контексті, розробляється автономно й розгортається незалежно.

Переваги мікро-сервісної архітектури:

- **Гнучкість і масштабованість**: легко адаптується до змін і підтримує високі навантаження.
- **Простота супроводу**: зручне обслуговування навіть у великих комплексних системах.
- **Автономність компонентів**: кожна служба має свій життєвий цикл і не залежить від інших.

2. Реалізація частини функціоналу робочої програми у вигляді класів і їх взаємодій для досягнення конкретних функціональних можливостей.

Серверна частина (Server-side):

org.ftp.command

Цей пакет містить логіку для обробки команд FTP-сервера, застосовуючи патерн **Command**:

- **builder**: Містить класи для створення та конфігурації команд.
- **container**: Забезпечує управління збереженням команд (наприклад, пул доступних команд).
- **impl**: Реалізація обробки команд:
    - **CommandName**: Перелік команд (наприклад, LIST, RETR, STOR).
    - **CommandProcessor**: Відповідає за обробку конкретної команди.
    - **CommandResponse**: Клас для формування відповіді на виконану команду.

org.ftp.config

Містить класи для налаштувань серверної частини:

- **DatabaseConfig**: Конфігурація бази даних для збереження інформації про користувачів, сесії та інші дані.

org.ftp.domain

Пакет для опису основних сутностей системи, наприклад, інформація про користувача, сесію або файли.

org.ftp.exception

Містить кастомні винятки для обробки специфічних помилок FTP-сервера, таких як відсутність доступу або неправильні команди.

org.ftp.memento

Реалізація патерна **Memento**, яка забезпечує збереження стану сесій або налаштувань для їхнього відновлення в майбутньому.

org.ftp.repository

Містить класи для взаємодії з базою даних:

- **impl**:
    - **Repository**: Базовий інтерфейс для роботи з базою даних.
    - **RepositoryFactory**: Фабрика для створення конкретних репозиторіїв.

org.ftp.service

Містить класи та інтерфейси, які реалізують бізнес-логіку FTP-сервера:

- **PasswordUtils**: Утилітарний клас для хешування та перевірки паролів.

org.ftp.visitor

Реалізація патерна **Visitor** для виконання специфічних операцій над об'єктами:

- **PermissionCheckVisitor**: Перевірка прав доступу для користувача.
- **Visitable**: Інтерфейс, який повинні реалізовувати класи, що підтримують патерн Visitor.
- **Visitor**: Базовий інтерфейс для реалізації відвідувачів.

Головні класи в кореневому пакеті:

- **FtpServer**: Основний клас, що запускає сервер та обробляє з'єднання.
- **LoggedState** та **NotLoggedState**: Реалізація патерна **State** для управління станами сесії (авторизований чи ні).
- **SessionState**: Інтерфейс для станів сесії.

- **StartupInitializer**: Відповідає за ініціалізацію конфігурацій при старті сервера.
- **UserSession**: Клас, який містить інформацію про поточну сесію користувача.

Структура серверної частини представлена на малюнку 2.1

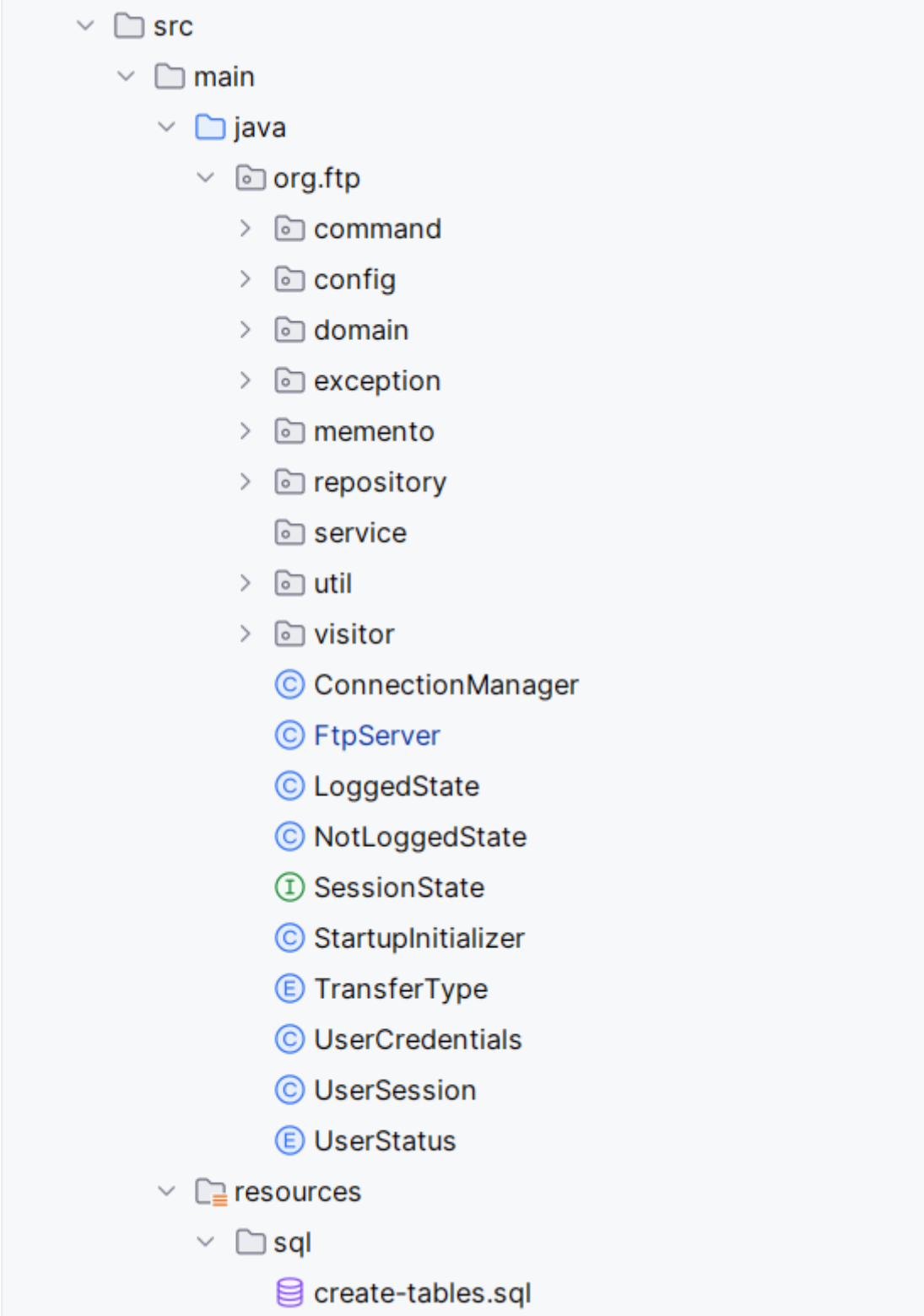


Рисунок 2.1 – Структура серверної частини

Клієнтська частина (Client-side):

FTP-клієнт, написаний на Java, призначений для взаємодії з FTP-сервером через консольний інтерфейс. Цей застосунок дозволяє:

1. Підключатися до FTP-сервера за вказаною IP-адресою та портом.
2. Виконувати автентифікацію користувача за допомогою команд USER та PASS.
3. Використовувати команди для передачі файлів:
    - **STOR**: Завантаження локальних файлів на сервер.
    - **RETR**: Завантаження файлів із сервера на локальний комп'ютер.
4. Використовувати режими з'єднання:
    - **PASV (пасивний режим)**.
    - **PORT (активний режим)**.
5. Вводити та виконувати стандартні FTP-команди (наприклад, LIST, QUIT).
6. Обробляти помилки автентифікації, з'єднання та доступу до файлів.
7. Забезпечувати перевірку форматів IP-адрес, портів, та правильність команд.

Особливості:

- Підтримка авторизації.
- Логіка перевірки доступу до директорій та файлів.
- Простий і зручний консольний інтерфейс для взаємодії з сервером.

Застосунок забезпечує обробку помилок і зручний формат виведення повідомлень для користувача.

Структура клієнтської частини представлена на малюнку 2.2

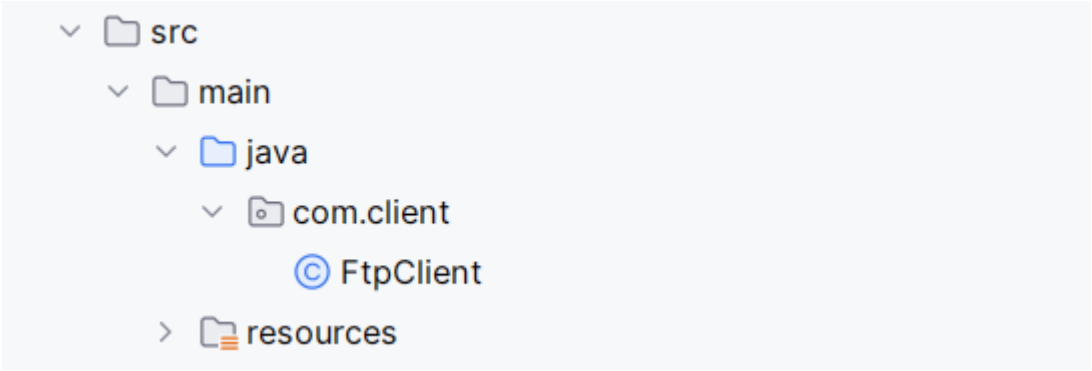


Рисунок 2.2 – Структура клієнтської частини

3. Реалізація взаємодії програми в одній з архітектур відповідно до обраної теми.

Клієнт-серверна архітектура

Основні етапи клієнт серверної архітектури:

- **З'єднання:** Клієнти з'єднуються з FTP-сервером через мережу, використовуючи протокол FTP. З'єднання встановлюється на визначеному порту (типово 21) із підтримкою авторизації через ім'я користувача та пароль.
- **Відправлення команди:** Клієнтська програма (FTP-клієнт) формує команди (наприклад, LIST, RETR, STOR) відповідно до протоколу FTP та відправляє їх на сервер для виконання певної операції.
- **Обробка сервером:** Сервер приймає команду, інтерпретує її, виконує необхідну логіку (таку як перевірка прав доступу, маніпуляції з файлами чи директоріями, доступ до бази даних) та формує відповідь відповідно до специфікацій FTP-протоколу.
- **Відправлення відповіді:** Сервер відправляє клієнту відповідь, яка може містити повідомлення про успіх операції, дані (наприклад, список файлів) або код помилки в разі некоректного виконання команди.
- **Обробка відповіді клієнтом:** Клієнтська програма отримує відповідь від сервера, обробляє її та відображає користувачеві результати (наприклад, список файлів у директорії або повідомлення про завершення завантаження файлу).

На рисунку 3.1 зображена діаграма клієнт-серверної взаємодії

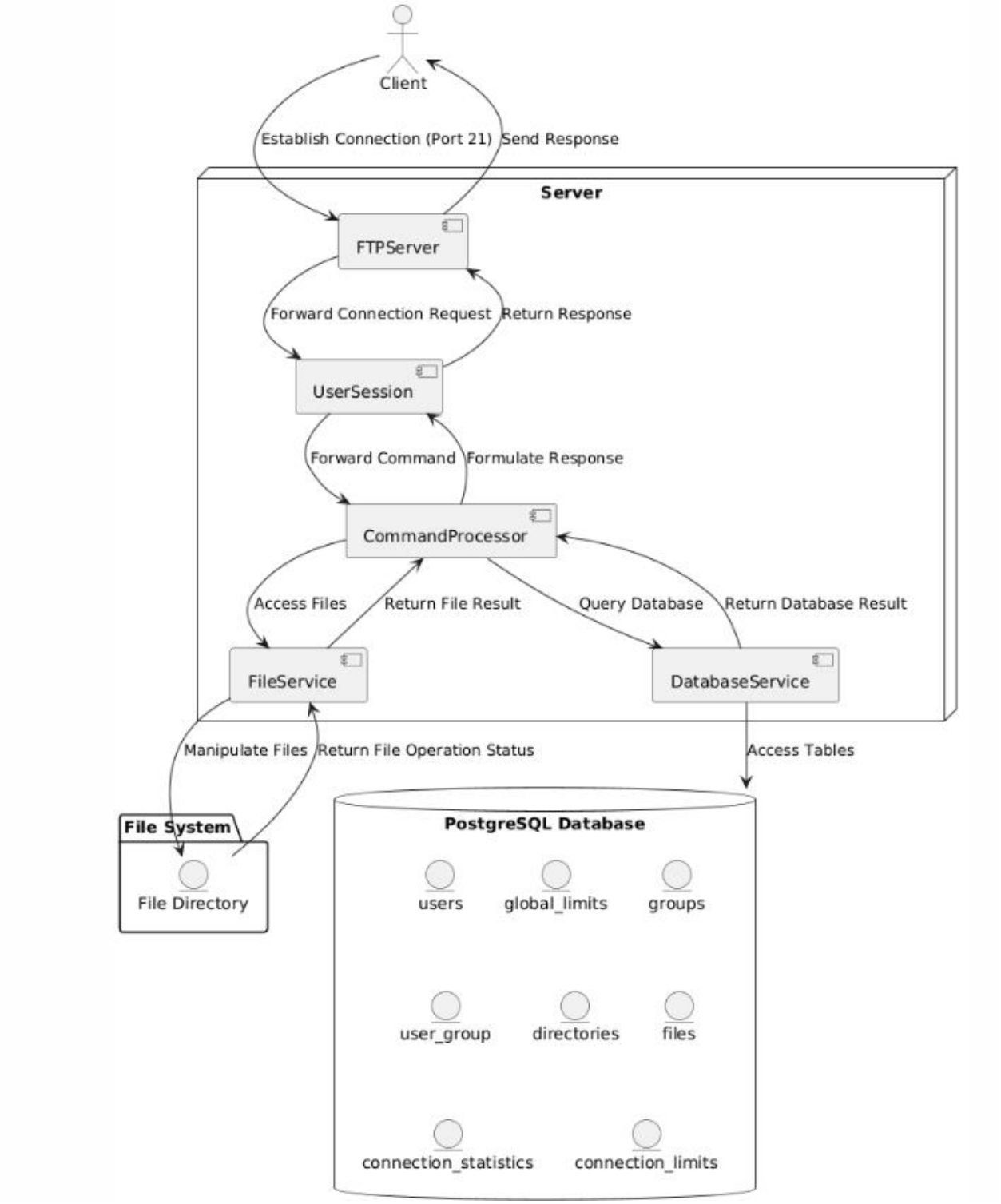


Рисунок 3 – діаграма клієнт-серверної взаємодії

На рисунку 3.2 зображено приклад роботи взаємодії з сервером зі сторони клієнта

Рисунок 3.2 взаємодії з сервером зі сторони клієнта

Висновок

Код можна знайти за посиланням - https://github.com/B1lok/trpz

У ході виконання лабораторної роботи було розглянуто три основні архітектурні шаблони взаємодії додатків: Client-Server, Peer-to-Peer (P2P) та

Service-Oriented Architecture (SOA). Було вивчено їхні особливості, переваги, недоліки та сфери застосування.

На практиці було реалізовано функціонал програми з використанням одного з розглянутих шаблонів, а саме Client-Server. Було продемонстровано, як клієнтська частина ініціює з’єднання з сервером, надсилає запити та отримує відповіді. Реалізація функціоналу включала взаємодію сервера з базою даних і файловою системою для зберігання та обробки інформації, що ілюструє ключові аспекти цієї архітектури.