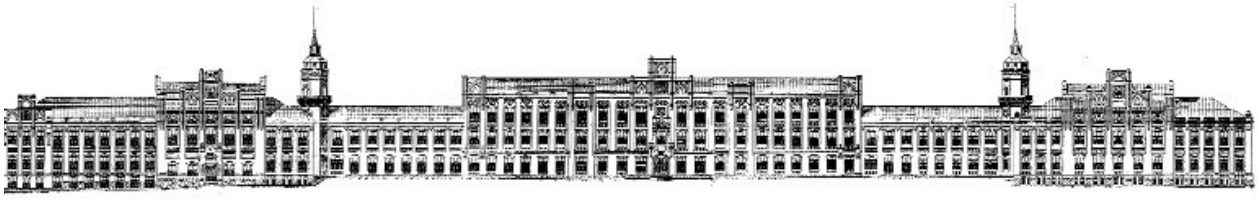

Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №9
Тема: «Взаємодія компонентів системи»

Виконала
студентка групи ІА-32:
Іванова Анастасія
Юріївна

Перевірив:
Мягкий Михайло
Юрійович

Зміст

Тема проекту

Варіант: 26
Опис теми: Download manager (iterator, command, observer, template method, composite, p2p) Інструмент для скачування файлів з інтернету по протоколах http або https з можливістю продовження завантаження в зупиненому місці, розподілу швидкостей активним завантаженням, ведення статистики завантажень, інтеграції в основні браузери (firefox, opera, internet explorer, chrome).

Теоретичні відомості

Архітектура Peer-to-Peer (P2P) є децентралізованою моделлю організації розподілених систем, у якій кожен вузол мережі одночасно виконує роль і клієнта, і сервера. На відміну від класичної клієнт-серверної архітектури, де існує центральний сервер, що керує обміном даних, у P2P-взаємодії управління та відповідальність розподіляються між усіма учасниками системи. Завдяки цьому мережа стає більш стійкою до збоїв, оскільки відсутня єдина точка відмови. Децентралізація також сприяє підвищенню надійності та забезпечує кращу масштабованість.
Ключовою особливістю архітектури P2P є рівноправність усіх вузлів. Кожен учасник може як запитувати інформацію, так і надавати її іншим, а ролі вузлів не є фіксованими й можуть змінюватися залежно від навантаження чи конкретної задачі. Це дозволяє ефективно розподіляти ресурси між багатьма користувачами, що є важливою перевагою при роботі з великими обсягами даних або при значному числі підключень. Учасники мережі можуть спільно надавати доступ до файлів, дискового простору, обчислювальної потужності та інших ресурсів, що робить архітектуру гнучкою та адаптивною.
Архітектура P2P знайшла широке застосування у сучасних інформаційних системах. До найпоширеніших прикладів належать файлообмінні протоколи, криптовалюти та блокчейн-технології, системи інтернет-телефонії та відеоконференцій, а також платформи для розподілених обчислень. У таких системах розподіл навантаження між вузлами дозволяє уникати перевантаження серверів та забезпечує швидку та надійну роботу сервісів незалежно від кількості користувачів.
Разом із перевагами архітектура P2P має й певні обмеження. Однією з основних проблем є забезпечення безпеки, оскільки відсутність централізованого контролю ускладнює верифікацію та захист даних. Крім того, складність становить синхронізація інформації між вузлами, оскільки дані можуть бути розподілені неоднорідно. Ще однією проблемою є пошук ресурсів у великій кількості вузлів без централізованого індексу, що потребує спеціальних алгоритмів маршрутизації та оптимізації. Незважаючи на ці недоліки,

архітектура P2P залишається ефективним рішенням для побудови масштабованих і відмовостійких систем.

Хід роботи

У межах лабораторної роботи необхідно було додати до мого застосунку підтримку Peer-to-Peer взаємодії між двома незалежними екземплярами програми. Архітектурно це рішення було реалізоване шляхом розширення існуючої моделі MVC та додаванням спеціальних компонентів P2P-рівня, які забезпечують обмін списками завантажень між вузлами.

Компоненти P2P-рівня:

PeerController:
Це REST-контролер, який відповідає за надання списку локальних завантажень для зовнішніх вузлів. При зверненні до нього інший вузол отримує перелік локальних завантажень у форматі DTO.

Фрагмент коду:

```
public class PeerController {

   private final DownloadRepo downloads;

   @GetMapping("/downloads")
   public List<PeerDownloadDto> getDownloads() {
      return downloads.findAll().stream()
            .map(d -> new PeerDownloadDto(
                  d.getId(),
                  d.getFileName(),
                  d.getTotalBytes(),
                  d.getUrl()
            ))
            .toList();
   }
}
```

Роль:
бути «точкою входу» для зовнішніх вузлів;
забезпечувати стандартизовану відповідь у форматі JSON;
ізолювати внутрішню структуру застосунку від сторонніх клієнтів.

PeerClient:
Це клієнтська частина P2P-системи. PeerClient виконує HTTP-запити до іншого вузла, використовуючи RestTemplate, та отримує список завантажень того вузла.

Фрагмент коду:

```
public class PeerClient {

   private final RestTemplate rest = new RestTemplate();

   public List<PeerDownloadDto> fetchDownloads(String host) {
      String url = "http://" + host + "/peer/downloads";
      PeerDownloadDto[] arr = rest.getForObject(url, PeerDownloadDto[].class);
      return arr != null ? Arrays.asList(arr) : List.of();
   }
}
```

Роль:
забезпечити ініціацію P2P-комунікації та отримання даних від іншого вузла.

DownloadController (оновлений):
У контролері, який відповідає за роботу зі списком завантажень, було додано новий обробник.
Під час виклику цього методу користувач передає адресу іншого вузла, наприклад localhost:8081.
DownloadController:

- звертається до PeerClient і отримує список завантажень вузла B;
- для кожного отриманого об’єкта створює новий запис у власному сховищі DownloadService;
- перенаправляє користувача назад на сторінку зі списком завантажень.

Фрагмент коду:

```
@PostMapping("/import")
   public String importFromPeer(
         @RequestParam String peer,
         @AuthenticationPrincipal UserDetails auth
   ) {
      var user = users.findByUsername(auth.getUsername()).orElseThrow();

      var remoteDownloads = peerClient.fetchDownloads(peer);

      for (var rd : remoteDownloads) {
         downloadService.addUrl(user.getId(), rd.url());
      }

      return "redirect:/downloads";
   }
```

Роль:
забезпечити зв’язок між UI та P2P-клієнтом;

виконувати логіку імпорту;
контролювати валідацію та формат введеного адресного рядка.

Опис роботи архітектури:

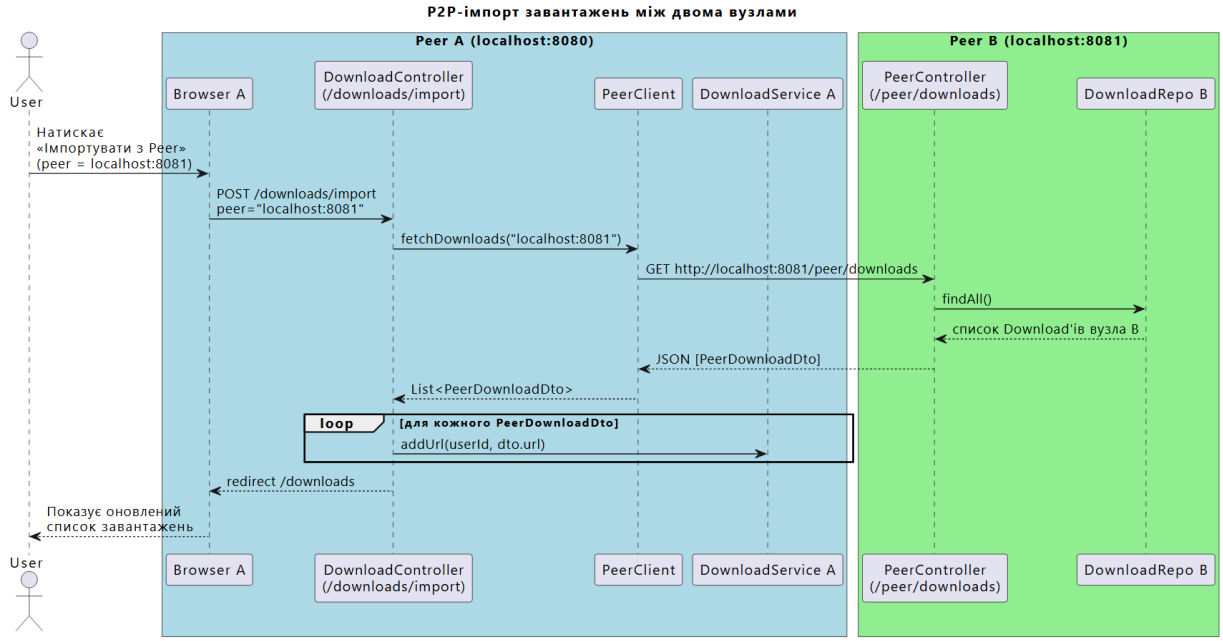

Рис 1. Діаграма процесу обміну

На діаграмі зображено процес взаємодії двох однорангових вузлів у моєму застосунку під час імпорту завантажень з одного вузла на інший. Взаємодія починається з того, що користувач на вузлі A натискає кнопку «Імпортувати з Peer» та вказує адресу іншого вузла. Після цього браузер надсилає HTTP-запит на маршрут /downloads/import, який обробляється контролером завантажень на локальному вузлі. Саме цей контролер ініціює процес обміну даними. Контролер звертається до допоміжного компонента PeerClient, який відповідає за мережеву взаємодію між вузлами. PeerClient формує запит до іншого вузла системи за відкритою адресою /peer/downloads, доступною без автентифікації. У відповідь вузол B приймає цей запит у своєму PeerController. Контролер звертається до локального репозиторію завантажень, отримує повний список своїх записів і перетворює їх у формат DTO. Після цього список передається назад вузлу A у вигляді JSON-масиву.
Отримавши JSON-відповідь, PeerClient перетворює її у список об’єктів PeerDownloadDto та повертає контролеру завантажень вузла A. Після цього запускається цикл, у якому кожен елемент, отриманий від вузла B, імпортується у локальну систему. Для цього викликається метод addUrl сервісу завантажень, що створює відповідні записи у базі даних вузла A. Таким чином, усі завантаження з іншого вузла відтворюються у локальному середовищі. Після завершення імпорту контролер перенаправляє користувача на сторінку списку завантажень. Браузер оновлює інтерфейс, і користувач бачить розширений перелік, який тепер включає елементи, отримані з вузла B. Взаємодія завершується відображенням оновленого списку.

Представлений сценарій показує, як здійснюється передача інформації між незалежними вузлами, які не мають спільної бази даних і взаємодіють виключно через REST-інтерфейс.

Реалізація у застосунку:

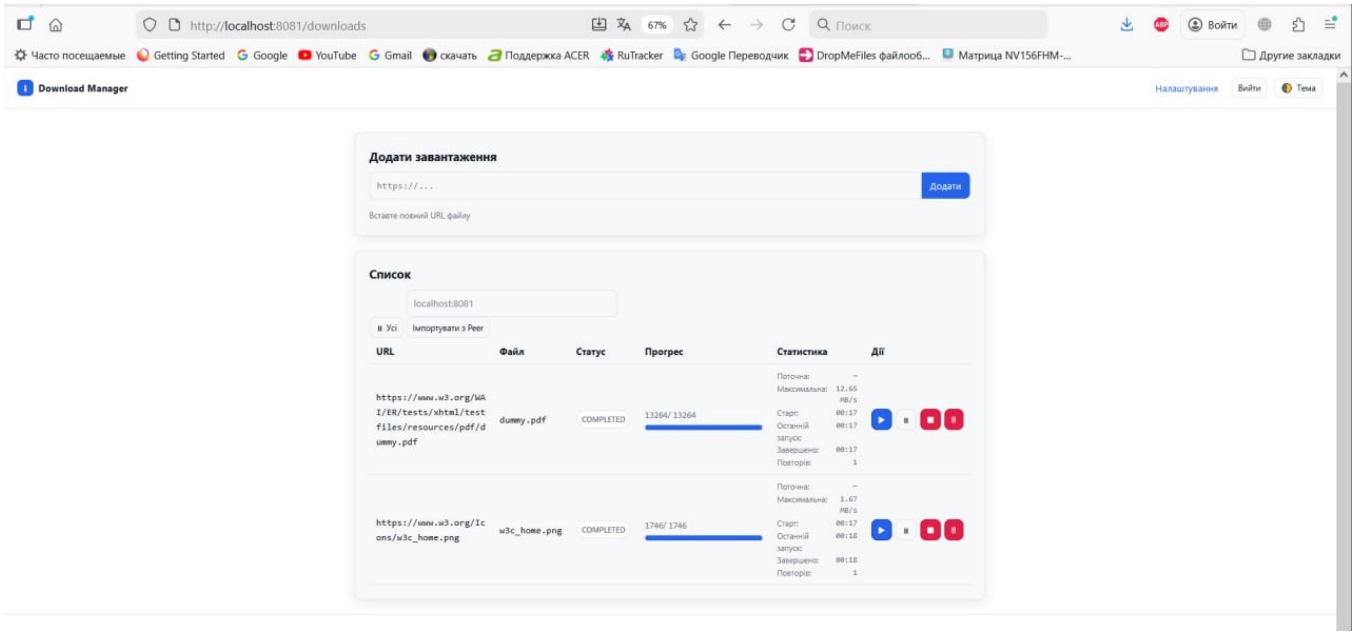

Рис 1. Сторінка з завантаженнями на вузлі B (8081).

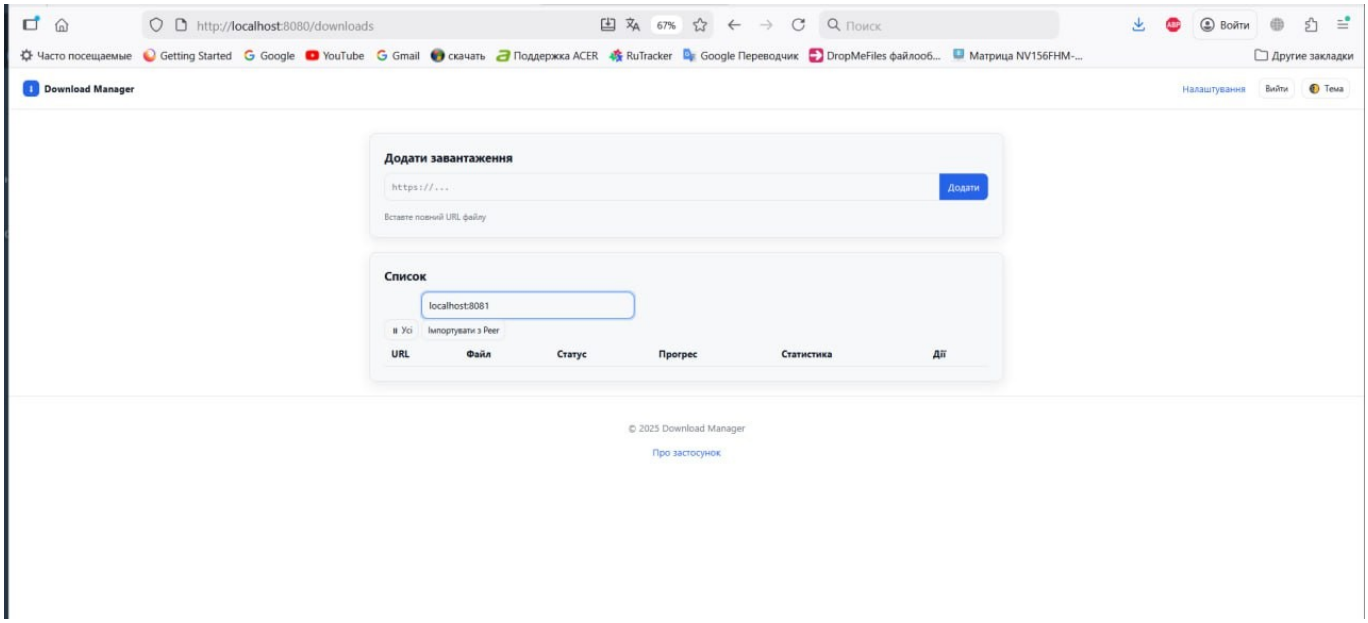

Рис 2. Сторінка завантажень на стороні A (8080) до передачі

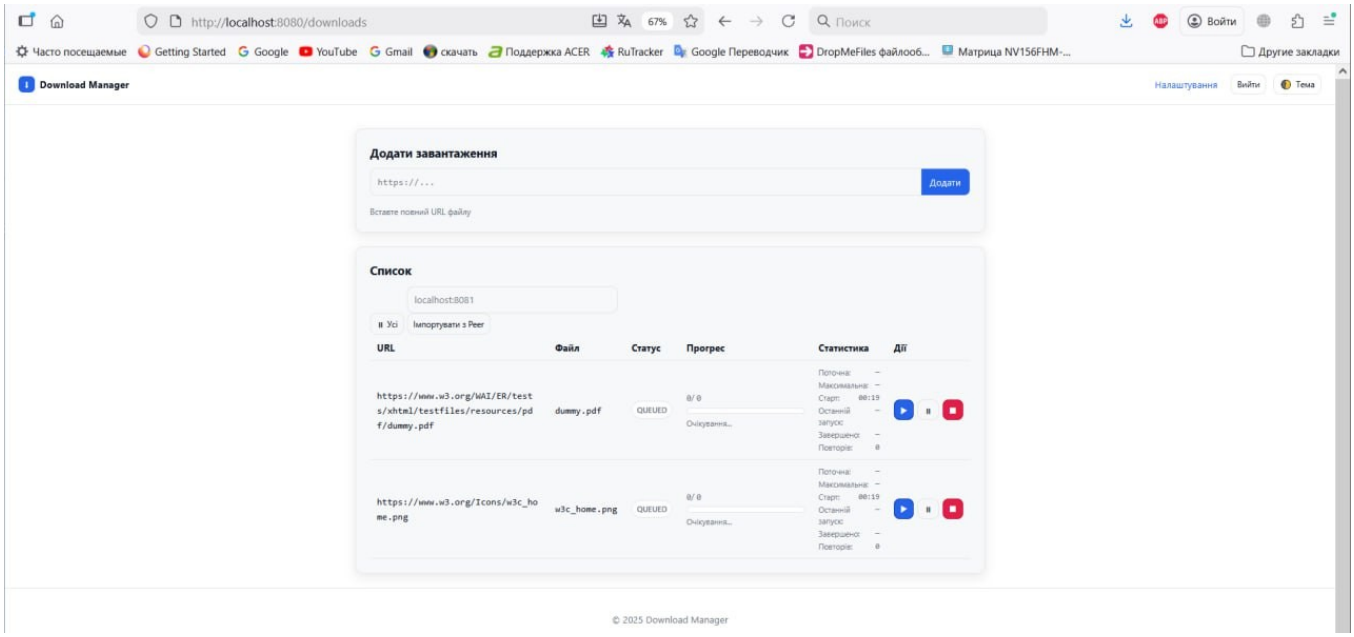

Рис 3. Сторінка завантажень на стороні A (8080) після передачі

Висновки:

У рамках цієї лабораторної роботи в застосунку було реалізовано підтримку однорангової (P2P) взаємодії між двома (можна більше) незалежними вузлами системи. На відміну від класичного підходу «клієнт–сервер», де один центральний сервер обслуговує всі запити, у P2P-моделі кожен екземпляр програми одночасно може виступати як клієнтом, так і сервером. У моїй реалізації кожен вузол самостійно зберігає дані про власні завантаження та надає мінімальний REST-інтерфейс для обміну інформацією з іншими вузлами. Кожен вузол працює в ізольованому середовищі, має власну базу даних і власний життєвий цикл.

P2P-обмін у моєму застосунку реалізовано у вигляді невеликого функційного «мікропротоколу», що складається з відкритого ендпойнту /peer/downloads і клієнтського компонента PeerClient, який виконує HTTP-запити до іншого вузла. Через те, що кожен вузол запускається на окремому порту, система легко масштабується навіть у локальному середовищі, дозволяючи емулювати взаємодію кількох інстансів програми.