

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
із дисципліни «*Технології розробки програмного забезпечення*»
Тема: «*Взаємодія компонентів системи*»

Виконав:

Студент групи IA-34
Ястремський Богдан

Перевірив:

асистент кафедри ІСТ
Мягкий Михайло Юрійович

Тема: Взаємодія компонентів системи.

Мета: Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Service-oriented Architecture), та реалізувати в проєктованій системі одну із архітектур..

Застосунок (№6): Web-browser (proxy, chain of responsibility, factory method, template method, visitor, p2p).

Веб-браузер повинен мати можливість зробити наступне: мати адресний рядок для введення адреси сайту, переміщатися і відображати структур html документа, переглядати підключений javascript та css файли, перегляд всіх підключених ресурсів (зображень), коректна обробка відповідей з сервера (коди відповідей HTTP) – переходи при перенаправленнях, відображення сторінок 404 і 502/503.

Короткі теоретичні відомості:

Клієнт-серверна архітектура

Клієнт-серверні додатки — це розподілені додатки, де є два основні типи: клієнти, які представляють програму користувачеві, та сервери, що зберігають і обробляють дані. Існують два варіанти клієнтів:

1. Тонкий клієнт: Більшість операцій виконується на сервері, а клієнт лише відображає дані. Це знижує навантаження на клієнт, але все навантаження лягає на сервер. Приклад — класичні Web-застосунки.
2. Товстий клієнт: Більшість логіки обробки даних відбувається на стороні клієнта, що зменшує навантаження на сервер. Сервер виступає лише як доступ до інших ресурсів. Приклад — мобільні та десктопні застосунки (Evernote, Viber).

Проміжний варіант — SPA (Single Page Application), де більшість логіки працює на клієнті, але збереження даних потребує доступу до сервера.

Клієнт-серверна взаємодія зазвичай організована в три рівні:

- Клієнтська частина: Відповідає за візуальне відображення та логіку взаємодії з користувачем.

- Загальна частина (middleware): Містить спільні дані та класи.
- Серверна частина: Відповідає за бізнес-логіку, зберігання та обмін даними.

Ця архітектура допомагає ефективно організувати розподілені додатки з різними типами клієнтів.

P2P-архітектура

Peer-to-Peer (P2P) архітектура — це модель мережової взаємодії, де кожен вузол (комп'ютер або пристрій) є одночасно клієнтом і сервером. Всі вузли мають рівні права для обміну даними, ресурсами та виконання завдань. На відміну від клієнт-серверної моделі, де є чітке розділення між клієнтами та серверами, P2P дозволяє учасникам взаємодіяти без централізованого сервера.

Основні принципи P2P:

- Децентралізація: Відсутність центрального сервера, що знижує залежність від одного вузла і підвищує стійкість до збоїв і атак.
- Рівноправність вузлів: Кожен вузол може бути одночасно клієнтом і сервером.
- Розподіл ресурсів: Вузли надають ресурси, такі як обчислювальна потужність, дисковий простір або файли.

P2P застосовується в:

- Файлообмінниках (BitTorrent).
- Криптовалютах та блокчайн-технологіях.
- Інтернет-телефонії та відеоконференціях (Skype, Zoom).
- Розподілених обчисленнях (SETI@home, BOINC).

Основні проблеми P2P:

- Безпека.
- Синхронізація даних.
- Пошук ресурсів: ефективність пошуку знижується зі збільшенням кількості вузлів, і для покращення результату потрібні спеціальні алгоритми.

Сервіс-орієнтована архітектура

Сервіс-орієнтована архітектура (SOA) — модульний підхід до розробки ПЗ, заснований на використанні розподілених, слабо пов'язаних сервісів з стандартизованими інтерфейсами для взаємодії через стандартизовані протоколи.

Історично SOA з'явилася як альтернатива монолітним архітектурам, де вся система розроблялася як одне ціле.

Основні характеристики SOA:

- Модульність: Програмні комплекси реалізуються як набір веб-сервісів, які взаємодіють по HTTP через SOAP або REST.
- Лоуз-капаулінг (слабке зв'язування): Сервіси взаємодіють через обмін повідомленнями, без спеціальних інтеграцій для доступу до однієї бази даних.
- Обгортки для застарілих систем: Сервіси можуть бути обгортками для старих систем, що дозволяє знизити вартість їх переробки та полегшити інтеграцію з новими системами.

Сервіси реєструються на спеціальних сервісах, що дозволяє розробникам знаходити та використовувати їх. Часто для обміну даними використовується централізований компонент — шина даних (Enterprise Service Bus).

Мікросервісна архітектура є подальшим розвитком SOA, що включає нові технологічні досягнення в інформаційних технологіях.

Мікросервісна архітектура

Мікросервісна архітектура — підхід до створення серверного додатку як набору малих, автономних служб, орієнтованих на конкретні можливості і бізнес-логіку. Кожна служба працює у своєму процесі та взаємодіє з іншими через протоколи, такі як HTTP/HTTPS, WebSockets або AMQP.

Основні характеристики мікросервісів:

- Автономність: Кожен мікросервіс розробляється і розгортається незалежно, забезпечуючи чітко визначені межі і бізнес-логіку в рамках конкретного обмеженого контексту.
- Взаємодія через повідомлення: Мікросервіси взаємодіють за допомогою повідомень між собою.
- Масштабованість: Мікросервіси сприяють високій масштабованості та можливості супроводження великих комплексних систем завдяки незалежності їх життєвих циклів.

Мікросервіси є основою для створення гнучких і легко розвиваючих систем, що складаються з незалежно розгортаючих служб, що дає можливість ефективного управління і масштабування в складних середовищах.

Хід роботи:

1. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

Я реалізував клас PeerInfo, в якому збираю інформацію про піра: ідентифікатор, ім'я, IP, порт, онлайн/неонлайн, останній раз коли був побачений.

Також я створив клас BrowsingHistoryEntry, в якому збираю інформацію про історію пошуку, якою буду ділитися з пірами (посилання, назва, дата візиту, час візиту)

Аналогічно, є клас P2PMessage, який зберігає словник можливих повідомлень, які будуть відображатися. Клас містить тип повідомлення, інформацію про відправника, вкладення, часову мітку.

Крім того, є інтерфейс P2PMessageListener, який містить метод onMessageReceived, який викликається при отриманні повідомлення.

Найголовніший клас – P2PNode, він і є клієнтом, і сервером. І він містить інформацію про: ідентифікатор вузла, ім'я вузла, порт, підключені піри, їх з'єднання, пул, обробники повідомлень і чи запустився сервер. Клас містить методи старту серверу, прийняття і обробки з'єднань, під'єднання до піра і його прослуховування, відправка повідомлень і бродкаст, видалення піра, перевірка наявності, зупинки і повідомлення слухачів.

Також я оновив інтерфейс браузера. Він містить поля для підключення пірів (за localhost ip, але порт будь-який вільний), кнопку з'єднання, окремий sidebar для мережі, а саме:

- Підключені піри
- Лог активності
- Чат
- К-сть підключених пірів.

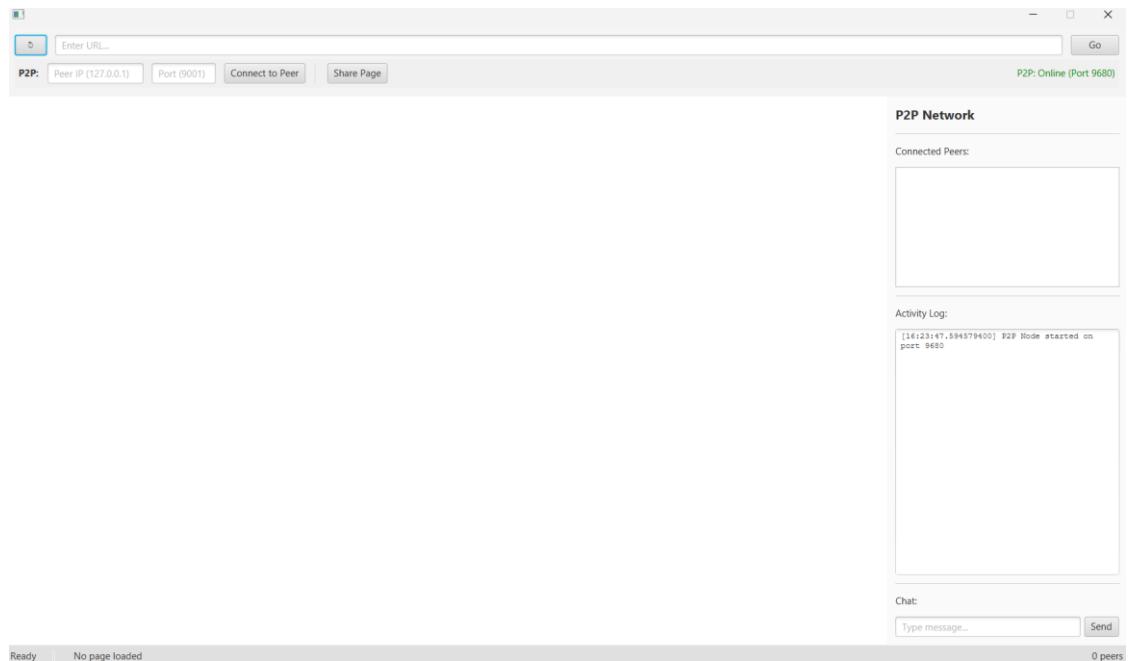


Рис. 9.1 – UI застосунку

Контролер отримав методи ініціалізації мережі, обробки повідомлень, підключення, відключення, спільної історії пошуку.

2. Реалізувати функціонал для роботи в розподіленому оточенні відповідно до обраної теми.

Тут буде продемонстровано роботу мережі.

- Відкриємо два екземпляри застосунку

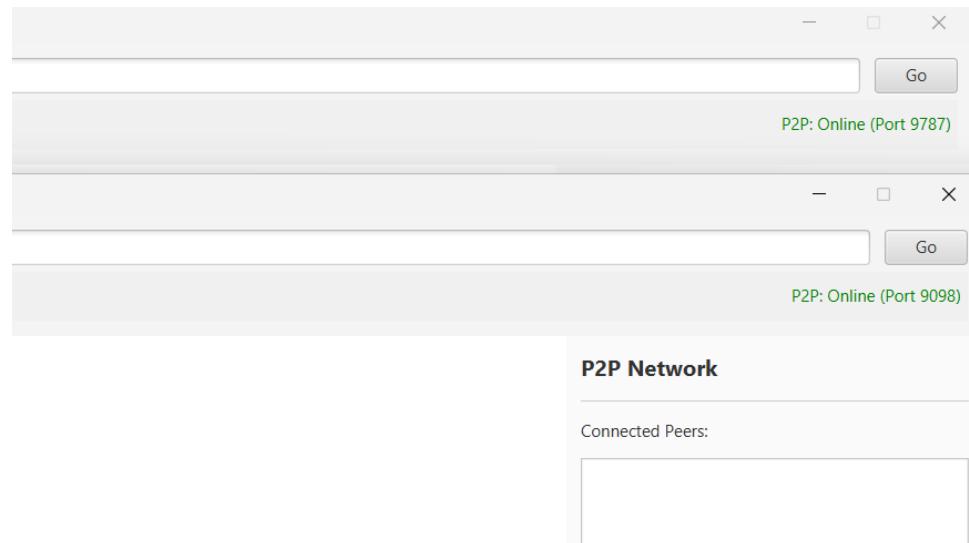


Рис. 9.2 – Два екземпляри на вільних портах 9787 та 9098

- Під'єднаємо їх один до одного:

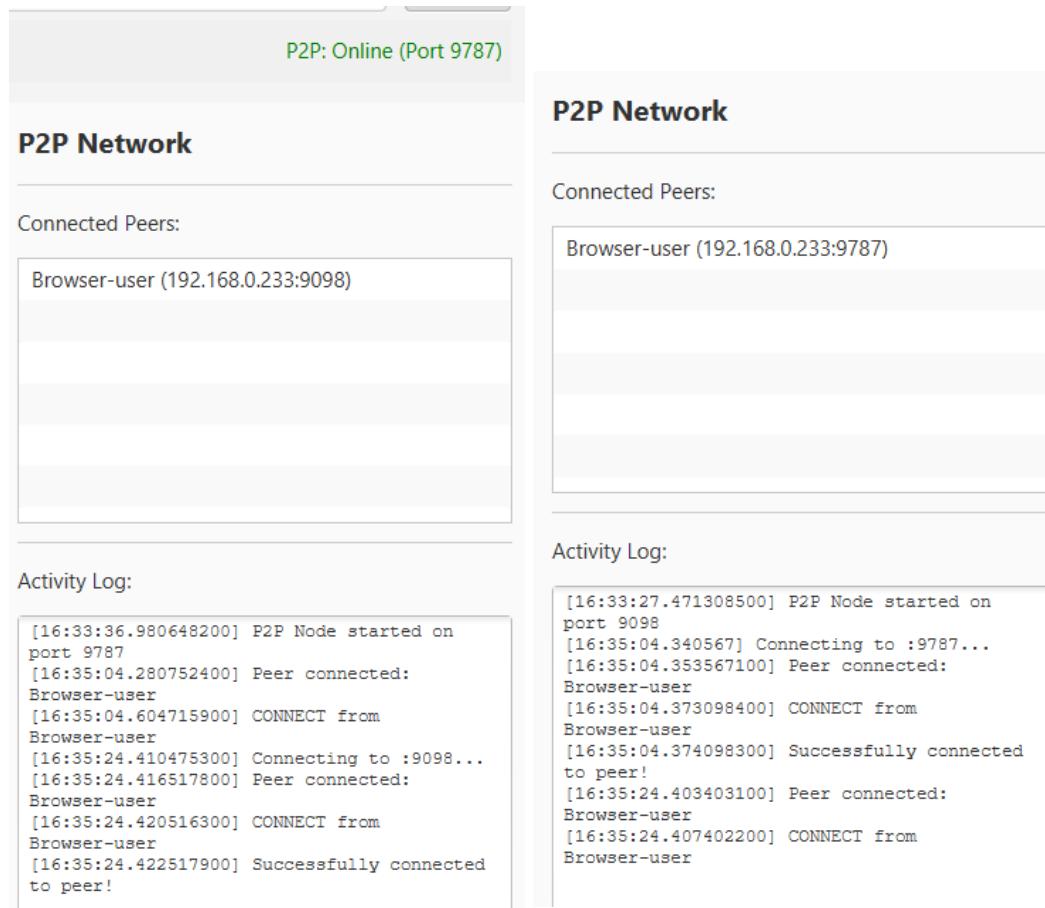


Рис. 9.3 – Вигляди сайдбарів обох застосунків.

- Використаємо чат:

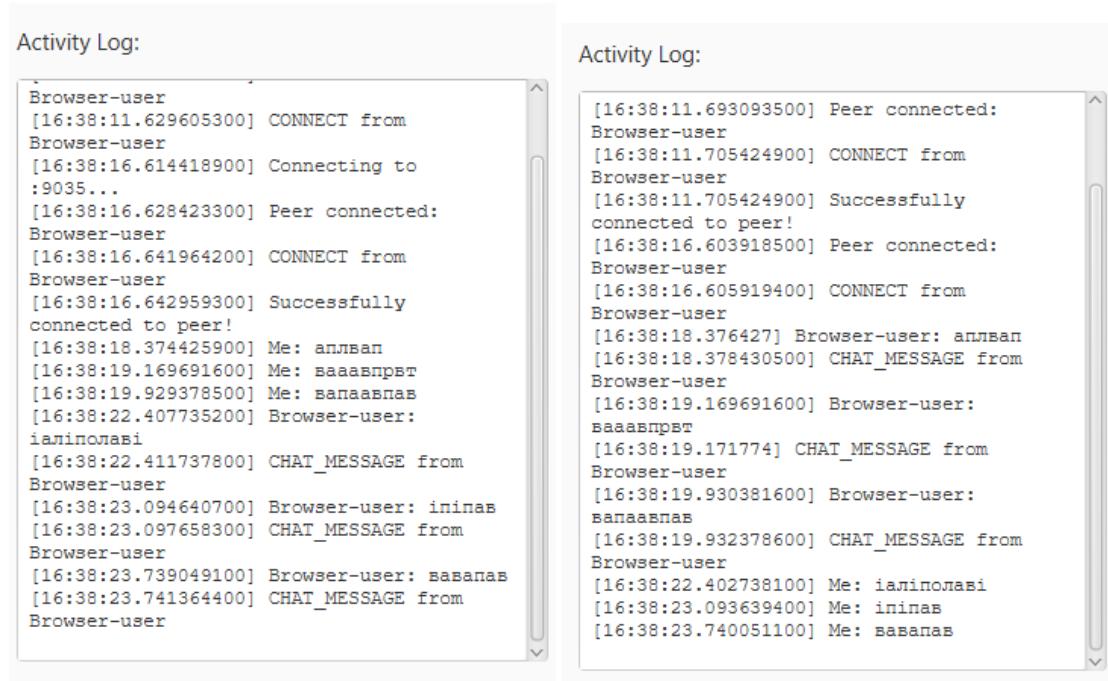


Рис. 9.4 – Вигляди сайдбарів при обміні повідомленнями (були використані інші порти для тесту)

- Використаємо обмін сторінками:

Activity Log: <pre>[16:42:30.853165600] P2P Node started on port 9012 [16:42:38.354836300] Connecting to :9012... [16:42:38.364836900] Peer connected: Browser-user [16:42:38.377839700] CONNECT from Browser-user [16:42:38.379836500] Successfully connected to peer! [16:42:44.697406700] Peer connected: Browser-user [16:42:44.701410500] CONNECT from Browser-user [16:43:01.449225100] Shared history: https://nv.ua from Browser-user [16:43:01.450223500] SHARE_HISTORY from Browser-user [16:43:08.906539200] Browser-user shared: https://nv.ua [16:43:08.910694] SHARE_PAGE from Browser-user [16:43:29.216030600] Broadcasted current page to all peers</pre>	<pre>[16:42:16.928488100] P2P Node started on port 9012 [16:42:38.306177800] Peer connected: Browser-user [16:42:38.318206] CONNECT from Browser-user [16:42:44.704404] Connecting to :9853... [16:42:44.708664800] Peer connected: Browser-user [16:42:44.711978600] CONNECT from Browser-user [16:42:44.711978600] Successfully connected to peer! [16:43:08.888012800] Broadcasted current page to all peers [16:43:27.645440] Shared history: https://example.com from Browser-user [16:43:27.647438700] SHARE_HISTORY from Browser-user [16:43:29.240662100] Browser-user shared: https://example.com [16:43:29.243677] SHARE_PAGE from Browser-user</pre>
---	---

Рис. 9.5— Вигляди сайдбарів при обміні сторінками (були використані інші порти для тесту)

Висновок: на даному лабораторному занятті я познайомився з тим, як взаємодіють додатки, і реалізував однорангову мережу p2p у своєму браузері. Велика перевага такої мережі в тому, що мені не довелося створювати окремий сервер чи наділяти якогось певного піра правами адміністратора – така архітектура не містить центрального сервера, а кожен вузол є рівноправним, і може як приймати, так і надсилю повідомлення (тобто виступає гібридом клієнта і сервера). Також я з легкістю можу масштабувати мережу просто під'єднавшись до іншого піра без суттєвих змін в коді.

Відповіді на контрольні питання:

1. Що таке клієнт-серверна архітектура?

Клієнт-серверні додатки — це розподілені додатки, де є два основні типи: клієнти, які представляють програму користувачеві, та сервери, що зберігають і обробляють дані.

2. Розкажіть про сервіс-орієнтовану архітектуру.

Сервіс-орієнтована архітектура (SOA) — модульний підхід до розробки ПЗ, заснований на використанні розподілених, слабо пов'язаних сервісів з стандартизованими інтерфейсами для взаємодії через стандартизовані протоколи.

3. Якими принципами керується SOA?

- Модульність: Програмні комплекси реалізуються як набір веб-сервісів, які взаємодіють по HTTP через SOAP або REST.
- Лоуз-капаулінг (слабке зв'язування): Сервіси взаємодіють через обмін повідомленнями, без спеціальних інтеграцій для доступу до однієї бази даних.
- Обгортки для застарілих систем: Сервіси можуть бути обгортками для старих систем, що дозволяє знизити вартість їх переробки та полегшити інтеграцію з новими системами.

4. Як між собою взаємодіють сервіси в SOA?

Лоуз-капаулінг (слабке зв'язування): Сервіси взаємодіють через обмін повідомленнями, без спеціальних інтеграцій для доступу до однієї бази даних.

5. Як розробники знають про існуючі сервіси і як робити до них запити?

Сервіси реєструються на спеціальних сервісах, що дозволяє розробникам знаходити та використовувати їх. Часто для обміну даними використовується централізований компонент — шина даних (Enterprise Service Bus).

6. У чому полягають переваги та недоліки клієнт-серверної моделі?

Тонкі клієнти:

- Переваги: централізоване управління оновленнями, простота розгортання, низькі вимоги до клієнтської частини.
- Недоліки: високе навантаження на сервери, потреба у стабільному та швидкому з'єднанні з сервером.

Товсті клієнти:

- Переваги: зменшене навантаження на сервер, клієнт може працювати автономно.
- Недоліки: складність оновлення, оскільки кожен клієнт потребує індивідуальних оновлень, важчий для масштабування.

SPA:

- Переваги: зменшене навантаження на сервер, легше оновлення, зручність використання.
- Недоліки: вимагає постійного доступу до сервера для нормальної роботи.

7. У чому полягають переваги та недоліки однорангової моделі взаємодії?

Переваги:

- Децентралізація – відсутність центрального сервера, що зменшує залежність від одного вузла, підвищуючи стійкість мережі до збоїв і атак.
- Рівноправність вузлів – кожен вузол може виконувати одночасно функції клієнта (отримувати ресурси) і сервера (надавати ресурси).
- Розподіл ресурсів – вузли надають доступ до своїх власних ресурсів, таких як обчислювальна потужність, дисковий простір або файли

Недоліки:

- До основних проблемних зон можна віднести безпеку, синхронізацію даних та пошук ресурсів.
- Через централізацію складно контролювати дані, які передаються.
- Ефективність пошуку даних знижується зі збільшенням кількості вузлів у мережі і для підвищення ефективності пошуку потрібно застосовувати спеціальні алгоритми.

8. Що таке мікро-сервісна архітектура?

Мікросервісна архітектура — підхід до створення серверного додатку як набору малих, автономних служб, орієнтованих на конкретні можливості і бізнес-логіку.

9. Які протоколи використовуються для обміну даними в мікросервісній архітектурі?

Кожна служба працює у своєму процесі та взаємодіє з іншими через протоколи, такі як HTTP/HTTPS, WebSockets або AMQP.

10. Чи можна назвати підхід сервіс-орієнтованою архітектурою, коли ми в проекті між шаром веб-контролерів та шаром доступу до даних реалізуємо шар бізнес-логіки у вигляді сервісів?

Так, цей підхід можна вважати сервіс-орієнтованою архітектурою, оскільки бізнес-логіка реалізована у вигляді незалежних сервісів, які взаємодіють між собою та з іншими шарами додатку. SOA передбачає розбиття додатку на окремі сервіси, які можуть бути масштабованими, повторно використовуваними та незалежними.