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

«Взаємодія компонентів системи»

Виконав
студент групи ІА–34:
Бородай А.С.

Київ 2025

Зміст

Теоретичні відомості 3
Клієнт-серверна архітектура 3
Peer-to-Peer архітектура 3
Сервіс-орієнтована архітектура 4
Мікро-сервісна архітектура 4
Хід роботи 6
Component Diagram (SOA) 7
Висновок 10

Теоретичні відомості

Клієнт-серверна архітектура

Модель розподіленої системи, в якій існує чітке розділення між клієнтами (які ініціюють запити) та серверами (які їх обробляють та надають ресурси або послуги). Існують дві основні моделі клієнтів:

Тонкий клієнт: виконує мінімум логіки, в основному відображає інтерфейс, а обробка даних відбувається на сервері. Це спрощує розгортання та оновлення системи.

Товстий клієнт: значна частина логіки реалізована на стороні клієнта, що розвантажує сервер та дозволяє працювати офлайн. Сервер у цьому випадку часто виступає як точка доступу до даних або зв'язку між клієнтами.

Взаємодія в клієнт-серверних системах часто будується за трирівневою структурою: клієнтський рівень, серверний рівень та спільний рівень.

.

Peer-to-Peer архітектура

Peer-to-Peer (P2P) архітектура – це модель мережевої взаємодії, в якій кожен вузол є одночасно клієнтом і сервером. У цій архітектурі всі вузли мають рівні права та можливості для обміну даними, ресурсами або виконання завдань. P2P-мережа дозволяє учасникам взаємодіяти безпосередньо, без необхідності в централізованому сервері. Основними принципами P2P-архітектури є: децентралізація, рівноправність вузлів, розподіл ресурсів.

Основними сферами є файлообмінники, криптовалюти та інші блокчейн-технології, інтернет телефонія та відеоконференції, розподілені обчислення.

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.

Сервіс-орієнтована архітектура

Підхід до розробки програм, заснований на використанні незалежних, слабкозв’язаних сервісів, які взаємодіють через стандартні інтерфейси та протоколи (наприклад, HTTP з SOAP або REST). Кожен сервіс інкапсулює певну бізнес-функцію та може бути використаний повторно. SOA виникла як альтернатива монолітним системам і часто реалізується через веб-служби. Для спрощення взаємодії сервісів може використовуватися шина даних, а самі сервіси реєструються в спеціальних реєстрах для пошуку та використання.

Мікро-сервісна архітектура

Мікросервісна архітектура це підхід до розробки програмного забезпечення, при якому серверний додаток будується як сукупність невеликих автономних служб. Кожна така служба, що називається мікросервісом, працює у власному процесі та відповідає за окрему бізнес-функцію в рамках чітко визначеного контексту. Взаємодія між окремими мікросервісами відбувається через легкі протоколи, такі як HTTP чи AMQP, зазвичай з використанням повідомлень.

Ключовою характеристикою архітектури є принцип незалежного розгортання: кожен мікросервіс розробляється, впроваджується та масштабується окремо від інших. Така модульність забезпечує високу гнучкість, сприяє швидкій розробці та полегшує супровід складних великомасштабних систем. Завдяки автономності життєвих циклів окремих компонентів система стає стійкішою до змін і легше адаптується до вимог масштабування. Цей підхід є логічним розвитком ідей сервіс-орієнтованої архітектури, зосередженим на максимальній декомпозиції та незалежності служб..

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.

Завдання:

• Ознайомитись з короткими теоретичними відомостями.

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

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

• Реалізувати взаємодію розподілених частин:

– Для клієнт-серверних варіантів: реалізація клієнтської і серверної частини додатків, а також загальної частини (middleware); зв'язок клієнтської і серверної частин за допомогою WCF, TcpClient, .NET-Remoting на розсуд виконавця.

– Для однорангових мереж: реалізація взаємодії клієнтських додатків за допомогою WCF Peer to peer channel.

– Для SOA додатків: реалізація сервісу, що надає послуги клієнтським застосуванням; викладання сервісу в хмару або підняття у вигляді Web Service на локальній машині; використання токенів для передачі даних про автентифікації, двостороннє шифрування.

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє спроєктовану архітектуру. Навести фрагменти програмного коду, які є суттєвими для відображення реалізованої архітектури.

Хід роботи

20. **Mind-mapping software**

(strategy, prototype, abstract factory, bridge, composite, SOA)

Візуальний додаток для складання "карт пам'яті" з можливістю роботи з декількома картами (у вкладках), автоматичного промальовування ліній, додавання вкладених файлів, картинок, відеофайлів (попередній перегляд); можливість додавання значків категорій / терміновості, обведення областей карти (поділ пунктирною лінією).

Результат

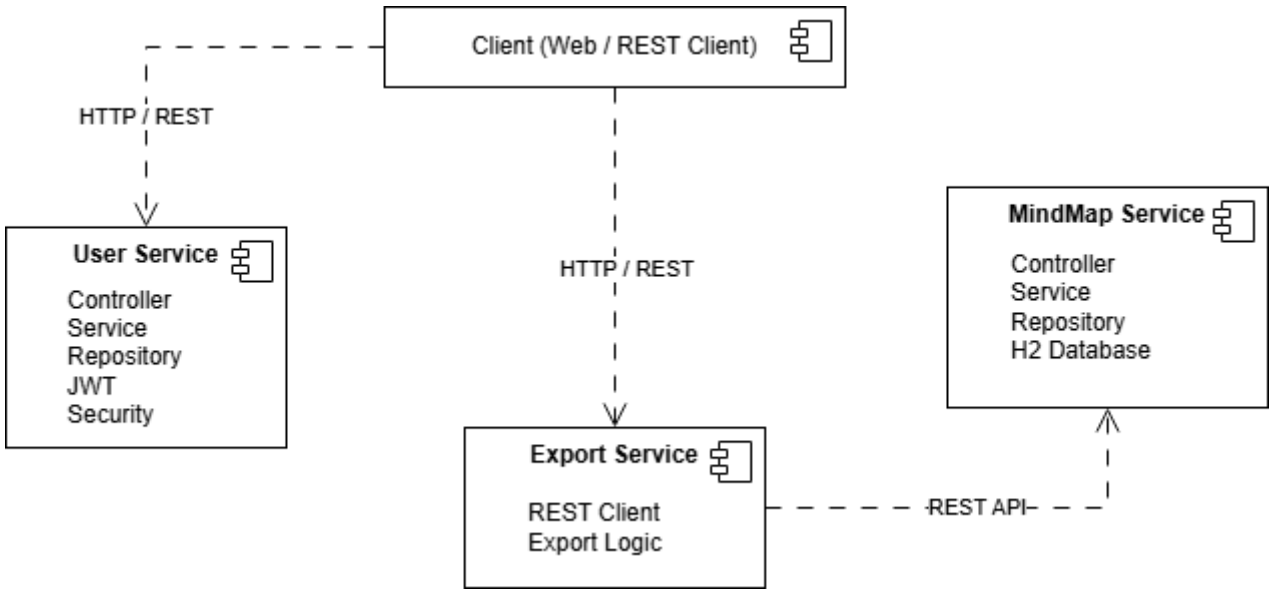
У межах виконання роботи було реалізовано сервісно-орієнтовану архітектуру для роботи з ментальними картами. Система складається з трьох автономних сервісів, кожен з яких виконує чітко визначену функцію.

User Service відповідає за управління користувачами системи. У межах цього сервісу реалізовано операції створення, перегляду, оновлення та видалення користувачів, а також механізми автентифікації та авторизації. Для контролю доступу використовується JWT-токен, який генерується після успішної перевірки облікових даних користувача.

MindMap Service призначений для роботи з ментальними картами як графовими структурами. Сервіс забезпечує збереження та обробку ментальних карт, їх нод та ребер у спільній базі даних. Даний сервіс виконує роль внутрішнього бізнес-сервісу та не надає прямого доступу клієнту.

Export Service відповідає за завантаження ментальних карт у різних форматах. Сервіс отримує дані ментальних карт через REST-виклики до MindMap Service та виконує їх перетворення у текстові формати TXT, CSV та JSON. Export Service виступає єдиною точкою доступу клієнта до функціональності експорту.

Component Diagram (SOA)



У реалізованій сервісно-орієнтованій архітектурі прямий зв’язок між клієнтським застосунком та сервісом керування ментальними картами (MindMap Service) відсутній. Це є свідомим архітектурним рішенням, яке відповідає принципам SOA, зокрема принципам слабкої зв’язаності та інкапсуляції бізнес-логіки.

Взаємодія клієнта з даними ментальних карт здійснюється через Export Service, який виконує функцію фасаду. Export Service інкапсулює логіку отримання даних з MindMap Service, їх агрегацію та перетворення у необхідні формати (TXT, CSV, JSON), забезпечуючи єдину точку доступу для клієнта.

Такий підхід дозволяє зменшити залежність клієнта від внутрішньої структури системи, підвищити модульність архітектури та спростити подальший розвиток і масштабування сервісів без впливу на клієнтську частину.

У кожному сервісі використовується типовий набір шарів, характерний для Spring Boot застосунків:

- Controller – приймає HTTP-запити та повертає відповіді клієнту;
- Service – містить бізнес-логіку сервісу;
- Repository – забезпечує доступ до бази даних через JPA;
- Entity – описує структуру даних, що зберігаються у базі;
- DTO – використовується для передачі даних між сервісами.

Оскільки ці типи класів мають однакове призначення у всіх сервісах, їх функціональність описується один раз.

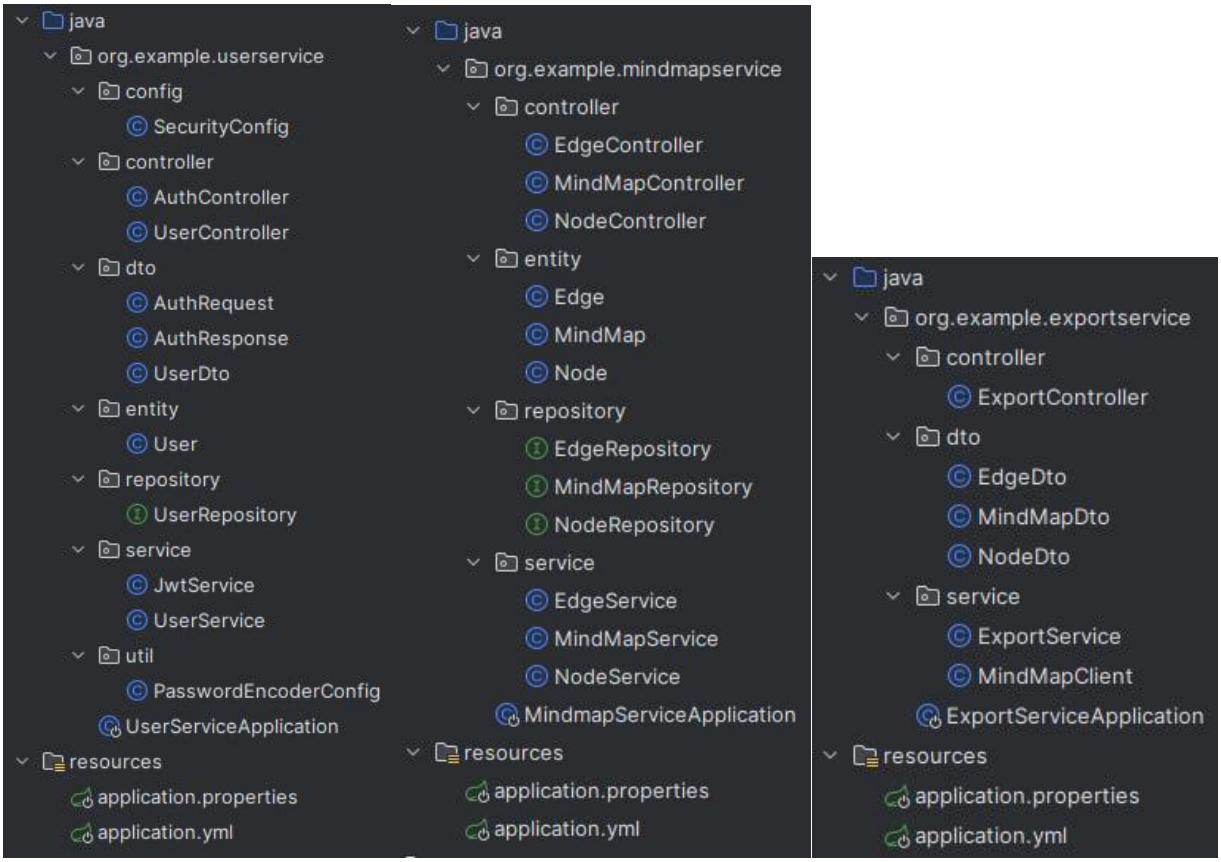


Рисунок 1.1-1.3 Вигляд створених файлів для кожного сервісу.

SecurityConfig

AuthController

UserController

AuthRequest

AuthResponse

UserDto

User

UserRepository

JwtService

UserService

PasswordEncoderConfig

UserServiceApplication

EdgeController

MindMapController

NodeController

Edge

MindMap

Node

EdgeRepository

MindMapRepository

NodeRepository

EdgeService

MindMapService

NodeService

MindmapServiceApplication

ExportController

EdgeDto

MindMapDto

NodeDto

ExportService

MindMapClient

ExportServiceApplication

Взаємодія між сервісами відбувається виключно через REST-інтерфейси відповідно до принципів SOA.

На першому етапі користувач проходить автентифікацію в User Service та отримує JWT-токен. Надалі клієнт використовує цей токен для доступу до функціональності експорту ментальних карт.

При виконанні запиту на експорт клієнт звертається до Export Service, який, у свою чергу, надсилає REST-запит до MindMap Service для отримання даних конкретної ментальної карти. MindMap Service завантажує дані карти, нод та ребер з бази даних і повертає їх Export Service. Після цього Export Service виконує перетворення отриманих даних у вибраний формат та повертає результат клієнту.

Прямий доступ клієнта до MindMap Service відсутній, що дозволяє інкапсулювати внутрішню структуру системи та забезпечити слабку зв’язаність між компонентами.

Висновок

Під час виконання лабораторної роботи я вивчив основні види взаємодії додатків (Client-Server, Peer-to-Peer, Service-oriented Architecture) та реалізував в проєктованій системі архітектуру SOA.

Реалізована система відповідає принципам сервісно-орієнтованої архітектури, забезпечує чіткий розподіл відповідальності між сервісами та демонструє коректну взаємодію автономних компонентів через REST-інтерфейси.

Відповіді на питання для самоперевірки

1. Що таке клієнт-серверна архітектура?

Архітектура розподілених систем, де існує чітке розділення ролей: клієнти ініціюють запити, а сервери їх обробляють та надають ресурси чи послуги. Існують тонкі клієнти (обробка на сервері) та товсті клієнти (значна частина логіки на стороні клієнта).

2. Розкажіть про сервіс-орієнтовану архітектуру.

Це модульний підхід до розробки, заснований на використанні розподілених, слабкозв’язаних сервісів, що надають певні бізнес-функції. Сервіси взаємодіють через стандартизовані інтерфейси та протоколи. Це альтернатива монолітним системам, що дозволяє реалізувати можливості як набір веб-служб.

3. Якими принципами керується SOA?

Модульність, слабке зв’язування, стандартизована взаємодія через інтерфейси та можливість повторного використання сервісів. Сервіси розробляються автономно та інкапсулюють конкретну бізнес-логіку.

4. Як між собою взаємодіють сервіси в SOA?

Сервіси взаємодіють виключно через обмін повідомленнями, використовуючи стандартизовані протоколи, такі як HTTP з SOAP або REST. Вони не розділяють спільну базу даних напряму, що забезпечує їхню незалежність.

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

Сервіси зазвичай реєструються в спеціальних реєстрах або каталогах. Розробники можуть знайти потрібний сервіс у такому реєстрі, отримати доступ до його стандартизованого інтерфейсу та викликати його функції через визначений протокол.

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

Переваги: централізоване управління даними, безпека, легша підтримка і оновлення (особливо для тонких клієнтів).

Недоліки: залежність від доступності сервера, можливі вузькі місця та обмеження масштабування через централізацію.

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

Переваги: децентралізація, відмовостійкість, відсутність єдиної точки відмови, ефективне використання ресурсів мережі.

Недоліки: складність забезпечення безпеки, проблеми з синхронізацією даних, зниження ефективності пошуку ресурсів у великих мережах.

8. Що таке мікро-сервісна архітектура?

Це підхід до створення серверного додатку як набору малих, незалежних служб. Кожен мікросервіс виконує окрему бізнес-функцію, працює у власному процесі та розгортається автономно, взаємодіючи з іншими через легкі протоколи.

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

Для обміну даними використовуються легкі протоколи, такі як HTTP/HTTPS, WebSockets або AMQP, часто з використанням механізмів обміну повідомленнями.

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

Ні, у SOA сервіси є слабкозв’язаними, автономними компонентами, що взаємодіють через стандартні інтерфейси, часто між різними системами. Реалізація

шару бізнес-логіки у вигляді внутрішніх сервісів не відповідає основним принципам SOA, таким як незалежне розгортання та взаємодія через мережу.