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

Лабораторна робота №9
«Технології розроблення програмного забезпечення»
Тема: « E-mail клієнт»

Виконав:
Студент групи
ІА-34
Марченко А.О.
Перевірив:
Мягкий Михайло Юрійович

Київ 2025

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.

15. **E-mail клієнт** (singleton, builder, decorator, template method, interpreter, client-server)

Поштовий клієнт повинен нагадувати функціонал поштових програм Mozilla Thunderbird, The Bat і т.д. Він повинен сприймати і коректно обробляти pop3/smtp/imap протоколи, мати функції автонастройки основних поштових провайдерів для України (gmail, ukr.net, i.ua), розділяти повідомлення на папки/категорії/важливість, зберігати чернетки незавершених повідомлень, прикріплювати і обробляти прикріплені файли.

Теоритичні відомості

Клієнт-серверна архітектура

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.

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. Прикладом тонкого клієнта є класичні Web-застосунки.

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. Перевагою такого підходу є менші вимоги до серверної частини. Також перевагою, при певному підході до реалізації, є можливість працювати клієнтам без тимчасового доступу до серверу. Прикладом товстого клієнта можна назвати мобільні застосунки, або десктоп застосунки. Наприклад, Evernote, Viber, MS Outlook, комп’ютерні антивіруси, ігри, що потребують інсталяції (The Sims, GTA, …) та інші.

Проміжним варіантом можна назвати SPA (Single Page Application) – це

товсті Web-клієнти, які при старті кожен раз завантажуються з сервера, а надалі працюють з сервером через web-API

Peer-to-Peer архітектура

Peer-to-Peer (P2P) архітектура – це модель мережевої взаємодії, в якій кожен вузол (комп'ютер або пристрій) є одночасно клієнтом і сервером. У цій архітектурі всі вузли мають рівні права та можливості для обміну даними, ресурсами або виконання завдань. На відміну від клієнт-серверної моделі, де є чітке розділення на клієнти й сервери, P2P-мережа дозволяє учасникам взаємодіяти безпосередньо, без необхідності в централізованому сервері.

Основними принципами P2P-архітектури є:

- 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.

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) сервісів або служб, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами.

Історично сервіс-орієнтована архітектура появилася як альтернатива монолітній архітектурі, в які вся система розроблялася та розгорталася як одне ціле.

Згідно SOA сервіси реєструються на спеціальних сервісах і будь-яка команда розробників, якій потрібен доступ може знайти їх та використовувати.

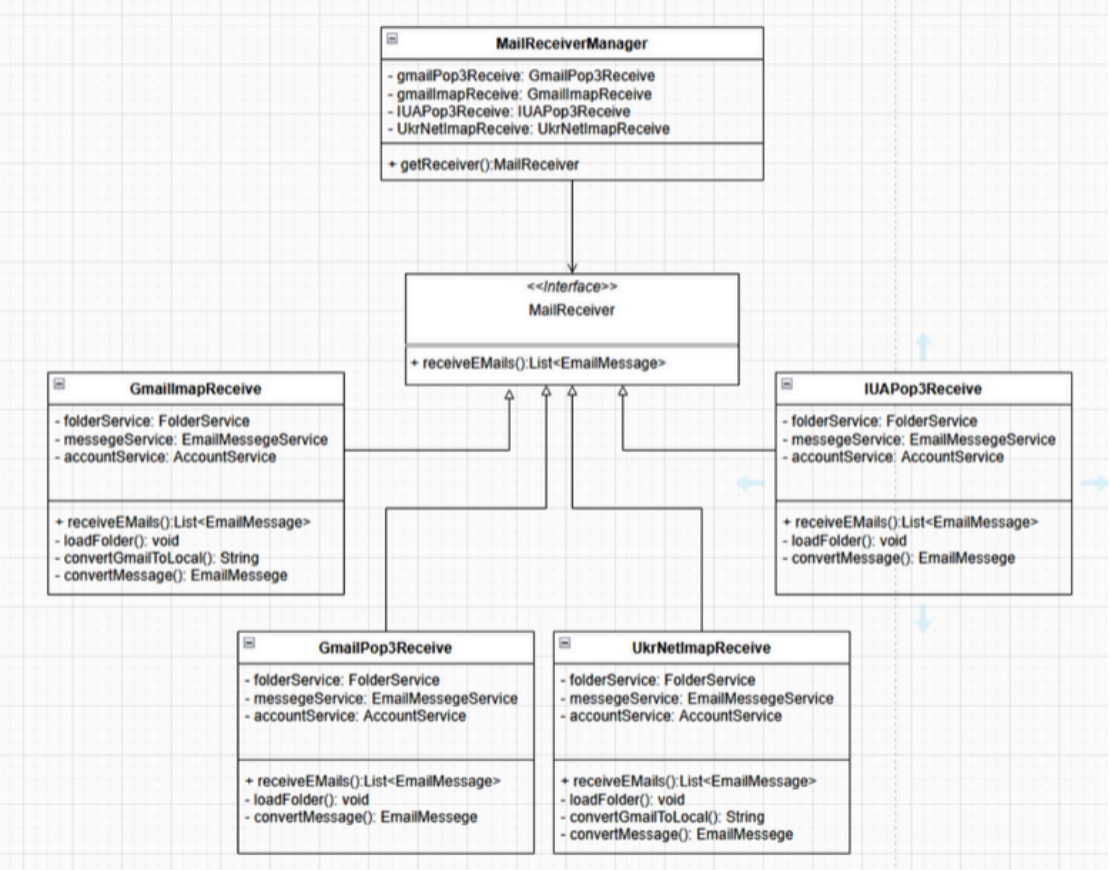
Часто реалізація SOA покладається на використання централізованого програмного компонента для обміну даними – шину даних (Enterprise Service Bus).

Мікросервісна архітектура є подальшим розвитком сервіс-орієнтованої архітектури з використанням нових напрацювань у інформаційних технологіях.

Мікро-сервісна архітектура

Сама назва дає зрозуміти, що мікро-сервісна архітектура є підходом до створення серверного додатку як набору малих служб. Це означає, що архітектура мікро-сервісів 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. Кожен мікросервіс реалізує специфічні можливості в предметній області і свою бізнес-логіку в рамках конкретного обмеженого контексту, повинна розроблятись автономно і розвертатись незалежно.

Хід роботи



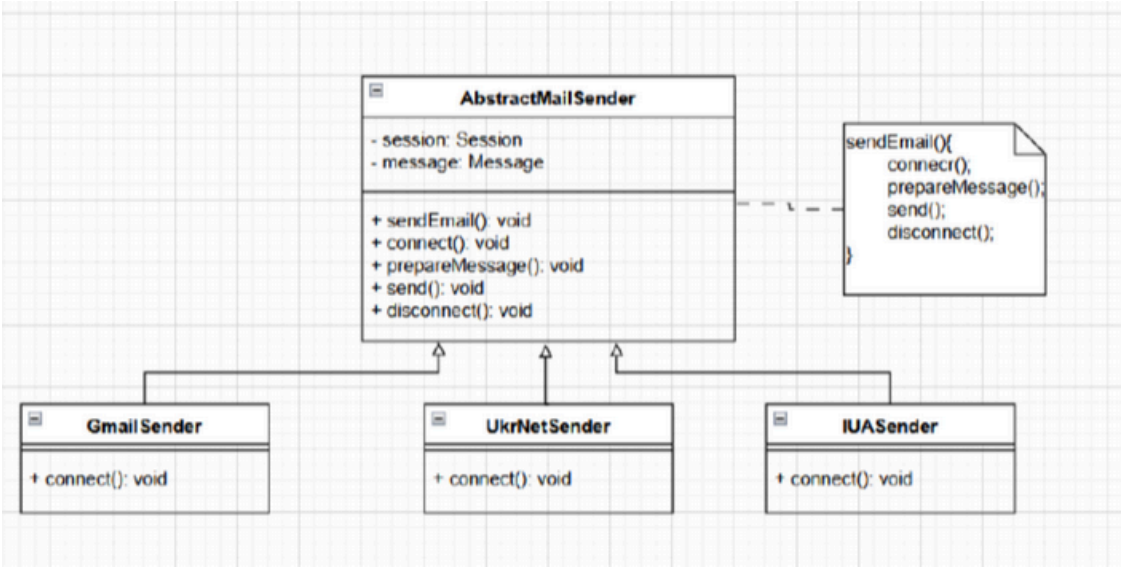
На діаграмі зображено структуру системи, у якій реалізовано клієнт-серверну архітектуру для отримання електронної пошти за протоколами IMAP та POP3. Суть такої архітектури полягає в тому, що клієнтський додаток виступає клієнтом, який підключається до поштових серверів (Gmail, IUA, UkrNet) за стандартними протоколами доступу до поштової скриньки.

Усі класи, зображені на схемі, є різними реалізаціями інтерфейсу MailReceiver, що визначає єдиний метод receiveEMails(). Залежно від того, до якого сервера потрібно підключитися, менеджер MailReceiverManager повертає конкретну реалізацію. Кожен з цих класів є клієнтом, який встановлює з’єднання із сервером пошти за IMAP або POP3, автентифікується через AccountService, завантажує потрібні папки через FolderService та отримує, обробляє й конвертує повідомлення через EmailMessageService. Саме використання IMAP і POP3 показує, що система взаємодіє з віддаленими поштовими серверами у режимі

клієнт-сервер: сервери зберігають пошту, а клієнт отримує її, використовуючи стандартизовані мережеві протоколи.

Посилання на код:

https://github.com/BeautifulBublik/TRPZ/tree/master/EmailClient/src/main/java/com/example/emailclient/service/receiver



На цій діаграмі показано структуру відправлення електронних листів, яка також побудована за принципом клієнт-серверної архітектури, але цього разу через протокол SMTP (Simple Mail Transfer Protocol). У такій моделі програма виступає клієнтом SMTP, а поштові сервери Gmail, UkrNet, IUA — це SMTP-сервери, які приймають та надсилають листи далі за призначенням.Конкретні класи GmailSender, UkrNetSender, IUASender розширюють базовий абстрактний відправник і реалізують специфічний спосіб підключення для кожного поштового сервера. Завдяки цьому програма поводиться як SMTP-клієнт, який підключається до реальних серверів постачальників поштових послуг, автентифікується і передає їм уже сформовані повідомлення.Таким чином, ця схема демонструє класичну модель клієнт - сервер SMTP, де клієнтський застосунок виконує всі кроки відправлення листів, а сервери відповідають за приймання та

подальшу маршрутизацію електронної пошти.

Посилання на код:

https://github.com/BeautifulBublik/TRPZ/tree/master/EmailClient/src/main/java/com/example/emailclient/service/sender

Питання до лабораторної роботи

1. Що таке клієнт-серверна архітектура?

   Клієнт-серверна архітектура — це модель взаємодії в комп'ютерній мережі, де функції розподілені між клієнтами (пристроями користувачів) та потужними серверами, що надають послуги та ресурси. Клієнт надсилає запит, а сервер його обробляє і відправляє відповідь, забезпечуючи спільне використання даних, програм та ресурсів у мережі, як у випадку з веб-сайтами

2. Розкажіть про сервіс-орієнтовану архітектуру.

   Сервісно-орієнтована архітектура (SOA) — це

   підхід до розробки програмного забезпечення, що розбиває великі системи на набір незалежних, слабко пов'язаних компонентів, які називаються сервісами, що взаємодіють через стандартизовані інтерфейси, щоб надавати бізнес-функціонал. Ці сервіси працюють як "чорні ящики", інкапсулюючи деталі реалізації, і можуть бути легко комбіновані та повторно використані для побудови складних додатків, підвищуючи гнучкість, масштабованість та швидкість розробки

3. Якими принципами керується SOA?

   Архітектура SOA ґрунтується на принципах, що забезпечують гнучкість та незалежність програмних компонентів. У її основі лежить слабке зв’язування, коли сервіси мінімально залежать один від одного та можуть змінюватися без впливу на систему. Кожен сервіс має стандартизований інтерфейс і чіткий контракт, який визначає його функції та спосіб взаємодії. Важливою є автономність — сервіс самостійно керує своїми

даними та логікою. Завдяки абстракції приховується внутрішня реалізація, а повторне використання та композиція сервісів дозволяють швидко будувати складні бізнес-процеси. У результаті SOA забезпечує масштабованість, сумісність і зручне управління сервісами в розподілених системах.

4. Як між собою взаємодіють сервіси в SOA?

Сервіси в SOA взаємодіють через стандартизовані інтерфейси (WSDL, OpenAPI), обмінюючись повідомленнями (XML, JSON) за допомогою протоколів (SOAP, REST), що дозволяє їм бути слабко пов'язаними, незалежними від платформ, працювати асинхронно через посередника (реєстр, шину ESB, оркестратор) та дозволяти динамічне виявлення й повторне використання функціоналу, досягаючи масштабованості та гнучкості.

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

У SOA розробники дізнаються про доступні сервіси через спеціальний реєстр або каталог, де зберігається інформація про всі їхні можливості та адреси. Кожен сервіс має чітко визначений контракт, наприклад WSDL, який описує структуру запитів і відповідей. Завдяки цьому розробник легко розуміє, як правильно звертатися до сервісу. Для взаємодії використовуються стандартизовані протоколи, зокрема SOAP або HTTP, а інструменти розробки можуть автоматично генерувати клієнтський код на основі контракту. Це робить пошук сервісів і роботу з ними передбачуваною та зручною.

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

Переваги клієнт-серверної моделі полягають у тому, що всі дані та логіка зосереджені на сервері, що забезпечує централізоване керування, безпеку та просте оновлення системи. Клієнти можуть бути легкими, а сервер — потужним, що дозволяє ефективно обробляти великі обсяги запитів. Крім того, така модель добре масштабуються за рахунок додавання нових

серверних ресурсів. Її недоліки пов’язані з тим, що сервер стає єдиною точкою відмови: якщо він не працює, клієнти також не можуть виконувати свої функції. Також можливе перевантаження сервера при великій кількості одночасних звернень, а розгортання та підтримка такої інфраструктури можуть вимагати значних витрат.

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

Основна її перевага полягає в тому, що всі вузли одночасно є і клієнтами, і серверами, тому система не залежить від єдиного центрального сервера. Завдяки цьому вона краще масштабується: що більше вузлів підключається, то більше сумарних ресурсів (трафіку, зберігання, обчислень) доступно. Також P2P підвищує стійкість — навіть якщо частина вузлів виходить із ладу, система продовжує працювати.Недоліком однорангової моделі є складність координації: дані можуть бути розподілені між різними вузлами, тому важче забезпечити цілісність, безпеку та однакову версію інформації. Також якість роботи залежить від продуктивності та стабільності кожного окремого учасника, що може призводити до нестабільності мережі. Крім того, такі системи складніше захищати й адмініструвати, оскільки немає центрального контролю.

8. Що таке мікро-сервісна архітектура?

Мікросервісна архітектура — це підхід до розробки ПЗ, коли великий додаток розбивається на сукупність невеликих, незалежних сервісів (мікросервісів), кожен з яких виконує певну бізнес-функцію, спілкується з іншими через API та може бути розроблений, розгорнутий і масштабований окремо, використовуючи різні технології. Цей підхід забезпечує гнучкість, швидкість розробки та відмовостійкість, на відміну від традиційного моноліту, де все є єдиною програмою

9. Які протоколи використовуються для обміну даними в мікросервісній

архітектурі?

У мікросервісній архітектурі для обміну даними найчастіше використовують синхронні протоколи (REST/HTTP/HTTPS, gRPC) для запит-відповідь, а також асинхронні протоколи та брокери повідомлень (AMQP, MQTT, Kafka, RabbitMQ) для подієво-орієнтованої комунікації, з обов'язковим застосуванням SSL/TLS для безпеки, часто в поєднанні з API-шлюзами та сервісними сітками

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, реалізацію шару бізнес-логіки у вигляді сервісів між контролерами та шаром доступу до даних не можна вважати сервіс-орієнтованою архітектурою. У цьому випадку сервіси є внутрішніми компонентами одного застосунку та працюють у межах одного процесу, тоді як SOA передбачає незалежні, мережеві сервіси з чіткими контрактами та можливістю автономного існування. Такий підхід правильніше назвати багатошаровою архітектурою, а не сервіс-орієнтованою.

Висновок: У проєкті клієнт-серверна архітектура була використана для отримання та надсилання електронної пошти, оскільки взаємодія з поштовими серверами Gmail, UkrNet та IUA здійснюється виключно через мережеві протоколи IMAP/POP3 та SMTP. Застосунок виступає клієнтом, який встановлює з’єднання з віддаленими серверами, автентифікувати, отримує або надсилає листи. Такий підхід забезпечує стандартизовану комунікацію, сумісність з різними провайдерами поштових послуг і можливість легко розширювати систему новими серверами без зміни загальної логіки роботи.Також були досліджені Peer-to-Peer, Service-oriented Architecture.