**ЛЕКЦІЯ 2: МОДЕЛІ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ**

Клієнт-серверна архітектура є фундаментальною моделлю в інформаційних технологіях, яка визначає структуру взаємодії між клієнтами, що запитують ресурси або послуги, і серверами, що надають їх. Ця архітектура стала основою для багатьох сучасних додатків, які використовуються у веб-середовищі, корпоративних мережах та мобільних платформах. Однак клієнт-серверна архітектура не є однорідною: існують різні моделі, що розрізняються за складністю, функціональністю та масштабованістю. У цьому контексті важливо розглянути дворівневі та багаторівневі архітектури, протоколи взаємодії, що забезпечують комунікацію між клієнтом і сервером, а також переваги та недоліки різних моделей клієнт-серверної архітектури.

**Існують концепції побудови системи клієнт-сервер:**

1. Слабкий клієнт – потужний сервер. У такій моделі вся обробка інформації перенесена на сервер, а у клієнта права доступу суворо обмежені. Сервер відправляє відповідь, яка не вимагає додаткової обробки. Клієнт взаємодіє з користувачем: складає та відправляє запит, приймає результат і виводить інформацію на екран.
2. Сильний клієнт – концепція, в якій частина обробки інформації надається клієнтові. У такому випадку сервер виступає сховищем даних, а вся робота по обробці та подання інформації переноситься на комп'ютер клієнта.

Система (додаток), яка заснована на клієнт-серверній взаємодії, включає три основних компоненти: **представлення даних**, **прикладний компонент**, **компонент управління ресурсами і їх зберігання**.

**ВАЖЛИВИЙ АСПЕКТ**

Ще варто відзначити, що в **основі взаємодії клієнт-сервер лежить принцип того, що таку взаємодію починає клієнт**: сервер лише відповідає клієнту і повідомляє про те, чи може він надати послугу клієнту і якщо може, то на яких умовах.

Не має значення, де фізично знаходиться клієнт і сервер. Клієнтське програмне забезпечення та серверне програмне забезпечення зазвичай встановлено на різних машинах, але також вони можуть працювати і на одному комп'ютері.

Цю концепцію розробили як перший крок у бік спрощення складної системи. Має такі сильні сторони:

* **Спрощення логіки**: сервер нічого не знає про клієнта і як він використовуватиме його дані надалі.
* Можуть бути **слабкі клієнти**: всі ресурсомісткі завдання можна перенести на сервер.
* Незалежний розвиток коду клієнтів та коду сервера.
* Багато різних клієнтів, приклад - Tomcat і різні браузери.

Найбільш базовий варіант взаємодії клієнта та сервера представлено на рисунку:



Тут важливо відзначити дві деталі. По-перше, з рисунка видно, що до одного сервера може звертатися безліч клієнтів. По-друге, вони можуть до нього звертатися одночасно. Це також важлива частина роботи сервера.

Один клієнт зазвичай взаємодіє з одним користувачем, тому часто навіть авторизація не потрібна. Однак сервер обробляє запити тисяч клієнтів, і під час обровки запиту йому потрібно вміти відрізняти авторизацію від автентифікації.

Важливо і те, що сервер обробляє тисячі запитів паралельно. А це означає, що під час розробки коду для бекенда завжди потрібно буде думати над завданням одночасного доступу до ресурсів. Також у коду сервера дуже висока ймовірність race condition (перегони потоків), deadlock (взаємне блокування потоків).

**Іще один важливий момент.** Архітектура клієнт-сервер визначає лише загальні принципи взаємодії між комп'ютерами, деталі взаємодії визначають різні протоколи.

Ця концепція (клієнт-сервер) говорить нам, що потрібно розділяти машини в мережі на клієнтські, яким завжди щось треба і на серверні, які дають те, що треба. У цьому взаємодія завжди починає клієнт, а правила, за якими відбувається взаємодія, визначає протокол.

Існує два види архітектури взаємодії клієнт-сервер: перший отримав назву **дворівнева архітектура клієнт-серверної взаємодії**, другий – б**агаторівнева архітектура клієнт-сервер** (іноді його називають трирівневою архітектурою або триланковою архітектурою, але це окремий випадок).

**2.1 ДВОРІВНЕВІ ТА БАГАТОРІВНЕВІ АРХІТЕКТУРИ**

Дворівнева архітектура, також відома як монолітна, є найпростішою формою клієнт-серверної моделі. У цій архітектурі клієнт і сервер виконують усі необхідні функції в межах одного рівня. Зазвичай сервер обробляє всі аспекти бізнес-логіки, управління даними та інтерфейсу користувача, тоді як клієнт виступає переважно як інтерфейс для взаємодії з користувачем. Така архітектура добре підходить для простих додатків, де обсяг обробки даних не є надто великим, а також коли немає потреби у масштабуванні або розподілі обов’язків між різними компонентами системи.

Дворівнева архітектура має певні переваги, такі як простота розробки та експлуатації. Зокрема, розробникам не потрібно турбуватися про складність комунікації між різними рівнями системи, що дозволяє зосередитися на реалізації основної функціональності. Однак існують і суттєві недоліки. Дворівневі системи зазвичай погано масштабуються, оскільки збільшення кількості користувачів або даних призводить до перевантаження сервера. Крім того, у випадку збою серверу вся система стає недоступною, що може бути критичним для бізнес-додатків.

На противагу дворівневим, багаторівневі архітектури розподіляють функціональність між декількома рівнями, кожен з яких відповідає за певний аспект роботи системи.

**Дворівнева архітектура** складається з двох вузлів:

* сервер, який відповідає за отримання запитів і відправку відповідей клієнту, використовуючи при цьому лише власні ресурси;
* клієнт, який представляє користувацький інтерфейс.

Принцип роботи полягає в тому, що сервер отримує запит, обробляє його і відповідає безпосередньо, без використання сторонніх ресурсів.



**Трирівнева архітектура** – це найпоширеніша архітектура взаємодії в інтернеті. Вона з'явилася, коли серверну частину дворівневої архітектури розділили на дві частини: **шар логіки** та **шар даних**.

Виглядає це приблизно так:



**Шар клієнта** – це частина «розподіленої програми», яка відповідає за взаємодію з користувачем. Цей шар не повинен містити бізнес-логіку та зберігати критично важливі дані. Також він не повинен взаємодіяти з шаром бази даних безпосередньо, а лише через бізнес-логіку.

Однак якась логіка тут все ж є. По-перше, це взаємодія з користувачем через інтерфейс, валідація даних, що вводяться ним, робота з локальними файлами. Ще сюди можна віднести все, що стосується авторизації користувача та шифрування даних під час роботи з сервером.

По-друге, це нескладна логіка бізнесу. Наприклад, інтернет-магазин надіслав список товарів. На стороні клієнта ми можемо їх відсортувати, відфільтрувати. І примітивне зберігання даних тут теж є: кешування, кукі залогіненого користувача тощо.

**Шар бізнес-логіки** розташовується на другому рівні: на ньому зосереджена більша частина бізнес-логіки. За його межами залишаються лише фрагменти, що експортуються на клієнта, а також елементи логіки, занурені в базу даних (збережені процедури та тригери).

Дуже часто сервери бізнес-логіки містять не лише цю саму логіку, але й вирішують завдання масштабування: код розбивається на частини та розноситься на різні сервери. Деякі особливо популярні сервіси можуть запускатися на десятках серверів. Навантаженням між ними керує load balancer.

Серверні програми зазвичай проектуються таким чином, щоб можна було легко запустити ще одну копію сервера, і вона почала працювати в кооперації з іншими його копіями. Тобто навіть у процесі написання серверного коду в тебе ніколи не буде гарантій, що твій статичний клас запущено в єдиному екземплярі.

**Шар даних** забезпечує зберігання даних і виноситься на окремий рівень, реалізується, зазвичай, засобами систем управління базами даних (СУБД), підключення до цього компоненту забезпечується лише з рівня сервера застосунків.

**Причини відокремлення шару даних**

Відділення шару даних у повноцінний третій шар відбулося з багатьох причин, але найголовніша з них — це зростання навантаження на сервер.

По-перше, **бази даних почали вимагати багато пам'яті** та процесорного часу на обробку даних. Тому їх почали виносити на окремі сервери.

Бекенд при збільшеному навантаженні можна було легко дублювати і підняти десять копій одного сервера, а дублювати базу даних не було змоги - база все ще залишалася єдиним і неподільним компонентом системи.

По-друге, **бази даних стали розумними** — у них виникла власна бізнес-логіка. Вони стали підтримувати процедури, що зберігаються, тригери, власні мови типу PLSQL. І навіть з'явилися програмісти, які почали писати код, який виконується всередині СУБД.

Усю логіку, яка не була зав'язана на даних, виносили в бекенд і паралельно запускали на десятках серверів. Все, що було критично зав'язано на даних, залишалося всередині СУБД. Там проблеми навантаження, яке зросло, доводилося вирішувати своїми методами:

* Кластер бази даних — група серверів БД, які зберігають одні й самі дані і синхронізують їх у певному протоколі.
* Шардування — дані дробляться на логічні блоки і розносяться на різні сервери БД. Дуже складно підтримувати зміни БД за такого підходу.
* Підхід NoSQL — зберігання даних у БД, які побудовані для зберігання величезної кількості даних. Часто це навіть не бази, а специфічні файлові сховища. Дуже бідний функціонал у порівнянні з реляційними базами даних.
* Кешування даних. Замість простого кешу на рівні бази даних з'явилися цілі СУБД для кешування, які зберігали результат лише в пам'яті.

Зрозуміло, що для управління цим зоопарком серверних технологій потрібні були окремі люди та/або цілі команди, що призвело до винесення шару даних в окремий шар.

**Важливо!**Усі передові технології народжуються у надрах великих ІТ-корпорацій, коли старі підходи вже не можуть впоратися з новими викликами. Винесення баз даних в окремий шар вигадав не якийсь програміст, а група інженерів десь у надрах Oracle або IBM.

**Цікаво!**Коли Цукерберг починав писати Facebook, він працював просто на PHP+MySQL. Коли кількість користувачів досягла мільйонів, написали спеціальний транслятор, який переклав весь PHP код на C++ і скомпілювали його в нативний машинний код.

Також MySQL не здатний зберігати дані мільярдів користувачів, тому Facebook розробив концепцію NoSQL баз даних і написав потужну серверну NoSQL-СУБД — Cassandra. До речі, вона повністю написана на Java.

**Неоднозначність розташування логіки програми**

І хоча трирівнева архітектура практично однозначно розподіляє ролі між її частинами, не завжди можна правильно визначити, в яке саме місце системи потрібно додати нову частину бізнес-логіки (новий код).

Приклад такої неоднозначності бачимо на рисунку нижче:

Сірим фоном залита серверна частина, білим — клієнтська. Хороший приклад останнього підходу (крайній правий варіант) — це сучасні мобільні програми. На стороні клієнта (на телефоні) міститься представлення (відображення), логіка та дані. І лише інколи ці дані синхронізуються із сервером.

Приклад крайнього лівого варіанту — це типовий PHP-сервер, у якого вся логіка знаходиться на сервері, і він віддає клієнту вже статичний HTML. Десь між цими двома крайніми варіантами і буде ваш проект.

Трирівневу архітектуру можна розширити до **багаторівневої (N-tier, Multi-tier)** способом встановлення додаткових серверів. Багаторівнева архітектура дозволяє підвищити ефективність роботи інформаційної системи, а також оптимізувати розподіл її програмно-апаратних ресурсів.



Взаємодія клієнт-сервер дозволяє розділяти функціонал і обчислювальне навантаження між клієнтськими додатками (замовниками послуг) і серверними додатками (постачальниками послуг).

Багаторівнева архітектура також дозволяє покращити безпеку системи, оскільки рівень даних можна розташувати за межами доступу клієнтів, забезпечуючи централізований контроль доступу та шифрування. Це знижує ризик несанкціонованого доступу до критично важливих даних. Крім того, завдяки розподілу функцій між різними рівнями, такі системи зазвичай є більш надійними, оскільки вихід з ладу одного з компонентів не обов’язково призводить до повного відключення системи. Наприклад, якщо виникають проблеми на рівні бізнес-логіки, рівень інтерфейсу користувача може продовжувати працювати, відображаючи попередньо завантажені дані або показуючи повідомлення про тимчасову недоступність сервісу.

Незважаючи на очевидні переваги, багаторівневі архітектури мають і свої недоліки. Основним з них є складність розробки та підтримки таких систем. Кожен рівень повинен бути розроблений таким чином, щоб ефективно взаємодіяти з іншими рівнями, що потребує глибокого розуміння принципів розподілених систем і мережевих технологій. Крім того, збільшення кількості рівнів підвищує затримку у виконанні запитів, оскільки дані повинні передаватися між різними серверами. Це може бути критично важливим для систем, де важлива висока швидкість обробки даних.

**2.2 ПРОТОКОЛИ ВЗАЄМОДІЇ: HTTP, FTP, TCP/IP**

Архітектура клієнт-сервер визначає принципи спілкування між комп'ютерами, а правила і взаємодії визначені в протоколі.

Для забезпечення ефективної та надійної взаємодії між клієнтами та серверами використовуються різні протоколи комунікації. Протоколи визначають правила та формати обміну даними, що дозволяє клієнтам і серверам розуміти один одного та забезпечувати належний рівень взаємодії.

**Мережевий протокол** – це набір правил, за якими відбувається взаємодія між комп'ютерами в мережі.

Одними з найважливіших протоколів, що використовуються у клієнт-серверних архітектурах, є HTTP, FTP і TCP/IP.

**Мережеві протоколи:**

**TCP/IP** – набір (стек) протоколів передачі даних. TCP/IP – це позначення всієї мережі, яка працює на основі двох протоколів – TCP і IP.

**TCP (Transfer Control Protocol)** – протокол, який служить для встановлення надійного з'єднання між двома пристроями, передачі інформації і підтвердження її отримання.

**IP (Internet Protocol)**– інтернет протокол, який відповідає за правильність доставки повідомлень за певною адресою. При цьому дані розбиваються на пакети, які можуть доставлятися по-різному.

**MAC (Media Access Control)** – протокол, за допомогою якого відбувається ідентифікація мережевих пристроїв. Всі пристрої, підключені до інтернету, мають свою унікальну MAC адресу.

**ICMP (Internet control message protocol)** – протокол, який відповідає за обмін інформацією, але не використовується для передачі даних.

**UDP (User datagram protocol)**– протокол, який керує передачею інформації, але інформація не проходить перевірку при отриманні. Даний протокол працює швидше, ніж TCP.

**HTTP (Hyper Text Transfer Protocol)** – протокол передачі гіпертексту, на основі якого працюють всі сайти. Він запитує необхідні дані у віддаленій системи (веб-сторінки, файли).

**FTP (File Transfer Protocol)**– протокол передачі файлів зі спеціального файлового сервера на комп'ютер користувача.

**SSH (Secure Shell)** – протокол, який служить для забезпечення віддаленого керування системою по захищеному каналу.

**POP3 (Post Office Protocol)** – стандартний протокол поштового з'єднання, який відповідає за доставку пошти.

**SMTP (Simple Mail Transfer Protocol)**– протокол, який визначає правила для передачі пошти. Відповідає за повернення або підтвердження про доставку, оповіщення про помилку.

**HTTP, або Hypertext Transfer Protocol**, є основним протоколом, що використовується для передачі даних у веб-середовищі. Цей протокол був розроблений для передачі гіпертексту — документації, що містить посилання на інші документи, що дозволяє створювати зв'язки між різними ресурсами в Інтернеті. HTTP працює за принципом запит-відповідь: клієнт (наприклад, веб-браузер) надсилає запит до сервера, а сервер обробляє цей запит і надсилає відповідь, що може містити HTML-сторінку, файл або інші дані. Основою HTTP є технологія *клієнт-сервер,* тобто завжди виділяються два учасники обміну: клієнт, який робить запит (HTTP Request), і сервер, який відповідає клієнту на даний запит (передаючи по мережі пакет HTTP Response).



На цьому один *крок* протоколу закінчується, і даний цикл може повторюватися як завгодно велике число разів. Особливістю HTTP є той факт, що сам по собі протокол не підтримує інформацію про сесії, не зобов'язаний що-небудь знати про історію. Іншими словами, протокол не передбачає збереження стану. Є запит, і є відповідь на нього. Причому, як правило, відразу після відповіді сервер розірве ТСР-з'єднання з клієнтом з метою більш дбайливого використання власних ресурсів. При необхідності клієнти, які використовують HTTP, можуть самостійно зберігати інформацію про історію даного сеансу. Для цих цілей існують два механізми:

* - *куки* -cookies-невеликі пакети даних, що зберігаються на стороні клієнта і передаються разом із запитом. Як правило, cookie має два атрибути, що визначають його збереження клієнтом, тобто можливість збереження інформації на диску. У разі, якщо cookie дозволяється зберігати інформацію у клієнта на диску, вказується *час життя* даного cookie на стороні клієнта, після чого він буде вважатися застарілим, неактуальним. Зокрема, в якості cookie може виступати унікальний ідентифікатор сесії на стороні сервера;
* - *сесії* - HttpSessions - на стороні сервера. Сесія є програмно-доступний об'єкт, який формується сервером і містить в собі інформацію про роботу з даним клієнтом. Об'єкт сесії унікальний для сеансу роботи з кожним клієнтом. Зазвичай сесія має ідентифікатор, який зберігається в cookies на клієнті. Механізм сесій приховує подробиці реалізації, розробнику не потрібно відстежувати, що з інформації запам'ятовується на клієнтської частини, а що - на серверної. Досить просто отримати об'єкт *сесії* і думати про нього, що він однозначно визначає сеанс даного клієнта.

**FTP, або File Transfer Protocol,** є протоколом, розробленим для передачі файлів між клієнтами і серверами. На відміну від HTTP, який використовується переважно для передачі веб-контенту, FTP призначений для більш широкого спектру завдань, пов'язаних з обміном файлами. FTP дозволяє користувачам завантажувати файли на сервер (upload) або завантажувати їх з сервера (download), а також керувати файлами на сервері (наприклад, створювати, видаляти або переміщувати їх). FTP працює на основі двох з'єднань: командного, що використовується для передачі команд і відповіді сервера, і з'єднання для передачі даних, що використовується для фактичного обміну файлами. Цей підхід забезпечує високу гнучкість і дозволяє керувати передачею файлів більш ефективно.



Однак FTP має певні недоліки, пов'язані з безпекою, оскільки він передає дані у відкритому вигляді, що може призвести до їх перехоплення. Для забезпечення безпеки передавання файлів використовуються розширення протоколу, такі як FTPS (FTP Secure) і SFTP (SSH File Transfer Protocol), які додають шифрування до передачі даних.

**TCP/IP** є базовим набором протоколів, на якому побудовано весь Інтернет. Цей набір протоколів визначає, як дані мають бути сегментовані, передані та зібрані знову у коректному порядку. TCP (Transmission Control Protocol) відповідає за забезпечення надійності передачі даних, забезпечуючи контроль за доставкою кожного пакету та їх правильним складанням на стороні отримувача.



IP (Internet Protocol), у свою чергу, відповідає за маршрутизацію пакетів даних через мережу, використовуючи унікальні IP-адреси для ідентифікації пристроїв у мережі. TCP/IP є основою для більшості сучасних мережевих протоколів, включаючи HTTP та FTP. Завдяки своїй гнучкості та надійності, TCP/IP став стандартом де-факто для передачі даних в Інтернеті, забезпечуючи роботу широкого спектру додатків і сервісів.

3. ПЕРЕВАГИ ТА НЕДОЛІКИ РІЗНИХ МОДЕЛЕЙ

Кожна модель клієнт-серверної архітектури має свої унікальні переваги та недоліки, які роблять її більш або менш придатною для певних типів додатків і сценаріїв використання. Розуміння цих особливостей є ключовим для вибору оптимальної архітектури для конкретного проекту.

Двоорівнева архітектура, як вже зазначалося, є простою та легкою у реалізації. Вона підходить для невеликих додатків або систем, де масштабування не є критичним фактором. Основною перевагою Дворівневої архітектури є її простота: розробники можуть зосередитися на розробці одного компонента, не турбуючись про складні взаємодії між різними частинами системи. Це може значно знизити витрати на розробку та підтримку системи, особливо у випадках, коли проект має обмежений бюджет або короткий час розробки. Однак ця ж простота може стати і недоліком у випадку збільшення навантаження на систему. Дворівневі архітектури, як правило, погано масштабуються, оскільки всі функції обробляються одним сервером, що може швидко стати вузьким місцем у системі. Крім того, дворівневі архітектури є менш надійними, оскільки відмова одного сервера може призвести до повної недоступності системи.

Багаторівнева архітектура, зокрема трирівнева, пропонує більш високу гнучкість та масштабованість. Розподіл функцій між різними рівнями дозволяє ефективніше розподіляти навантаження та забезпечує кращу надійність системи. Наприклад, у випадку збільшення кількості користувачів можна додати додаткові сервери на рівні бізнес-логіки або бази даних, не змінюючи архітектуру клієнтської частини. Це дозволяє забезпечити стабільну роботу системи навіть при значних навантаженнях. Крім того, багаторівнева архітектура зазвичай краще підходить для великих корпоративних систем, де важливо забезпечити централізоване управління даними та безпеку. Наприклад, розташування сервера бази даних за межами прямого доступу клієнтів дозволяє централізовано контролювати доступ до даних і забезпечувати їх захист від несанкціонованого доступу.

Однак багаторівнева архітектура має і свої недоліки, головним з яких є складність розробки та підтримки системи. Розробка багаторівневої системи вимагає значних зусиль з боку розробників, оскільки кожен рівень повинен бути розроблений окремо і забезпечити ефективну взаємодію з іншими рівнями. Це може призвести до збільшення часу розробки та витрат на підтримку системи. Крім того, збільшення кількості рівнів може призвести до зростання затримок у передачі даних, що може бути критичним для систем, де важлива висока швидкість обробки запитів. Наприклад, у веб-додатках, де кожен запит користувача повинен пройти через кілька рівнів перед тим, як отримати відповідь, такі затримки можуть знизити загальну продуктивність системи та вплинути на користувацький досвід.

Слід також зазначити, що вибір між дворівневою та багаторівневою архітектурою не є бінарним. Існують проміжні варіанти, де клієнт і сервер обробляють лише частину функцій системи, залишаючи інші на стороні інших серверів або сервісів. Такі моделі можуть бути компромісом між простотою реалізації та масштабованістю системи, забезпечуючи достатню гнучкість для більшості додатків.

Протоколи взаємодії також впливають на вибір архітектури та її ефективність. Наприклад, HTTP є дуже гнучким і добре підходить для веб-додатків, але відсутність пам'яті може бути обмеженням для додатків, що потребують збереження стану. Використання сесій або інших механізмів для збереження стану може ускладнити розробку та вплинути на продуктивність системи. FTP є ефективним для передачі файлів, але має проблеми з безпекою, що може обмежити його використання у критичних системах. TCP/IP, з іншого боку, є надійним і універсальним, але може бути складним у налаштуванні для специфічних завдань, таких як реалізація складних протоколів або забезпечення високої продуктивності у великих мережах.

У кінцевому підсумку, вибір архітектури клієнт-серверної взаємодії залежить від конкретних вимог проекту, обмежень бюджету та ресурсів, а також від прогнозованого навантаження на систему. Дворівневі архітектури можуть бути ефективними для невеликих додатків, де простота і швидкість розробки мають ключове значення, тоді як багаторівневі архітектури є кращим вибором для великих корпоративних систем, де важливі масштабованість, надійність і безпека. Протоколи взаємодії також повинні вибиратися відповідно до специфіки системи та її вимог до продуктивності і безпеки.

Розвиток інформаційних технологій та постійна еволюція програмного забезпечення вимагають від розробників гнучкості та адаптивності у виборі архітектурних моделей. Знання переваг і недоліків різних клієнт-серверних архітектур дозволяє створювати ефективні та надійні системи, здатні задовольняти потреби користувачів і забезпечувати стабільну роботу навіть у умовах значного навантаження.

Корисно для читання:

<https://dou.ua/forums/topic/44636/>