**ЛЕКЦІЯ 1: ВСТУП ДО КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ**

****

**1. ВСТУП**

Клієнт-серверна архітектура є основною моделлю побудови програмних систем, яка використовується в сучасних інформаційних технологіях. Вона визначає, як компоненти програмного забезпечення взаємодіють між собою, щоб забезпечити виконання певних завдань. Модель такої системи полягає в тому, що клієнт відправляє запит на сервер, де він обробляється, і готовий результат відправляється клієнтові. Сервер може обслуговувати кілька клієнтів одночасно. Якщо одночасно приходить більше одного запиту, то вони встановлюються в чергу і виконуються сервером послідовно. Іноді запити можуть мати пріоритети. Запити з більш високими пріоритетами повинні виконуватися раніше.



Ця модель лежить в основі багатьох популярних технологій, таких як веб-додатки, мобільні додатки, хмарні сервіси, бази даних і навіть деякі елементи розподілених систем.

**2. ОСНОВНІ ПОНЯТТЯ КЛІЄНТ-СЕРВЕРНОЇ ВЗАЄМОДІЇ**

**Клієнт-серверна модель** передбачає розподіл завдань між двома основними компонентами: клієнтом і сервером. **Клієнт** — це програма, що ініціює запит, а **сервер** — програма, що обробляє запити і надає відповідні дані чи послуги. Ця модель є **асиметричною**, оскільки клієнт і сервер мають різні ролі та функціональність.

Діяльність система досить проста: клієнт відправляє серверу запит, той його обробляє і відправляє назад результати. У її структурі передбачено чітке розмежування того, хто ставить завдання і того, хто їх виконує.

Модель клієнт сервер передбачає використання наступних компонентів:

**Клієнт.** Це робоча станція з однією вихідною точкою кінцевий користувач. До його обов'язків входить відправка запитів та отримання відповіді. Їхня кількість може бути будь-якою.

**Сервер.** В одному контурі передбачена лише одна структурна одиниця. Взаємодіє зі всіма клієнтами, приєднаними до нього. Обробляє поставлені завдання, складає та спрямовує відповіді.

**Мережа.** Структурна одиниця, призначена здійснення обміну даними. Поєднує окремі робочі машини із загальними ресурсами.

**Програми.** ПЗ, з якого виконується обробка даних, забезпечення фізичного розподілу між основними структурними елементами. Передбачені окремі продукти для сервера, спрямовані на збирання інформації, роботу з нею та зберігання, а також для клієнта.

**2.1 Клієнт**

Клієнт — це програмне забезпечення, яке виконує запити до сервера і використовує його ресурси для виконання завдань. У контексті веб-додатків, клієнтом найчастіше виступає браузер або мобільний додаток. Клієнт формує HTTP-запит, що надсилається серверу, який обробляє його і відправляє відповідь.

* **Інтерактивність**: Клієнт забезпечує взаємодію з кінцевим користувачем.
* **Локальне виконання**: Клієнт може виконувати певні операції на стороні користувача, зменшуючи навантаження на сервер.
* **Запити до сервера**: Клієнт формує запити до сервера, які можуть містити дані користувача або іншу інформацію, необхідну для обробки на сервері.

**2.2 Сервер**

Сервер — це програмне забезпечення або апаратне забезпечення, яке надає послуги або відповіді на запити клієнта. Сервер обробляє запити, виконує відповідні операції (наприклад, доступ до бази даних, виконання скриптів) і повертає результати клієнту.

* **Обробка запитів**: Сервер приймає запити від клієнтів, обробляє їх і повертає результати.
* **Зберігання даних**: Сервери часто використовуються для зберігання даних у базах даних.
* **Захист даних**: Сервери відповідають за безпеку даних та забезпечення конфіденційності інформації.

Сервер працює в розрахованому на багато користувачів режимі, тобто може одночасно підтримувати роботу з великою кількістю клієнтів. Машина автоматично виставляє пріоритети, орієнтуючись на черговість надходження запитів, обробляє їх та надсилає назад результати. Сервер дозволяє налаштовувати пріоритети не лише виходячи з черги надходження запитів, а й за іншими параметрами, як приклад – за важливістю. Тоді такі запити будуть опрацьовані першими, навіть якщо надійдуть пізніше.

Вся система «клієнт сервер» працює так, що рядові користувачі навіть не підозрюють про пріоритетність та інші нюанси. Їхні запити обробляються дуже швидко, незалежно від того, чи дивляться вони відео, читають книги, вивчають необхідну інформацію, роблять покупки в інтернет-магазинах та ін.

Здебільшого у ролі клієнта виступає браузер певного користувача. А ось як сервер може використовуватися:

* апаратне забезпечення типу http;
* кластери серверів;
* локальні машини.

Обмін даними між вузлами здійснюється на підставі мережевих протоколів. І для кожної з послуг передбачено свій торент. Варіантів дуже багато. Так, для обробки запитів користувача призначається протокол http. Він передбачає дотримання чіткої структури з даними, які потрібно надати. Вимоги пред'являються навіть оформлення. У відповідь на http-запит сервер надсилає HTML документ. Запит від клієнта супроводжується додатковими даними, які дозволяють серверу зрозуміти, як з ними працювати. У відповіді від серверного обладнання також йдуть сторонні коди, крім даних, що запитуються. Вони допоможуть браузеру правильно зрозуміти повідомлення сервера. Тобто вся ініціатива у роботі йде від клієнта. Сервер лише дає відповідь на запит і вказує, які умови для цього потрібно дотриматись.

Одночасна присутність на одному сайті декількох користувачів може призвести до надходження на один дата-центр великої кількості звернень. Але тут є свої обмеження, пов'язані з потужністю із пропускною спроможністю серверного обладнання, характером запитів.

**3. ІСТОРІЯ РОЗВИТКУ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ**

Історія клієнт-серверної архітектури розпочалася в 1960-х роках, коли почали з'являтися перші мережеві системи. Проте справжній розвиток цієї архітектури почався у 1980-х роках з поширенням персональних комп'ютерів та локальних мереж.

Етапи розвитку клієнт-серверної архітектури є важливою частиною історії комп'ютерних наук, оскільки вони демонструють, як змінювалася та еволюціонувала взаємодія між комп'ютерними системами та користувачами. Кожен етап має свої особливості, технологічні досягнення та вплив на подальший розвиток інформаційних технологій. Нижче представлено детальний розгляд основних етапів розвитку клієнт-серверної взаємодії.

* 1. **Епоха мейнфреймів (1960-1970-ті роки)**
		1. **Характеристика мейнфреймів**

Мейнфрейми були великими, потужними комп'ютерами, які використовувалися для обробки великих обсягів даних та виконання складних обчислень. У той час всі обчислення виконувалися централізовано на мейнфреймі, а кінцеві користувачі взаємодіяли з ним через термінали, які фактично не мали власної обчислювальної потужності.

* + 1. **Особливості взаємодії**
* **Централізоване обчислення**: Всі обчислення і зберігання даних здійснювалися на мейнфреймі.
* **Термінали**: Користувачі використовували "дурні" термінали для введення команд та отримання результатів.
* **Відсутність розподілу обов'язків**: Термінали не виконували жодних обчислень, лише передавали введені дані до мейнфрейму.
	+ 1. **Обмеження**
* **Низька ефективність**: Високе навантаження на мейнфрейм, оскільки всі обчислення здійснювалися на ньому.
* **Відсутність масштабованості**: Додавання нових терміналів могло знизити продуктивність системи.
* **Відсутність автономності клієнтів**: Термінали повністю залежали від мейнфрейму.
	1. **Перехід до клієнт-серверної моделі (1980-ті роки)**
		1. **Поява персональних комп'ютерів**

У 1980-х роках з'явилися персональні комп'ютери (ПК), що стали доступними для широкого кола користувачів. Це викликало потребу в нових моделях взаємодії між комп'ютерами, які могли б використовувати локальні ресурси ПК та централізовані ресурси серверів.

* + 1. **Поява перших клієнт-серверних систем**
* **Локальні обчислення**: ПК почали виконувати частину обчислень локально, що зменшувало навантаження на сервери.
* **Розподіл функцій**: Клієнти стали відповідальними за взаємодію з користувачем, а сервери — за обробку даних і управління ресурсами.
	+ 1. **Введення локальних мереж (LAN)**
* **LAN**: Локальні мережі дозволили з'єднувати ПК і сервери в межах одного офісу або будівлі.
* **Покращення взаємодії**: ПК могли працювати незалежно, але водночас звертатися до серверів для отримання додаткових ресурсів або даних.
	1. **Золотий вік клієнт-серверної архітектури (1990-ті роки)**
		1. **Поява Інтернету та WWW**
* **Всесвітня павутина**: Інтернет і WWW зробили клієнт-серверну архітектуру основною моделлю для веб-додатків.
* **Веб-браузери**: Клієнтами стали веб-браузери, а серверами — веб-сервери, які обробляли HTTP-запити.
	+ 1. **Масштабованість і розподіленість**
* **Розподілені обчислення**: Клієнт-серверні системи стали більш розподіленими, дозволяючи різним серверам обробляти різні частини запитів.
* **Глобальний доступ**: Користувачі могли отримувати доступ до ресурсів з будь-якої точки світу, де є Інтернет.
	+ 1. **Поява технологій баз даних**
* **Розвиток СУБД**: Сервери баз даних, такі як MySQL, Oracle, стали ключовими компонентами клієнт-серверних систем.
* **SQL-протоколи**: Впровадження стандартних протоколів для взаємодії між клієнтами і серверами баз даних.
	1. **Епоха мобільних додатків і хмарних обчислень (2000-ті роки - теперішній час)**
		1. **Розвиток мобільних технологій**
* **Мобільні клієнти**: З розвитком смартфонів і мобільних додатків клієнт-серверна архітектура стала включати мобільні клієнти, які взаємодіяли з серверами через API.
* **RESTful API**: З'явилися стандартизовані способи взаємодії клієнтів з серверами, такі як RESTful API та GraphQL.
	+ 1. **Виникнення хмарних обчислень**
* **Хмарні сервіси**: Хмарні платформи, такі як AWS, GCP, Azure, дозволили розгортати сервери і бази даних у хмарі, забезпечуючи високу доступність і масштабованість.
* **Мікросервіси**: Розвиток мікросервісної архітектури, де додаток складається з набору незалежних сервісів, які взаємодіють між собою через API.
	+ 1. **Сучасні виклики і технології**
* **Нові протоколи**: Впровадження нових протоколів, таких як HTTP/2, HTTP/3, WebSocket, які покращують швидкість і ефективність взаємодії.
* **Безпека**: Удосконалення методів забезпечення безпеки даних, включаючи шифрування, автентифікацію та контроль доступу.
* **Реальний час**: Розвиток технологій для обробки даних у реальному часі, таких як WebSocket і сервери для стрімінгу.
	1. **Майбутнє клієнт-серверної архітектури**
		1. **Інтернет речей (IoT)**
* **Розширення клієнтів**: Нові типи клієнтів, включаючи пристрої IoT, які потребують надійної і масштабованої клієнт-серверної взаємодії.
* **Реальна часова обробка**: Збільшення вимог до обробки даних у реальному часі, особливо в умовах IoT.
	+ 1. **Квантові обчислення**
* **Нова ера обчислень**: Потенційний вплив квантових обчислень на архітектуру клієнт-серверної взаємодії, особливо у сфері криптографії і складних обчислень.
	+ 1. **Автономні системи**
* **Автономні клієнти**: Розвиток автономних клієнтів, які здатні приймати рішення на основі складних алгоритмів, таких як штучний інтелект і машинне навчання, і взаємодіяти з серверами для доступу до ресурсів або додаткових даних.

**4. ТИПОВІ ПРИКЛАДИ ВИКОРИСТАННЯ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ**

Клієнт-серверна архітектура використовується в багатьох сучасних програмних системах. Деякі з найпоширеніших прикладів:

**4.1 Веб-додатки**

Веб-додатки є класичним прикладом клієнт-серверної архітектури. У цьому випадку клієнтом виступає веб-браузер, який надсилає HTTP-запити до веб-сервера. Сервер обробляє запит, звертається до бази даних, виконує необхідні обчислення і повертає HTML-сторінку, яку браузер відображає користувачеві.

* **Клієнт**: Веб-браузер (Google Chrome, Mozilla Firefox, Safari).
* **Сервер**: Веб-сервер (Apache, Nginx).
* **Протокол**: HTTP/HTTPS.

**4.2 Мобільні додатки**

Мобільні додатки також використовують клієнт-серверну архітектуру. Клієнтська частина (мобільний додаток) взаємодіє з серверною частиною (бекендом) через API, зазвичай RESTful або GraphQL. Серверна частина відповідає за обробку запитів, роботу з базами даних і надання відповідей у вигляді JSON або XML.

* **Клієнт**: Мобільний додаток (iOS, Android).
* **Сервер**: Бекенд-сервер (Node.js, Django, Spring).
* **Протокол**: HTTP/HTTPS, WebSocket.

**4.3 Електронна пошта**

Електронна пошта також побудована на основі клієнт-серверної архітектури. Клієнт — це поштовий клієнт (наприклад, Outlook, Gmail), який взаємодіє з поштовим сервером для відправлення і отримання електронних листів.

* **Клієнт**: Поштовий клієнт (Outlook, Thunderbird, Gmail).
* **Сервер**: Поштовий сервер (SMTP-сервер для відправки, IMAP або POP3-сервер для отримання).
* **Протокол**: SMTP, IMAP, POP3.

**4.4 Бази даних**

Система управління базами даних (СУБД) є ще одним прикладом клієнт-серверної архітектури. Клієнтська частина може бути у вигляді веб-додатку або настільного додатку, який взаємодіє з сервером бази даних для виконання запитів SQL.

* **Клієнт**: Веб-додаток або настільний додаток (MySQL Workbench, pgAdmin).
* **Сервер**: СУБД (MySQL, PostgreSQL, Oracle).
* **Протокол**: SQL-протокол для обміну даними між клієнтом і сервером.

**5. ПЕРЕВАГИ ТА НЕДОЛІКИ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ**

Клієнт-серверна архітектура має як свої переваги, так і недоліки. Розуміння цих аспектів є критично важливим для розробки ефективних і масштабованих програмних рішень.

**5.1 Переваги**

1. **Централізоване управління**: Сервери контролюють доступ до даних і ресурсів, що забезпечує високу безпеку і зручність в управлінні.
2. **Масштабованість**: Можливість додавання нових клієнтів без значного збільшення навантаження на систему.
3. **Розподіл завдань**: Обчислювальні ресурси можуть бути розподілені між клієнтом і сервером, що дозволяє ефективно використовувати потужності обох.
4. **Захист даних**: Сервери забезпечують централізоване зберігання та захист даних, що зменшує ризики витоку інформації.
5. **Підтримка різних платформ**: Клієнт-серверна архітектура дозволяє взаємодіяти між різними платформами (наприклад, Windows-клієнт і Linux-сервер).

**5.2 Недоліки**

1. **Залежність від сервера**: Якщо сервер виходить з ладу, вся система може перестати працювати.
2. **Навантаження на сервер**: Сервери можуть стати вузьким місцем у системі при високих навантаженнях.
3. **Безпека**: Центральні сервери стають мішенню для атак, тому необхідно забезпечувати надійний захист.
4. **Складність розробки**: Розробка клієнт-серверних систем може бути складнішою через необхідність забезпечення ефективної взаємодії між клієнтом і сервером.
5. **Витрати на інфраструктуру**: Підтримка серверів і необхідної інфраструктури може бути дорогою.

**6. ОСНОВНІ ПРИНЦИПИ РОБОТИ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ**

Клієнт-серверна архітектура базується на декількох ключових принципах, які забезпечують ефективність і надійність взаємодії між клієнтами і серверами.

**6.1 Розподіл обов’язків**

Основний принцип клієнт-серверної архітектури полягає в чіткому розподілі обов'язків між клієнтом і сервером. Клієнт відповідає за ініціювання запитів і відображення результатів, тоді як сервер обробляє ці запити, виконує складні обчислення і управління даними.

**6.2 Запит-відповідь**

Взаємодія в клієнт-серверній архітектурі зазвичай базується на моделі запит-відповідь. Клієнт надсилає запит до сервера, сервер обробляє цей запит і повертає відповідь клієнту. Цей цикл є основою роботи більшості клієнт-серверних додатків.

**6.3 Незалежність платформ**

Клієнт і сервер можуть працювати на різних платформах і використовувати різні технології. Наприклад, клієнт може бути веб-додатком, який працює в браузері на Windows, а сервер може працювати на Linux.

**6.4 Мережева взаємодія**

Клієнт і сервер зазвичай взаємодіють через мережу, використовуючи протоколи, такі як TCP/IP, HTTP, або WebSocket. Ця взаємодія може відбуватися як у межах локальної мережі, так і через Інтернет.

**7. МАЙБУТНЄ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ**

Клієнт-серверна архітектура продовжує еволюціонувати з розвитком нових технологій і підходів. Розподілені системи, мікросервіси, хмарні обчислення і нові протоколи комунікації відкривають нові можливості для покращення цієї архітектури.

**7.1 Мікросервіси**

Мікросервісна архітектура є розвитком клієнт-серверної моделі, де монолітні серверні додатки поділяються на набір незалежних мікросервісів. Кожен мікросервіс виконує певну функцію і може взаємодіяти з іншими через чітко визначені API. Такий підхід дозволяє підвищити гнучкість і масштабованість системи.

**7.2 Хмарні обчислення**

Хмарні сервіси, такі як Amazon Web Services (AWS), Google Cloud Platform (GCP) і Microsoft Azure, надають можливості для гнучкого розгортання і масштабування клієнт-серверних додатків. Хмари дозволяють розробникам створювати і керувати серверами, базами даних і іншими сервісами без необхідності підтримки фізичної інфраструктури.

**7.3 Нові протоколи та стандарти**

Нові протоколи, такі як HTTP/2, HTTP/3, WebSocket і GraphQL, дозволяють покращити ефективність і продуктивність клієнт-серверних систем. Вони забезпечують швидшу передачу даних, підтримку реального часу і більш ефективне використання мережевих ресурсів.

**ВИСНОВОК**

Клієнт-серверна архітектура є однією з найважливіших моделей у сучасних інформаційних технологіях. Вона забезпечує ефективну взаємодію між різними компонентами системи, дозволяє масштабувати додатки, забезпечує високий рівень безпеки та надійності. Розуміння основ цієї архітектури є ключовим для розробників, архітекторів систем і IT-інженерів. Розвиток клієнт-серверної архітектури пройшов довгий шлях від мейнфреймів до сучасних хмарних і мобільних систем. Кожен етап цього розвитку відображає зміну потреб користувачів, технологічний прогрес і еволюцію підходів до обробки та зберігання даних. Майбутнє клієнт-серверної архітектури залежатиме від нових викликів, таких як Інтернет речей, квантові обчислення і автономні системи, що ще більше підвищить її значимість у світі інформаційних технологій.