**Back-end development & frameworks in server-side software**

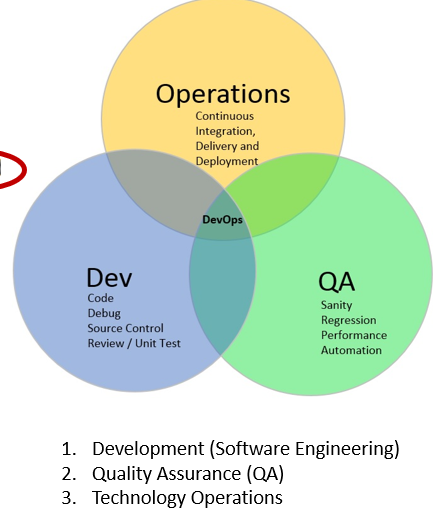
**Framework –** бібліотеки мови програмування серверної частини, які будують серверну структуру сайту.

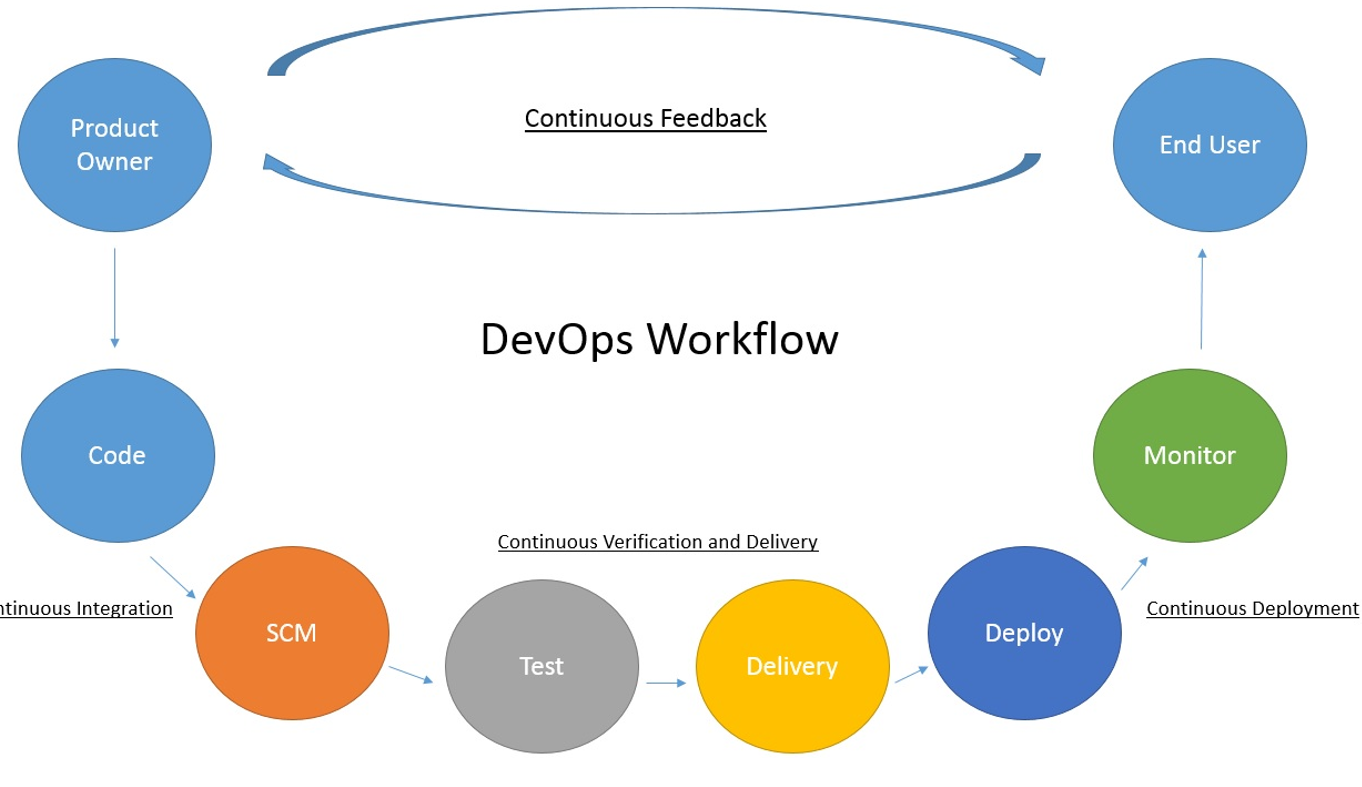
**Server-side scripts -** обробляють запити та отримують те, що їм потрібно з бази даних.

**APIs structure -** як відбувається обмін даними між базою даних і будь-яким програмним забезпеченням, яке до неї звертається.

**Stack –** включає в себе БД, фреймворки, сервери, операційні системи.

**DevOps**

****

****

**Domain Name System (DNS)**

Перекладає доменні імена, які легше запам’ятовуються, на цифрові IP-адреси.

По суті, це назва певного сайту в url, бо так то сайт має своє ip(аналогія з номерами телефону).

www(імя компа) youtube(домен 2 рівня) .com(першого рівня TLD or gTLD or ccTLD) (також кореневий домен)

DNS resolver(локальний) – сервер провайдера інету

**Three primary kinds of selectors**

* За тегом (h1)
* За id (#element\_id)
* За назвою класу (.myClass)

**jQuery features**

**Маніпуляція DOM** – спрощує вибір елементів DOM перглядає і змінює їх. Використовує кросбраузерний механізм Sizzle.

**Обробка подій** - пропонує елегантний спосіб фіксувати різноманітні події.

**Підтримка AJAX**

**Анімація** – має багато вбудованих анімаційних ефектів.

WEBPACK довзоляє модульний підхід до розробки.

**ECMAScript –** стандарт для мови сценаріїв. Він найбільш відомий як стандарт JavaScript, призначений для забезпечення [взаємодії](https://en.wikipedia.org/wiki/Interoperability) веб [-сторінок](https://en.wikipedia.org/wiki/Web_page) у різних [веб-переглядачах](https://en.wikipedia.org/wiki/Web_browser) .

**Features of TypeScript**

TypeScript це по суті просто JavaScript. Він підтримує інші бібліотеки JS. Будь-який дійсний файл .js можна скомпілювати з іншими файлами TypeScript. TypeScript переноситься між браузерами, пристроями та операційними системами.

**Benefits of TypeScript**

Транспілятор TypeScript забезпечує функцію перевірки помилок. TypeScript компілює код і генерує помилки компіляції.

TypeScript постачається з додатковою системою статичного введення та виведення типу через TLS (мовна служба TypeScript).

TypeScript підтримує визначення типів для існуючих бібліотек JavaScript.

TypeScript підтримує такі концепції об’єктно-орієнтованого програмування, як класи, інтерфейси, успадкування тощо.

**Some features of AngularJS**

Автоматична синхронізація даних між компонентами моделі та представлення.

Контролер — це функції JavaScript, прив’язані до певної області.

Директиви є маркерами елементів DOM (таких як елементи, атрибути, css тощо). Їх можна використовувати для створення власних тегів HTML, які слугуватимуть новими власними віджетами. AngularJS має вбудовані директиви (ngBind, ngModel...)

Шаблони – це відрендерене подання з інформацією від контролера та моделі. Це може бути один файл (наприклад, index.html) або кілька переглядів на одній сторінці з використанням "часток".

**Two basic principles of webpack**

З Webpack все може бути модулем. Це включає файли JS, CSS, зображення та HTML. Отже, будь-який артефакт можна розділити на невеликі, керовані частини для повторного використання.

Webpack завантажує лише те, що вам потрібно, коли вам це потрібно. Webpack генерує кілька менших «пакетних» файлів, що дозволяє завантажувати частини програми асинхронно. Таким чином, користувачам не потрібно чекати невиправдано довгого завантаження, щоб почати використовувати додаток.

**Qooxdoo**

Кросбраузерні DOM маніпуляції для покращення вебсайтів за допомогою багатого досвіду користувача

Створення мобільних аплікух на основних мобільних платформах без використання HTML

Створення десктопних застосунків без знань HTML/CSS

Використовуйте той самий шаблон ООП, відомий на стороні клієнта, повторно використовуйте код і створюйте файли, які можна розгорнути у своєму серверному середовищі.

**Client-server model**

Модель «клієнт-сервер» — це розподілена структура програми, яка розподіляє завдання або робочі навантаження між постачальниками ресурсів або послуг, які називаються серверами, і запитувачами послуг, які називаються клієнтами.

Клієнт-хост і сервер-хост мають дещо різні значення, ніж клієнт і сервер.

**Multi-tier architectures**

архітектури «жирний клієнт/тонкий сервер».

архітектури тонкого клієнта/жирного сервера

Трирівневі архітектури: і клієнт, і сервер бази даних звільнені від надання бізнес-логіки

Концептуально, багаторівнева архітектура є результатом повторного застосування парадигми клієнт/сервер

**Layered Architecture**

**Shared**

Ентіті може містити інші ентіті

Сутність може використовувати один або кілька перерахувань для своїх властивостей.

Усі компоненти рівня можуть посилатися на сутності та перерахування

**Data**

Компонент доступу до даних посилається на таблицю або подання в БД

Компонент доступу до даних може керувати кількома пов’язаними таблицями, наприклад Orders і OrderItems.

Для керування доступом до зовнішніх служб слід використовувати агент даних (відомий як Service Agent)

Для керування доступом до файлів слід використовувати агент даних (відомий як файловий агент)

**Business**

**Services**

**Presentation**

**NodeJS**

Для створення HTTP сервера, TCP сервера, DNS сервера, Static File сервер, Web chat app, онлайн ігри

Не потрібно використовувати:

Коли важкі і потребуючі процесора обчислення, бо івентлуп потребує CPU(проца)

NodeJS API продовжує змінювати багато однієї версії на іншу, і зворотна сумісність дуже мала. Більшість пакунків також нестабільні. Тому виробництво ще не готове

NodeJS не підходить для фреймворків додатків корпоративного рівня, таких як Spring, Django. Програми, написані на таких платформах, призначені для високої взаємодії з користувачем і містять складну бізнес-логіку

Yahoo!, LinkedIn, eBay, Dow Jones написані на ньому

NodeJS значною мірою покладається на модулі, у попередніх прикладах потрібно ключове слово завантажувати модулі http & net.

**HTTP 1.1 Protocol**

TCP — це протокол транспортного рівня, а HTTP — це протокол прикладного рівня, який працює через TCP

**HTTP 1.1 Method and Description**

GET - для отримання інформації з заданого сервера за допомогою заданого URI

HEAD - Те саме, що й GET, але передає лише рядок стану та розділ заголовка.

POST - використовується для надсилання даних на сервер, наприклад інформації про клієнта, завантаження файлів тощо за допомогою форм HTML.

PUT - Замінює всі поточні представлення цільового ресурсу завантаженим вмістом

DELETE - Видаляє всі поточні представлення цільового ресурсу, задані URI.

OPTIONS - Описує варіанти зв'язку для цільового ресурсу.

CONNECT - Встановлює тунель до сервера, ідентифікованого заданим URI.

TRACE - Виконує перевірку зациклення повідомлення на шляху до цільового ресурсу.

**Nginx Web Server**

Керований подіями — це підхід до обробки різноманітних завдань як подій. Наприклад, вхідне підключення є подією, читання диска є подією тощо. Ідея полягає в тому, щоб не витрачати ресурси сервера, якщо немає події для обробки.

Підключення користувачів NGINX обробляються у високоефективних циклах виконання всередині обмеженої кількості однопотокових процесів, які називаються робочими. Кожен працівник може обробляти тисячі одночасних підключень і запитів на секунду.

NGINX не створює новий процес або потік для кожного з’єднання.

Асинхронний означає, що цикл виконання не застряє на певних подіях.

Однопотоковий означає, що багато підключень користувачів можуть оброблятися одним робочим процесом, що, у свою чергу, допомагає уникнути надмірного перемикання контексту та сприяє більш ефективному використанню пам’яті та ЦП.

Відповідно до сайту проекту, NGINX забезпечує роботу 40 відсотків із 10 000 найзавантаженіших сайтів Інтернету та понад 20 відсотків усіх веб-сайтів, включаючи Dropbox, Github і Zappos.

Версія спільноти NGINX має всі необхідні функції та можливості для створення веб-сайтів і служб, які потребують продуктивності, масштабованості та надійності.

NGINX PLUS додає такі готові для підприємства функції, як балансування навантаження, постійність сеансу, перевірки працездатності, моніторинг і розширене керування.

**PostgreSQL vs. MySQL**

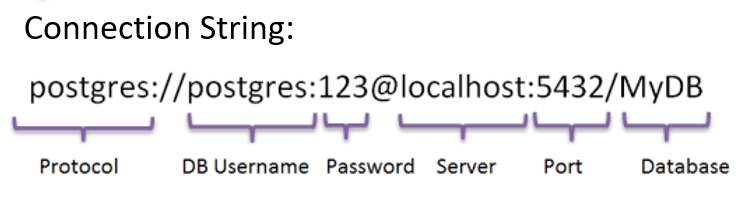
Сумісність зі стандартом ANSI? – MySQL підртримує нестандартні розширення. Але в Postgres вони вбудовані з самого початку

Підзапити можуть бути проблематичними – вони краще реалізовані в Postgres

Підтримка JSON і NoSQL – така можливість появилась в Postgres

Краще ліцензування – теж Postgres

Розширення сервера – обидві мають подібну систему, але Postgres підтримує більше мов



**Key Principles of Software Architecture**

**Поділ турбот** - Розділіть свою програму на окремі функції з якомога меншим перетином у функціональності. Важливим фактором є мінімізація точок взаємодії для досягнення високої когезії та низького зв’язку

**Принцип єдиної відповідальності** - Кожен компонент або модуль має відповідати лише за певну функцію чи функціональність, або сукупність пов’язаних функцій.

**Принцип найменших знань (також відомий як Закон Деметри або LoD)** - Компонент або об'єкт не повинен знати про внутрішні деталі інших компонентів або об'єктів.

**Не повторюйся** – певна функціональність повинна бути реалізована лише в одному компоненті.

**Мінімізуйте попередній дизайн** - Проектуйте лише те, що необхідно.

Архітектурний шаблон (архітектурний стиль) — це загальне, багаторазове рішення поширеної проблеми в архітектурі програмного забезпечення.

**Sphinx –** тулза для створення документації(Python)

**DevOps**

Кінцевою метою є розгортання оновлень програмного забезпечення для виробництва якомога частіше

Ansible – популярна, потужна тулза для девопсів

**Управління конфігурацією**– налаштування сервера/мережевих пристроїв

**Розгортання** – розгортання нового програмного забезпечення через розподілену інфраструктуру.

**Оркестровка** – розгортання змін у певному порядку, наприклад, спочатку веб-сервер, sql db тощо.

**Ініціалізація** – ініціалізація нових серверів у публічній хмарі, як-от Rackspace, EC2 або Azure.

SSH забезпечує безпечний канал через незахищену мережу в архітектурі клієнт-сервер.

Ansible керує машинами Windows за допомогою PowerShell

Ansible використовує Playbooks (у форматі YAML), які є основними сценаріями.

Playbooks посилаються на хости, які потрібно налаштувати за допомогою впорядкованого списку завдань. Виконання завдання виконується для всіх хостів паралельно.

На основі посібника Ansible генерує скрипти Python.

Ansible використовує спокусливу мову Jinja2.

Деякі інші переваги Ansible:

Використання YAML для підручника, легкого для читання та розуміння.

На віддалених хостах не потрібні плагіни чи агенти.

Для серверів потрібен лише ssh і Python 2.5+.

Управління конфігурацією на основі Push, коли нові конфігурації надсилаються на пристрої без періодичної перевірки їхніх агентів.

Jenkins дозволяє continuous integration

Головною метою CI є запобігання проблемам інтеграції концепція вводить сервери збірки, які автоматично запускають модульні тести періодично або навіть після кожного коміту та повідомляють про результати розробникам.

**Clouds**

Хмарні обчислення — це обчислювальна інфраструктура та модель програмного забезпечення для повного доступу до спільних пулів настроюваних ресурсів (наприклад, комп’ютерних мереж, серверів, сховищ, програм і послуг), які можна швидко надати з мінімальними зусиллями керування, часто через Інтернет.

Моделі розгортання: приватна хмара, публічна хмара, гібридна хмара.

**"five essential characteristics"**

**Самообслуговування на вимогу.** Споживач може в односторонньому порядку надати обчислювальні можливості, такі як серверний час і мережеве сховище, за потреби автоматично, не вимагаючи взаємодії людини з кожним постачальником послуг.

**Широкий доступ до мережі.** Можливості доступні через мережу та доступ до них через стандартні механізми, які сприяють використанню неоднорідними тонкими або товстими клієнтськими платформами (наприклад, мобільними телефонами, планшетами, ноутбуками та робочими станціями).

**Об'єднання ресурсів**. Обчислювальні ресурси постачальника об’єднуються для обслуговування кількох споживачів за допомогою моделі з кількома клієнтами, причому різні фізичні та віртуальні ресурси динамічно призначаються та перепризначаються відповідно до попиту споживачів.

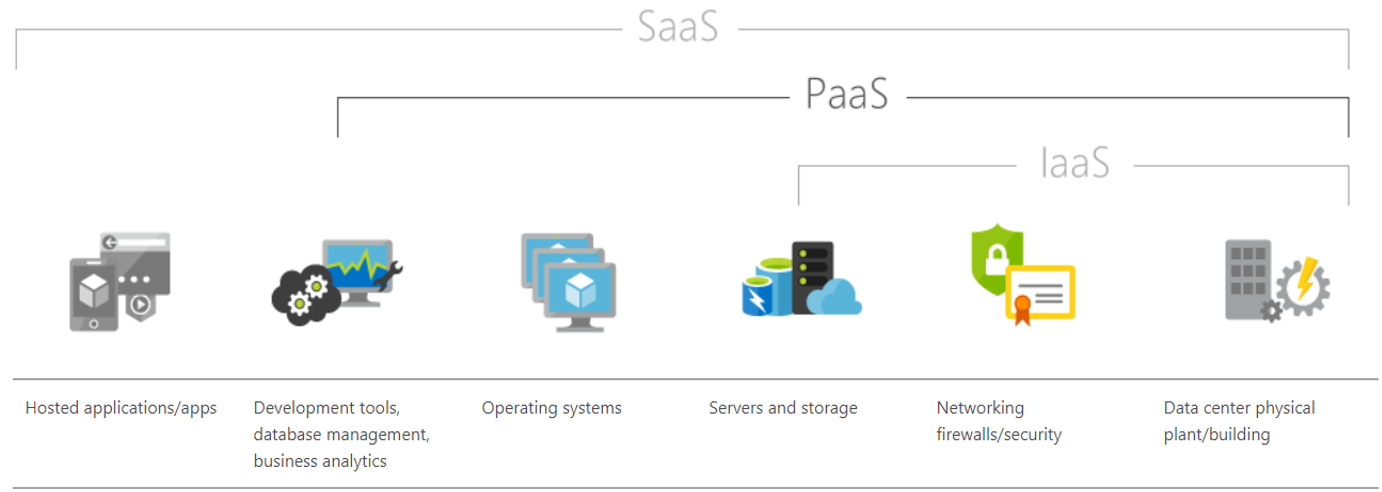
**Швидка еластичність**. Можливості можна еластично надавати та вивільняти, у деяких випадках автоматично, для швидкого масштабування назовні та всередину відповідно до попиту. Для споживача можливості, доступні для забезпечення, часто здаються необмеженими і можуть бути використані в будь-якій кількості в будь-який час.

**Розмірене обслуговування**. Хмарні системи автоматично контролюють і оптимізують використання ресурсів, використовуючи можливості вимірювання на певному рівні абстракції, що відповідає типу послуги (наприклад, зберігання, обробка, пропускна здатність і активні облікові записи користувачів).

**Інфраструктура як послуга (IaaS).** Можливість, яка надається споживачеві, полягає в забезпеченні обробки, зберігання, мереж та інших фундаментальних обчислювальних ресурсів, де споживач може розгортати та запускати довільне програмне забезпечення, яке може включати операційні системи та програми.

**Платформа як послуга (PaaS).** Можливість, яка надається споживачу, полягає в розгортанні в хмарній інфраструктурі створених споживачем або придбаних додатків, створених за допомогою мов програмування, бібліотек, служб та інструментів, які підтримуються постачальником.

**Програмне забезпечення як послуга (SaaS).** Споживачеві надається можливість використовувати програми постачальника, що працюють у хмарній інфраструктурі.



**Безпека як послуга (SECaaS).** Бізнес-модель, у якій великий постачальник послуг інтегрує свої служби безпеки в корпоративну інфраструктуру на основі передплати більш економічно ефективно, ніж більшість фізичних осіб або корпорацій, які можуть надати самостійно, якщо врахувати загальну вартість володіння

**Мобільний «бекенд» як послуга (MBaaS)**. Розробникам веб-додатків і мобільних додатків надається спосіб зв’язати свої додатки з хмарним сховищем і службами хмарних обчислень за допомогою інтерфейсів програмування додатків (API), відкритих для їхніх додатків і користувальницьких комплектів розробки програмного забезпечення (SDK).

**Безсерверні обчислення** — це модель виконання хмарних обчислень, у якій хмарний провайдер динамічно керує розподілом машинних ресурсів.

**Containers**

Контейнери — це спосіб запуску програмного забезпечення в ізольованих середовищах.

Контейнерні додатки пропонують більшу портативність і легкість розгортання.

Контейнери віртуалізують операційну систему замість апаратного забезпечення, контейнери є більш портативними та ефективними

**Модель взаємозв’язку відкритих систем (модель OSI)** — це концептуальна модель, яка характеризує та стандартизує комунікаційні функції телекомунікаційної або обчислювальної системи без урахування їх внутрішньої структури та технології, що лежить в основі. Його метою є сумісність різноманітних систем зв’язку зі стандартними протоколами. Модель розбиває комунікаційну систему на рівні абстракції.

**HTML**

«Елементи» та «теги» — це терміни, які часто плутають. Документи HTML містять теги, але не містять елементів. Елементи генеруються лише після кроку аналізу з цих тегів

Оскільки HTML базується на SGML. Стандартна узагальнена мова розмітки (SGML) — це стандарт для визначення узагальнених мов розмітки для документів

Вхідні дані для процесу синтаксичного аналізу HTML складаються з потоку кодових точок Unicode, який проходить через етап токенізації, за яким слідує етап побудови дерева.

Результатом є об’єкт Document.

У звичайному випадку дані, які обробляються на етапі токенізації, надходять із мережі, але вони також можуть надходити зі сценарію, запущеного в агенті користувача, напр. за допомогою API document.write().

**HTML5 vs HTML4: General Differences**

**Спрощений і зрозумілий синтаксис** - HTML5 версія має простіший синтаксис

**Мультимедійні елементи** - HTML5 підтримує інтеграцію мультимедійних файлів за допомогою тегів. У HTML4 версії потрібно додаткові плагіни.

**Доступ до географічного розташування користувача** - HTML5 легко отримати місцезнаходження користувача

**Сховище на стороні клієнта** - У HTML5 зберігання важливих даних чз кеш програми, до якої можна отримати доступ чз JS. У HTML4 використовувався обмежений кеш

**Зв'язок клієнт-сервер** - HTML5 пітримує вебсокети, що забезпечує повний дуплексний зв'язок. У HTML4 зв'язок був чз потокове передавання

**Механізм обробки потоків JavaScript** - HTML5 містить JS Web Worker API, який дозволяє виконувати JavaScript і інтерфейс браузера в окремих потоках.

**Сумісність з браузером** – HTML5 в цьому плані відстає від HTML4.

**Tag Differences in HTML4 and HTML5**

**Видалено <Applet> і додано <Object> у HTML5**

**<Acronym> видалено та <Abbr> додано в HTML5** - використовувався для відображення абревіатур у веб-переглядачі

**Різниця у використанні тегу <hr>** - використовувався для малювання лінії в HTML4, а 5 версії для визначення тематичної розриву на веб-сторінці.

**Різниця у використанні тегу <a>** - використовувався як прив’язка, а також для посилання на посилання. У HTML5 тег <a> використовується лише як гіперпосилання

**Атрибут схеми видалено з тегу <meta> в HTML5** - Тег визначається в розділі заголовка документа HTML і містив інформацію про дані.

**Web server**

Веб-сервер — це комп’ютерна система, яка обробляє запити через HTTP — основний мережевий протокол, який використовується для розповсюдження інформації у Всесвітній павутині.

З боку апаратного забезпечення веб-сервер — це комп’ютер, який зберігає файли компонентів веб-сайту і доставляє їх на пристрій кінцевого користувача. Він підключений до Інтернету та доступний через доменне ім’я, наприклад mozilla.org.

З боку програмного забезпечення веб-сервер містить кілька частин, які контролюють, як веб-користувачі отримують доступ до розміщених файлів, принаймні HTTP-сервер. HTTP-сервер — це частина програмного забезпечення, яка розуміє URL-адреси (веб-адреси) і HTTP (протокол, який використовує ваш браузер для перегляду веб-сторінок).

Основною функцією веб-сервера є зберігання, обробка та доставка веб-сторінок клієнтам. Зв'язок між клієнтом і сервером відбувається за допомогою протоколу передачі гіпертексту (HTTP).

Статичний веб-сервер або стек складається з комп’ютера (апаратного забезпечення) з HTTP-сервером (програмне забезпечення). Ми називаємо це «статичним», тому що сервер надсилає свої розміщені файли «як є» у ваш браузер

Динамічний веб-сервер складається зі статичного веб-сервера та додаткового програмного забезпечення, найчастіше сервера додатків і бази даних. Ми називаємо це «динамічним», оскільки сервер додатків оновлює розміщені файли перед тим, як надсилати їх у ваш браузер через сервер HTTP

**Hosting files**

Веб-сервер спочатку має зберігати файли веб-сайту, а саме всі HTML-документи та пов’язані з ними ресурси, включаючи зображення, таблиці стилів CSS, файли JavaScript, шрифти та відео.

**Communicating through HTTP**

По-друге, веб-сервер забезпечує підтримку HTTP, який:

* Текстовий. Усі команди є відкритим текстом і зрозумілі людині.
* Без громадянства. Ні сервер, ні клієнт не пам'ятають попередніх зв'язків. Наприклад, покладаючись лише на HTTP, сервер не може запам’ятати пароль, який ви ввели, або крок, на якому ви перебуваєте в транзакції. Для таких завдань вам потрібен сервер додатків

Тільки клієнти можуть робити HTTP-запити, і то тільки до серверів. Сервери можуть лише відповідати.

Запитуючи файл через HTTP, клієнти повинні надати URL-адресу файлу.

Веб-сервер повинен відповідати на кожен запит HTTP, принаймні повідомленням про помилку.

**Optimizing NGINX Speed for Serving Content**

**Увімкнення sendfile**. За замовчуванням NGINX самостійно обробляє передачу файлів і копіює файл у буфер перед його надсиланням. Увімкнення директиви sendfile скасує крок копіювання даних у буфер і дозволить пряме копіювання даних з одного файлового дескриптора до іншого.

Крім того, щоб запобігти тому, щоб одне швидке з’єднання повністю займало робочий процес, ви можете обмежити обсяг даних, що передаються в одному виклику sendfile(), визначивши директиву sendfile\_max\_chunk.

Використання AJAX дозволяє JavaScript на веб-сторінці надсилати HTTP-запити на сервер, не залишаючи сторінки.

Інтерактивні веб-програми можуть використовувати цю техніку для оновлення веб-сторінки, не викликаючи відволікаючого мерехтіння, яке спостерігається, коли браузер завантажує нову сторінку.

**Sending Parameters with AJAX**

AJAX часто називають механізмом RPC, у якому створюється певний список параметрів для виклику на стороні сервера, а результати повертаються у зручному форматі, такому як XML, CSV або JSON.

Багато розробників використовують бібліотеки для вирішення проблем кросбраузерності та спрощення того, що по суті є «шаблонним» кодом. Одним із найпоширеніших є jQuery, який забезпечує підтримку AJAX, а також багато інших функцій.

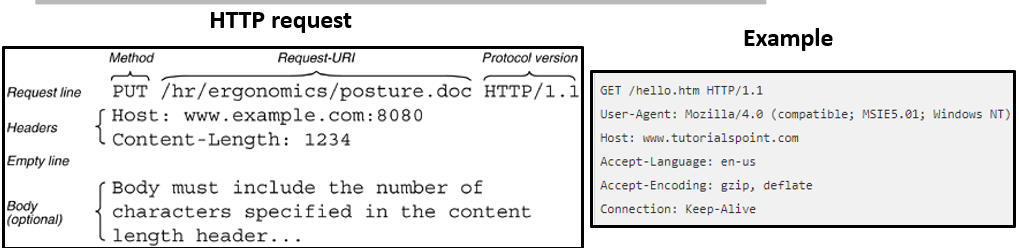
Майже будь-який елемент HTML може бути цільовим одержувачем запиту AJAX у jQuery.

**Hypertext Transfer Protocol (HTTP)**

HTTP базується на TCP/IP комунікаційному протоколі, який служить для того, щоб доставляти дані(різні файли програми) у World Wide Web.

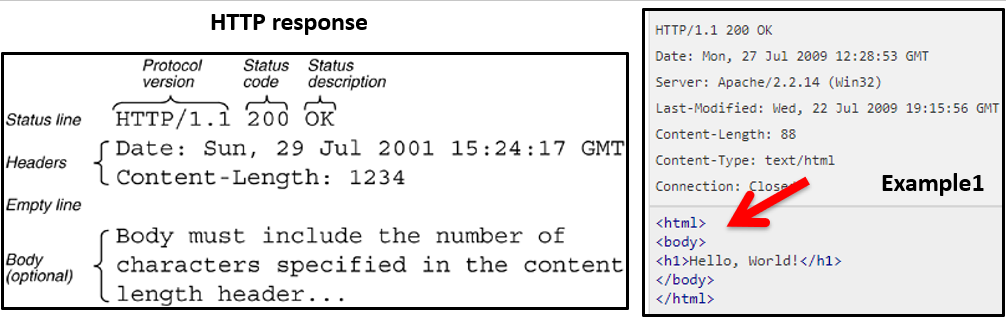
**Властивості:**

* HTTP не підтримує зєднання, тобто після запиту клієнт відєднується, і коли в сервера є відповідь, повторно встановлює зяднання
* HTTP не залежить від медіа, тобто чз нього можна надсилати файли будь якого типу, головне щоб клієнт і сервер вміли обробляти ці файли. Необхідно, щоб клієнт, а також сервер вказали тип вмісту за допомогою відповідного типу MIME
* HTTP не має статусу, тобто сервер і клієнт знають один про одного тільки під час самого зєднання. Тому ні клієнт, ні браузер не можуть зберігати інформацію між різними запитами на веб-сторінках



**Request-URI**:

* Зірочка \* використовується, коли запит HTTP стосується не певного ресурсу, а самого сервера.
* AbsoluteURI використовується, коли HTTP-запит надсилається до проксі. Проксі запитується переслати запит або послугу з дійсного кешу та повернути відповідь.
* Визначте ресурс на вихідному сервері або шлюзі



**Status-Code**

* **1xx: Інформаційний**. Запит отримано, процес триває.
* **2xx: Успіх.** Дія була успішно прийнята, зрозуміла та прийнята.
* **3xx: Перенаправлення**. Необхідно виконати подальші дії, щоб завершити запит.
* **4xx: помилка клієнта**. Запит містить неправильний синтаксис або його неможливо виконати.
* **5xx: Помилка сервера**. Сервер не зміг виконати очевидно дійсний запит

**Методи:**

GET – витягує різні дані з веб серверу по вказаних параметрах в URl

**HTTP header fields**

**Загальний заголовок -** поля заголовка мають загальне застосування як для повідомлень запиту, так і для відповіді.

**Заголовок запиту** клієнта

**Заголовок відповіді** клієнта

**Заголовок сутності**: ці поля заголовка визначають метаінформацію про тіло сутності або, якщо тіла немає, про ресурс, визначений у запиті.

**Some Important General Headers**

Поле заголовка Cache-Control використовується для визначення директив, які ПОВИННІ виконуватися всією системою кешування

Поле заголовка підключення дозволяє відправнику вказати параметри, які потрібні для цього конкретного підключення. Наприклад:

***Connection: close***

***Connection: keep-alive***

Заголовок Upgrade дозволяє клієнту вказати, які додаткові протоколи зв’язку він підтримує та хотів би використовувати, якщо сервер визнає за доцільне перемикати протоколи.

Поле заголовка Transfer-Encoding вказує, який тип трансформації було застосовано до тіла повідомлення, щоб безпечно передати його між відправником і одержувачем.

**Some Important Client Request Headers**

Поле заголовка «**Accept**» можна використовувати для визначення певних типів носіїв, прийнятних для відповіді.

Поле заголовка Accept-Encoding схоже на Accept, але обмежує кодування вмісту, прийнятне у відповіді

Значення поля заголовка ***Authorization*** складається з облікових даних, що містять інформацію автентифікації агента користувача для області запитуваного ресурсу.

Значення поля заголовка Cookie містить пару ім’я/значення інформації, що зберігається для цієї URL-адреси: Cookie: name1=value1;name2=value2

Поле заголовка Host використовується для вказівки Інтернет-хосту та номера порту (80 за замовчуванням) запитуваного ресурсу.

**Some Important Server Response Headers**

Поле заголовка «**Location**» використовується для перенаправлення одержувача в інше місце, ніж URI запиту для завершення

Поле заголовка « **Server** » містить інформацію про програмне забезпечення, яке використовує вихідний сервер для обробки запиту

Поле заголовка Set-Cookie містить інформацію про назву та значення (і деякі параметри), яку потрібно зберегти для цієї URL-адреси

**Some Important Entity Headers**

У полі заголовка «**Allow**» перелічено набір методів, які підтримує ресурс, визначений Request-URI

Заголовок Content-Length вказує на розмір тіла сутності

Заголовок Expires містить дату/час, після яких відповідь вважається застарілою

**Transfer sizes and number of objects**

**Latency**. Одним із випадків, коли справді потрібна низька затримка, є певні види відео, як-от відеоконференції, ігри тощо, де немає лише попередньо створеного потоку для надсилання.

**Head of line blocking**. HTTP Pipelining — це спосіб надсилання іншого запиту під час очікування відповіді на попередній запит. Це дуже схоже на чергу біля каси в банку чи в супермаркеті.

**Things done to overcome latency: Spriting, Inlining** (закодовані файли base64 у css), **Concatenation** (об’єднання js-файлів), **Sharding** (кількість підключень до одного хосту обмежена: 6-8, використовується кілька хостів).

**HTTP2**

http2 — це двійковий протокол, який значно полегшує кадрування.

HTTP/2 жертвує частиною цієї простоти заради ефективності.

Основною метою якого є підвищення ефективності запитів на додаткові ресурси

**Multiplexing**. Ідентифікатор потоку, згаданий у попередньому розділі, пов’язує кожен кадр, надісланий через http2, із «потоком». Потік — це незалежна двонаправлена послідовність кадрів, якими обмінюються клієнт і сервер у рамках з’єднання http2.

**Multiplexing the streams** означає, що пакети з багатьох потоків змішуються через одне з’єднання. Важливим є порядок, у якому кадри надсилаються в межах потоку.

**Priorities and dependencies**. Кожен потік також має пріоритет (також відомий як «вага»). Клієнт також може повідомити серверу, від якого іншого потоку залежить цей потік. Це дозволяє клієнту побудувати «дерево» пріоритетів, де кілька «дочірніх потоків» можуть залежати від завершення «батьківських потоків»

**Header compression**. HTTP — це протокол без збереження стану

**Compression problems**. Стиснення HTTPS виявилося вразливим до атак BREACH і RIME

**Reset**. Одним із недоліків HTTP 1.1 є те, що коли HTTP-повідомлення було надіслано з Content-Length певного розміру, ви не можете просто зупинити його.

**Сервер push**. Ця функція також відома як «кеш-пам’ять». Ідея полягає в тому, що якщо клієнт запитує ресурс X, сервер може знати, що клієнт, ймовірно, також захоче отримати ресурс Z, і надсилає його клієнту без запиту. Сервер push — це те, що клієнт повинен явно дозволити серверу

**Flow Control**. Кожен окремий потік http2 має власне оголошене вікно потоку, для якого інша сторона може надсилати дані. Це вікно можна розширити.

**Alternative Services**. Сервер надішле заголовок Alt-Svc:, повідомляючи клієнту про альтернативну службу: інший маршрут до того самого вмісту, використовуючи іншу службу, хост і номер порту.

**HTTPS**

***Secure Sockets Layer (SSL)*** розроблено для шифрування будь-якого мережевого трафіку на основі TCP/IP і забезпечує такі можливості:

* Запобігає прослуховування
* Запобігає підробці або повторному відтворенню повідомлень
* Використовує сертифікати для автентифікації серверів і, за бажанням, клієнтів

**What Is Encryption?**

Шифрування – за допомогою математичних функцій перетворює звичайний текст у зашифрований.

Шифр зазвичай залежить від частини допоміжної інформації, яка називається ключем.

Симетричне, також відоме як шифрування закритим ключем.

Асиметричне, також відоме як шифрування з відкритим ключем

**Digital Signature**

Цифровий підпис — це зашифрований дайджест повідомлення.

Дайджест повідомлення створюється:

Отримання повідомлення та використання алгоритму хешування для обчислення рядка зазвичай довжиною 128 біт. Цей рядок називається дайджестом повідомлення. (MD5 — поширений алгоритм хешування).

Дайджест повідомлення зашифровано закритим ключем відправника.

Цифровий підпис служить трьом важливим цілям:

* Перевіряє цілісність даних
* Забезпечує автентифікацію відправника.
* Відповідальність
* Втрата закритого ключа означає, що всі комунікації з цифровим підписом все ще можуть бути відхилені.
* Додавання цифрового підпису запобігає відмові

**Digital Certificate**

Цифровий сертифікат використовується для підтвердження того, що користувач, який надсилає повідомлення, є тим, за кого себе видає, і для надання одержувачу засобів для кодування відповіді.

Ті, хто бажає надсилати зашифровані повідомлення, отримують цифровий сертифікат від центру сертифікації

Орган сертифікації видає зашифрований цифровий сертифікат.

Одержувач зашифрованого повідомлення використовує відкритий ключ центру сертифікації для декодування цифрового сертифіката, доданого до повідомлення.

Одержувач перевіряє його як видане сертифікуючим органом.

Потім він отримує відкритий ключ відправника та ідентифікаційну інформацію, що зберігається в цифровому сертифікаті.

Маючи цю інформацію, одержувач може надіслати зашифровану відповідь

Цифровий сертифікат зазвичай містить усе наведене нижче:

* Термін придатності.
* Серійний номер.
* Цифровий сертифікат органу сертифікації.
* Цифровий підпис центру сертифікації.
* Найменування органу сертифікації.
* Відкритий ключ центру сертифікації.
* Цифровий підпис власника.
* Адреса електронної пошти власника.
* Поштова адреса власника.
* Відкритий ключ власника.

**Man-In-The-Middle Attack**

Атака Man-In-The-Middle — це коли зловмисник може читати, вставляти та змінювати за бажанням повідомлення між двома сторонами, не знаючи жодної сторони, що зв’язок між ними зламано.

Атака Man-In-The-Middle легко працює на протоколі узгодження ключів, якщо використовується без автентифікації (орган сертифікації може забезпечити автентифікацію).

**SSL Security**

Веб-сервер, на якому працює SSL, забезпечує:

* Конфіденційність (запобігання прослуховування).
* Автентифікація (запобігання видаванню себе за іншу особу).
* Цілісність (запобігання модифікації).

SSL використовує різні алгоритми, кожен з яких має інший ключ, щоб досягти всіх трьох.

* Використання приватного ключа під час сеансів шифрування.
* Використання шифрування з відкритим ключем для автентифікації клієнтів і серверів.
* Використання приватного ключа для шифрування даних програми.

**SSL Record Protocol**

Протокол SSL Record дозволяє інкапсуляцію протоколів вищого рівня, таких як:

* Протокол рукостискання SSL
* Протокол сповіщень SSL
* HTTP

**SSL Alert Protocol**

Цей протокол повідомляє про проблеми

Основні сповіщення про помилки:

**unexpected\_message**. Отримано невідповідне повідомлення. Це сповіщення завжди є фатальним і ніколи не повинно спостерігатися під час зв’язку між належними реалізаціями.

**bad\_record\_mac**. Це сповіщення повертається, якщо отримано запис із неправильним кодом автентифікації повідомлення. Це повідомлення завжди фатальне.

**decompression\_failure**. Функція декомпресії отримала неправильні вхідні дані (наприклад, дані, які будуть розширені до надмірної довжини). Це повідомлення завжди фатальне.

**handshake\_failure**. Вказує, що відправник не зміг узгодити прийнятний набір параметрів безпеки з урахуванням доступних варіантів. Це фатальна помилка.

**no\_certificate**. Може бути надіслано у відповідь на запит сертифікації, якщо відповідний сертифікат недоступний.

**bad\_certificate**. Сертифікат був пошкоджений, ймовірно, містив цифровий підпис, який не було перевірено належним чином.

**WebSocket**

**WebSocket** — це комп’ютерний протокол зв’язку, який забезпечує повнодуплексні канали зв’язку через одне з’єднання TCP.

Протокол WebSocket є незалежним протоколом на основі TCP. Його єдине відношення до HTTP полягає в тому, що його рукостискання інтерпретується HTTP-серверами як запит на оновлення.

Клієнт встановлює з’єднання WebSocket за допомогою процесу, відомого як рукостискання WebSocket. Цей процес починається з того, що клієнт надсилає стандартний HTTP-запит на сервер. Заголовок Upgrade додається до цього запиту, який інформує сервер про те, що клієнт бажає встановити з’єднання WebSocket

Якщо сервер підтримує протокол WebSocket, він погоджується на оновлення та повідомляє про це через заголовок Upgrade у відповіді

Якщо доступ до сторінки чз HTTP запити, то можна використовувати звичайний WS, або захищений(WSS). Але якщо це HTTPS, тоді тільки WSS.

**Core JavaScript**

Мова високого рівня, динамічна, нетипізована інтерпретована, добре підходить для стилів об'єктно-орієнтованого та функціонального програмування.

Чутлива до регістру(HTML ні)

Літерал — це значення даних, яке з’являється безпосередньо в програмі

Ідентифікатор JavaScript має починатися з літери, підкреслення (\_) або знака долара ($).

**Types, Values, and Variables**

JavaScript представляє числа в 64-розрядному форматі з плаваючою комою

Будь-яке значення JavaScript можна перетворити на логічне значення

null — це ключове слово мови, яке обчислюється спеціальним значенням, яке зазвичай використовується для позначення відсутності значення.

undefined значення представляє більш глибокий вид відсутності. Це значення змінних, які не були ініціалізовані, і значення, яке ви отримуєте, коли запитуєте значення властивості об’єкта або елемента масиву, який не існує.

**The Global Object**

Глобальний об’єкт — це звичайний об’єкт JavaScript, який виконує дуже важливу мету: властивості цього об’єкта — це глобально визначені символи, які доступні для програми JavaScript.

Коли інтерпретатор JavaScript запускається (або щоразу, коли веб-браузер завантажує нову сторінку), він створює новий глобальний об’єкт і надає йому початковий набір властивостей, які визначають:

* глобальні властивості, такі як undefined, Infinity і NaN
* глобальні функції, такі як isNaN(), parseInt() і eval()
* такі функції конструктора, як Date(), RegExp(), String(), Object() і Array()
* глобальні об’єкти, такі як Math і JSON

У JavaScript на стороні клієнта об’єкт Window служить глобальним об’єктом для всього коду JavaScript, який міститься у вікні браузера, яке воно представляє. Цей глобальний об’єкт Window має самопосилальну властивість вікна, яку можна використовувати замість цього для посилання на глобальний об’єкт. Об’єкт Window визначає основні глобальні властивості, але він також визначає чимало інших глобальних параметрів, які є специфічними для веб-браузерів і клієнтського JavaScript.

**Wrapper Objects**

String не є об’єктами, то чому вони мають властивості? Кожного разу, коли ви намагаєтеся звернутися до властивості рядка s, JavaScript перетворює значення рядка в об’єкт, ніби викликаючи новий String(s).

Числа та логічні значення мають методи з тієї ж причини, що й рядки: тимчасовий об’єкт створюється за допомогою конструктора Number() або Boolean(), і метод вирішується за допомогою цього тимчасового об’єкта.

Рядки, числа та логічні значення поводяться як об’єкти, коли ви намагаєтеся прочитати з них значення властивості (або методу). Але якщо ви намагаєтеся встановити значення властивості, ця спроба мовчки ігнорується: зміна вноситься на тимчасовий об’єкт і не зберігається.

Зауважте, що можливо (але майже ніколи не потрібно чи корисно) створювати об’єкти-огортки

**Immutable Primitive Values and Mutable Object References**

Примітивні(числа, null, ..) незмінні. Але можна змінити String, наприклад(бо це масив символів). Але це не зовсім зміна, це поверення нового рядка

Обєкти можна змінювати, і їх не можна порівняти

Об’єкти іноді називають посилальними типами, щоб відрізнити їх від примітивних типів JavaScript. Об’єкти порівнюються за посиланням: два значення об’єкта однакові тоді і тільки тоді, коли вони посилаються на той самий основний об’єкт

**Conversions and Equality(Перетворення та рівність)**

Строгий оператор рівності ===, який не виконує перетворення під час тестування на рівність

Оператор if перетворює undefined на false, але оператор == ніколи не намагається перетворити свої операнди на логічні значення.

Найпростіший спосіб виконати явне перетворення типу - це використовувати такі функції

Деякі оператори JavaScript виконують неявне перетворення типів і іноді використовуються для цілей перетворення типів. Якщо один операнд оператора + є рядком, він перетворює інший на рядок. Унарний оператор + перетворює свій операнд на число. І унарний! оператор перетворює свій операнд на логічний і заперечує його.

Перетворення об’єктів у логічні значення є тривіальним: усі об’єкти (включно з масивами та функціями) перетворюються на істинні. Це стосується навіть об’єктів-огорток

Перетворення об’єкта в рядок і об’єкта в число виконуються шляхом виклику методу об’єкта, який потрібно перетворити (toString(), valueOf()). Хост-об’єкти (визначені, наприклад, веб-браузерами) можуть перетворюватися на числа та рядки відповідно до власних алгоритмів.

***To convert an object to a string, JavaScript takes these steps:***

1. Якщо об’єкт має метод toString(), JavaScript викликає його. Якщо він повертає примітивне значення, JavaScript перетворює це значення на рядок (якщо воно ще не є рядком) і повертає результат цього перетворення.
2. Якщо об’єкт не має методу toString() або якщо цей метод не повертає примітивне значення, тоді JavaScript шукає метод valueOf(). Якщо метод існує, JavaScript викликає його. Якщо значення, що повертається, є простим, JavaScript перетворює це значення на рядок (якщо воно ще не є) і повертає перетворене значення.
3. Інакше JavaScript не може отримати примітивне значення ні з toString(), ні з valueOf(), тому видає TypeError

***To convert an object to a number, JavaScript does the same thing, but it tries the  
valueOf() method first:***

1. Якщо об’єкт має метод valueOf(), який повертає примітивне значення, JavaScript перетворює (якщо необхідно) це примітивне значення на число та повертає результат.
2. В іншому випадку, якщо об’єкт має метод toString(), який повертає примітивне значення, JavaScript перетворює та повертає значення.
3. Інакше JavaScript видає TypeError

**Repeated and Omitted Declarations(Повторні та пропущені декларації)**

У нестрогому режимі, якщо ви присвоюєте значення неоголошеній змінній, JavaScript фактично створює цю змінну як властивість глобального об’єкта, і вона працює так само, як правильно оголошена глобальна змінна.

Однак це погана звичка та джерело помилок, і ви завжди повинні оголошувати свої змінні за допомогою var

**Variable Scope**

JavaScript використовує область видимості функції: змінні видимі у функції, у якій вони визначені, і в будь-яких функціях, вкладених у цю функцію.

Область видимості функції JavaScript означає, що всі змінні, оголошені у функції, видимі в усьому тілі функції. Цікаво, що це означає, що змінні навіть видимі до того, як вони оголошені. Ця функція JavaScript неофіційно відома як підйом: код JavaScript поводиться так, ніби всі оголошення змінних у функції (але не будь-які пов’язані призначення) «підняті» до верхньої частини функції.

Якщо ви використовуєте var для оголошення змінної, властивість, яка створюється, не налаштовується, що означає, що її не можна видалити за допомогою оператора delete.

JavaScript дозволяє посилатися на глобальний об’єкт за допомогою ключового слова this, але не дає нам жодного способу посилатися на об’єкт, у якому зберігаються локальні змінні

**The Scope Chain(Ланцюжок масштабів)**

Коли ця функція викликається, вона створює новий об’єкт для зберігання своїх локальних змінних і додає цей новий об’єкт до збереженого ланцюжка областей, щоб створити новий, довший ланцюжок, який представляє область для виклику цієї функції.

Це стає більш цікавим для вкладених функцій, оскільки кожного разу, коли викликається зовнішня функція, внутрішня функція визначається знову. Оскільки ланцюжок області видимості відрізняється під час кожного виклику зовнішньої функції, внутрішня функція дещо відрізнятиметься кожного разу, коли вона визначається — код внутрішньої функції буде ідентичним під час кожного виклику зовнішньої функції, але ланцюжок області видимості, пов’язаний із цією код буде іншим.

**The delete operation**

деякі вбудовані властивості ядра та клієнтські властивості захищені від видалення, а визначені користувачем змінні, оголошені оператором var, не можна видалити.

Функції, визначені за допомогою оператора функції та оголошених параметрів функції, також не можна видалити.

**Objects in details**

З кожним об’єктом JavaScript пов’язаний другий об’єкт JavaScript (або нульовий, але це рідко). Цей другий об’єкт відомий як **прототип**, а перший об’єкт успадковує властивості від прототипу

зв'язаний ряд об'єктів-прототипів відомий як **ланцюг прототипів**

Оператор delete видаляє лише власні властивості, а не успадковані. (Щоб видалити успадковану властивість, ви повинні видалити її з об’єкта прототипу, у якому вона визначена. Це впливає на кожен об’єкт, який успадковує цей прототип)

**Property Attributes**

Окрім назви та значення, властивості мають атрибути, які вказують, чи можна їх писати, нумерувати та налаштовувати. У ECMAScript 3 немає способу встановити ці атрибути: усі властивості, створені програмами ECMAScript 3, доступні для запису, перерахування та конфігурації, і неможливо це змінити.

**Arrays**

Рядки поводяться як масиви, доступні лише для читання. Такі методи масиву, як push(), sort(), reverse() і splice(), змінюють масив на місці й не працюють із рядками. Спроба змінити рядок за допомогою методу масиву, однак, не викликає помилки: вона просто зазнає невдачі!

**Functions**

На додаток до аргументів, кожен виклик має ще одне значення — **контекст виклику**, тобто значення ключового слова this. Зауважте, що це ключове слово, а не ім’я змінної чи властивості. Синтаксис JavaScript не дозволяє призначити цьому значення

Оскільки функції є об’єктами, ви можете встановлювати властивості для них і навіть викликати для них методи

Вкладені функції не успадковують this значення свого виклику

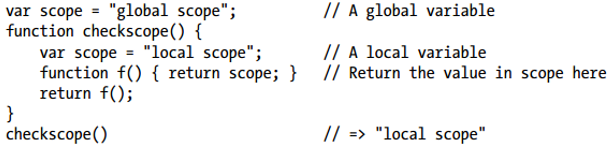
Функції JavaScript — це об’єкти, і, як і всі об’єкти JavaScript, вони мають методи. Два з цих методів, call() і apply(), викликають функцію опосередковано.

Коли функція викликається з меншою кількістю аргументів, ніж оголошених параметрів, додаткові параметри встановлюються на невизначене значення. У тілі функції аргументи ідентифікатора посилаються на об’єкт Arguments для цього виклику.

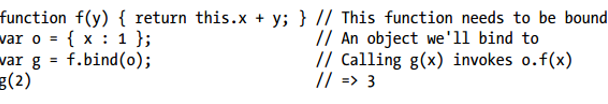
Об’єкт Arguments визначає властивості callee і caller, властивість callee посилається на поточну запущену функцію, caller є нестандартною, але загальновживаною властивістю, яка посилається на функцію, яка викликала цю

**Closures(Закриття)**

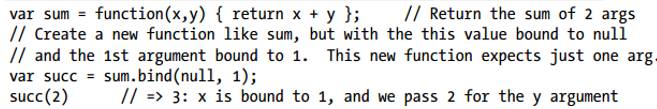
функції JavaScript виконуються з використанням ланцюжка областей видимості, який діяв на момент їх визначення. Вкладену функцію f() було визначено в ланцюжку області видимості, в якій змінна scope була прив’язана до значення «локальна область». Це зв’язування все ще діє, коли f виконується, незалежно від того, звідки він виконується. Тож останній рядок коду вище повертає «локальний діапазон», а не «глобальний контроль»



**The bind() Method (ECMAScript 5)**

Коли ви викликаєте метод bind() у функції f і передаєте об’єкт o, метод повертає нову функцію. Виклик нової функції (як функції) викликає вихідну функцію f як метод

Метод bind() виконує часткове застосування (каррінг у функціональному програмуванні): усі аргументи, які ви передаєте bind() після першого, пов’язуються разом із значенням this.



**Classes and Prototypes**

Клас JavaScript — це набір об’єктів, які успадковують властивості від того самого об’єкта-прототипу

Два об'єкти є екземплярами одного класу тоді і тільки тоді, коли вони успадковують від того самого об'єкта-прототипу. Функція-конструктор, яка ініціалізує стан нового об’єкта, не є фундаментальною: дві функції-конструктор можуть мати властивості прототипу, які вказують на той самий об’єкт-прототип. Тоді обидва конструктори можна використовувати для створення екземплярів одного класу.

Будь-яку функцію JavaScript можна використовувати як конструктор, а виклики конструктора потребують властивості прототипу. Тому кожна функція JavaScript (крім функцій, які повертає метод ECMAScript 5 Function.bind()) автоматично має властивість прототипу

**Augmenting Classes (Розширення класів)**

Механізм успадкування на основі прототипу JavaScript є динамічним: об’єкт успадковує властивості від свого прототипу, навіть якщо прототип змінюється після створення об’єкта. Це означає, що ми можемо розширювати класи JavaScript, просто додаючи нові методи до їхніх об’єктів-прототипів.

**Classes and Types**

Нагадаємо, JavaScript визначає невеликий набір типів: null, undefined, boolean, number, string, function і object.

Жоден із описаних вище методів визначення класу об’єкта не є безпроблемним, принаймні в JavaScript на стороні клієнта.

Альтернативою є обійти проблему: замість того, щоб запитувати «який клас цього об’єкта?» натомість ми запитуємо: «Що може робити цей об’єкт?»

Цей підхід до програмування поширений у таких мовах, як Python і Ruby, і називається качиним типом за цим виразом (який часто приписують поету Джеймсу Віткомбу Райлі)

Якщо об’єкт має метод toJSON(), JSON.stringify() не серіалізує об’єкт, а замість цього викликає toJSON() і серіалізує значення (примітивне або об’єктне), яке повертає.

Оператори рівності JavaScript порівнюють об’єкти за посиланням, а не за значенням.

Щоб увімкнути перевірку екземплярів вашого класу на рівність, визначте метод екземпляра під назвою equals(). Він має приймати один аргумент і повертати істину, якщо цей аргумент дорівнює об’єкту, до якого він викликається.

Метод compareTo() повинен приймати один аргумент і порівнювати його з об’єктом, для якого метод викликається. Якщо цей об’єкт менший за аргумент, compareTo() має повернути значення менше нуля.

**Borrowing Methods(Способи запозичення)**

Одну функцію можна навіть використовувати як метод кількох класів.

Більшість вбудованих методів класу Array, наприклад, визначені узагальнено, і якщо ви визначаєте клас, екземпляри якого є об’єктами, подібними до масиву, ви можете скопіювати функції з Array.prototype в об’єкт-прототип вашого класу.

Якщо розглядати JavaScript через призму класичних об’єктно-орієнтованих мов, використання методів одного класу як методів іншого класу можна розглядати як форму множинного успадкування.

Однак JavaScript не є класичною об’єктно-орієнтованою мовою, і для опису повторного використання цього методу ми можемо використовувати неофіційний термін запозичення.

**Private State**

Ми можемо апроксимувати поля приватного екземпляра за допомогою змінних (або аргументів), зафіксованих у закриття виклику конструктора, який створює екземпляр. Для цього ми визначаємо функції всередині конструктора (щоб вони мали доступ до аргументів і змінних конструктора) і призначаємо ці функції властивостям новоствореного об’єкта.

**Subclasses**

Ключем до створення підкласів у JavaScript є правильна ініціалізація об’єкта-прототипу. Якщо клас B продовжує A, то B.prototype має бути спадкоємцем A.prototype

ECMAScript 5 дозволяє нам більш надійно інкапсулювати наші змінні стану, визначаючи методи отримання та встановлення властивостей, які не можна видалити.

**Preventing Class Extensions**

Зазвичай вважається особливістю JavaScript те, що класи можна динамічно розширювати шляхом додавання нових методів до об’єкта-прототипу.

ECMAScript 5 дозволяє запобігти цьому, якщо ви хочете. Object.preventExtensions() робить об’єкт нерозширюваним, що означає, що до нього не можна додавати нові властивості.

Object.seal() робить це ще далі: він запобігає додаванню нових властивостей, а також робить усі поточні властивості неконфігурованими, тому їх неможливо видалити.

Ще одна динамічна особливість JavaScript — це можливість замінювати (або «виправляти») методи об’єкта

Ви можете запобігти подібним змінам, зробивши свої методи екземпляра доступними лише для читання. Object.freeze() виконує те ж саме, що й Object.seal(), але також робить усі властивості доступними лише для читання та неконфігурованими.

Якщо ви хочете перевизначити успадковану властивість лише для читання, вам потрібно використовувати Object.defineProperty() або Object.defineProperties() або Object.create(), щоб створити нову властивість

**Modules**

Мета модулів — дозволити складати великі програми за допомогою коду з різних джерел

Модулі мають загальнодоступний API, який вони експортують: це функції, класи, властивості та методи, призначені для використання іншими програмістами.

Модулі, які мають більше ніж один елемент у своєму API, можуть повертати об’єкт простору імен.

Як альтернатива, якщо глобальний об’єкт простору імен уже визначено, функція модуля може просто встановити властивості цього об’єкта напряму, і взагалі нічого не повертати.

Регулярний вираз — це об’єкт, який описує шаблон символів

**Client JS**

Деякі веб-сторінки представляють статичну інформацію, і їх можна назвати документами. (Подання цієї статичної інформації може бути досить динамічним — через JavaScript, — але сама інформація є статичною.)

Інші веб-сторінки більше нагадують програми, ніж документи

Об’єкт Window є основною точкою входу до всіх функцій JavaScript і API на стороні клієнта. Він представляє вікно веб-браузера або фрейм, і ви можете посилатися на нього за допомогою вікна ідентифікатора. Об’єкт Window визначає такі властивості, як location, який посилається на об’єкт Location, який визначає URL-адресу, яка зараз відображається у вікні, і дозволяє сценарію завантажити нову URL-адресу у вікно:

Зауважте, що наведений вище код явно не використовує властивість window. У JavaScript на стороні клієнта об’єкт Window також є глобальним об’єктом. Це означає, що об'єкт Window знаходиться у верхній частині ланцюга видимості.

**Window object**

Однією з найважливіших властивостей об’єкта Window є документ: він посилається на об’єкт Document, який представляє вміст, що відображається у вікні. Об’єкт Document має такі важливі методи, як getElementById().

Кожен об’єкт Element має властивості style та className, які дозволяють сценаріям визначати стилі CSS для елемента документа або змінювати імена класів CSS, які застосовуються до елемента

Іншим набором важливих властивостей об’єктів «Вікно», «Документ» і «Елемент» є властивості обробника подій. Обробники подій дозволяють коду JavaScript змінювати поведінку вікон, документів і елементів, які складають ці документи.

Одним із найважливіших обробників подій є обробник **onload** об’єкта Window. Він запускається, коли вміст документа, що відображається у вікні, стабільний і готовий до маніпулювання. Код JavaScript зазвичай міститься в обробнику події onload

**JavaScript in Web Documents**

Програма JavaScript може керувати вмістом документа через об’єкт Document і об’єкти Element, які він містить. Він може змінювати подання цього вмісту, створюючи стилі та класи CSS.

І він може визначати поведінку елементів документа, реєструючи відповідні обробники подій.

Комбінація вмісту, презентації та поведінки, що піддається сценарію, називається динамічним HTML або DHTML

**JS in Web App**

Маючи на увазі це поняття веб-браузера як спрощеної ОС, ми можемо визначити веб-програми як веб-сторінки, які використовують JavaScript для доступу до більш розширених послуг (таких як мережеві, графічні та зберігання даних), які пропонують браузери.

Найвідомішим із цих розширених сервісів є об’єкт XMLHttpRequest, який забезпечує роботу в мережі за допомогою сценаріїв HTTP-запитів. Веб-сторінки з додатками Ajax утворюють основу того, що відомо як «Веб 2.0».

Специфікація HTML5 визначає API для зберігання графічних даних, геолокації, керування історією та фонових потоків.

Добре розроблений веб-документ продовжуватиме працювати з вимкненим JavaScript. Веб-програми, за визначенням, є програмами JavaScript, які використовують служби типу ОС, надані веб-браузером, і не очікується, що вони працюватимуть з вимкненим JavaScript.

**Embedding JavaScript in HTML**

Код JavaScript на стороні клієнта вбудовується в документи HTML чотирма способами:

* Вбудований, між парою тегів <script> і </script>
* Із зовнішнього файлу, визначеного атрибутом src тегу <script>
* В атрибуті обробника подій HTML, наприклад onclick або onmouseover
* В URL-адресі, яка використовує спеціальний протокол javascript

**Execution of JavaScript Programs(Виконання програм на JavaScript)**

Усі ці окремі фрагменти коду спільно використовують єдиний глобальний об’єкт Window. Це означає, що всі вони бачать той самий об’єкт Document і мають спільний набір глобальних функцій і змінних

**Якщо веб-сторінка містить вбудований фрейм (за допомогою елемента <iframe>), код JavaScript у вбудованому документі має інший глобальний об’єкт, ніж код у документі для вбудовування, і його можна вважати окремою програмою JavaScript.**

Якщо документ-контейнер і документ, що міститься, походять з одного сервера, код в одному документі може взаємодіяти з кодом в іншому, і ви можете розглядати їх як дві взаємодіючі частини однієї програми, якщо хочете.

Scripts зазвичай виконуються в тому порядку, в якому вони з’являються в документі.

Після завантаження документа та виконання всіх сценаріїв виконання JavaScript переходить у другу фазу. Цей етап є асинхронним і керується подіями. Під час цієї фази, керованої подіями, веб-браузер викликає обробник подій у відповідь на події, які відбуваються асинхронно

Коли синтаксичний аналізатор HTML зустрічає елемент <script>, він повинен за замовчуванням запустити сценарій, перш ніж він зможе відновити синтаксичний аналіз і відтворення документа (якщо вихідний код сценарію міститься у зовнішньому файлі, указаному атрибутом src, це означає, що частини документа, які слідують за сценарієм, не відображатимуться у браузері, доки сценарій не буде завантажено та виконано).

атрибути **defer** і **async** — це способи повідомити браузеру, що пов’язаний сценарій не використовує document.write() і не буде генерувати вміст документа, і тому браузер може продовжувати аналізувати та відтворювати документ під час завантаження сценарію.

Атрибут defer змушує браузер відкладати виконання сценарію до моменту, коли документ буде завантажено та проаналізовано та готовий до роботи.

Атрибут async змушує браузер запускати сценарій якнайшвидше, але не блокувати розбір документа під час завантаження сценарію.

Якщо тег <script> має обидва атрибути, браузер, який підтримує обидва, враховуватиме атрибут async і ігноруватиме атрибут defer.

**Зверніть увагу, що deferred сценарії виконуються в тому порядку, в якому вони з’являються в документі. Асинхронні сценарії запускаються під час завантаження, що означає, що вони можуть виконуватися не по порядку.**

Ви можете завантажувати та виконувати сценарії асинхронно, навіть у браузерах, які не підтримують атрибут async, динамічно створюючи елемент <script> і вставляючи його в документ.

**Event-Driven JavaScript**

Функції обробника подій. Потім ці функції викликаються асинхронно, коли відбуваються події, для яких вони були зареєстровані.

Події, цілями яких є елементи в документі, часто поширюються вгору по дереву документів у процесі, відомому як «спливання». Якщо користувач клацає мишкою на елементі <button>, на кнопці запускається подія клацання. Якщо ця подія не обробляється (і її розповсюдження зупиняється) функцією, зареєстрованою на кнопці, подія випливає до будь-якого елемента, у який вкладена кнопка, і буде викликано будь-який обробник події клацання, зареєстрований у цьому елементі контейнера.

Якщо ви хочете написати модуль коду, який може безпечно реєструвати обробники подій, навіть якщо інший модуль уже зареєстрував обробник для тієї самої події на тій же цілі, ви повинні використовувати метод під назвою addEventListener(), який дозволяє реєструвати кілька слухачі

Функції, передані в setTimeout(), реєструються інакше, ніж справжні обробники подій, і їх зазвичай називають «зворотними викликами» замість «обробників», але вони асинхронні, як і обробники подій.

**Client-Side JavaScript Threading Model**

Ви можете маніпулювати вмістом документа, знаючи, що жоден інший потік не намагається змінити його в той самий час, і вам ніколи не потрібно турбуватися про блокування, тупикові блокування чи умови конкуренції.

Однопотокове виконання означає, що веб-браузери повинні припинити реагувати на введення користувача під час виконання сценаріїв і обробників подій. Тому обробники подій не повинні працювати надто довго.

Якщо є можливість розбити довготривале обчислення на окремі підзавдання, ви можете використовувати такий метод, як setTimeout(), щоб запускати підзадачі у фоновому режимі, оновлюючи індикатор виконання для користувача.

HTML5 визначає «веб-працівника», який є фоновим потоком для виконання інтенсивних обчислювальних завдань без зависання інтерфейсу користувача. Код, який виконується в робочому веб-потоці, не має доступу до вмісту документа, не ділиться жодним станом з основним потоком або іншими робочими потоками та може спілкуватися з основним потоком та іншими робочими лише через асинхронні події.

**Client-Side JavaScript Timeline in detail(хронологія)**

Веб-браузер створює об’єкт Document і починає аналізувати веб-сторінку, додаючи об’єкти Element до документа під час аналізу елементів HTML та їх текстового вмісту. На цьому етапі властивість document.readyState має значення «loading».

Коли аналізатор зустрічає елементи <script> без атрибутів async і defer, він додає ці елементи до документа, а потім виконує вбудований або зовнішній сценарій. Ці синхронні сценарії можуть бачити власний елемент <script> і вміст документа, який знаходиться перед ним.

Коли аналізатор зустрічає елемент <script> із встановленим атрибутом async, він починає завантажувати текст сценарію та продовжує аналізувати документ. Асинхронні сценарії не повинні використовувати метод JavaScriptdocument.write(). Вони можуть бачити свій власний елемент <script> і всі елементи документа, які передують йому, і можуть мати або не мати доступу до додаткового вмісту документа.

Коли документ повністю проаналізовано, властивість document.readyState змінюється на «interactive».

Будь-які сценарії, для яких встановлено атрибут defer, виконуються в тому порядку, в якому вони з’явилися в документі. (У цей час також можуть виконуватися асинхронні сценарії). Відкладені сценарії мають доступ до повного дерева документів і не повинні використовувати метод document.write().

Браузер запускає подію DOMContentLoaded для об’єкта Document. Це означає перехід від фази синхронного виконання сценарію до асинхронної фази виконання програми, керованої подіями. (Однак зауважте, що ще можуть існувати асинхронні сценарії, які на даний момент ще не виконано).

Наразі документ повністю проаналізовано, але браузер може все ще чекати завантаження додаткового вмісту, наприклад зображень. Коли весь такий вміст завантажується та коли всі асинхронні сценарії завантажено та виконано, властивість document.readyState змінюється на «complete», а веб-браузер запускає подію завантаження об’єкта Window.

З цього моменту обробники подій викликаються асинхронно у відповідь на події, введені користувачем, мережеві події, закінчення таймера тощо.

**Для дуже довгих документів або дуже повільних мережевих з’єднань теоретично можливо, що веб-браузер відобразить частину документа та дозволить користувачеві почати взаємодію з ним до того, як усі сценарії будуть виконані.**