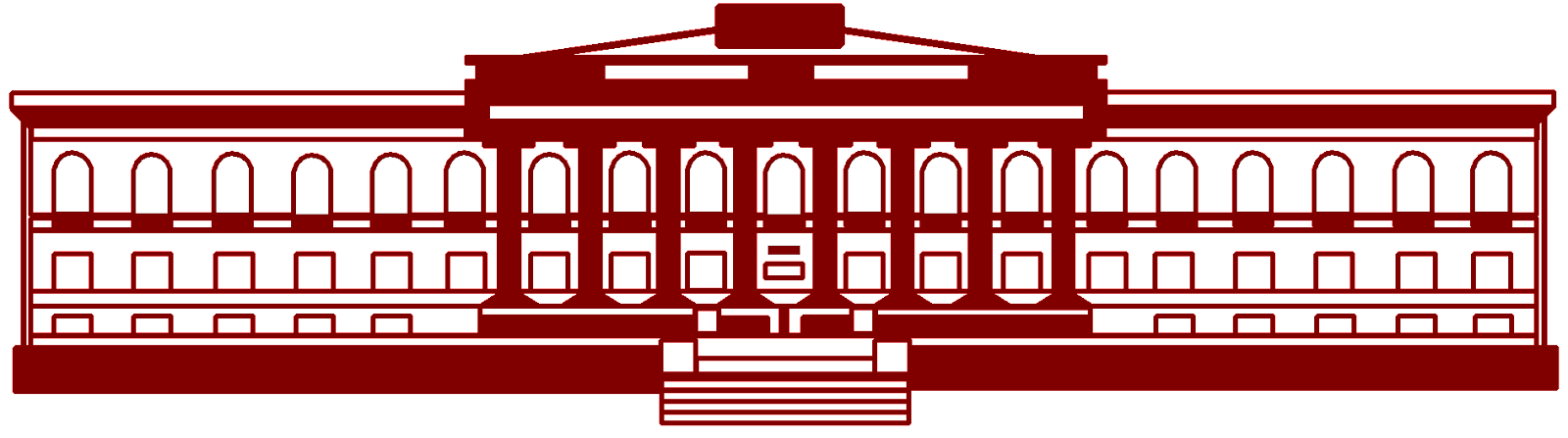
**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**імені ТАРАСА ШЕВЧЕНКА**

Факультет інформаційних технологій

**Кафедра прикладних інформаційних систем**



**Звіт**

до виконання лабораторної роботи №5

з дисципліни «**Веб-технології і програмування сайтів**»

на тему:

« Знайомство з роботою зі запитами в мові JavaScript: HTTP\s, AJAX, CORS, JSON »

**Виконано:**

студ. групи ПП- 31, підгрупа 2

Шкандюк Анною Леонідівною

**Перевірено:**

Пирог М.В.

**Київ – 2023**

1. **Мета роботи:**

Отримати навички роботи з отриманням данних.

1. **Завдання та їх виконання:**

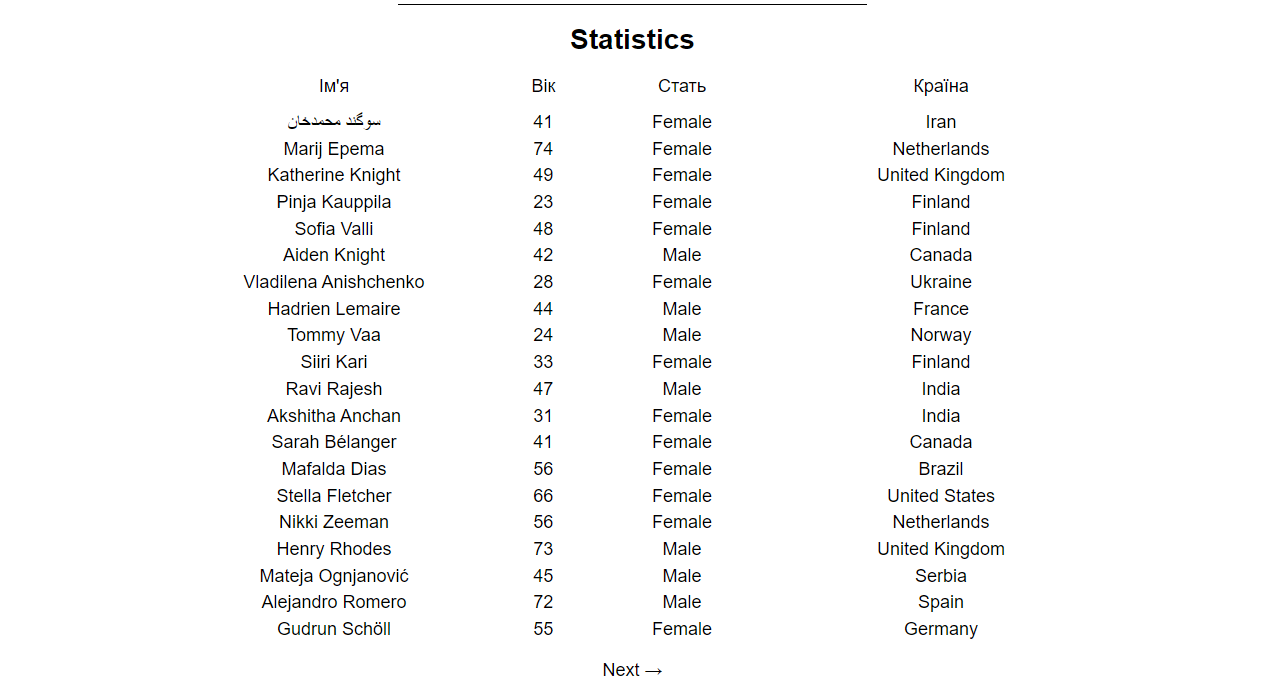
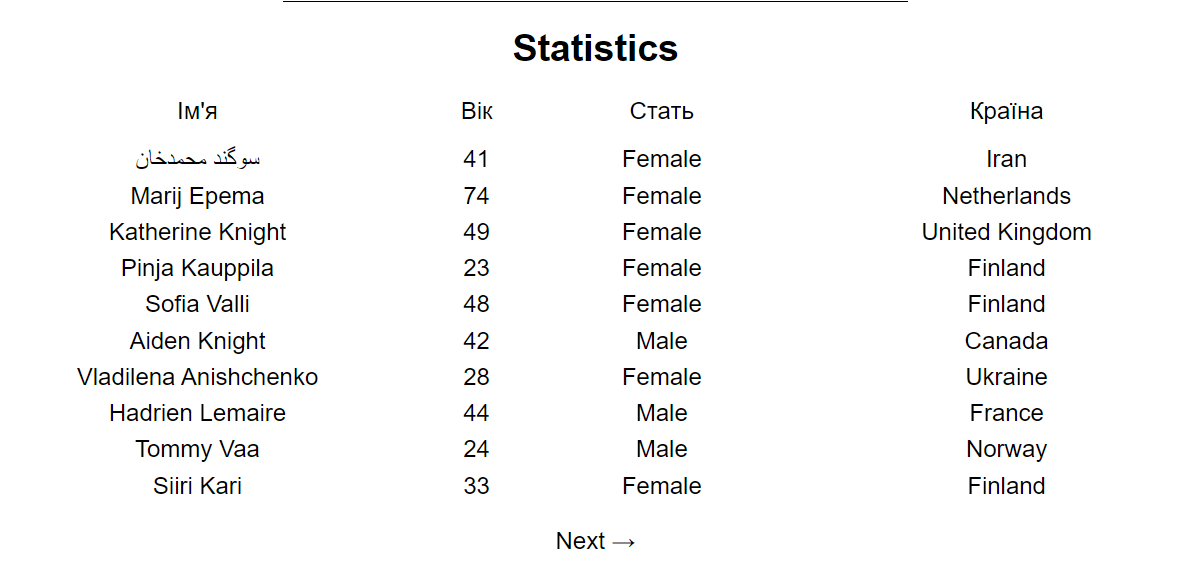
**Завдання 1**. Зробити запит за списком користувачів https://randomuser.me/api. Повертаючи 50 користувачів.



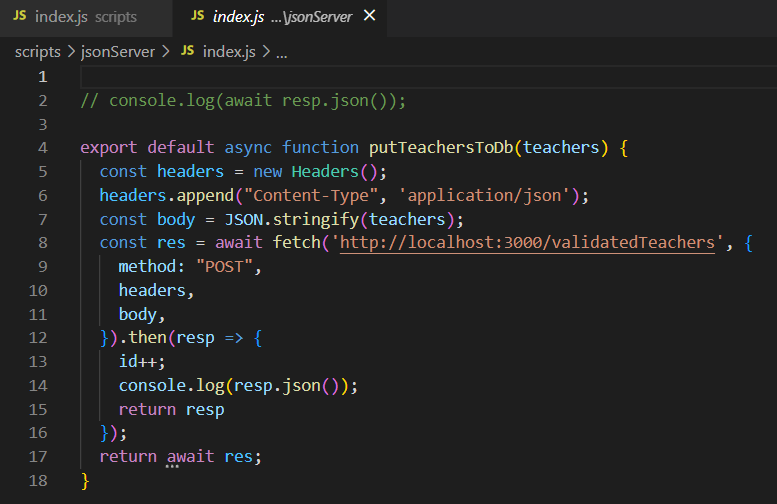
**Завдання 2.** Замінити сортування, фільтрацію, валідацію та пошук на роботу з данними із запиту. Оновити статистику на роботу з данними, що повертаються із запиту. Якщо кількість користувачів змінюється (фільтрація, пошук, додали нових), це відображається у статистиці.

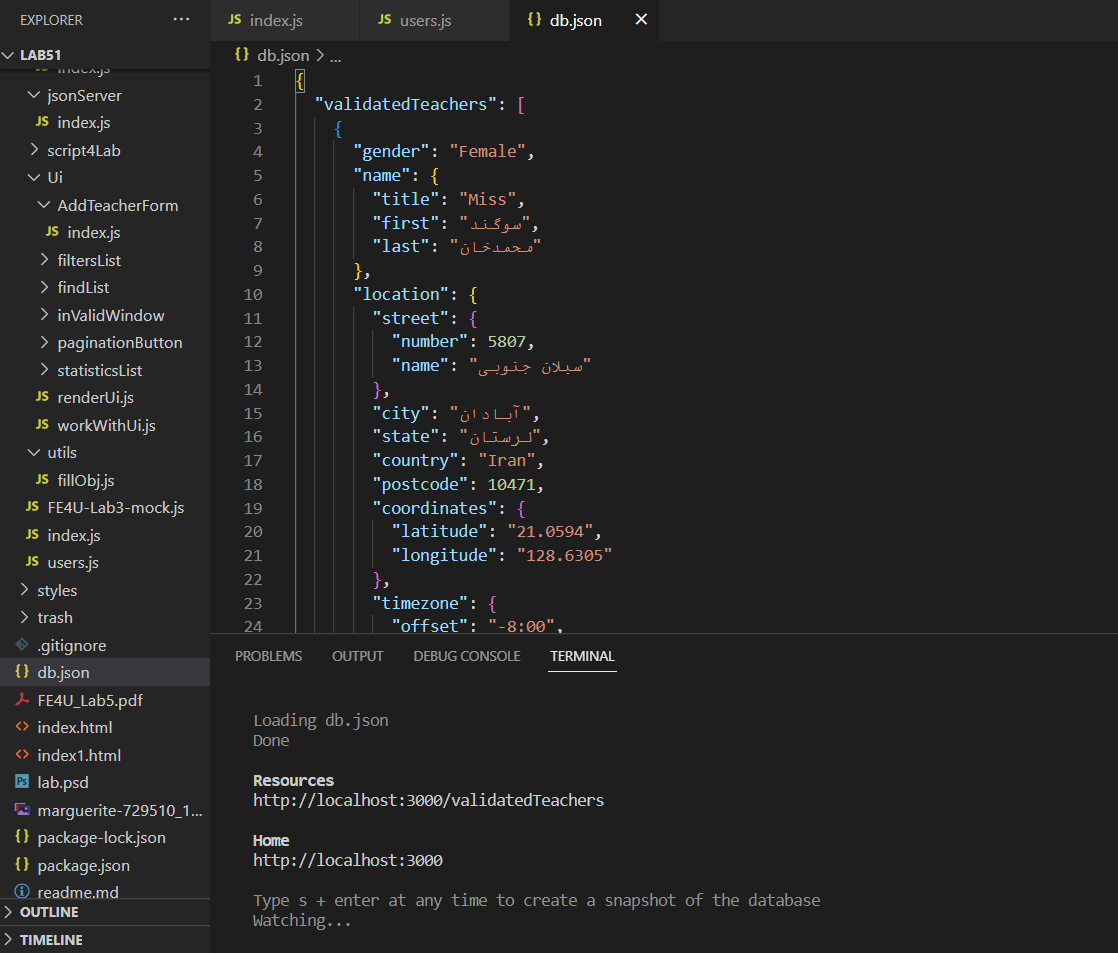
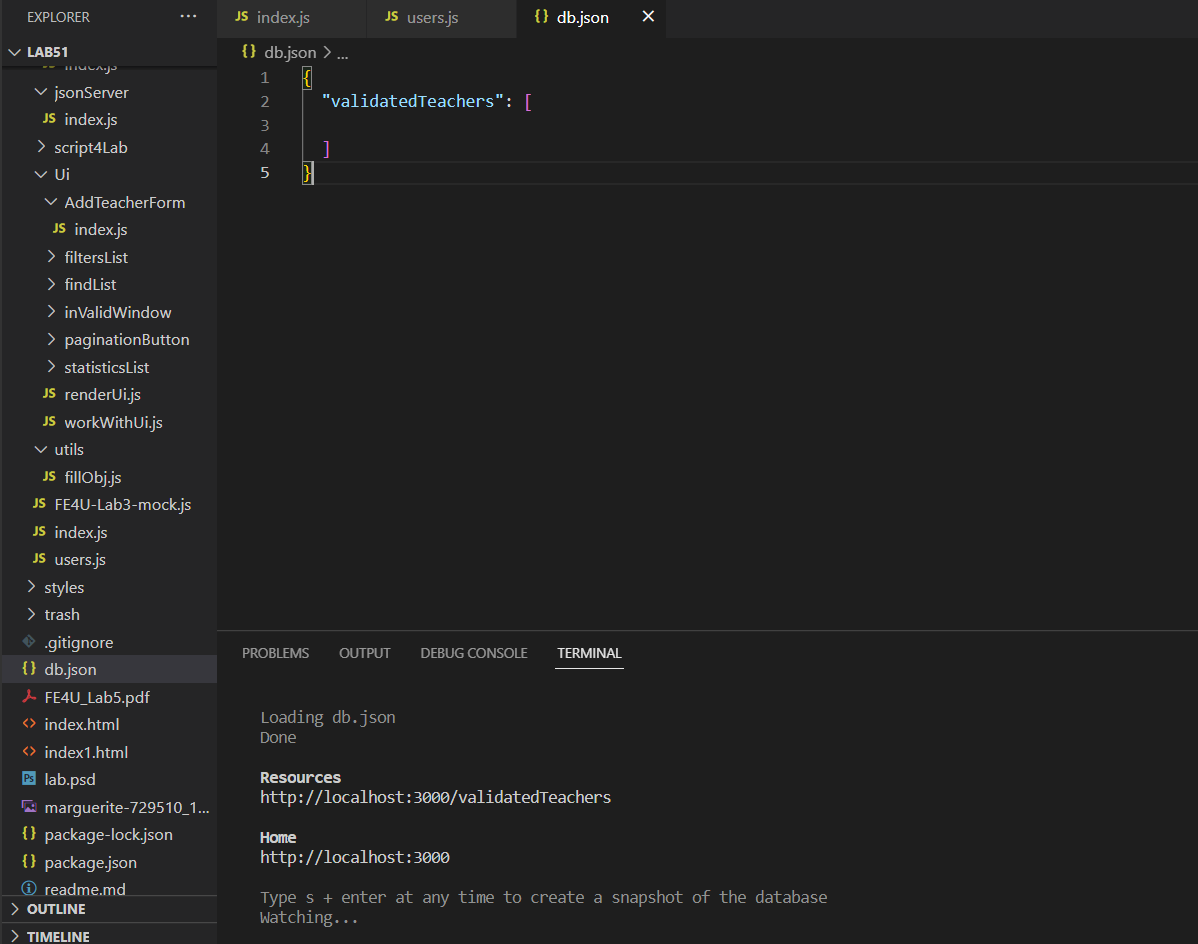
Це завдання виконалося автоматично внаслідок заміни початкових юзерів, тому при ініціалізації додатку, ми робимо запит, отримуючи новий список юзерів, які вже передаються в наступні етапи: валідацію, доповнення, виведення на сторінку...

**Завдання 4**. Додати пагінацію: робити запит на наступні 10 користувачів при натиску на кнопку «далі» (додані користувачівідображаються на сторінці).



**Завдання 5.** Додати до package.json як devDependency пакет json-server. Та налаштувати так, щоб при сабміті форми, на нього відправлявся POST запит з провалідованними данними з форми (опціонально).





1. **Висновки до роботи**

За результатами проробленої роботи можемо сказати, що отримали навички роботи з отриманням данних.

1. **Контрольні запитання**
2. Що таке *HTTP, HTTPS*?

HTTP і HTTPS - це протоколи передачі даних в мережі Інтернет, які використовуються для обміну інформацією між клієнтами та серверами. Ось їхні основні визначення:

1. HTTP (Hypertext Transfer Protocol):

HTTP - це протокол передачі гіпертекстової інформації, який використовується для здійснення комунікації між веб-браузерами (клієнтами) та веб-серверами. Використовуючи HTTP, браузери можуть отримувати веб-сторінки, зображення, відео, аудіо та інші дані з веб-серверів. Однак дані, передані через HTTP, не шифруються, що робить їх вразливими до перехоплення та зловживання.

2. HTTPS (Hypertext Transfer Protocol Secure):

HTTPS - це розширення протоколу HTTP з додатковим шаром безпеки. Він використовує шифрування SSL/TLS для захисту переданих даних між браузером і веб-сервером. Це означає, що дані, які передаються між клієнтом і сервером, шифруються, що ускладнює їх перехоплення та забезпечує більшу конфіденційність та безпеку.

1. Які порти за замовчуванням використовує протокол *HTTP*?

Протокол HTTP за замовчуванням використовує порт 80 для комунікації між клієнтами та веб-серверами. Коли ви вводите URL без вказання порта, браузер автоматично використовує порт 80 для HTTP-з'єднань. Наприклад:

http://example.com

Це означає, що браузер встановить з'єднання з веб-сервером за допомогою порту 80 для отримання вмісту.

Проте, якщо ви хочете вказати інший порт, наприклад, якщо ваш веб-сервер працює на іншому порті, ви можете вказати його у URL. Наприклад:

http://example.com:8080

У цьому випадку браузер встановить з'єднання з веб-сервером за допомогою порту 8080 замість стандартного порту 80.

Важливо пам'ятати, що при використанні HTTPS, який використовує шифрування SSL/TLS, стандартний порт змінюється на 443. Тобто, HTTPS використовує порт 443 за замовчуванням.

1. Типи *HTTP*-повідомлень

HTTP-протокол використовує різні типи повідомлень для обміну даними між клієнтами та серверами. Основні типи HTTP-повідомлень включають:

1. Запити (Request):

Запити відправляються від клієнта до сервера для отримання певних ресурсів або виконання дій на сервері. Запити мають наступну структуру:

METHOD URL HTTP/версія

Заголовки

Порожній рядок

Тіло запиту (не завжди присутнє)

Основні методи запитів включають:

- GET: Отримання ресурсу.

- POST: Відправка даних для обробки на сервер.

- PUT: Оновлення існуючого ресурсу або створення нового.

- DELETE: Видалення ресурсу.

2. Відповіді (Response):

Відповіді надсилаються від сервера до клієнта як реакція на запит. Вони містять статус запиту та можуть також містити дані або вміст запиту. Відповіді мають наступну структуру:

HTTP/версія Статус-код Пояснення

Заголовки

Порожній рядок

Тіло відповіді (не завжди присутнє)

Статус-код вказує на результат виконання запиту. Деякі загальні статус-коди:

- 200 OK: Успішний запит.

- 201 Created: Ресурс успішно створено.

- 400 Bad Request: Неправильний або нечинний запит.

- 404 Not Found: Ресурс не знайдено.

- 500 Internal Server Error: Помилка на сервері.

3. Перенаправлення (Redirection):

Вони вказують клієнтові перейти на іншу адресу для отримання ресурсу. Вони пов'язані зі статус-кодами 3xx, наприклад, 301 Moved Permanently або 302 Found.

4. Помилки (Error):

Ці повідомлення вказують на помилку або неможливість виконання запиту. Вони пов'язані зі статус-кодами 4xx (клієнтська помилка) та 5xx (помилка сервера).

HTTP-повідомлення грають важливу роль у взаємодії між веб-браузерами та веб-серверами, дозволяючи клієнтам взаємодіяти з веб-ресурсами та отримувати результати від серверів.

1. Загальна структура *HTTP-*повідомлення.

HTTP-повідомлення має загальну структуру, яка складається з рядків, заголовків та тіла. В залежності від типу повідомлення (запит або відповідь), структура може відрізнятися. Ось загальна структура для запиту і відповіді:

Структура HTTP-запиту:

Метод URL HTTP/версія

Заголовки

Порожній рядок

Тіло запиту (не завжди присутнє)

Структура HTTP-відповіді:

HTTP/версія Статус-код Пояснення

Заголовки

Порожній рядок

Тіло відповіді (не завжди присутнє)

Давайте подивимося на кожну частину більш детально:

1. Метод:

В HTTP-запиті, метод вказує, яку дію потрібно виконати над ресурсом на сервері. Популярні методи включають GET, POST, PUT, DELETE та інші.

2. URL:

URL (Uniform Resource Locator) вказує на шлях до ресурсу, з яким ви взаємодієте.

3. HTTP/версія:

Версія протоколу HTTP, яку використовує запит або відповідь. Наприклад, HTTP/1.1.

4. Статус-код:

В HTTP-відповіді, статус-код вказує на результат виконання запиту. Це числовий код, який вказує на успіх, помилку або інші стани.

5. Пояснення:

В HTTP-відповіді, пояснення деталізує статус-код, надаючи розуміння того, що трапилося під час виконання запиту.

6. Заголовки:

Заголовки містять метадані про запит або відповідь, такі як тип вмісту, розмір, дата, кешування тощо.

7. Тіло запиту або відповіді:

Тіло містить фактичні дані, що передаються у запиті або відповіді. Не завжди присутнє, наприклад, у запитах типу GET може бути відсутнім.

1. Перерахуйте методи *HTTP*, дайте визначення кожного з них.

Основні методи HTTP використовуються для взаємодії між клієнтами та серверами. Ось перелік методів разом з їх визначеннями:

1. GET:

Метод GET використовується для отримання ресурсу з сервера. Клієнт надсилає запит на сервер і отримує відповідь з даними ресурсу. Цей метод повинен бути безпечним та не залишати змін у сервері.

2. POST:

Метод POST використовується для відправки даних на сервер з метою обробки або збереження. Це може бути створення нового ресурсу або виконання дії, що змінює стан на сервері.

3. PUT:

Метод PUT використовується для оновлення існуючого ресурсу або створення нового, якщо ресурс відсутній. Клієнт надсилає дані для оновлення вказаного ресурсу на сервері.

4. DELETE:

Метод DELETE використовується для видалення ресурсу на сервері. Клієнт надсилає запит на видалення вказаного ресурсу.

5. PATCH:

Метод PATCH використовується для часткового оновлення ресурсу. Він дозволяє клієнту вказати конкретні зміни, які слід внести до ресурсу на сервері.

6. HEAD:

Метод HEAD подібний до GET, але сервер відправляє лише заголовки відповіді, без фактичного вмісту ресурсу. Використовується для отримання метаданих без фактичних даних.

7. OPTIONS:

Метод OPTIONS використовується для отримання списку підтримуваних методів, заголовків та інших параметрів сервера для конкретного ресурсу.

8. CONNECT:

Метод CONNECT використовується для встановлення мережевого з'єднання до вказаного ресурсу, який може бути оброблений як тунель.

9. TRACE:

Метод TRACE використовується для отримання діагностичної інформації від сервера. Це дозволяє клієнту перевірити, як проксі або сервер обробляє запит.

Ці методи визначають, яку дію слід виконати з ресурсом на сервері під час взаємодії між клієнтом та сервером.

1. Коди стану *HTTP*-відповідей

HTTP-відповіді містять коди стану, які вказують на результат виконання запиту. Кожен код стану відповіді має числове значення та асоційований з ним текстовий пояснюючий опис. Ось деякі з найпоширеніших кодів стану HTTP-відповідей:

1xx - Інформаційні коди:

- 100 Continue: Сервер готовий продовжити обробку запиту після отримання заголовків клієнта.

2xx - Успішні коди:

- 200 OK: Успішний запит, відповідь містить потрібні дані.

- 201 Created: Ресурс успішно створено.

- 204 No Content: Успішний запит, але відповідь не містить вмісту.

3xx - Коди перенаправлення:

- 301 Moved Permanently: Ресурс більше не доступний, перенаправлення на нову адресу.

- 302 Found: Тимчасове перенаправлення на іншу адресу.

- 304 Not Modified: Ресурс не змінився, можна використовувати кешовану версію.

4xx - Клієнтські помилки:

- 400 Bad Request: Неправильний або нечинний запит.

- 401 Unauthorized: Авторизація потрібна або неуспішна.

- 403 Forbidden: Доступ заборонено.

- 404 Not Found: Ресурс не знайдено.

5xx - Помилки сервера:

- 500 Internal Server Error: Внутрішня помилка сервера.

- 501 Not Implemented: Запит не підтримується сервером.

- 503 Service Unavailable: Сервіс тимчасово недоступний.

Ці коди стану допомагають клієнтам та розробникам розуміти результати запитів та дізнатися про стан ресурсів на сервері.

1. Що таке *HTTP cookie*?

HTTP-кукі (англ. HTTP cookie) - це невеликий фрагмент інформації, який веб-сервер надсилає і зберігає на браузері користувача. Куки використовуються для збереження певних даних на стороні клієнта та обміну цими даними між сервером та браузером під час HTTP-запитів та відповідей. Куки дозволяють веб-сайтам "запам'ятовувати" користувачів та зберігати інформацію про їхні дії на сайті.

Кожне куки має наступні атрибути:

1. Ім'я: Унікальне ім'я, яке ідентифікує куки на браузері.

2. Значення: Дані, які зберігаються в куки.

3. Домен: Домен веб-сайту, якому належить куки.

4. Шлях: Шлях на веб-сайті, для якого куки є доступним.

5. Термін дії: Термін, протягом якого куки є активним.

6. Захищено (Secure): Куки буде передано лише через захищене з'єднання HTTPS.

7. HTTP Only: Куки доступне лише через HTTP-запити, а не JavaScript.

8. Схвалення (SameSite): Вказує, як куки повинно використовуватись при кросс-сайтових запитах.

Куки можуть бути використані для різних цілей:

- Зберігання ідентифікаторів сесій та авторизації.

- Запам'ятовування налаштувань користувача на веб-сайті.

- Відстеження активності користувача на сайті для аналітики.

- Зберігання товарів у кошику покупок на інтернет-магазинах.

- Персоналізація контенту для користувачів.

Проте слід пам'ятати, що використання куки може підняти питання щодо конфіденційності та приватності даних користувачів. Тому важливо використовувати куки обережно та дотримуватися вимог законодавства щодо захисту персональних даних.

1. Що таке *AJAX*?

AJAX (Asynchronous JavaScript and XML) - це технологія, яка дозволяє взаємодіяти з сервером та оновлювати вміст веб-сторінки без необхідності повного перезавантаження сторінки. Використовуючи AJAX, можна відправляти та отримувати дані асинхронно, тобто без зупинки виконання інших функцій на сторінці.

Основні складові AJAX:

1. Асинхронність: Запити до сервера та отримання відповідей відбуваються асинхронно, що дозволяє продовжувати взаємодію зі сторінкою без блокування.

2. JavaScript: Використовується JavaScript для ініціювання запитів до сервера та обробки отриманих даних.

3. XMLHttpRequest (XHR): Це об'єкт у браузері, який дозволяє виконувати асинхронні запити до сервера і отримувати дані з сервера.

4. Обмін даними: Зазвичай використовується формат JSON (не обов'язково XML) для передачі даних між клієнтом та сервером.

Використання AJAX дозволяє створювати більш динамічні та швидкодіючі веб-додатки, оскільки можна оновлювати лише ту частину сторінки, яка потребує змін. Це робить користувачів взаємодію з веб-сайтом більш зручною та ефективною.

1. З якою метою використовується технологія *AJAX* при побудові сучасних *Web* додатків?

Технологія AJAX (Asynchronous JavaScript and XML) використовується при побудові сучасних веб-додатків з метою поліпшення користувацького досвіду, зменшення затримок та підвищення ефективності взаємодії між користувачем та сервером. Основні цілі використання AJAX включають:

1. Динамічність: AJAX дозволяє оновлювати частини веб-сторінки асинхронно без потреби повного перезавантаження сторінки. Це створює враження відзначної динаміки та швидкодії.

2. Зменшення затримок: Завдяки асинхронності, користувачі можуть отримувати дані з сервера без очікування повного завантаження сторінки. Це допомагає знизити час очікування та поліпшити загальний час реакції додатка.

3. Споживання ресурсів: Використання AJAX дозволяє передавати лише ті дані, які необхідні для оновлення вмісту, зменшуючи обсяг передачі даних і використання мережевих ресурсів.

4. Персоналізація: AJAX дозволяє створювати персоналізовані веб-додатки, які адаптуються до дій та вибору користувача без потреби повного оновлення сторінки.

5. Динамічна завантаження контенту: Вміст, такий як зображення, відео або дані, може завантажуватися лише при необхідності, що покращує швидкість завантаження сторінок.

6. Онлайн-комунікація: AJAX дозволяє реалізувати функціональність онлайн-чату, коментарів, спільної роботи над документами та інші форми інтерактивної комунікації між користувачами.

Загалом, використання AJAX допомагає створювати більш ефективні, зручні та корисні веб-додатки, які відповідають сучасним вимогам користувачів до продуктивності та досвіду взаємодії.

1. Що таке *JSON*?

JSON (JavaScript Object Notation) - це легкий формат обміну даними, який використовується для передачі та збереження структурованих інформаційних об'єктів між різними додатками. JSON є текстовим форматом, що спрощує зчитування та запис даних, і використовується часто в веб-додатках для передачі даних між клієнтом та сервером.

Основні характеристики JSON:

1. Синтаксис: JSON представляє дані у вигляді пар "ключ": "значення", розділених комами. Об'єкти у форматі JSON включаються у фігурні дужки `{}`, а масиви - у квадратні дужки `[]`.

2. Типи даних: JSON підтримує ряд стандартних типів даних, такі як рядки, числа, булеві значення, масиви, об'єкти та значення `null`.

3. Незалежність від мови: JSON може використовуватися в багатьох мовах програмування, і дані, представлені у форматі JSON, можна легко конвертувати у внутрішні об'єкти або структури даних.

4. Читабельність: Формат JSON є зручним для людського читання та редагування, що полегшує роботу з ним.

5. Ієрархія та вкладеність: В JSON можна створювати складні структури даних, вкладаючи об'єкти та масиви один в одного.

Приклад JSON-об'єкту:

{

"name": "John",

"age": 30,

"isStudent": false,

"hobbies": ["reading", "coding", "hiking"],

"address": {

"street": "123 Main St",

"city": "New York"

}

}

JSON використовується в широкому спектрі додатків, зокрема в веб-додатках для обміну даними між клієнтом та сервером, в API для передачі даних між різними службами, в налаштуваннях програм та багатьох інших випадках, де потрібно передавати структуровані дані.

1. Що таке *CORS*? В яких випадках використовується ця технологія?

CORS (Cross-Origin Resource Sharing) - це механізм безпеки веб-браузерів, який дозволяє веб-сторінкам запитувати ресурси з інших доменів (origin) та обмінюватися даними між різними доменами на стороні клієнта. Основна мета CORS полягає в тому, щоб запобігти потенційно небезпечним сценаріям атак, таким як перехоплення даних або виконання зловмисного коду з іншого домену.

У випадках, коли сторінка або додаток виконує запити до іншого домену, браузер може використовувати CORS, щоб визначити, чи дозволено виконувати такі запити. Контроль доступу до ресурсів здійснюється за допомогою HTTP-заголовків, які додаються до відповіді сервера.

CORS використовується у таких випадках:

1. AJAX-запити на інші домени: Коли ви виконуєте AJAX-запити зі сторінки на сервер іншого домену, браузер використовує CORS для перевірки, чи дозволено отримувати дані з цього домену.

2. Вбудовані фрейми і скрипти: Якщо ви вбудовуєте фрейми або скрипти з іншого домену, браузер також перевіряє політику CORS для визначення можливості доступу.

3. API запити: Якщо ви створюєте власне API та дозволяєте іншим додаткам звертатися до нього, ви можете налаштувати CORS, щоб забезпечити безпеку.