Масштабована Векторна Графіка (SVG, Scalable Vector Graphics) — це формат зображень, який ми часто використовуємо для дизайну веб-сайтів та інтерфейсів користувачів. Він базується на XML, що дозволяє нам створювати зображення з векторної графіки, які можуть бути масштабовані без втрати якості. Це означає, що ми можемо збільшувати чи зменшувати розмір зображення, і воно залишатиметься різким і чітким на будь-якому дисплеї.

Ми використовуємо SVG з кількох причин. По-перше, він дозволяє нашим веб-сторінкам завантажуватися швидше, оскільки векторні зображення часто мають менший розмір файлу порівняно з растровими зображеннями. По-друге, SVG підтримує прозорість і може бути легко анімованим або стилізованим за допомогою CSS. Це робить його ідеальним для логотипів, іконок, та інших дизайнерських елементів, які потребують гнучкості в дизайні та високої якості при будь-якому розширенні.

Крім того, SVG дозволяє нам додавати інтерактивні елементи та анімації без використання додаткових плагінів або бібліотек, забезпечуючи більш багаті та динамічні користувацькі інтерфейси. Завдяки своїй сумісності та гнучкості, SVG став невід'ємною частиною нашого інструментарію при розробці веб-сайтів та додатків.

**Елементи для створення фігур**

У нашій роботі з Масштабованою Векторною Графікою (SVG), ми часто використовуємо різні елементи для створення графічних об'єктів. Ось основні з них та їх призначення:

* <rect>: цей елемент дозволяє нам створювати прямокутники, включаючи квадрати. Ми можемо керувати його розмірами, кольором заливки та обводки.
* <circle>: за допомогою цього елемента ми формуємо кола, вказуючи їх радіус та центр.
* <ellipse>: цей елемент використовується нами для створення еліпсів, для яких ми можемо задати радіуси по горизонталі та вертикалі.
* <line>: за його допомогою ми малюємо прямі лінії, вказуючи їх початкову та кінцеву точки.
* <polyline>: цей елемент дозволяє нам створювати ламані лінії, визначаючи серію точок.
* <polygon>: за допомогою цього елемента ми формуємо багатокутники, вказуючи вершини. Він схожий на <polyline>, але автоматично з'єднує останню точку з першою.
* <path>: цей елемент найбільш універсальний, оскільки він дозволяє нам створювати довільні форми, використовуючи набір команд та параметрів.
* <text>: ми використовуємо цей елемент для додавання тексту до наших SVG зображень, де можемо керувати його стилем, розміром та положенням.

Кожен з цих елементів має своє специфічне призначення та набір атрибутів, які дозволяють нам створювати складні та динамічні векторні графіки. Це надає нам великі можливості для дизайну веб-сайтів, рекламних матеріалів, інтерфейсів користувачів, та інших цифрових активів, де потрібна гнучкість, масштабованість та висока якість зображень.

Error

**Декілька простих прикладів SVG**

1. **Прямокутник**:

<svg width="100" height="100"> <!-- Визначаємо розмір SVG області -->

<rect width="50" height="50" style="fill:rgb(0,0,255);stroke-width:3;stroke:rgb(0,0,0)" />

<!-- Створюємо прямокутник синього кольору розміром 50x50 пікселів з чорним контуром товщиною 3 пікселі -->

</svg>

1. Цей код створює прямокутник синього кольору розміром 50x50 пікселів.
2. **Коло**:

<svg width="100" height="100"> <!-- Визначаємо розмір SVG області -->

<circle cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="red" />

<!-- Створюємо червоне коло з радіусом 40 пікселів і чорним контуром товщиною 3 пікселі -->

</svg>

1. Цей код створює червоне коло з радіусом 40 пікселів і чорним контуром.
2. **Лінія**:

<svg height="210" width="500"> <!-- Визначаємо розмір SVG області -->

<line x1="0" y1="0" x2="200" y2="200" style="stroke:rgb(255,0,0);stroke-width:2" />

<!-- Створюємо червону лінію, що проходить від точки (0,0) до точки (200,200) товщиною 2 пікселі -->

</svg>

1. Цей код створює червону лінію, що проходить від точки (0,0) до точки (200,200).
2. **Текст**:

<svg height="210" width="500"> <!-- Визначаємо розмір SVG області -->

<text x="10" y="20" style="font-family:Verdana;font-size:35;fill:blue">Hello SVG</text>

<!-- Створюємо текст "Hello SVG", написаний синім кольором шрифтом Verdana розміром 35 -->

</svg>

1. Цей код створює текст “Hello SVG”, написаний синім кольором шрифтом Verdana розміром 35.

SVG дозволяє створювати багато інших форм та об’єктів, включаючи складні форми та анімації. Це дуже потужний інструмент для створення графіки в веб-дизайні.

**Методи стилізації SVG**

Методи стилізації SVG дозволяють нам контролювати вигляд векторних зображень за допомогою CSS і вбудованих атрибутів стилю. Це робить SVG ідеальним для дизайну веб-сайтів, де потрібно легко змінювати вигляд графіки без зміни самого зображення. Використання стилізації дозволяє нам забезпечити консистентність дизайну, оптимізувати час завантаження сторінок та поліпшити управління графічними ресурсами.

Розглянемо, як можна стилізувати SVG за допомогою CSS. У цьому прикладі ми створимо прямокутник і застосуємо до нього стилі за допомогою атрибуту style.

<svg width="100" height="100">

<rect width="90" height="90" x="5" y="5" style="fill:blue; stroke:black; stroke-width:5; opacity:0.9" />

</svg>

У цьому коді ми визначаємо SVG область розміром 100x100 пікселів і створюємо всередині неї прямокутник з наступними стилями:

* fill:blue - заливаємо прямокутник синім кольором.
* stroke:black - задаємо колір обводки чорним.
* stroke-width:5 - встановлюємо товщину обводки в 5 пікселів.
* opacity:0.9 - задаємо прозорість зображення на рівні 90%.

Цей приклад ілюструє, як ми можемо легко застосовувати різні стилі до SVG елементів, використовуючи вбудовані атрибути стилю. Це дає нам гнучкість у визначенні вигляду векторної графіки прямо в HTML, забезпечуючи швидку та ефективну розробку дизайну.

## Частина 2.

### Способи вставки SVG в HTML та CSS

Для вставки SVG в веб-сторінки ми маємо кілька різних методів, які дозволяють нам вибирати найкращий спосіб в залежності від наших потреб. Кожен з них має свої переваги та використовується для різних сценаріїв:

1. **Використання тега <img>**: Цей метод дозволяє нам вставляти SVG як зображення на сторінку. Це простий і швидкий спосіб додати векторну графіку без можливості маніпулювати її частинами через CSS або JavaScript.
2. **Приклад коду**:

<img src="path/to/your-image.svg" alt="Описання зображення">

1. Цей метод підходить для статичних зображень, де не потрібна взаємодія чи анімація.
2. **Використання тега <object>**: SVG можна вставити через <object>, що дозволяє зберегти векторність і, в деяких випадках, взаємодіяти зі SVG через DOM.
3. **Приклад коду**:

<object type="image/svg+xml" data="path/to/your-image.svg">Ваш браузер не підтримує SVG</object>

1. Цей метод може бути корисним, коли потрібно зберегти векторні властивості зображення і мати можливість взаємодії з ним.
2. **Використання тега <embed>**: Подібно до <object>, тег <embed> може бути використаний для вставки SVG, забезпечуючи взаємодію через DOM.
3. **Приклад коду**:

<embed type="image/svg+xml" src="path/to/your-image.svg" />

1. Цей спосіб дозволяє вбудовувати SVG з можливістю доступу та маніпуляції через JavaScript.
2. **Інлайново через <svg>**: Вставка SVG коду безпосередньо в HTML дає найбільшу гнучкість, дозволяючи стилізувати та маніпулювати кожною частиною SVG за допомогою CSS і JavaScript.
3. **Приклад коду**:

<svg width="100" height="100">

<circle cx="50" cy="50" r="40" stroke="green" stroke-width="4" fill="yellow" />

</svg>

1. Цей метод ідеально підходить для інтерактивних або анімованих векторних графік.
2. **Як фон через CSS**: SVG можна використовувати як фонове зображення елементів веб-сторінки через CSS.
3. **Приклад коду**:

.element {

background-image: url('path/to/your-image.svg');

}

1. Це дозволяє легко змінювати векторні фони без необхідності зміни HTML коду, ідеально підходить для декоративних елементів.

Кожен із цих методів має свої особливості та застосування, вибір залежить від конкретних потреб проєкту та вимог до взаємодії з SVG.

### Повторне використання зображень

Для повторного використання SVG зображень у наших проєктах, ми часто застосовуємо елементи <symbol> та <use>. Це дозволяє нам ефективно організувати наш код і зменшити його дублювання, спрощуючи одночасне управління та оновлення множинних інстанцій SVG елементів.

**Елемент <symbol>**

Елемент <symbol> використовується для оголошення графічних об'єктів, які можна повторно використовувати в SVG документі за допомогою елемента <use>. Елементи <symbol> не відображаються в документі до тих пір, поки не будуть викликані через <use>, що робить їх ідеальними для визначення "шаблонів" графіки, які потрібно використовувати кілька разів.

**Елемент <use>**

Елемент <use> використовується для створення копій SVG елементів, які були визначені раніше за допомогою <symbol> або безпосередньо в <svg>. Це дозволяє нам вставляти копії оригінальних елементів без необхідності копіювати їх код, спрощуючи управління та оновлення зображень.

Приклад коду:

<svg style="height: 0">

<symbol id="icon-heart" viewBox="0 0 100 100">

<!-- SVG контент для іконки "серце" -->

<path d="M10,30 ... Z" fill="red"/>

</symbol>

</svg>

<!-- Використання іконки "серце" у документі -->

<svg width="100" height="100">

<use xlink:href="#icon-heart"></use>

</svg>

<svg width="50" height="50">

<use xlink:href="#icon-heart"></use>

</svg>

У цьому прикладі ми визначаємо іконку "серце" як <symbol> з ідентифікатором icon-heart і потім використовуємо її двічі в нашому документі з різними розмірами за допомогою елемента <use>. Це демонструє, як ми можемо легко повторно використовувати SVG зображення, зберігаючи при цьому чистоту та організованість нашого коду.

### SVG спрайти

SVG спрайт - це збірка SVG-іконок у одному файлі. Ми використовуємо SVG спрайти для оптимізації завантаження веб-сторінок, адже завантаження одного файлу ефективніше, ніж багатьох маленьких зображень. Вони дозволяють нам використовувати багато іконок без необхідності збільшення кількості HTTP-запитів. Також SVG спрайти підтримують масштабування без втрати якості та можуть бути легко стилізовані за допомогою CSS.

Приклад коду SVG спрайта:

<svg style="display: none;">

<symbol id="icon-phone" viewBox="0 0 24 24">

<path d="M6.62 10.79a15.053 15.053 0 006.59 6.59l2.2-2.2a1.5 1.5 0 011.68-.27c.75.3 1.56.46 2.4.46.83 0 1.5.67 1.5 1.5v3.5c0 .83-.67 1.5-1.5 1.5A18 18 0 013 3C3 2.17 3.67 1.5 4.5 1.5h3.5c.83 0 1.5.67 1.5 1.5 0 .84-.16 1.65-.46 2.4a1.5 1.5 0 01-.27 1.68l-2.2 2.21z"></path>

</symbol>

<symbol id="icon-email" viewBox="0 0 24 24">

<path d="M20 4H4c-1.1 0-2 .9-2 2v12c0 1.1.9 2 2 2h16c1.1 0 2-.9 2-2V6c0-1.1-.9-2-2-2zm0 4l-8 5-8-5V6l8 5 8-5v2z"></path>

</symbol>

<!-- Більше SVG іконок -->

</svg>

У цьому коді ми створили SVG, що містить два символи: icon-phone і icon-email. Кожен символ має свій id, який ми можемо використовувати для посилання на іконку в HTML. Відображення іконок в HTML може бути здійснено за допомогою:

<svg><use xlink:href="#icon-phone"></use></svg>

<svg><use xlink:href="#icon-email"></use></svg>

Тут <use> вказує на id символу в SVG спрайті, дозволяючи нам повторно використовувати SVG-графіку без необхідності дублювати її код.

## Частина 1.

**Accessibility (Доступність** або **A11Y**) визначає підходи та практики, спрямовані на створення вебсайтів та веб-додатків, які є легкодоступними та зручними у використанні для всіх користувачів, включаючи осіб з різноманітними формами обмежень.

Ми прагнемо до розробки таких рішень, щоб забезпечити рівний доступ до інформації та сервісів, використовуючи стандарти та технології, які дозволяють людям з обмеженими можливостями користуватися веб-ресурсами без бар'єрів. Це включає застосування альтернативних текстів для зображень, підтримку екранних читачів, забезпечення навігації за допомогою клавіатури та створення адаптивних макетів.

Важливість такого підходу полягає не тільки в етичному аспекті забезпечення рівних прав та можливостей для всіх користувачів, але й у відповідності до законодавчих вимог та покращенні загальної якості веб-дизайну, що сприяє більшій залученості користувачів та покращенню пошукової оптимізації.

Error

### Види обмежень

Обмеження можуть бути ситуативними, тимчасовими та постійними, і вони впливають на спосіб взаємодії людей з нашими продуктами та середовищем.

* **Ситуативні обмеження** виникають у певних обставинах або унікальних ситуаціях. Наприклад, яскраве сонячне світло, що ускладнює бачення екрану смартфона, або гучне оточення, що перешкоджає чутному сприйняттю. Ми розглядаємо ситуативні обмеження, щоб наші продукти були зручними у різноманітних умовах використання.
* **Тимчасові обмеження** з'являються на обмежений часовий період. Це може бути зламана рука, що тимчасово обмежує мобільність, або втрата голосу, що ускладнює спілкування. Ми враховуємо тимчасові обмеження, щоб забезпечити, що наші продукти залишаються доступними навіть під час тимчасових змін у здатностях користувачів.
* **Постійні обмеження** є тривалими і включають в себе стани, як-от втрата слуху або обмежена мобільність. Ми інтегруємо рішення для постійних обмежень, щоб наші продукти були доступними та використовувані всіма людьми, незалежно від їхніх фізичних можливостей.

Врахування всіх трьох видів обмежень є критично важливим для створення інклюзивних продуктів та сервісів, які можуть використовувати всі люди, незалежно від їхнього стану або обставин. Це дозволяє нам забезпечити рівний доступ та можливості для всіх, сприяючи розширенню нашої аудиторії та покращенню загального досвіду користувачів.

Error

### Принципи доступності

Принципи доступності, визначені у Web Content Accessibility Guidelines (WCAG, Керівні вказівки щодо доступності веб-контенту), включають сприйнятливість, керованість, зрозумілість та надійність.

Ці принципи служать основою для створення веб-ресурсів, які можуть бути легко використані всіма людьми, включаючи осіб з обмеженими можливостями.

* **Сприйнятливість** означає, що користувачі повинні мати можливість сприймати інформацію та користувацький інтерфейс через свої засоби взаємодії, незалежно від того, чи використовують вони технології адаптації, як-от екранні читачі.
* **Керованість** гарантує, що користувачі можуть керувати елементами інтерфейсу та навігувати через контент. Це включає забезпечення доступності всіх функцій з клавіатури та розумне використання навігаційних допоміжних засобів.
* **Зрозумілість** стосується того, наскільки легко користувачі можуть розуміти інформацію та як керувати інтерфейсом. Це вимагає чіткого та простого мовлення, передбачуваності інтерфейсу та помилок, що мінімізують дизайн.
* **Надійність** забезпечує стабільність і сумісність контенту з різними браузерами, технологіями адаптації та іншими користувацькими агентами, зокрема з мобільними пристроями.

Ці принципи використовуються для створення інклюзивного цифрового середовища, яке враховує потреби всіх користувачів, знижуючи бар'єри доступу та сприяючи рівним можливостям у цифровому просторі. Вони допомагають нам забезпечити, що наші веб-ресурси є доступними, зручними та корисними для якнайширшого кола користувачів.

Error

### Як писати frontend з accessibility

При розробці фронтенду з урахуванням доступності, ми дотримуємося наступних рекомендацій:

1. **Використовуємо лендмарк-елементи** для організації структури сторінки, що дозволяє користувачам з екранними читачами легко навігувати по сторінці.
2. **Застосовуємо атрибути ARIA (Accessible Rich Internet Applications)** для поліпшення доступності компонентів веб-інтерфейсу, особливо коли стандартні HTML-елементи не можуть повністю задовольнити потреби в доступності.
3. **Забезпечуємо навігацію з клавіатури** для всіх інтерактивних елементів, дозволяючи користувачам, які не можуть або не бажають використовувати мишу, взаємодіяти з нашим сайтом.
4. **Додаємо підписи до зображень** за допомогою атрибуту alt, що дозволяє користувачам з порушеннями зору зрозуміти, що зображено на них.
5. **Підписуємо всі поля форм** використовуючи тег label, забезпечуючи зрозумілість і доступність форм для всіх користувачів.
6. **Уникаємо використання капчі**, оскільки вона може створювати бар'єри для людей з різними обмеженнями. Якщо потрібно захистити форми, шукаємо альтернативні методи верифікації.
7. **Коли використовуємо сторонні бібліотеки**, переконуємося в їхній сумісності з принципами доступності або шукаємо альтернативи, які відповідають цим вимогам.

Додатково, важливо **перевіряти доступність сторінок за допомогою спеціалізованих інструментів** та випробувань реальними користувачами, щоб забезпечити високий рівень доступності нашого веб-контенту. Це допомагає нам ідентифікувати та усунути потенційні проблеми, перш ніж вони вплинуть на кінцевого користувача.

## Частина 2.

### Best Practices

Для підвищення доступності веб-сайтів ми дотримуємося низки кращих практик у наступних аспектах:

**1.Зображення**

Для зображень завжди використовуємо атрибут **alt** для детального опису, що допомагає користувачам з візуальними обмеженнями зрозуміти контент.

<img src="logo.png" alt="Логотип компанії">

**2.Форми**

Маркуємо кожне поле форми за допомогою елемента **label**, щоб полегшити навігацію і зрозумілість.

<label for="email">Електронна адреса:</label>

<input type="email" id="email" name="email">

**3.Сторінка і верстка**

Застосовуємо семантичні HTML-елементи, такі як <header>, <nav>, <main>, <footer>, для структурування контенту, що сприяє кращій навігації за допомогою екранних читачів. Задаємо мову сторінки за допомогою атрибута lang.

<html lang="uk">

<main>

<header>...</header>

<nav>...</nav>

...

</main>

</html>

**4.Заголовки**

Використовуємо ієрархію заголовків (<h1> до <h6>) для організації контенту, що допомагає користувачам зрозуміти структуру сторінки.

<h1>Головний заголовок сторінки</h1>

<h2>Підзаголовок розділу</h2>

**5.Меню**

Для навігації використовуємо **<nav>** і списки для структурування посилань, забезпечуючи логічну структуру та доступність.

<nav>

<ul>

<li><a href="#home">Головна</a></li>

<li><a href="#about">Про нас</a></li>

</ul>

</nav>

**6.HTML5**

Застосовуємо семантичні елементи HTML5 для кращої структуризації та семантики веб-сторінок.

<article>

<section>

<header>...</header>

<p>...</p>

</section>

</article>

**7.Кнопки**

Для інтерактивних елементів ми використовуємо елемент <button> замість <a> або <div>, щоб забезпечити правильне сприйняття і взаємодію з елементом.

<button type="submit">Надіслати</button>

Використання цих практик дозволяє нам створювати веб-сайти, які є доступними для широкого кола користувачів, включаючи осіб з обмеженими можливостями, та сприяти інклюзивності в цифровому просторі.

### WAI-ARIA

WAI-ARIA, що розшифровується як Web Accessibility Initiative - Accessible Rich Internet Applications, є набором специфікацій, розроблених для покращення доступності веб-контенту та веб-додатків для людей з обмеженими можливостями.

Ми використовуємо WAI-ARIA, щоб зробити веб-елементи, які зазвичай не доступні (наприклад, динамічний контент і складні віджети інтерфейсу), більш зрозумілими та керованими для асистивних технологій, таких як екранні читачі. Це дозволяє нам створювати веб-сайти та додатки, які можуть бути легко використані всіма користувачами, незалежно від їхніх фізичних обмежень.

Застосування WAI-ARIA сприяє створенню інклюзивного цифрового середовища, де кожен може отримати доступ до інформації та сервісів без бар'єрів.

### Ролі елементів

У контексті WAI-ARIA, ролі елементів визначають тип контенту або функціональності, які вони представляють на веб-сторінці, допомагаючи асистивним технологіям, як-от екранні читачі, правильно інтерпретувати та взаємодіяти з контентом. Це сприяє підвищенню доступності для людей з обмеженими можливостями. Ось деякі категорії ролей у WAI-ARIA:

1. **Widget Roles:** Включають ролі для інтерактивних елементів інтерфейсу, таких як кнопки (button), перемикачі (radio), вкладки (tab), підказки (tooltip), елементи дерева (treeitem) та інші. Ми використовуємо ці ролі для вказівки на інтерактивні віджети на веб-сторінці, що дозволяє користувачам керувати ними за допомогою клавіатури або голосових команд.
2. **Composite Roles:** Описують складні віджети, які складаються з декількох елементів, наприклад, групи радіокнопок (radiogroup), дерево (tree), меню (menu), список вкладок (tablist). Використання цих ролей допомагає організувати більш складні структури інтерфейсу та забезпечити логічну навігацію між їх компонентами.
3. **Document Structure Roles:** Використовуються для опису структурного контенту на сторінці, включаючи статті (articles), заголовки (heading), групи (group), зображення (img). Ці ролі допомагають визначити структуру документа та забезпечити коректне сприйняття контенту користувачами асистивних технологій.
4. **Landmarks:** Це ролі, які допомагають користувачам асистивних технологій ідентифікувати основні частини веб-сторінки, такі як навігація (navigation), основний контент (main), бокові панелі (complementary), колонтитули (contentinfo) тощо. Використання landmark ролей сприяє кращій орієнтації на сторінці та ефективній навігації.

Використання цих ролей дозволяє нам створювати більш доступні веб-ресурси, які можуть бути легко навіговані та використані людьми з різними обмеженнями, сприяючи інклюзивності та рівному доступу до інформації та сервісів в інтернеті.

### Стани елементів

У контексті WAI-ARIA, ми використовуємо атрибути для вказівки станів елементів, визначення відносин між елементами та управління динамічним контентом. Ці атрибути дозволяють нам зробити веб-інтерфейси більш доступними для користувачів з обмеженими можливостями, зокрема для тих, хто використовує асистивні технології, як-от екранні читачі. Ось основні категорії цих атрибутів:

**Атрибути станів віджетів**

* **aria-hidden** вказує, чи елемент візуально прихований від користувача. Ми використовуємо його для управління відображенням елементів, щоб асистивні технології могли ігнорувати ті, які не мають бути доступними.
* **aria-checked** використовується для індикації стану вибраності елементів управління, таких як чекбокси або радіо кнопки.
* **aria-disabled** позначає, що елемент віджету вимкнено і не може взаємодіяти з користувачем.
* **aria-required** вказує, що елемент форми є обов'язковим для заповнення користувачем.

**Атрибути відносин**

* **aria-describedby** дозволяє зв'язати елемент з описом, надаючи додаткову інформацію, яка не входить безпосередньо в контент елемента.
* **aria-labelledby** зв'язує елемент з іншим елементом, який містить його мітку, забезпечуючи зрозумілість та контекст.

**Регіони з динамічним контентом**

* **aria-live** використовується для позначення регіону, який автоматично оновлюється (наприклад, з повідомленнями про помилки або оновлення стану), і дозволяє асистивним технологіям отримувати оновлення в реальному часі без необхідності перезавантаження сторінки.

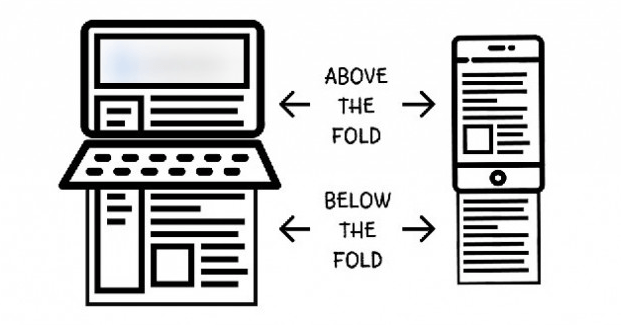
Ці атрибути відіграють ключову роль у забезпеченні доступності веб-додатків, дозволяючи користувачам з обмеженими можливостями краще навігувати та взаємодіяти з контентом, підвищуючи загальну інклюзивність та доступність вебу.

## Critical CSS

Критичний CSS – це мінімальний набір стилів, необхідних для того, щоб правильно відобразити верхню частину веб-сторінки (видиму частину без прокрутки). Ми використовуємо його для того, щоб забезпечити швидше завантаження вмісту сторінки для користувача. Основна мета – поліпшити швидкість і процес рендерингу сторінки, видаляючи будь-які перешкоди, які можуть сповільнити відображення важливого вмісту.

Інтегруючи critical CSS безпосередньо в HTML-документ, ми можемо змусити браузер завантажувати і відображати важливий вміст сторінки до того, як він завантажить весь CSS. Це особливо корисно для користувачів з повільними інтернет-з'єднаннями або на пристроях з обмеженою продуктивністю. Після завантаження та рендерингу критично важливого вмісту, браузер може продовжити завантаження решти ресурсів сторінки, таких як весь CSS, JavaScript, зображення тощо.

Використання critical CSS підвищує загальну швидкість завантаження сторінок, покращує взаємодію користувачів з сайтом і може позитивно вплинути на показники SEO, оскільки швидкість завантаження є одним з факторів ранжування в пошукових системах.



### Практичне використання

Critical CSS це та частина CSS, яка потрібна для відображення частини сторінки, видимої користувачу при першому завантаженні (above the fold). Вона має бути вбудованою безпосередньо в HTML, щоб сторінка могла відображатися швидше, оскільки браузер не муситиме чекати завантаження зовнішніх CSS-файлів.

Приклад того, як може виглядати використання critical CSS в HTML-документі:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

/\* Вставте сюди critical CSS \*/

body, h1, p {

margin: 0;

padding: 0;

}

/\* Базові стилі для верхньої частини сторінки (above the fold) \*/

header {

background-color: #333;

color: white;

padding: 20px;

text-align: center;

}

/\* Інші стилі, які є критично важливими для початкового вигляду сторінки \*/

</style>

</head>

<body>

<header>

<h1>Welcome to My Website</h1>

<p>This is above the fold content</p>

</header>

<!-- Вміст сторінки -->

<script>

/\* Ваш JavaScript \*/

</script>

</body>

</html>

Приклади плагінів для автоматизації процесу генерації і вбудовування critical CSS:

1. **Critical by Addy Osmani** - Node.js модуль для екстракції та управління critical-path CSS.
2. Webpack плагін **Critical** - плагін для Webpack, який інтегрує critical CSS у процес збірки.
3. Плагіни для WordPress, як Autoptimize або WP Rocket, які мають опції для оптимізації CSS та JavaScript, включно з генерацією critical CSS.

## Progressive CSS

Прогресивний CSS - це стратегія веб-дизайну, яка спочатку надає всім доступ до базового вмісту та функціональності веб-сторінки, а потім користувачам з додатковими можливостями браузера або швидкішим Інтернетом надається покращена версія.

Цей підхід до стилізації веб-сторінок дозволяє нам подолати обмеження традиційного CSS, покращуючи швидкість завантаження, SEO та підтримку старіших браузерів. Замість того, щоб застосовувати всі стилі одразу, ми розділяємо їх на базові (для всіх) та прогресивні (для сучасних браузерів).

Використовуючи прогресивний CSS, ми можемо створити різні рівні підтримки, які надаються на основі можливостей браузера: типографія; типографія та колір; та типографія, колір та макет.

Прогресивний CSS також допомагає в підтримці відмінностей між браузерами, використовуючи техніки, як-от "feature detection", де ми застосовуємо стилі на основі наявності певних функцій в браузері, замість того, щоб спиратися на визначення конкретного браузера чи його версії.

Цей підхід дозволяє нам покращити якість відображення сайту для всіх користувачів, незалежно від їх пристрою та браузера. Прогресивна CSS оптимізує UX та перформанс сайту, забезпечуючи мінімально необхідний рівень доступності та функціональності вмісту, поки повні стилі ще завантажуються.

Таким чином, прогресивний CSS дозволяє веб-сторінкам виглядати і функціонувати добре незалежно від того, чи користувач має найновіший браузер чи використовує застаріле обладнання або повільне інтернет-з'єднання. Це також підвищує шанси, що користувачі з різноманітними умовами перегляду матимуть позитивний досвід взаємодії з сайтом.

### Практичне використання

Прогресивний CSS може включати використання таких підходів:

1. **Feature Queries** (@supports) для застосування стилів у випадку, якщо браузер підтримує певні CSS властивості.
2. **CSS Custom Properties** (змінні CSS) для легкої кастомізації тем і стилів.
3. **Responsive Design** через медіа-запити для адаптації до різних розмірів екранів.
4. **Graceful Degradation** або **Progressive Enhancement** для забезпечення базового досвіду на старих браузерах та поступового покращення для новіших.

**Приклад використання:**

/\* Основні стилі, які застосовуються для всіх браузерів \*/

.button {

background-color: blue;

color: white;

padding: 10px 20px;

border: none;

transition: background-color 0.3s ease;

}

/\* Покращення для браузерів, які підтримують CSS змінні та grid layout \*/

@supports (display: grid) and (background-color: var(--color)) {

.container {

display: grid;

grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));

}

.button {

background-color: var(--primary-color, blue);

padding: var(--button-padding, 10px 20px);

}

}

/\* Медіа-запити для адаптації до мобільних пристроїв \*/

@media (max-width: 768px) {

.button {

padding: 8px 15px;

}

}

У цьому прикладі, базові стилі для кнопки застосовуються завжди, але якщо браузер підтримує CSS змінні та grid layout, то застосовуються додаткові стилі. Медіа-запити використовуються для адаптації кнопки до мобільних пристроїв.

**Приклади плагінів:**

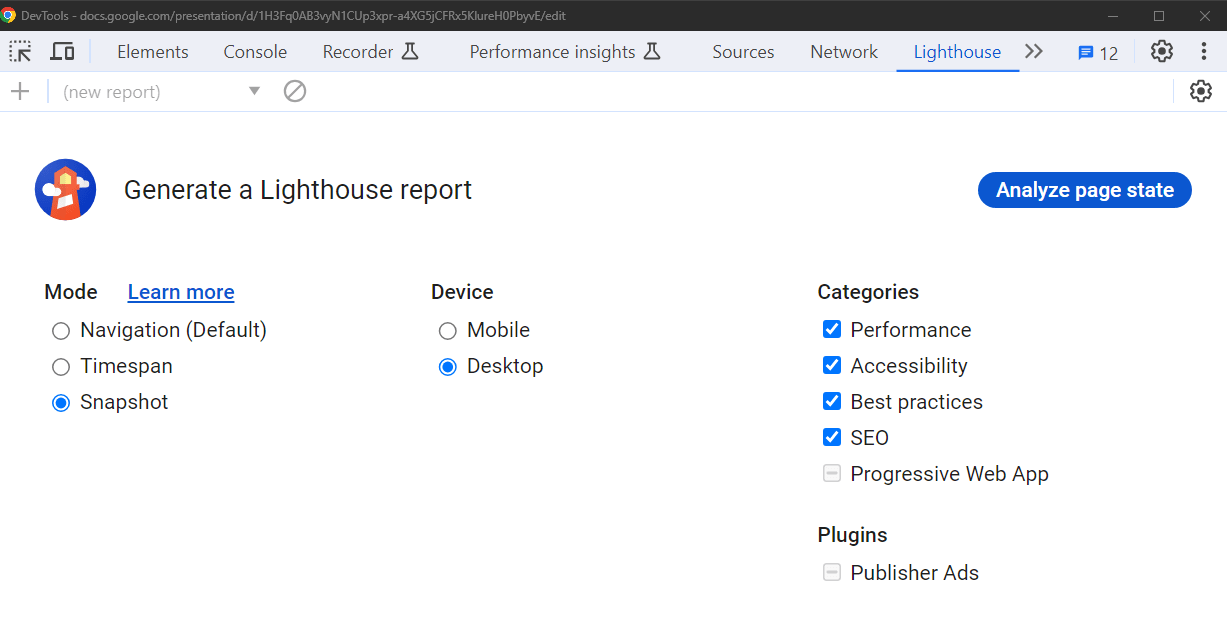
* **PostCSS** і його плагіни, такі як:
* postcss-preset-env для використання майбутніх CSS фіч.
* autoprefixer для автоматичного додавання вендорних префіксів.
* cssnano для мініфікації CSS.
* **Stylelint** для лінтування CSS коду з метою забезпечення консистентності і уникнення помилок.
* **Modernizr** для feature detection на стороні клієнта.

Для отримання інформації про найактуальніші плагіни у 2024 році, вам слід перевірити останні ресурси для розробників, репозиторії інструментів або відповідні ком'юніті.

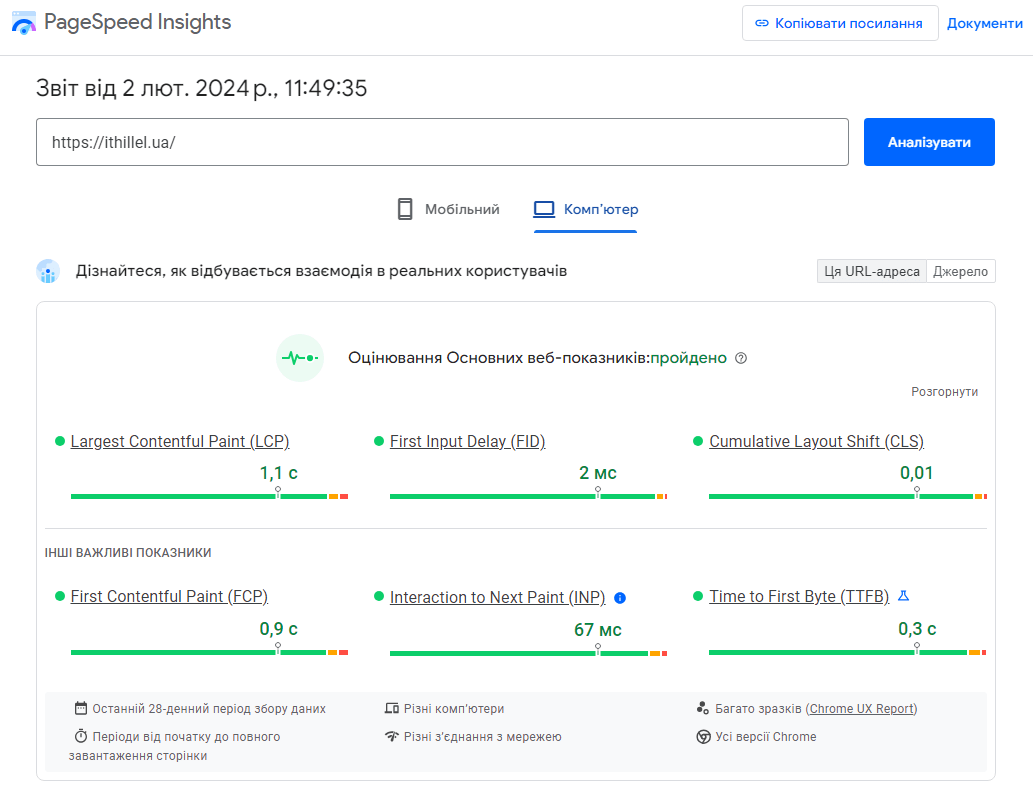
## Оптимізація шаблона (HTML, CSS project optimization)

Оптимізація веб-проектів є критичною для забезпечення кращого досвіду користувачів та підвищення ефективності сайту. Далі ми розглянемо ключові методики оптимізації HTML та CSS, які можуть допомогти уникнути спільних пасток, які сповільнюють завантаження та рендеринг веб-сторінок.

* **Уникнення білого екрана під час завантаження:** Належне використання критичного CSS і асинхронне завантаження дозволяють сторінці швидше відображатися, зменшуючи час до першого байта (Time To First Byte, TTFB – це метрика, яка вимірює час від моменту здійснення користувачем запиту на сторінку до моменту отримання першого байта відповіді від сервера).
* **Підключення файлів стилів і скриптів:** Розміщення посилань на CSS у <head> та скриптів перед </body>, використання async і defer можуть суттєво знизити час завантаження.
* **Використання одного стильового файлу і мініфікація:** Це зменшує кількість запитів до сервера та розмір файлів, прискорюючи час завантаження.
* **Аналіз вкладки Network у DevTools:** Дозволяє виявити файли, які потребують оптимізації або відкладеного завантаження.
* **Оптимізація зображень:** Правильний вибір форматів, стиснення і відкладене завантаження (lazy loading) зображень можуть значно покращити продуктивність.
* **Використання Lighthouse:** Цей інструмент аналізує веб-сторінки на предмет продуктивності, доступності та SEO, надаючи звіти та рекомендації.



* **Застосування PageSpeed Insights:** Сервіс від Google, який надає аналітику та поради щодо покращення швидкості завантаження веб-сторінок.



Ефективна оптимізація веб-сторінок вимагає уважного вивчення та застосування найкращих практик у кодуванні та ресурсному менеджменті. Враховуючи вищенаведені стратегії, ми можемо значно підвищити швидкість завантаження сторінок, покращити взаємодію користувачів із сайтом та посилити загальну продуктивність веб-проекту. Завжди важливо пам'ятати, що оптимізація – це безперервний процес, який потребує регулярного аналізу та оновлення, щоб забезпечити найкращий результат.