Андрій Овчаров

Рекомендації для ефективної веброзробки: від верстки до “WordPress”

Посібник містить практичні рекомендації з розробки вебсайтів.

Зміст:

Загальні технічні вимоги до вебсайту

1. Загальні вимоги:

- Сумісність з сучасними веб-браузерами: Google Chrome, Firefox, Safari, Microsoft Edge (останні стабільні версії).
- Адаптивний дизайн для коректного відображення на мобільних пристроях, планшетах та десктопах.
- Оптимізована швидкість завантаження сторінок (бажано щоб він був в зеленій зоні)
- Використання валідного HTML5 та CSS3 коду відповідно до стандартів W3C.

2. Вимоги до користувацького інтерфейсу (UI/UX):

- Інтуїтивно зрозуміла навігація.
- Чітка типографіка та контрастність текстів.
- Кнопки заклику до дії (Call-to-Action) повинні бути видимими та привертати увагу.
- Дотримання принципів доступності WCAG 2.1 (рівень AA).

3. Функціональні вимоги:

- Форми зворотного зв'язку (з перевіркою на коректність введених даних).
- Пошуковий функціонал (за наявності великої кількості контенту).
- Інтеграція зі сторонніми сервісами (Google Maps, аналітика, соціальні мережі).
- Можливість багатомовності сайту (опціонально).

4. Вимоги до безпеки:

- SSL-сертифікат для забезпечення безпечного з'єднання (HTTPS).
- Захист від SQL-ін'єкцій та XSS-атак.
- Регулярне створення резервних копій даних.
- Захист форм від спаму (reCAPTCHA або аналогічні рішення).

5. Вимоги до продуктивності:

- Використання сучасних інструментів оптимізації (Lazy Loading, оптимізація зображень).
- Підтримка кешування сторінок та статичних ресурсів.
- Мінімізація JavaScript та CSS файлів.

6. Вимоги до контенту:

- Семантична структура HTML для SEO-оптимізації.
- Актуальний контент з можливістю легкого редагування.
- Підтримка мультимедіа (зображень, відео, аудіо).

7. Вимоги до серверної інфраструктури:

- Підтримка PHP 8.0+ та баз даних MySQL або PostgreSQL.
- Використання хмарного хостингу або VPS з високою доступністю (uptime не менше 99.9%).
- Моніторинг і логування помилок.

8. Вимоги до тестування:

- Перевірка коректної роботи на різних пристроях і браузерах (кросбраузерне та кросплатформне тестування).
- Функціональне тестування (перевірка форм, кнопок, посилань).
- Перевірка продуктивності та швидкості завантаження.

Вимоги WCAG 2.1 (рівень AA)

Рівень **AA** стандарту WCAG 2.1 забезпечує доступність веб-сайтів для більш широкого кола користувачів, включно з людьми з порушеннями зору, слуху, рухових функцій та когнітивних здібностей.

1. Принцип 1: Сприйнятність (Perceivable)

1.1 Текстові альтернативи:

- Надання текстових описів для всіх нетекстових елементів (зображення, діаграми, іконки) через атрибут alt.

1.2 Мультимедіа:

- Для відео з аудіо необхідно надавати субтитри або текстові транскрипти.
- Аудіоконтент без відео повинен супроводжуватись текстовими описами.

1.3 Адаптація:

- Контент має коректно відображатися незалежно від орієнтації пристрою (портретна або альбомна).
- Використання семантичної HTML-структури (`<header>`, `<main>`, `<section>`, `<footer>`) для покращення доступності.

1.4 Відмінність контенту:

- Контраст між текстом і фоном має становити щонайменше 4.5:1 для звичайного тексту і 3:1 для великого тексту (18px і більше).
- Можливість зміни розміру тексту на 200% без втрати функціональності.
- Не використовувати кольори як єдиний спосіб передавання інформації (наприклад, додавати підкреслення для посилань).

2. Принцип 2: Операбельність (Operable)

2.1 Доступність через клавіатуру:

- Усі функціональні елементи сайту мають бути доступними та функціональними без миші, лише через клавіатуру.

2.2 Досить часу:

- Додавати можливість подовження часу для виконання дії або вимкнення таймерів.

2.3 Запобігання нападам епілепсії:

- Не використовувати контент із миготінням частотою понад три рази на секунду.

2.4 Навігація:

- Забезпечити видимість поточного фокусу (наприклад, рамка навколо активного посилання).

- Надавати можливість пропуску повторюваного контенту (наприклад, посилання «Пропустити до основного контенту»).

- Створювати описові заголовки сторінок і коректні назви посилань.

3. Принцип 3: Зрозумілість (Understandable)

3.1 Читабельність:

- Надання тексту зрозумілою мовою, зрозуміла структура заголовків і абзаців.

- Визначення основної мови контенту через атрибут `<html lang="uk">`.

3.2 Передбачуваність:

- Елементи інтерфейсу мають поводитися передбачувано (наприклад, меню розкривається за натисканням).

3.3 Допомога при заповненні форм:

- Надання текстових підказок для полів форми.

- Показ повідомлень про помилки з інструкціями щодо їх виправлення.

4. Принцип 4: Надійність (Robust)

4.1 Сумісність:

- Код веб-сайту повинен бути валідним і сумісним з допоміжними технологіями, такими як екранні зчитувачі.

- Використання атрибутів `aria-label, aria-describedby` для покращення доступності елементів.

Рекомендації з SEO та доступності

Ці рекомендації сприятимуть не лише поліпшенню SEO вашого сайту, але й забезпечать доступність для всіх користувачів, незалежно від їхніх можливостей.

1. **Оптимізація мета-тегів**: Додайте мета-теги, які описують зміст сторінки (наприклад, `meta name="description"`). Вони допоможуть покращити видимість сайту в результатах пошукових систем.

2. **Семантична розмітка HTML**: Використовуйте семантичні елементи, такі як `<header>`, `<main>`, `<footer>`, `<section>`, `<article>`, щоб полегшити індексацію та покращити доступність сайту.

3. **Оптимізація заголовків (H1, H2, H3)**: Використовуйте заголовки для структурування контенту. Основний заголовок повинен бути всередині тегу `<h1>` та лише один на сторінці, а інші заголовки повинні йти по порядку: `<h2>`, `<h3>` і т. д.

   Заголовки варто додавати до окремих блоків з контентом. Якщо не впевнений у приналежності заголовка, то використай тег `<span>`.

   Заголовки бажано використовувати залежно від контексту.

   Якщо підзаголовок має смислову ієрархію, використовуйте заголовок нижчого рівня:

   ```
   <h2>Title</h2>
   <h3>Subtitle</h3>
   <p>Text</p>
   ```

   Якщо підзаголовок є лише декоративним елементом, використовуйте звичайний тег, наприклад, <p> або , з відповідним класом для стилізації.

   ```
   <h2>Title</h2>
   <p class="subtitle">Subtitle</p>
   <p>Text</p>
   ```

4. **Використання ключових слів**: Інтегруйте відповідні ключові слова в заголовки та текст на сторінці, не забуваючи, що вони повинні виглядати природно.

   Встановлюйте **мета-теги** для покращення SEO. Важливі теги:

   - `<meta name="description" content="...">` – опис сторінки.
   - `<meta name="keywords" content="...">` – ключові слова.

5. **Альт-тексти для зображень**: Зображення повинні мати атрибути `alt`, що точно описують зміст зображення, щоб полегшити індексацію зображень і покращити доступність.

6. **Структура URL**: Створюйте короткі та чіткі URL-адреси, що містять ключові слова латиницею. Наприклад: www.example.com/tehnichni-vimogi.

Не пишіть кирилицею!

7. **Внутрішні посилання**: Використовуйте внутрішні посилання між сторінками сайту, щоб полегшити навігацію та допомогти пошуковим системам індексувати ваш контент.

8. **Оптимізація швидкості завантаження**: Використовуйте інструменти для перевірки швидкості (наприклад, Google PageSpeed Insights) і оптимізуйте ваші ресурси (зображення, CSS, JavaScript), щоб прискорити завантаження сторінок. Бажано щоб всі сторінки вашого сайту перебували в зеленій зоні.

9. **Мобільна оптимізація**: Переконайтесь, що сайт добре виглядає і працює на мобільних пристроях, адже це важливо для SEO. Використовуйте адаптивний дизайн для забезпечення комфортного перегляду на різних екранах.

10. **Мікророзмітка (Schema Markup)**: Додайте мікророзмітку до важливих елементів, таких як відгуки, ціни, події, щоб покращити видимість у пошукових системах і збагачення результатів.

11. **Безпека та зовнішні посилання**: Завжди додавайте атрибут `rel="noopener noreferrer"` до зовнішніх посилань, щоб підвищити безпеку, а також використовуйте HTTPS для захищеного з'єднання.

```
    <a href="https://facebook.com" target="_blank" rel="noopener
noreferrer">
    <img src="icons/facebook.svg" alt="Facebook"/>Facebook
    </a>
```

Рекомендації для написання чистого, ефективного та підтримуваного HTML-коду

Ці рекомендації допоможуть вам писати чистий, ефективний і доступний HTML, що сприятиме покращенню продуктивності сайту, забезпеченню кращої підтримки користувачів та підвищенню видимості в пошукових системах.

1. Структурованість та семантика

- Використовуйте **семантичні HTML-елементи**, щоб покращити доступність і SEO.

  Наприклад:

- `<header>, <footer>, <nav>, <section>, <article>, <aside>, <main>.`

- Визначте заголовки з використанням `<h1>, <h2>, <h3>` і так далі відповідно до ієрархії, що дозволяє забезпечити правильну структуру контенту.

- Використовуйте теги `<p>, <ul>, <ol>, <li>` для текстових блоків і списків.

```
<header>
    <h1>Назва веб-сайту</h1>
    <nav>
        <ul>
            <li><a href="#home">Головна</a></li>
            <li><a href="#about">Про нас</a></li>
            <li><a href="#contact">Контакти</a></li>
        </ul>
    </nav>
</header>
```

2. Мінімізація зайвих тегів

- Уникайте зайвих або непотрібних тегів, таких як `<div> і <span>,` якщо можна використовувати семантичні елементи.

- Використовуйте **CSS для стилізації** замість додавання додаткових HTML-елементів. Наприклад, замість використання `<div class="container">` для оформлення можна використовувати семантичні блоки.

3. Атрибути `alt` та `aria`

- Завжди додавайте **атрибут alt** до зображень для покращення доступності. Використовуйте описовий текст, що відображає зміст зображення.

  ```
  <img src="image.jpg" alt="Картинка з пейзажем на заході сонця">
  ```

- Для елементів, таких як кнопки, лінки та інші інтерактивні елементи, використовуйте **атрибути aria** для покращення доступності. Наприклад:

  ```
  <button aria-label="Закрити вікно">X</button>
  ```

- Якщо елемент не має текстового опису (наприклад, іконка), додавайте aria-`hidden="true":`

  ```
  <svg aria-hidden="true" width="24" height="24">
      <use href="#icon-twitter"></use>
  ```

```
</svg>
```

4. Іменування класів і атрибутів

- Використовуйте **чітке і описове іменування класів**, щоб код був легким для читання та підтримки. Рекомендується використовувати методологію **BEM** (Block, Element, Modifier):

```
<div class="button button--primary">Кнопка</div>
```

- Використовуйте **атрибути для підвищення доступності** (наприклад, role, aria-label) там, де це необхідно, щоб полегшити взаємодію з контентом для людей з обмеженими можливостями.

5. Валідація та коректність

- Завжди перевіряйте ваш HTML на **валідність** за допомогою інструментів, таких як **W3C Validator**.

- Не забувайте про **закриваючі теги**. Хоча HTML5 дозволяє деякі теги не закривати (наприклад, <li>), краще завжди дотримуватись правильного синтаксису.

6. Структура документу

- Використовуйте правильну структуру HTML-документу, включаючи всі необхідні метатеги, наприклад, `<meta charset="UTF-8">` для визначення кодування:

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <meta name="description" content="Опис веб-сайту">
    <title>Назва сторінки</title>
</head>
<body>
    <!-- контент сторінки -->
</body>
</html>
```

7. Доступність та UX

- Забезпечте **доступність для користувачів з обмеженими можливостями**. Використовуйте **табличні структури** тільки для даних, а не для макетів.

- Забезпечте, щоб усі інтерактивні елементи (наприклад, кнопки, лінки) були доступні через клавіатуру та мали зрозуміле описове повідомлення або підказку (якщо це необхідно).

8. Мобільна адаптивність

- Використовуйте **метатег viewport** для забезпечення адаптивності сторінки на мобільних пристроях:

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

- Використовуйте **відповідні одиниці вимірювання** для шрифтів і відступів, щоб забезпечити хорошу адаптивність на різних екранах (наприклад, rem та em).

9. Оптимізація зображень

- Використовуйте **формат зображень WebP** або **SVG**, коли це можливо, щоб зменшити розмір файлів.

- Не забувайте додавати **атрибут alt** для всіх зображень, навіть якщо вони не мають текстового опису (наприклад, логотипи).

11. Коментарі та документація

- Додавайте **коментарі** до складних або нестандартних частин коду, щоб інші розробники могли легко зрозуміти вашу логіку.

```
<!-- Меню навігації -->
<nav>
    <ul>
        <li><a href="#home">Головна</a></li>
    </ul>
</nav>
<!-- Меню навігації end -->
```

12. Використання сучасних атрибутів

- Використовуйте нові HTML5 атрибути, такі як:
    - `<input type="email">` для полів електронної пошти.

- `<input type="tel">` для номерів телефонів.
- `<input type="date">` для вибору дат.
- `<progress>` для відображення індикатора прогресу.

13. Оптимізація коду

- **Мінімізуйте HTML** для продуктивності, видаляючи зайві пробіли та коментарі під час фінальної версії.

- Використовуйте **відкладене завантаження скриптів** (attribut async або defer), щоб прискорити завантаження сторінки.

14. Аналіз продуктивності

- Використовуйте інструменти для перевірки швидкості завантаження сторінки (наприклад, **Google PageSpeed Insights**) і оптимізуйте ваш HTML-код відповідно.

15. Міжнародна підтримка

- Використовуйте атрибут lang в тегу <html> для вказівки мови сторінки:

```
<html lang="uk">
```

16. Вірно оформлюйте лінки та іконки

1. Якщо на сайті є кнопки то бажано додати `aria-label=" "` з описом куди вона веде.

2. Якщо це лінк на соціальну мережу то:

```
<a href="https://facebook.com" aria-label="Facebook"
target="_blank" rel="noopener noreferrer">
    <img src="icons/facebook.svg" alt="" />
</a>
```

3. Якщо іконки разом із текстом:
Якщо є іконка + текст, то атрибут `aria-label` вже не потрібний.

```
<a href="https://facebook.com" target="_blank" rel="noopener
noreferrer">
    <img src="icons/facebook.svg" alt="Facebook" />
    Facebook
</a>
```

4. Якщо використовуєте SVG іконки:

```
<a href="https://twitter.com" aria-label="Twitter" target="_blank"
rel="noopener noreferrer">
    <svg aria-hidden="true" focusable="false" width="24"
height="24">
        <use href="#icon-twitter"></use>
    </svg>
</a>
```

Рекомендації для написання чистого, ефективного та підтримуваного CSS та SCSS-коду

Ці рекомендації допоможуть вам писати чистий, ефективний та підтримуваний CSS/SCSS-код, що покращить продуктивність, читабельність і підтримуваність проекту.

1. Чистий і зрозумілий код

- **Іменування класів і змінних**: Використовуйте зрозумілі та описові імена для класів, змінних, міксинів, функцій. Імена мають чітко відображати їхнє призначення.

- **Конвенції з іменування**: Використовуйте стандарти, такі як **BEM (Block, Element, Modifier)** для CSS-класів, щоб зберігати структуру та уникати конфліктів.

```
.button {
    &__icon {
        margin-right: 8px;
    }
    &--primary {
        background-color: blue;
    }
}
```

2. Модульність та повторне використання

- Розбивайте код на міні-частини за допомогою **модульного підходу**, використовуючи SCSS-файли, такі як _variables.scss, _mixins.scss,

_buttons.scss, _grid.scss тощо.

- Використовуйте **мікси**, **функції** та **змінні** для повторно використовуваних стилів, щоб полегшити підтримку та розвиток коду.

```
@mixin button($color) {
    background-color: $color;
    color: white;
    padding: 10px 15px;
    border-radius: 5px;
}

.button {
    @include button(blue);
}
```

3. Наслідування та використання властивостей CSS

- Використовуйте **CSS Flexbox** або **Grid** для створення гнучких та адаптивних макетів замість застарілих методів (наприклад, float).

- Не використовуйте застарілі властивості, такі як float для макетів, коли є новіші і ефективніші рішення.

4. Організація та структура коду

- Дотримуйтесь структури, яка дозволяє легко орієнтуватися в проекті. Це можуть бути:
    - Загальні стилі (фонові кольори, шрифти тощо)
    - Модулі та компоненти
    - Сторінки та секції
    - Адаптивність
    - Інтерфейсні елементи (кнопки, форми, іконки)

- Використовуйте коментарі, щоб позначити різні розділи коду.

```
// Секція для кнопок
.button { ... }
```

5. Іменування змінних та міксинів

- Використовуйте **SCSS змінні** для визначення кольорів, розмірів, відступів, щоб уніфікувати проект.

```
$primary-color: #3498db;
$font-size: 16px;
```

- Для міксинів іменування повинно бути чітким і зрозумілим. Наприклад, якщо міксин для кнопки, використовуйте назву button():

```
@mixin button($color) { ... }
```

6. Адаптивність

- Використовуйте медіа-запити для забезпечення адаптивності дизайну на різних пристроях.

- Для адаптивних макетів використовуйте **відносні одиниці виміру** (em або rem), щоб шрифти та елементи масштабувалися на різних екранах.

```
.container {
    width: 100%;
    max-width: 1200px;
    margin: 0 auto;
}

@media (max-width: 768px) {
    .container {
        width: 90%;
    }
}
```

7. Використання CSS препроцесорів (SCSS)

- **SCSS** дозволяє використовувати вкладені правила, що покращують читабельність та структуру коду, але не зловживайте вкладеністю більше ніж 2 рівні.

- Використовуйте **модифікатори** для налаштування різних варіантів елементів.

- Використовуйте **імпорти** для кращої організації та структурування коду.

```
@import 'variables';
@import 'buttons';

.card {
    background-color: $card-background;
    @include box-shadow(0 2px 4px rgba(0, 0, 0, 0.1));
}
```

8. Управління кольорами та шрифтами

- Визначайте палітри кольорів через змінні SCSS для зручності редагування та однотипності.

- Використовуйте **системи шрифтів** для визначення базових шрифтів через змінні.

```
$primary-color: #1abc9c;
$font-family: 'Helvetica', sans-serif;
```

9. Уникайте дублювання коду

- Створюйте загальні міксини або функції для часто використовуваних властивостей або конструкцій.

- Дотримуйтесь принципу DRY (Don't Repeat Yourself) та уникайте дублювання стилів.

10. Використання CSS властивостей замість JavaScript

- Використовуйте CSS для анімацій і переходів замість JavaScript, коли це можливо, щоб покращити продуктивність.

```
.button {
    transition: background-color 0.3s ease;
    &:hover {
        background-color: $hover-color;
    }
}
```

11. Оптимізація

- **Мінімізація**: Після завершення розробки застосовуйте інструменти для мінімізації файлів CSS, наприклад, **PostCSS** або **CSSNano**.

- **Стиснення зображень**: Оптимізуйте зображення перед завантаженням на сайт.

12. Підтримка браузерів

- Використовуйте інструменти, такі як **Autoprefixer**, для автоматичного додавання префіксів для підтримки старих версій браузерів.

```
npm install autoprefixer
```

13. Використання Flexbox і CSS Grid

- Використовуйте **Flexbox** для вирівнювання елементів по рядку або стовпцю, коли необхідно.

- Використовуйте **CSS Grid** для створення складних макетів, де елементи можуть бути розташовані в сітці.

```
.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
}
```

14. Покращення продуктивності

- Зменшуйте кількість DOM-елементів, до яких застосовуються складні стилі.

- Використовуйте **CSS-анімації** для покращення взаємодії з користувачем, але намагайтеся обмежити їх складність, щоб не впливати на продуктивність.

15. Використання змінних CSS (CSS Custom Properties)

- Якщо підтримка браузерів дозволяє, використовуйте **CSS Custom Properties** (змінні CSS) для зберігання значень, таких як кольори, шрифти, розміри тощо.

```
:root {
    --primary-color: #3498db;
    --font-size: 16px;
}

.button {
    background-color: var(--primary-color);
    font-size: var(--font-size);
}
```

16. Документація

- Документуйте складні частини стилів, використовуючи коментарі або спеціальні файли документації, щоб пояснити, як працює певний стиль або міксин.

17. Використання методологій для організації коду

- Якщо проект великий, використовуйте методології типу **SMACSS** або **OOCSS** для організації стилів, щоб легко масштабувати проект.

Практичні рекомендації для розробки вебсайтів із використанням Tailwind CSS

Ці рекомендації допоможуть ефективно використовувати можливості Tailwind CSS для створення швидких, адаптивних та сучасних вебсайтів із чистим кодом та оптимальною підтримкою користувацького досвіду.

1. Структура коду та організація класів

- **Логічний порядок класів:** Дотримуйтеся послідовності у розташуванні класів — спочатку розмір, потім колір, відступи та розташування.

  Приклад:

  ```
  <div class="p-4 bg-gray-100 text-center rounded-lg">Content</div>
  ```

- **Групування стилів:** Використовуйте @apply для створення багаторазових класів у файлах CSS.

  Приклад:

  ```
  .btn-primary {
    @apply bg-blue-500 text-white py-2 px-4 rounded;
  }
  ```

2. Оптимізація швидкості

- **JIT-компіляція:** Увімкніть Just-In-Time компіляцію для генерації лише необхідних класів.

- **Пурифікація стилів:** Налаштуйте видалення невикористовуваних стилів у production-версіях.

  Приклад у tailwind.config.js:

  ```
  module.exports = {
    purge: ['./src/**/*.html', './src/**/*.js'],
    darkMode: 'media', // або 'class'
  };
  ```

3. Адаптивний дизайн

- **Медіазапити:** Використовуйте префікси для різних розмірів екранів (sm:, md:, lg:, xl:, 2xl:).

  Приклад:

```
<div class="p-4 md:p-8 lg:p-12">Content</div>
```

Перелік кастомних префіксів для брейкпоінтів Taliwindcss:

sm — для екрану шириною 640px і більше.

md — для екрану шириною 768px і більше.

lg — для екрану шириною 1024px і більше.

xl — для екрану шириною 1280px і більше.

2xl — для екрану шириною 1536px і більше.

При цьому, класи, що використовуються без брейкпоінту, застосовуються до всіх екранів.

Класи з брейкпоінтом застосовуються лише до відповідної ширини екрану та вище.

```
<div class="p-4 sm:p-6 md:p-8 lg:p-12 xl:p-16">
Це елемент з адаптивними відступами
</div>
```

p-4: відступи для всіх розмірів екрану.

sm:p-6: для екранів шириною 640px і більше відступи будуть 6.

md:p-8: для екранів шириною 768px і більше відступи будуть 8.

lg:p-12: для екранів шириною 1024px і більше відступи будуть 12.

xl:p-16: для екранів шириною 1280px і більше відступи будуть 16.

Це дозволяє створити стиль, який підлаштовується під розмір екрану.

Якщо для вашого макету цього переліку префіксів бракує, створюйте свої в

tailwind.config.js:

```
module.exports = {
  theme: {
   extend: {
     screens: {
        'xs': '480px', // Новий брейкпоінт для маленьких екранів
        '3xl': '1600px', // Брейкпоінт для дуже великих екранів
      },
    },
  },
};
```

Приклад як використовувати новий брейкпоінт в HTML:

```
<div class="bg-gray-100 xs:bg-blue-500 3xl:bg-green-500 p-4">
  Контент для різних розмірів екранів
</div>
```

4. Робота з кольорами та темною темою

- **Кастомні кольори:** Додавайте свої кольори через конфігурацію tailwind.config.js:

  ```
  module.exports = {
    theme: {
      extend: {
        colors: {
          brand: '#1a202c',
        },
      },
    },
  };
  ```

- **Темна тема:** Використовуйте класи dark: для підтримки темної теми.

  Приклад:

  ```
  <div class="bg-white dark:bg-gray-800">Content</div>
  ```

5. Зручність читання та підтримка коду

- **Чистий HTML:** Уникайте надмірного використання класів; групуйте стилі через компоненти.
- **Коментарі:** Додавайте коментарі у файли конфігурації та кастомні CSS.

6. Відповідність вимогам Accessibility

- **ARIA-атрибути:** Використовуйте aria-label для кнопок та інтерактивних елементів.

  Приклад:

  ```
  <button class="bg-blue-500 p-2" aria-label="Submit form">Submit</button>
  ```

- **Фокусні стани:** Завжди забезпечуйте видимий фокус для інтерактивних елементів.

  Приклад:

  ```
  <button class="focus:outline-none focus:ring-2 focus:ring-blue-500">Click me</button>
  ```

7. Типографіка та шрифти

- Використовуйте утиліти типографіки для контролю розмірів шрифтів, міжрядкових відступів та ваги тексту.

  Приклад:

  ```
  <p class="text-lg font-semibold leading-6">Lorem Ipsum</p>
  ```

Рекомендації для написання ефективного, чистого та підтримуваного JavaScript-коду

Ці рекомендації допоможуть вам писати чистий, ефективний та підтримуваний JavaScript-код, який буде зрозумілим і зручним як для вас, так і для інших розробників.

1. **Чистий і зрозумілий код**

- **Іменування змінних та функцій**: Використовуйте зрозумілі імена для змінних, функцій та класів, які чітко відображають їхнє призначення. Наприклад, замість x або y, використовуйте `userName, totalAmount, isActive`.

- **Конвенції з іменування**: Використовуйте camelCase для змінних та функцій, PascalCase для класів і компонентів.

```
let userName = 'John';
function calculateTotal() { ... }
class UserProfile { ... }
```

2. **Уникайте глобальних змінних**

- Змінні, які не є необхідними глобальними, повинні бути обмежені локальним контекстом (у межах функцій чи блоків).

- Встановлюйте **strict mode** ('use strict';) для уникнення небажаних глобальних змінних.

```
'use strict';
let message = 'Hello!';
```

3. **Модульність**

- Структуруйте ваш код у вигляді невеликих модулів. Це полегшує тестування, відлагодження та повторне використання коду.

- Використовуйте ES6 модулі (import/export).

```
// module.js
export function greet(name) {
    return `Hello, ${name}!`;
}

// app.js
import { greet } from './module.js';
console.log(greet('John'));
```

4. **Уникайте повторення коду**

- Використовуйте функції та методи для повторюваних частин коду. Це спрощує обслуговування та забезпечує більшу гнучкість.

```
function calculateDiscount(price, discount) {
    return price - (price * discount);
}
```

5. **Чітка обробка помилок**

- Завжди обробляйте можливі помилки, особливо при роботі з асинхронними запитами, користувацьким ввідними даними чи API.
- Використовуйте try...catch для синхронного коду та .catch() або async/await для асинхронних операцій.

```
javascript
КопіюватиРедагувати
try {
    const result = await fetchData();
} catch (error) {
    console.error('Error fetching data:', error);
}
```

6. **Коментарі до коду**

- Додайте коментарі для складних або неочевидних частин коду, але не перебільшуйте. Код повинен бути максимально самодокументованим через зрозуміле іменування.

```
// Функція для обчислення податку
function calculateTax(price) {
    return price * 0.15;
}
```

7. **Застосування принципів DRY (Don’t Repeat Yourself)**

- Уникайте дублювання коду. Якщо частина коду використовується більше одного разу, винесіть її в окрему функцію або метод.

- Створюйте утилітні функції для повторюваних операцій.

8. **Асинхронний код**

- Використовуйте `async/await` для асинхронного коду замість `then()/catch()`, коли це можливо. Це робить код лаконічнішим і зручнішим для читання.

```
async function fetchData() {
    try {
        const response = await fetch('https://api.example.com');
        const data = await response.json();
```

```
            return data;
        } catch (error) {
            console.error('Error:', error);
        }
    }
```

9. **Оптимізація продуктивності**

- Уникайте надмірних циклів або операцій, які можуть негативно вплинути на продуктивність, особливо в реальному часі (наприклад, обробка подій у великих масивах).

- Використовуйте **debouncing** або **throttling** для оптимізації подій (наприклад, прокручування або введення).

10. **Тестування**

- Пишіть тести для вашого коду. Це може бути юніт-тестування або інтеграційне тестування, залежно від проекту.

- Використовуйте бібліотеки, такі як Jest або Mocha для юніт-тестування.

11. **Залишайте код гнучким і масштабованим**

- Створюйте код, який легко змінювати та розширювати. Уникайте жорстко закодованих значень і забезпечте можливість для розширення.

- Приділяйте увагу архітектурним принципам, таким як **SOLID**.

12. **Уникайте використання** `eval()`

- `eval()` може призвести до серйозних проблем з безпекою та продуктивністю, тому намагайтеся уникати його використання.

13. **Оптимізація пам'яті**

- Переконайтесь, що ви звільняєте ресурси після використання (наприклад, видаляти обробники подій чи інтервали, коли вони більше не потрібні).

14. **Стандарти кодування**

- Дотримуйтеся одного стилю кодування в команді. Ви можете використовувати лінтери (наприклад, ESLint) та форматувальники (наприклад, Prettier), щоб забезпечити консистентність коду.

15. **Не забувайте про доступність**

- Забезпечте, щоб ваш JavaScript не порушував доступність веб-сайту. Враховуйте користувачів з обмеженими можливостями та використовуйте доступні елементи інтерфейсу.

Рекомендації для написання чистого, ефективного та підтримуваного PHP-коду

Ці рекомендації допоможуть вам писати чистий, ефективний та безпечний PHP-код, що полегшує підтримку та розвиток проектів.

1. Чистий і зрозумілий код

- **Іменування змінних та функцій**: Використовуйте зрозумілі та описові імена для змінних, функцій та класів. Імена мають чітко відображати їхнє призначення, щоб інші розробники могли легко зрозуміти ваш код.

- **Для ініціалізації змінних, використовуйте `let` або `const`, замість `var`.**

- **Конвенції з іменування**: Використовуйте стандарт PSR-1 і PSR-2 для PHP, де функції та методи пишуться в стилі camelCase, а класи в стилі PascalCase.

```
$userName = 'John';
function calculateTotal() { ... }
class UserProfile { ... }
```

2. Модульність

- Розбивайте код на функції та класи для кращої організації та повторного використання. Це також полегшує тестування.

- Використовуйте **composer** для керування залежностями та автозавантаження класів через PSR-4.

```
composer require vendor/package
```

3. Уникайте глобальних змінних

- Мінімізуйте використання глобальних змінних, обмежуйте їх використання в локальних функціях. Замість глобальних змінних використовуйте ін'єкцію

залежностей або параметри функцій.

- Завжди намагайтеся працювати з даними, які передаються в функцію, а не з глобальними.

4. Використання ООП (Об'єктно-орієнтоване програмування)

- ООП дозволяє зробити код більш структурованим, повторно використовуваним і розширюваним.

- Використовуйте класи та об'єкти для інкапсуляції логіки. Дотримуйтеся принципів SOLID для написання ефективного ООП-коду.

5. Обробка помилок і винятків

- Важливо правильно обробляти помилки та винятки. Використовуйте конструкції try...catch для асинхронного та синхронного коду.

- Завжди реєструйте помилки та помилки бази даних у логах для подальшого відстеження та виправлення.

```
try {
    $result = $db->query('SELECT * FROM table');
} catch (Exception $e) {
    error_log($e->getMessage());
    echo 'Error: ' . $e->getMessage();
}
```

6. Чітка структура коду та коментарі

- Додавайте коментарі, особливо для складних або неочевидних частин коду. Коментарі повинні пояснювати, чому ви використовуєте певні рішення, а не просто повторювати те, що вже є в коді.

- Використовуйте PHPDoc для документування функцій та класів.

```
/**
 * Calculates the total price including tax.
 *
 * @param float $price The original price.
 * @param float $taxRate The tax rate.
 * @return float The total price including tax.
 */
function calculateTotal($price, $taxRate) {
    return $price * (1 + $taxRate);
```

```
}
```

7. Уникання дублювання коду (DRY)

- Пишіть функції для повторюваних операцій, щоб уникнути дублювання коду.
- Використовуйте шаблони проектування (наприклад, Singleton, Factory) для повторно використовуваних частин коду.

8. Безпека

- **Ін'єкції SQL**: Завжди використовуйте підготовлені запити або ORM для запобігання SQL-ін'єкціям.

```
$stmt = $db->prepare('SELECT * FROM users WHERE email = :email');
$stmt->bindParam(':email', $email);
$stmt->execute();
```

- **Захист від XSS**: Завжди очищайте введення користувача, яке відображається на веб-сторінках.

```
$safeOutput = htmlspecialchars($userInput, ENT_QUOTES, 'UTF-8');
```

- **CSRF**: Використовуйте токени для захисту від міжсайтових підробок запитів (CSRF).

9. Використання шаблонів (Templating)

- Використовуйте шаблонізатори, такі як **Twig** або **Blade**, для відділення логіки від представлення. Це полегшує підтримку та покращує читабельність.

```
echo $twig->render('index.twig', ['name' => 'John']);
```

10. Використання Composer та Автозавантаження

- Використовуйте **Composer** для управління залежностями та автоматичного завантаження класів згідно з PSR-4.
- Встановіть залежності та забезпечте автозавантаження:

```
composer install
```

11. Тестування

- Пишіть юніт-тести для вашого коду, використовуючи PHPUnit.
- Використовуйте тести для перевірки функцій, класів та методів.

```
phpunit --testdox
```

12. Оптимізація продуктивності

- **Кешування**: Використовуйте кешування для зберігання результатів запитів або обчислень, що часто повторюються, щоб зменшити навантаження на сервер.

- **Оптимізація запитів до бази даних**: Завжди оптимізуйте SQL-запити, уникаючи зайвих приєднань та підвищуючи ефективність.

- **Аналіз продуктивності**: Використовуйте інструменти профілювання коду, такі як **Xdebug**, для пошуку місць, де можна покращити продуктивність.

13. Сторонні бібліотеки та фреймворки

- Використовуйте перевірені бібліотеки та фреймворки для спрощення розробки, як-от **Laravel**, **Symfony**, **CodeIgniter**.

- Пам'ятайте, що сторонні рішення допомагають швидше реалізувати стандартні функціональності, але варто обережно ставитися до їхнього вибору.

14. Робота з великими проектами

- Для великих проектів застосовуйте **MVC** (Model-View-Controller) або інші архітектурні патерни, що дозволяють краще організувати код.

- Використовуйте системи контролю версій, такі як **Git**, для управління змінами в коді.

15. Логування та моніторинг

- Використовуйте інструменти для логування (наприклад, **Monolog**) для зберігання помилок, попереджень та інформаційних повідомлень.

- Використовуйте моніторинг для відстеження стану вашого серверу та додатку.

16. Сумісність з PHP версіями

- Переконайтесь, що ваш код сумісний з останньою стабільною версією PHP, використовуючи можливості нових версій, але не порушуючи сумісність з попередніми версіями, якщо потрібно.

Рекомендації для перенесення сайту з верстки на “WordPress” та створення нової теми

Ці рекомендації допоможуть вам перенести сайт з шаблону на WordPress без втрати функціональності та дизайну, при цьому забезпечуючи хорошу продуктивність та безпеку.

1. **Аналіз структури шаблону**

Перед тим як перенести шаблон на WordPress, важливо **зрозуміти структуру** HTML/CSS шаблону, який ви переносите. Зверніть увагу на:

- Основні сторінки (головну, контактну, сторінки продуктів чи послуг).
- Використання підключених стилів і скриптів.
- Структуру заголовків, текстів і медіафайлів.

2. Створення теми WordPress

Створіть нову тему WordPress, базуючись на шаблоні. Використовуйте наступні кроки:

- Створіть **style.css**, де вказуєте дані теми.
- Створіть **index.php**, **header.php**, **footer.php**, **sidebar.php** для основної структури шаблону.
- Розбийте шаблон на частини: переносите HTML-код із шаблону в WordPress-шаблон відповідно до PHP-шаблонів.

3. Конвертація HTML-коду в PHP

Переведіть статичний HTML-код в динамічний PHP-код, використовуючи стандартні функції WordPress:

- Замість статичних заголовків та навігації використовуйте wp_head(), wp_footer() для правильного підключення стилів, скриптів і мета-тегів.
- Замість статичних елементів меню використовуйте функцію wp_nav_menu() для відображення меню.

- Використовуйте **get_template_part()** для підключення частин шаблону, таких як заголовок, футер або бічна панель.

4. Робота з контентом

Переносіть контент із шаблону в WordPress:

- Створіть сторінки та пости для статичного контенту.
- Використовуйте **Custom Post Types (CPT)** для нестандартних типів контенту, таких як портфоліо, відгуки, продукти тощо.
- Використовуйте поля Custom Fields (поля для збереження додаткової інформації) для динамічного виведення контенту.

5. Робота з медіа-файлами

Переконайтесь, що всі медіа-файли (зображення, відео) правильно імпортуються в бібліотеку медіафайлів WordPress. Для цього:

- Використовуйте функцію **wp_upload_dir()**, щоб правильно вивести зображення в WordPress.
- Переконайтесь, що розміри зображень відповідають вимогам, і використовуйте вбудовані можливості WordPress для адаптивності.

6. SEO-оптимізація

При перенесенні сайту на WordPress переконайтесь, що всі важливі SEO-елементи не втрачаються:

- Використовуйте плагіни для SEO, такі як **Yoast SEO** або **Rank Math**, для налаштування мета-тегів, заголовків, URL та інших важливих аспектів SEO.
- Забезпечте правильне використання семантичних тегів і правильних заголовків (`<h1>`, `<h2>`, `<h3>`).

7. Перевірка і адаптація шаблону до мобільних пристроїв

Після перенесення шаблону на WordPress, перевірте, чи збереглась адаптивність:

- Переконайтесь, що шаблон використовує медіа-запити для коректного відображення на різних пристроях.

- Використовуйте **CSS-фреймворки**, такі як **Tailwind CSS** або **Bootstrap**, для оптимізації адаптивності.

8. Оптимізація швидкості

Оптимізуйте швидкість завантаження сайту:

- Використовуйте плагіни для кешування, наприклад **W3 Total Cache** або **WP Rocket**.

- Оптимізуйте зображення за допомогою плагінів, як **Smush** або **ShortPixel**.

- Мінімізуйте CSS і JS файли, використовуючи плагіни, такі як **Autoptimize**.

9. Безпека

Зберігайте безпеку сайту під час перенесення:

- Використовуйте SSL (HTTPS) для захисту переданих даних.

- Встановіть плагіни для безпеки, наприклад **Wordfence Security** або **Sucuri**.

- Регулярно оновлюйте ядро WordPress, плагіни і тему для забезпечення безпеки.

10. Налаштування редагування контенту

Враховуючи, що WordPress дозволяє користувачам редагувати контент через адмінпанель:

- Переконайтесь, що всі елементи контенту доступні для редагування (наприклад, через Gutenberg або ACF — Advanced Custom Fields).

- Якщо потрібна певна кастомізація контенту, можна створити власні блоки для Gutenberg або кастомні поля для керування контентом через панель адміністратора.

11. Перевірка сайту

Після перенесення перевірте сайт на:

- **Коректне відображення** на всіх браузерах.

- **Тестування швидкості** за допомогою інструментів, таких як Google PageSpeed Insights.

- **Тестування функціональності** (форми, кнопки, інтерактивні елементи).

- **Перевірка на помилки** (перевірка через інструменти для розробників).

12. Запуск на продуктивному сервері

Перед запуском сайту на продуктивному сервері:

- Перевірте правильність налаштувань URL, щоб уникнути помилок 404.

- Переконайтесь, що всі плагіни, шаблони та налаштування коректно працюють.

Рекомендації для розробки сайту на “WordPress” за допомогою плагіну “Elementor” та кастомної теми

Ці рекомендації допоможуть вам розробити та підтримувати високоякісні та ефективні сайти на WordPress за допомогою Elementor, забезпечуючи оптимальний функціонал, швидкість і зручність для користувачів.

1. Планування структури сайту

- **Структуруйте сайт** перед початком роботи з Elementor. Перш ніж розпочати розробку, створіть карту сайту, яка відображатиме основні розділи та сторінки. Це дозволить вам краще організувати контент і визначити, які елементи будуть використовуватись на кожній сторінці.

- Використовуйте **шаблони сторінок** Elementor для швидкої та ефективної роботи. Наприклад, стандартні шаблони для головної, контактної сторінки та блогу.

2. Вибір правильного шаблону (теми) для Elementor

- Вибирайте тему, сумісну з Elementor. Наприклад, **Hello Elementor** — це легка, мінімалістична тема, спеціально розроблена для роботи з Elementor.

Переконайтесь, що вибрана тема не має надмірного коду, який може вплинути на швидкість завантаження сайту. Тема має бути **легкою та швидкою**.

Кращі безкоштовні шаблони для лендінгу:

1. **Astra**
   - Легка, адаптивна та швидка тема з готовими демо для лендінгів.

2. **Hello Elementor**
   - Легка та швидка тема для розробки лендінгів

3. **Neve**
   - Сучасна та багатофункціональна тема для створення ефективних лендінгів.

4. **Hestia**
   - Односторінкова тема з стильним дизайном та сумісністю з Elementor.

5. **Phlox**
   - Пропонує анімації та макети для креативних лендінгів.

6. **Blocksy**
   - Легка тема із сучасними дизайнами для інтерактивних лендінгів.

3. Використання адаптивного дизайну

- **Перевіряйте адаптивність** на різних пристроях. Elementor має інструменти для редагування сторінок для мобільних телефонів, планшетів та десктопів, але потрібно регулярно перевіряти, як сторінка виглядає на різних розмірах екранів.

- **Налаштування мобільної версії**: зменшуйте розміри шрифтів, приховуйте або переміщуйте елементи, щоб покращити перегляд на мобільних пристроях.

4. Оптимізація швидкості

- Використовуйте **статичний контент**, коли це можливо, і мінімізуйте використання складних анімацій і ефектів, щоб не погіршити швидкість завантаження сайту.

- **Оптимізуйте зображення** за допомогою плагінів, таких як **Smush** або **ShortPixel**, щоб зменшити їхній розмір без втрати якості.

- Використовуйте плагіни для **кешування** (наприклад, **WP Rocket** або **W3 Total Cache**) для прискорення завантаження сторінок.

5. Оптимізація SEO

- Встановіть плагін для SEO, наприклад **Yoast SEO** або **Rank Math**, щоб автоматично генерувати мета-теги, заголовки і URL-адреси, оптимізовані для пошукових систем.

- Використовуйте **SEO-структуру заголовків** (h1, h2, h3), щоб полегшити індексацію вашого контенту пошуковими системами.

- Враховуйте SEO в контексті візуальних елементів, таких як **alt-атрибути для зображень**.

6. Використання глобальних стилів

- Використовуйте **глобальні стилі Elementor** для створення уніфікованого вигляду на всіх сторінках: встановіть стандартні шрифти, кольори та інтервали. Це дозволяє легко редагувати стилі на рівні всього сайту.

- Створіть **глобальні блоки** для повторюваних елементів (наприклад, хедери, футери, кнопки), щоб прискорити роботу та забезпечити єдність стилю на всіх сторінках.

7. Робота з віджетами

- Використовуйте **вбудовані віджети Elementor**, але не забувайте про ефективність. Використання занадто багатьох віджетів може уповільнити роботу сайту. Наприклад, використовуйте віджети для кнопок, форм, галерей і контактних даних лише тоді, коли це необхідно.

- **Створіть свої віджети** за допомогою інструментів як **Elementor Pro** для додавання специфічних функцій, яких немає в стандартній версії.

8. Використання плагінів Elementor

- Використовуйте плагіни для розширення функціоналу Elementor. Популярні плагіни включають:

    - **Elementor Custom Widgets** для додавання власних віджетів.

    - **Elementor Header & Footer Builder** для кастомізації хедерів та футерів.

    - **Essential Addons for Elementor** для додаткових елементів дизайну.

9. Забезпечення доступності (Accessibility)

- Elementor має вбудовані інструменти для полегшення створення доступних сайтів, але завжди перевіряйте контент за допомогою інструментів

доступності, таких як **WAVE** або **Axe**.

- Використовуйте правильні атрибути alt для зображень, додавати текстові альтернативи до кнопок і форм, а також забезпечуйте достатній контраст кольорів для користувачів з обмеженими можливостями.

10. Безпека сайту

- Переконайтесь, що ви регулярно **оновлюєте Elementor** та інші плагіни, щоб уникнути вразливостей безпеки.

- Використовуйте плагіни безпеки, наприклад **Wordfence** або **Sucuri**, для захисту від атак та зловмисного програмного забезпечення.

11. Використання Elementor Pro

- Якщо є потреба в додатковому функціоналі, розгляньте можливість використання **Elementor Pro**. Ця платна версія пропонує багато додаткових можливостей, таких як шаблони для тем, додаткові віджети, можливість редагування WooCommerce, глобальні віджети та багато іншого.

12. Тестування та перевірка

- Тестуйте сайт на різних браузерах і пристроях, щоб впевнитись, що все працює коректно. Elementor має вбудовані інструменти для попереднього перегляду на різних пристроях, але все ж краще перевіряти вручну.
- Перевірте **швидкість сайту** за допомогою інструментів на кшталт **Google PageSpeed Insights** або **GTmetrix** і виконайте оптимізацію, якщо це потрібно.

13. Резервне копіювання

- Регулярно робіть **резервні копії сайту**, особливо перед великими змінами. Це дозволить швидко відновити сайт у разі проблем.

Рекомендації для розробки сайту на “WordPress” з використанням плагіну “WooCommerce” та кастомної теми

Ці рекомендації допоможуть вам створити ефективний, функціональний та оптимізований інтернет-магазин на базі WooCommerce із чудовим досвідом для користувачів.

1. Планування структури магазину

- **Продумайте архітектуру сайту:** категорії, підкатегорії та атрибути товарів (наприклад, колір, розмір).

- Визначте, які сторінки потрібні: головна, магазин, сторінка товару, сторінка кошика, оформлення замовлення та сторінка "Дякуємо за замовлення".

- Заздалегідь підготуйте опис продуктів, якісні зображення та інформацію про способи доставки.

2. Вибір шаблону (теми) для WooCommerce

- Використовуйте теми, сумісні з WooCommerce, такі як **Astra**, **OceanWP**, **Hello Elementor**, або теми з офіційного репозиторію WordPress.

- Переконайтесь, що тема підтримує адаптивний дизайн та відповідає вимогам SEO.

Кращі безкоштовні шаблони для WooCommerce:

1. **Storefront**
   - Офіційна тема WooCommerce з відмінною оптимізацією для магазинів.

2. **OceanWP**
   - Відмінна підтримка WooCommerce, адаптивний дизайн та гнучкі налаштування.

3. **Astra**
   - Пропонує готові шаблони магазинів із чудовою швидкістю роботи.

4. **GeneratePress**
   - Легка та швидка тема з відмінною підтримкою WooCommerce.

5. **Zakra**
   - Прекрасно підходить для стильних магазинів із WooCommerce.

6. **Blocksy**
   - Має сучасні макети товарних сторінок для WooCommerce.

7. **Neve**
   - Підходить для малих і середніх магазинів із WooCommerce.

3. Налаштування основних сторінок магазину

- **Сторінка продукту:** використовуйте функціонал коротких описів, галерей товарів та варіативних товарів.
- **Сторінка оформлення замовлення:** зробіть її максимально зрозумілою та лаконічною. Видаліть непотрібні поля, якщо це не обов’язково.
- **Кошик:** передбачте можливість швидкого редагування кількості товарів або їх видалення.

4. Оптимізація швидкості сайту

- **Оптимізуйте зображення:** використовуйте плагіни, такі як **Smush** або **Imagify**, для зменшення розміру зображень.
- Встановіть плагіни для кешування, такі як **WP Rocket** або **LiteSpeed Cache**, щоб прискорити завантаження сайту.
- Мінімізуйте використання зайвих плагінів та анімацій.

5. SEO-оптимізація магазину

- Використовуйте плагіни для SEO, такі як **Yoast SEO** або **Rank Math**, щоб оптимізувати мета-теги товарів, заголовки та опис.
- Зосередьтесь на створенні унікальних описів товарів та оптимізованих URL-адрес.
- Додавайте **alt-теги** для всіх зображень товарів.

6. Налаштування способів оплати та доставки

- Інтегруйте популярні платіжні шлюзи: **LiqPay**, **Stripe**, **PayPal**, або локальні сервіси.
- Налаштуйте варіанти доставки (безкоштовна, фіксована вартість, розрахунок за вагою).
- Перевірте функціональність обраних способів оплати та доставки.

7. Оптимізація UX/UI магазину

- Переконайтесь, що кнопки «Купити» та «Оформити замовлення» легко знайти.
- Налаштуйте фільтри товарів (за ціною, брендом, категоріями).
- Використовуйте функцію «Швидкий перегляд» для товарів.

8. Безпека сайту

- Використовуйте плагіни для безпеки, такі як **Wordfence** або **Sucuri Security**.
- Переконайтесь, що на сайті використовується SSL-сертифікат.
- Регулярно оновлюйте WooCommerce, тему та плагіни.

9. Інтеграція з маркетинговими інструментами

- Налаштуйте **Google Analytics** для відстеження конверсій та поведінки користувачів.
- Встановіть плагіни для розсилок, такі як **MailChimp for WooCommerce**, для автоматизації електронних листів.
- Використовуйте плагіни для збору відгуків, наприклад **Customer Reviews for WooCommerce**.

10. Доступність та мобільна адаптивність

- Перевірте магазин на різних пристроях, щоб впевнитись, що він коректно працює на мобільних телефонах і планшетах.
- Оптимізуйте адаптивність кнопок, форм та зображень.

11. Функціонал для користувачів

- Додайте функцію облікового запису користувача для збереження історії покупок.
- Реалізуйте функцію «Список бажань» за допомогою плагінів, таких як **YITH WooCommerce Wishlist**.

- Переконайтесь, що користувачі можуть легко знайти інформацію про повернення товарів та політику конфіденційності.

12. Тестування магазину

- Проведіть повне тестування процесу купівлі, включаючи додавання товарів у кошик, оформлення замовлення та перевірку способів оплати.

- Перевірте відображення товарів і коректність їх фільтрації.

13. Резервне копіювання

- Регулярно створюйте резервні копії сайту за допомогою плагінів, таких як **UpdraftPlus** або **BackupBuddy**.