

Р. Ю. Чехместрук

Основы WEB-дизайну

Частина 1

Міністерство освіти і науки України
Вінницький національний технічний університет

Р. Ю. Чехместрук

Основи WEB-дизайну

Частина 1

Електронний навчальний посібник

Вінниця
ВНТУ
2026

УДК 004.92
Ч-56

Рекомендовано до видання Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 8 від 23.12.2025 р.)

Рецензенти:

С. В. Павлов, доктор технічних наук, професор ВНТУ

О. В. Войцеховська, кандидат технічних наук доцент, ВНТУ

В. Ф. Граняк, кандидат технічних наук доцент, ВДПУ

Чехместрук Р. Ю.

Ч-56 Основи WEB-дизайну. Частина 1 : навчальний посібник для студентів спеціальності [Електронний ресурс] / Чехместрук Р. Ю. – Вінниця : ВНТУ, 2026. – (PDF, 145 с.).

У навчальному посібнику наведено систематизований виклад теоретичних основ дисципліни «Основи WEB-дизайну» та рекомендовану літературу. Також розглянуто ключові поняття web-дизайну, етапи життєвого циклу web-проєкту, основи UX-проєктування, візуального дизайну, а також принципи семантичної верстки з використанням HTML і стилізації за допомогою CSS. Матеріал структуровано за розділами, що охоплюють теоретичні аспекти композиції, типографіки, теорії кольору, доступності та інклюзивного дизайну, а також практичні основи побудови адаптивних макетів із застосуванням Flexbox і CSS Grid. Посібник розроблено відповідно до навчальної програми дисципліни «Основи WEB-дизайну» і призначений для здобувачів вищої освіти, викладачів та всіх, хто опановує базові принципи сучасного web-дизайну.

УДК 004.92

Зміст

ПЕРЕДМОВА	4
РОЗДІЛ 1 ВСТУП У WEB-ДИЗАЙН.....	5
1.1 Основні поняття та ролі.....	5
1.2 Життєвий цикл розробки Web-проєкту	12
РОЗДІЛ 2 UX-ПРОЄКТУВАННЯ	19
2.1 Основи User Experience	19
2.2 Проєктування взаємодії (Interaction Design)	23
РОЗДІЛ 3 ВІЗУАЛЬНИЙ ДИЗАЙН.....	29
3.1 Основи композиції	29
3.2 Типографіка у Web-дизайні	35
3.3 Теорія кольору	43
3.4 Доступність (Accessibility) та інклюзивний дизайн	51
РОЗДІЛ 4 HTML: СТРУКТУРА ТА СЕМАНТИКА.....	60
4.1 Вступ до HTML.....	60
4.2 Семантичний HTML5	63
4.3 Основні HTML-теги.....	68
4.4 HTML-форми	75
РОЗДІЛ 5 CSS: СТИЛІЗАЦІЯ ТА LAYOUT.....	83
5.1 Основи CSS.....	83
5.2 Блокова модель (Box Model)	91
5.3 Текст та шрифти	96
5.4 Позиціонування елементів	104
5.5 Flexbox Layout	114
5.6 CSS Grid Layout.....	121
5.7 Псевдокласи та псевдоеlementи	127
5.8 Трансформації та переходи	137
ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	143

ПЕРЕДМОВА

Навчальний посібник «Основи WEB-дизайну. Частина 1», розроблено спеціально для здобувачів спеціальності 121 – «Інженерія програмного забезпечення» Вінницького національного технічного університету. Для майбутніх інженерів програмного забезпечення важливо зрозуміти, що вебдизайн — це не просте візуальне оформлення, а насамперед, проектування складних систем комунікації між людиною та машиною. Дизайн дає продукту «обличчя» та «характер» (зовнішній вигляд, зручність, інтуїтивність), тоді як програмування дає йому «життя» (функціональність, логіку). Пам'ятайте: поганий дизайн зробить продукт неможливим для використання, навіть якщо він бездоганно працює з технічного погляду, що підкреслює критичну важливість дизайну як невід'ємної частини інженерного процесу. Цей посібник охоплює фундаментальні аспекти, необхідні для створення сучасних, ефективних та доступних вебрішень: від знайомства з основними поняттями UI та UX, ролями в команді розробки та життєвим циклом Web-проєкту (Розділ 1); через заглиблення в основи UX-проектування, методи дослідження цільової аудиторії, створення Персон, Карт шляху користувача, проектування взаємодії, включно роботу зі скетчами та вайрфреймами (Розділ 2); до ключових знань з візуального дизайну (композиція, типографіка, теорія кольору), з особливою увагою до Доступності (Accessibility) та інклюзивного дизайну (Розділ 3). Технічна реалізація охоплена фундаментальними знаннями з HTML: структура та семантика, необхідними для структурної основи будь-якої вебсторінки, з акцентом на семантичні теги HTML5 та роботу з формами (Розділ 4) та CSS: Стилізація та Layout, де детально розглянуто блокову модель, позиціонування, а також сучасні системи компоновання Flexbox та CSS Grid (Розділ 5). Глибоке розуміння обох аспектів — UX/UI та технічної реалізації — дозволить вам створювати технічно досконалі продукти, якими приємно користуватися та які успішно вирішують завдання користувачів. Бажаємо Вам успіху у вивченні основ WEB-дизайну та написанні чистого, ефективного та семантично правильного коду!

РОЗДІЛ 1

ВСТУП У WEB-ДИЗАЙН

1.1 Основні поняття та ролі

Ласкаво просимо до курсу «Основи Web-дизайну»! Для вас, як для майбутніх інженерів програмного забезпечення, важливо з самого початку зрозуміти, що вебдизайн – це не мистецтво у чистому вигляді, хоча естетика і відіграє в ньому важливу роль. Це, насамперед, **проектування** – це проектування складних систем комунікації між людиною та машиною, в нашому випадку – вебсайтом або вебдодатком. Якщо програмування дає продукту «життя» (функціональність, логіку, обробку даних), то дизайн дає йому «обличчя» та «характер» (зовнішній вигляд, зручність, інтуїтивність). Користувач не бачить ваш елегантний код чи оптимізовані алгоритми, але він напряду взаємодіє з результатом вашої спільної з дизайнером роботи. Погана архітектура може зробити сайт повільним або нестабільним, але поганий дизайн зробить його неможливим для використання, навіть якщо він працює бездоганно з технічного погляду. Це підкреслює критичну важливість дизайну як невід'ємної частини інженерного процесу.

Web-дизайн – це комплексна діяльність, що охоплює значно більше, ніж просто візуальне оформлення. Вона містить: **проектування інформаційної архітектури**, що визначає, як організувати контент, щоб користувач міг легко та логічно знайти те, що йому потрібно, мінімізуючи когнітивне навантаження. Це також **проектування взаємодії (Interaction Design)**, яке деталізує, як користувач буде взаємодіяти з елементами сайту (кнопками, формами, меню, анімаціями) і який відгук він отримуватиме від системи. Далі йде **проектування інтерфейсу (User Interface Design)**, що відповідає за візуальне подання всіх елементів, щоб вони були не лише естетично привабливими, а й зрозумілими, інтуїтивними та відповідали психологічним моделям сприйняття людини. І, нарешті, не менш важливою є **технічна реалізація**, що забезпечує адаптивність дизайну до різних пристроїв (від смартфонів до широкоформатних моніторів), доступність для людей з обмеженими можливостями (Accessibility) та оптимізацію для швидкого завантаження, що є критичним для утримання користувачів та SEO. Еволюція вебдизайну пройшла шлях від простих текстових сторінок 90-х, через епоху Flash-анімацій та перевантажених графікою сайтів початку 2000-х, до сучасного мінімалізму, орієнтованого на мобільні пристрої, швидкість та максимальну зручність. Сьогодні хороший дизайн — це дизайн, який непомітний, він просто працює,

дозволяючи користувачеві зосередитися на контенті та своїх завданнях, а не на складнощах інтерфейсу.

У сучасному вебдизайні ви постійно будете зустрічати два терміни: **UI** та **UX**. Важливо розуміти їхню різницю та взаємозв'язок, адже це два найбільш вживані, і водночас найбільш плутані терміни у веброзробці. Вони не є взаємовиключними, а скоріше доповнюють один одного, формуючи цілісний досвід взаємодії з продуктом.

User Experience (UX), або досвід користувача, – це про **відчуття та враження**. Це сукупність всього досвіду, який отримує людина під час взаємодії з вашим продуктом, включно його використання, сприйняття, емоції та реакції. UX-дизайн фокусується на тому, щоб зробити продукт: **корисним** (він має вирішувати конкретну проблему користувача або задовольняти його потребу), **зручним** (ним має бути легко та ефективно користуватися, мінімізуючи зусилля), **бажаним** (його візуальний вигляд, брендінг та загальна взаємодія мають викликати позитивні емоції та бажання використовувати продукт знову), **доступним** (ним можуть користуватися люди з різними фізичними та когнітивними можливостями, що є вимогою інклюзивного дизайну) та **надійним** (користувач має довіряти вашому продукту та інформації, яку він надає). UX-дизайнер – це детектив, психолог та архітектор в одній особі. Його робота починається задовго до рисування першої кнопки і містить глибокі **дослідження**: аналіз конкурентів, інтерв'ю з потенційними користувачами, створення детальних «персон» (архетипів цільових користувачів з їхніми потребами та болями), аналіз сценаріїв використання. Далі йде **проектування**: створення карт шляху користувача (Customer Journey Maps), розробка інформаційної архітектури, створення скетчів (швидких начерків ідей), вайрфреймів (низькодеталізованих «креслень» сторінок, що показують розташування елементів без візуального оформлення) та інтерактивних прототипів. І, нарешті, критично важливим є **тестування**: перевірка прототипів та готового продукту на реальних користувачах для виявлення проблем та слабких місць, що дозволяє ітеративно покращувати досвід. Якщо уявити будівництво будинку, UX-дизайнер — це архітектор, який проектує фундамент, кількість кімнат, їхнє розташування, ширину коридорів, логістику переміщення мешканців, щоб їм було комфортно та функціонально жити. Він дбає про те, щоб будинок був міцним, зручним та відповідав потребам мешканців.

На рис. 1.1 показано концепцію UX (User Experience) як великої парасольки. Під парасолькою розміщені основні дисципліни, які охоплюють UX.

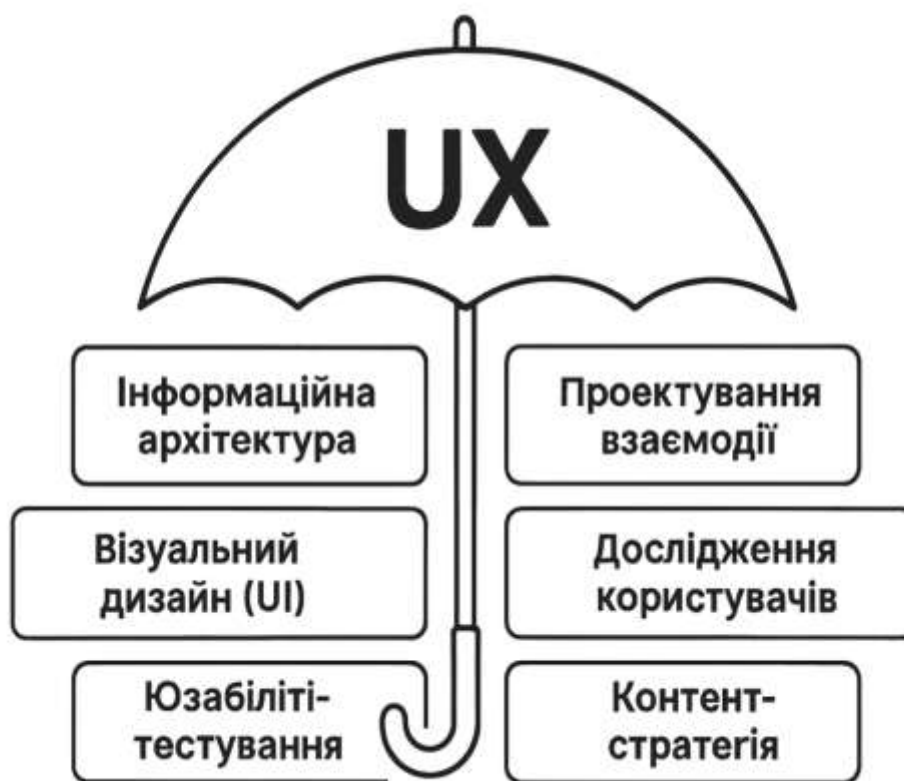


Рисунок 1.1 – Концепція UX (User Experience) як великої парасольки

Натомість, **User Interface (UI)**, або інтерфейс користувача, — це **візуальна частина** того, що було спроектовано на етапі UX. Це те, з чим користувач безпосередньо взаємодіє та що він бачить на екрані: кнопки, меню, шрифти, кольори, іконки, зображення, анімації. UI-дизайн фокусується на тому, щоб зробити інтерфейс: **зрозумілим** (користувач має інтуїтивно розуміти, що означає кожен елемент і що станеться після взаємодії з ним, без зайвих роздумів), **консистентним** (елементи дизайну — кнопки, іконки, кольори, типографіка — мають бути однаковими та передбачуваними на всіх сторінках сайту, створюючи відчуття цілісності), **привабливим** (естетика важлива для створення позитивного першого враження, довіри та задоволення від використання продукту). UI-дизайнер — це художник та декоратор, який працює в рамках, визначених UX. Він працює з **кольором** (створення гармонійних колірних палітр, що відповідають бренду та психології сприйняття), **типографікою** (підбір шрифтів, їх розмірів, накреслень та інтерліньяжу для забезпечення читабельності та візуальної ієрархії), **композицією** (розташування елементів на сторінці з урахуванням принципів балансу, контрасту, близькості та вирівнювання), **графікою** (створення іконок, ілюстрацій,

фотографій та інших візуальних елементів, що підсилюють повідомлення та естетику). Повертаючись до аналогії з будинком, UI-дизайнер — це дизайнер інтер'єру. Він вибирає колір стін, тип підлоги, розставляє меблі, підбирає світильники, картини та декор. Він робить простір красивим та гармонійним, спираючись на функціональний план архітектора. Для вас, як для розробників, важливо розуміти, що UX — це логіка і структура, а UI — це візуалізація цієї логіки. Ваш код буде реалізовувати і те, і інше, тому глибоке розуміння обох аспектів дозволить вам створювати технічно досконалі продукти, якими приємно користуватися та які успішно вирішують завдання користувачів. Пам'ятайте, що поганий UI може зруйнувати навіть найкращий UX, а хороший UI не врятує продукт з поганим UX.

Розробка вебсайту — це командна робота, де кожен учасник має свою унікальну роль та зону відповідальності, а успіх проекту залежить від їхньої злагодженої взаємодії. **Product Manager (Менеджер продукту)** є «капітаном корабля» або «візіонером». Він відповідає за стратегію продукту, визначає *що* потрібно робити, *для кого* і *навіщо*. Його завдання містить глибокий аналіз ринку, спілкування з клієнтами, формування бізнес-вимог та пріоритизацію функціоналу. Він є мостом між бізнесом та командою розробки. **UI/UX Designer**, як ми вже знаємо, відповідає за те, *як* продукт буде виглядати і відчуватися. Він перетворює абстрактні вимоги Product Manager у конкретні, візуальні та інтерактивні дизайн-макети та прототипи, проводячи дослідження, створюючи вайрфрейми, макети та тестуючи їх. **Frontend Developer** — це ви у найближчому майбутньому. Ви берете затверджені макети від дизайнера і перетворюєте їх на живий, інтерактивний код, що виконується у браузері користувача. Основні інструменти: HTML (для структури та семантики контенту), CSS (для стилізації та візуального оформлення) та JavaScript (для інтерактивності та динамічної поведінки). **Backend Developer** буде "мозок" продукту. Він працює на сервері, створює бази даних, розробляє API (Application Programming Interface), через які Frontend-частина обмінюється даними з сервером, забезпечуючи логіку роботи, безпеку та масштабованість. **QA Engineer (Інженер з якості)** є тестувальником. Його завдання — знайти помилки (баги) та невідповідності вимогам до того, як їх знайдуть кінцеві користувачі. Він перевіряє функціональність, юзабіліті, продуктивність та сумісність продукту на різних пристроях та у різних браузерах. У великих командах також можуть бути **DevOps інженери**, які відповідають за автоматизацію процесів розгортання, моніторингу та підтримки інфраструктури, забезпечуючи безперебійну роботу сервісу.

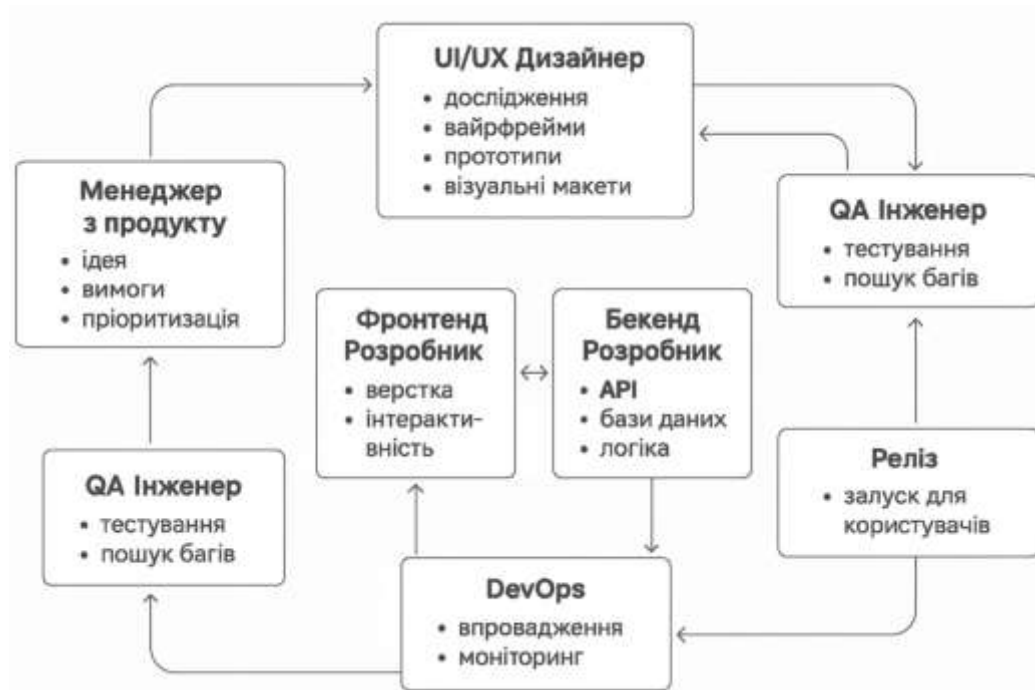


Рисунок 1.2 – Блок-схема, що показує спрощений процес роботи над новою функцією (фічею) у вебпроекті

Вам не потрібно вміти рисувати дизайн з нуля, але ви маєте вільно орієнтуватися в інструментах, з яких будете отримувати завдання та з якими будете взаємодіяти. Сучасні інструменти дизайну дозволяють не просто дивитися на картинку, а й отримувати всю необхідну технічну інформацію, що значно спрощує процес верстки та розробки. **Figma** є де-факто стандартом індустрії та лідером серед інструментів для UI/UX дизайну. Це хмарний сервіс, що не потребує встановлення (окрім шрифтів, якщо вони локальні), і працює прямо у браузері. Це забезпечує ідеальну спільну роботу в реальному часі: дизайнер рисує, а ви миттєво бачите зміни та можете взаємодіяти з макетом. У Figma ви, як розробник, будете активно використовувати режим **Inspect** для перегляду розмірів елементів, кольірних кодів (HEX, RGB, HSL), властивостей шрифтів (font-family, font-size, line-height), відступів (margin, padding). Figma автоматично генерує відповідний CSS-код для елементів, що є величезною допомогою. Функція **Export** дозволяє експортувати будь-яку іконку, зображення чи графічний елемент у потрібному форматі (SVG для векторної графіки, PNG для растрової з прозорістю, JPG для фотографій) та розмірі. Ви також можете залишати **Comments** безпосередньо на макеті, щоб уточнити деталі у дизайнера, а інтерактивні **Prototypes** дозволяють вам натискати на елементи та розуміти логіку переходів між сторінками та анімації, що є критичним для реалізації інтерактивної частини. Інші популярні інструменти, такі як **Sketch** (який довгий час був лідером, але працює лише

на macOS) та **Adobe XD** (частина екосистеми Adobe, що інтегрується з Photoshop та Illustrator), також використовуються, але Figma є лідером для командної роботи завдяки своїй кросплатформеності та функціям для спільної роботи. Ваше головне завдання як розробника — навчитися «читати» макет, тобто вміти взяти з нього всю необхідну інформацію для верстки, не турбуючи дизайнера через дрібниці, та розуміти, як дизайн має поводитися в різних сценаріях (наприклад, на мобільних пристроях або при зміні розміру вікна).

Завдання для демонстрації

1. Відкрийте сайт **Behance** (<https://www.behance.net>) або **Dribbble** (<https://dribbble.com>) (це портфоліо-платформи для дизайнерів), або будь-який інший сайт, що має якісний дизайн.

2. Знайдіть приклад веб-сайту, який вам подобається, і який має як десктопну, так і мобільну версію.

3. Спробуйте усно «декомпілювати» його дизайн, аналізуючи його з погляду UI та UX:

UX-аналіз:

1. Яка головна мета цього сайту?
2. Для кого він створений?
3. Які основні завдання користувач може вирішити на цьому сайті?
4. Наскільки легко знайти ключову інформацію (наприклад, контакти, ціни, кнопку «купити» або форму реєстрації)?
5. Чи є навігація інтуїтивно зрозумілою?
6. Чи є якісь елементи, які викликають роздратування або змушують задуматися?
7. Як сайт поводить у разі помилок (наприклад, неправильне введення у форму)?

UI-аналіз:

1. Яка кольорова палітра використовується (2-3 основних кольори, акцентні кольори)?
2. Які шрифти (один для заголовків, інший для основного тексту, їх розміри та накреслення)?
3. Як розташовані елементи на сторінці (композиція, сітки)?
4. Чи є елементи інтерфейсу послідовними на різних сторінках (кнопки, іконки, поля введення)?
5. Чи є візуальна ієрархія (що привертає увагу насамперед)?
6. Як виглядає мобільна версія — чи адаптується дизайн, чи є окрема мобільна верстка?
7. Які анімації використовуються і чи покращують вони досвід користувача?

Питання для самоконтролю

1. Поясніть своїми словами, чому вебдизайн є інженерною, а не суто художньою дисципліною, наводячи конкретні приклади з проєктування інформаційної архітектури та взаємодії.

2. Користувач скаржиться, що не може знайти кнопку «Кошик» на сайті інтернет-магазину, хоча вона візуально дуже приваблива. Це проблема UX чи UI? Обґрунтуйте свою відповідь, посилаючись на визначення цих понять.

3. Опишіть типовий робочий процес передачі завдання від дизайнера до Frontend-розробника з використанням Figma. Яку саме інформацію розробник може отримати з макета самостійно, не звертаючись до дизайнера?

4. Чому для Backend-розробника, який не працює з візуальною частиною, все одно може бути корисним розуміння UX-принципів? Наведіть приклад, як поганий UX може вплинути на роботу Backend.

5. Що таке «консистентність» в UI-дизайні і чому вона критично важлива для зручності користувача та довіри до продукту? Наведіть приклад порушення консистентності та його можливі наслідки.

6. Які основні переваги Figma перед іншими дизайн-інструментами для командної роботи та взаємодії з розробниками?

Рекомендоване практичне завдання

1. **Обов'язково.** Створіть акаунт на **Figma.com**. Пройдіть короткий вступний тур, який пропонує сервіс, та ознайомтеся з основними функціями інтерфейсу.

2. **Аналіз конкурентів.** Виберіть 2-3 сайти-конкуренти у будь-якій сфері (наприклад, два інтернет-магазини електроніки, три служби доставки їжі, два новинні портали). Проведіть їхній детальний порівняльний аналіз за критеріями:

- **Перше враження (UI).** Який сайт виглядає найбільш сучасно, професійно та викликає найбільше довіри? Опишіть, які елементи дизайну (кольори, шрифти, композиція) створюють це враження.
- **Простота навігації (UX).** На якому сайті вам було найлегше знайти каталог товарів/послуг, інформацію про доставку/оплату, контакти? Оцініть кількість кліків, необхідних для досягнення ключових розділів.
- **Ефективність ключової дії (UX).** Наскільки легко виконати цільову дію (наприклад, додати товар у кошик, оформити замовлення, підписатися на розсилку)? Чи є якісь перешкоди або зайві кроки?
- **Мобільна версія (Технічна реалізація та UX).** Відкрийте всі вибрані сайти на своєму смартфоні. Чи зручно ними користуватися? Чи адаптується дизайн під розмір екрана, чи є елементи, які важко натиснути або прочитати? Порівняйте досвід використання на

десктопі та мобільному пристрої.

- Результати занотуйте у вигляді детального звіту (мінімум 1 сторінка тексту), обґрунтовуючи свої висновки конкретними прикладами.

3. Практика у Figma

- Знайдіть у розділі «Community» простий, безкоштовний «UI Kit» (наприклад, за запитом «Material Design UI Kit» або «iOS UI Kit»).
- Створіть новий файл у Figma та спробуйте на новому полотні (фреймі) зібрати з готових компонентів:
 - картку товару (зображення, назва, ціна, короткий опис, кнопка «Додати в кошик»);
 - форму реєстрації (поля для імені, email, пароля, кнопка «Зареєструватися»);
 - елемент навігації (наприклад, верхнє меню з кількома пунктами та логотипом).
- Спробуйте також використати режим «Inspect» для одного з елементів, щоб побачити, як Figma генерує CSS-властивості.

1.2 Життєвий цикл розробки Web-проєкту

Створення вебсайту, як і будь-якого складного програмного забезпечення, є багатогранним і багатоетапним процесом, а не одноразовою дією. Цей процес, відомий як **Життєвий цикл розробки програмного забезпечення (Software Development Life Cycle, SDLC)**, описує послідовність фаз, через які проходить продукт від зародження початкової ідеї до його остаточного виведення з експлуатації. Розуміння цих етапів є критично важливим для ефективного планування ресурсів, управління часом, мінімізації ризиків та забезпечення злагодженої роботи команди, дозволяючи уникнути хаосу та непередбачених проблем. Кожен етап SDLC має свої цілі, завдання та очікувані результати, і їхнє послідовне та якісне виконання є запорукою успіху проєкту.

Традиційно, SDLC містить такі ключові фази: спочатку відбувається **планування та аналіз вимог**, де збирається вся необхідна інформація від замовника та майбутніх користувачів, визначаються бізнес-цілі проєкту, цільова аудиторія та детальний функціонал майбутнього сайту, а також технічні та нефункціональні вимоги (наприклад, продуктивність, безпека, масштабованість). На цьому етапі створюються технічні завдання та специфікації. Далі йде **проєктування**, на якому створюється архітектура програмного забезпечення, розробляється інформаційна архітектура сайту (структура контенту та навігації), формується UX-дизайн (через створення вайрфреймів, прототипів та карт шляху користувача) та UI-дизайн (візуальні макети, дизайн-системи). Після цього настає етап **розробки (кодування)**, під час якого програмісти пишуть безпосередньо код на

основі затверджених макетів та технічних специфікацій, перетворюючи дизайн на функціональний продукт. За розробкою слідує **тестування**, метою якого є ретельна перевірка продукту на наявність помилок (багів), відповідність початковим вимогам, коректність роботи на різних пристроях, у різних браузерах та за різних умов. Успішно протестований продукт переходить до фази **розгортання (запуску)**, коли сайт публікується на хостингу, налаштовуються сервери, домени, і він стає доступним для кінцевих користувачів. І, нарешті, життєвий цикл завершується етапом **підтримки та оновлення**, що передбачає постійний моніторинг роботи сайту, виправлення помилок, які можуть виникнути під час експлуатації, додавання нового функціоналу, оновлення контенту та адаптацію до змін у технологіях та вимогах ринку. Цей етап є безперервним і забезпечує довгострокову життєздатність продукту.

Існують різні методології, які по-різному організовують ці етапи, адаптуючи їх під специфіку проєкту та команди. Серед них варто виділити дві основні групи: класичну модель «Водоспад» та сучасні гнучкі підходи, такі як Agile та Scrum.

Модель «Водоспад» (Waterfall) є лінійним, послідовним підходом, де кожен етап строго слідує за попереднім. Це нагадує потік води, що падає зі сходинки на сходинку, і повернення на попередній етап є вкрай складним, дорогим або взагалі непередбаченим. Процес тут чітко визначений: Аналіз → Проєктування → Розробка → Тестування → Запуск. Кожен етап має чітко визначені вхідні та вихідні дані, які мають бути повністю завершені перед переходом до наступного. Перевагами такого підходу є його чітка структура, можливість точного прогнозування термінів та бюджету на самому початку проєкту, а також велика кількість детальної документації, що може бути корисною для великих, регульованих проєктів. Однак, головним недоліком є низька гнучкість: якщо на пізніх етапах виявляється, що початкові вимоги були неправильними, неповними або змінилися, внести корективи стає дуже важко і дорого, оскільки це потребує повернення на попередні фази. Крім того, кінцевий продукт користувач бачить лише в самому кінці циклу, що збільшує ризик невідповідності очікуванням. Ця модель досі використовується у проєктах з дуже чіткими, заздалегідь відомими і незмінними вимогами, наприклад, у деяких державних замовленнях, розробці медичного або військового програмного забезпечення, де зміни є вкрай небажаними або неможливими.



Рисунок 1.3 – Схематичне зображення моделі «Водоспад»

На противагу «Водоспаду», **Гнучка розробка (Agile)** – це філософія, що базується на наборі принципів, які пріоритезують гнучкість, співпрацю, швидку реакцію на зміни та постійне вдосконалення. Замість того, щоб намагатися реалізувати весь проєкт одразу, робота розбивається на короткі, керовані цикли, відомі як **ітерації** або **спринти**.

Scrum є найпопулярнішим фреймворком для впровадження Agile-підходу. У Scrum проєкт ділиться на невеликі, функціональні частини, над якими команда працює протягом фіксованих, коротких періодів (зазвичай 1-4 тижні), що називаються спринтами. Важливо, що в кінці кожного спринта команда зобов'язана мати готовий до показу, працюючий фрагмент продукту, який може бути продемонстрований замовнику. Це дозволяє отримувати постійний зворотний зв'язок та адаптувати напрямок розробки. Ключові ролі в Scrum містять **Product Owner** (відповідає за бачення продукту та пріоритизацію завдань), **Scrum Master** (фасилітує процес, усуває перешкоди) та **Development Team** (самоорганізована команда, що виконує роботу). Основні церемонії Scrum: **Планування**

спринта (визначення завдань на спринт), **Щоденний Scrum** (короткі зустрічі для синхронізації), **Огляд спринта** (демонстрація результатів замовнику) та **Ретроспектива спринта** (аналіз процесу та пошук шляхів покращення).

Переваги Agile та Scrum очевидні: висока гнучкість дозволяє вносити зміни «на льоту», замовник бачить результати дуже швидко і може надавати постійний зворотний зв'язок, а щоденна комунікація в команді сприяє швидкому вирішенню проблем та підвищенню якості. Недоліком може бути складність точного прогнозування фінальних термінів та бюджету на самому початку проєкту, а також вимога до високої самоорганізації команди та постійної взаємодії.

Проте, більшість сучасних вебпроєктів, особливо стартапи та проєкти з мінливими вимогами, активно використовують саме Agile-методології. Для вас, як для майбутніх інженерів, робота в Agile-команді потребуватиме відмінних комунікаційних навичок, вміння працювати в команді, готовності до постійних змін та здатності швидко адаптуватися, адже ви будете тісно співпрацювати з дизайнерами, тестувальниками та менеджерами щодня.

Переходячи від методологій розробки до креативного процесу, варто звернути увагу на **креативну стратегію Уолта Діснея**. Це не методологія розробки програмного забезпечення, а скоріше потужна **модель мислення**, яка допомагає структурувати процес генерації, оцінення та реалізації ідей. Вона дозволяє ефективно переходити від абстрактної ідеї до конкретного плану, використовуючи три послідовні ролі, які команда (або навіть одна людина) «приміряє» на себе. Ця стратегія є чудовим інструментом для подолання внутрішніх конфліктів між творчістю, реалізмом та критичним мисленням, дозволяючи кожній фазі проявитися повною мірою, не заважаючи іншим.

Перша фаза — це **Мрійник (The Dreamer)**. На цьому етапі немає жодних обмежень. Головне завдання — генерувати якомога більше ідей, навіть найфантастичніших, без будь-якої цензури чи самокритики. Критика на цьому етапі категорично заборонена, оскільки вона може задушити творчий потік. Ключове питання, яке ставить собі Мрійник: «Що, якби...?». Наприклад, для вебсайту це можуть бути такі ідеї: «Що, якби наш сайт вітав користувача по імені та пропонував персоналізований контент на основі його попередніх дій?», «Що, якби фон сайту змінювався залежно від часу доби, погоди за вікном або навіть настрою користувача, визначеного за його активністю?», «Що, якби ми зробили інтерактивний 3D-тур по нашому офісу або продукту прямо на головній сторінці, дозволяючи користувачам вільно досліджувати простір?» або «Що, якби навігація була у вигляді зоряної карти, де кожен розділ — це сузір'я, а перехід між ними супроводжувався б плавною анімацією?». На цьому етапі важливо зафіксувати всі ідеї щодо функціонала, дизайну, емоцій, які

має викликати сайт, а також можливих інтеракцій. Це чистий «мозковий штурм», спрямований на розширення горизонтів та пошук нестандартних рішень.

Після того, як Мрійник вичерпав усі свої ідеї, настає черга **Реаліста (The Realist)**. На цій фазі час спуститися з небес на землю. Реаліст бере ідеї Мрійника і починає думати, як їх можна втілити в життя, перетворюючи фантазії на конкретні плани дій. Він не критикує самі ідеї, а шукає конкретні шляхи їх реалізації, оцінюючи доступні ресурси, технології та терміни. Ключове питання Реаліста: «Як ми можемо це зробити?» Розглянемо приклади обробки ідей Мрійника: для ідеї «Вітати по імені та пропонувати персоналізований контент» Реаліст скаже: «Ми можемо це зробити. Нам потрібна система реєстрації/логіну з базою даних користувачів. Після входу користувача ми зберігаємо його ім'я та історію переглядів/покупок. Backend має створити API для отримання цих даних, а Frontend — зверстати сторінку з динамічним привітанням та блоками персоналізованих рекомендацій. Потрібно врахувати механізми кешування для швидкості». Щодо ідеї «Фон залежно від погоди або настрою»: «Це цікаво. Для погоди нам знадобиться інтеграція з API сервісу погоди (наприклад, OpenWeatherMap) та визначення геолокації користувача. Для настрою — це складніше, можливо, аналіз поведінки користувача на сайті або використання сторонніх бібліотек для аналізу тексту. Під час завантаження сторінки ми маємо асинхронно отримувати дані та динамічно змінювати CSS-властивості фону. Це може вплинути на продуктивність та SEO, тому потрібно буде оптимізувати завантаження фонових зображень та скриптів». А для ідеї «Інтерактивний 3D-тур»: «Технічно можливо за допомогою технологій WebGL, Three.js або Babylon.js. Потрібен 3D-моделер для створення сцени та Frontend-розробник з досвідом роботи з цими бібліотеками. Це дуже ресурсозатратно як на етапі розробки, так і для кінцевого користувача (високі вимоги до потужності пристрою та швидкості інтернету). Можливо, варто розглянути альтернативи, такі як панорамні фото 360° або відео-тур, які легше реалізувати та оптимізувати?». Реаліст створює покроковий план дій для кожної життєздатної ідеї, оцінюючи необхідні ресурси, технології, потенційні складнощі та можливі альтернативи.

Остання, але дуже важлива фаза — це **Критик (The Critic)**. Критик бере план Реаліста і намагається знайти в ньому слабкі місця, потенційні ризики, проблеми та недоліки. Його мета — не «вбити» ідею, а зробити її сильнішою, передбачивши всі можливі негаразди та підводні камені, які можуть виникнути на етапі реалізації або експлуатації. Ключове питання Критика: «Що може піти не так?» Продовжуючи наші приклади: для ідеї «Вітати по імені та персоналізований контент» Критик запитає: «А що, якщо у користувача дуже довге ім'я або ім'я містить спеціальні символи? Чи не зламається наша верстка? Які дані ми збираємо і як забезпечуємо

їхню конфіденційність згідно з GDPR/іншими нормами? Чи не буде персоналізація занадто нав'язливою? Що показувати незареєстрованому користувачу? Чи не буде це дратувати постійних відвідувачів, які не хочуть логінитись?». Щодо «Фону залежно від погоди або настрою»: «API погоди може бути платним, мати обмеження на кількість запитів або перестати працювати. Що ми покажемо в такому випадку? Чи не буде динамічний фон відволікати користувача від головного контенту? Чи варта ця фіча витрачених ресурсів, якщо вона може негативно вплинути на швидкість завантаження та доступність? Як це вплине на користувачів з фоточутливістю?». І для «Інтерактивного 3D-туру»: «Це точно не буде працювати на старих комп'ютерах, мобільних телефонах або у випадку повільного інтернету. Ми можемо втратити значну частину аудиторії. Чи є реальна бізнес-цінність у 3D-турі, чи це просто «вау-ефект»? Які альтернативи ми можемо запропонувати користувачам, які не можуть його завантажити? Які витрати на підтримку та оновлення 3D-контенту?». Після аналізу Критика ідеї, що «вижили» та були вдосконалені, повертаються до Реаліста для доопрацювання плану, враховуючи виявлені ризики та проблеми. Цей цикл може повторюватися кілька разів, доки ідея не стане максимально стійкою, продуманою та готовою до реалізації.

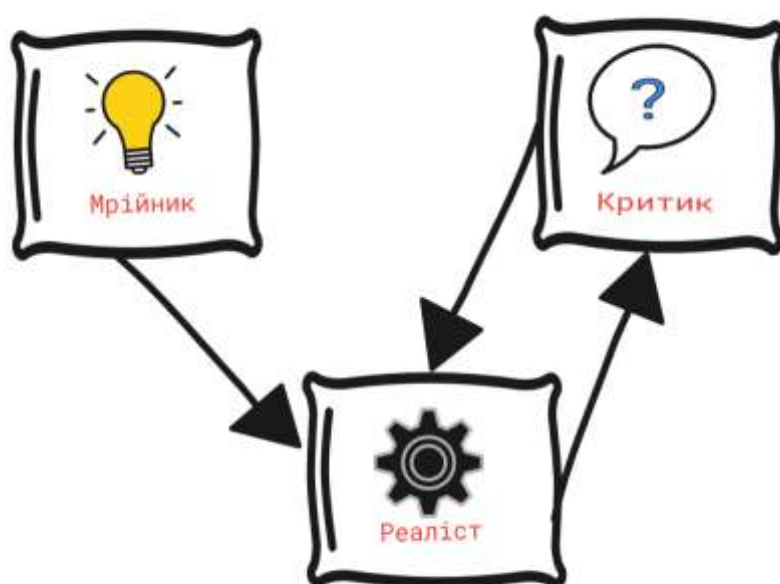


Рисунок 1.4 – Схема моделі Діснея. Три кімнати (або три зони), з'єднані стрілками в цикл.

Модель Діснея ідеально вписується в початкові етапи **Планування та Проєктування** SDLC. Фаза Мрійника відповідає початковому брейншторму з клієнтом та командою, де народжуються всі ідеї щодо майбутнього сайту, без обмежень. Роль Реаліста виконують UX-дизайнери та провідні розробники (Tech Lead), які перетворюють ці ідеї на конкретні вайрфрейми, прототипи та технічні специфікації, оцінюючи їхню реалістичність.

А Критик – це роль, яку виконують тестувальники, сам клієнт, і вся команда, аналізуючи прототипи та ставлячи питання: «Чи зручно це?», «Чи зрозуміло це?», «Чи не буде це занадто дорого в розробці?», «Чи відповідає це бізнес-цілям?». Використання такого підходу дозволяє відсіяти слабкі та нежиттєздатні ідеї на ранньому етапі, значно заощадивши час і гроші, та суттєво покращити якість фінального продукту, роблячи його більш стійким до викликів реального світу.

Завдання для демонстрації

Візьмемо ідею: «Створити веб-сайт для університетської бібліотеки».

1. **Мрійник:** Вся група накидує ідеї. Жодної критики.
2. **Реаліст:** Для 2-3 найцікавіших ідей пропонується план реалізації.
3. **Критик:** Інша група ставить питання до плану Реаліста.

Питання для самоконтролю

1. У чому ключова відмінність між Waterfall та Agile?
2. Опишіть роль кожного з трьох персонажів моделі Уолта Діснея.
3. На якому етапі SDLC найбільш доцільно застосовувати модель Діснея?
4. Що таке «спринт» у Scrum і який результат очікується в кінці кожного спринта?
5. Ідея Мрійника: «Давайте зробимо так, щоб курсор миші на нашому сайті залишав за собою слід із зірочок!». Які питання поставив би Реаліст? А які — Критик?

Рекомендоване практичне завдання

1. **Аналіз.** Виберіть будь-який великий вебдодаток. Спробуйте припустити, яку методологію розробки він використовує. Знайдіть одну функцію, яка вам не подобається, і зіграйте роль Критика.
2. **Практика моделі Діснея.** Придумайте ідею для простого мобільного додатку або вебсайту. Застосуйте фази Мрійника, Реаліста та Критика для її аналізу.

РОЗДІЛ 2 UX-ПРОЄКТУВАННЯ

2.1 Основи User Experience

У попередньому розділі ми вже торкалися поняття User Experience (UX) як сукупності відчуттів та вражень, які отримує користувач під час взаємодії з продуктом. Тепер ми заглибимося в цю тему, адже розуміння UX є фундаментальним для створення не просто функціональних, а й успішних вебсайтів та додатків. UX – це не лише про зручність, це про те, наскільки продукт відповідає потребам, очікуванням та емоціям користувача. Це дисципліна, що поєднує психологію, соціологію, дизайн та інженерію, щоб створити максимально ефективну та приємну взаємодію. Поганий UX може призвести до того, що користувач швидко покине сайт, навіть якщо він має унікальний функціонал. Навпаки, добре продуманий UX створює лояльність, підвищує конверсію та забезпечує позитивне «сарафанне радіо».

Основою будь-якого якісного UX-дизайну є глибоке розуміння цільової аудиторії. Неможливо створити продукт, який буде зручним для всіх, тому UX-дизайнери використовують різноманітні методи дослідження, щоб ідентифікувати та зрозуміти своїх користувачів. Ці дослідження допомагають відповісти на ключові питання: хто наші користувачі? Які їхні цілі? Які проблеми вони намагаються вирішити? Які їхні звички, мотивації та обмеження? Без цих знань дизайн буде базуватися на припущеннях, що часто призводить до створення продуктів, які нікому не потрібні або якими незручно користуватися. Серед найпоширеніших методів UX-досліджень можна виділити інтерв'ю з користувачами, опитування, фокус-групи, аналіз конкурентів, юзабіліті-тестування, аналіз веб-аналітики та, звичайно, створення персон та карт шляху користувача.

Одним з найефективніших інструментів для систематизації та візуалізації інформації про цільову аудиторію є **Персони**. Персона — це не реальна людина, а архетипний, вигаданий образ ключового користувача, створений на основі даних, отриманих під час UX-досліджень. Це детальний опис типового представника цільової аудиторії, який містить демографічні дані (вік, стать, освіта, професія), психографічні характеристики (цілі, мотивації, болі, страхи, звички, інтереси), технологічні навички та поведінкові патерни. Мета створення персон — зробити абстрактну аудиторію більш конкретною та емпатичною для команди розробки. Замість того, щоб говорити «наші користувачі», команда може звертатися до «Марії, 28 років, маркетолога, яка шукає швидкі рішення для аналізу даних». Це допомагає всім членам команди — від дизайнерів до розробників та менеджерів — краще зрозуміти, для кого вони створюють продукт, і приймати рішення, орієнтовані на реальні потреби. Зазвичай для одного проєкту створюють 3-5 ключових персон, які

є представниками різних сегментів аудиторії. Кожна персона має ім'я, фотографію (часто стокову), цитату, що відображає її ставлення, та детальний опис. Це дозволяє команді постійно тримати в голові образ користувача, для якого вони працюють, і перевіряти кожне дизайнерське чи інженерне рішення на відповідність його потребам.



Рисунок 2.1 – Приклад картки персони

Після того, як персони створені, наступним важливим кроком є візуалізація їхньої взаємодії з продуктом або сервісом. Для цього використовується **Карта шляху користувача (Customer Journey Map, CJM)**. CJM — це візуальне подання всього досвіду, який користувач проходить під час досягнення певної мети, взаємодіючи з продуктом або послугою. Вона охоплює всі точки контакту (touchpoints) — від першого усвідомлення потреби до завершення дії та подальшого досвіду. Карта шляху користувача допомагає команді побачити продукт очима користувача, виявити «болючі точки» (pain points), моменти розчарування або труднощів, а також можливості для покращення.

Типова CJM містить такі елементи:

1. **Етапи (Phases).** Основні кроки, які проходить користувач (наприклад, «Усвідомлення потреби», «Пошук інформації», «Вибір продукту», «Покупка», «Використання», «Підтримка»).
2. **Дії (Actions).** Що робить користувач на кожному етапі (наприклад, «Гуглить», «Читає відгуки», «Додає в кошик»).
3. **Думки (Thoughts).** Про що думає користувач на кожному етапі (наприклад, «Чи це надійно?», «Скільки це коштує?», «Чи легко буде повернути?»).
4. **Емоції (Emotions).** Які почуття відчуває користувач (радість, розчарування, збентеження, задоволення). Часто відображається графіком «кривої емоцій».
5. **Точки контакту (Touchpoints).** З чим або ким взаємодіє користувач (вебсайт, мобільний додаток, електронна пошта, телефонний дзвінок, соціальні мережі, фізичний магазин).
6. **Можливості (Opportunities).** Ідеї для покращення досвіду на кожному етапі, виявлені під час аналізу.

Створення CJM є спільним процесом, в якому беруть участь UX-дизайнери, продакт-менеджери, маркетологи та навіть розробники. Для інженерів розуміння CJM є надзвичайно цінним, оскільки дозволяє не просто кодувати функціонал, а розуміти його контекст, вплив на користувача та місце в загальному шляху.

Таблиця 2.1 – Приклад спрощеної карти шляху користувача (CJM).

Пошук	Вибір	Покупка
Дії: - Вивчення поширених сайтів -Пошук відгуків - Вибір товару/послуги	Дії: - Порівняння варіантів - Читання характеристик - Вибір конкретного товару	Дії: - Додавання до кошика - Вибір способу оплати - Оформлення замовлення
Думки: - «Знайду щось хороше» - «Чи це вигідно?»	Думки: - «Це відповідає моїм потребам» - «Чи швидко доставлять?»	Думки: - «Все просто» </br>- «Чи зручно оплатити?»
Емоції: - Зацікавленість - Невпевненість	Емоції: - Впевненість - Надія	Емоції: - Задоволення - Трохи хвилювання
Точки контакту: - Веб-сайт - Відгуки - Пошукові системи	Точки контакту: - Сайт - Чат - Телефонна підтримка	Точки контакту: - Кошик - Сторінка оплати - Підтвердження замовлення

Це допомагає приймати більш обґрунтовані технічні рішення, які дійсно покращують досвід, а не просто додають нові функції.

Наприклад, якщо CJM показує, що користувачі відчують розчарування на етапі заповнення форми, розробник може запропонувати технічні рішення для її спрощення, такі як автозаповнення, валідація в реальному часі або інтеграція з платіжними системами.

Емпатія – це ключове слово в UX-дизайні. Це здатність поставити себе на місце користувача, відчувати його проблеми та зрозуміти його потреби. Інструменти, такі як персони та CJM, допомагають розвивати цю емпатію в команді. Для інженерів програмного забезпечення, які часто зосереджені на логіці та ефективності коду, розвиток емпатії до користувача є важливим кроком до створення по-справжньому успішних продуктів. Це дозволяє перейти від мислення «як це працює?» до «як це відчувається для користувача?» і «як це допомагає йому досягти мети?». Розуміння UX-принципів дозволяє розробникам не просто реалізовувати технічні завдання, а бути активними учасниками процесу створення продукту, пропонуючи рішення, які покращують досвід користувача, а не лише задовольняють функціональні вимоги. Це також допомагає уникнути конфліктів між дизайнерами та розробниками, оскільки обидві сторони працюють над спільною метою – створенням найкращого продукту для користувача.

Завдання для демонстрації

1. Створення персони. Оберемо типового користувача для уявного вебсайту (наприклад, сайт для замовлення їжі). Спільно з аудиторією створимо одну персону, заповнивши її ключові характеристики: ім'я, вік, професія, цілі, болі, мотивації. Обговоримо, як ці характеристики впливають на дизайн сайту.

2. Розробка CJM. Для вибраної персони та сайту для замовлення їжі, спільно розробимо спрощену карту шляху користувача. Пройдемося основними етапами (наприклад, «Відчуття голоду», «Пошук ресторану», «Вибір страви», «Оформлення замовлення», «Очікування доставки», «Споживання їжі»), визначимо дії, думки, емоції та точки контакту на кожному етапі. Виявимо потенційні «болючі точки» та можливості для покращення.

Питання для самоконтролю

1. Чому UX-дизайн є настільки важливим для успіху вебпродукту?
2. Що таке «персона» і чому вона є ефективним інструментом у UX-дослідженнях?
3. Які основні елементи містить Карта шляху користувача (CJM)?
4. Як розуміння CJM може допомогти Frontend-розробнику у його роботі?
5. Поясніть поняття «емпатія в дизайні» і чому вона є ключовою для UX-дизайнера.

Рекомендоване практичне завдання

1. **Створення персони.** Виберіть будь-який вебсайт або мобільний додаток, яким ви часто користуєтесь. Створіть для нього 2-3 персони, які є представниками різних типових користувачів. Для кожної персони детально опишіть: ім'я, вік, професію, цілі, болі, мотивації, технологічні навички. Поясніть, як ці персони можуть впливати на дизайн та функціонал продукту.
2. **Розробка CJM.** Для однієї з створених персон розробіть детальну Карту шляху користувача для виконання типової задачі на вибраному сайті/додатку (наприклад, «замовлення товару», «пошук інформації», «реєстрація»). Визначте етапи, дії, думки, емоції та точки контакту. Виділіть «болючі точки» та запропонуйте 2-3 можливості для покращення досвіду на кожному етапі.

2.2 Проектування взаємодії (Interaction Design)

Раніше ми вже розглянули фундаментальні поняття UX та UI, а також життєвий цикл розробки вебпроекту. Сьогодні ми заглибимося в один з ключових аспектів UX-дизайну — **Проектування взаємодії (Interaction Design, IxD)**. Це дисципліна, що фокусується на створенні значущих взаємодій між користувачами та продуктами. Вона вивчає, як користувачі взаємодіють з інтерфейсом, як вони сприймають його поведінку, і як зробити цю взаємодію максимально інтуїтивною, ефективною та приємною. IxD відповідає на питання: «Як користувач буде виконувати завдання?», «Який відгук він отримає від системи?», «Як система реагуватиме на його дії?». Це не просто про те, як виглядають елементи, а про те, як вони *працюють* і *поводяться*. Це міст між візуальним дизайном (UI) та функціональністю (розробкою), що забезпечує логічний та послідовний досвід користувача.

Проектування взаємодії охоплює п'ять основних вимірів, які допомагають дизайнерам структурувати свою роботу та аналізувати взаємодію: **слова** (текст, що використовується в інтерфейсі, має бути чітким, зрозумілим, лаконічним та відповідати тону бренду; це містить мікротексти, підказки, повідомлення про помилки), **візуальні подання** (графічні елементи, іконки, зображення, типографіка, колірні схеми, що формують візуальний інтерфейс та направляють погляд користувача), **фізичні об'єкти або простір** (пристрої, з якими взаємодіє користувач – смартфон, планшет, десктоп, VR-гарнітура, а також фізичне середовище, в якому відбувається взаємодія), **час** (анімація, переходи, відео, звуки, що змінюються з часом і надають динаміки взаємодії, забезпечуючи зворотний зв'язок та покращуючи сприйняття) та **поведінка** (як продукт реагує на дії користувача, як він поводить себе в різних сценаріях, включно обробку помилок, завантаження даних, відповіді на введення).

Для інженера програмного забезпечення розуміння IxD є критично важливим, оскільки саме ви будете реалізовувати цю поведінку. Знання принципів IxD дозволить вам не просто кодувати функціонал, а створювати інтерактивні елементи, які дійсно покращують досвід користувача, а не викликають роздратування. Наприклад, правильна анімація може підтвердити дію користувача, зробити інтерфейс більш «живим» та зрозумілим, тоді як неправильна – заплутати, сповільнити його або навіть викликати дискомфорт. Це також стосується швидкості відгуку інтерфейсу: затримки у відповіді системи на дії користувача можуть значно погіршити досвід, навіть якщо візуально все виглядає добре.

Процес проєктування взаємодії часто починається з етапу ідеації та швидкої візуалізації ідей. Тут на допомогу приходять **скетчі** та **вайрфрейми**, які є першими кроками до матеріалізації абстрактних концепцій.

Скетчі (Sketches) – це найшвидший і найпростіший спосіб візуалізувати початкові ідеї інтерфейсу. Це чорнові, часто від руки нарисовані начерки, які не потребують художніх навичок. Їхня мета — швидко зафіксувати ідею, розташування основних елементів на екрані, логіку переходу між екранами та ключові функціональні блоки. Скетчі дозволяють експериментувати з великою кількістю варіантів за дуже короткий час, не витрачаючи ресурси на деталізацію чи досконалість. Вони є чудовим інструментом для мозкового штурму в команді, оскільки кожен може швидко нарисувати свою ідею, подати її та обговорити з колегами, сприяючи швидкому обміну думками та ітераціям. Для розробника скетчі можуть бути першим візуальним поданням майбутнього функціоналу, допомагаючи зрозуміти загальну концепцію та основні потоки користувача, ще до того, як буде створено детальний дизайн. Це дозволяє розробнику вже на ранніх етапах формувати уявлення про архітектуру компонентів та можливі технічні рішення.

Вайрфрейми (Wireframes) – це більш деталізовані, але все ще низькодеталізовані «креслення» сторінок вебсайту або додатку. Вони фокусуються виключно на структурі, розташуванні елементів, інформаційній ієрархії та функціональності, повністю ігноруючи візуальний дизайн (кольори, шрифти, зображення, стилі). Вайрфрейми часто називають «скелетом» або «каркасом» інтерфейсу, оскільки вони показують лише його структурну основу. Вони можуть бути нарисовані від руки, але частіше створюються за допомогою спеціалізованих програм, таких як Figma, Balsamiq, Axure, Adobe XD. Основна мета вайрфреймів — перевірити логіку навігації, розташування ключових елементів, потоки користувача та інформаційну архітектуру до того, як буде витрачено значний час та ресурси на візуальний дизайн. Вони слугують своєрідним «планом поверху» для веб-сторінки. Для розробника вайрфрейми є важливим документом, що показує, які елементи мають бути на сторінці,

де вони розташовані, як вони взаємодіють та які дані відображають. Це допомагає планувати структуру HTML, CSS та JavaScript, а також оцінювати складність верстки та інтеграції функціоналу. Вайрфрейми також є чудовим інструментом для комунікації між дизайнерами, розробниками та замовниками, дозволяючи всім сторонам узгодити функціонал та структуру до початку візуального оформлення.

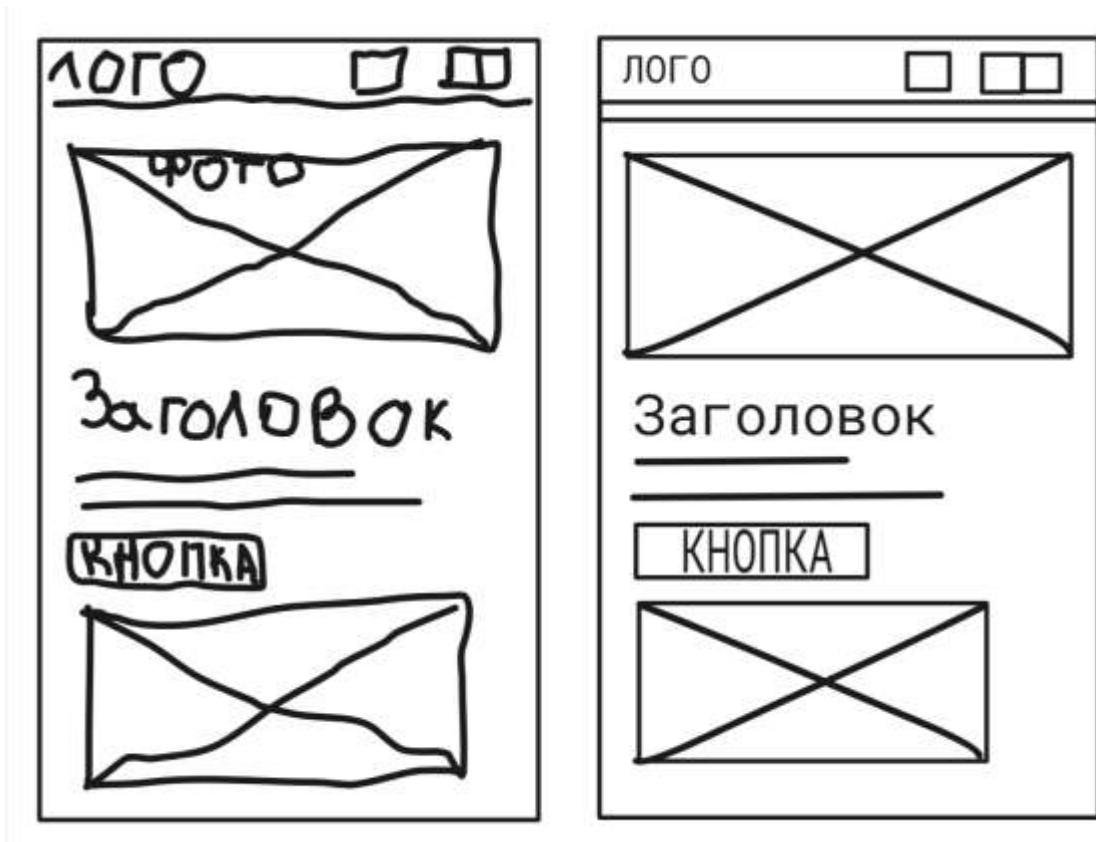


Рисунок 2.2 – Приклад скетчу та вайрфрейму для однієї і тієї самої сторінки

Після затвердження вайрфреймів настає етап створення **прототипів**. **Прототип** – це інтерактивна модель продукту, яка імітує його функціональність та взаємодію. Прототипи можуть бути різного рівня деталізації: від низькодеталізованих, що базуються на вайрфреймах і фокусуються лише на потоках користувача, до високодеталізованих, які майже не відрізняються від фінального продукту з погляду візуального оформлення та інтерактивності. Їхня головна мета – дозволити користувачам та замовникам протестувати продукт до початку розробки, виявити проблеми юзабіліті, зібрати зворотний зв'язок та перевірити гіпотези. Прототипування дозволяє заощадити значні кошти та час, оскільки виправляти помилки на етапі дизайну набагато дешевше, ніж на етапі розробки або після запуску продукту. Це також дозволяє швидко ітерувати та вдосконалювати дизайн на основі реальних даних. Для

розробника прототипи є живим, інтерактивним макетом, який показує, як саме має працювати інтерфейс, які анімації мають бути, як відбувається перехід між екранами, які стани мають мати елементи (наприклад, активна кнопка, неактивне поле). Це значно полегшує розуміння завдання та його реалізацію, дозволяючи розробнику передбачати складнощі та планувати архітектуру коду.

Figma є одним з найпотужніших та найпопулярніших інструментів для прототипування, що завоював лідерські позиції завдяки своїй хмарній природі та можливостям для спільної роботи. Вона дозволяє не лише створювати візуальні макети (UI-дизайн), але й легко перетворювати їх на інтерактивні прототипи, що імітують реальну поведінку додатка. Процес прототипування у Figma складається з кількох ключових кроків:

1. **Створення фреймів (екранів).** Кожен екран вашого сайту або додатка створюється як окремий фрейм. Figma пропонує готові шаблони фреймів для різних пристроїв (iPhone, Android, Desktop, Tablet), що спрощує початок роботи.

2. **Додавання та організація елементів.** Розміщення кнопок, полів введення, зображень, тексту, іконок та інших UI-компонентів на фреймах. Важливо використовувати компоненти та стилі для забезпечення консистентності й прискорення роботи.

3. **Визначення взаємодій.** За допомогою вкладки «Prototype» ви можете з'єднувати елементи одного фрейма з іншими фреймами. Наприклад, клік на кнопку «Увійти» може вести на фрейм «Особистий кабінет». Ви можете вказати тригер (наприклад, «On Click», «On Drag», «After Delay»), дію (наприклад, «Navigate to», «Open Overlay», «Swap With») та цільовий фрейм.

4. **Налаштування анімацій та переходів.** Figma дозволяє додавати різні типи анімацій (Smart Animate, Instant, Dissolve, Push, Slide, Move In/Out) для переходів між екранами, роблячи прототип більш реалістичним та приємним для використання. «Smart Animate» є особливо потужним інструментом, який автоматично анімує зміни між схожими елементами на різних фреймах.

5. **Тестування прототипу.** Ви можете запустити прототип у режимі презентації та взаємодіяти з ним, як з реальним додатком. Це дозволяє проводити юзабіліті-тестування, збирати зворотний зв'язок від замовників та потенційних користувачів, виявляти проблеми та швидко вносити зміни. Figma також дозволяє ділитися прототипами за посиланням, що спрощує процес ревізії.

Для розробника вміння працювати з прототипами Figma означає можливість не просто бачити статичний макет, а розуміти динаміку взаємодії, послідовність дій користувача та очікувану поведінку інтерфейсу. Це дозволяє більш точно реалізовувати функціонал, передбачати можливі проблеми та пропонувати оптимальні технічні

рішення. Наприклад, якщо прототип показує складну анімацію або перехід, розробник може заздалегідь оцінити її технічну складність, вибрати відповідні CSS-властивості (`transition`, `animation`, `transform`) або JavaScript-бібліотеки, а також запропонувати альтернативи, якщо це необхідно з погляду продуктивності або сумісності з браузерами. Розуміння того, як дизайнер задумав взаємодію, допомагає уникнути розбіжностей та прискорити процес розробки.



Рисунок 2.3 – Скріншот інтерфейсу Figma в режимі прототипування

Проектування взаємодії – це безперервний процес, який не закінчується на етапі прототипування або навіть запуску продукту. Навіть після розгортання важливо постійно аналізувати поведінку користувачів (за допомогою вебаналітики, теплових карт, записів сесій, А/В тестування) та ітеративно покращувати взаємодію. Для інженера це означає, що ваша робота не обмежується лише написанням коду. Ви є частиною команди, яка постійно прагне зробити продукт кращим для користувача. Розуміння IxD дозволяє вам бути більш проактивним, пропонувати власні ідеї щодо покращення взаємодії (наприклад, оптимізація швидкості завантаження, покращення відгуку інтерфейсу), ефективніше співпрацювати з дизайнерами, перетворюючи їхні ідеї на реальність, яка приносить задоволення користувачам. Це також сприяє створенню більш гнучкої та масштабованої архітектури коду, яка зможе легко адаптуватися до майбутніх змін у дизайні взаємодії. В кінцевому підсумку, успішний веб-продукт – це результат тісної співпраці між усіма членами команди, де

кожен розуміє та цінує внесок іншого, а головною метою є створення виняткового досвіду для користувача.

Завдання для демонстрації

1. **Створення скетчів.** Нарисуйте на папері 3-5 швидких скетчів для головної сторінки уявного сайту (наприклад, інтернет-магазину книг). Зосередьтеся на розташуванні основних блоків: шапка, навігація, банер, блок з товарами, футер.

2. **Створення вайрфреймів у Figma.** На основі одного зі скетчів створіть низькодеталізований вайрфрейм у Figma. Використовуйте прямокутники для блоків, лінії для тексту, плейсхолдери для зображень. Не додавайте кольори та шрифти.

3. **Прототипування у Figma.** Створіть ще один фрейм для сторінки товару. З'єднайте кнопку «Детальніше» на головній сторінці з фреймом сторінки товару за допомогою функції прототипування у Figma. Продемонструйте перехід.

Питання для самоконтролю

1. Що таке Проектування взаємодії (Interaction Design) і чим воно відрізняється від UI-дизайну?

2. Яка основна мета використання скетчів та вайрфреймів у процесі дизайну?

3. Чому прототипування є важливим етапом перед початком розробки?

4. Які можливості Figma роблять її потужним інструментом для прототипування?

5. Як розуміння Interaction Design допомагає Frontend-розробнику у його роботі?

Рекомендоване практичне завдання

1. **Створення вайрфреймів.** Виберіть будь-який існуючий вебсайт (наприклад, новинний портал, блог, особистий сайт). Створіть 3-5 вайрфреймів для його ключових сторінок (головна, сторінка статті/товару, сторінка контактів). Використовуйте Figma або інший інструмент для вайрфреймінгу. Зосередьтеся на структурі та розташуванні елементів, а не на візуальному оформленні.

2. **Прототипування.** На основі створених вайрфреймів створіть інтерактивний прототип у Figma. З'єднайте елементи навігації та кнопки, щоб імітувати переходи між сторінками. Додайте прості анімації (наприклад, «Smart Animate» або «Push») для переходів. Протестуйте свій прототип і переконайтеся, що навігація працює логічно.

РОЗДІЛ 3 ВІЗУАЛЬНИЙ ДИЗАЙН

3.1 Основи композиції

У веб-дизайні, як і в будь-якому візуальному мистецтві, **композиція** є ключовим елементом, що визначає, наскільки ефективно та привабливо буде сприйматися інформація. Композиція – це мистецтво та наука організації візуальних елементів на сторінці таким чином, щоб вони створювали єдине, гармонійне ціле, направляли погляд користувача та ефективно передавали повідомлення. Добре продумана композиція не лише робить сайт естетично приємним, але й значно покращує його юзабіліті та доступність, допомагаючи користувачеві швидко знаходити потрібну інформацію, розуміти структуру контенту та ефективно взаємодіяти з інтерфейсом. Для інженера програмного забезпечення розуміння принципів композиції є важливим для того, щоб не просто відтворювати макети дизайнера, а й розуміти логіку їх побудови, вміти адаптувати їх під різні розміри екранів, забезпечувати їхню гнучкість та, за необхідності, пропонувати технічні рішення, які підтримують та посилюють дизайнерські задуми. Це дозволяє створювати не просто функціональні, а й візуально привабливі та інтуїтивно зрозумілі інтерфейси.

Одним з найважливіших принципів композиції є **баланс**. Баланс у дизайні – це візуальна рівновага елементів на сторінці, що створює відчуття стабільності, гармонії та цілісності. Він запобігає відчуттю «перевантаженості» однієї частини сторінки та «порожнечі» іншої. Баланс може бути досягнутий кількома основними способами:

Симетричний баланс. Елементи розташовані рівномірно по обидва боки від центральної осі (вертикальної або горизонтальної), створюючи дзеркальне відображення. Це надає дизайну відчуття формальності, стабільності, надійності, авторитетності та порядку. Часто використовується в корпоративних сайтах, новинних порталах, банківських додатках, де важлива серйозність та довіра. Однак, надмірне використання симетрії може зробити дизайн передбачуваним, статичним та нудним, позбавивши його динаміки та оригінальності.

Асиметричний баланс. Елементи різного розміру, форми, кольору та візуальної ваги розташовані таким чином, що візуально врівноважують один одного, не будучи дзеркальним відображенням. Наприклад, великий, візуально важкий елемент з одного боку може бути врівноважений кількома меншими, але візуально активними елементами, з іншого. Асиметричний баланс створює більш динамічний, сучасний, цікавий та енергійний дизайн, але потребує більш тонкого відчуття пропорцій, візуальної ваги елементів та їхнього розташування. Він часто

використовується в креативних портфоліо, блогах, художніх галереях, де важлива оригінальність, привернення уваги та виразність.

Радіальний баланс. Елементи розташовані навколо центральної точки, створюючи відчуття руху від центра або до центра. Це може бути використано для створення фокусу або динаміки, наприклад, у кругових меню або діаграмах.

Для досягнення балансу дизайнери враховують візуальну вагу елементів, яка залежить від їхнього розміру (більші елементи важчі), кольору (темніші та яскравіші кольори мають більшу вагу), текстури, форми та розташування на сторінці (елементи ближче до центру мають меншу вагу, ніж ті, що на периферії). Розуміння балансу дозволяє розробнику правильно розподіляти елементи в адаптивній верстці, щоб сайт виглядав гармонійно та збалансовано на будь-якому пристрої, незалежно від розміру екрана.

Контраст – це різниця між елементами, яка привертає увагу, допомагає розрізняти їх та створює візуальний інтерес. Контраст є ключовим для створення візуальної ієрархії, забезпечення читабельності та акцентування уваги на важливих деталях. Існує кілька типів контрасту, які можуть бути використані як окремо, так і в комбінації:

Колірний контраст. Різниця між кольорами (наприклад, чорний текст на білому фоні, яскрава кнопка на нейтральному фоні, використання комплементарних кольорів). Важливо використовувати достатній колірний контраст для тексту та інтерактивних елементів, щоб забезпечити доступність для людей з порушеннями зору та відповідність стандартам WCAG (Web Content Accessibility Guidelines).

Розмірний контраст. Різниця у розмірах елементів (наприклад, великий заголовок та дрібний текст, велике зображення та маленькі іконки, велика кнопка дії та менші допоміжні посилання). Це допомагає виділити важливі елементи та створити візуальну ієрархію.

Типографічний контраст. Різниця у шрифтах, їхньому накресленні (жирний, курсив, світлий), розмірі, кольорі, стилі (Serif проти Sans-serif). Використання різних шрифтових пар або варіацій одного шрифту може створити цікавий та функціональний контраст, що покращує читабельність та візуальну привабливість.

Контраст форми. Різниця між геометричними (квадрати, кола) та органічними (плавні, природні) формами, гострими кутами та плавними лініями. Це може додати динаміки та інтересу до дизайну.

Контраст текстури. Різниця у візуальній або тактильній текстурі елементів (наприклад, гладка кнопка проти фону з зернистою текстурою). У вебі це частіше імітується візуально.

Ефективне використання контрасту дозволяє направляти погляд користувача, виділяти ключову інформацію, створювати фокусні точки та робити інтерфейс більш динамічним та цікавим. Для розробника це

означає точне відтворення розмірів шрифтів, колірних схем, відступів та інших стильових властивостей, щоб зберегти задуманий дизайнером контраст та забезпечити його функціональність.

Візуальна ієрархія – це організація елементів на сторінці таким чином, щоб їхня важливість була очевидною для користувача. Вона відповідає на питання: «Що я маю побачити насамперед?», «Що є найважливішим на цій сторінці?», «Який шлях я маю пройти, щоб знайти потрібну інформацію?». Візуальна ієрархія створюється за допомогою розміру, кольору, контрасту, розташування, негативного простору, типографіки, анімації та інших візуальних атрибутів. Наприклад, великий, жирний заголовок привертає увагу швидше, ніж дрібний текст. Яскрава, контрастна кнопка «Купити» буде помітнішою, ніж посилання на політику конфіденційності. Добре продумана візуальна ієрархія дозволяє користувачеві швидко сканувати сторінку, знаходити потрібну інформацію, розуміти структуру контенту та ефективно взаємодіяти з інтерфейсом, що є критично важливим для юзабіліті та конверсії. Без чіткої ієрархії сторінка виглядає хаотичною, і користувач губиться в інформації. Для розробника це означає точне дотримання стилів та розмірів, визначених дизайнером, щоб зберегти задуману ієрархію та забезпечити її правильне відображення на всіх пристроях.

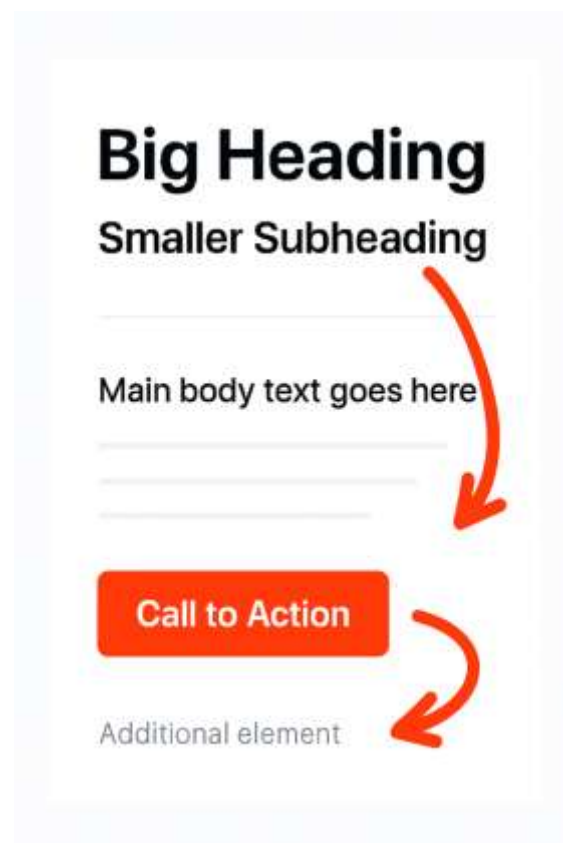


Рисунок 3.1 – Приклад сторінки вебсайту з чітко вираженою візуальною ієрархією

Сітки (Grid Systems) є фундаментальним інструментом для створення впорядкованої, гармонійної та послідовної композиції. Сітка – це система невидимих ліній (горизонтальних та вертикальних), які допомагають дизайнерам та розробникам вирівнювати елементи, створювати рівномірні відступи, забезпечувати послідовність у розташуванні контенту та підтримувати візуальну цілісність на різних сторінках. Використання сіток надає дизайну відчуття структурованості, професіоналізму, передбачуваності та значно полегшує його адаптацію під різні розміри екранів та пристроїв. Найпопулярнішими є колонкові сітки, де сторінка ділиться на певну кількість колонок (наприклад, 12 або 16), що дозволяє гнучко розміщувати елементи різної ширини, які займають певну кількість цих колонок. Існують також модульні сітки (з рівномірними горизонтальними та вертикальними модулями) та ієрархічні сітки (що базуються на візуальній важливості елементів). Для розробника сітки є основою для побудови адаптивної верстки за допомогою сучасних CSS-технологій, таких як CSS Grid або Flexbox. Розуміння сіток дозволяє ефективно перетворювати дизайн-макет на працюючий код, забезпечуючи точне вирівнювання, відступи та правильну поведінку елементів у разі змінення розміру екрана. Це також спрощує співпрацю з дизайнерами, оскільки обидві сторони говорять однією мовою структури.

Патерни читання (F/Z-патерни) описують типову поведінку погляду користувача під час сканування вебсторінки. Дослідження показують, що користувачі не читають вебсторінки повністю, слово за словом, а сканують їх у пошуках ключової інформації, заголовків, зображень та інтерактивних елементів. Два найпоширеніші патерни сканування:

F-патерн. Користувач спочатку сканує верхню частину сторінки горизонтально (перша «перекладина F»), потім трохи нижче знову горизонтально, але на меншу відстань (друга «перекладина F»), а потім вертикально вниз по лівому краю (ніжка «F»). Цей патерн характерний для сторінок з великою кількістю тексту (наприклад, новинні статті, блоги, результати пошуку), де користувач шукає ключові слова або заголовки. Важлива інформація та Call to Action мають бути розміщені вздовж цих ліній.

Z-патерн. Користувач починає з верхнього лівого кута, рухається горизонтально до верхнього правого, потім по діагоналі до нижнього лівого, і завершує горизонтальним рухом до нижнього правого кута. Цей патерн частіше зустрічається на сторінках з меншою кількістю тексту, але з чітко вираженими візуальними елементами та Call to Action (наприклад, лендінги, сторінки продуктів, рекламні банери). Ключові елементи (логотип, головний заголовок, зображення, Call to Action) розміщуються у кутах «Z».

Розуміння цих патернів дозволяє дизайнерам розміщувати найважливішу інформацію та елементи дії (Call to Action) у ключових

точках, де погляд користувача буде їх шукати, оптимізуючи сприйняття та конверсію. Для розробника це означає, що під час верстання необхідно зберігати логіку розташування елементів, щоб не порушити природний шлях сканування сторінки, навіть за адаптації під різні розміри екрана. Це також допомагає зрозуміти, чому певні елементи мають бути розташовані саме так, а не інакше.

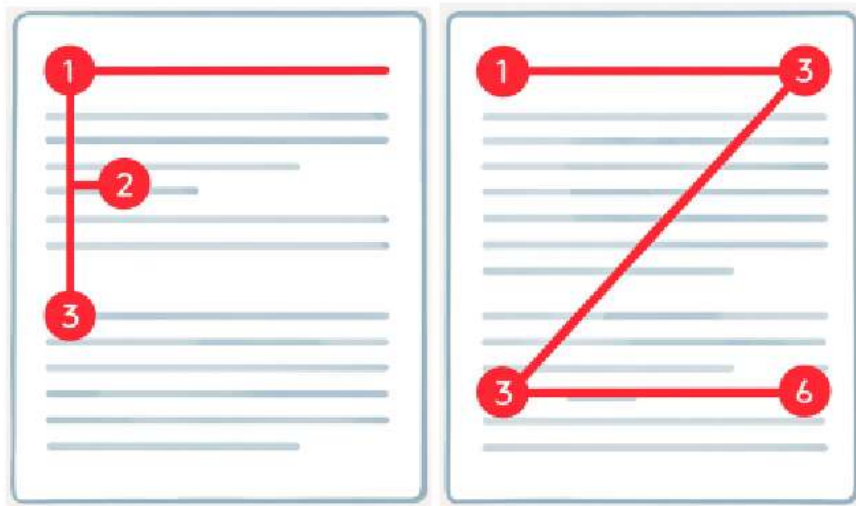


Рисунок 3.2 – Схематичне зображення F-патерну та Z-патерну читання

Негативний простір (White Space) – це порожній простір між елементами дизайну: між текстом, зображеннями, кнопками, блоками, абзацами, рядками. Всупереч назві, він не є «порожнім» або «марним». Негативний простір є одним з найпотужніших інструментів композиції, який покращує читабельність, фокусує увагу на ключових елементах, створює відчуття простору, елегантності та професіоналізму. Достатня кількість негативного простору дозволяє «дихати» контенту, зменшує візуальний шум, знижує когнітивне навантаження на користувача та покращує загальне сприйняття інформації. Він допомагає групувати пов'язані елементи (принцип близькості) та розділяти непов'язані, створюючи чітку візуальну структуру. Наприклад, більший негативний простір навколо заголовка робить його більш помітним та виділеним, а достатній простір між абзацами покращує читабельність тексту. Негативний простір може бути **активним** (навмисно створений для фокусування уваги або створення динаміки) та **пасивним** (просто простір, що виникає між елементами). Для розробника це означає точне відтворення відступів (margin, padding), визначених дизайнером, оскільки

саме вони формують негативний простір. Ігнорування негативного простору або його неправильне використання може призвести до «задушеного», перевантаженого дизайну, який важко сприймати, що негативно впливає на юзабіліті та естетику. Розуміння важливості негативного простору дозволяє розробнику не просто копіювати значення, а розуміти їхню функцію та вплив на кінцевий продукт.

Всі ці принципи композиції – баланс, контраст, візуальна ієрархія, сітки, патерни читання та негативний простір – працюють разом, створюючи цілісний, ефективний та привабливий вебдизайн. Розуміння їхньої взаємодії дозволяє інженерам не просто перетворювати макети на код, а бути активними учасниками процесу створення продукту, який є не лише функціональним, але й візуально привабливим, інтуїтивно зрозумілим та зручним для користувача. Це сприяє кращій співпраці з дизайнерами та створенню високоякісних цифрових продуктів.

Завдання для демонстрації

1. **Аналіз балансу.** Відкрийте два сайти: один з симетричним дизайном (наприклад, класичний новинний портал) та один з асиметричним (наприклад, креативне портфоліо). Обговоріть, як досягається баланс у кожному випадку.

2. **Аналіз контрасту та ієрархії.** Відкрийте будь-який сайт з добре продуманим дизайном. Визначте, як використовується колірний, розмірний та типографічний контраст для створення візуальної ієрархії. Які елементи привертають увагу насамперед?

3. **Аналіз сіток та негативного простору.** Використовуючи інструменти розробника в браузері (F12), увімкніть відображення сітки (якщо сайт її використовує) та проаналізуйте відступи (margin, padding) між елементами. Обговоріть, як негативний простір впливає на читабельність та сприйняття контенту.

Питання для самоконтролю

1. Поясніть різницю між симетричним та асиметричним балансом у вебдизайні. Наведіть приклади, де кожен з них буде доречним.

2. Які типи контрасту ви знаєте і як вони використовуються для створення візуальної ієрархії?

3. Що таке візуальна ієрархія і чому вона є критично важливою для юзабіліті вебсайту?

4. Як сітки допомагають у створенні гармонійної композиції та адаптивної верстки?

5. Опишіть F-патерн та Z-патерн читання. Як знання цих патернів впливає на розташування ключових елементів на сторінці?

6. Чому негативний простір є важливим інструментом композиції, а не просто «порожнім місцем»?

Рекомендоване практичне завдання

1. **Аналіз композиції.** Виберіть 3 різні вебсайти (наприклад, інтернет-магазин, блог, корпоративний сайт). Для кожного сайту проаналізуйте:

- тип балансу (симетричний/асиметричний);
- використання контрасту (колірний, розмірний, типографічний);
- візуальну ієрархію (які елементи є найпомітнішими, як направляється погляд);
- наявність та використання сіток (спробуйте увімкнути інструменти розробника в браузері та знайти елементи сітки);
- використання негативного простору;
- оформіть свої спостереження у вигляді короткого звіту.

2. **Практика у Figma (або на папері).** Створіть макет простої сторінки (наприклад, картки товару або блог-посту), застосовуючи принципи композиції:

- використовуйте асиметричний баланс;
- створіть чітку візуальну ієрархію за допомогою розміру та контрасту;
 - застосуйте сітку (наприклад, 12-колонкову) для вирівнювання елементів;
 - активно використовуйте негативний простір для покращення читабельності та фокусування уваги;
- поясніть, як ви застосували кожен з принципів.

3.2 Типографіка у Web-дизайні

У світі вебдизайну, де візуальна комунікація є ключовою, **типографіка** відіграє одну з найважливіших ролей. Типографіка – це мистецтво та техніка організації шрифту для того, щоб зробити письмову мову легкою для читання, привабливою та ефективною для передачі повідомлення. Це не просто вибір «красивого» шрифту, а глибоке розуміння того, як різні шрифти, їхні розміри, інтервали та розташування впливають на сприйняття інформації, емоційний відгук користувача та загальну естетику сайту. Для інженера програмного забезпечення, який працює з версткою, розуміння основ типографіки є критично важливим. Це дозволяє не лише точно відтворювати дизайнерські макети, але й розуміти, чому саме такі шрифти та їхні властивості були вибрані, як вони впливають на читабельність та доступність, і як правильно реалізувати їх за допомогою CSS. Недооцінення типографіки може призвести до того, що навіть найцінніший контент залишиться непоміченим або буде важким для сприйняття, що негативно позначиться на юзабіліті та конверсії вебресурсу.

Перш ніж заглибитися в деталі, важливо розрізняти терміни **шрифт (font)** та **гарнітура (typeface)**.

Гарнітура – це сімейство шрифтів, що мають спільні дизайнерські характеристики та єдиний візуальний стиль (наприклад, Arial, Times New Roman, Open Sans, Roboto). Це, по суті, дизайн літер.

Шрифт – це конкретна реалізація гарнітури з певним розміром, накресленням та стилем (наприклад, Arial Regular 16 pt, Bold Italic 12 pt, Times New Roman 14 pt). Тобто, гарнітура – це «сім'я», а шрифт – це «член сім'ї» з певними атрибутами. У повсякденній мові ці терміни часто використовуються як синоніми, але в професійному контексті важливо розуміти цю різницю. Вибір правильної гарнітури та її шрифтів може кардинально змінити сприйняття контенту, вплинути на настрій сайту, його брендинг та ефективність комунікації. Наприклад, елегантний Serif-шрифт може надати сайту відчуття класики та авторитетності, тоді як чистий Sans-serif створить сучасний та мінімалістичний вигляд.

Шрифти можна класифікувати за кількома основними категоріями, кожна з яких має свої особливості, історичне походження та оптимальні сценарії застосування:

- **Serif (із засічками).** Це шрифти, які мають невеликі декоративні елементи (засічки) на кінцях штрихів літер. Історично вони походять від римських написів на камені. Засічки візуально з'єднують літери в слово, створюючи горизонтальну лінію, що, як вважається, допомагає оку легше слідувати по тексту. Вони асоціюються з традицією, елегантністю, авторитетом, класикою та надійністю. Завдяки засічкам, які візуально з'єднують літери, Serif-шрифти традиційно вважаються більш читабельними для великих обсягів друкованого тексту (книги, газети). Приклади: Times New Roman, Georgia, Lora, Playfair Display. У вебі їх часто використовують для заголовків, цитат або для сайтів, що прагнуть створити класичний, «книжковий» або вишуканий вигляд (наприклад, сайти музеїв, юридичних фірм, модних брендів).
- **Sans-serif (без засічок).** Ці шрифти не мають засічок, що надає їм більш сучасного, чистого, мінімалістичного та прямолінійного вигляду. Вони з'явилися пізніше, у 19–20 століттях, і стали популярними завдяки своїй простоті та функціональності. Sans-serif шрифти чудово підходять для відображення на екранах (монітори, смартфони, планшети), оскільки відсутність дрібних деталей покращує розбірливість за низької роздільної здатності та на малих розмірах. Приклади: Arial, Helvetica, Open Sans, Roboto, Montserrat. Sans-serif шрифти є стандартом для основного тексту на більшості веб-сайтів, а також для інтерфейсів користувача, де важлива чіткість та швидке сканування інформації.

- **Display (декоративні).** Це шрифти, розроблені спеціально для привернення уваги та використання у великих розмірах. Вони часто мають унікальні, виразні, іноді химерні форми, але зазвичай погано читаються у малих розмірах або великих обсягах тексту. Використовуються для великих заголовків, логотипів, банерів, рекламних матеріалів, де потрібно створити особливий настрій, підкреслити бренд або передати певну емоцію. Приклади: Lobster, Pacifico, Impact. Їх слід використовувати дуже обережно та дозовано.
- **Script (рукописні).** Імітують рукописний текст або каліграфію. Вони додають елегантності, особистого дотику, витонченості та індивідуальності. Однак, як і Display-шрифти, їхня читабельність може бути низькою у великих обсягах тексту або малих розмірах. Використовуються для акцентів, підписів, запрошень, вітальних листівок, де потрібен особливий, персоналізований стиль. Приклади: Great Vibes, Dancing Script, Allura.
- **Monospace (моноширинні).** Це шрифти, у яких кожна літера та символ займає однакову горизонтальну ширину, незалежно від її форми (наприклад, «i» займає стільки ж місця, скільки «w»). Це робить їх ідеальними для відображення коду, таблиць, або для створення ефекту друкарської машинки. Вони забезпечують чітке вирівнювання стовпців, що є критично важливим для читання коду. Приклади: Courier New, Fira Code, Consolas, Source Code Pro. Для розробників це особливо важлива категорія, оскільки саме такі шрифти використовуються в IDE та текстових редакторах, забезпечуючи легкість читання та відладки налагодження коду.

Категорії шрифтів	
Serif	мають невеликі штрихи (нахилки) на кінцях букв. Попри це, це найпоширеніший вибір для друкування ЗМІ та інших тривалих текстів.
Sans-serif	не мають штрихів на кінцях. Як правило, використовуються для веб-сайтів.
Display Decorative!	екстравагантні шрифти, розроблені для привертання уваги. Можуть робити заголовки та місця для акцентів.
<i>Script Cursive</i>	виглядають як рукописні шрифти. Як правило, використовуються для урочистих подій, таких як запрошення.
Monospace	популярні для програмного коду завдяки рівномірному інтервалу. Часто їх можна знайти у табуляційній ідентичності відповідні електронних пристроїв та техніки.

Рисунок 3.3 – Приклади різних категорій шрифтів

Вибір одного шрифту рідко буває достатнім для повноцінного вебсайту. Часто дизайнери використовують **шрифтові пари** – комбінації двох або більше гарнітур, які гармонійно доповнюють одна одну. Мета шрифтової пари – створити візуальний інтерес, покращити візуальну ієрархію та забезпечити функціональне розмежування різних типів контенту (наприклад, заголовки, основний текст, цитати), не створюючи візуального шуму або дисонансу. Існують загальні правила та рекомендації для вибору шрифтових пар:

- **Контраст, але не конфлікт.** Шрифти мають бути достатньо різними, щоб їх можна було легко розрізнити та щоб вони виконували різні функції, але не настільки, щоб вони «сперечалися» між собою або створювали візуальний хаос. Наприклад, добре працює поєднання контрастних категорій, таких як Serif для заголовків (для надання авторитетності) та Sans-serif для основного тексту (для кращої читабельності на екрані).
- **Гармонія.** Незважаючи на контраст, необхідно, щоб шрифти мали спільні риси або налаштування, щоб створювати єдине ціле та підтримувати загальний стиль бренду. Це може бути схожа висота літер, спільні пропорції або схожий історичний період походження.
- **Функціональність.** Один шрифт зазвичай використовується для заголовків (може бути більш виразним, декоративним, привертаючим увагу), інший – для основного тексту (має бути максимально читабельним та розбірливим). Іноді додається третій шрифт для акцентів або спеціальних елементів.
- **Обмеження кількості.** Зазвичай рекомендується використовувати не більше 2-3 різних гарнітур на одному сайті, щоб уникнути візуального перевантаження та зберегти консистентність.

Для інженера, який працює з CSS, це означає правильне використання властивостей ``font-family`` (з вказанням кількох шрифтів як fallback), ``font-size``, ``font-weight`` (для різних накреслень) та ``line-height`` для реалізації цих пар. Важливо також враховувати, що не всі шрифти підтримують кирилицю, і завжди мати запасні (fallback) шрифти у ``font-family`` на випадок, якщо основний шрифт не завантажиться або не підтримуватиме потрібні символи. Використання ``@font-face`` для підключення кастомних шрифтів також потребує уваги до продуктивності та ліцензування.

Читабельність (Readability) та **розбірливість (Legibility)** – це два взаємопов'язані, але різні аспекти типографіки, які є критично важливими для ефективної передачі інформації. **Розбірливість** стосується того, наскільки легко розрізнити окремі літери та символи. Вона залежить від дизайну самої гарнітури (наприклад, чи не надто схожі «l» та «l», «0» та

«О», чи достатньо чіткі контури літер). **Читабельність** стосується того, наскільки легко та комфортно читати великі обсяги тексту. Вона залежить не лише від гарнітури, а й від її розміру, інтерліньяжу (відстані між рядками), довжини рядка, контрасту тексту з фоном, кернінгу/трекінгу та загального розташування тексту на сторінці.

Для забезпечення оптимальної читабельності у вебi потрібно враховувати такі параметри:

- **Розмір шрифту.** Для основного тексту на десктопі рекомендований розмір 16–18 px (або 1 em/1.125 em). На мобільних пристроях може бути трохи меншим, але не менше 14 px, щоб уникнути напруги очей. Важливо використовувати відносні одиниці (rem, em, vw) для адаптивності та масштабованості тексту.
- **Інтерліньяж (Line-height).** Відстань між базовими лініями сусідніх рядків тексту. Зазвичай оптимальне значення становить 1.4–1.6 від розміру шрифту. Занадто малий інтерліньяж робить текст «стиснутим» і важким для читання, занадто великий – розриває зв'язок між рядками, ускладнюючи перехід погляду. Правильний інтерліньяж допомагає оку легко переходити від кінця одного рядка до початку наступного.
- **Довжина рядка.** Оптимальна довжина рядка для читабельності становить 45–75 символів (включно пробіли). Занадто короткі рядки змушують очі часто перескакувати, що втомлює. Занадто довгі – ускладнюють пошук початку наступного рядка та втрату місця. Для адаптивного дизайну це часто досягається за допомогою обмеження ширини текстового блока.
- **Контраст.** Достатній контраст між текстом та фоном є обов'язковим не лише для естетики, а й для доступності. Потрібно уникати низькоконтрастних поєднань (наприклад, світло-сірий текст на білому фоні). Використання інструментів перевірки контрасту (наприклад, WebAIM Contrast Checker) допомагає відповідати стандартам доступності (WCAG 2.1 AA або AAA).
- **Кернінг (Kerning).** Регулювання відстані між окремими парами літер для покращення візуального вигляду та усунення небажаних проміжків (наприклад, між «AV», «Wa», «To»). Зазвичай автоматично налаштовується браузером та шрифтами, але може бути скоригований вручну для великих заголовків або логотипів за допомогою CSS-властивості `letter-spacing`.
- **Трекінг (Tracking).** Регулювання відстані між усіма літерами у слові або блоці тексту. Використовується для ущільнення або розрідження тексту. Зазвичай використовується для заголовків

або акцентів, рідше для основного тексту, оскільки може погіршити читабельність.

1. Вузький блок, малий інтерліньяж

Цей текст має вузький блок і малий міжрядковий інтервал. Його важко читати, оскільки рядки тісно розташовані, а очі часто змушені повертатися на новий рядок.

2. Широкий блок, малий інтерліньяж

Цей текст має широку довжину рядка, але малий інтерліньяж. Через велику ширину очам важко знаходити початок наступного рядка, що знижує комфорт читання.

3. Вузький блок, великий інтерліньяж

Тут текст має вузький блок і великий міжрядковий інтервал. Це покращує читабельність, але надто вузький блок може створювати надмірну фрагментацію.

4. Широкий блок, великий інтерліньяж

Оптимальний варіант: широка довжина рядка з достатнім інтерліньяжем. Такий текст легко читати, оскільки очі мають достатньо простору між рядками, а довжина рядка не перевантажує зір.

Рисунок 3.4 – Приклад, що демонструє вплив інтерліньяжу та довжини рядка на читабельність

Для інженера, який працює з CSS, всі ці параметри реалізуються за допомогою відповідних властивостей: ``font-family``, ``font-size``, ``line-height``, ``letter-spacing`` (для трекінгу та кернінгу), ``word-spacing``, ``text-align``, ``color``, ``background-color``. Важливо не лише знати ці властивості, а й розуміти, як вони впливають на кінцевий досвід користувача. Дотримання принципів типографіки дозволяє створювати вебінтерфейси, які не лише виглядають професійно та естетично, але й є максимально зручними, доступними та ефективними для широкого кола користувачів. Це також

стосується оптимізації завантаження шрифтів (використання ``font-display``, ``preload``, ``woff2`` форматів) для забезпечення швидкої появи тексту на сторінці, що є важливим аспектом продуктивності та UX.

Завдання для демонстрації

1. **Аналіз класифікації шрифтів.** Відкрийте Google Fonts. Знайдіть приклади шрифтів для кожної з категорій (Serif, Sans-serif, Display, Script, Monospace). Обговоріть, де кожен з них був би доречним.

2. **Аналіз шрифтових пар.** Відкрийте сайт Fontjoy (<https://fontjoy.com/>) або Google Fonts Pairings (https://fonts.google.com/knowledge/pairing_fonts). Продемонструйте, як підбираються гармонійні шрифтові пари. Обговоріть, чому певні комбінації працюють, а інші – ні.

3. **Експерименти з читабельністю.** Використовуючи інструменти розробника в браузері (F12), змініть властивості ``font-size``, ``line-height``, ``letter-spacing`` та ``width`` для основного тексту на будь-якому сайті. Продемонструйте, як ці зміни впливають на читабельність тексту. Знайдіть оптимальні та найгірші варіанти.

Питання для самоконтролю

1. Поясніть різницю між термінами «шрифт» та «гарнітура».
2. Назвіть основні категорії шрифтів та наведіть приклади їхнього використання у вебдизайні.
3. Які принципи потрібно враховувати під час вибору шрифтових пар?
4. У чому полягає різниця між читабельністю та розбірливістю? Які параметри впливають на читабельність тексту?
5. Як інтерліньяж та довжина рядка впливають на комфорт читання великих обсягів тексту?
6. Чому моноширинні шрифти є кращим вибором для відображення коду?

Рекомендоване практичне завдання

1. **Аналіз типографіки на сайтах.** Виберіть 3 різні вебсайти (наприклад, новинний портал, інтернет-магазин, особистий блог). Для кожного сайту проаналізуйте:

1. Які гарнітури використовуються для заголовків та основного тексту? До яких категорій вони належать?
2. Чи є шрифтова пара гармонійною та функціональною?
3. Оцініть читабельність основного тексту (розмір шрифту, інтерліньяж, довжина рядка, контраст). Чи відповідає вона рекомендаціям?
4. Чи є проблеми з розбірливістю окремих символів?
5. Оформіть свої спостереження у вигляді короткого звіту.

2. Експерименти з CSS:

1. Створіть простий HTML-файл з кількома абзацами тексту та заголовками. Застосуйте до нього різні CSS-властивості, експериментуючи з:

 `font-family` (спробуйте різні шрифтові пари з Google Fonts).

 `font-size` (для заголовків та основного тексту).

 `line-height`.

 `letter-spacing`.

 `color` та `background-color` (для забезпечення контрасту).

Зробіть скріншоти різних варіантів та опишіть, який варіант, на вашу думку, є найбільш читабельним та чому.

2. У пристроях або для користувачів з обмеженими можливостями. Тому важливо збалансувати ефективність і вражаючий вигляд з доступністю.

3. Для створення ефекту паралаксу за допомогою CSS можна використовувати різні підходи. Ось декілька основних CSS селекторів та властивостей, які можна використовувати:

а) `background-attachment: fixed;` Цей селектор дозволяє закріпити фонове зображення на екрані, тобто воно залишатиметься на місці під час прокрутки сторінки.

```
.parallax-section {  
  background-image: url('background.jpg');  
  background-attachment: fixed;  
  background-size: cover;  
}
```

б) `transform: translate();` Цей селектор дозволяє зміщувати елементи на екрані. Його можна використовувати для створення ефекту руху шарів під час прокрутки.

```
.parallax-layer {  
  transform: translate(0, -50%);  
}
```

в) `transform: translate();` Цей селектор дозволяє зміщувати елементи на екрані. Його можна використовувати для створення ефекту руху шарів під час прокрутки.

```
@keyframes parallax {  
  from {  
    transform: translate(0, -100%);  
  }  
  to {  
    transform: translate(0, 100%);  
  }  
}  
.parallax-layer {
```

```
    animation: parallax 10s infinite alternate;
}
```

г) perspective: Ця властивість визначає глибину перспективи елемента. Використовуйте її для створення тривимірного візуального ефекту.

```
.parallax-container {
    perspective: 100px;
}
```

д) perspective: Ця властивість визначає глибину перспективи елемента. Використовуйте її для створення тривимірного візуального ефекту.

```
.parallax-container {
    perspective-origin: 50% 50%;
}
```

е) transform-style: Ця властивість визначає, як дочірні елементи взаємодіють з перспективою відносно батьківського елемента.

```
.parallax-layer {
    transform-style: preserve-3d;
}
```

ж) opacity: Ця властивість визначає прозорість елемента. Використовуйте її для створення ефекту прозорості шарів.

```
.parallax-layer {
    opacity: 0.8;
}
```

и) z-index визначає порядок накладання елементів на площині з від'ємним значенням, які знаходяться ближче до глядача, і елементів зі значенням більше за нуль, які знаходяться далі від глядача.

```
.parallax-layer-1 {
    z-index: 1;
}
.parallax-layer-2 {
    z-index: 2;
}
```

3.3 Теорія кольору

Колір є одним з найпотужніших інструментів у арсеналі вебдизайнера, здатним миттєво привернути увагу, створити певне естетичне враження та викликати глибокий емоційний відгук. Він не лише формує візуальну привабливість інтерфейсу, але й має значний психологічний вплив на користувача, викликаючи певні асоціації, впливаючи на настрій та навіть на поведінку.

Теорія кольору – це систематизований набір принципів та рекомендацій щодо використання кольорів, які допомагають дизайнерам створювати гармонійні, ефективні, зрозумілі та функціональні візуальні

рішення. Для інженера програмного забезпечення розуміння теорії кольору є надзвичайно важливим. Це дозволяє не просто механічно копіювати колірні коди з макета, а розуміти логіку їхнього вибору, вміти адаптувати колірні схеми для різних станів елементів (наприклад, активна кнопка, наведена кнопка, неактивна кнопка, стан фокусу), забезпечувати доступність (достатній контраст для людей з порушеннями зору) та ефективно працювати з CSS-властивостями, що відповідають за колір. Глибоке розуміння кольору дозволяє розробнику бути більш проактивним у співпраці з дизайнерами, пропонувати технічно обґрунтовані рішення та створювати інтерфейси, які є не лише красивими, але й функціональними та інклюзивними.

Психологія кольору у вебдизайні: емоції та асоціації

Кожен колір має свою унікальну психологічну асоціацію, яка може відрізнятися залежно від культурного контексту, особистого досвіду та навіть гендерних відмінностей. Однак, існують і загальні закономірності, які дизайнери використовують для передачі певного настрою, цінностей бренду й впливу на емоції та поведінку користувача. Розуміння цих асоціацій дозволяє створювати більш цілеспрямований та ефективний дизайн.

Червоний. Асоціюється з енергією, пристрастю, любов'ю, небезпекою, терміновістю, агресією. Це колір, що привертає увагу, стимулює дію та підвищує пульс. У вебдизайні часто використовується для кнопок «Купити», «Зареєструватися», попереджень, повідомлень про помилки, а також для акцентів, що потребують негайної уваги. Надмірне використання може викликати тривогу або агресію.

Помаранчевий. Символізує ентузіазм, творчість, тепло, дружність, доступність, оптимізм. Це більш м'який та грайливий варіант червоного. Часто асоціюється з молодіжними брендами, інноваціями, продуктами для дітей. Може використовуватися для кнопок Call to Action, банерів, що закликають до дії, або для створення відчуття тепла та гостинності.

Жовтий. Колір оптимізму, радості, щастя, сонця, інтелекту, а також обережності. Він є дуже помітним і може використовуватися для акцентів, привернення уваги до важливих елементів. Однак, у великих кількостях або за неправильного поєднання може бути дратівливим або асоціюватися з дешевиною. Часто використовується для виділення знижок, акцій, або для брендів, що прагнуть передати відчуття енергії та позитиву.

Зелений. Асоціюється з природою, свіжістю, зростанням, здоров'ям, гармонією, спокоєм, безпекою, грошима. Це колір балансу та оновлення. Часто використовується для екологічних брендів, фінансових послуг, кнопок «Успіх», «Підтвердити», «Оплачено», а також для фонів, що створюють відчуття спокою та довіри.

Синій. Символізує довіру, стабільність, надійність, спокій, професіоналізм, технології, інтелект. Це найпопулярніший колір у

корпоративному дизайні, соціальних мережах, фінансових установах, оскільки він викликає відчуття безпеки та авторитету. Може використовуватися як основний колір для фонів, заголовків, посилань. Надмірне використання може створити відчуття холоду або відстороненості.

Фіолетовий. Асоціюється з розкішшю, креативністю, мудрістю, духовністю, таємницею, королівською владою. Це колір, що поєднує стабільність синього та енергію червоного. Часто використовується для брендів, орієнтованих на жінок, для продуктів преміум-класу, а також у креативних індустріях.

Рожевий: Символізує ніжність, романтику, жіночність, грайливість, співчуття. Популярний у б'юті-індустрії, дитячих товарах, а також для брендів, що прагнуть передати відчуття м'якості та турботи.

Коричневий. Асоціюється з землею, натуральністю, стабільністю, комфортом, простотою, надійністю. Використовується для органічних продуктів, кав'ярень, меблевих магазинів, створюючи відчуття затишку та природності.

Чорний. Символізує елегантність, владу, вишуканість, таємницю, формальність, розкіш. Часто використовується для люксових брендів, мінімалістичного дизайну, а також як основний колір для тексту. Може створювати відчуття драматизму або серйозності.

Білий. Асоціюється з чистотою, простотою, невинністю, мінімалізмом, простором, свіжістю. Є основою для більшості вебсайтів, створює відчуття легкості, відкритості та дозволяє іншим кольорам виділятися. Важливий для створення негативного простору.

Сірий. Символізує нейтральність, баланс, професіоналізм, стриманість, стабільність. Часто використовується для фонів, допоміжних елементів, тексту, що не є основним, а також для створення тіней та градієнтів. Може бути елегантним, але надмірне використання може зробити дизайн нудним або депресивним.

Важливо пам'ятати, що психологія кольору не є універсальною і може змінюватися залежно від культурного контексту, особистого досвіду користувача та навіть відтінку самого кольору. Однак, ці загальні асоціації є хорошою відправною точкою для вибору колірної палітри, яка відповідатиме цілям, бренду та цільовій аудиторії проєкту. Ефективний дизайнер завжди враховує ці нюанси, щоб колір працював на користь продукту.

Колірне коло Іттена та гармонійні колірні схеми

Колірне коло Іттена – це фундаментальний інструмент у теорії кольору, розроблений художником Йоганнесом Іттенем, який візуалізує взаємозв'язок між кольорами та допомагає створювати гармонійні поєднання. Воно складається з 12 кольорів, організованих таким чином:

Первинні кольори (Primary Colors). Червоний, жовтий, синій. Це кольори, які не можна отримати шляхом змішування інших кольорів. Вони є основою для всіх інших кольорів.

Вторинні кольори (Secondary Colors). Помаранчевий (червоний + жовтий), зелений (жовтий + синій), фіолетовий (синій + червоний). Отримуються шляхом змішування двох первинних кольорів.

Третинні кольори (Tertiary Colors). Отримуються шляхом змішування первинного та вторинного кольорів, що розташовані поруч на колірному колі (наприклад, червоно-помаранчевий, жовто-зелений, синьо-фіолетовий). Ці 12 кольорів формують повне колірне коло.

Колірне коло є основою для створення гармонійних колірних схем, які є фундаментом для колірної палітри вебсайту. Гармонійні схеми забезпечують візуальну привабливість та баланс. ***Основні колірні схеми:***

Монохроматична (Monochromatic). Використання різних відтінків, тонів та насиченості одного кольору. Створює дуже спокійний, елегантний, витончений та гармонійний вигляд. Легко досягти балансу, але може бути дещо нудною без додаткових акцентів. Чудово підходить для мінімалістичного дизайну або для фонів.

Аналогова (Analogous). Використання 2–4 кольорів, що розташовані поруч на колірному колі (наприклад, синій, синьо-зелений, зелений). Створює спокійний, приємний для ока та гармонійний дизайн, оскільки кольори природно поєднуються. Часто один колір домінує, інші використовуються для підтримки. Добре підходить для створення природних, органічних палітр.

Комплементарна (Complementary). Використання двох кольорів, що розташовані один навпроти одного на колірному колі (наприклад, червоний та зелений, синій та помаранчевий, жовтий та фіолетовий). Створює сильний контраст, динаміку та привертає увагу. Часто один колір використовується як основний, інший – для акцентів або Call to Action. Потрібно обережно використовувати, щоб уникнути візуального конфлікту.

Тріадна (Triadic). Використання трьох кольорів, рівновіддалених один від одного на колірному колі (наприклад, червоний, жовтий, синій). Створює яскравий, збалансований та динамічний вигляд. Один колір зазвичай домінує, інші використовуються для акцентів. Це одна з найпопулярніших схем, оскільки вона забезпечує достатній контраст, зберігаючи гармонію.

Тетрадна (Tetradic/Double Complementary). Використання чотирьох кольорів, що утворюють прямокутник на колірному колі (дві пари комплементарних кольорів). Це найскладніша схема для досягнення гармонії, оскільки потребує ретельного балансування чотирьох кольорів, але вона може бути дуже багатою, динамічною та виразною. Потрібно ретельно вибирати домінувальний колір та використовувати інші як акценти.

Спліт-комплементарна (Split-Complementary). Використання основного кольору та двох кольорів, що є сусідніми до його комплементарного кольору. Це забезпечує сильний візуальний контраст, але є менш напруженим, ніж чиста комплементарна схема, оскільки використовує більше відтінків. Це хороший компроміс між динамікою та гармонією.

Колірне Коло Іттена (12 Кольорів)



Рисунок 3.5 – Колірне коло Іттена з позначенням первинних, вторинних та третинних кольорів

Колірні моделі у веброзробці: RGB, HSL та інші

У веброзробці ми працюємо з цифровими колірними моделями, які дозволяють комп'ютерам інтерпретувати та відображати кольори. Дві найпоширеніші, які використовуються в CSS, це RGB та HSL, але варто згадати й інші:

RGB (Red, Green, Blue). Це адитивна колірна модель, яка використовується для відображення кольорів на екранах (монітори, телевізори, смартфони, планшети). Колір створюється шляхом додавання (змішування) червоного, зеленого та синього світла. Кожен компонент має значення від 0 до 255. Наприклад, `rgb(255, 0, 0)` – це чистий червоний, `rgb(0, 0, 0)` – чорний (відсутність світла), `rgb(255, 255, 255)` – білий (максимум світла). У CSS часто використовується шістнадцятковий формат (hex-код), наприклад, `#FF0000` для червоного, `#000000` для чорного, `#FFFFFF` для білого. Також існує `rgba()`, де `'a'` означає альфа-канал (прозорість) від 0 (повністю прозорий) до 1 (повністю непрозорий). RGB є найбільш поширеним для визначення точних кольорів у дизайні та розробці.

HSL (Hue, Saturation, Lightness). Це більш інтуїтивна колірна модель, яка краще відповідає людському сприйняттю кольору, оскільки вона описує колір через його відтінок, насиченість та яскравість. Вона складається з трьох компонентів:

- ✓ **Hue (Відтінок).** Колір на колірному колі, вимірюється в градусах від 0 до 360 (0/360 = червоний, 60 = жовтий, 120 = зелений, 180 = блакитний, 240 = синій, 300 = фіолетовий).
- ✓ **Saturation (Насиченість).** Інтенсивність кольору вимірюється у відсотках від 0 % (сірий) до 100 % (чистий, найбільш насичений колір).
- ✓ **Lightness (Яскравість/Світлість).** Світлість кольору вимірюється у відсотках від 0 % (чорний) до 100 % (білий). 50 % – це чистий колір без додавання чорного чи білого.

Наприклад, `hsl(0, 100 %, 50 %)` – це чистий червоний. `hsla()` також включає альфа-канал. HSL є дуже зручним для створення монохроматичних та аналогових палітр, оскільки легко змінювати насиченість та яскравість, зберігаючи відтінок, що дозволяє швидко генерувати варіації кольору для різних станів елементів (наприклад, hover-стан кнопки).

CMYK (Cyan, Magenta, Yellow, Key/Black). Це субтрактивна колірна модель, яка використовується в поліграфії для друку. Вона не використовується безпосередньо у вебдизайні, але дизайнери, які працюють як для вебу, так і для друку, мають знати про її існування та відмінності від RGB.

HEX (шістнадцятковий код). Це скорочений спосіб подання RGB-кольорів, дуже поширений у веброзробці. Формат `#RRGGBB`, де RR, GG, BB – шістнадцяткові значення для червоного, зеленого та синього компонентів, відповідно (від 00 до FF). Наприклад, `#FF0000` – червоний, `#000000` – чорний, `#FFFFFF` – білий. Існує також скорочений формат `#RGB` (наприклад, `#F00` для червоного), якщо кожна пара символів однакова.

Для інженера важливо знати всі ці моделі, оскільки вони використовуються в CSS. RGB/Hex є більш поширеними для точного визначення кольору, тоді як HSL може бути зручнішим для маніпуляцій з кольором та створення динамічних колірних ефектів.

Створення колірної палітри: гармонія та функціональність

Колірна палітра – це ретельно підібраний набір кольорів, які використовуються в дизайні вебсайту або додатка. Добре продумана палітра створює єдиний візуальний стиль, підтримує ідентичність бренду, покращує юзабіліті та забезпечує ефективну візуальну комунікацію. Зазвичай палітра містить кілька категорій кольорів, кожна з яких виконує свою функцію:

Основні кольори (Primary Colors). 1–2 кольори, що домінують у дизайні та асоціюються з брендом. Вони використовуються для великих блоків, заголовків, ключових елементів інтерфейсу. Це кольори, які користувач запам'ятовує як «кольори бренду».

Додаткові кольори (Secondary Colors). Кольори, що доповнюють основні та використовуються для фонів, допоміжних елементів, ілюстрацій, графіків. Вони розширюють палітру, не перевантажуючи її, і створюють візуальне різноманіття.

Акцентні кольори (Accent Colors). Яскраві, контрастні кольори, що використовуються для кнопок дії (Call to Action), важливих посилянь, сповіщень, іконок. Вони мають привертати увагу та направляти користувача до ключових інтерактивних елементів. Акцентний колір має виділятися на тлі основних та додаткових кольорів.

Нейтральні кольори (Neutral Colors). Відтінки сірого, білого, чорного, бежевого. Використовуються для тексту, фонів, роздільників, тіней, щоб не відволікати від основного контенту та забезпечити читабельність. Вони створюють «негативний простір» та допомагають іншим кольорам виділятися.

Кольори зворотного зв'язку (Feedback Colors). Спеціальні кольори для повідомлень про успіх (зелений), помилку (червоний), попередження (жовтий/помаранчевий) або інформацію (синій). Вони є універсально зрозумілими для користувачів.

Під час створення палітри важливо враховувати:

Бренд та його ідентичність. Кольори мають відповідати цінностям, місії та візуальному стилю бренду.

Цільова аудиторія. Психологія кольору та культурні асоціації, які можуть впливати на сприйняття.

Доступність. Критично важливий аспект. Необхідно забезпечити достатній контраст між текстом та фоном, а також між інтерактивними елементами, щоб сайт був зручним для людей з порушеннями зору. Використання інструментів перевірки контрасту є обов'язковим.

Гармонія. Використання колірних схем з колірного кола для створення візуально приємних поєднань.

Функціональність. Кожен колір у палітрі потрібно, щоб мав своє призначення та роль.

Існують численні онлайн-інструменти (наприклад, Adobe Color, Coolers.co, Paletton, Color Hunt), які допомагають дизайнерам та розробникам створювати, тестувати та експортувати колірні палітри. Для розробника це означає організацію колірних змінних у CSS (CSS Custom Properties або SASS/LESS змінні) для легкого управління та зміни кольорів у всьому проєкті, що забезпечує консистентність та спрощує рефакторинг. Розуміння того, як кольори взаємодіють і як вони впливають на користувача, дозволяє інженерам не лише точно реалізовувати дизайнерські рішення, але й брати активну участь у процесі створення візуально привабливих, функціональних та доступних вебінтерфейсів. Це допомагає уникнути поширених помилок, таких як низький контраст, що робить текст нечитабельним, або використання занадто великої кількості

кольорів, що створює візуальний хаос. Правильне використання кольору може значно покращити досвід користувача та ефективність вебсайту, сприяючи його успіху.

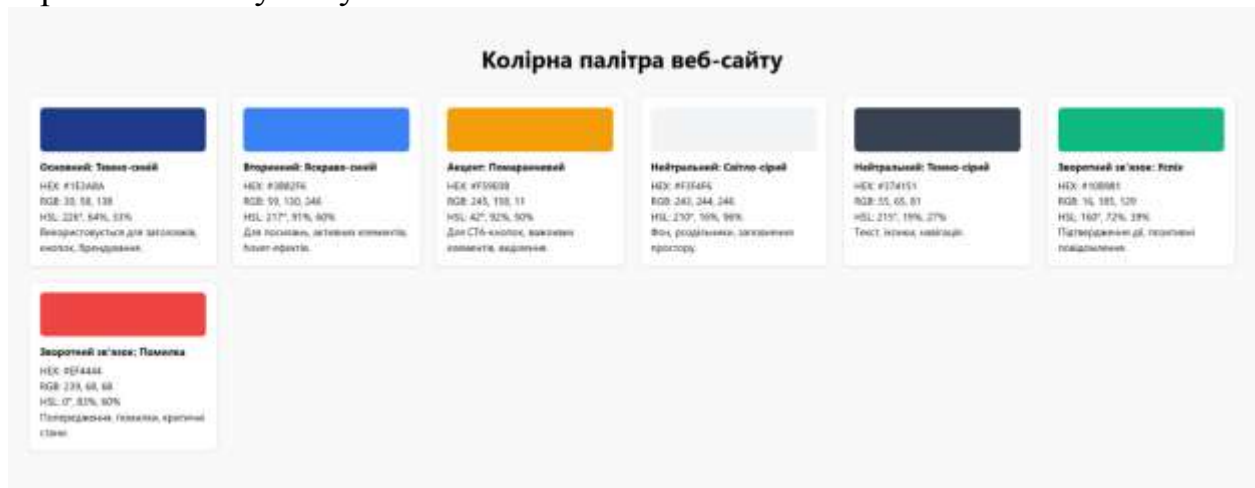


Рисунок 3.6 – Приклад добре продуманої колірної палітри для вебсайту

Має бути подано 5-7 кольорів: 1-2 основні, 1-2 додаткові, 1 акцентний, 2-3 нейтральні, а також, можливо, кольори для зворотного зв'язку (успіх, помилка, попередження). Кожен колір має бути показаний з його Hex-кодом та, можливо, RGB/HSB значеннями, а також коротким описом його призначення в палітрі.

Завдання для демонстрації

1. **Аналіз психології кольору.** Відкрийте кілька відомих вебсайтів (наприклад, Facebook, Coca-Cola, Greenpeace). Обговоріть, які основні кольори вони використовують і як ці кольори відповідають їхньому бренду та повідомленню.

2. **Робота з колірним колом.** Використовуйте онлайн-інструмент (наприклад, Adobe Color Wheel). Продемонструйте, як створювати різні колірні схеми (монохроматичну, аналогову, комплементарну, тріадну), змінюючи базовий колір.

3. **Створення палітри.** За допомогою інструменту (наприклад, Coolors.co) створіть невелику колірну палітру для уявного проєкту. Обговоріть вибір основних, додаткових та акцентних кольорів.

Питання для самоконтролю

1. Як колір впливає на психологію користувача у вебдизайні? Наведіть приклади.

2. Поясніть структуру колірного кола Іттена.

3. Опишіть основні колірні схеми (монохроматична, аналогова, комплементарна, тріадна) та їхнє застосування.

4. У чому різниця між колірними моделями RGB та HSL? Коли зручніше використовувати кожен з них?

5. Які типи кольорів зазвичай входять до добре продуманої колірної палітри?
6. Чому достатній колірний контраст є важливим для вебсайту?

Рекомендоване практичне завдання

1. Аналіз колірних палітр:

1. Виберіть 3 різні вебсайти. Для кожного сайту проаналізуйте:
 - Які основні кольори використовуються? Як вони відповідають бренду та цільовій аудиторії?
 - Яку колірну схему (або комбінацію схем) використовує сайт?
 - Як використовуються акцентні кольори? Чи є вони ефективними?
2. Оцініть колірний контраст тексту та інтерактивних елементів. Чи відповідає він стандартам доступності?
3. Оформіть свої спостереження у вигляді короткого звіту.

2. Створення колірної палітри:

1. Використовуючи онлайн-інструмент (наприклад, Adobe Color або Coolers.co), створіть власну колірну палітру для уявного вебпроєкту (наприклад, сайт кав'ярні, технологічного стартапу, екологічної організації).
2. Палітра має містити основні, додаткові, акцентні та нейтральні кольори.
3. Обґрунтуйте свій вибір кольорів, посилаючись на психологію кольору та вибрану колірну схему.
4. Збережіть палітру (наприклад, скріншот або експорт колірних кодів).

3.4 Доступність (Accessibility) та інклюзивний дизайн

У сучасному цифровому світі вебсайти та додатки стали невід'ємною частиною життя мільйонів людей, надаючи доступ до інформації, послуг, розваг та комунікації. Однак, не всі користувачі мають однаковий доступ до цих ресурсів та функціональності. Саме тут на перший план виходять концепції **доступності (Accessibility)** та **інклюзивного дизайну (Inclusive Design)**. Вебдоступність означає, що вебсайти, інструменти та технології розроблені та створені таким чином, щоб люди з різними видами інвалідності могли їх використовувати, сприймати, розуміти, навігувати та взаємодіяти з ними. Вона стосується не лише людей з постійними фізичними або сенсорними обмеженнями (такими як сліпота, глухота, порушення рухових функцій, когнітивні розлади), але й тих, хто має тимчасові (наприклад, зламана рука, тимчасова втрата зору після операції) або ситуаційні (наприклад, читання на яскравому сонці, використання

пристроїв однією рукою під час тримання дитини, повільне інтернет-з'єднання) обмеження.

Інклюзивний дизайн йде ще далі, це підхід до проектування, який з самого початку враховує максимально широкий спектр людських можливостей, потреб та досвідів. Це не про створення спеціальних, окремих версій сайту для людей з інвалідністю, а про створення єдиного, гнучкого та універсального дизайну, який працює для всіх, незалежно від їхніх здібностей, контексту використання чи технологій. Для інженера програмного забезпечення розуміння та застосування принципів доступності є не лише етичним обов'язком, що відображає соціальну відповідальність, але й важливою професійною навичкою, що дозволяє створювати якісні, конкурентоспроможні та законослухняні продукти, адже у багатьох країнах вимоги до вебдоступності закріплені на законодавчому рівні.

Чому веб-доступність важлива? Багатогранний підхід

Важливість вебдоступності можна розглядати з кількох взаємопов'язаних поглядів, кожна з яких підкреслює її критичну роль у сучасному цифровому суспільстві:

Етична та соціальна відповідальність. Кожна людина, незалежно від її фізичних, сенсорних чи когнітивних можливостей, має право на рівний доступ до інформації, освіти, працевлаштування, державних послуг та соціальної взаємодії, які все більше переміщуються в онлайн-простір. Вебдоступність забезпечує це право, усуваючи цифрові бар'єри та сприяючи інклюзивному суспільству. Це відображає фундаментальні принципи рівності та поваги до людської гідності.

Юридична відповідність та уникнення ризиків. У багатьох країнах світу (США з Section 508 та ADA, Європейський Союз з Європейським актом про доступність, Канада, Австралія та інші) існують закони та регуляції, в яких висуваються вимоги, щоб публічні та приватні вебсайти відповідали певним стандартам доступності, найчастіше WCAG. Недотримання цих вимог може призвести до серйозних юридичних наслідків, включно судові позови, значні штрафи, репутаційні втрати та необхідність дороговартісних переробок. Для компаній це прямий фінансовий та репутаційний ризик.

Бізнесова вигода та розширення аудиторії. Доступний вебсайт автоматично розширює коло потенційних користувачів, включно мільйони людей з інвалідністю та їхні родини, які становлять значний сегмент ринку. Це збільшує охоплення аудиторії, потенційний трафік, конверсію та, відповідно, прибуток. Крім того, покращення доступності часто призводить до покращення загального UX для всіх користувачів. Наприклад, чітка структура заголовків корисна для скрінрідерів, а також покращує сканування контенту для всіх. Субтитри до відео корисні не лише для глухих, але й для тих, хто дивиться відео без звуку в громадських

місцях. Швидка клавіатурна навігація зручна не лише для людей з порушеннями рухових функцій, але й для досвідчених користувачів, які віддають перевагу клавіатурі.

Покращення SEO (Search Engine Optimization). Пошукові системи, такі як Google, цінують добре структуровані, семантично правильні та доступні вебсайти. Багато практик доступності (наприклад, використання `alt`-текстів для зображень, правильна ієрархія заголовків, семантичний HTML) безпосередньо сприяють кращому індексуванню контенту пошуковими роботами, що позитивно впливає на ранжування у пошуковій видачі та збільшує органічний трафік.

Інновації та якість продукту. Проєктування з урахуванням доступності змушує команду мислити більш креативно та глибоко аналізувати функціонал, що часто приводить до інноваційних рішень, які приносять користь усім. Це підвищує загальну якість продукту, роблячи його більш надійним, гнучким та зручним.

WCAG: Міжнародні стандарти вебдоступності

Web Content Accessibility Guidelines (WCAG) – це міжнародні стандарти доступності вебконтенту, розроблені Консорціумом Всесвітньої павутини (W3C) у рамках Ініціативи вебдоступності (WAI). WCAG є найбільш авторитетним та широко прийнятим посібником з вебдоступності у світі. Вони надають детальні рекомендації щодо того, як зробити вебконтент доступним для людей з широким спектром інвалідності. WCAG організовані за чотирма основними принципами, які часто скорочують як **POUR**:

1. **Perceivable (Сприйманий).** Інформація та компоненти інтерфейсу користувача мають бути подані користувачам таким чином, щоб вони могли їх сприймати, незалежно від їхніх сенсорних можливостей. Це означає, що контент не може бути «невидимим» для всіх сенсорних модальностей. Приклади: надання текстових альтернатив для зображень (alt-текст), субтитри та текстові транскрипції для аудіо/відео контенту, достатній колірний контраст між текстом та фоном, можливість зміни розміру тексту без втрати функціональності або спотворення макета, використання різних сенсорних каналів (візуальний, слуховий, тактильний) для передачі інформації.

2. **Operable (Керований).** Компоненти інтерфейсу користувача та навігація мають бути керованими. Це означає, що користувачі мають мати можливість успішно взаємодіяти з інтерфейсом, незалежно від способу введення. Приклади: повна клавіатурна навігація (всі функції доступні без використання миші), наявність достатнього часу для читання та взаємодії (наприклад, відсутність автоматичних таймерів, які не можна вимкнути), відсутність контенту, що викликає судом (наприклад, швидкі миготіння, що можуть спровокувати епілептичні напади), логічна послідовність

фокусування під час навігації клавіатурою, можливість зупиняти, призупиняти або приховувати рухомий контент.

3. **Understandable (Зрозумілий).** Інформація та робота користувацького інтерфейсу мають бути зрозумілими. Це означає, що контент та інтерфейс не можуть бути занадто складними, заплутаними або непередбачуваними. Приклади: текст, що читається та зрозумілий (використання простої мови, глосаріїв), передбачувана поведінка інтерфейсу (елементи поводяться так, як очікує користувач, навігація є послідовною), надання допомоги користувачам уникнути та виправляти помилки (чіткі повідомлення про помилки, підказки для виправлення, можливість скасування дій).

4. **Robust (Надійний).** Контент має бути досить надійним, щоб його могли інтерпретувати різні користувацькі агенти, включно допоміжні технології, а також щоб він залишався доступним під час розвитку технологій. Це означає, що контент має бути сумісним з теперішніми та майбутніми інструментами. Приклади: чистий, валідний HTML, правильне використання ARIA-атрибутів для покращення семантики, щоб допоміжні технології могли коректно інтерпретувати складні інтерактивні елементи, забезпечення сумісності з різними браузерами та версіями операційних систем.



Рисунок 3.7 – Візуалізація принципів POUR WCAG

Чотири іконки або блоки, кожен з яких подає один принцип (Perceivable, Operable, Understandable, Robust) з коротким візуальним поясненням або ключовим словом. Наприклад, для Perceivable – іконка ока/вуха, для Operable – іконка клавіатури/миші, для Understandable – іконка мозку/знаку питання, для Robust – іконка шестерні/коду.

WCAG мають три рівні відповідності: А (найнижчий, мінімальний рівень доступності), АА (середній, найпоширеніший та рекомендований рівень для більшості вебсайтів, що забезпечує значний рівень доступності), ААА (найвищий, найскладніший для досягнення, часто застосовується для спеціалізованих ресурсів, наприклад, для людей з дуже серйозними

порушеннями). Для більшості вебсайтів та додатків ціллю є досягнення рівня AA.

Допоміжні технології (Assistive Technologies): міст до інформації

Люди з інвалідністю використовують спеціалізовані **допоміжні технології** для взаємодії з комп'ютерами та інтернетом. Вебсайт має бути спроектований та розроблений таким чином, щоб він був сумісним з цими технологіями, дозволяючи користувачам ефективно отримувати доступ до контенту та функціоналу:

Скрінрідери (Screen Readers). Програмне забезпечення, яке озвучує контент екрана для незрячих або людей з низьким зором. Вони перетворюють текст та елементи інтерфейсу на мовлення або виводять їх на брайлівський дисплей. Скрінрідери покладаються на семантичну структуру HTML, правильне використання alt-текстів для зображень, заголовків, посилань та ARIA-атрибутів для коректної інтерпретації контенту. Приклади: JAWS, NVDA, VoiceOver (macOS/iOS), TalkBack (Android).

Клавіатурна навігація. Багато людей не можуть використовувати мишу через порушення рухових функцій, тому вони взаємодіють з сайтом виключно за допомогою клавіатури (Tab для переходу між інтерактивними елементами, Shift+Tab для зворотного переходу, Enter/Space для активації елементів). Важливо, щоб всі інтерактивні елементи могли бути доступні та керовані з клавіатури, і щоб візуальний індикатор фокусу (`outline`) був завжди видимим, щоб користувач розумів, де він знаходиться на сторінці.

Екранні лупи (Screen Magnifiers). Програмне забезпечення, яке збільшує частину екрана, дозволяючи людям з низьким зором краще бачити контент. Важливо, щоб дизайн сайту добре масштабувався і не «ламався» у разі збільшення.

Голосове керування (Voice Control). Дозволяє керувати комп'ютером та веб-сайтами за допомогою голосових команд. Користувачі можуть диктувати текст, активувати посилання та кнопки голосом. Для ефективної роботи голосового керування важлива чітка семантика та доступні назви елементів.

Брайлівські дисплеї. Виводять текст у форматі Брайля, дозволяючи незрячим користувачам читати контент тактильно.

Основи реалізації доступності для Frontend-розробника. Практичні аспекти

Для Frontend-розробника реалізація доступності починається з глибокого розуміння та застосування правильної та семантичної верстки. Це не додаткова функція, а невід'ємна частина якісного коду:

1. **Семантичний HTML.** Використовуйте правильні HTML5-теги для їхнього призначення (`<header>`, `<nav>`, `<main>`, `<footer>`, `<button>`, `<a>`, `<form>`, `<input>`, `<label>`, `<article>`, `<section>`, `<aside>`). Це дозволяє скрінрідерам та іншим допоміжним технологіям правильно

інтерпретувати структуру сторінки та надавати користувачам контекст. Уникайте використання `div` для всього підряд, коли існують більш семантичні теги.

2. **Alt-текст для зображень.** Завжди надавайте осмислений та описовий `alt`-текст для зображень (``). `alt`-текст має передавати інформацію, яку незрячий користувач отримав би, бачачи зображення. Якщо зображення є чисто декоративним і не несе смислового навантаження, `alt` може бути порожнім (`alt=""`), щоб скрінрідер його ігнорував.

3. **Клавіатурна навігація та фокус.** Переконайтеся, що всі інтерактивні елементи (посилання, кнопки, поля форм, чекбокси, радіокнопки) доступні та керовані за допомогою клавіші Tab. Порядок фокусування має бути логічним (зліва направо, зверху вниз). Візуальний індикатор фокуса (`outline`) є обов'язковим для всіх інтерактивних елементів (можна стилізувати його за допомогою CSS, але ніколи не видаляйте його повністю за допомогою `outline: none;`).

4. **Контраст кольорів.** Забезпечуйте достатній контраст між текстом та фоном, а також між інтерактивними елементами та їхнім фоном. Використовуйте онлайн-інструменти для перевірки відповідності WCAG AA (наприклад, WebAIM Contrast Checker). Це критично важливо для людей з порушеннями зору та дальтонізмом.

5. **Розмір шрифту.** Дозволяйте користувачам змінювати розмір шрифту без порушення макета. Використовуйте відносні одиниці (rem, em) замість фіксованих пікселів для розмірів шрифтів та інших елементів, що дозволяє тексту масштабуватися.

6. **ARIA-атрибути (Accessible Rich Internet Applications).** Для складних інтерактивних елементів або кастомних компонентів, які не мають вбудованої семантики HTML (наприклад, вкладки, слайдери, модальні вікна, дерева), використовуйте ARIA-атрибути (`role`, `aria-label`, `aria-expanded`, `aria-haspopup`, `aria-live` тощо). ARIA надає допоміжним технологіям додаткову інформацію про роль, стан та властивості цих елементів, роблячи їх доступними. Використовуйте ARIA обережно і лише тоді, коли семантичний HTML не може вирішити проблему доступності.

7. **Навігація.** Забезпечте чітку та логічну структуру навігації. Використовуйте `<nav>` для основних блоків навігації. Надайте можливість пропустити навігацію (skip links) для користувачів скрінрідерів, щоб вони могли одразу перейти до основного контенту.

8. **Форми.** Надавайте `<label>` для кожного поля введення, пов'язуючи їх через атрибути `for`/`id`. Використовуйте атрибути, такі як `placeholder`, `aria-describedby`, `required`, `pattern`, та надавайте чіткі, читабельні скрінрідерами повідомлення про помилки та підказки.

9. **Мова.** Завжди вказуйте основну мову документа за допомогою атрибута `lang` в тегу `<html>` (наприклад, `<html lang="uk">`). Якщо на

сторінці є фрагменти іншою мовою, вказуйте `lang` для відповідних елементів.

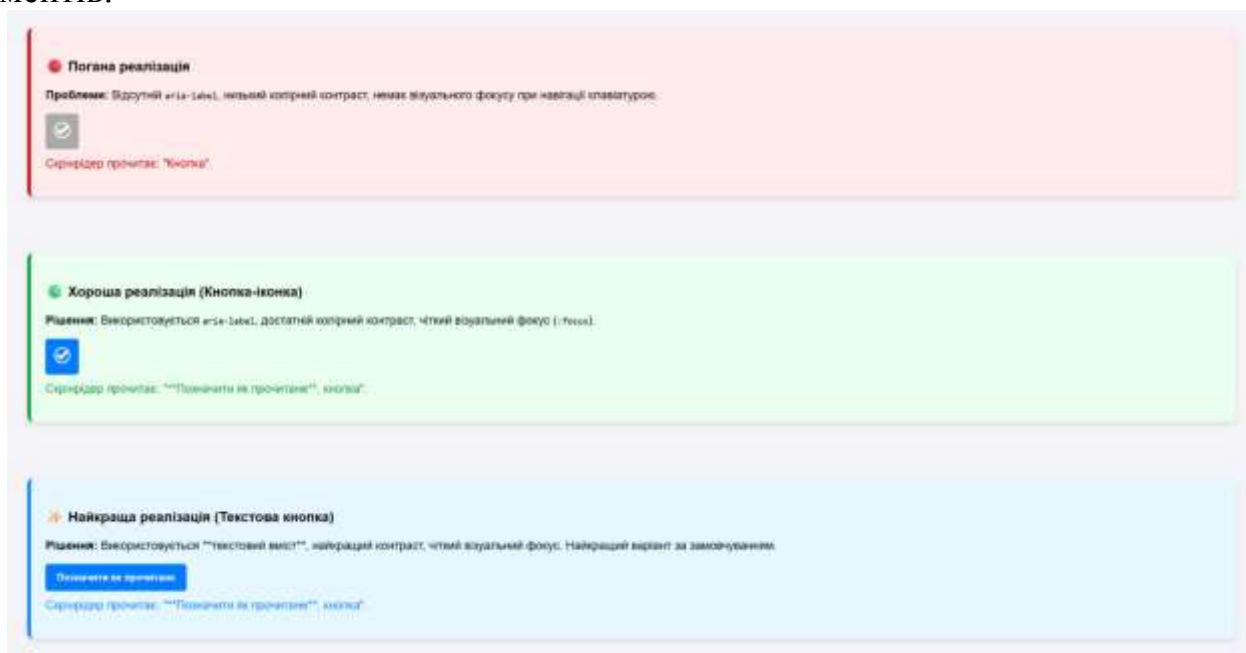


Рисунок 3.8 – Приклад поганої та хорошої реалізації доступності для одного елемента (наприклад, кнопки або посилання)

Погана реалізація: кнопка без текстового вмісту, лише іконка, без `alt` або `aria-label`, низький колірний контраст, відсутній візуальний фокус під час навігації клавіатурою. Хороша реалізація: кнопка з текстовим вмістом, або іконка з `aria-label`, достатній колірний контраст, чіткий візуальний фокус, коректний HTML-тег (`<button>`).

Інклюзивний дизайн розширює концепцію доступності, переходячи від «виправлення функцій для людей з інвалідністю» до «проектування для всіх з самого початку». Це означає – розглядати різноманітні потреби, досвіди та можливості людей як цінний ресурс, що стимулює інновації та покращує продукт для всіх. Його основні принципи:

Зрозумійте різноманітність. Усвідомлюйте, що кожен користувач унікальний, і його можливості можуть змінюватися з часом, ситуацією та контекстом.

Вирішуйте одну проблему для багатьох. Створюйте рішення, які приносять користь не лише людям з певними обмеженнями, а й ширшій аудиторії. Наприклад, голосове введення корисне не лише для людей з порушеннями рухових функцій, але й для тих, хто зайнятий іншими справами.

Забезпечте послідовний досвід. Створюйте дизайн, який є послідовним, передбачуваним та зручним незалежно від способу взаємодії, пристрою чи допоміжних технологій.

Надайте вибір. Дозвольте користувачам налаштовувати інтерфейс під свої потреби (наприклад, зміна розміру шрифту, колірної схеми, відключення анімацій).

Інклюзивний дизайн – це філософія, яка, будучи реалізованою, робить веб кращим для всіх. Це дозволяє уникнути багатьох проблем з доступністю на етапі розробки, оскільки вони закладаються в дизайн з самого початку. Для вас, як майбутніх інженерів, це є фундаментом для створення якісного, етичного та конкурентоспроможного програмного забезпечення, яке відповідає потребам сучасного суспільства.

Завдання для демонстрації

1. **Використання скрінрідера.** Продемонструвати роботу скрінрідера (наприклад, NVDA) на прикладі як доступного, так і недоступного вебсайту. Звернути увагу на те, як скрінрідер читає alt-тексти, заголовки, посилання. Підкреслити важливість семантичного HTML.

2. **Клавіатурна навігація.** На прикладі вебсайту продемонструвати навігацію виключно за допомогою клавіатури (Tab, Shift+Tab, Enter, Space). Показати, як має працювати видимий фокус. Звернути увагу на елементи, які недоступні з клавіатури.

3. **Перевірка контрасту.** Використати онлайн-інструмент (наприклад, WebAIM Contrast Checker) для перевірки контрасту кольорів тексту та фону на різних елементах вебсайту. Показати, як виправити недостатній контраст.

Питання для самоконтролю

1. У чому різниця між вебдоступністю та інклюзивним дизайном?
2. Назвіть чотири принципи WCAG (POUR) та наведіть по одному прикладу для кожного.
3. Які допоміжні технології використовуються людьми з інвалідністю для взаємодії з вебом? Як Frontend-розробка впливає на їхню роботу?
4. Наведіть 3–4 основні рекомендації для Frontend-розробника щодо забезпечення доступності вебсайту.
5. Чому важливо використовувати `alt`-тексти для зображень?
6. Що таке ARIA-атрибути і коли їх потрібно використовувати?

Рекомендоване практичне завдання

1. Аудит доступності:

1. Виберіть будь-який вебсайт, яким ви часто користуєтесь. Проведіть мініаудит його доступності, перевіряючи:
 - ✓ **Клавіатурну навігацію.** Чи можна повністю взаємодіяти з сайтом лише за допомогою клавіатури? Чи видимий фокус?

- ✓ **Alt-тексти.** Для 3–5 зображень перевірте наявність та змістовність `alt`-тексту (через інструменти розробника в браузері).
- ✓ **Контраст кольорів.** Для 2–3 основних елементів тексту перевірте контраст за допомогою інструменту WebAIM Contrast Checker.
- ✓ **Семантичність.** Оцініть, наскільки правильно використовуються HTML-теги для заголовків, списків, кнопок (через інструменти розробника).

2. Опишіть виявлені проблеми та запропонуйте можливі шляхи їх вирішення.

2. Реалізація доступного компонента:

1. Створіть простий HTML-компонент, наприклад, кастомну кнопку або посилання. Спроектуйте його таким чином, щоб він був максимально доступним:
2. Використайте семантичний HTML (`<button>` або `` з `role="button"`).
3. Забезпечте видимий стан фокусування (`:focus` стилі).
4. Надайте достатній колірний контраст.
5. Додайте `aria-label`, якщо текстовий вміст кнопки не є достатньо описовим.
6. Напишіть короткий опис того, як ви забезпечили доступність цього компонента.

РОЗДІЛ 4 HTML: СТРУКТУРА ТА СЕМАНТИКА

4.1 Вступ до HTML

HTML (HyperText Markup Language) є не просто мовою розмітки, а фундаментальним будівельним блоком, основою будь-якої вебсторінки та, по суті, всього інтернету. Це та мова, якою ми структуруємо контент, щоб він міг бути інтерпретований та відображений веббраузерами. Важливо розуміти, що HTML – це не мова програмування, оскільки він не виконує логічних операцій чи обчислень. Його основна функція – **визначення структури та змісту вебдокументів**: заголовків, параграфів, списків, зображень, посилань, таблиць, форм та інших елементів, які разом формують візуально та функціонально цілісну сторінку. Веббраузери (такі як Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge) читають HTML-документи, інтерпретують їхню розмітку і візуалізують її, перетворюючи сирий код на звичний нам графічний інтерфейс. Історія HTML бере свій початок у кінці 1980-х – початку 1990-х років, коли Тім Бернерс-Лі, працюючи в CERN, розробив концепцію World Wide Web та першу версію HTML для обміну науковою інформацією. З того часу HTML пройшов довгий шлях розвитку, еволюціонуючи від простих статичних сторінок до динамічних та інтерактивних вебдодатків. Сьогодні ми використовуємо його п'яту, найрозповсюдженішу та найсучаснішу версію – **HTML5**, яка принесла значні покращення у семантиці, мультимедіа та можливостях для розробки вебдодатків. Для майбутніх інженерів програмного забезпечення, особливо тих, хто планує працювати у Frontend-розробці, глибоке розуміння HTML є абсолютно фундаментальною навичкою. Це дозволяє не лише правильно та ефективно структурувати вебсторінки, а й забезпечувати їхню доступність для всіх користувачів, семантичність для кращого розуміння машинами та оптимізацію для пошукових систем, що є критично важливим для успіху будь-якого вебпроєкту.

Основні поняття HTML: Теги, Атрибути, Елементи – Будівельні блоки вебу

Будь-який HTML-документ, незалежно від його складності, складається з трьох ключових, взаємопов'язаних компонентів:

Теги (Tags). Це зарезервовані ключові слова мови розмітки, які використовуються для позначення початку та кінця HTML-елемента. Теги завжди оточені кутовими дужками ('<' та '>'). Більшість тегів є парними,

тобто мають відкривний тег (наприклад, `

` ) та закривний тег (наприклад, `

` ). Закривний тег відрізняється від відкривного наявністю косої риски (`/`) перед назвою. Деякі теги є одинарними або самозакривними (наприклад, `
` для розриву рядка, `` для зображення, `` для поля введення), оскільки вони не містять внутрішнього контенту. Теги говорять браузеру, як інтерпретувати і відображати контент, який вони охоплюють.

Атрибути (Attributes). Це додаткові властивості, які надають тегам додаткові характеристики, інструкції або метадані. Атрибути завжди вказуються у відкривному тегу, після назви тегу, і складаються з назви та значення, розділених знаком рівності та взятих у лапки, наприклад: `![Опис зображення](image.jpg)`. Атрибути можуть визначати джерело файлу, розмір, ідентифікатор, клас, мову, поведінку та багато іншого. Вони є ключовими для налаштування функціональності та зовнішнього вигляду елементів.

Елементи (Elements). Це повна структурна одиниця HTML-документа. Елемент містить в собі відкривний тег, всі його атрибути, вміст (текст, інші HTML-елементи) та закривний тег (якщо він є). Наприклад, `

Це мій перший параграф.</p>` є повним HTML-елементом, що містить текстовий контент. Елементи можуть бути вкладеними один в одного, формуючи деревоподібну структуру, відому як DOM (Document Object Model).

Базова структура HTML-документа. Скелет вебсторінки

Кожен HTML-документ має стандартну, обов'язкову структуру, яка говорить браузеру, як правильно його обробити та відобразити. Ця структура є основою для будь-якої вебсторінки:

1. `

2. `

3. `<head>` Цей розділ, або «голова» документа, містить метайнформацію про HTML-документ, яка не відображається безпосередньо на сторінці, але є надзвичайно важливою для браузерів, пошукових систем, соціальних мереж та інших сервісів. Тут розташовуються:

`<meta charset="UTF-8">`: вказує кодування символів для коректного відображення тексту, особливо для мов з нелатинськими алфавітами (як-от українська). UTF-8 є універсальним і рекомендованим кодуванням.

`<meta name="viewport" content="width=device-width, initial-scale=1.0">`: цей мета-тег є критично важливим для адаптивного дизайну. Він вказує браузеру, що ширина області перегляду (viewport) має дорівнювати ширині пристрою (`width=device-width`), а початковий масштаб (`initial-scale`) має бути 1.0. Без цього тегу мобільні браузери можуть відображати сторінку як на десктопі, а потім масштабувати її, що призводить до поганого досвіду користувача.

`<title>Назва сторінки</title>`: встановлює заголовок сторінки, який відображається у вкладці браузера, у закладках та як заголовок у результатах пошукових систем. Критично важливий для SEO та юзабіліті.

`<link rel="stylesheet" href="style.css">`: підключає зовнішні CSS-файли для стилізації HTML-документа. Атрибут `rel="stylesheet"` вказує на тип зв'язку, а `href` – на шлях до файлу стилів.

`<script src="script.js"></script>`: підключає JavaScript-файли для додавання інтерактивності. Часто рекомендується розміщувати теги `<script>` в кінці `<body>` для оптимізації завантаження сторінки, щоб HTML та CSS завантажилися першими.

Інші мета-теги для SEO (description, keywords), Open Graph протокол для соціальних мереж, фавіконки тощо.

4. `<body>` Цей розділ, або «тіло» документа, містить весь вміст HTML-документа, який фактично відображається користувачеві у вікні браузера. Сюди входять текст, зображення, посилання, відео, таблиці, форми та всі інші візуальні та інтерактивні елементи, з якими взаємодіє користувач. Все, що ви бачите на вебсторінці, знаходиться всередині тегу `<body>`.

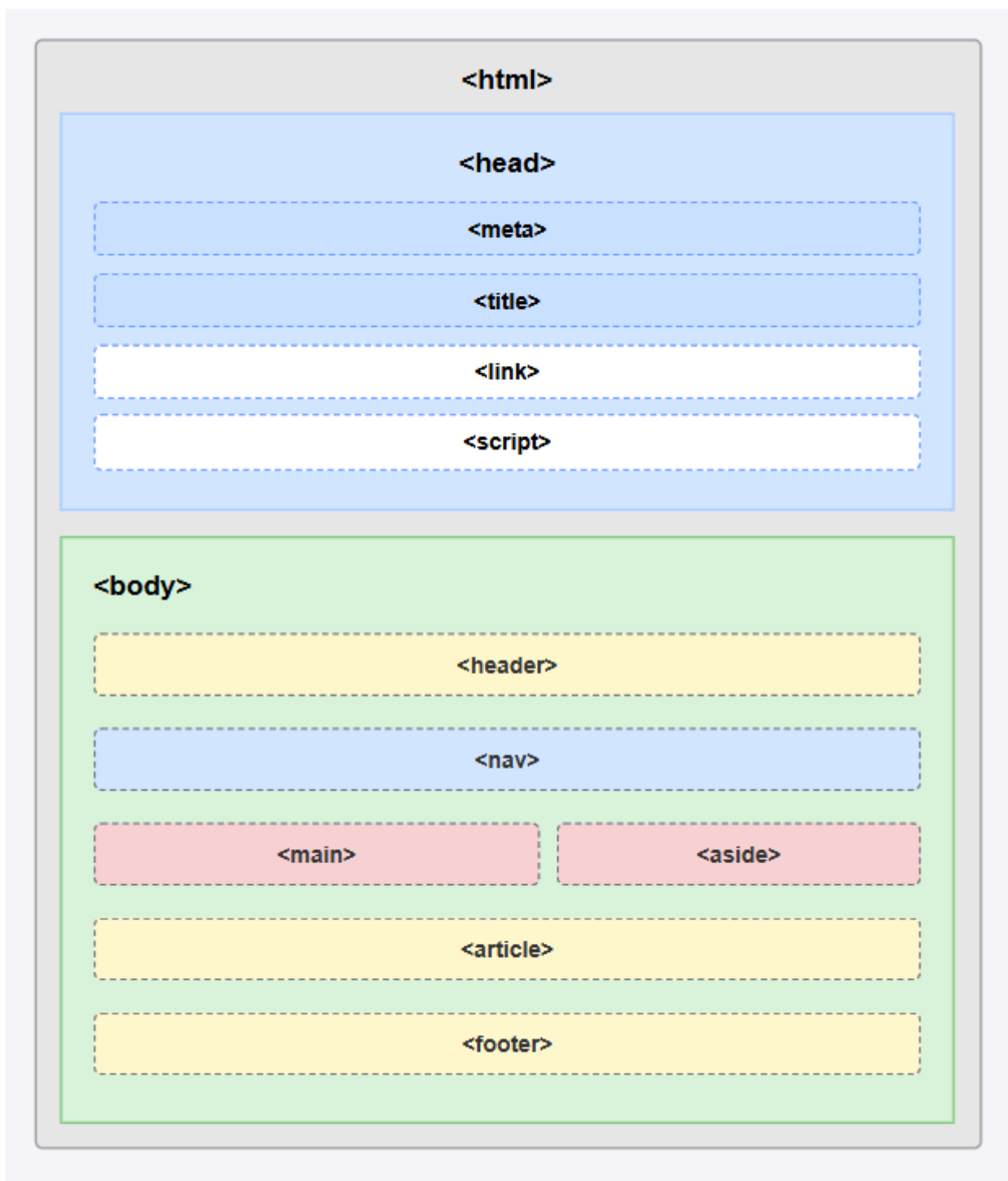


Рисунок 4.1 – Схематичне зображення базової структури HTML5-документа

4.2 Семантичний HTML5

Семантика HTML5: будемо зрозумілий веб для людей та машин

Одним з найважливіших нововведень HTML5, що кардинально змінило підхід до веброзробки, стало запровадження **семантичних тегів**. Семантика в HTML означає використання тегів, які передають не лише

спосіб відображення контенту (як це робили старі теги типу `<b>` для жирного тексту), але й його **значення, мету та структуру**. Раніше для структурування контенту розробники часто використовували універсальний тег `<div>` (division), надаючи йому ідентифікатори або класи (наприклад, `<div id="header">`, `<div class="main-content">`). Однак `<div>` не несе жодного смислового навантаження для браузера чи пошукових систем – це просто контейнер. З появою HTML5 ми отримали спеціалізовані теги, які чітко описують тип контенту, який вони містять, роблячи код більш зрозумілим та інформативним.

Чому використання семантичних тегів є настільки важливим для сучасного інженера програмного забезпечення?

Доступність (Accessibility). Семантичні теги значно покращують доступність вебсторінок для людей з інвалідністю. Допоміжні технології, такі як скрінрідери для незрячих або людей з низьким зором, краще розуміють структуру сторінки, якщо використовуються семантичні теги. Наприклад, скрінрідер може дозволити користувачеві швидко переходити від одного `<nav>` до іншого, відразу розуміючи, що це блок навігації, або від одного `<article>` до іншого, розуміючи, що це окремі статті. Це створює більш ефективний та зрозумілий досвід для користувачів допоміжних технологій.

SEO (Search Engine Optimization). Пошукові системи (Google, Bing, Yandex) використовують складні алгоритми для індексування та ранжування вебсайтів. Вони краще розуміють та індексують сайти, контент яких чітко структурований семантичними тегами, оскільки це допомагає їм краще зрозуміти зміст, контекст та важливість різних частин сторінки. Це може позитивно вплинути на позиції сайту у пошуковій видачі.

Підтримка та читабельність коду. Код, написаний з використанням семантичних тегів, стає значно читабельнішим та зрозумілішим для інших розробників (і для вас самих у майбутньому). Замість безлічі `div` з абстрактними класами типу `header-wrapper`, `main-content`, ми використовуємо інтуїтивно зрозумілі теги, такі як `<header>`, `<main>`, `<article>`. Це полегшує спільну роботу в команді, прискорює процес розробки та спрощує довгострокову підтримку проекту.

Кросбраузерність та майбутня сумісність. Семантичні теги краще підтримуються сучасними браузерами та забезпечують більш передбачуване відображення. Вони також є частиною вебстандартів, що гарантує їхню актуальність та сумісність у майбутньому.

Ключові семантичні теги HTML5 та їхнє призначення:

`<header>` – вступний контент або набір навігаційних посилань для свого найближчого елемента-предків-контенту (наприклад, `<article>`) або

для всього документа. Зазвичай містить логотип сайту, назву, слоган, основну навігацію, пошукову форму. На сторінці може бути кілька `<header>`, наприклад, один для всього документа, інший для кожної окремої `<article>`.

`<nav>` – навігаційні посилання – посилання, які дозволяють перейти на інші сторінки або до інших частин поточної сторінки. Це можуть бути: головне меню, хлібні крихти (breadcrumbs), посилання на розділи в межах сторінки. Зазвичай розташовується всередині `<header>` або `<footer>`, але може бути і окремим елементом.

`<main>` – основний, домінуючий контент документа. Вміст елемента `<main>` має бути унікальним для документа, тобто не містити повторюваних елементів, таких як навігація, логотипи, футери. На сторінці може бути лише один `<main>` елемент, і він не має бути вкладеним в `<article>`, `<aside>`, `<footer>`, `<header>` або `<nav>`.

`<article>` – самодостатній, незалежний блок контенту, який має сенс сам по собі і міг би бути розподілений незалежно від решти сторінки. Це може бути стаття в блозі, новинна публікація, коментар користувача, віджет, запис у форумі. Кожен `<article>` може містити власний `<header>`, `<footer>` та інші семантичні елементи.

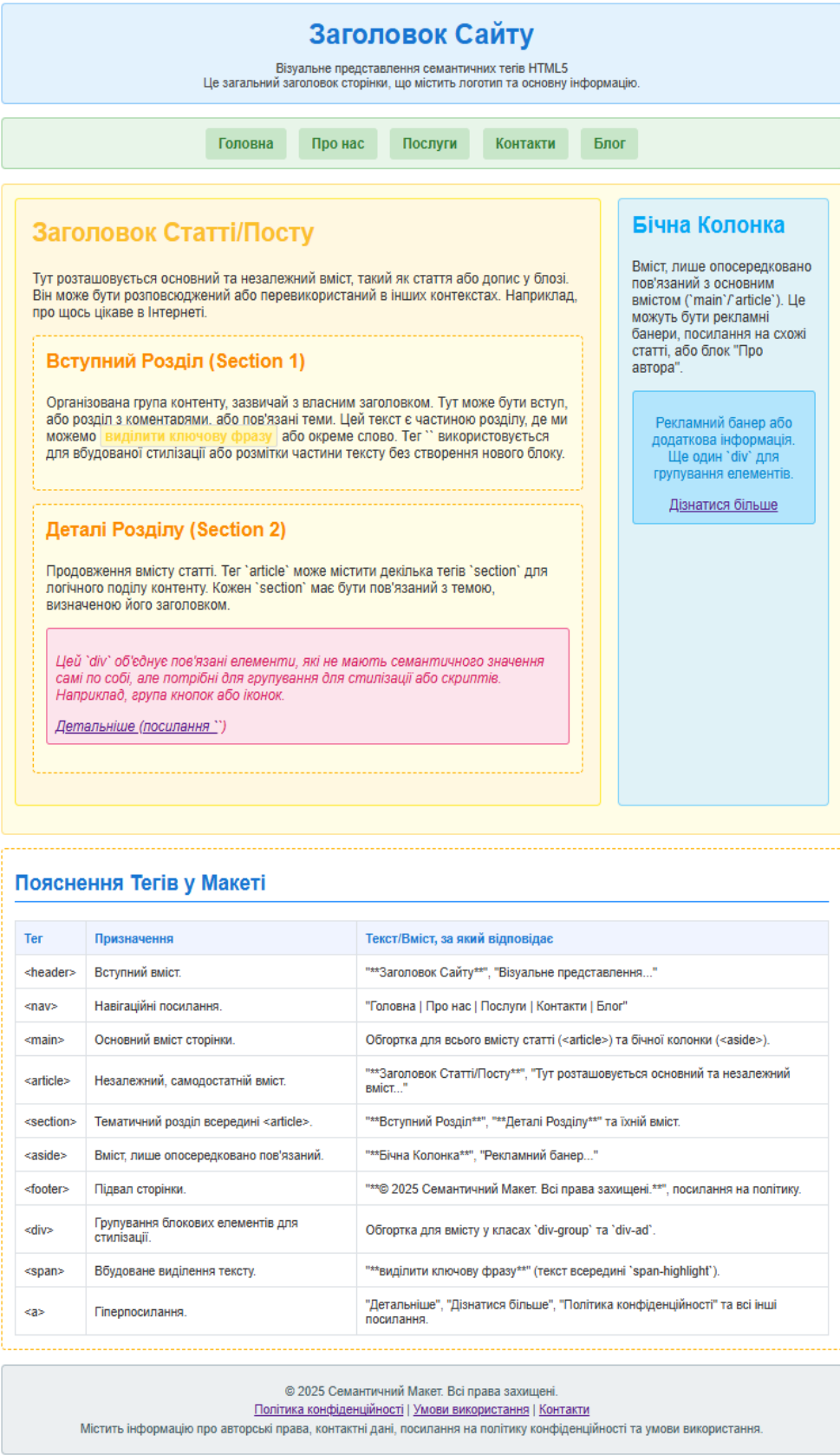
`<section>` – окремий, самостійний розділ документа. Це загальний контейнер для логічно пов'язаного контенту, який не є самодостатнім, як `<article>`, але має певну тематичну єдність. Наприклад, розділи «Про нас», «Послуги», «Відгуки», «Контакти» на сторінці. Кожен `<section>` зазвичай має заголовок (`<h1>`–`<h6>`).

`<aside>` – контент, який лише опосередковано пов'язаний з основним контентом документа. Це можуть бути бічні панелі, цитати, реклама, пов'язані посилання, рубрикатор, інформація про автора. Зазвичай розташовується збоку від основного контенту.

`<footer>` – нижній колонтитул для свого найближчого елемента-предків-секції або для всього документа. Зазвичай містить інформацію про автора, копірайт, посилання на політику конфіденційності, контактну інформацію, додаткову навігацію. Як і `<header>`, може бути кілька `<footer>` на сторінці.

`<figure>` та `<figcaption>` – тег `<figure>` використовується для об'єднання медіа-контенту (зображення, відео, аудіо, діаграми, фрагменти коду) з його заголовком (`<figcaption>`), роблячи їх семантично пов'язаними. Це покращує доступність та розуміння контенту.

`<time>` – використовується для позначення дати та часу, що може бути зрозуміло машинами (наприклад, для календарів або пошукових систем). Атрибут `<datetime>` дозволяє вказати дату у машиночитному форматі.



Правильне та послідовне використання семантичних тегів HTML5 дозволяє створювати більш структуровані, зрозумілі, доступні та оптимізовані для пошукових систем вебсторінки. Це є основою для подальшої ефективної стилізації за допомогою CSS та додавання інтерактивності за допомогою JavaScript. Як інженери, ваша відповідальність полягає в тому, щоб писати чистий, ефективний та семантично правильний HTML-код, який не лише працює, але й має сенс для людей та машин, сприяючи створенню якісного та інклюзивного вебу.

Завдання для демонстрації

1. **Базова структура HTML.** Створити порожній HTML-файл і набрати в ньому базову структуру HTML5-документа (`<!DOCTYPE>`, `<html>`, `<head>`, `<body>`). Пояснити призначення кожного тегу.

2. **Семантичні теги.** Додати до `<body>` елементи `<header>`, `<nav>`, `<main>`, `<footer>`. У середині `<main>` додати `<article>` та `<aside>`. Наповнити їх простим текстовим контентом. Продемонструвати, як браузер відображає цю структуру, навіть без CSS.

3. **Використання інструментів розробника.** Відкрити будь-який вебсайт в браузері, відкрити інструменти розробника (F12) і показати наявність та використання семантичних тегів HTML5 у структурі документа. Продемонструвати, як обхід DOM-дерева стає інтуїтивно зрозумілішим завдяки семантиці.

Питання для самоконтролю

1. Що таке HTML і яка його основна функція у веброзробці?
2. Поясніть, у чому різниця між тегом, атрибутом та елементом у HTML.
3. Навіщо потрібні `<head>` та `<body>` у структурі HTML-документа?
4. Перелічіть 5–6 важливих семантичних тегів HTML5 та поясніть їхнє призначення.
5. Чому використання семантичних тегів HTML5 важливе для доступності та SEO?
6. Який тег використовується для основного унікального контенту сторінки і скільки разів він може зустрічатися на сторінці?

Рекомендоване практичне завдання

1. **Створення сторінки з семантикою.** Створіть новий HTML-файл. Розробіть структуру простої вебсторінки (наприклад, сторінки про університетський захід або особистий блог) з використанням максимальної кількості семантичних тегів HTML5. Сторінка має містити: `<header>` з `<nav>`, `<main>` з `<article>` (яка містить заголовок, текст, зображення з `<figure>` та `<figcaption>`), `<aside>` та `<footer>`. Наповніть ці елементи відповідним, але простим текстовим контентом. Зробіть так, щоб документ

був валідним за HTML5-стандартами (можна перевірити онлайн-валідатором).

2. Аналіз семантики конкурентів. Виберіть 2–3 відомі вебсайти. Відкрийте їх у браузері та за допомогою інструментів розробника (F12) проаналізуйте їхню HTML-структуру. Зверніть увагу, наскільки активно вони використовують семантичні теги HTML5. Чи є якісь елементи, де, на вашу думку, можна було б покращити семантику? Зробіть скріншоти цікавих прикладів або недоліків.

4.3 Основні HTML-теги

Після того, як ми ознайомились з базовою структурою HTML-документа та значенням семантичних тегів HTML5, можемо перейти до детального аналізу елементів, які забезпечують наповнення вебсторінок змістовним контентом. Сьогодні ми розглянемо основні HTML-теги, що застосовуються для подання текстової інформації, створення інтерактивних гіперпосилань, вставлення зображень, а також структуризації даних у форматі списків і таблиць. Ґрунтовне розуміння цих фундаментальних елементів є необхідною передумовою професійної діяльності Frontend-розробника, оскільки вони становлять основу візуальної та інформаційної складової будь-якого вебресурсу. Коректне використання таких тегів не лише забезпечує адекватне відображення контенту в браузері, але й відіграє ключову роль у підвищенні доступності вебсторінки (Accessibility) та її оптимізації для пошукових систем (SEO).

Текст є основним носієм інформації на більшості вебсторінок, і HTML надає широкий набір тегів для його структуризації та форматування. Важливо використовувати ці теги семантично, тобто відповідно до їхнього призначення, а не лише для візуального ефекту (візуальне оформлення краще робити за допомогою CSS).

Заголовки (<h1> – <h6>). Ці теги використовуються для визначення заголовків різних рівнів. <h1> є найважливішим заголовком (зазвичай, головний заголовок сторінки), а <h6> — найменш важливим. Важливо дотримуватися ієрархії заголовків: на сторінці має бути лише один <h1>, а наступні заголовки мають йти послідовно (<h2> після <h1>, <h3> після <h2> тощо). Це критично важливо для SEO та доступності, оскільки скрінрідери використовують заголовки для навігації по сторінці.

```
html
<h1>Головний заголовок сторінки</h1>
<h2>Підзаголовок розділу</h2>
  <h3>Підзаголовок підрозділу</h3>
```

Параграфи (<p>). Тег <p> використовується для визначення текстових параграфів. Це найпоширеніший тег для основного текстового контенту. Браузери автоматично додають відступи до та після параграфів.

```
html
<p>Це перший параграф мого тексту.</p>
<p>А це другий параграф, що містить важливу
інформацію.</p>
```

Жирний текст (та)

 використовується для позначення тексту, який має сильне смислове значення або важливість. Браузери зазвичай відображають його жирним шрифтом. Це семантичний тег.

 використовується для привернення уваги до тексту, але без надання йому додаткового смислового значення. Браузери також відображають його жирним. Це несе менше семантичного навантаження, ніж .

```
html
<p>Цей текст є <strong>дуже важливим</strong>
для розуміння.</p>
<p>А цей текст просто <b>виділений жирним</b>
для візуального акценту.</p>
```

Курсив (та <i>)

 – використовується для позначення тексту, який має акцент або наголос. Браузери зазвичай відображають його курсивом. Це семантичний тег.

<i> – використовується для тексту, який виділений курсивом з візуальних причин, наприклад, для технічних термінів, іноземних слів, назв. Не несе сильного смислового навантаження.

```
html
<p>Я <em>дійсно</em> хочу, щоб ви це
запам'ятали.</p>
<p>Термін <i>HyperText Markup Language</i> є
ключовим.</p>
```

Розрив рядка (
). Одинарний тег, що створює розрив рядка, переносячи текст на новий рядок без створення нового параграфа. Використовуйте його обережно, переважно для адрес або віршів, де важливий саме розрив рядка, а не новий блок.

```
html
Адреса:<br>
Вул. Прикладна, 10<br>
Місто Київ
```

Горизонтальна лінія (<hr>): Одинарний тег, що створює горизонтальну лінію, яка використовується для тематичного розділення контенту. Це семантичний роздільник.

```
html
<p>Перший блок контенту.</p>
<hr>
<p>Другий блок контенту.</p>
```

Посилання (<a>, anchor) є фундаментальним елементом інтернету, що дозволяє користувачам переходити між вебсторінками або до різних частин однієї сторінки. Без посилань не було б World Wide Web. Тег <a> завжди використовується з атрибутом href .

- href (Hypertext Reference) вказує URL (Uniform Resource Locator) ресурсу, на який зроблено посилання. Це може бути інша вебсторінка, файл, електронна пошта або якір на поточній сторінці.

```
html
<a href="https://www.google.com">Перейти до
Google</a>
<a href="/about.html">Про нас</a>
<a href="#section1">Перейти до розділу 1</a>
<a href="mailto:info@example.com">Написати
нам</a>
```

- target: визначає, де відкриватиметься посилання.

_self (за замовчуванням): відкриває посилання в поточному вікні/вкладці.

_blank: відкриває посилання в новому вікні/вкладці.

Важливо: під час використання target="_blank" завжди додавайте rel="noopener noreferrer" для безпеки та запобігання фішинговим атакам.

```
html
<a href=https://jetiq.vntu.edu.ua/
target="_blank" rel="noopener noreferrer">Відкрити в
новій вкладці</a>
```

- title: надає додаткову інформацію про посилання, яка відображається як підказка у разі наведення курсору. Корисно для доступності.

Зображення (): візуальний контент

Зображення () є невід'ємною частиною сучасного вебдизайну, що дозволяє візуально збагачувати контент та покращувати сприйняття інформації. Тег є одинарним (самозакривним) і завжди потребує атрибутів src та alt.

- src (Source): вказує шлях до файлу зображення. Це може бути відносний шлях (наприклад, '/images/photo.jpg') або абсолютний URL.

- alt (Alternative text): надає текстовий опис зображення. Це критично важливо для доступності (скрінрідери озвучують цей текст для незрячих користувачів) та SEO (пошукові системи використовують його для розуміння змісту зображення). Якщо зображення є чисто декоративним, `alt` може бути порожнім (alt="").

- width та height: Вказують ширину та висоту зображення в пікселях. Їхнє використання допомагає браузеру зарезервувати місце для зображення до його повного завантаження, запобігаючи «стрибкам» макета (Layout Shift) та покращуючи продуктивність (Core Web Vitals).

```
html

```

- loading="lazy": атрибут для "лінивого" завантаження зображень, які знаходяться за межами початкової області перегляду (viewport). Зображення завантажуються лише тоді, коли користувач прокручує сторінку до них, що значно покращує швидкість завантаження сторінки.

```
<a href="https://vinnytsia.city/" target="_blank" title="Перейти на сайт Vinnytsia.City">
  
```

Рисунок 4.3 – Приклад HTML-коду, що демонструє використання тегів <a> та з усіма важливими атрибутами

Списки: організація переліків

HTML надає кілька типів списків для організації інформації, що робить її більш структурованою та легкою для сприйняття.

Ненумеровані (марковані) списки (, Unordered List): використовуються для переліку елементів, порядок яких не має значення. Кожен елемент списку позначається маркером (зазвичай кружечком або квадратиком).

- : контейнер для ненумерованого списку.

- (List Item): кожен елемент списку.

```
html
<ul>
  <li>Пункт 1</li>
  <li>Пункт 2</li>
  <li>Пункт 3</li>
</ul>
```

Нумеровані (впорядковані) списки (, Ordered List): використовуються для переліку елементів, порядок яких має значення (наприклад, кроки інструкції, рейтинг). Кожен елемент списку позначається числом або літерою.

- : контейнер для нумерованого списку.
- : кожен елемент списку.

```
html
<ol>
<li>Перший крок</li>
<li>Другий крок</li>
<li>Третій крок</li>
</ol>
```

Списки означень (<dl>, Description List): використовуються для створення списків термінів та їхніх означень. Корисні для глосаріїв, списків питань та відповідей.

- <dl>: контейнер для списку визначень.
- <dt> (Definition Term): термін, що визначається.
- <dd> (Definition Description): опис терміна.

```
html
<dl>
<dt>HTML</dt>
<dd>Мова розмітки гіпертексту.</dd>
<dt>CSS</dt>
<dd>Мова таблиць стилів.</dd>
</dl>
```

Таблиці (<table>): структуровані дані

Таблиці (<table>) використовуються для подання табличних даних, тобто інформації, яка логічно організована в рядки та стовпці. Важливо використовувати таблиці саме для табличних даних, а не для верстки макета сторінки (для цього існують CSS Grid та Flexbox).

Основні теги для створення таблиць:

<table>: контейнер для всієї таблиці.

<thead>: групує вміст заголовка таблиці.

<tbody>: групує основний вміст таблиці.

<tfoot>: групує вміст нижнього колонтитула таблиці.

<tr> (Table Row): визначає рядок таблиці.

<th> (Table Header): визначає заголовок стовпця або рядка. Зазвичай відображається жирним і по центру.

<td> (Table Data): визначає стандартну комірку даних у таблиці.

Атрибути для комірок таблиці:

- colspan: об'єднує комірку з кількома стовпцями.
- rowspan: об'єднує комірку з кількома рядками.

Приклад коду для HTML-таблиці:

```
<table border="1">
<tr>
<td colspan="2">Рядок 1 Стовпець 1 прос...ся на
два вправо</td>
<td>Рядок 1 Стовпець 3</td>
</tr>
<tr>
<td>Рядок 2 Стовпець 1</td>
<td rowspan="3">Рядок 2 Стовпець 2 прос...ся на 2
вниз</td>
<td>Рядок 2 Стовпець 3</td>
</tr>
<tr>
<td>Рядок 3 Стовпець 1</td>
<td>Рядок 3 Стовпець 3</td>
</tr>
</table>
```

Приклад таблиці з об'єднаними комітками

Рядок 1 Стовпець 1 простягається на два вправо		Рядок 1 Стовпець 3
Рядок 2 Стовпець 1	Рядок 2 Стовпець 2 простягається на 2 вниз	Рядок 2 Стовпець 3
Рядок 3 Стовпець 1		Рядок 3 Стовпець 3

Рисунок 4.4 – Приклад HTML-таблиці з використанням <thead>, <tbody>, <tfoot>, <th>, <td>, а також colspan та rowspan.

Правильне використання цих основних HTML-тегів є ключем до створення добре структурованих, семантичних та доступних вебсторінок. Це не лише полегшує роботу з контентом, але й забезпечує його коректне відображення та інтерпретацію різними пристроями та сервісами, включно пошукові системи та допоміжні технології. Як Frontend-розробники, ви будете постійно працювати з цими тегамі, тому глибоке розуміння їхнього призначення та атрибутів є фундаментальним для вашої професійної діяльності.

Завдання для демонстрації

1. Текстові теги. Створити HTML-сторінку, що містить заголовки різних рівнів, параграфи, жирний та курсивний текст, розриви рядків та горизонтальні роздільники. Продемонструвати, як браузер відображає ці елементи за замовчуванням.

2. Посилання та зображення. Додати на сторінку посилання на зовнішній ресурс (з `target="_blank"` та `rel="noopener noreferrer"`), внутрішнє посилання та посилання на електронну пошту. Вставити зображення з `alt`, `width` та `height` атрибутами.

3. Списки. Створити нумерований, нумерований та список означень, наповнивши їх відповідним контентом.

4. Таблиці. Створити просту таблицю з даними, використовуючи `<thead>`, `<tbody>`, `<tfoot>`, `<th>`, `<td>`. Продемонструвати використання `colspan` та `rowspan`.

Питання для самоконтролю

1. Які теги використовуються для заголовків і чому важливо дотримуватися їхньої ієрархії?

2. У чому різниця між `<strong>` та `<b>`, `<em>` та `<i>`?

3. Які атрибути є обов'язковими для тегу `<img>` і чому?

4. Коли потрібно використовувати `<ul>`, а коли `<ol>`?

5. Для чого призначені теги `<thead>`, `<tbody>`, `<tfoot>` у таблицях?

6. Поясніть призначення атрибутів `colspan` та `rowspan`.

Рекомендоване практичне завдання

1. Створення резюме у HTML: Створіть HTML-сторінку, яка буде вашим онлайн-резюме. Використайте всі вивчені теги:

- Заголовки для розділів (Контактна інформація, Досвід роботи, Освіта, Навички).
- Параграфи для описів.
- Нумеровані списки для навичок.
- Нумеровані списки для досвіду роботи (з датами).
- Посилання на ваш GitHub, LinkedIn, портфоліо.
- Ваше фото (з `alt` та розмірами).
- Таблицю для переліку курсів або проєктів.
- Зосередьтеся на семантиці та правильному використанні тегів.

2. Верстка простої статті. Візьміть будь-яку статтю з інтернету (наприклад, з Вікіпедії) та зверстайте її у HTML, використовуючи заголовки, параграфи, посилання, зображення, списки. Зверніть увагу на правильне використання тегів для структуризації контенту.

4.4 HTML-форми

Основа форми: тег <form>

Вебформи є одним з найважливіших елементів інтерактивності на будь-якому вебсайті. Вони дозволяють користувачам вводити інформацію, надсилати запити, реєструватися, авторизуватися, робити покупки, залишати коментарі та багато іншого. Без форм сучасний веб був би неможливим. Розуміння того, як створювати та правильно структурувати HTML-форми, є фундаментальною навичкою для Frontend-розробника. Правильно спроектована та реалізована форма не лише забезпечує збір необхідних даних, але й створює позитивний досвід користувача, мінімізуючи помилки та розчарування. Крім того, сучасні вебформи можуть містити валідацію даних, підказки та інтерактивні елементи, що роблять процес заповнення швидким, зручним і приємним для користувача.

Будь-яка HTML-форма починається з тегу <form>. Цей тег є контейнером для всіх елементів керування, які збирають дані від користувача.

Тег <form> має кілька важливих атрибутів:

- action: вказує URL-адресу, на яку будуть відправлені дані форми після її надсилання. Зазвичай це URL-адреса на сервері, яка обробляє дані.
- method: визначає HTTP-метод, який буде використовуватися для відправки даних форми. Найчастіше використовуються:

- GET: дані форми додаються до URL-адреси як параметри запиту. Використовується для запитів, які не змінюють стан сервера (наприклад, пошук). Дані видно в URL.

- POST: дані форми відправляються в тілі HTTP-запиту. Використовується для запитів, які змінюють стан сервера (наприклад, реєстрація, відправка коментаря, покупка). Дані не видно в URL, що робить його більш безпечним для конфіденційної інформації.

- enctype: визначає тип кодування даних під час відправки форми. Використовується з method="post".

- application/x-www-form-urlencoded (за замовчуванням): стандартне кодування.

- multipart/form-data: обов'язково використовується під час завантаження файлів на сервер.

- autocomplete: дозволяє браузеру автоматично заповнювати поля форми на основі попередньо введених даних користувача. Може бути on (увімкнено) або off (вимкнено). Можна застосовувати до окремих полів.

```
<form      action="/submit-form"      method="post"
autocomplete="on">
  <!--Елементи форми тут -->
</form>
```

Елементи введення: Тег `<input>`

Тег `<input>` є найуніверсальнішим елементом форми, який дозволяє створювати різні типи полів для введення даних. Його поведінка визначається атрибутом `type`.

- `type="text"`: однорядкове текстове поле для вільного введення. За замовчуванням.
- `type="password"`: поле для пароля. Символи під час введення приховуються.
- `type="email"`: поле для електронної пошти. Браузер може перевіряти формат email та пропонувати автозаповнення.
- `type="number"`: поле для введення чисел. Можна використовувати атрибути `min`, `max`, `step`.
- `type="date"`: поле для вибору дати (з календарем у більшості браузерів).
- `type="checkbox"`: прапорець. Дозволяє вибрати кілька варіантів. Використовується з атрибутом `value`.
- `type="radio"`: перемикач. Дозволяє вибрати лише один варіант з групи. Елементи однієї групи мають однаковий атрибут `name`.
- `type="file"`: поле для завантаження файлів. Використовується з `enctype="multipart/form-data"` у тегу `<form>`.
- `type="submit"`: кнопка для відправки форми.
- `type="reset"`: кнопка для скидання всіх полів форми до початкових значень.
- `type="hidden"`: приховане поле, яке не відображається користувачеві, але відправляється разом з іншими даними форми. Використовується для передачі службової інформації.

Інші важливі атрибути для `<input>`:

- `name`: обов'язковий атрибут, який визначає ім'я поля. Це ім'я використовується для ідентифікації даних на сервері.
- `value`: початкове значення поля або значення, яке відправляється для `checkbox` та `radio`.
- `placeholder`: текст-підказка, який відображається у полі, доки користувач не почне вводити дані.
- `disabled`: робить поле неактивним (не можна вводити дані, не відправляється з формою).
- `readonly`: робить поле доступним лише для читання (не можна змінювати, але відправляється з формою).
- `autofocus`: автоматично встановлює фокус на це поле під час завантаження сторінки.

Ці атрибути допомагають налаштувати поведінку полів введення та зробити форму більш зручною й інформативною для користувача.

Мітки для полів: Тег <label>

Тег <label> використовується для створення мітки (підпису) для елемента форми. Це критично важливо для доступності. Мітка пов'язується з полем вводу за допомогою атрибутів for та id.

- for: атрибут тегу <label>, значення якого має збігатися зі значенням атрибута id відповідного поля введення.

```
<label for="username">Ім'я користувача:</label>
<input type="text" id="username" name="username">
<input type="checkbox" id="agree" name="agree"
value="yes">
<label for="agree">Я погоджуюсь з умовами</label>
```

Багаторядкове текстове поле: Тег <textarea>

Тег <textarea> використовується для введення багаторядкового тексту (наприклад, коментарі, повідомлення). На відміну від <input type="text">, він не має атрибута value, а вміст розміщується між відкритим та закритим тегами.

- rows: визначає видиму кількість рядків.
 - cols: визначає видиму кількість стовпців.
- ```
<label for="comment">Ваш коментар:</label>
<textarea id="comment" name="comment" rows="5"
cols="30" placeholder="Введіть ваш коментар...">
</textarea>
```

### Кнопки: Тег <button>

Тег <button> використовується для створення кнопок. Він є більш гнучким, ніж <input type="button"> або <input type="submit">, оскільки може містити HTML-контент (зображення, текст, іконки).

- type: визначає поведінку кнопки.
- submit (за замовчуванням): відправляє форму.
- reset: скидає форму.
- button: звичайна кнопка, яка не має стандартної поведінки (використовується з JavaScript).

```
<button type="submit">Відправити</button>
<button type="button">Натисни мене</button>
```

### Низхідні списки: Тег <select>

Тег <select> використовується для створення низхідних списків або списків з можливістю множинного вибору.

- `<option>`: подає один варіант у списку. Атрибут `value` визначає значення, яке буде відправлено з формою.
- `<optgroup>`: групує пов'язані варіанти у низхідному списку.

```
<label for="country">Оберіть країну:</label>
<select id="country" name="country">
<option value="">--Оберіть-- </option>
<option value="ua"> Україна</option>
<option value="pl">Польща</option> </select>
```

Фрагмент коду для HTML-форми:

```
<form>
<table border="1" cellpadding="6"
cellspacing="0">
<tr>
<td><label for="name">Ім'я:</label></td>
<td><input type="text" id="name"
name="name"></td>
</tr>
<tr>
<td><label for="email">Email:</label></td>
<td><input type="email" id="email" name="email">
</td>
</tr>
<tr>
<td><label for="password">Пароль:</label></td>
<td><input type="password" id="password"
name="password"></td>
</tr>
<tr>
<td><label>Стать:</label></td>
<td>
<input type="radio" id="male" name="gender"
value="male">
<label for="male">Чоловіча</label>
<input type="radio" id="female" name="gender"
value="female">
<label for="female">Жіноча</label>
</td>
</tr>
<tr>
<td><label for="subscribe">Підписка:</label></td>
<td>
```

```



```

Імя: \_\_\_\_\_  
Email: \_\_\_\_\_  
Пароль: \_\_\_\_\_  
Стать:  
☐ Чоловіча  
☐ Жіноча  
Підписка:  
☒ Отримувати новини  
Про себе: \_\_\_\_\_  
Роль:  
☒ Користувач  
Надіслати Очистити

Рисунок 4.5 – Приклад HTML-форми, що містить різні типи полів <input> (text, email, password, checkbox, radio), <textarea>, <select> з <option> та <button>

## Валідація форм HTML5

HTML5 надає вбудовані механізми валідації форм, які дозволяють перевіряти дані на стороні клієнта (у браузері) до їхньої відправки на сервер. Це покращує досвід користувача, надаючи миттєвий зворотний зв'язок та зменшуючи кількість помилок.

- `required`: робить поле обов'язковим для заповнення.
- `pattern`: визначає регулярний вираз, якому має відповідати значення поля.
- `min`, `max`: для числових полів (`type="number"`, `type="range"`) визначають мінімальне та максимальне значення.
- `minlength`, `maxlength`: для текстових полів (`type="text"`, `type="password"`, `type="email"`, `type="tel"`, `<textarea>`) визначають мінімальну та максимальну довжину введеного тексту.
- `type="email"`, `type="url"`, `type="tel"`: браузер автоматично перевіряє формат введених даних.

```
<label for="username">Ім'я користувача (мін. 3
символи):</label>
```

```
<input type="text" id="username" name="username"
required minlength="3">
```

```
<label for="phone">Телефон (формат: 0XX-XXX-XX-
XX):</label>
```

```
<input type="tel" id="phone" name="phone"
pattern="[0-9]{3}-[0-9]{3}-[0-9]{2}-[0-9]{2}">
```

## Доступність форм

Доступність форм є критично важливою для користувачів допоміжних технологій. Основні принципи:

- Використання `<label>`: завжди пов'язуйте `<label>` з відповідним полем введення за допомогою `for/id`. Це дозволяє скрінрідерам озвучувати мітку за фокусування на полі.
- Порядок фокусування: переконайтеся, що порядок фокусування (за допомогою `Tab`) є логічним.
- Повідомлення про помилки: надавайте чіткі та зрозумілі повідомлення про помилки, які можуть бути прочитані скрінрідерами (наприклад, за допомогою `aria-live` регіонів або `aria-describedby`).
- `placeholder`: використовуйте `placeholder` як підказку, а не як заміну `<label>`, оскільки скрінрідери можуть ігнорувати `placeholder`.

Приклад коду з валідацією:

```
<form>
<label for="username">Ім'я користувача: </label>

<input type="text" id="username" name="username"
required pattern="[A-Za-zА-Яа-яІіїіЄє]{3,}">
<div class="error" id="nameError">Введіть
щонайменше 3 літери (без цифр).</div>

<label for="email">Email:</label>

<input type="email" id="email" name="email"
required>
<div class="error" id="emailError">Введіть
коректну адресу електронної пошти.</div>

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

Ім'я користувача: \_\_\_\_\_

(якщо не відповідає шаблону то "Введіть щонайменше 3 літери (без цифр)")

Email: \_\_\_\_\_

(якщо неправильний то "Введіть коректну адресу електронної пошти")

Рисунок 4.6 – Приклад форми з валідацією та елементами доступності.

Створення ефективних та доступних форм потребує уваги до деталей та розуміння потреб користувачів. Як Frontend-розробники, ви будете відповідальні за реалізацію цих форм, тому глибоке знання HTML-елементів, їхніх атрибутів та принципів валідації та доступності є ключем до успіху.

### Завдання для демонстрації

1. Базова форма: створити HTML-форму з полями для імені, email, пароля, прапорцем згоди та кнопкою відправки. Продемонструвати роботу GET та POST методів.

2. Різні типи <input>: додати до форми поля різних типів: number, date, radio (група з 2–3 варіантів), file. Пояснити їхнє призначення.

3. Валідація: додати до полів `required`, `minlength`, `maxlength`, `pattern`. Продемонструвати, як браузер виконує валідацію та показує повідомлення про помилки.

4. Доступність: переконатися, що всі поля мають пов'язані `<label>`. Продемонструвати навігацію по формі за допомогою клавіатури.

### **Питання для самоконтролю**

1. Які основні атрибути тегу `<form>` і для чого вони використовуються?
2. Назвіть 5 різних типів `<input>` та поясніть їхнє призначення.
3. Чому тег `<label>` є критично важливим для доступності форм?
4. У чому різниця між `<input type="submit">` та `<button type="submit">`?
5. Які атрибути HTML5 використовуються для клієнтської валідації форм?
6. Як забезпечити доступність форми для користувачів, які використовують скрінрідери?

### **Рекомендоване практичне завдання**

1. Створення форми реєстрації/авторизації: Створіть HTML-сторінку з формою реєстрації або авторизації. Форма має містити:

- поля для імені користувача, email, пароля (з підтвердженням);
- прапорець згоди з умовами використання;
- кнопку відправки.

2. Використайте атрибути `required`, `minlength`, `maxlength`, `pattern` для валідації.

3. Забезпечте повну доступність форми (використання `<label>`, логічний порядок фокусування).

4. Форма зворотного зв'язку: Створіть форму зворотного зв'язку, яка містить:

- поля для імені, email;
- низхідний список для вибору теми звернення;
- текстову область для повідомлення;
- кнопку відправки.

5. Реалізуйте валідацію та доступність.

## РОЗДІЛ 5 CSS: СТИЛІЗАЦІЯ ТА LAYOUT

### 5.1 Основи CSS

#### Синтаксис CSS: Як пишуться правила

Після того, як ми навчилися структурувати контент за допомогою HTML, настав час надати нашим веб-сторінкам візуальної привабливості та стилю. Саме для цього існує CSS (Cascading Style Sheets), або Каскадні таблиці стилів. CSS – це мова, яка описує, як HTML-елементи мають відображатися на екрані, папері або в інших медіа. Вона дозволяє відокремити вміст (HTML) від його подання (CSS), що є одним з фундаментальних принципів сучасного вебдизайну та розробки. Це відокремлення забезпечує гнучкість, дозволяючи змінювати зовнішній вигляд цілого вебсайту, змінюючи лише один CSS-файл, без необхідності редагувати кожен HTML-документ. CSS контролює кольори, шрифти, розміри, відступи, розташування елементів, анімації та багато іншого. Для Frontend-розробника глибоке розуміння CSS є такою ж важливою навичкою, як і знання HTML, оскільки саме CSS перетворює сиру структуру на візуально привабливий та функціональний інтерфейс. Крім того, сучасний CSS надає розширені можливості для створення адаптивного дизайну, медіа-запитів, трансформацій та інтерактивних ефектів, що робить вебсторінки зручними для користувачів на будь-яких пристроях.

Кожне правило CSS складається з двох основних частин: селектора та блоку оголошень.

- Селектор: вказує, до яких HTML-елементів буде застосовано правило. Селектор може бути назвою тегу, класом, ідентифікатором або іншим способом вибору елементів.

- Блок оголошень: містить одне або кілька оголошень, укладених у фігурні дужки {}. Кожне оголошення складається з властивості та її значення, розділених двокрапкою «:», і завершується крапкою з комою «;».

```
/*Це коментар у CSS */
p{/*Селектор:всі параграфи*/
color: blue; /* Оголошення: властивість color,
значення blue */
font-size: 16px; /* Оголошення: властивість font-
size, значення 16px*/
}
.my-class { /* Селектор: елементи з класом my-
class */
background-color:#f0f0f0;
padding:10px;
}
```

## Способи підключення CSS: де розміщувати стилі

Існує три основні способи підключення CSS до HTML-документа, кожен з яких має свої переваги та сценарії використання:

- Зовнішні таблиці стилів (External Style Sheets): це найбільш рекомендований та поширений спосіб. Стили зберігаються в окремому файлі з розширенням .css (наприклад, style.css) і підключаються до HTML-документа за допомогою тегу <link> у розділі <head>.

- *Переваги.* Чітке розділення HTML та CSS, легкість підтримки та зміни стилів для всього сайту (змінюючи один файл, ви змінюєте вигляд багатьох сторінок), кешування браузером (прискорює завантаження сторінок після першого відвідування).

- *Сценарій.* Ідеально підходить для великих проєктів та більшості веб-сайтів.

```
html
<head>
<link rel="stylesheet" href="style.css">
</head>
```

- Внутрішні таблиці стилів (Internal Style Sheets): стилі розміщуються безпосередньо в HTML-документі, всередині тегу <style> у розділі <head>.

- *Переваги.* Зручно для однієї сторінки або для швидкого тестування стилів.

- *Недоліки.* Змішування HTML та CSS, стилі не кешуються окремо, ускладнює підтримку, якщо стилі потрібно застосувати до кількох сторінок.

- *Сценарій.* Використовується рідко, переважно для невеликих, односторінкових проєктів або для стилів, які є унікальними для конкретної сторінки.

```
html
<head>
<style>
h1 {
color: green;
}
</style>
</head>
```

- Інлайн-стили (Inline Styles): стилі застосовуються безпосередньо до окремого HTML-елемента за допомогою атрибута style.

- *Переваги.* Найвищий пріоритет (специфічність), зручно для швидкого тестування або для застосування унікальних стилів до одного елемента.

- *Недоліки.* Повністю змішує HTML та CSS, ускладнює підтримку, не дозволяє перевизначати стилі з інших джерел, погано для доступності та SEO. Вважається поганою практикою.

- *Сценарій.* Використовується дуже рідко, переважно для динамічно генерованих стилів за допомогою JavaScript або для швидкого налагодження.

```
html
<p style="color: red; font-weight: bold;">Цей
текст червоний і жирний.</p>
```

## Каскадність та Специфічність

Назва «Каскадні таблиці стилів» не випадкова. Каскадність означає, що стилі можуть надходити з різних джерел (браузер, користувач, автор сайту) і застосовуватися до одного й того самого елемента. Браузер вирішує, який стиль застосувати, керуючись певними правилами. Специфічність – це міра того, наскільки конкретним є селектор. Чим вища специфічність, тим вищий пріоритет має правило. Порядок пріоритету (від найнижчого до найвищого):

- Універсальні селектори (\*)
- Селектори тегів (p, h1)
- Селектори класів (.my-class), селектори атрибутів ([type="text"]), псевдокласи (:hover)
- Селектори ID (#my-id)
- Інлайн-стилі (style="...")
- !important (потрібно уникати, оскільки порушує каскадність)

Якщо два правила мають однакову специфічність, застосовується те, що оголошено пізніше в коді.

## Базові селектори (елементів, класів, ID)

Селектори є серцем CSS, дозволяючи нам точно вказувати, до яких HTML-елементів застосовувати стилі. Існує багато типів селекторів, які можна комбінувати для досягнення високої точності.

Базові селектори:

- Універсальний селектор (\*). Вибирає всі HTML-елементи на сторінці. Часто використовується для скидання базових стилів.

```
css{
margin: 0;
padding: 0;
```

```
box-sizing: border-box;
}
```

- Селектор тегу (Type Selector): Вибирає всі елементи з вказаним ім'ям тегу.

```
css
p {
color: #333;
}
h1 {
font-size: 2em;
}
```

- Селектор класу (.class). Вибирає всі елементи, які мають вказаний атрибут class. Класи можуть бути застосовані до багатьох елементів.

```
html
<p class="highlight">Виділений текст</p>
<div class="highlight">Виділений блок</div>

css
.highlight {
background-color: yellow;
}
```

- Селектор ID (#id): Вибирає елемент з вказаним атрибутом id. ID має бути унікальним на сторінці.

```
html
<div id="main-header">...</div>
css
#main-header {
border-bottom: 1px solid #ccc;
}
```

## Комбінатори (нащадок, дитина, сусід)

Комбінатори використовуються для вибору елементів на основі їхнього розташування відносно інших елементів.

- Селектор нащадка (Descendant Selector, (пробіл)). Вибирає всі елементи, які є нащадками вказаного елемента (незалежно від рівня вкладеності).

```
css
div p {/Вибере всі <p> всередині будь-якого
<div>/
line-height: 1.5;
}
```

- Дочірній селектор (Child Selector, >). Вибирає елементи, які є прямими дочірніми елементами вказаного елемента.

```
css
ul > li {/Вибере тільки прямі всередині
/
list-style-type: square;
}
```

- Сусідній селектор (Adjacent Sibling Selector, +). Вибирає елемент, який є безпосереднім сусідом (наступним елементом на тому самому рівні) вказаного елемента.

```
css
h1 + p {/ Вибере <p>, який йде одразу після <h1>
/
margin-top: 0;
}
```

- Загальний сусідній селектор (General Sibling Selector, ~). Вибирає всі елементи, які є сусідами (наступними елементами на тому самому рівні) вказаного елемента.

```
css
h2 ~ p { / Вибере всі <p>, які йдуть після <h2>
на тому ж рівні /
text-indent: 20px;
}
```

## Селектори атрибутів

Вибирають елементи на основі наявності або значення їхніх атрибутів.

- [attribute]: вибирає елементи, що мають вказаний атрибут.

```
css
[title] { / Вибере всі елементи, що мають атрибут
title /
border-bottom: 1px dotted #999;
}
```

- [attribute="value"]: вибирає елементи, що мають вказаний атрибут з точним значенням.

```
css
input[type="submit"] { / Вибере <input> з
type="submit" /
background-color: #007bff;
color: white;
}
```

- [attribute~="value"]: вибирає елементи, що мають вказаний атрибут, значення якого містить вказане слово (розділене пробілами).

```
CSS
[class~="button"] { / Вибере елементи, у яких
клас містить слово "button" /
cursor: pointer;
```

- [attribute^="value"]: вибирає елементи, значення атрибута яких починається з вказаного рядка.

```
CSS
a[href^="https"] { / Вибере посилання, що
починаються з "https" /
color: green;
}
```

- [attribute\$="value"]: вибирає елементи, значення атрибута яких закінчується вказаним рядком.

```
CSS
a[href$=".pdf"] { / Вибере посилання, що
закінчуються на ".pdf" /
font-weight: bold;
}
```

- [attribute\*="value"]: вибирає елементи, значення атрибута яких містить вказаний рядок у будь-якому місці.

```
CSS
[class="icon"] { / Вибере елементи, у яких клас
містить "icon" */
vertical-align: middle;
}
```

Приклад коду:

```
<body>
 <h2>Приклад CSS-селекторів</h2>
 <p>Цей абзац обраний селектором тегу
 <code>p</code>.</p>
 <p class="my-class">Цей абзац має клас
 <code>my-class</code>.</p>
 <p id="my-id">Цей абзац має ID <code>my-
 id</code>.</p>
 <div>
 <p>Цей абзац всередині <code>div</code>
 (селектор <code>div p</code>).</p>
 <p class="my-class">Ще один абзац у div з
 класом <code>my-class</code>.</p>
 </div>
```

```


 Пункт списку 1 (директний нащадок
ul)
 Пункт списку 2
 Пункт списку 3

<form>
 <label for="name">Ім'я:</label>
 <input type="text" id="name"
name="name">

 <label for="email">Email:</label>
 <input type="email" id="email" name="email">
</form>
</body>

```

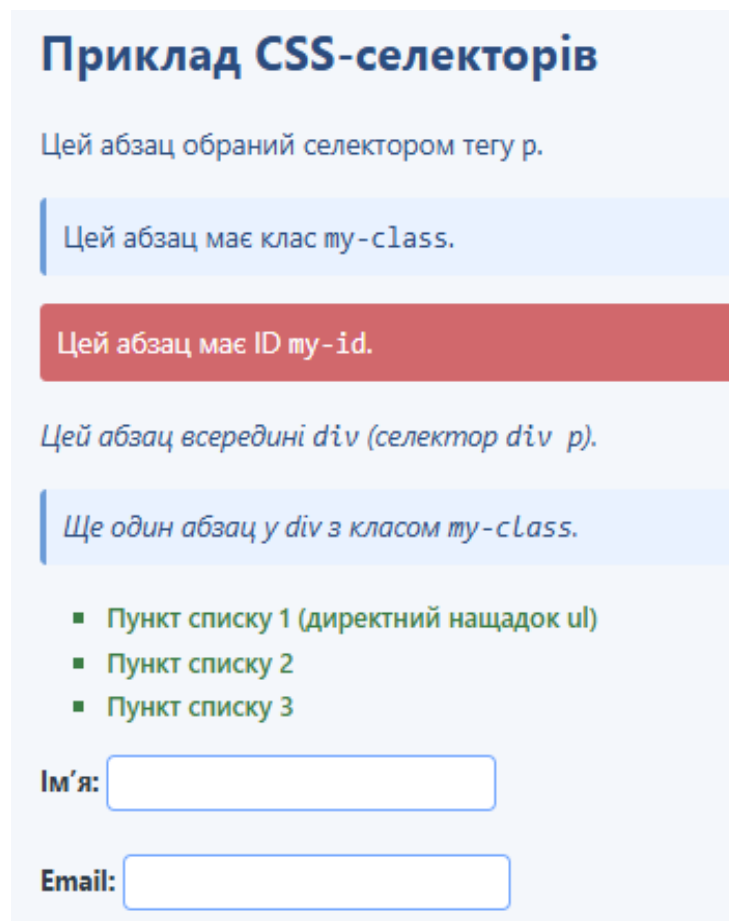


Рисунок 5.1 – Приклад HTML-коду з різними типами селекторів CSS та відповідним CSS-кодом.

Розуміння та вміле використання CSS-селекторів є ключовим для ефективної та чистої стилізації вебсторінок. Це дозволяє створювати гнучкі та підтримувані стилі, які легко адаптувати та розширювати.

## **Завдання для демонстрації**

1. Синтаксис та підключення. Створити HTML-сторінку та зовнішній CSS-файл. Підключити CSS-файл. Написати кілька базових CSS-правил для тегів h1 та p, використовуючи властивості color та font-size. Продемонструвати інлайн-стили та внутрішні стилі, пояснивши їхній пріоритет.

2. Базові селектори. Створити HTML-сторінку з різними елементами: кілька параграфів, один з яких має клас highlight, інший – ID unique-text. Додати div з класом container. Написати CSS-правила, використовуючи селектори тегів, класів та ID, щоб стилізувати ці елементи по-різному.

3. Комбінатори. Додати до HTML-сторінки вкладені списки (<ul> всередині <ul>), а також елементи h2 та p, що йдуть один за одним. Написати CSS-правила, використовуючи селектори нащадка, дочірнього, сусіднього та загального сусіднього, щоб продемонструвати їхню роботу.

4. Селектори атрибутів. Додати на сторінку посилання з href на https та .pdf, а також поля введення з type="text" та type="email". Написати CSS-правила, використовуючи різні селектори атрибутів, щоб стилізувати ці елементи.

## **Питання для самоконтролю**

1. Що таке CSS і яка його основна функція у веброзробці?
2. Опишіть синтаксис CSS-правила (селектор, властивість, значення).
3. Назвіть три способи підключення CSS до HTML-документа та вкажіть, який є найбільш рекомендованим і чому.
4. Поясніть поняття «каскадність» та «специфічність» у CSS.
5. Наведіть приклади базових селекторів (тег, клас, ID, універсальний).
6. Поясніть різницю між селектором нащадка та дочірнім селектором.
7. Наведіть 3 приклади селекторів атрибутів та поясніть, коли їх можна використовувати.

## **Рекомендоване практичне завдання**

1. Створення стилізованої сторінки. Створіть HTML-сторінку з різними елементами: заголовки, параграфи, списки, посилання, зображення. Створіть окремий CSS-файл і підключіть його. Застосуйте до елементів сторінки різні стилі, використовуючи:

- Селектори тегів для базових стилів.
- Селектори класів для повторюваних стилів (наприклад, .button, .card).
- Селектори ID для унікальних елементів (наприклад, #main-navigation).

- Комбінатори для стилізації вкладених елементів (наприклад, `nav a, ul > li`).
  - Селектори атрибутів для стилізації посилань за їхнім `href` або полів вводу за їхнім `type`.
  - Зосередьтеся на створенні візуально привабливої та добре структурованої сторінки.
  - Експерименти зі специфічністю: Створіть HTML-сторінку з одним параграфом, який має і клас, і ID.
2. Напишіть кілька CSS-правил, які націлені на цей параграф, використовуючи різні селектори (тег, клас, ID, інлайн-стиль).
  3. Змініть порядок правил та їхню специфічність, щоб зрозуміти, як браузер вибирає, який стиль застосувати. Опишіть свої спостереження.

## 5.2 Блокова модель (Box Model)

### Складові: `content`, `padding`, `border`, `margin`

У вебдизайні кожен HTML-елемент, який відображається на сторінці, розглядається браузером як прямокутний блок. Ця концепція відома як блокова модель (Box Model). Розуміння блокової моделі є абсолютно фундаментальним для будь-якого Frontend-розробника, оскільки вона визначає, як елементи займають простір на сторінці, як вони взаємодіють один з одним та як їх можна позиціонувати та стилізувати. Без чіткого розуміння Box Model неможливо ефективно керувати макетом сторінки, створювати адаптивний дизайн або точно відтворювати дизайнерські макети. Тому розглянемо детально складові блокової моделі: контент, внутрішні відступи (`padding`), рамки (`border`) та зовнішні відступи (`margin`), а також важливу властивість `box-sizing`, яка кардинально змінює поведінку розмірів елементів.

Блокова модель складається з чотирьох основних шарів, які оточують вміст елемента, наче цибулина:

- Контент (Content). Це центральна частина блока, де знаходиться фактичний вміст елемента – текст, зображення, відео або інші вкладені HTML-елементи. Розмір контенту визначається властивостями `width` та `height` (якщо `box-sizing` не змінено).
- Внутрішній відступ (Padding). Це простір між контентом елемента та його рамкою (`border`). `Padding` створює «повітря» навколо вмісту, покращуючи читабельність та візуальне сприйняття. Колір фону елемента поширюється на область `padding`. `Padding` може бути заданий для всіх чотирьох сторін (зверху, справа, знизу, зліва) окремо або одночасно.
- Рамка (Border). Це лінія, яка оточує `padding` та контент елемента. Рамка візуально відокремлює один елемент від іншого. Вона

має три основні властивості: `border-width` (товщина), `border-style` (стиль, наприклад, `solid`, `dotted`, `dashed`) та `border-color` (колір). Рамка також може бути задана для кожної сторони окремо.

- Зовнішній відступ (`Margin`). Це простір за межами рамки елемента, який відокремлює його від сусідніх елементів. `Margin` є прозорим і не успадковує колір фону елемента. Він використовується для створення відстані між блоками на сторінці. `Margin` також може бути заданий для всіх чотирьох сторін окремо або одночасно.

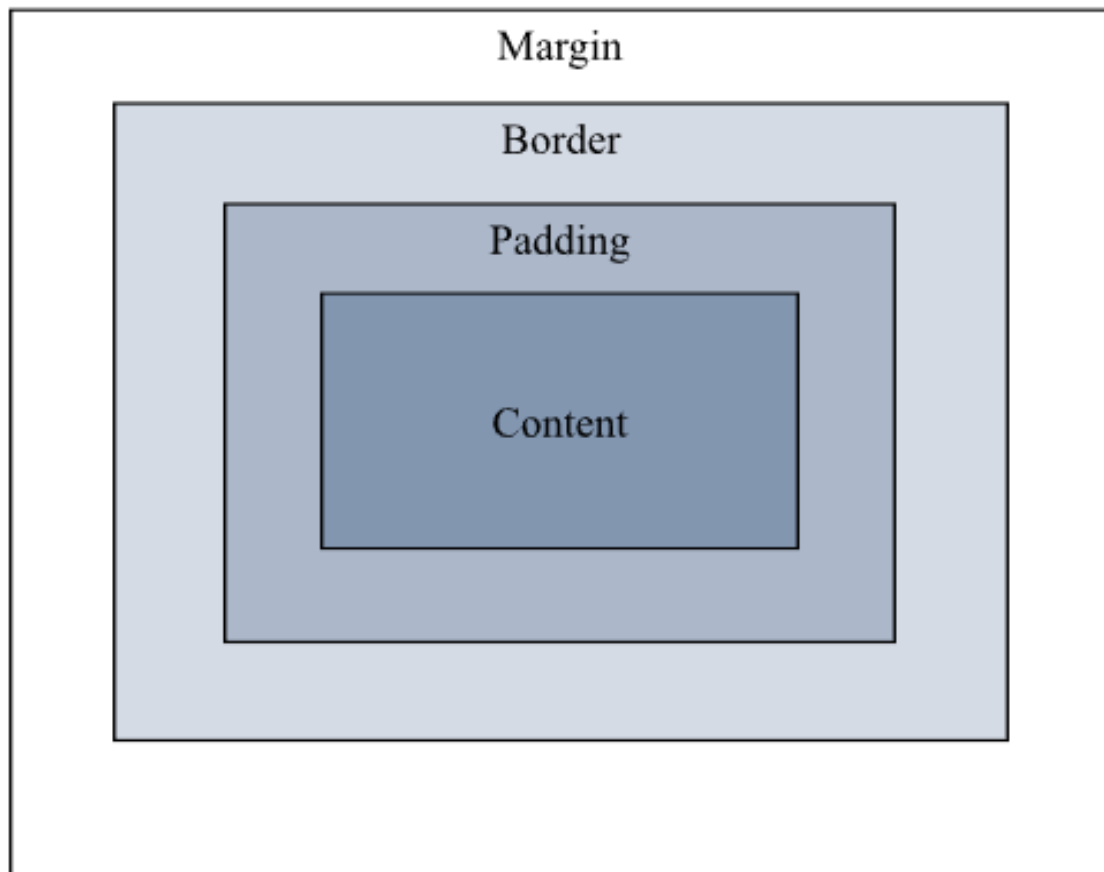


Рисунок 5.2 – Схематичне зображення блокової моделі (Box Model).

### Властивості `margin`, `padding`, `border`

Ці властивості дозволяють детально контролювати кожен шар блокової моделі.

1. `padding` (Внутрішній відступ)
  - `padding-top`, `padding-right`, `padding-bottom`, `padding-left`: задають відступи для кожної сторони окремо.
  - Скорочений запис `padding`:
  - `padding: 10px`; (всі 4 сторони по 10px)

- padding: 10px 20px; (зверху/знизу 10px, справа/зліва 20px)
- padding: 10px 20px 30px; (зверху 10px, справа/зліва 20px, знизу 30px)
- padding: 10px 20px 30px 40px; (зверху 10px, справа 20px, знизу 30px, зліва 40px - за годинниковою стрілкою)

## 2. border (Рамка)

- border-width, border-style, border-color: задають товщину, стиль та колір рамки.
- Скорочений запис border: border: 1px solid black; (товщина, стиль, колір).
- Можна задавати для кожної сторони окремо: border-top, border-right, border-bottom, border-left.
- Також можна задавати окремі властивості для кожної сторони: border-top-width, border-top-style, border-top-color.
- border-radius: дозволяє заокруглити кути рамки, створюючи більш м'який вигляд елементів.

## 3. margin (Зовнішній відступ)

- margin-top, margin-right, margin-bottom, margin-left: задають відступи для кожної сторони окремо.
- Скорочений запис margin: аналогічно padding.
- margin: auto;; для блокових елементів, що мають задану ширину, margin: auto; для margin-left та margin-right дозволяє центрувати елемент по горизонталі.

## 4. box-sizing` : керування розмірами елементів

Властивість box-sizing кардинально змінює те, як браузер розраховує загальну ширину та висоту елемента, враховуючи padding та border. Вона має два основні значення:

- content-box (за замовчуванням): ширина та висота, які ви задаєте елементу, застосовуються лише до області контенту. padding та border додаються до цієї ширини/висоти. Тобто, якщо ви задали width: 100px; padding: 10px; border: 1px solid; то фактична ширина елемента буде 100px + 10px (лівий padding) + 10px (правий padding) + 1px (лівий border) + 1px (правий border) = 122px. Це часто призводить до несподіваних результатів під час верстки.
- border-box: ширина та висота, які ви задаєте елементу, містять padding та border. Тобто, якщо ви задали width: 100px; padding: 10px; border: 1px solid; то фактична ширина елемента залишиться 100px. padding та border будуть «втиснуті» всередину цієї ширини, зменшуючи доступний простір для контенту. Це значно спрощує розрахунки розмірів елементів та верстку, особливо у випадку роботи з сітками та адаптивним дизайном.

```

/* Рекомендована практика для всіх елементів */
html{
 box-sizing: border-box;
}
,::before,*::after{
 box-sizing: inherit;
}

```

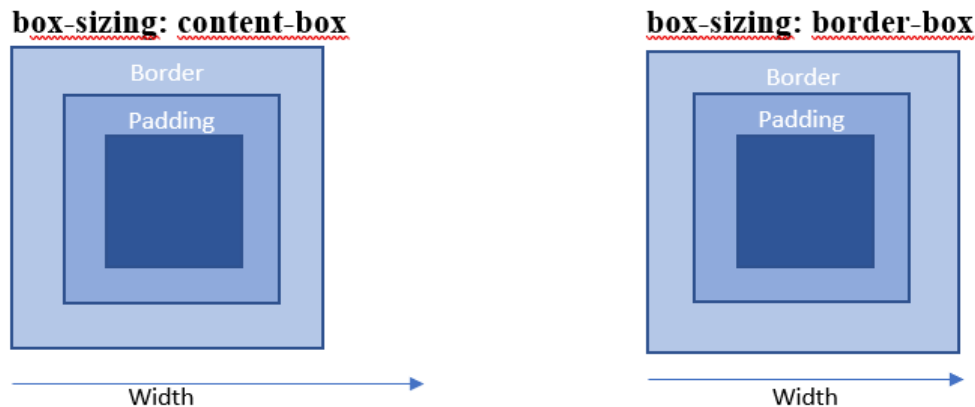


Рисунок 5.3 – Порівняння `box-sizing: content-box` та `box-sizing: border-box`

### Згорання зовнішніх відступів (Margin Collapsing)

Однією з особливостей `margin` є явище згорання зовнішніх відступів (`margin collapsing`). Це відбувається, коли вертикальні зовнішні відступи двох або більше блокових елементів, що йдуть один за одним, зливаються в один. Розмір цього згорнутого відступу дорівнює найбільшому з відступів. Це може відбуватися:

- між сусідніми елементами (верхній `margin-bottom` одного елемента та нижній `margin-top` наступного);
- між батьківським елементом та його першим/останнім дочірнім елементом (якщо немає `padding`, `border` або `min-height` між ними);
- між порожніми блоками.

Згорання відступів може бути несподіваним для новачків, але це стандартна поведінка CSS. Розуміння цього явища дозволяє уникнути небажаних відступів та ефективно керувати вертикальним інтервалом між елементами. Згорання відбувається лише для вертикальних відступів, горизонтальні відступи ніколи не згортаються.

Розуміння блокової моделі та пов'язаних з нею властивостей є ключовим для створення точних, адаптивних та візуально привабливих

вебмакетів. Як Frontend-розробники, ви будете постійно використовувати ці знання для позиціонування елементів, створення відступів та забезпечення правильного відображення контенту на будь-якому пристрої.

### **Завдання для демонстрації**

1. Box Model. Створити HTML-сторінку з кількома div елементами. Застосувати до них різні значення width, height, padding, border, margin. Використовуючи інструменти розробника в браузері (F12), продемонструвати блокову модель для кожного елемента, показуючи, як змінюються розміри контенту, padding, border та margin.

2. Box-sizing. Створити два div елементи з однаковими width, padding та border. До одного застосувати box-sizing: content-box;, до іншого — box-sizing: border-box;. Продемонструвати різницю у фактичних розмірах елементів за допомогою інструментів розробника.

3. Margin Collapsing. Створити два p елементи, що йдуть один за одним, з різними margin-bottom та margin-top. Продемонструвати, як їхні відступи згортаються. Потім додати border або padding між ними, щоб показати, як це запобігає згортанню.

### **Питання для самоконтролю**

1. Назвіть чотири складові блокової моделі CSS.
2. У чому різниця між padding та margin?
3. Які властивості використовуються для визначення рамки елемента?
4. Поясніть, як властивість box-sizing: border-box; змінює розрахунок розмірів елемента порівняно з content-box.
5. Що таке згортання зовнішніх відступів (Margin Collapsing) і коли воно відбувається?
6. Як можна центрувати блоковий елемент по горизонталі за допомогою margin?

### **Рекомендоване практичне завдання**

1. Створення картки товару: Створіть HTML-сторінку з кількома картками товару. Кожна картка повинна бути div елементом і містити зображення, заголовок, опис та кнопку. Застосуйте до карток стилі, використовуючи padding, border, margin для створення візуальних відступів та рамок. Використайте box-sizing: border-box; для всіх елементів, щоб спростити розрахунки розмірів.

2. Верстка секції з відгуками: створіть секцію з 3–4 відгуками користувачів. Кожен відгук має бути окремим блоком. Використайте margin для створення відстані між відгуками та padding для внутрішніх відступів у кожному відгуку. Експериментуйте з різними значеннями margin та padding, щоб досягти бажаного візуального ефекту.

## 5.3 Текст та шрифти

### Вибір шрифтів: Властивість `font-family`

Текст є основним носієм інформації на більшості вебсайтів, і його ефективна стилізація є ключовим аспектом вебдизайну. Типографіка в CSS дозволяє нам контролювати кожен аспект відображення тексту: від вибору шрифту та його розміру до міжрядкових інтервалів та вирівнювання. Правильна типографіка не лише робить контент візуально привабливим, але й значно покращує читабельність, розбірливість та загальний досвід користувача. Для Frontend-розробника глибоке розуміння CSS-властивостей для роботи з текстом та шрифтами є абсолютно необхідним. Це дозволяє точно відтворювати дизайнерські макети, забезпечувати адаптивність тексту до різних розмірів екранів, оптимізувати завантаження шрифтів та створювати доступні інтерфейси, де текст легко сприймається всіма користувачами.

Властивість `font-family` використовується для визначення сімейства шрифтів, яке буде застосовано до тексту. Це одна з найважливіших властивостей, що впливає на візуальне сприйняття та настрої сайту. Значення `font-family` може містити список назв шрифтів, розділених комами. Браузер спробує завантажити перший шрифт зі списку, якщо він недоступний, перейде до наступного, і так далі. Це називається `fallback-шрифтами` і є критично важливим для забезпечення консистентності відображення.

```
body{
font-family: 'Open Sans', Arial, sans-serif;
}
```

У цьому прикладі браузер спочатку спробує використати шрифт 'Open Sans'. Якщо він не знайдений або не завантажений, він перейде до Arial. Якщо і Arial недоступний, буде використано будь-який доступний шрифт без засічок (`sans-serif`) з системи користувача. Завжди рекомендується завершувати список загальним сімейством шрифтів (`generic family`) – `serif`, `sans-serif`, `monospace`, `cursive`, `fantasy` – щоб забезпечити хоча б базове відображення.

### Розмір шрифту `font-size` (px, em, rem, %)

Властивість `font-size` визначає розмір тексту. Правильний вибір розміру шрифту є ключовим для читабельності та візуальної ієрархії. Існує кілька одиниць виміру для `font-size`:

- `px` (пікселі): Абсолютна одиниця.  $1\text{px} = 1/96$  дюйма. Забезпечує точний контроль, але може створювати проблеми з доступністю, якщо користувач намагається змінити розмір шрифту в налаштуваннях браузера. Не рекомендується для основного тексту.

- **em** (відносно розміру шрифту батьківського елемента): 1em дорівнює поточному розміру шрифту батьківського елемента. Якщо батьківський елемент має font-size: 16px;, то 1.5em буде 24px. Зручно для масштабування тексту відносно батьківського контейнера.

- **rem** (відносно розміру шрифту кореневого елемента <html>): 1rem дорівнює розміру шрифту, встановленому для кореневого елемента <html>. Це найбільш рекомендована одиниця для адаптивного дизайну, оскільки дозволяє легко масштабувати весь текст на сторінці, змінюючи лише один font-size для <html>.

- **%** (відсотки): Відносно розміру шрифту батьківського елемента.

- **vw** (viewport width), **vh** (viewport height): Відносно ширини або висоти області перегляду. 1vw = 1% від ширини viewport. Корисно для адаптивної типографіки, коли розмір шрифту має змінюватися разом з розміром екрана.

```
html {
font-size: 16px; /* Базовий розмір для rem */
}
body{
font-size: 1rem; /* 16px */
}
h1{
font-size: 2.5rem; /* 40px */
}
.responsive-text{
font-size: 3vw; /* 3% від ширини viewport */
}
```

## Стилі шрифтів: font-weight, font-style, line-height, letter-spacing

- **font-weight**: Визначає товщину шрифту (жирність). Може приймати ключові слова (normal, bold) або числові значення від 100 до 900 (з кроком 100). 400 зазвичай відповідає normal, 700 – bold.

- **font-style**: Визначає стиль шрифту. normal, italic (курсив), oblique (похилий).

- **text-align**: Вирівнювання тексту всередині блока. left, right, center, justify (вирівнювання по ширині).

- **line-height**: Міжрядковий інтервал. Зазвичай задається без одиниць виміру (наприклад, 1.5), що робить його відносним до font-size елемента, або в em/rem. Оптимальне значення для читабельності 1.4-1.6.

- **letter-spacing**: Інтервал між літерами. Може бути позитивним (розширює) або негативним (стискає).

- **word-spacing**: Інтервал між словами.

- **text-decoration**: Додає або видаляє декоративні лінії до тексту. none (для видалення підкреслення у посилань), underline, overline, line-through.

- text-transform: Перетворює регістр тексту. none, uppercase, lowercase, capitalize.
- color: Колір тексту. Може бути заданий за назвою, hex-кодом, RGB або HSL.

Приклад коду :

```
body {
 font-family: Arial, sans-serif;
 margin: 40px;
 background: #f9f9f9;
 color: #333;
}

h1 {
 font-family: 'Georgia', serif;
 text-align: center;
 font-size: 2.5rem;
 font-weight: 700;
 line-height: 1.2;
 color: #2c3e50;
}

h2 {
 font-family: 'Verdana', sans-serif;
 font-size: 24px; /* пікселі */
 font-weight: 600;
 text-align: left;
 margin-top: 40px;
 color: #34495e;
}

p.example1 {
 font-family: 'Times New Roman', serif;
 font-size: 16px;
 line-height: 1.5; /* множник */
 text-align: justify;
}

p.example2 {
 font-family: 'Courier New', monospace;
 font-size: 1.2em; /* відносно батьківського
елемента */
 line-height: 1.8;
 text-align: left;
}

p.example3 {
```

```

 font-family: 'Roboto', sans-serif;
 font-size: 1rem; /* відносно кореневого
(html) */
 font-weight: 300;
 line-height: 2;
 text-align: right;
 }
 p.example4 {
 font-family: 'Tahoma', sans-serif;
 font-size: 20px;
 font-weight: 900;
 text-align: center;
 line-height: 1.1;
 color: #e74c3c;
 }
 .block {
 background: #fff;
 border: 1px solid #ccc;
 border-radius: 10px;
 padding: 20px;
 margin-bottom: 30px;
 box-shadow: 0 2px 4px rgba(0,0,0,0.1);
 }
</style>
</head>
<body>
 <h1>Демонстрація текстових властивостей у
CSS</h1>
 <div class="block">
 <h2>font-family, font-size у px, line-
height</h2>
 <p class="example1">
 Цей абзац використовує шрифт Times New
Roman, розмір 16px і висоту рядка
1.5.
 Текст вирівняно по ширині, тому відстань
між словами може змінюватися.
 </p>
 </div>
 <div class="block">
 <h2>font-family, font-size у em, line-
height</h2>
 <p class="example2">

```

Тут використано моноширинний шрифт **Courier New** і розмір **1.2em**.

Елементи **em** масштабуються відносно шрифту батьківського елемента.

```
</p>
</div>
<div class="block">
 <h2>font-family, font-size у rem, text-align:
right</h2>
```

```
 <p class="example3">
 Цей текст застосовує шрифт Roboto,
 розмір 1rem (відносно кореневого елемента)
 та вирівнювання праворуч. Також
 зменшено товщину шрифту (font-weight: 300) і
 збільшено міжрядковий інтервал.
```

```
 </p>
</div>
```

```
<div class="block">
 <h2>font-weight, text-align: center</h2>
 <p class="example4">
 Це яскравий приклад: шрифт Tahoma,
 дуже жирний (900),
 вирівнювання по центру та короткий
 міжрядковий інтервал (1.1).
```

```
 </p>
</div>
</body>
```

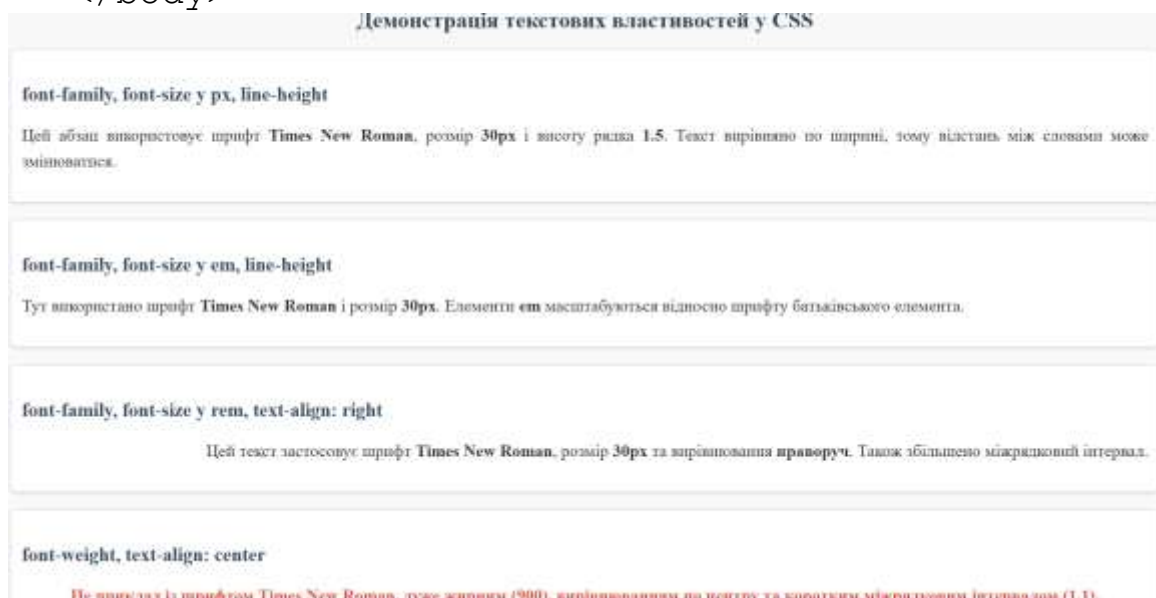


Рисунок 5.4 – Приклад HTML-коду з різними текстовими елементами

## Підключення власних шрифтів: @font-face

Хоча системні шрифти (Arial, Times New Roman) є універсальними, вони часто не відповідають естетичним вимогам дизайну. Для використання унікальних шрифтів, які не встановлені на комп'ютері користувача, ми використовуємо правило @font-face.

Правило @font-face дозволяє завантажувати шрифти з сервера та застосовувати їх до вебсторінки. Це дає дизайнерам повну свободу у виборі типографіки.

```
@font-face{
font-family: 'MyCustomFont'; /* Назва, яку ми будемо
використовувати в CSS */
src: url('fonts/MyCustomFont.woff2') format('woff2'), /* Шлях
до файлу шрифту */
url('fonts/MyCustomFont.woff') format('woff');
font-weight:normal;
font-style:normal;
font-display: swap; /* Оптимізація завантаження */
}

body{
font-family:'MyCustomFont', sans-serif;
}
```

- font-family: Назва, яку ви даєте своєму шрифту. Саме її ви будете використовувати у властивості font-family.

- src: Вказує шлях до файлів шрифту. Важливо надавати кілька форматів (woff2, woff, ttf, eot, svg) для кросбраузерної сумісності. woff2 є найбільш сучасним та ефективним.

- font-weight, font-style: Вказують, для якого накреслення та стилю шрифту призначений цей файл. Це дозволяє браузеру правильно підставляти файли шрифтів для font-weight: bold; або font-style: italic;.

- font-display: Важлива властивість для оптимізації завантаження шрифтів. swap дозволяє браузеру спочатку відобразити текст системним шрифтом, а потім, після завантаження кастомного шрифту, замінити його. Це запобігає «невидимому тексту» (FOIT - Flash Of Invisible Text) та покращує сприйняття швидкості завантаження.

## Google Fonts: підключення та використання

Google Fonts – це безкоштовна бібліотека вебшрифтів, яка значно спрощує процес підключення власних шрифтів. Замість того, щоб вручну завантажувати файли та писати @font-face правила, Google Fonts надає простий спосіб підключення шрифтів через <link> або @import.

```
<!--У розділі <head>-->
<link rel="preconnect"
href="https://fonts.googleapis.com"> <link rel="preconnect"
href="https://fonts.gstatic.com"
```

```

crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">
/* Після підключення в HTML */
body {
 font-family: 'Roboto', sans-serif;
}

```

Google Fonts автоматично генерує @font-face правила та надає оптимізовані файли шрифтів, що значно спрощує роботу розробника. Важливо вибирати лише ті накреслення (wght@400;700), які ви дійсно використовуєте, щоб не перевантажувати сторінку зайвими файлами шрифтів.

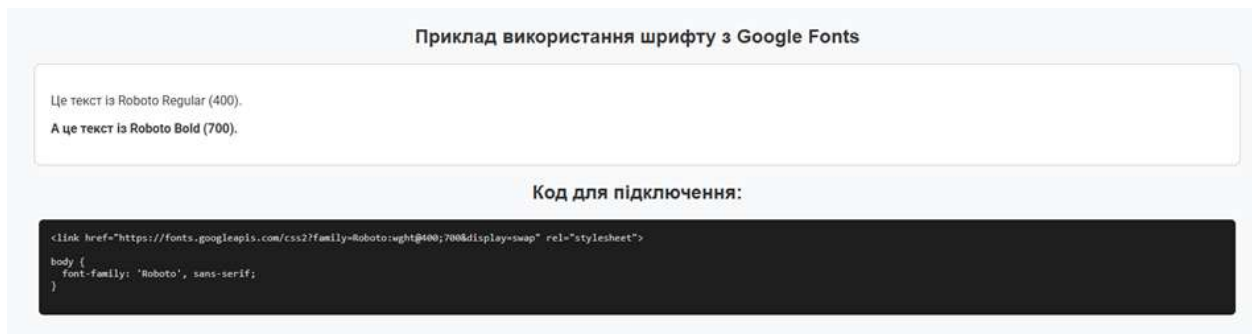


Рисунок 5.5 – Скріншот інтерфейсу Google Fonts, що демонструє вибір шрифту та генерацію коду для підключення

Правильна робота з текстом та шрифтами є ключовим елементом створення якісного вебінтерфейсу. Вона впливає на читабельність, естетику, доступність та продуктивність. Як Frontend-розробники, ви будете постійно застосовувати ці знання, тому глибоке розуміння CSS-властивостей для типографіки та методів підключення шрифтів є фундаментальним для вашої професійної діяльності.

## Завдання для демонстрації

1. Базові властивості тексту. Створити HTML-сторінку з кількома параграфами та заголовками. Застосувати до них різні font-family (системні шрифти), font-size (px, em, rem), font-weight, font-style, text-align, line-height, letter-spacing, text-decoration, text-transform. Продемонструвати, як ці властивості змінюють вигляд тексту.

2. Підключення Google Fonts:

- Підключити 2-3 шрифти з Google Fonts до HTML-сторінки.
- Застосувати їх до різних елементів (наприклад, один для заголовків, інший для основного тексту).
- Продемонструвати, як font-display: swap; впливає на завантаження.

3. `@font-face`: Створити папку `fonts`, завантажити туди якийсь безкоштовний шрифт (наприклад, з Font Squirrel). Підключити його за допомогою `@font-face` та застосувати до елементів на сторінці. Продемонструвати, як працюють fallback-шрифти, якщо кастомний шрифт не завантажився.

### Питання для самоконтролю

1. Яка основна функція властивості `font-family`? Чому важливо вказувати fallback-шрифти?
2. Назвіть та поясніть різні одиниці виміру для `font-size`. Які з них рекомендовані для адаптивного дизайну і чому?
3. Які CSS-властивості використовуються для керування товщиною та стилем шрифту?
4. Що таке `line-height` і як він впливає на читабельність тексту?
5. Поясніть призначення правила `@font-face` та його основні властивості.
6. Як Google Fonts спрощує роботу з підключенням шрифтів?

### Рекомендоване практичне завдання

1. Створення стилізованої статті. Візьміть текст будь-якої статті (наприклад, з Вікіпедії) та зверстайте її у HTML. Застосуйте до неї CSS-стилі, використовуючи:
  - Два шрифти з Google Fonts (один для заголовків, інший для основного тексту).
  - Оптимальні `font-size` (з використанням `rem`), `line-height`, `text-align` для читабельності.
  - `font-weight` та `font-style` для виділення важливих частин тексту.
  - `letter-spacing` або `word-spacing` для експериментів.
2. Зробіть так, щоб стаття виглядала професійно та була легкою для читання.
3. Експерименти з `@font-face`: Знайдіть та завантажте 2-3 безкоштовні шрифти з різних джерел (наприклад, Font Squirrel, DaFont). Підключіть їх до HTML-сторінки за допомогою правила `@font-face`, забезпечивши підтримку різних форматів (`woff2`, `woff`). Застосуйте ці шрифти до різних елементів на сторінці. Продемонструйте, як `font-display: swap`; впливає на відображення тексту під час завантаження шрифтів.

## 5.4 Позиціонування елементів

### Властивість display: Тип відображення елементів

Керування розташуванням елементів на вебсторінці є одним з найскладніших, але й найважливіших аспектів CSS. Без ефективного позиціонування неможливо створити візуально привабливий та функціональний макет. Ця лекція присвячена двом фундаментальним CSS-властивостям, які визначають, як елементи відображаються та де вони розташовуються: `display` та `position`. Розуміння цих властивостей є ключовим для будь-якого Frontend-розробника, оскільки вони є основою для побудови будь-якого макета, від простих текстових блоків до складних адаптивних інтерфейсів. Правильне використання `display` та `position` дозволяє точно відтворювати дизайнерські макети, забезпечувати кросбраузерну сумісність та створювати динамічні елементи, які поведуться передбачувано.

Властивість `display` визначає, як елемент відображається на сторінці, тобто який тип блока він генерує. Це фундаментальна властивість, яка впливає на поведінку елемента відносно інших елементів та на його здатність приймати властивості ширини, висоти, відступів тощо. Існує багато значень `display`, але найпоширенішими є:

1. `block` (блоковий елемент): Елементи з `display: block`; завжди починаються з нового рядка і займають всю доступну ширину батьківського контейнера (якщо не задано інше). Вони можуть мати задані `width`, `height`, `margin`, `padding`. Приклади блокових елементів за замовчуванням: `<div>`, `<p>`, `<h1>`-`<h6>`, `<ul>`, `<li>`, `<section>`, `<article>`. Блокові елементи є основою для вертикального розташування контенту.

2. `inline` (рядковий елемент): Елементи з `display: inline`; не починаються з нового рядка і займають лише стільки ширини, скільки потрібно для їхнього вмісту. Вони не можуть мати задані `width`, `height`, а вертикальні `margin` та `padding` ігноруються (горизонтальні застосовуються). Приклади рядкових елементів за замовчуванням: `<span>`, `<a>`, `<strong>`, `<em>`, `<img>`. Рядкові елементи використовуються для форматування тексту всередині рядка.

3. `inline-block` (рядково-блоковий елемент): Це гібридний тип. Елементи з `display: inline-block`; поведуться як рядкові елементи (не починаються з нового рядка і розташовуються поруч), але водночас можуть мати задані `width`, `height`, `margin`, `padding` (як блокові елементи). Це дуже зручно для створення горизонтальних меню, галерей зображень або елементів, які мають розташовуватися поруч, але мати контрольовані розміри. Однак, між `inline-block` елементами можуть з'являтися небажані пробіли через пробіли в HTML-коді.

4. none: Елемент повністю видаляється з потоку документа. Він не відображається і не займає місця на сторінці. Часто використовується для приховування/відображення елементів за допомогою JavaScript.

```
/*Приклади використання display*/
.block-element{
display:block;
width:100px;
height:100px;
background-color:lightblue;
margin:10px;
}
.inline-element{
display:inline;
background-color:lightgreen;
padding:5px;
}
.inline-block-element{
isplay:inline-block;
width:100px;
height:50px;
background-color:lightcoral;
margin:5px;
}
.hidden-element{
display:none;
}
```

### **Властивість position: точне позиціонування**

Властивість position дозволяє нам витягувати елементи з нормального потоку документа та розміщувати їх у довільному місці на сторінці. Вона працює в поєднанні з властивостями top, right, bottom, left.

- static (за замовчуванням): елемент знаходиться в нормальному потоці документа. Властивості top, right, bottom, left не мають жодного ефекту. Це значення за замовчуванням для всіх HTML-елементів.

- relative (відносне позиціонування): елемент залишається в нормальному потоці документа, але його можна змістити відносно його початкового положення за допомогою top, right, bottom, left. Інші елементи поведуться так, ніби елемент не був зміщений. relative часто використовується як контейнер для абсолютно позиціонованих дочірніх елементів.

```
css
.relative-box {
position: relative;
top: 20px; / Змістити на 20px вниз від
початкового положення /
```

```
 left: 30px; / Змістити на 30px вправо від
початкового положення /
 background-color: yellow;
}
```

● **absolute** (абсолютне позиціонування): елемент повністю витягується з нормального потоку документа. Його позиція визначається відносно найближчого позиціонованого предка (тобто предка, який має `position: relative, absolute, fixed` або `sticky`). Якщо такого предка немає, елемент позиціонується відносно `<body>`. Абсолютно позиціоновані елементи не займають місця в потоці документа, й інші елементи поведуться так, ніби їх не існує. Це дуже потужний інструмент для створення висхідних вікон, підказок, іконок, що накладаються на зображення.

```
css
.parent-relative {
position: relative;
width: 300px;
height: 200px;
border: 1px solid black;
}
.child-absolute {
position: absolute;
top: 10px;
right: 10px;
width: 50px;
height: 50px;
background-color: red;
}
```

● **fixed** (фіксоване позиціонування): елемент повністю витягується з нормального потоку документа і позиціонується відносно вікна браузера (viewport). Він залишається на одному місці, навіть під час прокручування сторінки. Часто використовується для фіксованих шапок, футерів, кнопок «нагору».

```
css
.fixed-header {
position: fixed;
top: 0;
left: 0;
width: 100%;
height: 60px;
```

```
background-color: #333;
color: white;
}
```

- **sticky** (липке позиціонування): це гібрид **relative** та **fixed**. Елемент поводить ся як **relative**, доки не досягне певної точки прокручування (визначеної **top**, **right**, **bottom**, **left**), після чого він стає **fixed**. Часто використовується для навігаційних меню, бічних панелей, які «прилипають» до верхньої частини екрана під час прокручування. Потребує батьківського контейнера з прокручуванням.

```
css
.sticky-sidebar {
position: sticky;
top: 0; / Прилипне до верху, коли досягне 0px від
верху viewport /
background-color: lightgray;
padding: 10px;
}
```

## **z-index: Керування шарами**

Коли елементи позиціонуються за допомогою **position** (крім **static**), вони можуть накладатися один на одного. Властивість **z-index** визначає порядок накладання елементів по осі **Z** (глибина). Елемент з більшим **z-index** буде відображатися поверх елементів з меншим **z-index**. **z-index** працює лише для позиціонованих елементів (**position: relative, absolute, fixed, sticky**).

```
.box-a{
position:absolute;
z-index:1;
background-color:red;
}

.box-b{
position:absolute;
z-index:2;
background-color:blue;/*Буде поверх box-a */
}
```

Приклад коду:

```
<!DOCTYPE html>
<html lang="uk">
<head>
```

```

<meta charset="UTF-8">
<meta name="viewport" content="width=device-
width, initial-scale=1.0">
<title>Position Demo</title>
<style>
 body {
 font-family: "Times New Roman", serif;
 font-size: 40px;
 margin: 20px;
 height: 2000px;
 }
 h1 {
 text-align: center;
 margin-bottom: 30px;
 font-size: 60px;
 }
 .box {
 width: 200px;
 height: 120px;
 color: #fff;
 text-align: center;
 line-height: 120px;
 margin: 40px 20px;
 border-radius: 12px;
 font-size: 40px;
 }
 .static {
 background: #3498db;
 position: static;
 }
 .relative {
 background: #27ae60;
 position: relative;
 top: 10px;
 left: 10px;
 margin-bottom: 80px;
 }
 .parent {
 background: #ddd;
 position: relative;
 padding: 40px;
 margin: 80px 0;
 font-size: 40px;
 height: 250px;
 }

```

```

 }
 .absolute {
 background: #e67e22;
 position: absolute;
 top: 40px;
 right: 40px;
 width: 200px;
 height: 120px;
 line-height: 120px;
 font-size: 40px;
 text-align: center;
 }
 .fixed {
 background: #9b59b6;
 position: fixed;
 bottom: 20px;
 right: 20px;
 width: 200px;
 height: 120px;
 line-height: 120px;
 border-radius: 12px;
 box-shadow: 0 0 10px rgba(0,0,0,0.3);
 font-size: 40px;
 text-align: center;
 }

 .sticky {
 background: #c0392b;
 position: sticky;
 top: 0;
 padding: 20px;
 text-align: center;
 font-weight: bold;
 color: #fff;
 font-size: 40px;
 margin-bottom: 40px;
 }
</style>
</head>
<body>
<h1>Position Demo</h1>
<div class="sticky">sticky (залишається зверху
при прокрутці)</div>
<div class="box static">static</div>

```

```

<div class="box relative">relative</div>
<div class="parent">
 Батьківський блок (relative)
 <div class="absolute">absolute</div>
</div>
<div class="fixed">fixed</div>
<p>Прокрути сторінку вниз, щоб побачити поведінку
sticky та fixed елементів.</p>
</body>
</html>

```

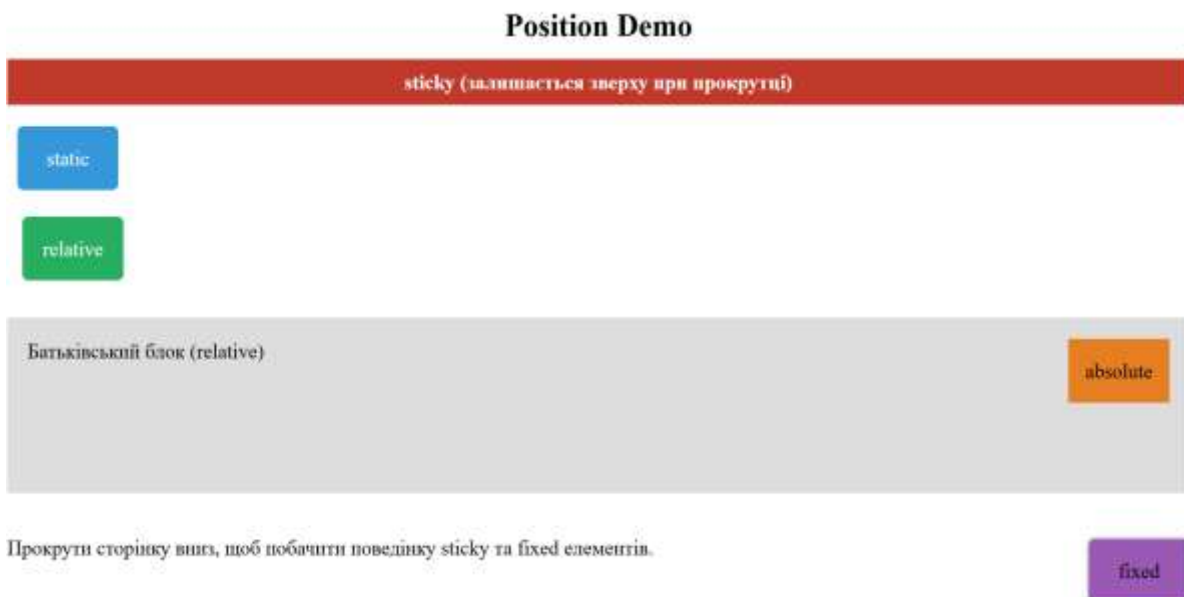


Рисунок 5.6 – Приклад, що демонструє різні значення position

## Потік документа (Document Flow)

Потік документа – це спосіб, яким браузер розташовує HTML-елементи на сторінці за замовчуванням. Блокові елементи розташовуються вертикально, один під одним, займаючи всю доступну ширину. Рядкові елементи розташовуються горизонтально, один за одним, займаючи лише необхідну ширину. Коли елемент позиціонується за допомогою position: absolute; або position: fixed;, він витягується з нормального потоку документа, і інші елементи поведуться так, ніби його не існує. Це важливо враховувати під час проєктування макетів, оскільки витягнуті елементи можуть перекривати інший контент або залишати порожнє місце.

## Контекст накладання (Stacking Context)

Контекст накладання – це тривимірна концепція, яка визначає, як елементи HTML розташовуються по осі Z (глибина) відносно один одного. Кожен елемент, який має position (крім static) та z-index (або інші

властивості, що створюють контекст накладання, наприклад, opacity, transform), створює новий контекст накладання. Елементи всередині одного контексту накладання розташовуються відповідно до їхнього z-index. Однак, z-index елемента діє лише в межах його власного контексту накладання. Це означає, що елемент з z-index: 1000; всередині батьківського елемента з z-index: 1; завжди буде під елементом з z-index: 2;, якщо останній знаходиться в іншому контексті накладання. Розуміння контекстів накладання є важливим для вирішення проблем з перекриттям елементів, особливо у складних інтерфейсах.

Приклад коду :

```
<!DOCTYPE html>
<html lang="uk">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-
width, initial-scale=1.0">
<title>Контекст накладання (z-index)</title>
<style>
 body {
 font-family: Arial, sans-serif;
 background: #f6f7fb;
 padding: 40px;
 }
 .parent {
 position: relative;
 width: 200px;
 height: 200px;
 border-radius: 10px;
 color: #fff;
 text-align: center;
 line-height: 200px;
 font-weight: bold;
 }
 .parent1 {
 background: #3498db;
 z-index: 1;
 }
 .parent2 {
 background: #e67e22;
 margin-top: -100px; /* перекриття */
 margin-left: 100px;
 z-index: 2;
 }
}
```

```

 .child {
 position: absolute;
 width: 100px;
 height: 100px;
 top: 50px;
 left: 50px;
 border-radius: 8px;
 line-height: 100px;
 }
 .child1 {
 background: rgba(231, 76, 60, 0.9); /*
червоний */
 z-index: 3;
 }
 .child2 {
 background: rgba(46, 204, 113, 0.9); /*
зелений */
 z-index: 1;
 }
 </style>
</head>
<body>
<h2>Контекст накладання (stacking context)</h2>
<p>Навіть якщо червоний блок (.child1) має
більший z-index (3), він не може перекрити
помаранчевий батьківський елемент (.parent2), бо його
батько (.parent1) має нижчий z-index (1).</p>
<div class="parent parent1">
 Батько 1 (z-index: 1)
 <div class="child child1">Дитина 1
(z-index:
3)</div>
</div>
<div class="parent parent2">
 Батько 2 (z-index: 2)
 <div class="child child2">Дитина 2
(z-index:
1)</div>
</div>
</body>
</html>

```

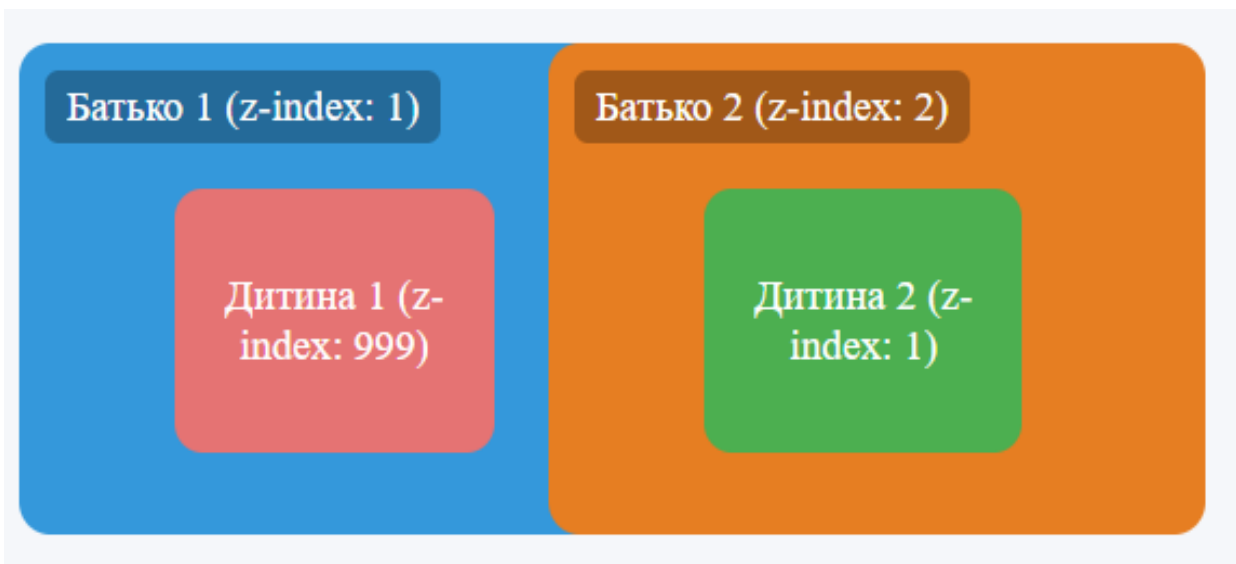


Рисунок 5.7 – Приклад, що демонструє контекст накладання

Ефективне використання властивостей `display` та `position` є ключовим для створення складних та адаптивних вебмакетів. Це дозволяє Frontend-розробнику точно контролювати розташування кожного елемента на сторінці, реалізовувати інтерактивні компоненти та забезпечувати візуальну відповідність дизайнерським макетам. Глибоке розуміння цих концепцій є фундаментальним для вашої професійної діяльності у веброботі.

### Завдання для демонстрації

1. `display`: Створити HTML-сторінку з кількома `div`, `span` та а елементами. Застосувати до них різні значення `display` (`block`, `inline`, `inline-block`). Продемонструвати, як це змінює їхню поведінку щодо ширини, висоти та розташування.
2. `position: relative`: Створити `div` з `position: relative` та змістити його за допомогою `top` та `left`. Показати, що інші елементи не змінюють свого положення.
3. `position: absolute`: Створити батьківський `div` з `position: relative`. Всередині нього створити дочірній `div` з `position: absolute` та позиціонувати його за допомогою `top` та `right`. Продемонструвати, як він витягується з потоку та позиціонується відносно батька.
4. `position: fixed`: Створити фіксований заголовок або кнопку «нагору» за допомогою `position: fixed`. Продемонструвати, що він залишається на місці під час прокручування.
5. `position: sticky`: Створити бічну панель або навігаційне меню з `position: sticky`. Продемонструвати, як воно «прилипає» до верху екрана під час прокручування.

6. z-index: Створити два абсолютно позиціоновані елементи, що перекриваються. Продемонструвати, як z-index змінює порядок їхнього накладання.

### **Питання для самоконтролю**

1. Поясніть різницю між display: block, display: inline та display: inline-block.
2. Які значення може приймати властивість position? Опишіть кожне з них.
3. У чому різниця між position: relative та position: absolute?
4. Коли доцільно використовувати position: fixed?
5. Що таке z-index і для чого він використовується?
6. Поясніть концепцію потоку документа та як позиціонування впливає на нього.

### **Рекомендоване практичне завдання**

1. Створення макета з display: Створіть HTML-сторінку з кількома блоками. Використайте display: inline-block; для створення горизонтального розташування елементів (наприклад, галерея зображень або горизонтальне меню). Експериментуйте з width, height, margin, padding для цих елементів.

2. Верстка висхідного вікна (модального вікна): Створіть HTML-сторінку з кнопкою. Після натискання на кнопку має з'являтися висхідне вікно (модальне вікно), яке перекриває весь контент сторінки. Використайте position: fixed; для фонового налагодження та position: absolute; (або fixed) для самого модального вікна. Забезпечте центрування модального вікна та можливість його закриття.

3. Фіксована шапка та липка бічна панель: Створіть макет сторінки з фіксованою шапкою (position: fixed;) та довгою бічною панеллю, яка «прилипає» до верху екрана під час прокручування (position: sticky;). Переконайтеся, що основний контент сторінки не перекривається фіксованими елементами.

## **5.5 Flexbox Layout**

### **Основні концепції Flexbox: Контейнер та елементи**

У сучасному вебдизайні створення гнучких та адаптивних макетів є необхідністю. Класичні методи позиціонування, як float або inline-block, були складними для підтримки, але CSS надає потужні інструменти, зокрема Flexbox (Flexible Box Layout). Flexbox – одновимірна система, що дозволяє розташовувати елементи по горизонталі або вертикалі, гнучко

розподіляти простір, змінювати розмір і порядок елементів. Це спрощує вирівнювання та створення адаптивних компонентів. Для Frontend-розробника Flexbox є ключовим інструментом, який дозволяє швидко створювати зручні й гармонійні інтерфейси, що підлаштовуються під різні екрани.

Flexbox працює за принципом «батько-діти». Для того, щоб HTML-елементи стали «гнучкими» та почали підкорятися правилам Flexbox, їхній батьківський контейнер має бути оголошений як flex-контейнер (Flex Container). Безпосередні дочірні елементи цього контейнера автоматично стають flex-елементами (Flex Items).

- Flex-контейнер. Елемент, до якого застосовано `display: flex` або `display: inline-flex`. Всі властивості Flexbox застосовуються саме до цього контейнера, впливаючи на його дочірні flex-елементи.

- Flex-елементи: Безпосередні дочірні елементи flex-контейнера. До них також можна застосовувати деякі властивості Flexbox, що впливають на їх поведінку всередині контейнера.

Flexbox оперує двома основними осями:

- Головна вісь (Main Axis): це основна вісь, вздовж якої розташовуються flex-елементи. Її напрямок визначається властивістю `flex-direction`.

- Поперечна вісь (Cross Axis): це вісь, перпендикулярна до головної осі. Вона використовується для вирівнювання елементів поперек головної осі.

Розуміння цих осей є ключовим, оскільки більшість властивостей Flexbox посиляються саме на них.

## Властивості Flex-контейнера

Властивості, що застосовуються до контейнера, керують розташуванням та поведінкою flex-елементів всередині нього.

- `display: flex` | `display: inline-flex`: оголошує елемент flex-контейнером. `flex` робить контейнер блоковим, а `inline-flex` – рядковим.

- `flex-direction: row` | `flex-direction: row-reverse` | `flex-direction: column` | `flex-direction: column-reverse`: визначає напрямок головної осі.

- `flex-direction: row` (за замовчуванням): головна вісь горизонтальна, елементи йдуть зліва направо.

- `flex-direction: row-reverse`: головна вісь горизонтальна, елементи йдуть справа наліво.

- `flex-direction: column`: головна вісь вертикальна, елементи йдуть зверху вниз.

- `flex-direction: column-reverse`: головна вісь вертикальна, елементи йдуть знизу вгору.

- `flex-wrap: nowrap | wrap | wrap-reverse`: визначає, чи будуть flex-елементи переноситися на новий рядок (якщо не вміщаються) чи залишатимуться в одному рядку.
- `nowrap` (за замовчуванням): елементи залишаються в одному рядку, можливо, стискаючись.
- `wrap`: елементи переносяться на новий рядок, якщо не вистачає місця.
- `wrap-reverse`: елементи переносяться на новий рядок, але в зворотному порядку.
- `flex-flow`: скорочений запис для `flex-direction` та `flex-wrap` (наприклад, `flex-flow: row wrap`);).
- `justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly`: вирівнює flex-елементи вздовж головної осі. Керує розподілом вільного простору.
- `flex-start` (за замовчуванням): елементи притискаються до початку головної осі.
- `flex-end`: елементи притискаються до кінця головної осі.
- `center`: елементи центруються по головній осі.
- `space-between`: елементи розподіляються рівномірно, перший елемент на початку, останній – в кінці головної осі.
- `space-around`: елементи розподіляються рівномірно, з рівномірним простором навколо кожного елемента.
- `space-evenly`: елементи розподіляються рівномірно, з рівномірним простором між кожним елементом, а також початком та кінцем контейнера.
- `align-items: flex-start | flex-end | center | baseline | stretch`: вирівнює flex-елементи вздовж поперечної осі (перпендикулярно до головної осі) в межах одного рядка.
- `stretch` (за замовчуванням): елементи розтягуються, щоб заповнити контейнер (якщо `auto-розмір`).
- `flex-start`: елементи притискаються до початку поперечної осі.
- `flex-end`: елементи притискаються до кінця поперечної осі.
- `center`: елементи центруються по поперечній осі.
- `baseline`: елементи вирівнюються по базовій лінії тексту.
- `align-content: flex-start | flex-end | center | space-between | space-around | space-evenly | stretch`: вирівнює рядки flex-елементів вздовж поперечної осі всередині flex-контейнера, що має кілька рядків (`flex-wrap: wrap`). Керує розподілом вільного простору між рядками.
- Аналогічно `justify-content` за значеннями, але працює з рядками, а не з елементами.

## Flex item properties

1. order (змінює порядок елементів)



2. flex-grow (елементи розширюються по-різному)



3. align-self (елемент вирівняний інакше)



Рисунок 5.8 – Ілюстрація, що демонструє основні властивості flex-контейнера

### Властивості Flex-елементів

Властивості, що застосовуються до безпосередніх дочірніх елементів flex-контейнера, керують їхньою індивідуальною поведінкою.

- **order:** <integer>: Змінює візуальний порядок flex-елементів. Елементи з меншим значенням order відображаються раніше. За замовчуванням 0.

```
css
.item-1 { order: 2; }
.item-2 { order: 1; } / Цей елемент
відобразиться першим /
```

- **flex-grow:** <number>: Визначає, наскільки flex-елемент може «рости» та займати доступний вільний простір у flex-контейнері. За замовчуванням 0 («не росте»).

```
css
.item-1 { flex-grow: 1; } / Займе весь
доступний простір /
.item-2 { flex-grow: 2; } / Займе вдвічі
більше місця, ніж item-1 /
```

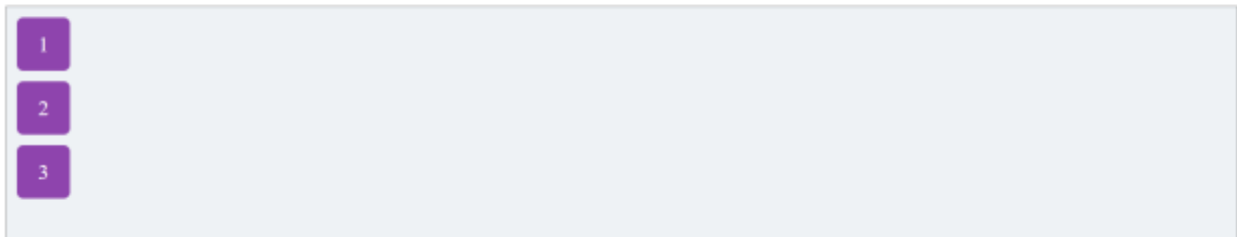
- **flex-shrink:** <number>: Визначає, наскільки flex-елемент може «стискатися», якщо доступного місця недостатньо. За замовчуванням 1 (буде стискатися). 0 означає, що елемент не буде стискатися (може спричинити переповнення).

- `flex-basis: <length> | auto`: Визначає початковий розмір flex-елемента перед тим, як буде розподілятися вільний простір. За замовчуванням `auto` (бере розмір контенту або `width/height`).
- `flex`: Скорочений запис для `flex-grow`, `flex-shrink` та `flex-basis` (наприклад, `flex: 1 1 auto`);
  - `flex: 1`; є скороченням для `flex: 1 1 0 %`;
  - `flex: auto`; є скороченням для `flex: 1 1 auto`;
  - `flex: none`; є скороченням для `flex: 0 0 auto`;
  - `align-self: flex-start | flex-end | center | baseline | stretch | auto`: Перевизначає значення `align-items` для окремого flex-елемента. `auto` означає успадкування від `align-items` батька.

1. `flex-direction: row`



2. `flex-direction: column`



3. `justify-content: space-between`

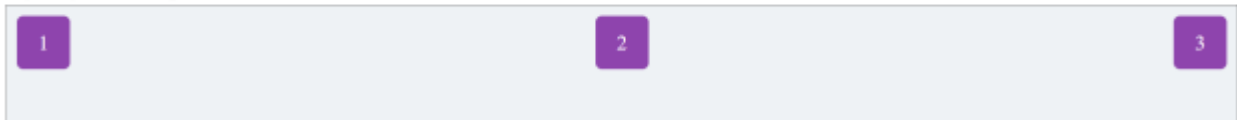


Рисунок 5.9 – Ілюстрація, що демонструє основні властивості flex-елементів

## Практичні приклади використання Flexbox

Flexbox є ідеальним для компоновання компонентів, що потребують гнучкого та динамічного розташування. Деякі типові сценарії:

- Навігаційні меню. Вертикальні або горизонтальні меню, де елементи рівномірно розподілені або центровані.
- Рівновисокі колонки. Створення колонок, які завжди мають однакову висоту, незалежно від вмісту.
- Вирівнювання елементів по центру: Вертикальне та горизонтальне центрування елемента в контейнері є надзвичайно простим з Flexbox.
- Картки товарів/послуг. Створення динамічних сіток карток, які автоматично підлаштовуються.
- Форми. Вирівнювання полів форми та кнопок.
- Футери. Елементи футера, розташовані по горизонталі.

```

<nav class="navigation">
Посилання 1
Посилання 2
Посилання 3
</nav>
<div class="card-container">
<div class="card">Карта 1</div>
<div class="card">Карта 2 з довшим текстом</div>
<div class="card">Карта 3</div>
</div>
<div class="centered-box">
<p>Я по центру!</p>
</div>
/* Приклад для навігації */
.navigation {
display: flex;
justify-content: space-around; /* Рівномірний
розподіл з
проміжками */
align-items: center; /* Вирівнювання по центру
поперечної
осі */
background-color: #f0f0f0;
padding: 10px;
}

/*Приклад для рівновисоких карток */
.card-container {
display: flex;
flex-wrap: wrap; /* Переносити на новий рядок */
gap: 20px; /* Проміжок між картками */
}

.card {
flex-grow: 1; /* Картки будуть рости, щоб зайняти
простір */
flex-basis: 200px; /* Мінімальна ширина картки */
border: 1px solid #ccc;
padding: 15px;
display: flex; /* Зробити картку flex-контейнером
для її вмісту */
flex-direction: column;
justify-content: space-between; /* Контент картки
розтягується */
}

/*Приклад для центрування */
.centered-box {

```

```

display: flex;
justify-content: center; /* Горизонтальне
центрування */
align-items: center; /* Вертикальне центрування */
height: 200px;
border: 1px solid blue;
}

```

Flexbox є потужним та гнучким інструментом, що кардинально спрощує створення адаптивних та динамічних макетів. Глибоке розуміння його концепцій та властивостей є обов'язковим для кожного Frontend-розробника, дозволяючи ефективно вирішувати складні завдання компоновання та створювати якісні, передбачувані та зручні інтерфейси.

### Завдання для демонстрації

1. Базовий Flex-контейнер. Створити div як flex-контейнер з трьома дочірніми div елементами (flex-елементами). Застосувати display: flex. Продемонструвати, як змінюється розташування елементів у разі зміни flex-direction (row, column).
2. Вирівнювання по головній осі: Експериментувати з justify-content (flex-start, flex-end, center, space-between, space-around, space-evenly) та показати, як елементи вирівнюються та розподіляється простір.
3. Вирівнювання по поперечній осі. Експериментувати з align-items (flex-start, flex-end, center, stretch) та показати, як елементи вирівнюються поперек головної осі.
4. flex-wrap. Додати більше flex-елементів до контейнера. Показати, як flex-wrap: wrap; дозволяє елементам переноситися на новий рядок.
5. Властивості flex-елементів. Застосувати order, flex-grow та align-self до окремих flex-елементів, щоб продемонструвати індивідуальну поведінку.

### Питання для самоконтролю

1. Що таке Flexbox і для чого він використовується?
2. Поясніть різницю між flex-контейнером та flex-елементом.
3. Що таке головна та поперечна осі у Flexbox і як flex-direction впливає на них?
4. Які властивості використовуються для вирівнювання елементів вздовж головної осі (justify-content) та поперечної осі (align-items)?
5. Як flex-wrap впливає на поведінку flex-елементів?
6. Які властивості flex-елемента дозволяють змінювати його порядок та розмір (flex-grow, flex-shrink, flex-basis)?

## Рекомендоване практичне завдання

1. Верстка навігаційного меню. Створіть HTML-структуру для головного меню (наприклад, `nav` з `ul` та `li`). Застосуйте Flexbox до `ul`, щоб створити горизонтальне меню, де елементи рівномірно розподілені по ширині контейнера та центровані по вертикалі. Додайте стилі для `hover` стану посилань.

2. Верстка секції з картками. Створіть секцію з 3–4 картками (кожна `div` з зображенням, заголовком та текстом). Використайте Flexbox, щоб картки розташовувалися горизонтально на широких екранах, автоматично переносилися на новий рядок на менших екранах (`flex-wrap: wrap`). Забезпечте рівні проміжки між ними (`gap` або `margin`). Переконайтеся, що всі картки мають однакову висоту, незалежно від вмісту (використайте `align-items: stretch`; або зробіть самі картки flex-контейнерами)

3. Центрування елемента. Створіть `div` елемент. Використайте Flexbox, щоб відцентрувати інший `div` (або будь-який інший блоковий елемент) як по горизонталі, так і по вертикалі всередині першого `div`.

## 5.6 CSS Grid Layout

### Основні концепції CSS Grid: Контейнер, елементи, лінії, доріжки, області

У сучасному вебдизайні створення двовимірних макетів, які добре адаптуються до різних екранів, є важливою задачею. Flexbox чудово підходить для одновимірного компонування, але для роботи з рядками і стовпцями одночасно використовується більш потужний інструмент – CSS Grid Layout. Grid дозволяє ділити сторінку на сітку з рядків і стовпців та легко розміщувати елементи в потрібних комірках. Це значно спрощує створення складних макетів без використання таблиць або складних комбінацій стилів. Для Frontend-розробника CSS Grid – незамінний інструмент для створення адаптивної та гнучкої структури сторінки.

Як і Flexbox, Grid працює за принципом «батько–діти». Для того, щоб HTML-елементи стали частиною сітки, їхній батьківський контейнер має бути оголошений як grid-контейнер (Grid Container). Безпосередні дочірні елементи цього контейнера автоматично стають grid-елементами (Grid Items).

- Grid-контейнер. Елемент, до якого застосовано `display: grid` або `display: inline-grid`. Всі властивості Grid застосовуються саме до цього контейнера, впливаючи на його дочірні grid-елементи.

- Grid-елементи. Безпосередні дочірні елементи grid-контейнера. До них також можна застосовувати деякі властивості Grid, що впливають на їхнє розташування та розмір всередині сітки.

- Grid-лінії (Grid Lines). Це горизонтальні та вертикальні лінії, які утворюють структуру сітки. Вони нумеруються, починаючи з 1.
- Grid-доріжки (Grid Tracks). Це простір між двома сусідніми grid-лініями. Доріжки можуть бути стовпцями (columns) (вертикальні доріжки) або рядками (rows) (горизонтальні доріжки).
- Grid-комірки (Grid Cells). Це найменші одиниці сітки, що утворюються на перетині рядка та стовпця.
- Grid-області (Grid Areas). Це прямокутні області, що складаються з однієї або кількох grid-комірок, які можуть бути названі та використані для розміщення grid-елементів.

Розуміння цих термінів є ключовим для ефективної роботи з CSS Grid.

## Властивості Grid-контейнера

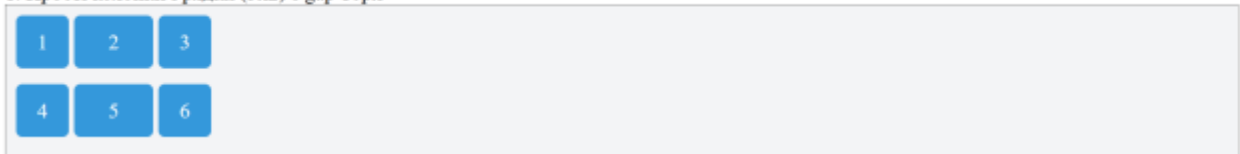
Властивості, що застосовуються до grid-контейнера, керують структурою сітки та розташуванням grid-елементів всередині неї.

- `display: grid | inline-grid`: оголошує елемент grid-контейнером. `grid` робить контейнер блоковим, а `inline-grid` – рядковим.
- `grid-template-columns: <track-list>`: визначає кількість та ширину стовпців сітки. Можна використовувати фіксовані значення (px, em, rem), відсотки (%), або гнучку одиницю fr.
- `grid-template-columns: 100px 1fr 2fr`; (перший стовпець 100px, другий займає 1 частину вільного простору, третій – 2 частини).
- `grid-template-columns: repeat(3, 1fr)`; (три стовпці однакової ширини).
- `grid-template-rows: <track-list>`: визначає кількість та висоту рядків сітки. Аналогічно `grid-template-columns`.
- `grid-template-rows: auto 100px`; (перший рядок автоматичної висоти, другий 100px).
- `grid-template-areas: <string-list>`: дозволяє створювати іменовані області в сітці, що значно спрощує візуальне проєктування макета. Кожен рядок у значенні – це рядок сітки, а назви областей – стовпці

```
css
.container {
display: grid;
grid-template-columns: 1fr 2fr 1fr;
grid-template-rows: auto 1fr auto;
grid-template-areas:
"header header header"
"sidebar main content"
"footer footer footer";
}
```

- `grid-gap` (або `gap`): Скорочений запис для `grid-row-gap` та `grid-column-gap`. Визначає проміжки між доріжками сітки.
- `grid-gap: 10px;` (10px між рядками та стовпцями).
- `grid-gap: 10px 20px;` (10px між рядками, 20px між стовпцями).
- `justify-items: start | end | center | stretch`: Вирівнює `grid`-елементи вздовж осі стовпців (горизонтально) всередині їхніх комірок.
- `align-items: start | end | center | stretch`: Вирівнює `grid`-елементи вздовж осі рядків (вертикально) всередині їхніх комірок.
- `justify-content: start | end | center | space-between | space-around | space-evenly`: Вирівнює всю сітку вздовж осі стовпців (горизонтально) всередині `grid`-контейнера.
- `align-content: start | end | center | space-between | space-around | space-evenly`: Вирівнює всю сітку вздовж осі рядків (вертикально) всередині `grid`-контейнера.

1. Прості колонки і рядки (3x2) з `gap 10px`



2. Використання `fr` одиниць (3 колонки, рівномірний поділ)

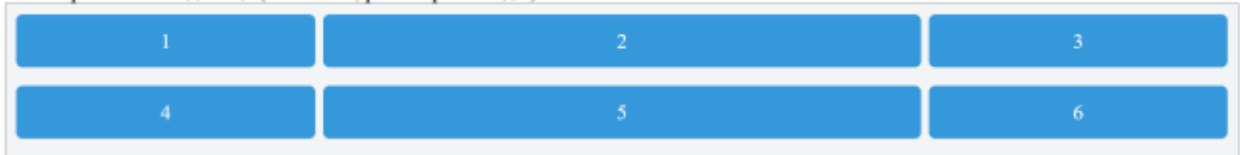


Рисунок 5.10 – Ілюстрація, що демонструє основні властивості `grid`-контейнера: `grid-template-columns`, `grid-template-rows` та `grid-gap`

## Властивості `Grid`-елементів

Властивості, що застосовуються до безпосередніх дочірніх елементів `grid`-контейнера, керують їхнім розташуванням та розміром всередині сітки.

- `grid-column-start`, `grid-column-end`: Визначають, з якої та по яку `grid`-лінію стовпців буде розташовуватися елемент.
- `grid-row-start`, `grid-row-end`: Визначають, з якої та по яку `grid`-лінію рядків буде розташовуватися елемент.
- `grid-column`: Скорочений запис для `grid-column-start` та `grid-column-end` (наприклад, `grid-column: 1 / 3`; або `grid-column: 1 / span 2`);
- `grid-row`: Скорочений запис для `grid-row-start` та `grid-row-end`.
- `grid-area: <name>`: Призначає `grid`-елементу іменовану область, визначену в `grid-template-areas`.

```
css
.header {
```

```

 grid-area: header;
 }
 .main-content {
 grid-area: main;
 }
}

```

- `justify-self: start | end | center | stretch`: Вирівнює окремий grid-елемент вздовж осі стовпців (горизонтально) всередині його комірки.
- `align-self: start | end | center | stretch`: Вирівнює окремий grid-елемент вздовж осі рядків (вертикально) всередині його комірки.

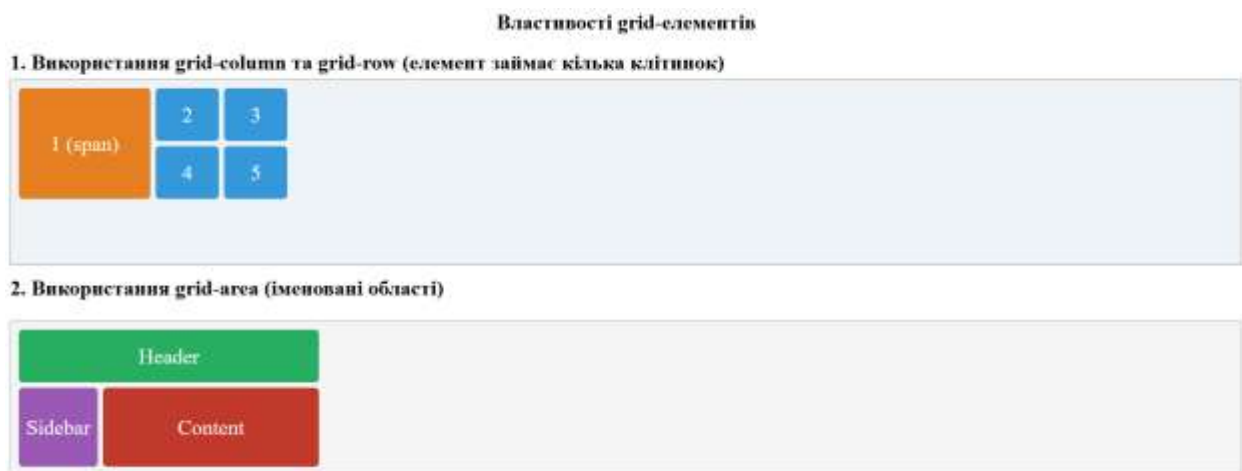


Рисунок 5.11 – Ілюстрація, що демонструє властивості grid-елементів: `grid-column`, `grid-row` та `grid-area`

## Практичні приклади використання CSS Grid

CSS Grid є ідеальним для створення основної структури сторінки та складних двовимірних макетів. Ось деякі типові сценарії,

- Основний макет сторінки: створення шапки, футера, бічних панелей та основного контенту.
- Галереї зображень: створення адаптивних галерей з різними розмірами зображень.
- Карточки товарів/послуг: розміщення елементів всередині картки (зображення, заголовок, ціна, кнопка) за допомогою Grid.
- Форми: складні форми з вирівнюванням полів у кілька стовпців.
- Дашборди: створення складних інформаційних панелей з різними віджетами.

```

<div class="grid-container">
 <header class="header">Шапка</header>
 <nav class="sidebar">Бічна панель</nav>

```

```

 <main class="main">Основний контент</main>
 <footer class="footer">Футер</footer>
</div>
.grid-container {
 display: grid;
 grid-template-columns: 200px 1fr; /* Бічна
панель 200px, контент - решта */
 grid-template-rows: auto 1fr auto; /* Шапка,
контент, футер */
 grid-template-areas:
 "header header"
 "sidebar main"
 "footer footer";
 min-height: 100vh; /* Щоб контейнер займав всю
висоту viewport */
}

.header {
 grid-area: header;
 background-color: lightblue;
 padding: 20px;
}

.sidebar {
 grid-area: sidebar;
 background-color: lightgreen;
 padding: 20px;
}

.main {
 grid-area: main;
 background-color: lightcoral;
 padding: 20px;
}

.footer {
 grid-area: footer;
 background-color: lightgray;
 padding: 20px;
}

/* Адаптивність для мобільних */
@media (max-width: 768px) {
 .grid-container {

```

```

 grid-template-columns: 1fr; /* Один стовпець
*/
 grid-template-rows: auto auto 1fr auto; /*
Шапка, бічна панель, контент, футер */
 grid-template-areas:
 "header"
 "sidebar"
 "main"
 "footer";
 }
}

```

CSS Grid є надзвичайно потужним та гнучким інструментом, що кардинально спрощує створення складних двовимірних адаптивних макетів. Глибоке розуміння його концепцій та властивостей є обов'язковим для кожного Frontend-розробника, дозволяючи ефективно вирішувати складні завдання компоновання та створювати якісні, передбачувані та зручні інтерфейси. Він ідеально доповнює Flexbox, де Flexbox використовується для компоновання елементів всередині одного рядка або стовпця, а Grid – для загальної структури сторінки.

### Завдання для демонстрації

1. Базовий Grid-макет. Створити div як grid-контейнер з кількома дочірніми div елементами. Використати grid-template-columns та grid-template-rows для створення простої сітки 3×2. Продемонструвати, як елементи автоматично розміщуються в комітках.
2. grid-template-areas. Створити макет сторінки (шапка, бічна панель, основний контент, футер) за допомогою grid-template-areas. Призначити кожному елементу відповідну область. Продемонструвати, як легко змінити розташування елементів, просто змінивши значення grid-template-areas.
3. Розміщення елементів за лініями. Використати grid-column та grid-row для того, щоб один з grid-елементів займав кілька стовпців або рядків.
4. Адаптивний Grid. Додати медіа-запит до прикладу з grid-template-areas, щоб на мобільних пристроях макет змінювався на одноколонковий.

### Питання для самоконтролю

1. У чому основна відмінність CSS Grid від Flexbox?
2. Поясніть різницю між grid-контейнером та grid-елементом.
3. Що таке grid-лінії, grid-доріжки та grid-області?

4. Як властивості `grid-template-columns` та `grid-template-rows` визначають структуру сітки?

5. Поясніть, як `grid-template-areas` спрощує створення складних макетів.

6. Які властивості використовуються для розміщення `grid`-елемента в певних комірках сітки?

### **Рекомендоване практичне завдання**

1. Верстка макета блога: створіть HTML-сторінку для блога. Використайте CSS Grid для створення основного макета, який містить:

- ✓ Шапку (header).
- ✓ Основний контент (з кількома статтями, кожна в `<article>`).
- ✓ Бічну панель (aside) з категоріями або останніми постами.
- ✓ Футер (footer).

2. Використайте `grid-template-areas` для визначення областей та `grid-gap` для проміжків. Зробіть макет адаптивним, щоб на мобільних пристроях бічна панель переміщалася під основний контент.

3. Створення галереї зображень. Створіть адаптивну галерею зображень за допомогою CSS Grid. Необхідно, щоб галерея мала 3 стовпці на десктопі, 2 стовпці на планшетах та 1 стовпець на мобільних пристроях. Використайте `repeat()` та `fr` одиниці для гнучкого розміру стовпців. Експериментуйте з `justify-items` та `align-items` для вирівнювання зображень всередині комірок.

## **5.7 Псевдокласи та псевдоелементи**

### **Псевдокласи: Стилізація елементів за станом або положенням**

У CSS ми маємо можливість стилізувати HTML-елементи не лише за їхньою назвою, класом чи ідентифікатором, але й залежно від їхнього стану або положення в структурі документа, а також додавати до них візуальні елементи, які не існують в HTML-кодi. Для цього використовуються псевдокласи (Pseudo-classes) та псевдоелементи (Pseudo-elements). Ці потужні інструменти значно розширюють можливості CSS, дозволяючи створювати динамічні, інтерактивні та візуально привабливі інтерфейси без необхідності додавати зайвий HTML або використовувати JavaScript для простих ефектів. Розуміння псевдокласів та псевдоелементів є ключовим для будь-якого Frontend-розробника, оскільки вони є невід'ємною частиною сучасної веброботи, покращуючи як UX, так і доступність.

Псевдоклас – це ключове слово, що додається до селектора, яке вказує на особливий стан елемента або його положення в дереві документа. Псевдокласи починаються з однієї двокрапки (:).

## 1. Псевдокласи взаємодії (User Action Pseudo-classes)

Ці псевдокласи дозволяють стилізувати елементи залежно від взаємодії користувача з ними:

- `:hover`: Застосовується, коли курсор миші наводиться на елемент. Часто використовується для зміни кольору посилань, кнопок або фону елементів, щоб надати візуальний зворотний зв'язок користувачеві.

```
`css
a:hover {
 color: #0056b3;
 text-decoration: underline;
}
button:hover {
 background-color: #0056b3;
 cursor: pointer;
}
```

- `:focus`: Застосовується, коли елемент отримує фокус (наприклад, за кліку на поле введення або під час навігації клавіатурою за допомогою Tab). Це критично важливо для доступності, оскільки візуальний індикатор фокусу допомагає користувачам, які використовують клавіатуру, розуміти, де вони знаходяться на сторінці. Завжди стилізуйте `:focus`!

```
css
input:focus {
 outline: 2px solid #007bff;
 border-color: #007bff;
 box-shadow: 0 0 0 0.2rem rgba(0, 123, 255, 0.25);
}
```

- `:active`: Застосовується, коли елемент активується користувачем (наприклад, під час натискання кнопки миші або клавіші Enter на посиланні). Зазвичай використовується для створення ефекту «натискання» кнопки.

```
`css
button:active {
 background-color: #004085;
 transform: translateY(1px); / Невеликий зсув вниз /
}
```

- `:visited`: Застосовується до посилань, які користувач вже відвідував. Браузери обмежують стилізацію цього псевдокласу з

міркувань приватності (можна змінювати лише color, background-color, border-color, outline-color, text-decoration-color).

- :link: Застосовується до посилань, які користувач ще не відвідував.

## 2. Структурні псевдокласи (Structural Pseudo-classes)

Ці псевдокласи дозволяють стилізувати елементи на основі їхнього положення в дереві документа:

- :first-child: Вибирає елемент, який є першим дочірнім елементом свого батька.

```
`css
ul li:first-child {
 font-weight: bold;
}
```

- :last-child: Вибирає елемент, який є останнім дочірнім елементом свого батька.

```
`css
ul li:last-child {
 border-bottom: none;
}
```

- :nth-child(n): Вибирає елемент, який є n-м дочірнім елементом свого батька. n може бути числом, ключовим словом (even, odd) або формулою (An+B).

```
`css
ul li:nth-child(odd) { / Кожен непарний елемент /
 background-color: #f0f0f0;
}
ul li:nth-child(2n) { / Кожен парний елемент /
 background-color: #e0e0e0;
}
```

- :only-child: Вибирає елемент, який є єдиним дочірнім елементом свого батька.

- :not(selector): Вибирає елементи, які не відповідають вказаному селектору. Дуже потужний для виключення елементів.

```
`css
p:not(.intro) {
 text-indent: 20px;
}
```

Приклад коду:

```
<!DOCTYPE html>
<html lang="uk">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-
width, initial-scale=1.0">
<title>Псевдокласи CSS</title>
<style>
 body {
 font-family: "Times New Roman", serif;
 font-size: 17px;
 margin: 0;
 height: 100vh;
 display: flex;
 flex-direction: column;
 justify-content: center; /* по вертикалі */
 align-items: center; /* по горизонталі */
 background: #f7f7f7;
 }
 h1 {
 text-align: center;
 font-size: 28px;
 font-weight: bold;
 margin-bottom: 30px;
 }
 h2 {
 font-size: 22px;
 margin-top: 30px;
 text-align: center;
 }
 ul {
 background: #eef1f5;
 padding: 20px;
 border-radius: 10px;
 width: 350px;
 list-style-type: none;
 }
 li {
 padding: 10px;
 border: 1px solid #ccc;
 margin-bottom: 10px;
 border-radius: 6px;
 }
}
```

```

/* Перший елемент */
li:first-child {
 background: #3498db;
 color: #000; /* змінено з білого на чорний */
 font-weight: bold;
}
/* Непарні елементи */
li:nth-child(odd):not(:first-child) {
 background: #d6eaff;
 border-color: #8fbde3;
}
button {
 font-size: 17px;
 padding: 10px 20px;
 border-radius: 8px;
 background: #2ecc71;
 border: none;
 cursor: pointer;
 margin-top: 10px;
}
button:hover {
 background: #27ae60;
 color: #fff;
}
input {
 font-size: 17px;
 padding: 10px;
 border-radius: 6px;
 border: 2px solid #ccc;
 width: 300px;
 margin-top: 10px;
}
input:focus {
 border-color: #e67e22;
 background: #fff4e6;
 outline: none;
}
}
</style>
</head>
<body>
<h1>Демонстрація псевдокласів CSS</h1>
<h2>Список зі стилями</h2>

 Перший елемент (first-child)

```

```

 Другий елемент
 Третій елемент (odd)
 Четвертий елемент
 П'ятий елемент (odd)

<h2>Кнопка (hover)</h2>
<button>Наведи на мене</button>
<h2>Поле вводу (focus)</h2>
<input type="text" placeholder="Клікни, щоб
побачити фокус">
</body>
</html>

```

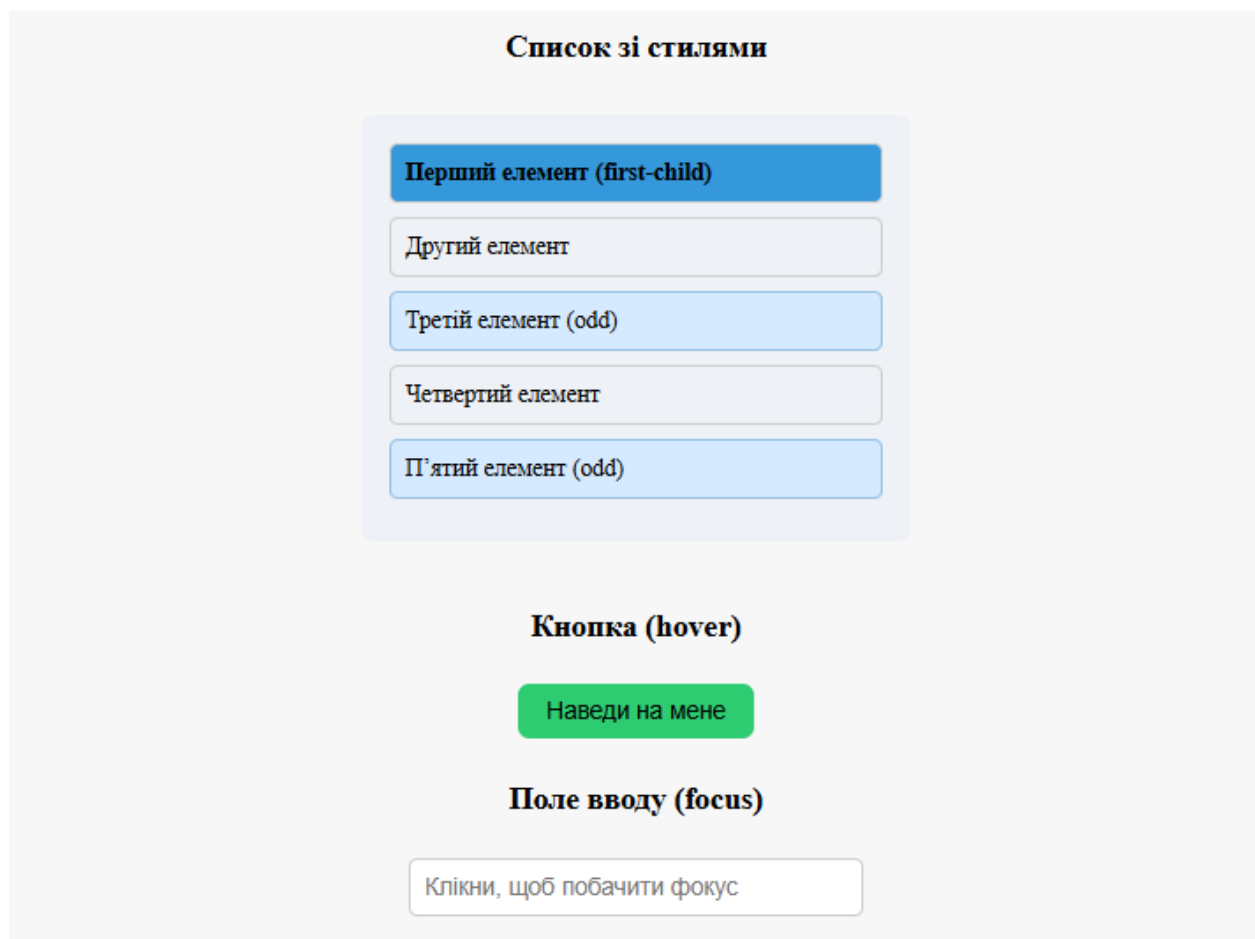


Рисунок 5.12 – Приклад HTML-коду зі списком (<ul> з <li>) та кнопками.

### Псевдоелементи: додавання та стилізація віртуального контенту

Псевдоелемент – це ключове слово, що додається до селектора, яке дозволяє стилізувати певну частину елемента або додати до нього віртуальний контент, який не існує в HTML-кодi. Псевдоелементи

починаються з двох двокрапок (::), хоча для сумісності зі старими браузерами можна використовувати одну (:).

- `::before`: створює псевдоелемент, який є першим дочірнім елементом вибраного елемента. Контент додається за допомогою властивості `content`.

- `::after`: створює псевдоелемент, який є останнім дочірнім елементом вибраного елемента. Контент також додається за допомогою властивості `content`.

Псевдоелементи `::before` та `::after` є блоковими за замовчуванням (або `inline` залежно від контексту) і можуть бути стилізовані як звичайні елементи (мати `width`, `height`, `background`, `border` тощо). Вони часто використовуються для:

- Додавання декоративних елементів (іконок, стрілок, ліній).
- Створення кастомних маркерів списків.
- Очищення плаваючих елементів (clearfix hack).
- Додавання цитат або інших текстових елементів.

```
/* Додавання іконки перед посиланням */
a::before{
 content: "\2192 "; /* Юнікод-символ стрілки */
 margin-right:5px;
 color:#007bff;
}
/* Створення декоративної лінії під заголовком */
h2{
 position:relative;
 padding-bottom:10px;
}
h2::after{
 content:"";
 position:absolute;
 left:0;
 bottom:0;
 width:50px;
 height:3px;
 background-color:#007bff;
}
```

- `::first-letter`: стилізує першу літеру першого рядка текстового блока. Часто використовується для створення ефекту буквиці.

- `::first-line`: стилізує перший рядок текстового блока.

- `::selection`: стилізує текст, виділений користувачем.

Приклад коду:

```
<!DOCTYPE html>
```

```

<html lang="uk">
<head>
<meta charset="UTF-8">
<title>Приклад ::before та ::after</title>
<style>
 body {
 font-family: 'Times New Roman', Times, serif;
 display: flex;
 justify-content: center; /* по горизонталі */
 align-items: center; /* по вертикалі */
 height: 100vh; /* повна висота
вікна */
 margin: 0;
 background-color: #f9f9f9;
 text-align: center; /* вирівнювання
тексту по центру */
 }
 .container {
 max-width: 600px;
 }
 h2 {
 position: relative;
 font-size: 32px;
 color: #333;
 margin-bottom: 20px;
 }
 /* Іконка перед заголовком */
 h2::before {
 content: "□";
 margin-right: 10px;
 font-size: 32px;
 }
 /* Декоративна лінія після заголовка */
 h2::after {
 content: "";
 display: block;
 width: 60px;
 height: 4px;
 background-color: #ff6347;
 margin: 10px auto 0 auto; /* по центру під
заголовком */
 border-radius: 2px;
 }
 p {

```

```

 font-size: 18px;
 line-height: 1.6;
 color: #555;
 }
</style>
</head>
<body>
<div class="container">
 <h2>Заголовок з іконкою</h2>
 <p>Це приклад параграфу під заголовком.
Псевдоелементи ::before та ::after додають
декоративні елементи без зміни HTML-коду.</p>
</div>

</body>
</html>

```

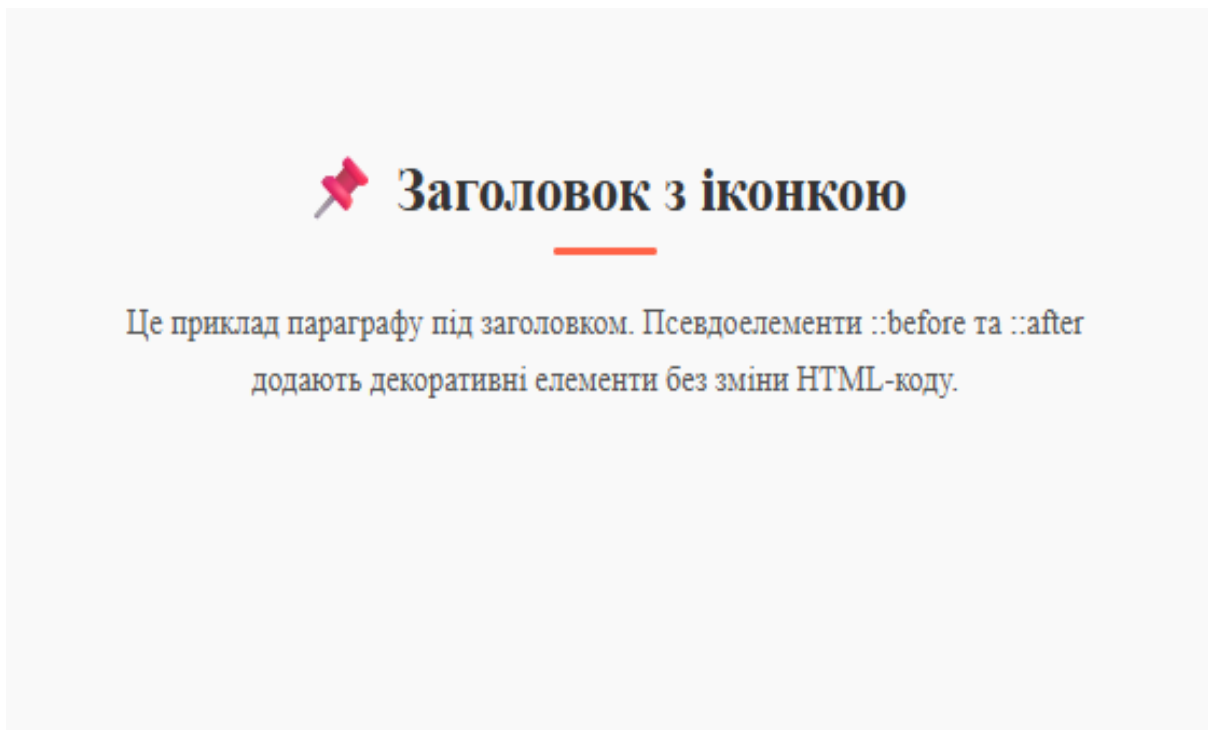


Рисунок 5.13 – Приклад HTML-коду з заголовком та параграфом

Псевдокласи та псевдоелементи є невід'ємною частиною сучасного CSS, дозволяючи створювати складні та динамічні візуальні ефекти без перевантаження HTML-коду. Розуміння їхнього призначення та вміле використання є ключовим для Frontend-розробника, що прагне створювати якісні, інтерактивні та доступні вебінтерфейси.

## Завдання для демонстрації

1. Псевдокласи взаємодії: Створити HTML-сторінку з посиланнями, кнопками та полями введення. Застосувати до них стилі для `:hover`, `:focus`, `:active`. Продемонструвати, як змінюються стилі за взаємодії з елементами.
2. Структурні псевдокласи: Створити нумерований список з 5-7 елементів. Застосувати стилі до `:first-child`, `:last-child`, `:nth-child(odd)`, `:nth-child(even)`. Продемонструвати, як стилізуються елементи залежно від їхнього положення.
3. Псевдоелементи `::before` та `::after`: Створити HTML-сторінку з заголовком та параграфом. Використати `::before` для додавання декоративної іконки перед заголовком. Використати `::after` для додавання декоративної лінії під параграфом. Продемонструвати, як ці псевдоелементи додають контент без зміни HTML.

## Питання для самоконтролю

1. У чому різниця між псевдокласом та псевдоелементом?
2. Назвіть 3 псевдокласи взаємодії та поясніть, коли вони застосовуються.
3. Чому стилізація `:focus` є критично важливою для доступності?
4. Як працює псевдоклас `:nth-child(n)`? Наведіть приклад використання з формулою.
5. Для чого використовуються псевдоелементи `::before` та `::after`?
6. Яка властивість є обов'язковою у випадку використанні `::before` та `::after` для додавання контенту?

## Рекомендоване практичне завдання

1. Стилізація інтерактивних елементів: Створіть HTML-сторінку з кількома кнопками та посиланнями. Застосуйте до них стилі для `:hover`, `:focus`, `:active`, щоб створити привабливі та інтуїтивно зрозумілі ефекти взаємодії. Переконайтеся, що `:focus` стилі чітко видимі та відрізняються від `:hover`.
2. Створення списку з кастомними маркерами: Створіть нумерований список. Використайте псевдоелемент `::before` для створення кастомних маркерів списку (наприклад, замість стандартних кружечків використовуйте іконки або спеціальні символи). Експериментуйте з позиціонуванням та стилізацією цих маркерів.
3. Декоративні елементи за допомогою `::after`: Створіть секцію з заголовком та текстом. Використайте псевдоелемент `::after` для додавання декоративного елемента (наприклад, короткої лінії, фігури) під заголовком або в кінці параграфа. Експериментуйте з `position: absolute`; для точного розташування псевдоелемента.

## 5.8 Трансформації та переходи

### CSS-трансформації (`transform`): Зміна положення, розміру, форми

Сучасний вебдизайн виходить далеко за межі статичних сторінок. Інтерактивність та динамічні візуальні ефекти відіграють ключову роль у створенні привабливого та захоплюючого користувацького досвіду. CSS надає потужні інструменти для реалізації таких ефектів без використання JavaScript для простих анімацій. Ця лекція Цей підрозділ присвячено двом основним властивостям, які дозволяють нам «оживити» елементи на сторінці: CSS-трансформації (`transform`) та плавні переходи (`transition`). Розуміння цих властивостей є критично важливим для будь-якого Frontend-розробника, оскільки вони дозволяють створювати анімації, які є продуктивними, гнучкими та покращують UX, надаючи візуальний зворотний зв'язок користувачеві.

Властивість `transform` дозволяє змінювати форму, розмір, положення та орієнтацію елемента у 2D або 3D просторі. Це відбувається без зміни нормального потоку документа, тобто елемент трансформується, але не впливає на розташування інших елементів. Трансформації є дуже продуктивними, оскільки вони виконуються графічним процесором (GPU) браузера

#### 2D-трансформації

- `translate(x, y)`: Переміщує елемент по осях X та Y. `translateX(x)` переміщує лише по X, `translateY(y)` – лише по Y. Значення можуть бути в px, %, em тощо.

```
css
.box:hover {
 transform: translate(10px, 20px); / Перемістити
на 10px вправо, 20px вниз /
}
```

- `scale(x, y)`: Змінює розмір елемента. `scaleX(x)` масштабує по X, `scaleY(y)` – по Y. Значення 1 – оригінальний розмір, 2 – подвійний розмір, 0.5 – половина розміру.

```
css
.box:hover {
 transform: scale(1.2); / Збільшити в 1.2 рази /
}
```

- `rotate(angle)`: Обертає елемент навколо його центральної точки. Кут задається в `deg` (градусах), `rad` (радіанах), `grad` (градах) або `turn` (обертах).

```
css
.box:hover {
 transform: rotate(45deg); / Обернути на 45
градусів за годинниковою стрілкою /
}
```

- `skew(angleX, angleY)`: Нахиляє елемент по осях X та Y. `skewX(angle)` нахиляє по X, `skewY(angle)` – по Y.

```
css
.box:hover {
 transform: skew(10deg, 5deg); / Нахилити на 10deg
по X, 5deg по Y /
}
```

- `matrix(a, b, c, d, e, f)`: Дозволяє застосовувати всі 2D-трансформації за допомогою однієї матриці перетворення. Це більш складний, але потужний спосіб.

### 3D-трансформації

3D-трансформації дозволяють переміщувати, масштабувати та обертати елементи у тривимірному просторі. Для їх коректного відображення часто потрібна властивість `perspective` на батьківському елементі.

- `translate3d(x, y, z)`: переміщує елемент по осях X, Y та Z. `translateZ(z)` переміщує лише по Z (ближче/далі до користувача).

- `scale3d(x, y, z)`: масштабує елемент по осях X, Y та Z.

- `rotateX(angle)`: обертает елемент навколо осі X.

- `rotateY(angle)`: обертает елемент навколо осі Y.

- `rotateZ(angle)`: обертает елемент навколо осі Z (аналогічно 2D `rotate`).

- `perspective(value)`: застосовується до батьківського елемента і створює перспективу для 3D-трансформацій дочірніх елементів. Чим менше значення, тим сильніша перспектива.

- `transform-origin`: визначає точку, навколо якої відбуваються трансформації (за замовчуванням `center center`). Може бути заданий у `px`, `%` або ключовими словами (`top left`, `bottom right`).

```
.card {
 transform-origin: center center;
 transition: transform 0.5s ease;
}
```

```

 .card:hover {
 transform: rotateY(180deg); /* Обертання навколо
oci Y */
 }
 .container {
 perspective: 1000px; /* Створює 3D-перспективу
для дочірніх елементів */
 }

```

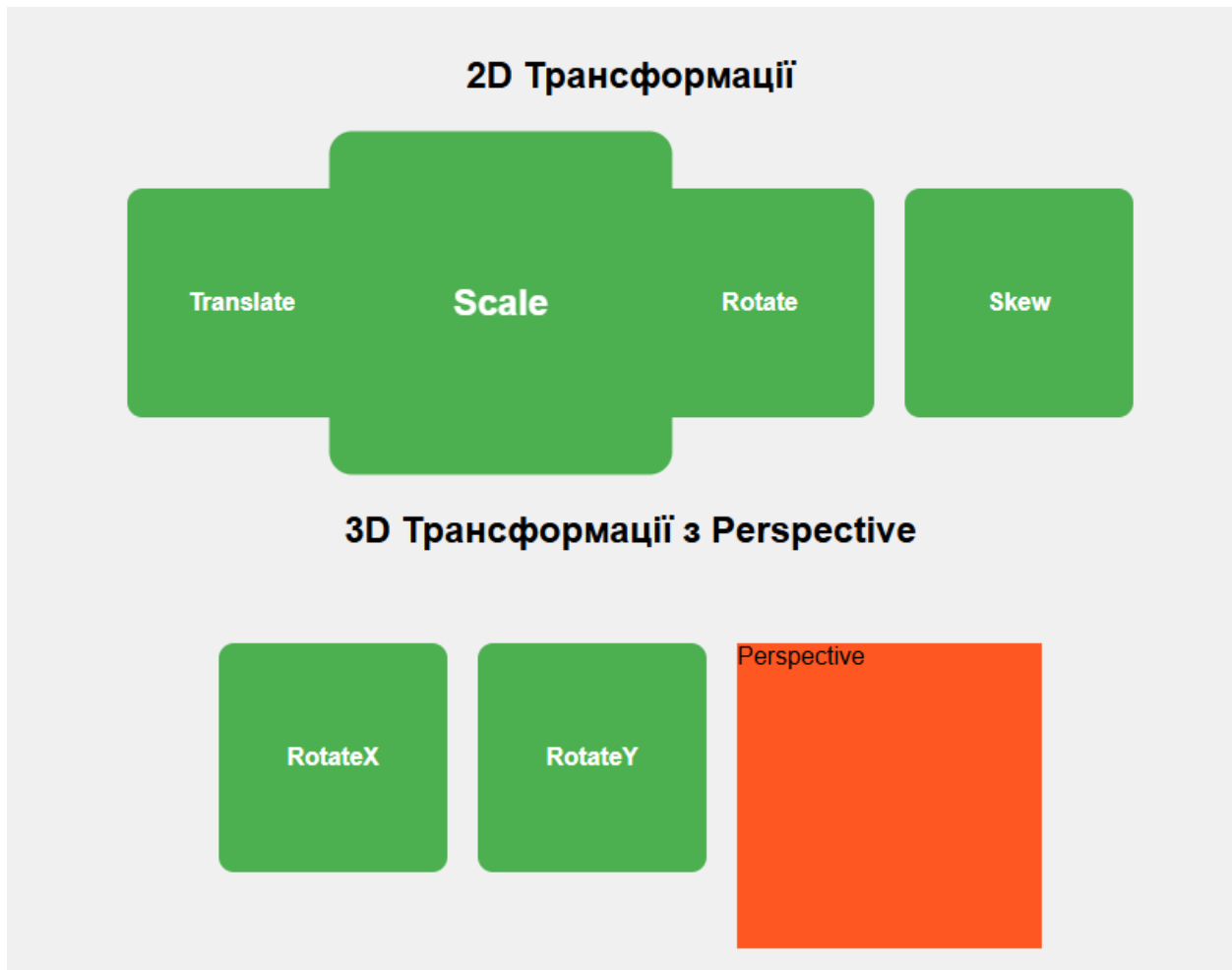


Рисунок 5.14 – Приклад, що демонструє різні 2D та 3D трансформації

### **Плавні переходи (transition): Анімація між станами**

Властивість `transition` дозволяє створювати плавні анімовані переходи між різними станами CSS-властивостей. Замість миттєвої зміни стилю, `transition` забезпечує плавний рух від одного значення до іншого, що значно покращує користувацький досвід, роблячи інтерфейс більш «живим» та зрозумілим.

Властивість `transition` є скороченим записом для чотирьох підвластивостей:

- `transition-property`: вказує, які CSS-властивості будуть анімовані. Може бути `all` (всі властивості, що підтримують анімацію) або конкретна властивість (наприклад, `color`, `background-color`, `transform`).
  - `transition-duration`: визначає тривалість переходу (в секундах `s` або мілісекундах `ms`).
  - `transition-timing-function`: визначає швидкість зміни анімації протягом її тривалості. Це так звані функції часу (Timing Functions):
    - `ease` (за замовчуванням): повільно починається, прискорюється в середині, повільно закінчується.
    - `linear`: рівномірна швидкість від початку до кінця.
    - `ease-in`: повільно починається, потім прискорюється.
    - `ease-out`: швидко починається, потім повільно закінчується.
    - `ease-in-out`: повільно починається і закінчується, швидко в середині.
  - `cubic-bezier(n, n, n, n)`: дозволяє створити власну функцію часу для повного контролю над швидкістю анімації.
  - `transition-delay`: визначає затримку перед початком переходу.
- ```
.button {
background-color: #007bff;
color: white;
padding: 10px 20px;
border: none;
transition: background-color 0.3s ease-in-out, transform
0.2s ease; /* Анімуємо дві властивості */
}

.button:hover {
background-color: #0056b3;
transform: scale(1.05); /* Збільшити при наведенні */
}
```

Практичне застосування та оптимізація

Трансформації та переходи є потужними інструментами для створення:

- Ефектів під час наведення (`hover effects`): зміна розміру кнопок, підсвічування посилань, обертання іконок.
- Слайдерів та каруселей: плавні переходи між слайдами.
- Модальних вікон та висхідних панелей: плавна поява та зникнення.
- Паралакс-ефектів: ілюзія глибини під час прокручування сторінки.

Оптимізація: для найкращої продуктивності завжди анімуйте властивості `transform` та `opacity`, оскільки вони не викликають перерахунку макета (`reflow`) та перерисовування (`repaint`) сторінки, а виконуються

графічним процесором. Анімація width, height, margin, padding може бути менш продуктивною, особливо на мобільних пристроях.

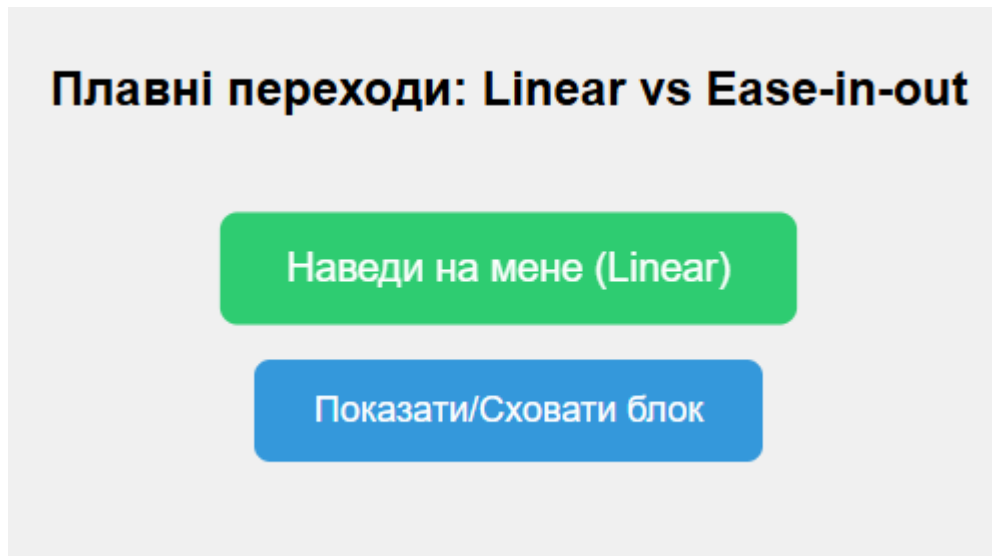


Рисунок 5.15 – Приклад, що демонструє плавні переходи

CSS-трансформації та переходи є невід'ємною частиною сучасного вебдизайну, дозволяючи створювати динамічні, інтерактивні та візуально привабливі інтерфейси. Розуміння їхнього призначення, можливостей та обмежень є ключовим для Frontend-розробника, що прагне створювати високоякісні та продуктивні вебдодатки. Вони дозволяють значно покращити користувацький досвід, роблячи взаємодію з сайтом більш інтуїтивною та приємною.

Завдання для демонстрації

1. 2D-трансформації: створити HTML-сторінку з кількома div елементами. Застосувати до них різні 2D-трансформації (translate, scale, rotate, skew) за наведення курсору (:hover). Продемонструвати, як transform-origin впливає на точку обертання/масштабування.
2. 3D-трансформації: створити батьківський div з perspective. Всередині нього створити дочірній div (наприклад, картку). У разі наведення курсору на картку застосувати rotateY(180deg) для створення ефекту перевертута. Продемонструвати, як perspective впливає на глибину.
3. Плавні переходи: додати до кнопок та посилань на сторінці властивість transition для background-color, color, transform. Експериментувати з transition-duration та transition-timing-function (ease, linear, ease-in-out). Продемонструвати різницю у швидкості та плавності анімації.

Питання для самоконтролю

1. Що таке CSS-трансформації і які їхні основні переваги?
2. Назвіть 3 основні 2D-трансформації та поясніть їхнє призначення.
3. Для чого використовується властивість `perspective` у 3D-трансформаціях?
4. Що таке `transition` і які його основні підвластивості?
5. Поясніть різницю між `linear` та `ease-in-out` функціями часу.
6. Які властивості рекомендується анімувати для найкращої продуктивності?

Рекомендоване практичне завдання

1. Створення інтерактивних карток. Створіть HTML-сторінку з кількома картками (наприклад, картки послуг або членів команди). У разі наведення курсору на кожну картку вона має:

- трохи збільшуватися (`scale`);
- змінювати колір фону;
- можливо, трохи підніматися (`translateY`).

Всі ці зміни мають відбуватися плавно за допомогою `transition`.

- Анімована кнопка. Створіть кнопку, яка у разі наведення курсору:
 - змінює колір фону;
 - має невеликий ефект «пульсації» або «підстрибування» (можна комбінувати `scale` та `translateY`);
 - текст кнопки може плавно змінювати колір.

Використайте різні `transition-duration` та `transition-timing-function` для різних властивостей, щоб створити цікавий ефект.

2. 3D-переворот елемента. Створіть `div` елемент, який у разі наведення курсору плавно перевертається на 180 градусів навколо осі Y, показуючи інший вміст на «зворотній» стороні. Використайте `perspective` на батьківському елементі та `transform: rotateY()` на дочірньому. (Для реалізації «зворотної» сторони можна використовувати псевдоелементи `::before`/`::after` або два вкладені `div` з `backface-visibility: hidden`);).

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Introduction to HTML - W3Schools [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/html/html_intro.asp
2. HTML Tables - W3Schools [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/html/html_tables.asp
3. HTML basics - Learn web development - MDN [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics
4. HTML редактор-пісочниця TryIT - W3Schools [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/html/tryit.asp?filename=tryhtml_intro
5. Гайд по CSS селекторам [Електронний ресурс]. – Режим доступу: <https://codeguida.com/post/69>
6. Що таке CSS - Ніжинський державний університет імені Миколи Гоголя [Електронний ресурс]. – Режим доступу: http://www.ndu.edu.ua/liceum/html/teor_CSS.pdf
7. Синтаксис JavaScript - W3Schools [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/js/js_syntax.asp
8. Змінні у JavaScript - W3Schools [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/js/js_variables.asp
9. Оператори JavaScript - W3Schools [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/js/js_operators.asp
10. Функції JavaScript - W3Schools [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/js/js_functions.asp
11. Арифметика у JavaScript - W3Schools [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/js/js_arithmetic.asp
12. SQLCourse - Безкоштовний онлайн-курс з SQL [Електронний ресурс]. – Режим доступу: <https://dou.ua/forums/topic/41184/>
13. Introduction to Databases - Курс на Coursera, який вводить у основи роботи з базами даних [Електронний ресурс]. – Режим доступу: <https://www.coursera.org>
14. PostgreSQL Tutorial - Тutorials з PostgreSQL, включаючи основні та складні концепції [Електронний ресурс]. – Режим доступу: <https://www.w3schools.com/postgresql/index.php>
15. MySQL Tutorial - Тutorials та приклади з MySQL [Електронний ресурс]. – Режим доступу: <https://www.w3schools.com/MySQL/default.asp>
16. SQLite Tutorial - Навчальний матеріал з SQLite, легкого використання реляційної бази даних [Електронний ресурс]. – Режим доступу: <https://www.sqlitetutorial.net/>
17. MongoDB University - Безкоштовні курси від MongoDB, включаючи вступні та поглиблені матеріали [Електронний ресурс]. – Режим доступу: <https://learn.mongodb.com/>

18. Choosing the Right Database for Your Application - Стаття, яка допомагає вибрати підходящу базу даних для вашого проекту [Електронний ресурс]. – Режим доступу: https://dev.to/saloman_james/how-to-choose-the-right-database-for-your-project-1cl

19. SQL vs NoSQL: High-Level Differences - Стаття, яка порівнює реляційні та NoSQL бази даних з високого рівня [Електронний ресурс]. – Режим доступу: <https://www.integrate.io/blog/the-sql-vs-nosql-difference/>

20. "How to Create a Parallax Scrolling Website": Ця стаття надає докладний огляд того, як створити веб-сайт з ефектом паралаксу за допомогою HTML, CSS і JavaScript [Електронний ресурс]. – Режим доступу: <https://www.geeksforgeeks.org/how-to-create-a-parallax-scrolling-website/>

21. "A Simple Parallax Scrolling Technique": Розглядає простий підхід до створення ефекту паралаксу та надає приклади коду [Електронний ресурс]. – Режим доступу: <https://www.smashingmagazine.com/2016/07/simple-parallax-scrolling-technique/>

22. "Advanced Parallax Scrolling Techniques Tutorial": Відеоурок демонструє різні техніки реалізації ефекту паралаксу із застосуванням HTML, CSS і JavaScript [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=JT_0mHd4mpA

23. "Creating a Parallax Website from Scratch": Ця стаття пропонує крок за кроком інструкції з створення власного веб-сайту з ефектом паралаксу від самого початку [Електронний ресурс]. – Режим доступу: <https://webdesign.tutsplus.com/tutorials/create-a-parallax-scrolling-website-using-stellarjs--cms-24964>

24. "The Parallax Effect in Web Design: Create Depth and Dimension": Ця стаття розглядає принципи та приклади використання ефекту паралаксу для створення вражаючих вебсайтів [Електронний ресурс]. – Режим доступу: <https://webflow.com/blog/parallax-scrolling>

25. "Introduction to Parallax Scrolling using CSS & JavaScript": Цей відеоурок на YouTube пропонує вступний огляд ефекту паралаксу та демонструє, як його реалізувати за допомогою CSS і JavaScript [Електронний ресурс]. – Режим доступу: <https://www.youtube.com/watch?v=0Hq7IEhF8nc>

26. MDN Web Docs: Responsive Web Design Basics: основи адаптивного веб-дизайну та приклади коду [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design_Basics

27. Google Developers: Responsive Web Design Fundamentals: ресурс надає глибоке розуміння принципів адаптивного веб-дизайну та кращі практики [Електронний ресурс]. – Режим доступу: <https://developers.google.com/web/fundamentals/design-and-ux/responsive>

28. CSS-Tricks: A Complete Guide to Flexbox: Гнучкий сітковий макет є одним з ключових елементів адаптивного дизайну. Ресурс допоможе вам зрозуміти всі аспекти flexbox [Електронний ресурс]. – Режим доступу: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

29. Responsive Images: Srcset & Sizes Explained: На цьому ресурсі ви дізнаєтесь про різні способи оптимізації та відображення зображень на різних пристроях [Електронний ресурс]. – Режим доступу: <https://www.responsivebreakpoints.com/>

30. Can I Use: Цей ресурс допоможе вам перевірити, які CSS-властивості, HTML-теги та JavaScript-функції підтримуються в різних браузерах, що є важливим для адаптивного веб-дизайну [Електронний ресурс]. – Режим доступу: <https://caniuse.com/>

31. Viewport Meta Tag: Стаття розглядає використання тега мета-відображення для керування відображенням контенту на різних пристроях [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/css/css_rwd_viewport.asp

32. Performance Testing Guidance for Web Applications від Microsoft [Електронний ресурс]. – Режим доступу: [https://learn.microsoft.com/en-us/previous-versions/msp-n-p/bb924375\(v=pandp.10\)](https://learn.microsoft.com/en-us/previous-versions/msp-n-p/bb924375(v=pandp.10))

33. Molyneaux, I. The Art of Application Performance Testing [Текст] / Ian Molyneaux. – 2-ге вид. – O'Reilly Media, 2014. – 257 с.

34. Killelea, P. Web Performance Tuning [Текст] / Patrick Killelea. – 2-ге вид. – O'Reilly Media, 2002. – 480 с.

35. Clarke, A. SEO 2023: Learn Search Engine Optimization with Smart Internet Marketing Strategies [Текст] / Adam Clarke. – CreateSpace Independent Publishing Platform, 2023. – 322 с.

36. Enge, E., Spencer, S., Stricchiola, J. The Art of SEO: Mastering Search Engine Optimization [Текст] / Eric Enge, Stephan Spencer, Jessie Stricchiola. – 3-тє вид. – O'Reilly Media, 2015. – 994 с.

37. Kent, P. SEO for Dummies [Текст] / Peter Kent. – 7-ме вид. – For Dummies, 2019. – 512 с.

38. Clay, B. Search Engine Optimization All-in-One For Dummies [Текст] / Bruce Clay. – 4-ге вид. – For Dummies, 2015. – 800 с.

39. Clarke, A. SEO 2022: Learn Search Engine Optimization with Smart Internet Marketing Strategies [Текст] / Adam Clarke. – CreateSpace Independent Publishing Platform, 2022. – 298 с.

40. French, G., Ward, E. Ultimate Guide to Link Building: How to Build Website Authority, Increase Traffic and Search Ranking with Backlinks [Текст] / Garrett French, Eric Ward. – 2-ге вид. – Entrepreneur Press, 2020. – 304 с.

41. The Beginner's Guide to SEO [Електронний ресурс] / Moz. – Режим доступу: <https://moz.com/beginners-guide-to-seo>

42. Bailyn, E. SEO Made Easy: Everything You Need to Know About SEO and Nothing More [Текст] / Evan Bailyn. – 2-ге вид. – Pearson Education, 2013. – 312 с.

43. McDonald, J. SEO Fitness Workbook: 2023 Edition: The Seven Steps to Search Engine Optimization Success on Google [Текст] / Jason McDonald. – CreateSpace Independent Publishing Platform, 2023. – 450 с.

44. Ducharme, R. Advanced SEO: Search Engine Optimization Secrets That Will Increase Your Search Engine Traffic [Текст] / Robert Ducharme. – CreateSpace Independent Publishing Platform, 2015. – 152 с.

Електронне навчальне видання

Роман Юрійович Чехместрук

Основи WEB-дизайну

Частина 1

Навчальний посібник

Рукопис оформив *Р. Чехместрук*

Редактор *Т. Старічек*

Оригінал-макет виготовила *Т. Старічек*

Підписано до видання 06.07.2026 р.
Гарнітура Times New Roman.
Зам. № P2026-081.

Видавець та виготовлювач –
Вінницький національний технічний університет,
Редакційно-видавничий відділ.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95, м. Вінниця, 21021.
press.vntu.edu.ua;
Email: rvv.vntu@gmail.com
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.