МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп'ютерних технологій

Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Алгоритмічне та програмне забезпечення розробки гнучких та адаптивних макетів на сайтах»

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки

Освітня програма Комп'ютерні науки

Виконав: студент групи МгІТ-21
Владислав БОБРОВНИК
Науковий керівник
к.т.н., доц. Геннадій МЕЛЬНИК
Рецензент_____

Київ 2024

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет Мехатроніки та комп'ютерних технологій

Кафедра Комп’ютерних наук

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп’ютерні науки

Освітня програма Комп’ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
_____Наталія ЧУПРИНКА

«_____»_____20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бобровник Владислав Андрійович

1. Тема кваліфікаційної роботи *Алгоритмічне та програмне забезпечення розробки гнучких та адаптивних макетів на сайтах,*

науковий керівник роботи Мельник Г.В.*, к.т.н., доцент,*
затверджені наказом КНУТД від «*03*» *вересня 2024 року* №*188-уч*

2. Вихідні дані до кваліфікаційної роботи Розробка кафедри комп’ютерних наук, матеріали за тематикою дипломної роботи
3.
4. Зміст кваліфікаційної роботи Розділ 1(Теоретичні оснорви розробки); Розділ 2(Проектування алгоритмічного та програмного забезпечення); Розділ 3 (Практична реалізація адаптованого макету); Висновки.

5. Дата видачі завдання 08.2024.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної магістерської роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	09.10.2024	
2	Розділ 1 Теоретичні основи розробки макетів	12.10.2024	
3	Розділ 2 Проектування алгоритмічного та програмного забезпечення	20.10.2024	
4	Розділ 3 Практична реалізація адаптованого макету	25.10.2024	
5	Висновки	25.10.2024	
6	Оформлення дипломної магістерської роботи (чистовий варіант)	30.10.2024	
7	Здача дипломної магістерської роботи на кафедру длярецензування (за 14 днів до захисту)	30.10.2024	
8	Перевірка дипломної магістерської роботи нанаявність ознак плагіату (за 10 днів до захисту)	30.10.2024	
9	Подання дипломної магістерської роботи у відділмагістратури для перевірки виконання додатку до індивідуального навчального плану		
10	Подання дипломної магістерської роботи на затвердження завідувачу кафедри (за 7 днів до захисту)		

Студент Владислав БОБРОВНИК

Науковий керівник роботи Геннадій МЕЛЬНИК

АНОТАЦІЯ

Алгоритмічне та програмне забезпечення розробки гнучких та адаптивних макетів на сайтах

Дипломна магістерська робота за спеціальністю 122 - «Комп’ютерні науки». – Київський національний університет технологій та дизайну, Київ, 2024 рік.

Дипломна робота присвячена розробці адаптивного веб-макету для спортивного клубу на прикладі кросфіт-університету. Під час роботи було створено веб-ресурс із сучасним дизайном, що забезпечує адаптивність та оптимізовану взаємодію користувачів на різних пристроях, включаючи мобільні телефони, планшети та десктопи.

Для вирішення поставленої задачі було застосовано сучасні технології HTML5, CSS3 та JavaScript, що дозволили реалізувати динамічний і гнучкий макет. Веб-ресурс був побудований з використанням CSS Grid та Flexbox для створення адаптивної сітки сторінки, що забезпечує коректне відображення контенту незалежно від розміру екрану. Інтерактивні елементи, такі як слайдери, мобільне меню та калькулятор BMI, були реалізовані за допомогою JavaScript.

У ході роботи було проведено тестування та оптимізацію макету для забезпечення його коректної роботи у всіх популярних браузерах та на різних пристроях. Також було досліджено та проаналізовано сучасні підходи до адаптивної верстки, впровадження медіа-запитів та оптимізації продуктивності веб-сайтів.

Ключові слова: адаптивний веб-макет, HTML5, CSS3, JavaScript, CSS Grid, Flexbox, кросбраузерне тестування.

ANNOTATION

Algorithmic and Software Solutions for Developing Flexible and Adaptive Layouts on Websites

Master's Thesis in the specialty 122 - "Computer Science". – Kyiv National University of Technologies and Design, Kyiv, 2024.

This master's thesis is dedicated to the development of an adaptive web layout for a sports club, using the example of a CrossFit university. During the project, a modern web resource was created, providing adaptive and optimized user interaction across different devices, including mobile phones, tablets, and desktops.

To achieve the objectives, modern technologies such as HTML5, CSS3, and JavaScript were applied, allowing the creation of a dynamic and flexible layout. The web resource was built using CSS Grid and Flexbox to create a responsive page grid that ensures proper display of content regardless of screen size. Interactive elements, such as sliders, a mobile menu, and a BMI calculator, were implemented using JavaScript.

Throughout the project, testing and optimization of the layout were conducted to ensure its proper functioning across all popular browsers and various devices. Modern approaches to adaptive layout, the implementation of media queries, and website performance optimization were also researched and analyzed.

Keywords: adaptive web layout, HTML5, CSS3, JavaScript, CSS Grid, Flexbox, cross-browser testing.

ЗМІСТ

ВСТУП

Розвиток веб-технологій на сучасному етапі неминуче залежить від вдосконалення алгоритмічного та програмного забезпечення, що забезпечує гнучкість та адаптивність веб-сайтів. Ці технології змінюють підхід до розробки інтерфейсів користувачів, дозволяючи створювати сайти, які автоматично підлаштовуються під різні пристрої та екрани. У світі, де кількість користувачів мобільних пристроїв постійно зростає, здатність сайтів адаптуватися до різних розмірів і роздільних здатностей екранів є вирішальним фактором для зручності користування та доступності веб-ресурсів.

Інтернет надає доступ до численних інструментів і технологій для створення таких гнучких та адаптивних макетів, що робить процес розробки ефективнішим та швидшим. Адаптивний дизайн веб-сторінок дозволяє покращити користувацький досвід, забезпечуючи оптимальну взаємодію незалежно від типу пристрою. Для цього застосовуються алгоритмічні підходи, які дозволяють реалізовувати адаптивність за допомогою медіазапитів, відносних одиниць виміру, гнучких сіток та інших інструментів, що допомагають підтримувати цілісність дизайну на будь-якому екрані.

Перехід на адаптивні веб-технології вимагає від розробників впровадження нових підходів до організації макетів. Це не лише сприяє підвищенню зручності користування, але й дозволяє зробити взаємодію з веб-ресурсами інтуїтивно зрозумілою, незалежно від того, чи використовується мобільний пристрій, планшет або настільний комп’ютер. Сучасне програмне забезпечення, таке як HTML5, CSS3, JavaScript, а також популярні фреймворки (Bootstrap, Foundation), надають можливість створювати гнучкі макети, які автоматично змінюють свій вигляд залежно від умов перегляду та динамічних потреб користувачів.

Адаптивний дизайн також дозволяє покращити не лише користувацький досвід, але й оптимізувати продуктивність веб-ресурсів, забезпечуючи швидке завантаження контенту та мінімізацію затримок. Це є особливо важливим з огляду на те, що швидкість завантаження сторінок впливає на користувацьке задоволення, а також на пошукові алгоритми, які враховують цей показник для ранжування сайтів у результатах пошуку.

Таким чином, враховуючи стрімкий розвиток веб-технологій, зростає необхідність інтенсивного впровадження адаптивних підходів до розробки веб-сайтів, використання алгоритмічних та програмних інструментів для підвищення ефективності створення гнучких макетів. Це забезпечить зручність користування, швидкість завантаження, комфортну взаємодію для користувачів різних пристроїв, а також допоможе підтримувати високі стандарти доступності та продуктивності веб-ресурсів.

Об'єктом дослідження є процес створення гнучких та адаптивних макетів для веб-сайтів, що дозволяють ефективно забезпечувати комфортну роботу на різних типах пристроїв.

Предметом дослідження є алгоритмічні та програмні засоби розробки гнучких і адаптивних макетів, зокрема технології HTML5, CSS3, JavaScript та популярні фреймворки.

Мета дослідження полягає у теоретичному обґрунтуванні та практичній реалізації алгоритмічних та програмних методів розробки адаптивних веб-сторінок, що дозволить підвищити ефективність взаємодії користувачів із сайтом незалежно від пристрою.

Основні **завдання** роботи:

1. Дослідити еволюцію веб-дизайну до гнучких та адаптивних макетів.
2. Проаналізувати основні методи та засоби розробки гнучких макетів.
3. Визначити найбільш підходящі інструменти для створення адаптивних макетів.
4. Розробити та протестувати гнучкий та адаптивний макет на практиці.
5. Оптимізувати макет для швидкої роботи на різних пристроях.

Практична цінність роботи полягає в розробці веб-ресурсу, що забезпечить адаптивну взаємодію користувачів із сайтом незалежно від типу пристрою, що підвищить ефективність роботи з ресурсами в сучасному мобільному світі. Розроблений макет може бути використаний у реальних проєктах для створення зручних та доступних веб-сторінок, що сприятиме покращенню користувацького досвіду.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ГНУЧКИХ ТА АДАПТИВНИХ МАКЕТІВ НА САЙТАХ

1.1. Еволюція веб-дизайну до гнучких та адаптивних макетів

Веб-дизайн пройшов довгий шлях від статичних сторінок до сучасних гнучких та адаптивних макетів, що можуть автоматично підлаштовуватися під різні типи пристроїв. Цей розвиток став відповіддю на зміни у способах споживання інформації, а також на появу нових технологій та інструментів для веб-розробки [1].

Ранні етапи розвитку веб-дизайну тісно пов'язані з появою перших веб-сайтів, що мали фіксовані розміри й базувалися на простих технологіях. Перший веб-сайт був створений в 1991 році Тімом Бернерсом-Лі, засновником Всесвітньої павутини. Ця сторінка, розміщена на CERN (Європейська організація з ядерних досліджень), містила виключно текстову інформацію та посилання на інші документи. У той час веб-дизайн не мав складних візуальних компонентів, а самі веб-сторінки не адаптувалися до розміру екранів, оскільки переважно використовувалися на настільних комп'ютерах з однаковими роздільними здатностями [1].

До середини 1990-х років веб-сайти продовжували бути досить простими і статичними. Вони створювалися з використанням HTML, що був основною мовою розмітки веб-сторінок. Це означало, що вся інформація на веб-сайті, включаючи текст і зображення, мала фіксоване розміщення. Контент не адаптувався до різних екранів або пристроїв, оскільки на той час це не було необхідністю — майже всі користувачі інтернету використовували настільні комп'ютери [1].

Розвиток веб-технологій у 90-х роках привів до появи каскадних таблиць стилів (CSS), що дозволило відокремити візуальне оформлення від структури контенту. Це стало можливим у 1996 році, коли W3C (Консорціум Всесвітньої павутини) затвердив першу версію CSS. CSS дозволив дизайнерам вперше контролювати вигляд сторінок за допомогою стилів, що включали кольори, шрифти, вирівнювання елементів тощо. Однією з основних переваг CSS стала можливість змінювати вигляд всього сайту, редагуючи лише один файл зі стилями,

що значно спрощувало процес редагування веб-сторінок [2].

Проте, макети сторінок залишалися фіксованими і не адаптувалися до різних розмірів екранів. Наприклад, типовий веб-сайт середини 90-х років мав ширину 800 пікселів, оскільки саме таку роздільну здатність мали більшість моніторів. Контент був зафіксований у цих межах, і якщо користувач відкривав сайт на іншому пристрої з іншою роздільною здатністю, макет міг виглядати неправильно або викликати труднощі з переглядом.

На кінець 90-х років, з поширенням інтернету, кількість користувачів зросла до сотень мільйонів. У 1996 році кількість інтернет-користувачів перевищила 100 мільйонів осіб, а до 2000 року ця цифра сягнула понад 400 мільйонів. Зі збільшенням кількості користувачів стали з'являтися нові типи пристроїв, включаючи перші мобільні телефони з доступом до інтернету. Це поставило перед веб-дизайнерами новий виклик: створювати сайти, що будуть зручними для перегляду не лише на настільних комп'ютерах, але й на мобільних пристроях [2].

З появою перших мобільних браузерів на початку 2000-х років стало зрозуміло, що фіксовані макети більше не відповідають потребам сучасного інтернету. Веб-сайти, розроблені для настільних комп'ютерів, були занадто важкими для мобільних телефонів та нечитабельними через малий розмір екранів. Як відповідь на це, з'явилися мобільні версії сайтів, які розроблялися окремо від основних настільних версій. Наприклад, сайт міг мати адресу "m.example.com" для мобільних пристроїв і "www.example.com" для настільних комп'ютерів. Проте підтримка двох версій одного сайту була трудомісткою та неефективною, оскільки вимагала більше ресурсів для розробки та оновлень.

Ця проблема стала ще більш актуальною у 2007 році, коли Apple презентувала iPhone, перший смартфон з повноцінним браузером, що дозволяв переглядати веб-сторінки так само, як на настільному комп'ютері. Це змінило підхід до розробки сайтів, адже фіксовані макети стали застарілими для нових умов використання інтернету на мобільних пристроях [3].

У 2010 році Ітан Маркотт запропонував концепцію адаптивного веб-дизайну (responsive web design), яка дозволяла веб-сайтам автоматично адаптуватися до різних розмірів екранів за допомогою медіазапитів CSS. Завдяки цій технології стало можливим створювати єдиний сайт, який підлаштовувався під будь-який

пристрій, незалежно від його роздільної здатності [3].

У 2015 році з'явилася нова технологія, Flexbox, яка спростила створення складних макетів із гнучким розташуванням елементів. У 2020 році кількість інтернет-користувачів сягнула 4,6 мільярда, і адаптивний веб-дизайн став стандартом для розробки веб-ресурсів. На рис. 1.1. можемо переглянути розвиток кількості користувачів у інтернеті.

Рис. 1.1. – Ріст кількості інтернет-користувачів

Адаптивний веб-дизайн став вирішенням низки проблем, що виникли через зростання різноманітності пристроїв, якими користуються для доступу до веб-сторінок. У 2010 році Ітан Маркотт запропонував концепцію "responsive web design" (адаптивного веб-дизайну), яка кардинально змінила підхід до розробки сайтів. Головною особливістю цього підходу стало використання медіазапитів CSS для автоматичного підлаштування веб-сторінок під різні розміри екранів. Раніше веб-дизайнери часто створювали окремі версії сайтів для мобільних пристроїв та настільних комп'ютерів, що було незручно й затратно [4].

Медіазапити дозволяють змінювати стилі залежно від параметрів пристрою, таких як ширина або висота екрана. Наприклад, при ширині екрана менше ніж 768 пікселів сайт може автоматично перебудуватися так, щоб контент був зручним для перегляду на мобільному пристрої. Важливим інструментом для цього стало використання відносних одиниць виміру, як-от відсотки (%). Замість фіксованих пікселів, що були поширені раніше, елементи сторінки змінюються пропорційно

розміру екрана, забезпечуючи однакову зручність користування незалежно від типу пристрою [5].

Одночасно з поширенням адаптивного веб-дизайну з'явилися потужні фреймворки, такі як Bootstrap і Foundation. Ці інструменти надали розробникам можливість швидко створювати адаптивні сайти, використовуючи гнучкі сітки та готові компоненти, які автоматично підлаштовуються під різні розміри екранів. Bootstrap, наприклад, використовує 12-колонну сітку, яка дозволяє ефективно розподіляти елементи на сторінці і робити їх адаптивними. Ці фреймворки суттєво спростили процес розробки, дозволяючи створювати сайти, що однаково добре виглядають на мобільних телефонах, планшетах та настільних комп'ютерах [6].

Подальший розвиток адаптивного веб-дизайну відбувся завдяки появі таких інструментів, як Flexbox і CSS Grid. Flexbox (2015) дозволив легко вирішувати завдання розташування елементів у рядках і стовпцях. Він автоматично розподіляє елементи сторінки з урахуванням їхнього розміру та відступів, забезпечуючи гнучке розміщення контенту на різних екранах. Це суттєво спростило процес створення складних макетів, які раніше вимагали великої кількості CSS-коду. Відповідно на рис.1.2. можемо переглянути діаграму яка описує вплив різних технологій на адаптивний веб-дизайн.

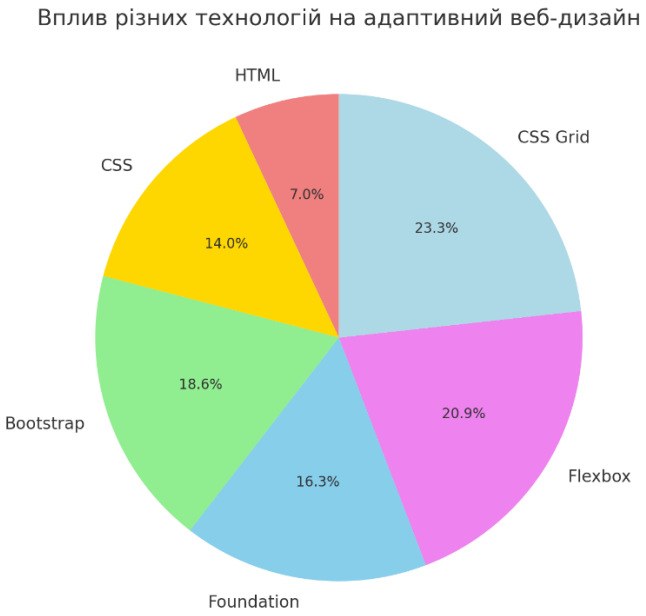

Рис.1.2. – Вплив різних технологій на адаптивний веб-дизайн

CSS Grid (2017) пішов ще далі, запропонувавши систему двовимірних сіток для розміщення елементів на сторінці. CSS Grid дозволяє створювати складні

макети з різноманітними зонами та областями, які можуть легко адаптуватися до зміни розміру екрана. Це дає розробникам ще більший контроль над виглядом і поведінкою елементів сторінки, зокрема, можливість легко переміщувати блоки контенту в залежності від ширини пристрою. Зведену таблицю 1.1. можемо переглянути на яка демонструє розвиток технологій у веб-дизайні [2].

Таблиця 1.1.

Впровадження технологій у веб-дизайн

Технологія	Рік впровадження	Основна функція
HTML	1991	Створення структури сторінки
CSS	1996	Оформлення та стилізація сторінки
Bootstrap	2011	Гнучка сітка та компоненти для адаптивності
Foundation	2011	Гнучка сітка для швидкого створення макетів
Flexbox	2015	Гнучке розташування елементів у рядках та стовпцях
CSS Grid	2017	Двовимірні сітки для складних макетів

Flexbox та CSS Grid стали революційними рішеннями у створенні адаптивних макетів, дозволяючи розробникам уникнути фіксованих розмірів елементів і створювати гнучкі інтерфейси, які підлаштовуються під будь-який екран. Завдяки Flexbox, елементи на сторінці автоматично вирівнюються відповідно до наявного простору, що дозволяє зменшити кількість коду для позиціонування та вирівнювання елементів. Він ефективно використовується для горизонтального або вертикального розміщення елементів і забезпечує динамічний розподіл простору між ними.

CSS Grid, у свою чергу, дозволяє створювати двовимірні макети з використанням як рядків, так і колонок, що значно розширює можливості дизайну. Це рішення дозволяє створювати складні сітки, де кожен елемент може бути точно позиціонований в залежності від розміру екрану або контенту. Наприклад, у великих дисплеях можна використовувати багатосторонні макети, а на мобільних пристроях ці ж елементи можуть перебудовуватися у вертикальні блоки для полегшення перегляду. Flexbox був введений у специфікацію CSS у 2009 році, а

CSS Grid став офіційною частиною стандарту у 2017 році. Ці дати відзначають важливі етапи в еволюції веб-дизайну, кожен з яких значно спростив процес створення адаптивних та гнучких веб-інтерфейсів [2].

Сучасний адаптивний веб-дизайн став необхідністю через збільшення кількості пристроїв з різними розмірами екранів. Зокрема, смартфони і планшети мають набагато менші екрани, ніж настільні комп'ютери, що вимагає використання адаптивних технологій для забезпечення зручного перегляду та навігації.

Одним із прикладів успішного впровадження адаптивних макетів є веб-сайти новин та електронної комерції. У таких випадках важливо, щоб користувачі могли зручно переглядати контент незалежно від того, чи використовують вони мобільний телефон, планшет або настільний комп'ютер.

Для порівняння можемо розглянути діаграму, що показує різницю у поведінці користувачів на різних пристроях.

Таблиця 1.2.

Поведінка користувачів на різних пристроях

Тип пристрою	Середній час на сайті (хвилин)	Середня кількість переглядів сторінок
Настільні комп'ютери	5:30	3.5
Смартфони	4:20	2.8
Планшети	4:50	3.1

З діаграми видно, що користувачі на настільних комп'ютерах мають більше часу на перегляд контенту, але на мобільних пристроях цей час скорочується, і відповідно збільшується важливість ефективності адаптивного дизайну.

Подальший розвиток технологій адаптивного дизайну передбачає ширше використання CSS Grid і Flexbox, а також інтеграцію з JavaScript-бібліотеками, такими як React або Vue.js. Це дозволить створювати ще більш динамічні інтерфейси, які зможуть відповідати на дії користувачів у реальному часі, автоматично адаптуючись до умов, які змінюються (наприклад, розмір екрану, режим перегляду, тощо).

Еволюція веб-дизайну від статичних макетів до сучасних адаптивних рішень є природною відповіддю на зміни у способах споживання контенту. Використання технологій, таких як CSS Grid, Flexbox, Bootstrap, а також новітніх бібліотек JavaScript, забезпечує гнучкість, динамічність і адаптивність сучасних веб-

додатків. У майбутньому можна очікувати ще більшої автоматизації та покращення адаптивних рішень, що дозволить створювати сайти, які зручні у використанні на будь-яких пристроях, незалежно від їх розміру чи параметрів.

1.2. Аналіз основних методів та засобів розробки гнучких макетів

Розробка гнучких макетів є важливою складовою сучасного веб-дизайну, оскільки вона дозволяє веб-сторінкам адаптуватися до різних розмірів екранів та типів пристроїв. У процесі створення таких макетів використовуються різні методи та засоби, які спрощують роботу веб-розробників і забезпечують зручність користувачів. Нижче розглянуто основні методи та засоби розробки гнучких макетів.

Одним із найважливіших методів створення гнучких макетів є використання відносних одиниць виміру. Відносні одиниці виміру дозволяють елементам на веб-сторінці адаптуватися до різних розмірів екранів і вікон браузера, забезпечуючи гнучкість дизайну. Відмінність від статичних одиниць, таких як пікселі, полягає в тому, що пікселі фіксовані і не змінюються при зміні розміру вікна або пристрою, тоді як відносні одиниці автоматично підлаштовуються під розмір контейнера або шрифту. Основними відносними одиницями виміру є відсотки (%), em та rem [10].

Відсотки є найпоширенішою відносною одиницею виміру для визначення ширини і висоти елементів на сторінці. Вони задають розміри елементів як відсоткове співвідношення до розміру батьківського елемента або контейнера. Це дозволяє веб-елементам автоматично змінювати свої розміри відповідно до розмірів вікна браузера або контейнера, в якому вони знаходяться. Приклад можемо розглянути на рис. 1.3.

```
<div>
  Звичайний текст
  <div style="font-size: 100%;">Привіт! Це дипломна робота!
    <div style="font-size: 150%;">Живи, а працюй у вільний час!
      <div>Підписуйся!</div>
      <div style="font-size: 150%;">Став лайк!</div>
      <div style="font-size: 80%;">І коментар напиши!</div>
    </div>
  </div>
</div>
```

Рис.1.3. Приклад відносної одиниці виміру відсотки

На зображенні показаний HTML-код, у якому використовуються елементи , а для кожного з них заданий різний розмір шрифту за допомогою властивості font-size у відсотках. Кожен наступний елемент всередині має більший або менший розмір шрифту, зокрема 100%, 150%, 80% тощо.

Цей підхід є ключовим для забезпечення гнучкого макета, оскільки сторінки автоматично підлаштовуються під різні екрани – від великих настільних моніторів до мобільних телефонів.

Одиниця em використовується для задання відносних розмірів шрифтів, відступів, полів, висоти рядків та інших параметрів. Вона базується на розмірі шрифту батьківського елемента, що означає, що розмір в em залежить від контексту, в якому він використовується. Якщо батьківський елемент має розмір шрифту 16px, то 1em дорівнює 16px . Приклад можемо розглянути на рис.1.4.

```
<div>
  Звичайний текст
  <div style="font-size: 1em;">Привіт! Це дипломна робота!
    <div style="font-size: 1.5em;">Живи, а працюй у вільний час!
      <div>Підписуйся!</div>
      <div style="font-size: 1.5em;">Став лайк!</div>
      <div style="font-size: 0.8em;">І коментар напиши!</div>
    </div>
  </div>
</div>
```

Рис.1.4. – Приклад одиниці em

На зображенні показаний HTML-код із використанням елементів , де для кожного блоку задаються розміри шрифту за допомогою font-size у одиницях em. Внутрішні блоки мають різні значення розмірів шрифту: 1em, 1.5em, 0.8em тощо.

Одиниця em корисна для створення дизайнів, у яких розмір тексту і відступи автоматично змінюються разом із батьківським елементом. Проте варто враховувати, що використання em може призвести до "каскадного ефекту", коли розміри шрифтів стають занадто великими або занадто маленькими [13]

Одиниця rem є подібною до em, але з однією важливою різницею: rem базується на розмірі шрифту кореневого елемента (зазвичай, це тег <html>), а не батьківського елемента. Це робить rem передбачуванішою, оскільки її значення не залежить від вкладеності елементів. Встановлення глобального розміру шрифту для кореневого елемента дозволяє контролювати всі елементи на сторінці через

один параметр. Приклад можемо розглянути на рис.1.5.

```
<div>
  Звичайний текст
  <div style="font-size: 1rem;">Привіт! Це дипломна робота!
    <div style="font-size: 1.5rem;">Живи, а працюй у вільний час!
      <div>Підписуйся!</div>
      <div style="font-size: 1.5rem;">Став лайк!</div>
      <div style="font-size: 0.8rem;">І коментар напиши!</div>
    </div>
  </div>
</div>
```

Рис.1.5. – Приклад одиниці rem

Використання rem є популярним через його стабільність і передбачуваність. Веб-розробники можуть змінювати розмір шрифтів на всій сторінці, змінюючи лише розмір шрифту кореневого елемента. Відповідно представимо усі одиниці виміру у таблиці 1.3.

Таблиця 1.3.

Основні одиниці виміру

Одиниця	**Опис**	**Приклад використання**	**Особливості**
%	Відсоткове співвідношення до батьківського елемента	width: 50%;	Підходить для гнучких макетів
em	Базується на розмірі шрифту батьківського елемента	font-size: 1.5em;	Добре підходить для масштабування тексту
rem	Базується на розмірі шрифту кореневого елемента	font-size: 2rem;	Незалежно від контексту елемента
vw	Відсоток від ширини вікна браузера	width: 50vw;	Підлаштовується під ширину вікна
vh	Відсоток від висоти вікна браузера	height: 100vh;	Корисно для встановлення висоти елементів
vmin	Відсоток від найменшого значення між vw і vh	font-size: 2vmin;	Використовується для адаптивних шрифтів
vmax	Відсоток від найбільшого значення між vw і vh	font-size: 2vmax;	Зручний для елементів, що повинні масштабуватися до найбільшого параметру вікна
fr	Одиниця виміру для моделі Grid, означає частку простору	grid-template-columns: 1fr 2fr;	Частка від доступного простору у CSS Grid

Відносні одиниці виміру, такі як **em**, **rem**, **vw**, та **vh**, забезпечують гнучкість і

адаптивність веб-дизайну, дозволяючи елементам автоматично підлаштовуватися під різні розміри екранів. Це дає змогу створювати макети, які добре виглядають на будь-яких пристроях і підтримують масштабованість, зберігаючи узгодженість елементів. Однак використання **em** може призвести до каскадних змін у розмірах елементів, що ускладнює контроль за зовнішнім виглядом. Також одиниці **vw** та **vh** можуть викликати проблеми з точністю масштабування на різних інтерфейсах, особливо при зміні розміру вікна. Надмірне використання цих одиниць може зробити дизайн складним для підтримки.

Медіазапити — це один із найважливіших інструментів у сучасному веб-дизайні, який дозволяє створювати адаптивні макети. Адаптивні макети гарантують, що веб-сторінки коректно відображаються на різних пристроях, зокрема на мобільних телефонах, планшетах, ноутбуках і настільних комп'ютерах. За допомогою медіазапитів можна визначити, як повинні змінюватися стилі CSS залежно від характеристик пристрою користувача, таких як:

- ширина або висота вікна браузера.
- Роздільна здатність екрана (наприклад, кількість пікселів на дюйм).
- Орієнтація пристрою (портретна або ландшафтна).
- Співвідношення сторін (aspect ratio).
- Можливості користувача, наприклад, наявність сенсорного екрану або стилуса.

Медіазапити дозволяють визначати стилі для різних умов, що значно полегшує розробку адаптивних сайтів. Зокрема, можна:

1. застосовувати різні стилі CSS для різних пристроїв.
2. Оптимізувати веб-дизайн для екранів різних розмірів, зокрема мобільних пристроїв, планшетів і настільних комп'ютерів.
3. Покращити користувацький досвід шляхом адаптації макета під різні умови роботи (наприклад, висока або низька роздільна здатність, можливості пристрою тощо).

Медіазапити часто використовують для визначення стилів, які змінюються залежно від ширини вікна браузера. Наприклад, можна змінювати ширину контейнерів, розміри шрифтів або елементів інтерфейсу. На рис.1.6. можемо розглянути приклад

```css
/* Стилі для настільних пристроїв */
@media (min-width: 1024px) {
  .container {
    width: 80%;
    margin: 0 auto;
  }
}

/* Стилі для планшетів */
@media (max-width: 1024px) and (min-width: 768px) {
  .container {
    width: 90%;
    padding: 20px;
  }
}

/* Стилі для мобільних пристроїв */
@media (max-width: 768px) {
  .container {
    width: 100%;
    padding: 10px;
  }
}
```

Рис.1.6. – Приклад використання медіазапитів

Цей CSS-код використовує медіазапити для адаптації контейнера на різних пристроях. Для настільних комп'ютерів, коли ширина екрану більше 1024 пікселів, контейнер займає 80% ширини екрану і центрований за допомогою відступу (margin). На планшетах, при ширині від 768 до 1024 пікселів, контейнер розширюється до 90% і додається внутрішній відступ у 20 пікселів. На мобільних пристроях, коли ширина менше 768 пікселів, контейнер займає всю ширину екрана (100%) з меншим внутрішнім відступом у 10 пікселів.

Використання медіазапитів у веб-розробці має кілька важливих переваг. Вони дозволяють адаптувати сайти для різних пристроїв, що забезпечує якісний користувацький досвід на великих екранах і мобільних пристроях. Це зменшує витрати на розробку, оскільки немає потреби створювати окремі версії сайту для кожного типу пристроїв. Медіазапити також дозволяють оптимізувати контент, забезпечуючи ефективне завантаження ресурсів, зокрема зображень та інших елементів, для кожного типу екранів. Однак є й недоліки. Медіазапити ускладнюють тестування, оскільки необхідно перевіряти сайт на різних пристроях, що вимагає додаткових зусиль. Крім того, занадто велика кількість медіазапитів може ускладнити підтримку проекту, а покриття всіх можливих екранів іноді може бути неповним, що призводить до некоректного відображення на деяких пристроя

Також варто розглянути гнучкі сітки (Grid Layouts) є потужним інструментом у сучасному веб-дизайні для створення складних макетів, що автоматично

адаптуються до різних розмірів екранів і забезпечують високий рівень контролю над розміщенням елементів. Існують два ключові підходи до створення гнучких сіток: CSS Grid Layout та Flexbox [5].

CSS Grid — це система двовимірних сіток, яка дозволяє розробникам створювати складні й гнучкі макети для веб-сторінок. Вона забезпечує контроль над розміщенням елементів як у горизонтальному, так і у вертикальному напрямках. CSS Grid дозволяє використовувати точні координати для елементів і легко змінювати розміри та положення залежно від доступного простору. Одна з основних переваг CSS Grid полягає в тому, що можна створювати різні макети для різних розмірів екранів, просто змінюючи налаштування сітки. Приклад можемо розглянути на рис.1.7.

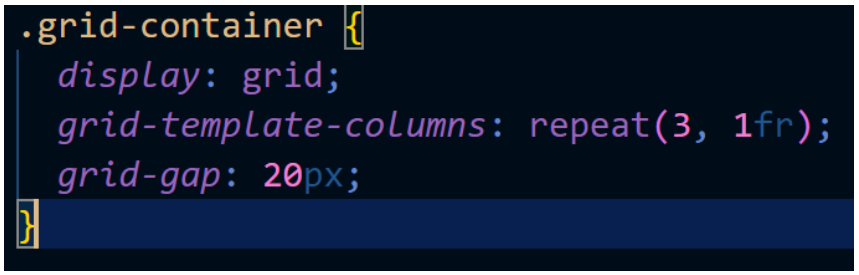

Рис.1.7. – Приклад використання CSS Grid

Цей код створює сітку з трьох стовпців, де кожен стовпець займає рівну частину доступного простору. Властивість grid-gap додає 20 пікселів відступу між стовпцями. CSS Grid дозволяє створювати динамічні сітки, які легко адаптуються під будь-які розміри екранів.

Flexbox — це ще один важливий інструмент для розробки гнучких макетів, що використовується для одновимірних макетів, тобто розташування елементів у рядках або стовпцях. Flexbox дозволяє контролювати розміри та положення елементів, рівномірно розподіляючи простір між ними. Його основною перевагою є гнучкість у вирівнюванні елементів, що робить його ідеальним для вирішення завдань, пов'язаних з позиціонуванням елементів у межах одного виміру (ряд або стовпець). На рис.1.8. можемо розглянути приклад.

```
.flex-container {
  display: flex;
  justify-content: space-between;
}
```

Рис.1.8. – Приклад використання Flexbox

Цей код налаштовує контейнер як гнучкий блок, а властивість justify-content: space-between розподіляє простір рівномірно між дочірніми елементами. Flexbox зручний для випадків, коли потрібно створити адаптивні макети для елементів, які мають змінювати свій розмір у відповідь на зміни розміру вікна браузера або екрана пристрою.

Відмінності між CSS Grid та Flexbox

1. Одновимірність проти двовимірності: Flexbox найкраще підходить для управління елементами в одному вимірі (ряд або стовпець), тоді як CSS Grid дозволяє контролювати макет у двох вимірах (як у рядках, так і в стовпцях).

2. Гнучкість: Flexbox забезпечує автоматичне розміщення та вирівнювання елементів у межах доступного простору, ідеальний для вирівнювання елементів у рядках або стовпцях. CSS Grid, у свою чергу, дозволяє більш точно контролювати розташування елементів у сітці за допомогою координат.

3. Складність макетів: для складних багаторівневих макетів із великою кількістю елементів CSS Grid забезпечує більше можливостей і є більш потужним інструментом. Flexbox ж найкраще підходить для простіших макетів і вирівнювання.

Bootstrap і Foundation — це два популярні CSS-фреймворки, які використовуються для створення адаптивних веб-дизайнів. Обидва фреймворки надають набір готових компонентів і сіток, що дозволяють розробникам легко створювати макети, які автоматично підлаштовуються під різні розміри екранів, від мобільних пристроїв до настільних комп'ютерів. Вони значно спрощують процес розробки, оскільки містять вже готові класи і медіазапити для різних пристроїв, що прискорює розробку та зменшує кількість коду, який потрібно писати вручну.

Bootstrap є одним із найбільш використовуваних фреймворків у світі веб-розробки. Він використовує 12-колонну сітку, яка дозволяє створювати адаптивні макети за допомогою класів, таких як .col- з різними префіксами для різних розмірів екранів (XS, SM, MD, LG, XL). Це дозволяє дизайнерам легко налаштовувати ширину колонок залежно від розміру екрана, що робить сайт зручним для користувачів на будь-якому пристрої.

Bootstrap також надає ряд компонентів, таких як навігаційні меню, кнопки, форми, каруселі та багато інших. Всі ці компоненти вже налаштовані для адаптивної роботи, що дозволяє швидко створювати сучасні веб-сайти без необхідності писати багато додаткового CSS-коду.

Foundation, подібно до Bootstrap, пропонує адаптивну сітку та компоненти. Його гнучка сітка дозволяє створювати макети, які автоматично підлаштовуються під різні розміри екранів. Foundation більше орієнтується на швидкість розробки та простоту використання. Він також надає компоненти, схожі на Bootstrap, але робить більший акцент на кастомізації, що дає розробникам більше можливостей для налаштування зовнішнього вигляду компонентів відповідно до конкретних потреб проекту. Зведену таблицю 1.4. порівняння розглянемо далі

Таблиця 1.4.

Таблиця порівняння Bootstrap і Foundation

Фреймворк	**Особливості**	**Переваги**	**Недоліки**
Bootstrap	12-колонна сітка, готові компоненти, медіазапити	Легко використовувати, популярний, велика спільнота	Менше можливостей для кастомізації
Foundation	Гнучка сітка, акцент на кастомізації	Більше можливостей для налаштування, швидкість розробки	Менш популярний, менша спільнота

Важливим аспектом адаптивного дизайну є правильне використання зображень і медіа. Для забезпечення швидкого завантаження сторінки та оптимальної якості зображень на різних пристроях використовуються відносні одиниці або спеціальні формати, такі як srcset. Цей підхід дозволяє браузеру вибрати найбільш відповідне зображення залежно від розміру екрана користувача. Це забезпечує ефективну роботу сайту навіть на пристроях із різними роздільними

здатностями, що дозволяє покращити користувацький досвід і зменшити час завантаження сторінки.

Використання фреймворків, таких як Bootstrap і Foundation, значно спрощує процес створення адаптивних веб-сторінок. Вони пропонують готові рішення для створення адаптивних макетів та компоненти, які легко налаштовуються під різні екрани. Крім того, використання медіазапитів, гнучких сіток та відносних зображень дозволяє оптимізувати сайт для різних пристроїв, забезпечуючи комфортний користувацький досвід і ефективну роботу веб-ресурсу.

1.3. Вибір засобів розробки та технологій

У сучасній веб-розробці поєднання технологій, таких як CSS Grid, Flexbox, React, JavaScript (JS) та HTML, є ефективним інструментом для створення адаптивних, гнучких і масштабованих інтерфейсів користувача. Кожна з цих технологій відіграє певну роль у побудові веб-додатків, забезпечуючи високу гнучкість та контроль над компонентами та макетами.

CSS Grid — це двовимірна система макетування, яка дозволяє розробникам створювати адаптивні веб-дизайни з використанням як рядків, так і колонок. Ця технологія надає можливість проектувати складні макети, які легко пристосовуються до різних розмірів екранів.

У системі CSS Grid існує кілька ключових понять. Контейнер сітки — це елемент, який містить структуру сітки, а його дочірні елементи називаються елементами сітки. Лінії сітки розділяють сітку на горизонтальні та вертикальні частини, які формують треки сітки — це простори між двома лініями сітки, що створюють ряди або колонки. Область сітки — це прямокутний простір всередині сітки, обмежений чотирма лініями сітки. Приклади сіток можна розглянути на рис. 1.9.

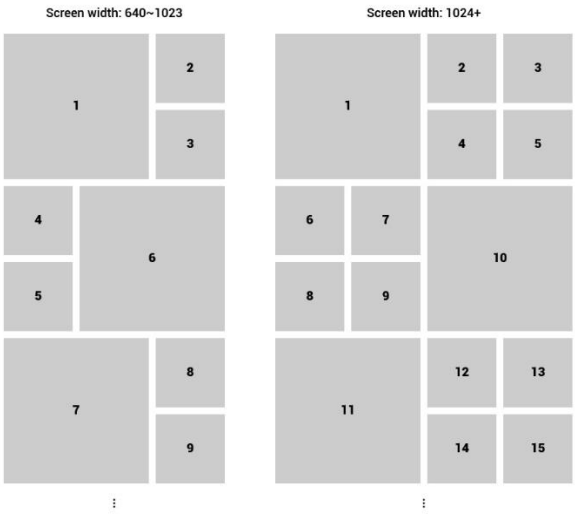

Рис.1.9. – Сітки CSS Grid

Сітка може бути явною або неявною. Явна сітка — це та, що створюється розробником з чітко визначеними рядками і колонками, в той час як неявна сітка створюється автоматично, коли вміст перевищує кількість визначених рядків або колонок. Крім того, в CSS Grid існує можливість встановлювати відступи між рядами та колонками, що називається прогалинами сітки.

Flexbox — це одновимірна система для створення гнучких макетів у веб-дизайні. Вона оптимізована для вирівнювання елементів у рядок або стовпець, забезпечуючи простий і зручний контроль за їхнім розташуванням, навіть якщо розміри елементів змінюються динамічно. Приклад можемо розглянути на рис.1.10.

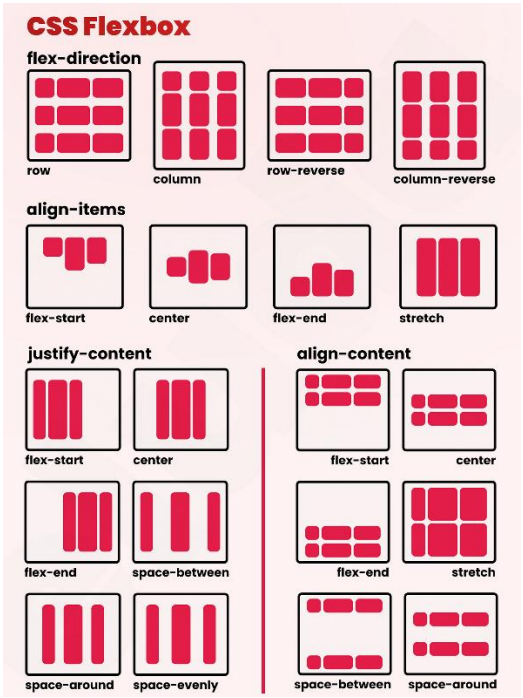

Рис.1.10. – CSS Flexbox приклад

Контейнер Flexbox дозволяє створювати контекст, у якому розташовані елементи Flexbox. Ці елементи можуть вирівнюватися в одному напрямку — горизонтально або вертикально, завдяки чому їх можна легко розміщувати та масштабувати в межах контейнера. Однією з ключових особливостей Flexbox є здатність автоматично розподіляти доступний простір між елементами, вирівнюючи їх рівномірно або у відповідності до визначених параметрів.

Flexbox надає гнучкість у керуванні відстанями між елементами, вирівнюванням і порядком елементів, що дозволяє легко адаптувати макети під різні розміри екранів. Порівняльну таблицю 1.4. технологій розглянемо далі

Таблиця 1.4.

Порівняльна таблиця між CSS Grid і Flexbox

Характеристика	CSS Grid	Flexbox
Вимірність	Двовимірна (колонки та рядки)	Одновимірна (тільки рядок або колонка)
Вирівнювання	Легке вирівнювання елементів у двох вимірах	Гнучке вирівнювання в одному вимірі
Розподіл простору	Може створювати явні та неявні сітки	Автоматично розподіляє простір між елементами
Гнучкість	Висока для створення складних макетів	Оптимізована для вирівнювання у рядок/стовпець
Користувацький контроль	Високий контроль над усіма елементами макету	Просте керування вирівнюванням і порядком

JavaScript (JS) — це динамічна мова програмування, яка використовується для додавання інтерактивності до веб-сайтів і веб-додатків. Вона підтримується всіма сучасними браузерами і є однією з ключових технологій для створення інтерактивних інтерфейсів користувача поряд з HTML і CSS. Логотип JS можемо переглянути на рис.1.11.

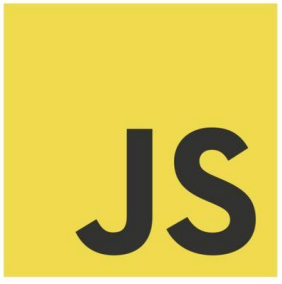

Рис.1.11. – Логотип мови програмування JavaScript

Основні властивості JavaScript:

1. Кросплатформенність. JS підтримується усіма браузерами та операційними системами, що робить його універсальним для веб-розробки.

2. Динамічність. JS дозволяє змінювати контент на веб-сторінках без перезавантаження сторінки. Це досягається за допомогою технології AJAX (Asynchronous JavaScript and XML), яка дозволяє виконувати запити на сервер у фоновому режимі.

3. Підтримка подій. JavaScript може реагувати на різноманітні події, такі як кліки, прокрутка сторінки, введення даних у форму тощо. Це забезпечує високу інтерактивність і взаємодію користувача з веб-додатком.

4. Об’єктно-орієнтованість. Хоча JS не є класичною об’єктно-орієнтованою мовою, він використовує об’єкти для представлення даних і створення більш складних конструкцій.

5. Асинхронне програмування. JavaScript підтримує асинхронні виклики, завдяки чому операції, які займають багато часу (наприклад, завантаження даних з сервера), не блокують виконання інших частин коду. Для цього застосовуються функції зворотного виклику (callbacks), обіцянки (promises) та асинхронні функції (async/await).

6. Інтеграція з HTML та CSS. JS тісно інтегрований з HTML і CSS, що дозволяє змінювати структуру, вигляд та поведінку елементів на сторінці в реальному часі.

JS є важливою складовою сучасних веб-додатків завдяки його здатності динамічно змінювати вміст сторінки, керувати мультимедіа, а також обробляти користувацькі дії в режимі реального часу. Він використовується для валідації форм, створення анімацій, інтерактивних карт, динамічних меню та інших елементів, які покращують користувацький досвід.

Окрім браузерної сторони, JavaScript використовується і на серверній стороні за допомогою Node.js — платформи, яка дозволяє виконувати JavaScript поза браузером, що розширює можливості для розробки серверів та бекенду.

React — це популярна JavaScript-бібліотека для створення інтерфейсів користувача. Вона була створена Facebook і призначена для побудови компонентів, які можна повторно використовувати, що полегшує управління складними інтерфейсами у великих веб-додатках. На рис. 1.11. можемо переглянути логотип

фреймворку.

Рис.1.11. – Логотип фреймворку React

Основні концепції React:

1. Компонентна архітектура. Вся структура інтерфейсу в React складається з компонентів — незалежних блоків коду, які можуть мати свою логіку і вигляд. Кожен компонент може бути багаторазово використаний на сторінці. Компоненти можуть бути як функціональними, так і класовими.

2. Віртуальний DOM. React використовує віртуальний DOM (Document Object Model) для оптимізації оновлення сторінок. При зміні стану компонентів, React створює копію реального DOM у пам’яті та оновлює тільки ті частини інтерфейсу, які змінилися, замість повного перерендерингу сторінки. Це значно підвищує продуктивність додатків.

3. Одностороння передача даних. У React дані передаються від батьківського компонента до дочірніх через властивості (props). Це забезпечує чітку структуру та прогнозованість поведінки додатка.

4. JSX (JavaScript XML). React використовує синтаксис JSX, який дозволяє розробникам писати HTML-подібний код безпосередньо в JavaScript. Це спрощує розробку компонентів і робить код більш читабельним.

5. Декларативний підхід. React дозволяє описувати, як інтерфейс повинен виглядати в залежності від стану компонента, а не управляти безпосередньо маніпуляціями з DOM. Це робить код більш передбачуваним і полегшує обслуговування великих додатків.

React-компоненти мають певний життєвий цикл, що включає в себе стадії їхнього створення, оновлення та знищення. Це дозволяє виконувати певні дії на різних етапах існування компонента, наприклад, завантажувати дані або очищати ресурси перед знищенням компонента.

1. Монтування — етап, коли компонент з’являється на сторінці.
2. Оновлення — етап, коли компонент перерендерюється у відповідь на зміну стану або отримання нових властивостей.
3. Демонтаж — етап, коли компонент видаляється зі сторінки.

Ключовим аспектом у React є стан компонентів. Стан — це набір даних, що змінюються протягом часу. При зміні стану відбувається перерендеринг компонента, і інтерфейс оновлюється відповідно до нового стану. Для управління станом використовуються хуки, такі як useState та useEffect, що дозволяють керувати змінами в функціональних компонентах.

HTML (HyperText Markup Language) — це стандартна мова розмітки для створення веб-сторінок. Вона використовується для структурування контенту на веб-сторінках, задаючи їм певний формат і визначаючи, як елементи будуть відображатися в браузері. HTML складається з тегів, які дозволяють додавати текст, зображення, посилання, таблиці, форми та інші елементи.

Кожен HTML-документ має стандартну структуру, що починається з оголошення типу документа (<!DOCTYPE html>) і містить такі основні елементи, як <html>, <head> і <body>. В розділі <head> розміщуються метадані, включаючи заголовки сторінки, підключення стилів та скриптів, а в <body> — основний вміст сторінки.

Ключовими елементами HTML є заголовки (<h1> - <h6>), параграфи (<p>), списки (<ul>, <ol>), посилання (<a>), зображення (<img>) та форми для збору даних від користувача. HTML забезпечує базову структуру, до якої за допомогою CSS і JavaScript додається стиль та інтерактивність, що робить її основою для всіх веб-додатків та сайтів.

У роботі буде використовуватися середовище розробки Visual Studio Code. Це потужний та зручний інструмент, який дозволить ефективно працювати з різними мовами програмування та технологіями, необхідними для розробки веб-додатку. Основна перевага Visual Studio Code полягає в його гнучкості та можливості підключати різноманітні розширення, що значно полегшить роботу з такими технологіями, як HTML, CSS, JavaScript та React. Інтерфейс можемо переглянути на рис. 1.12.

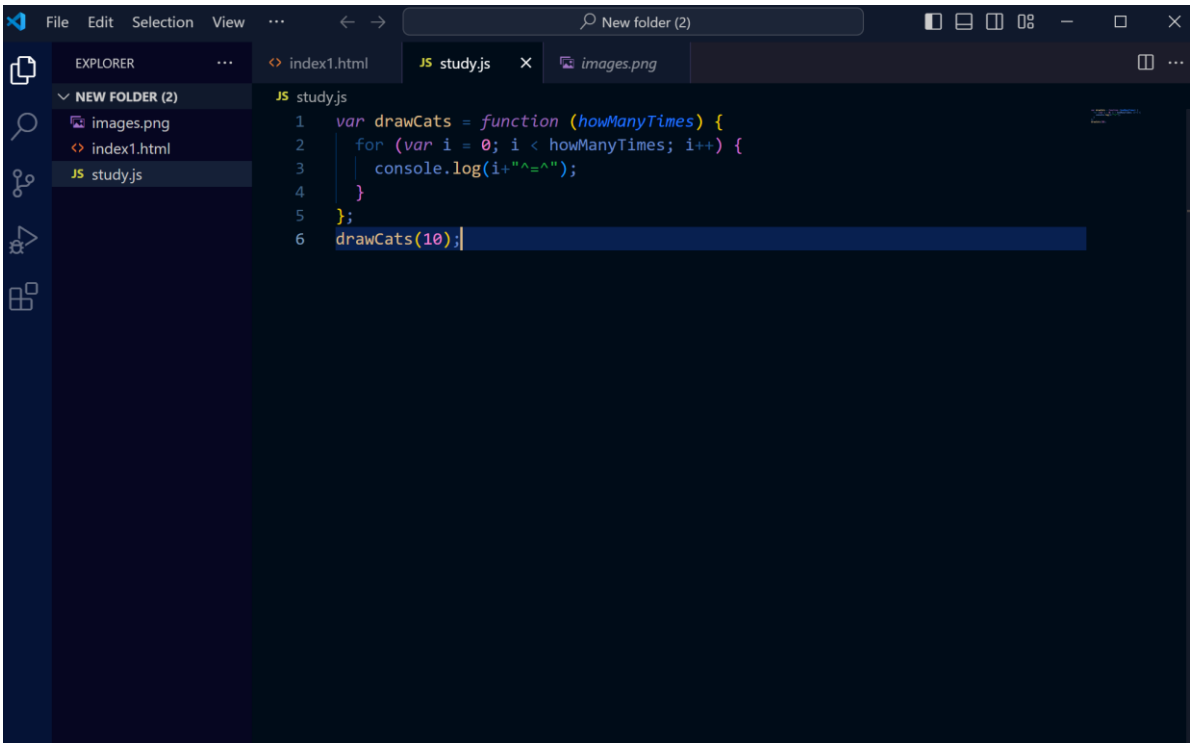

Рис.1.12. – Інтерфейс програмного середовища

Однією з важливих функцій Visual Studio Code є інтеграція з Git, що дозволить вести контроль версій коду. Ми зможемо легко відстежувати всі зміни в проекті, працювати з гілками, а також синхронізувати роботу з віддаленими репозиторіями, що сприятиме зручній командній роботі.

Також у Visual Studio Code є підтримка розширень для React, які забезпечать швидке автодоповнення коду та полегшать роботу з JSX. Це дозволить спростити процес створення компонентів та структурування проекту.

Крім того, завдяки розширенню Live Server, ми зможемо миттєво переглядати всі зміни в браузері під час розробки. Це буде особливо корисним для швидкої перевірки верстки та функціоналу веб-сторінок.

Отже, використання Visual Studio Code дозволить ефективно та зручно працювати над нашим проектом, забезпечуючи необхідні інструменти для розробки, налагодження та контролю версій коду.

1.4. Висновки до першого розділу

У першому розділі було розглянуто теоретичні основи створення гнучких та адаптивних макетів на веб-сайтах, що є невід’ємною частиною сучасної веб-розробки. Аналіз показав, що веб-дизайн пройшов значну еволюцію: від статичних сторінок з фіксованими макетами до інтерактивних, адаптивних рішень, що

змінюються в залежності від розміру екрану, орієнтації пристрою та інших параметрів. Цей розвиток став можливим завдяки впровадженню нових підходів та інструментів, що дозволяють розробникам створювати інтерфейси, які однаково добре працюють на різних платформах — від мобільних пристроїв до настільних комп’ютерів.

Основна увага була приділена аналізу сучасних методів та засобів розробки гнучких макетів. Використання таких технологій, як CSS Grid та Flexbox, дозволяє будувати складні сіткові структури, які легко адаптуються до змін екрану та вмісту. CSS Grid забезпечує двовимірне розташування елементів (як по вертикалі, так і по горизонталі), тоді як Flexbox оптимізовано для одновимірних макетів. Ці інструменти дозволяють розробникам точно контролювати розташування і пропорції елементів, що є важливим для створення адаптивних інтерфейсів.

Крім того, було розглянуто роль медіа-запитів (media queries) як одного з ключових інструментів для адаптації веб-сторінок під різні розміри екранів. Медіа-запити дозволяють змінювати стилі в залежності від умов, таких як ширина вікна браузера або орієнтація екрану. Це забезпечує можливість створювати макети, які підлаштовуються під конкретні параметри пристрою, зберігаючи при цьому зручність користування.

Не менш важливим було дослідження ролі фреймворків, таких як Bootstrap та Foundation, які значно спрощують процес розробки адаптивних макетів. Ці фреймворки надають готові сіткові системи та компоненти інтерфейсу, що дозволяє розробникам зосередитися на функціональності сайту, а не витрачати час на створення базової структури з нуля. Використання таких інструментів сприяє стандартизації підходів до розробки та прискорює процес створення веб-додатків.

Ще одним важливим аспектом є використання бібліотек JavaScript, таких як React, для створення адаптивних та інтерактивних компонентів. React дозволяє динамічно змінювати структуру сторінки в залежності від розміру екрану або інших змінних. Завдяки компонентній архітектурі React, кожен елемент інтерфейсу можна окремо контролювати та адаптувати під різні пристрої.

Таким чином, у цьому розділі було доведено важливість комплексного підходу до розробки адаптивних макетів, який включає використання різних технологій та інструментів для досягнення високої якості інтерфейсів. Технології

CSS Grid, Flexbox, медіа-запити, а також популярні фреймворки та бібліотеки JavaScript є важливими компонентами сучасної веб-розробки, що дозволяють створювати сайти, які є не лише функціональними, але й зручними у користуванні на будь-яких пристроях.

У підсумку, вибір правильних засобів та технологій для створення адаптивних макетів дозволяє забезпечити високу продуктивність, гнучкість та зручність веб-додатків, що є ключовими факторами успішної веб-розробки у сучасному світі.

РОЗДІЛ 2. ПРОЕКТУВАННЯ АЛГОРИТМІЧНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ РОЗРОБКИ МАКЕТІВ

2.1. Теоретичний аналіз гнучких та адаптивних макетів

Теоретичний аналіз гнучких та адаптивних макетів у веб-дизайні зосереджується на принципах і технологіях, які забезпечують комфортний та ефективний перегляд веб-сайтів на різних пристроях, таких як комп’ютери, планшети, смартфони та інші гаджети з різними розмірами екранів. Розвиток мобільних технологій та різноманіття пристроїв вимагав від веб-дизайнерів розробки нових підходів до створення інтерфейсів, що автоматично адаптуються під конкретний розмір і роздільну здатність екрану, щоб зберегти зручність користування.

Гнучкі сітки є фундаментом сучасного адаптивного веб-дизайну, де замість фіксованих одиниць виміру, таких як пікселі, застосовуються відносні одиниці, що дозволяє забезпечити гнучкість макету та його адаптацію до різних розмірів екрану.

Відсоткові значення застосовуються для визначення ширини елементів відносно батьківського контейнера. Наприклад, якщо елемент має ширину 50%, він займе рівно половину ширини контейнера, незалежно від того, чи це 1200 пікселів на настільному комп’ютері або 400 пікселів на мобільному пристрої.

Для прикладу, розглянемо ситуацію, коли ширина вікна браузера становить 1000 пікселів:

- Елемент з шириною 25% займе 250 пікселів (1000 * 0.25 = 250).
- Якщо вікно зменшити до 600 пікселів, той самий елемент автоматично стане 150 пікселів завширшки (600 * 0.25 = 150).

Одиниці EM і REM дозволяють задавати розміри елементів у пропорції до розміру шрифту. Це особливо корисно для створення масштабованих компонентів на веб-сторінці.

1. EM залежить від розміру шрифту батьківського елемента. Наприклад, якщо батьківський елемент має шрифт розміром 16 пікселів, а дочірній елемент встановлений на 1.5em, його розмір становитиме 24 пікселі (16 * 1.5 = 24).

2. REM базується на розмірі шрифту кореневого елемента документа (зазвичай html). Якщо розмір шрифту на рівні html встановлено як 16 пікселів, то

елемент з шириною 2rem завжди буде дорівнювати 32 пікселі (16 * 2 = 32), незалежно від батьківських елементів.

- Якщо кореневий елемент має розмір шрифту 16 пікселів:
  - Елемент, що має 1rem, матиме розмір 16 пікселів (1 * 16 = 16).
  - Елемент з розміром 1.5rem матиме 24 пікселі (1.5 * 16 = 24).

Уявімо, що шрифт кореневого елемента збільшився до 20 пікселів:

- Елемент з 1rem тепер буде 20 пікселів (1 * 20 = 20).
- Елемент з 1.5rem дорівнюватиме 30 пікселям (1.5 * 20 = 30).

У випадку, якщо для веб-дизайну використовуються відсотки для ширини колонок, а EM і REM для шрифтів і відступів, це дозволяє веб-сторінці адаптуватися до різних екранів, не втрачаючи своєї логіки або зручності у використанні.

Така комбінація гнучких сіток і відносних одиниць виміру дозволяє легко масштабувати веб-дизайн для будь-яких пристроїв, незалежно від того, чи це екран у 1200 пікселів, чи мобільний пристрій з шириною 320 пікселів.

Адаптивний контент є одним з ключових елементів сучасного веб-дизайну, який забезпечує комфортний перегляд веб-сайтів на різних пристроях. Важливо, щоб контент автоматично підлаштовувався під розміри екрану користувача, щоб покращити зручність використання та оптимізувати швидкість завантаження. Особливо це важливо для мобільних пристроїв, де обмежені ресурси можуть впливати на швидкість завантаження та продуктивність.

Оптимізація зображень для різних пристроїв відіграє ключову роль у прискоренні завантаження веб-сторінок. Для мобільних користувачів рекомендується використовувати менші за розміром зображення, що зменшує навантаження на трафік і прискорює час відкриття сторінки. Для десктопів і планшетів, де більші екрани можуть підтримувати вищу роздільну здатність, важливо використовувати відповідні розміри зображень.

Наприклад:

- на смартфонах розмір зображення може варіюватися від 300 до 500 пікселів для швидкого завантаження.
- Для планшетів варто використовувати зображення з розміром 700-900 пікселів, що забезпечує баланс між якістю та швидкістю.

- На десктопах оптимально використовувати зображення з розміром понад 1000 пікселів, оскільки екран дозволяє відобразити контент у високій якості.

Важливим аспектом адаптивного контенту є можливість змінювати макет сторінки залежно від типу пристрою.

- На мобільних пристроях макет може автоматично змінюватися для полегшеного перегляду, з використанням спрощених текстів та великих кнопок.
- Для планшетів варто додавати більше інтерактивного контенту, такого як анімації, слайдери або таблиці.
- Для десктопів відображаються всі мультимедійні елементи (відео, анімація, великі зображення).

Таблицю 2.1. контенту на різних пристроях можна розглянути далі.

Таблиця 2.1. **Таблиця контенту для різних пристроїв**

Пристрій	Тип контенту	Рекомендований розмір зображень	Шрифти та заголовки	Інтерактивні елементи	Мультимедіа
Смартфони	Спрощений текст, великі кнопки, мінімум зображень	300-500px	Основний текст: 16-18px Заголовки: 24-32px	Кнопки великих розмірів, адаптовані для сенсорних екранів	Зазвичай відсутні або обмежені до мінімальних GIF чи анімацій
Планшети	Середній текст, інтерактивні елементи, більш складні компоненти	700-900px	Основний текст: 18-20px Заголовки: 32-40px	Інтерактивні компоненти, наприклад, слайдери, таблиці, форми	Підтримуються GIF-анімації, прості відео, інтерактивні графіки
Десктопи	Повний контент з мультимедіа, великі зображення, відео	1000px і більше	Основний текст: 20-24px Заголовки: 40-48px	Повний набір інтерактивних елементів (слайдери, таблиці, форми, анімації)	Підтримуються повноекранні відео, складні інтерактивні елементи, анімації високої якості

На смартфонах контент спрощується для забезпечення швидкого завантаження і легкості використання. Смартфони мають менші екрани і зазвичай використовуються на ходу, тому користувачі потребують швидкого доступу до важливої інформації. Це означає, що дизайн має бути мінімалістичним, зосередженим на найважливіших елементах.

Тип контенту: Для мобільних пристроїв основний акцент робиться на великий текст, який легко читати навіть на маленьких екранах, та великі кнопки, що дозволяють зручно натискати їх на сенсорних екранах. Навігація повинна бути спрощеною, без надлишкових елементів, щоб не відволікати користувача. Мультимедійні елементи використовуються дуже обмежено, з акцентом на економію трафіку і швидке завантаження.

Розмір зображень: Оптимальні розміри зображень для смартфонів становлять 300-500 пікселів. Це дозволяє швидко завантажувати сторінку, зберігаючи при цьому базову якість зображень, які добре виглядатимуть на маленьких екранах. Всі зображення автоматично підлаштовуються під розмір екрану, щоб не створювати горизонтальної прокрутки і забезпечити максимальну зручність.

На планшетах можливості для відображення контенту розширюються завдяки більшим екранам, тому тут можна використовувати більше інтерактивних елементів і складніший контент. Планшети зазвичай використовуються для перегляду мультимедійного контенту та для більш глибокого взаємодії з веб-сайтами, тому важливо адаптувати контент під цей формат.

Тип контенту: Для планшетів можна використовувати складніші макети, інтерактивні елементи, такі як слайдери або таблиці. Текст також може бути детальнішим, оскільки екран дозволяє відображати більше інформації. Окрім того, на планшетах зручно використовувати розділення сторінок на декілька колонок або панелей, що покращує навігацію.

Розмір зображень: Рекомендований розмір зображень для планшетів становить 700-900 пікселів. Це дозволяє зберегти якість зображень при перегляді на більших екранах, не впливаючи значно на швидкість завантаження. Також планшети можуть відображати більше деталей, тому варто використовувати зображення високої якості.

На десктопах контент може бути найскладнішим і найповнішим, оскільки великі екрани надають можливість відображати мультимедіа, відео та анімацію в найкращій якості. Користувачі десктопів зазвичай мають доступ до швидкого інтернету, тому можна використовувати всі можливості сучасного веб-дизайну.

Тип контенту: Десктопи дозволяють відображати повний контент веб-сторінки. Це можуть бути великі мультимедійні файли, складні інтерактивні елементи, багатосторінкові структури та насичений текстовий контент. Крім того, тут можна реалізовувати багатосторінкові макети, інтегрувати різноманітні анімації та відео. Навігація також може бути більш складною і багаторівневою.

Розмір зображень: Для десктопів рекомендується використовувати зображення з розміром 1000 пікселів і більше. Великі екрани дозволяють відображати високоякісні зображення без втрати деталей, що підвищує візуальну привабливість сайту. Крім того, для десктопів можна використовувати повноекранне відео та складні інтерактивні графіки.

Адаптивний контент забезпечує гнучке відображення сторінок на різних пристроях, що значно покращує користувацький досвід. Для досягнення цього важливо оптимізувати зображення, зменшуючи їх розмір на мобільних пристроях і використовуючи великі зображення на десктопах. Контент також повинен динамічно змінюватися залежно від розміру екрану: для смартфонів — спрощення, для планшетів — додаткові інтерактивні елементи, а для десктопів — повний мультимедійний контент.

Основна мета адаптивного контенту полягає у тому, щоб забезпечити зручний доступ до веб-сайтів для користувачів різних пристроїв. Завдяки використанню різних підходів до відображення контенту на смартфонах, планшетах і десктопах, можна досягти балансу між швидкістю завантаження і якістю користувацького досвіду.

Медіа-запити (Media Queries) — це інструмент, що дозволяє веб-дизайнерам адаптувати веб-сторінки до різних умов перегляду, таких як розмір екрана, орієнтація пристрою чи щільність пікселів на екрані. Вони є ключовим елементом сучасного адаптивного дизайну, дозволяючи змінювати стилі сторінок залежно від характеристик пристрою, яким користується відвідувач сайту. Це дозволяє створювати веб-ресурси, що виглядають однаково привабливо і функціонально на

смартфонах, планшетах, ноутбуках та великих моніторах.

Медіа-запити дозволяють, наприклад, змінювати розміри шрифтів, макети сторінки або зображення залежно від ширини екрана. Наприклад, медіа-запит із параметром **max-width: 768px** означає, що певні стилі будуть застосовуватися лише на пристроях, ширина екрана яких не перевищує 768 пікселів, що характерно для мобільних пристроїв. Аналогічно, медіа-запит із параметром **min-width: 1024px** застосовує стилі до великих екранів, як-от планшети або десктопи, ширина яких перевищує 1024 пікселі.

Цей підхід дає можливість оптимізувати дизайн під різні умови перегляду і забезпечити зручність використання для кожного користувача. Веб-сайти можуть мати спрощений макет на мобільних пристроях для полегшення навігації, а на десктопах – виглядати більш складно, з використанням додаткових елементів, таких як великі зображення, відео та інтерактивні блоки.

Ключові параметри медіа-запитів дозволяють гнучко змінювати поведінку елементів сторінки можна роглянути у табл. 2.2.

Таблиця 2.2.

Основні параметри медіа-запитів

Параметр	Опис	Приклад
max-width	Максимальна ширина екрана. Застосовується для мобільних пристроїв або малих екранів.	@media screen and (max-width: 768px)
min-width	Мінімальна ширина екрана. Використовується для планшетів або великих екранів.	@media screen and (min-width: 1024px)
orientation	Орієнтація екрана: портретна або ландшафтна.	@media screen and (orientation: portrait)
aspect-ratio	Відношення ширини до висоти екрана.	@media screen and (aspect-ratio: 16/9)
resolution	Роздільна здатність екрана, вимірювана в DPI (dots per inch).	@media screen and (min-resolution: 300dpi)

Ці параметри дозволяють створювати гнучкі макети, які підлаштовуються під різні розміри екранів. Наприклад, завдяки параметру **orientation**, можна налаштувати веб-сторінку так, щоб вона змінювала свій вигляд залежно від того, чи знаходиться пристрій у вертикальній (портретній) чи горизонтальній (ландшафтній) орієнтації.

Підтримка різних роздільних здатностей екранів, особливо на сучасних високоякісних дисплеях, таких як Retina-дисплеї, стала невід'ємною частиною

розробки веб-сайтів. Використовуючи параметр **resolution**, веб-дизайнери можуть забезпечити оптимальне відображення контенту на екранах з високою щільністю пікселів, що робить зображення більш чіткими та насиченими.

Технології, які використовуються разом із медіа-запитами, такі як Flexbox та CSS Grid, дозволяють створювати складні макети, що адаптуються до різних розмірів екранів. Завдяки цим технологіям розташування елементів на сторінці може змінюватися автоматично залежно від розмірів і пропорцій екрану користувача, забезпечуючи, щоб сторінка залишалася зручною і зрозумілою незалежно від пристрою. Це робить сайт однаково зручним як на маленькому смартфоні, так і на великому екрані настільного комп'ютера. Відповідно на рис.2.1. можемо розглянути графік який відповідає зростанню використання медіа-запитів у веб-дизайні за період з 2015 до 2023 року.

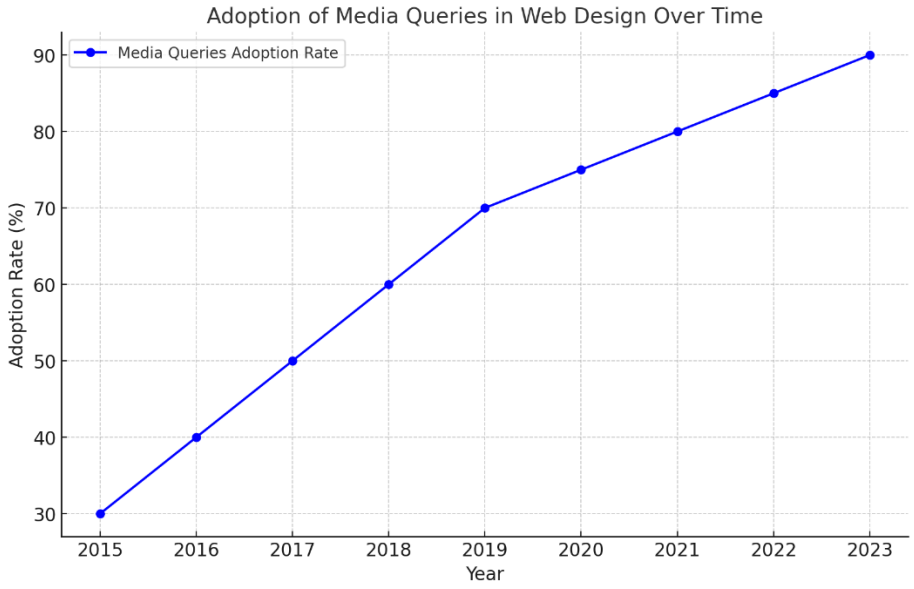

Рис.2.1. – Графік зростання використання медіа-запитів у веб-дизайні

Адаптивні зображення, зокрема, дозволяють завантажувати різні варіанти зображень залежно від розміру екрану. Це зменшує навантаження на сервер і покращує продуктивність сайту. Наприклад, на мобільному пристрої може завантажуватися менше зображення з меншою роздільною здатністю, тоді як для великого екрана буде підвантажуватися зображення високої якості.

Таким чином, медіа-запити є невід'ємною частиною адаптивного дизайну, що дозволяє веб-дизайнерам створювати сайти, які підлаштовуються під будь-який пристрій і надають найкращий користувацький досвід, незалежно від того, чи це смартфон, планшет або настільний комп'ютер.

Гнучкі зображення та мультимедіа є важливими компонентами адаптивного веб-дизайну, оскільки вони дозволяють контенту автоматично змінювати розміри відповідно до ширини контейнера або екрану користувача. Це забезпечує зручне та оптимальне відображення контенту на різних пристроях, таких як смартфони, планшети чи настільні комп'ютери.

Гнучкі зображення змінюють свої розміри залежно від ширини контейнера, в якому вони знаходяться. Це досягається за допомогою CSS-властивості max-width: 100%, яка гарантує, що зображення не буде виходити за межі контейнера і підлаштовуватиметься під розміри екрана. Наприклад, якщо зображення розміщене у контейнері шириною 70% від ширини вікна браузера, то й зображення займатиме 70% ширини контейнера, незалежно від того, який розмір вікна браузера.

Щоб зберегти пропорції зображення, зазвичай використовують властивість height: auto. Це дозволяє автоматично регулювати висоту зображення залежно від його ширини, що забезпечує правильне відображення без деформації. Такий підхід дозволяє уникати проблем із горизонтальною прокруткою або неправильно відображеними зображеннями на менших екранах.

Однак, хоча зображення і може змінювати свої розміри відповідно до екрану, його розмір файлу залишається незмінним. Це може впливати на швидкість завантаження сайту, особливо на мобільних пристроях із повільнішим інтернетом. Для вирішення цієї проблеми використовуються адаптивні зображення, які дозволяють завантажувати різні версії одного зображення для різних пристроїв. Це означає, що для мобільних пристроїв може бути завантажена менша версія зображення, а для десктопів — більша. Це значно покращує продуктивність сайту.

Мультимедіа

Щодо мультимедійного контенту, наприклад, відео або інтерактивних елементів, застосовується подібний підхід. Відео може автоматично змінювати свої розміри залежно від ширини контейнера, зберігаючи правильні пропорції завдяки властивості height: auto. Це дозволяє мультимедійним елементам бути однаково зручними для користувачів як на маленьких екранах смартфонів, так і на великих екранах настільних комп'ютерів.

Атрибути srcset та sizes для зображень в HTML допомагають браузерам вибирати найбільш відповідну версію зображення залежно від розміру екрану або

роздільної здатності дисплея. Це дозволяє завантажувати оптимальну версію зображення на кожному пристрої. Наприклад, мобільний пристрій може завантажити зображення меншого розміру для збереження трафіку та прискорення завантаження, тоді як пристрій із дисплеєм високої роздільної здатності завантажить зображення з кращою якістю.

Переваги гнучких зображень і мультимедіа:

1. зручність для користувачів: гнучкі зображення і мультимедіа дозволяють адаптувати контент для користувачів з різними розмірами екранів, роблячи сайти більш зручними.

2. Покращення продуктивності: використання адаптивних зображень допомагає зменшити розмір файлів для мобільних пристроїв, що знижує час завантаження сторінки і покращує продуктивність сайту.

3. Оптимізація для дисплеїв високої роздільної здатності: завдяки технологіям адаптивного завантаження мультимедіа, зображення і відео зберігають високу якість на дисплеях з високою роздільною здатністю, таких як Retina-дисплеї.

Таблицю 2.3. порівняння можна розглянути далі.

Таблиця 2.3.

Таблиця порівняння підходів до використання зображень

Підхід	Опис	Переваги	Недоліки
Фіксовані зображення	Зображення має постійні розміри, які не змінюються залежно від ширини екрану.	Простота у реалізації	Може викликати горизонтальну прокрутку на менших екранах, погіршуючи користувацький досвід.
Гнучкі зображення	Зображення змінюють свої розміри відповідно до ширини контейнера (наприклад, max-width: 100%).	Підлаштовуються під різні екрани, зберігаючи пропорції.	Розмір файлу залишається незмінним, що може впливати на швидкість завантаження.
Адаптивні зображення	Використовуються різні версії одного зображення для різних пристроїв за допомогою srcset і	Покращена продуктивність, оптимізація швидкості завантаження,	Потребує більше ресурсів для реалізації та підтримки.

	sizes.	збереження пропорцій на всіх пристроях.	

Розглянемо на рис.2.2. з адаптивними зображеннями на веб-сторінці за допомогою елементів srcset і sizes.

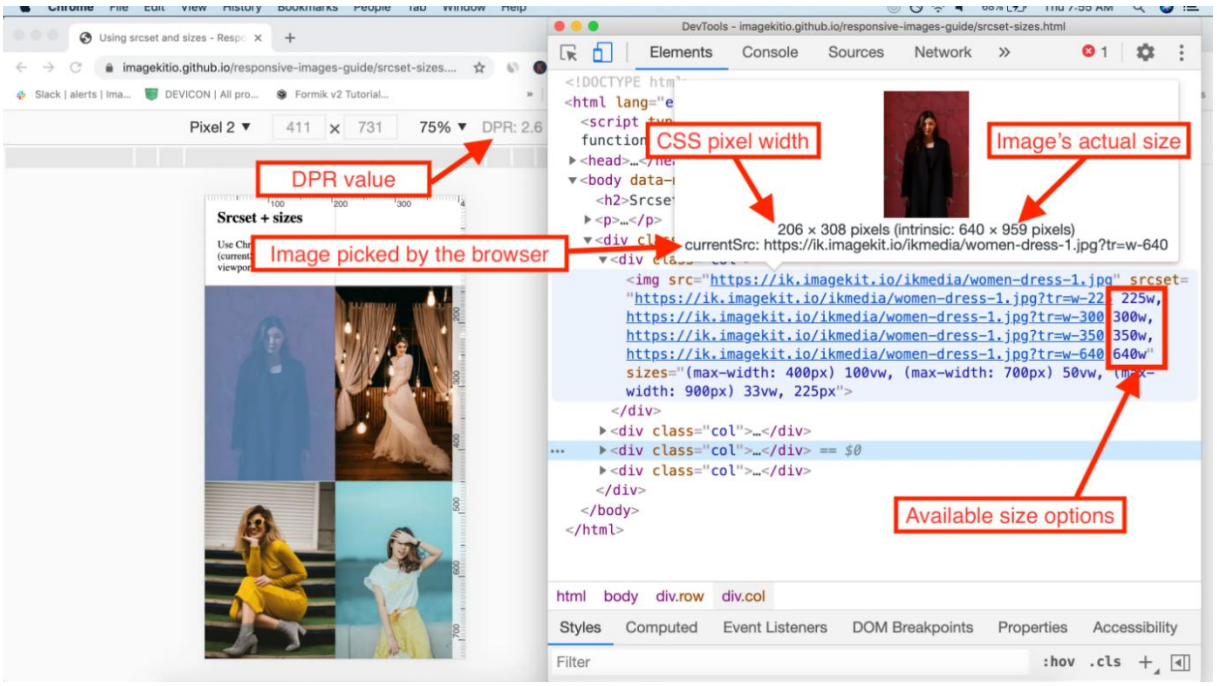

Рис.2.2. – Адаптивні зображення на веб-сторінці за допомогою елементів srcset і sizes

На зображенні можемо переглянути параметри відповідно розглянемо їх детальніше:

1. DPR (Device Pixel Ratio): це показник, який вказує на щільність пікселів пристрою. У цьому випадку це 2.6, що означає, що браузеру потрібно зображення з вищою роздільною здатністю для чіткого відображення на пристрої з таким співвідношенням.

2. CSS Pixel Width: відображає ширину зображення у CSS пікселях (у цьому прикладі – 206 пікселів). Це те, як браузер "бачить" розміри зображення в контексті CSS.

3. Image’s Actual Size: тут видно фактичний розмір зображення у пікселях, яке було завантажено браузером (640 × 959 пікселів).

4. Image Picked by the Browser: це вказує на конкретний URL зображення, яке було обране браузером з різних доступних варіантів (вказано у srcset).

5. Available Size Options: у елементі srcset зазначено кілька варіантів зображень різної ширини, наприклад, 225w, 300w, 350w, і 640w. Це дозволяє

браузеру завантажити оптимальне зображення, залежно від ширини екрану та DPR пристрою.

Адаптивні зображення є критично важливими для сучасного веб-дизайну, оскільки вони дозволяють забезпечити швидке завантаження сторінок і оптимальну якість зображень на різних пристроях. Однією з найважливіших технологій для цього є Client Hints (натяки від клієнта), які дозволяють браузерам відправляти інформацію про пристрій на сервер, щоб отримати оптимізовані зображення залежно від параметрів екрану, таких як розмір і роздільна здатність. Ці технології значно покращують користувацький досвід, дозволяючи адаптувати контент під конкретні пристрої без надмірного навантаження на ресурси мережі.

Client Hints включають натяки на DPR (співвідношення пікселів), Width (ширина зображення) і Viewport-Width (ширина вікна перегляду). Вони забезпечують правильний вибір зображення відповідно до реальної роздільної здатності та ширини екрану пристрою, на якому відбувається перегляд сторінки. Завдяки цьому підходу браузер автоматично підбирає зображення, яке найбільше відповідає характеристикам пристрою, на якому завантажується сторінка, що особливо важливо для мобільних пристроїв.

На рисунку 2.3. показана таблиця, яка демонструє підтримку натяків від клієнта різними браузерами, такими як Chrome, Firefox, Safari, Opera, і мобільними версіями цих браузерів. У таблиці зеленим кольором позначено браузери, що підтримують ці функції, а червоним — ті, що не підтримують. У верхній частині таблиці також вказано загальний відсоток підтримки натяків від клієнта на всіх пристроях, що становить 74,98%.

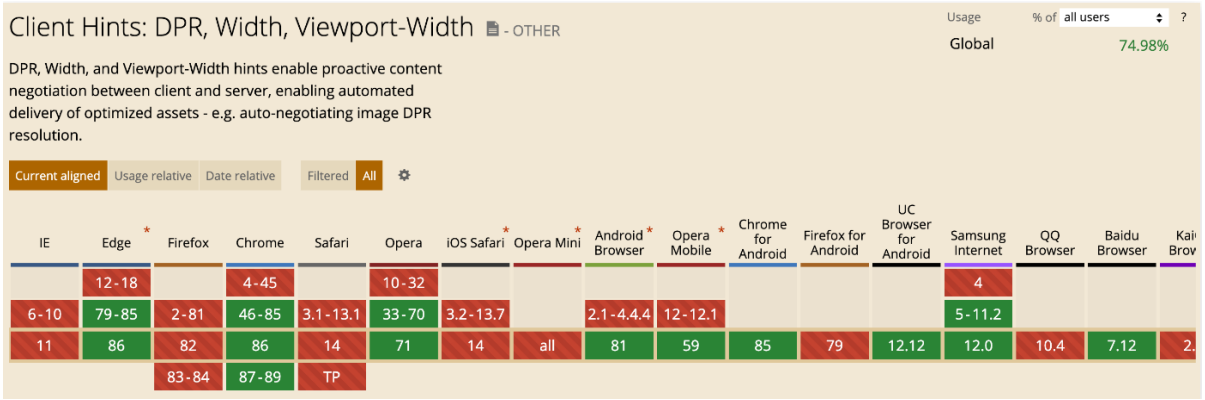

Рис.2.3. – Таблиця яка демонструє підтримку натяків від клієнта різними браузерами

Розглянемо рисунок 2.4, рисунок 2.5. яке ми реалізували для опису адаптивних зображень

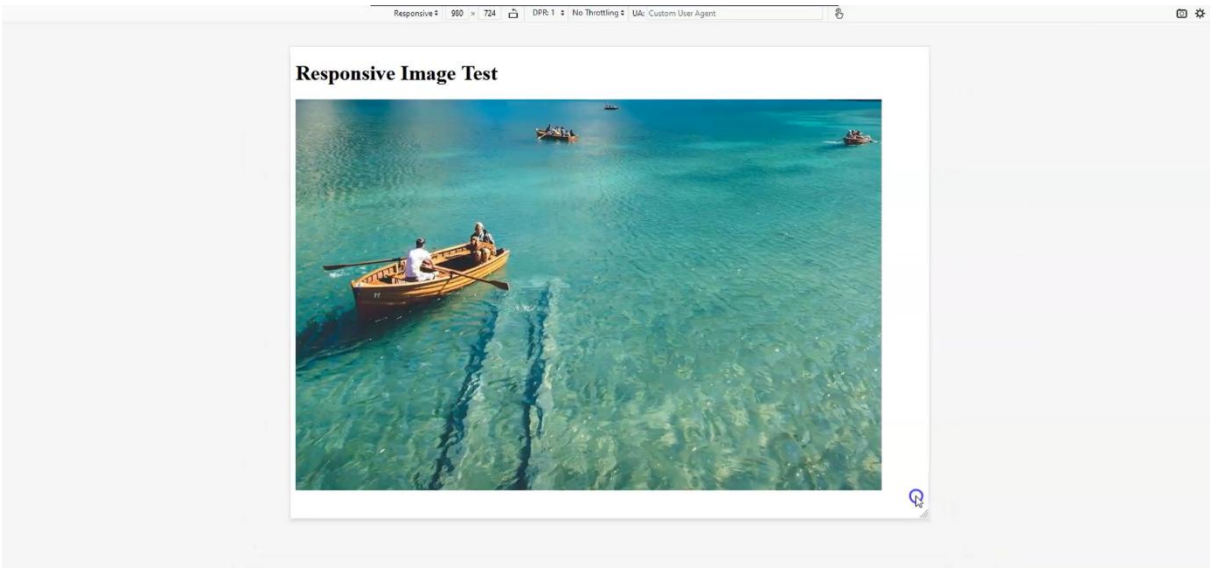

Рис.2.4. – Початкове зображення

Далі ми починаємо рухати зображення і відповідно результат можемо переглянути на рис.2.5.

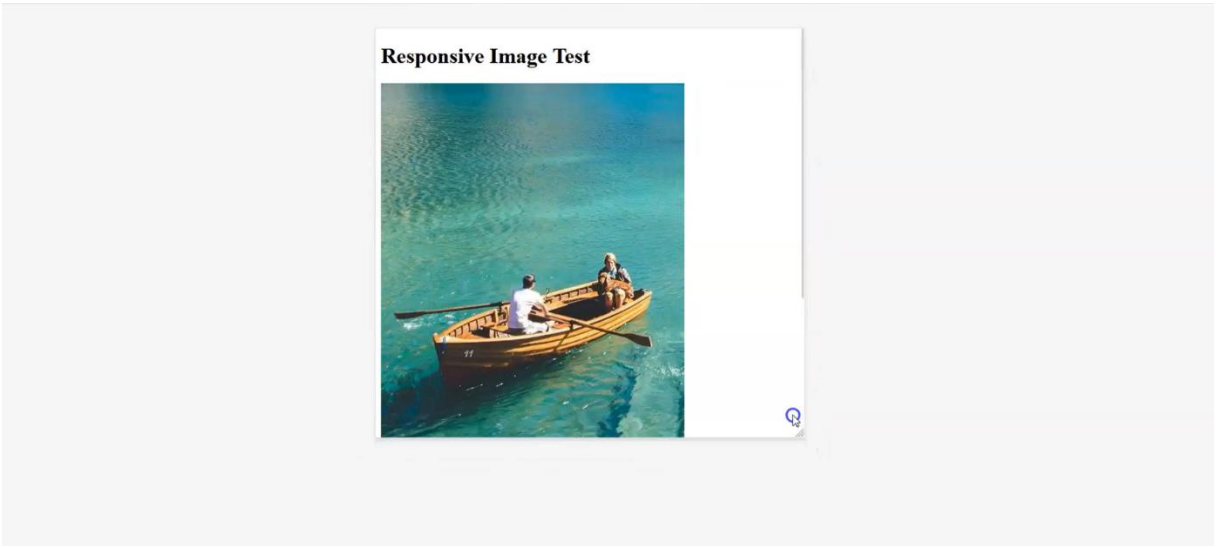

Рис.2.5. – Рух зображення і відповідна його зміна

Для реалізації адаптивних зображень на веб-сторінці використовується елемент <picture>, який дозволяє підлаштовуватися під різні розміри екранів та роздільну здатність пристроїв. Цей підхід забезпечує вибір найбільш відповідного зображення залежно від характеристик пристрою користувача, наприклад, розміру екрану або щільності пікселів. У тегах <source> визначаються різні варіанти зображень, які браузер може вибирати, орієнтуючись на медіа-запити. У випадках, коли не спрацьовують жодні умови, завантажується зображення з тегу <img> як запасний варіант. Це оптимізує завантаження зображень і покращує

продуктивність веб-сайту на мобільних пристроях, планшетах та настільних комп'ютерах. На рис.2.6. можемо переглянути код.

```
<picture>
  <source
    srcset="
      images/art-direction/lake-with-mountains.jpg    1x,
      images/art-direction/lake-with-mountains-2x.jpg 2x
    "
    media="(min-width: 1280px)"
    width="1200"
    height="800"
  />

  <source
    srcset="
      images/art-direction/lake-without-mountains.jpg    1x,
      images/art-direction/lake-without-mountains-2x.jpg 2x
    "
    media="(min-width: 640px) and (max-width: 1279px)"
    width="900"
    height="600"
  />

  <img
    src="images/art-direction/only-a-boat.jpg"
    srcset="images/art-direction/only-a-boat-2x.jpg 2x"
    alt="Lake"
    width="450"
    height="800"
  />
</picture>
```

Рис.2.6. – Реалізація правильного форматування адаптивних зображень

Елемент <picture> містить кілька джерел зображень у тегі <source>, кожне з яких має свій набір умов у медіа-запитах. Якщо розмір екрану більше за 1280px, браузер завантажує зображення lake-with-mountains.jpg, яке має версії для дисплеїв із різною щільністю пікселів (1x і 2x). Якщо ширина екрану перебуває між 640px і 1279px, завантажується інше зображення lake-without-mountains.jpg. Якщо розмір екрану менший за 640px, то за замовчуванням завантажується зображення only-a-boat.jpg, також у версіях для дисплеїв 1x і 2x.

Відповідно на кінець розглянемо рис.2.7. який демонструє графік розмірів зображень для різних підходів

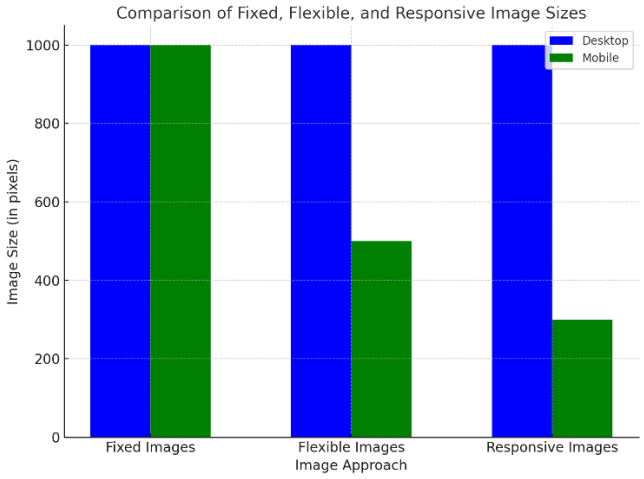

Рис.2.7. – Графік розмірів різних зображень для різних підходів

Адаптивний контент покращує користувацький досвід і дозволяє веб-сторінкам бути доступними та зручними на різних пристроях. Оптимізація зображень та динамічна зміна макету забезпечують швидку роботу сайту і його зручне використання, незалежно від типу пристрою користувача.

2.2. Значення алгоритмічних підходів у розробці макетів

Алгоритмічні підходи в розробці веб-макетів мають вирішальне значення для створення гнучких, адаптивних і оптимізованих веб-сторінок. Вони дозволяють автоматично підлаштовувати компоненти макета під різні розміри екранів, забезпечуючи високий рівень взаємодії з користувачами незалежно від їхніх пристроїв.

Один з ключових алгоритмічних підходів — це CSS Grid, який дозволяє створювати двовимірні макети для складніших веб-додатків. Використовуючи Grid, дизайнери можуть точно контролювати, як блоки розташовані в сітці, надаючи гнучкість у їхньому вирівнюванні та масштабуванні. Grid-система забезпечує автоматичне розташування елементів, що дозволяє адаптуватися до змін ширини вікна перегляду без потреби змінювати кожен блок вручну. Це дозволяє розробникам створювати багаторівневі та багатоколонкові макети, які гармонійно виглядатимуть на будь-якому пристрої.

Ще один важливий інструмент — це Flexbox, який найкраще підходить для одномірних макетів. Flexbox спрощує вирівнювання елементів у контейнерах як по горизонталі, так і по вертикалі, забезпечуючи динамічний розподіл простору між блоками. Алгоритми Flexbox автоматично змінюють розмір і положення елементів, щоб вони завжди відповідали параметрам батьківського контейнера, незалежно від розміру екрану або кількості контенту.

Окрім CSS Grid і Flexbox, важливу роль грають і алгоритми, що управляють адаптивним зображенням. За допомогою таких інструментів, як srcset і sizes, браузери можуть автоматично вибирати найбільш відповідну версію зображення залежно від щільності пікселів і розміру екрану. Це дозволяє зменшити час завантаження сторінки, зберігаючи якість зображення на високому рівні, навіть для пристроїв із високою роздільною здатністю.

Крім того, важливою є оптимізація швидкості завантаження за допомогою алгоритмів, які знижують кількість ресурсів, що витрачаються на завантаження контенту. Сучасні браузери можуть використовувати клієнтські натяки (client hints) для того, щоб вирішувати, яке зображення чи ресурс краще завантажити, залежно від стану мережі користувача.

Медіа-запити є одним з основних інструментів адаптивного дизайну, які використовуються для зміни стилів або структури компонентів на основі розміру екрану. В React це можна реалізувати за допомогою бібліотек, таких як react-responsive, або простого CSS. Це дозволяє ефективно керувати рендерингом різних компонентів, коли користувачі переходять з одного пристрою на інший (наприклад, з десктопа на мобільний телефон). Медійні запити дозволяють використовувати один і той самий компонент для різних розмірів екрану, що значно знижує навантаження на розробку і підтримку.

Одним з ключових алгоритмів в адаптивних макетах є зміна структури компонентів залежно від розміру вікна браузера. В React це можна реалізувати через управління станом компонентів за допомогою хуків useState і useEffect. Такий підхід дозволяє додаткам бути максимально динамічними, змінюючи як відображення, так і логіку роботи компонентів на основі поточного стану вікна. Це особливо важливо для додатків, що вимагають різної логіки поведінки на різних пристроях, як-от для інтерактивних додатків або аналітичних панелей.

У сучасному адаптивному дизайні JavaScript дозволяє впроваджувати клієнтські запити, такі як Device Pixel Ratio (DPR), Viewport Width, які дають браузерам можливість отримувати інформацію про екран і підбирати найбільш оптимальне зображення або макет для пристрою. Це дозволяє завантажувати легші або більш чіткі зображення для різних розмірів екранів, що покращує продуктивність додатка і час завантаження. За допомогою window.matchMedia() можна створювати медіа-запити прямо в JavaScript, що дозволяє більш гнучко керувати рендерингом компонентів. Клієнт (браузер користувача) надсилає запит на отримання зображення із сервера, вказуючи розмір зображення (400 пікселів) та коефіцієнт пікселів пристрою (DPR), який у цьому випадку дорівнює 2. У відповідь сервер надсилає відповідне зображення, яке відповідає цим вимогам: у цьому випадку це зображення шириною 400 пікселів і розміром 100KB відповідно на

рис.2.8. можемо переглянути детальніше.

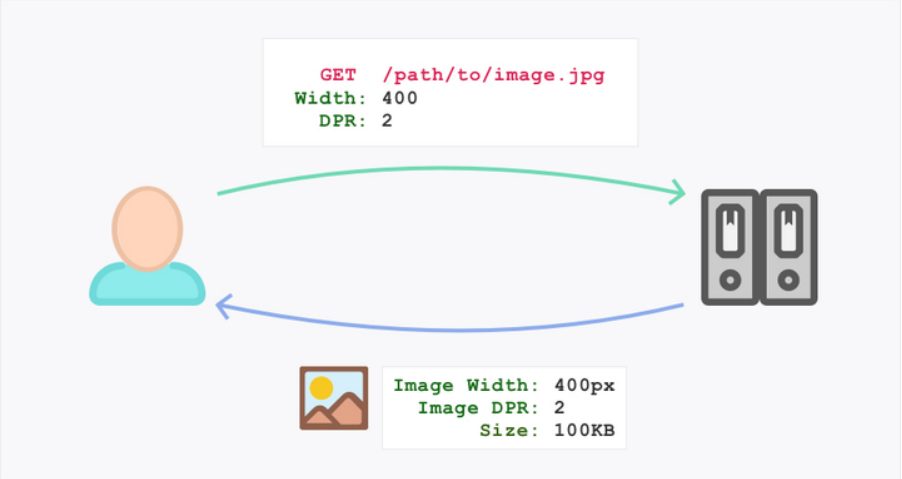

Рис.2.8. – Запит на отримання зображення із сервера

Коефіцієнт пікселів пристрою (DPR) важливий, оскільки він дозволяє серверу доставляти зображення з достатньою роздільною здатністю, щоб воно виглядало чітким на екранах з високою щільністю пікселів, таких як сучасні смартфони та монітори з високою роздільною здатністю. Для пристрою з DPR 2 сервер може відправити зображення, яке фактично має вдвічі більшу роздільну здатність, що дозволяє уникнути пікселізації на екрані користувача.

Цей процес забезпечує доставку оптимізованого зображення для пристрою користувача, зменшуючи зайве використання пропускної здатності та покращуючи час завантаження сторінки. Представимо адаптивність на рис.2.9.

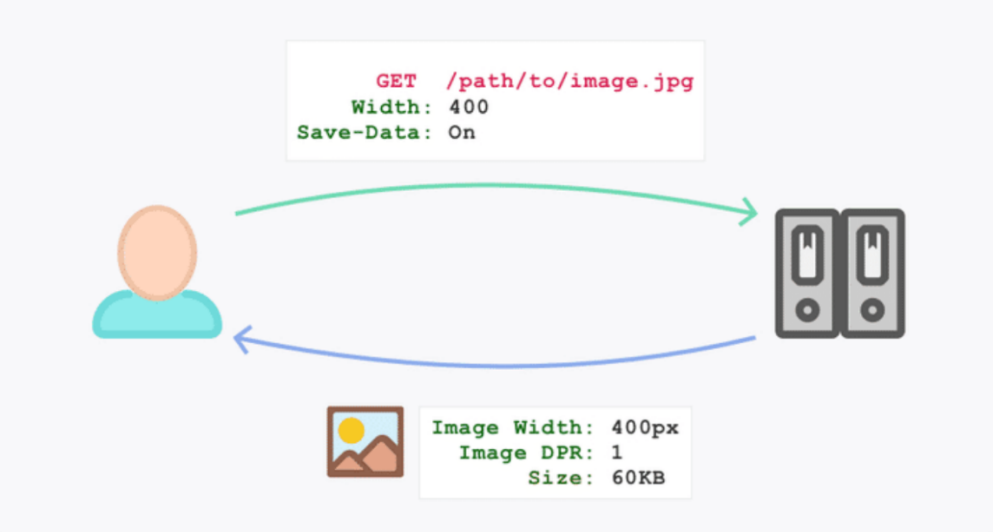

Рис.2.9. - Процес оптимізації завантаження зображень за допомогою клієнтських запитів.

Клієнт (браузер користувача) відправляє запит на отримання зображення із зазначеними параметрами, такими як ширина зображення та включений режим

економії даних. Сервер, отримавши ці дані, вибирає найбільш підходяще зображення, враховуючи ширину екрану та налаштування економії трафіку. У відповідь сервер відправляє зображення оптимізованого розміру для покращення швидкості завантаження та зменшення споживання даних. Цей підхід дозволяє зменшити трафік для користувачів з повільним інтернетом і покращити загальний досвід роботи з веб-сайтом. Відповідно програмний код який реалізовує цю ідею можемо розглянути далі на рис. 2.10.

```
// Перевірка ширини екрана за допомогою matchMedia
function setResponsiveImage() {
  const imgElement = document.getElementById('responsive-img');
  // Якщо ширина екрана менша або дорівнює 768 пікселів
  if (window.matchMedia("(max-width: 768px)").matches) {
    imgElement.src = 'image-small.jpg';  // Встановлюємо зображення для мобільних
  }
  // Якщо ширина екрана більша за 768 пікселів і менша за 1280 пікселів
  else if (window.matchMedia("(min-width: 769px) and (max-width: 1280px)").matches) {
    imgElement.src = 'image-medium.jpg';  // Встановлюємо зображення для планшетів
  }
  // Якщо ширина екрана більше 1280 пікселів
  else {
    imgElement.src = 'image-large.jpg';  // Встановлюємо зображення для десктопів
  }
  // Додаткова перевірка на коефіцієнт пікселів (DPR)
  if (window.devicePixelRatio > 1) {
    imgElement.src = 'image-high-res.jpg';  // Встановлюємо зображення високої роздільної здатності
  }
}
// Виклик функції при завантаженні сторінки
window.addEventListener('load', setResponsiveImage);
// Виклик функції при зміні розміру вікна браузера
window.addEventListener('resize', setResponsiveImage);
```

Рис. 2.10. – Програмний код адаптивного завантаження зображень на основі розміру екрана і коефіцієнта пікселів пристрою (DPR)

За допомогою функції window.matchMedia() перевіряється ширина вікна браузера, що дозволяє динамічно змінювати джерело зображення для мобільних, планшетів і десктопів. Додатково враховується значення DPR, яке допомагає завантажувати зображення високої роздільної здатності для пристроїв із дисплеями Retina або іншими екранами з високою щільністю пікселів. Такий підхід дозволяє оптимізувати продуктивність веб-сторінки, зменшуючи обсяг даних для завантаження і покращуючи досвід користувача.

Алгоритм лінивого завантаження (lazy loading) у JavaScript та React

спрямований на підвищення ефективності та швидкодії веб-додатка, особливо при роботі з великою кількістю зображень або компонентів. Його суть полягає в тому, щоб відкласти завантаження ресурсів, які не знаходяться у полі зору користувача, до моменту, коли вони стануть необхідними (наприклад, при прокручуванні сторінки вниз). Це зменшує навантаження на початкове завантаження сторінки і економить пропускну здатність мережі, що особливо корисно для мобільних пристроїв або при повільному інтернет-з'єднанні.

У React найчастіше використовують бібліотеки, такі як React Lazy і Suspense, для реалізації цієї технології. React.lazy() дозволяє завантажувати компоненти лише тоді, коли вони потрібні, а Suspense показує резервний інтерфейс (наприклад, індикатор завантаження) до моменту, коли компонент буде завантажений. Наприклад, зображення можуть бути завантажені лише після того, як користувач прокрутить сторінку до місця їх відображення, використовуючи подію прокрутки або Intersection Observer API.

Логіка лінівого завантаження в адаптивному дизайні може також поєднуватися з медіа-запитами, що дозволяє завантажувати оптимізовані версії зображень для різних пристроїв. Наприклад, для мобільних пристроїв завантажується менша версія зображення, що значно прискорює час завантаження та зменшує обсяг переданих даних.

Основні етапи алгоритму:

1. Спостереження за компонентами чи зображеннями на сторінці (наприклад, через Intersection Observer API).
2. Визначення моменту, коли елемент входить у видиму частину екрана.
3. Завантаження ресурсу (зображення або компонента) у момент, коли він потрібен користувачу.
4. Після завантаження відображається готовий контент.

Це підвищує продуктивність, скорочує час першого рендеру сторінки, особливо на мобільних пристроях, і значно покращує користувацький досвід. Розлянемо приклад рис. 2.11.

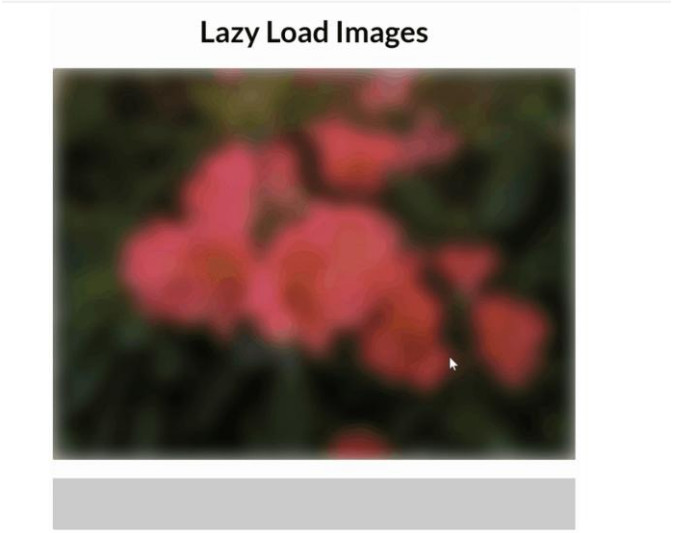

Рис.2.11. – Алгоритм лінивого завантаження (економія пам’яті)

Спочатку завантажується низькороздільна версія (розмита) зображення, коли користувач прокручує сторінку. Коли зображення з'являється у видимій області екрану, завантажується повна версія зображення у високій якості, замінюючи розмиту версію, що представлено на рис.2.12. Такий підхід покращує продуктивність, скорочуючи час початкового завантаження, особливо для сторінок із великою кількістю зображень, а також економить трафік, що особливо корисно для мобільних користувачів або при повільному з'єднанні.

Рис.2.12. – Прогружена фотографія у повній якості

На рис. 2.13. представлено консоль браузера, яка відображає список ресурсів, завантажених за допомогою клієнтських запитів (XHR) або методів fetch у JavaScript.

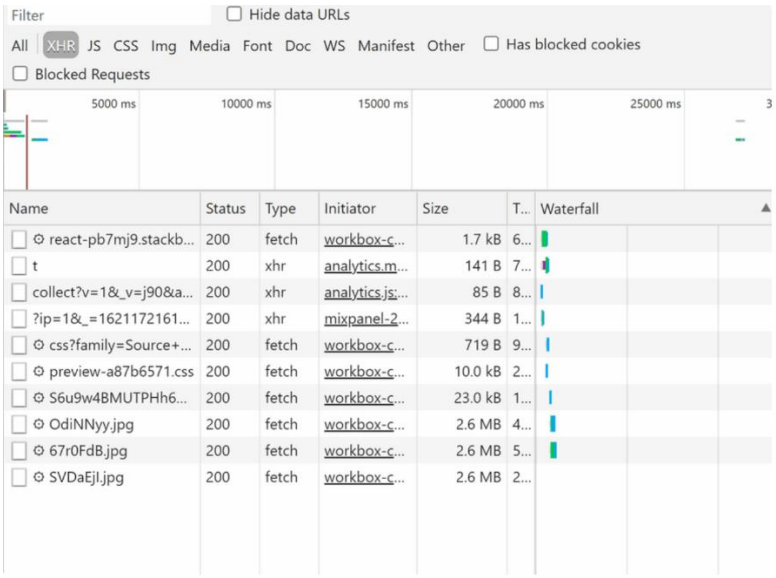

Рис.2.13. – Список ресурсів завантажених за допомогою клієнтських запитів

Кожен рядок таблиці містить інформацію про ресурс: його ім'я (Name), статус завантаження (Status), тип запиту (Type), ініціатор (Initiator), розмір файлу (Size) та графік часу завантаження (Waterfall). Графік Waterfall показує, як і коли відбувалося завантаження файлів у хронологічному порядку. Це дозволяє розробникам аналізувати продуктивність вебсторінки та діагностувати проблеми з мережею або часом відгуку сервера. Відповідно програмний код який ми реалізували можемо переглянути на рис.2.14.

```
import React, { useState, useRef } from 'react';
import classnames from 'classnames';
import { useIntersection } from './intersectionObserver';
import './imageRenderer.scss';
const ImageRenderer = ({ url, thumb, width, height }) => {
  const [isInView, setIsInView] = useState(false);
  const imgRef = useRef();
  useIntersection(imgRef, () => {
    setIsInView(true);
  });
  return (
    <div
      className="image-container"
      ref={imgRef}
      style={{
        paddingBottom: `${(height / width) * 100}%`,
        width: '100%'
      }}
    >
```

Рис. 2.14. – Відображення зображень із застосуванням техніки "лінивого завантаження"

Компонент використовує хук useState для збереження стану, чи зображення знаходиться у полі зору користувача, і useRef для створення референції на HTML-елемент. За допомогою кастомного хука useIntersection, який слідкує за появою зображення в полі зору, компонент починає завантажувати зображення лише тоді, коли воно стане видимим на екрані, тим самим оптимізуючи ресурси.

Представимо порівняння алгоритмів у вигляді таблиці 2.4.

Таблиця 2.4.

Порівняння алгоритмів для адаптивних макетів

Алгоритм	**Переваги**	**Недоліки**	**Сфера використання**
Медіа-запити в CSS	Прості у використанні, забезпечують гарну адаптацію макета	Обмежені можливості для динамічного рендерингу, неможливо змінювати логіку компонентів	Статичні веб-сайти, мобільні версії сайтів
Управління станом у React	Дозволяє динамічно змінювати компоненти і їх поведінку, реагує на зміну розміру вікна	Потрібні додаткові ресурси для управління станом, може бути складно підтримувати у великих додатках	Інтерактивні веб-додатки, панелі керування
CSS Flexbox	Гнучке управління простором між елементами, підходить для одновимірних макетів	Обмежена підтримка для складних двовимірних макетів	Простий лінійний дизайн
CSS Grid	Підтримка двовимірних макетів, зручна для складних ієрархій	Може бути складним у налаштуванні для невеликих проєктів	Великі веб-додатки, де потрібна гнучкість
Lazy Loading в JavaScript	Оптимізує продуктивність, дозволяє зменшити час завантаження додатка	Не завжди працює ефективно для дуже динамічних додатків	Галереї зображень, великі ресурси
Client-Side Hints	Оптимізує відображення зображень і ресурсів на різних пристроях	Потребує підтримки браузера, не завжди працює з усіма користувацькими агентами	Веб-сайти з великими зображеннями, мультимедіа

Алгоритми адаптивного дизайну в React і JavaScript дозволяють створювати більш гнучкі і інтерактивні інтерфейси, які можуть автоматично адаптуватися під

умови користувача. Кожен з описаних алгоритмів має свої переваги і недоліки, і вибір залежить від конкретних вимог проєкту. Flexbox та Grid забезпечують відмінні інструменти для побудови макетів, тоді як медіа-запити та клієнтські запити дають можливість тонкого налаштування стилів і поведінки компонентів.

2.3. Визначення вимог до адаптивних веб-макетів

Вимоги до програмного забезпечення є основою для створення якісного продукту, який відповідає потребам користувачів і бізнесу. Вони визначають, яким чином повинна працювати система, які функції вона повинна виконувати, які обмеження слід врахувати, а також як повинна забезпечуватися її безпека та зручність у використанні. Вимоги можуть бути різних типів: функціональні, нефункціональні, вимоги до безпеки та інтерфейсу. Функціональні вимоги визначають конкретні можливості системи, нефункціональні — якості, які система повинна мати, такі як продуктивність, масштабованість та надійність. Вимоги до безпеки забезпечують захист даних та захист від зловмисних дій, а вимоги до інтерфейсу описують, як система повинна взаємодіяти з користувачем, забезпечуючи зручність та інтуїтивність. Всі ці типи вимог відіграють важливу роль у забезпеченні створення якісного програмного продукту, що задовольняє всі потреби кінцевих користувачів і бізнесу.

Функціональні вимоги для адаптивних макетів визначають, як саме веб-сайт повинен адаптуватися до різних розмірів екранів і пристроїв. Макет повинен автоматично підлаштовуватися під розмір і орієнтацію екрана, забезпечуючи оптимальне відображення контенту на смартфонах, планшетах та настільних комп'ютерах. Всі елементи сторінки, такі як зображення, текстові блоки та кнопки, повинні масштабуватися і змінювати своє розташування залежно від доступного простору. Система повинна забезпечувати правильну послідовність відображення важливого контенту, зокрема пріоритетним є основний контент, а вторинні елементи можуть приховуватися або змінюватися для полегшення перегляду. Зображення та інші медіа-файли повинні завантажуватися в оптимальних розмірах, щоб забезпечити швидкість роботи сайту, а також відповідати якості для кожного типу екрана. Інтерактивні елементи інтерфейсу повинні залишатися зручними у використанні на сенсорних пристроях, а навігація — інтуїтивною незалежно від

розміру екрану. Загальну таблицю з функціональними вимогами розглянемо далі у табл.2.5.

Таблиця 2.5.

Функціональні вимоги розроблювальної системи

№	Функціональна вимога
1	Макет веб-сайту автоматично підлаштовуватися під розмір і орієнтацію екрана.
2	Елементи сторінки, такі як зображення, текстові блоки та кнопки, повинні змінювати свій розмір і положення залежно від доступного простору.
3	Основний контент має пріоритет у відображенні, а вторинні елементи можуть приховуватися або адаптуватися.
4	Зображення завантажуються в оптимальних розмірах відповідно до типу пристрою.
5	Інтерактивні елементи інтерфейсу повинні залишатися зручними для використання на сенсорних екранах.
6	Навігація є інтуїтивною та зручною для використання на пристроях різних розмірів.
7	Шрифт автоматично масштабується, щоб залишатися читабельним на будь-якому пристрої.
8	Макет підтримує зміну вигляду залежно від роздільної здатності екрана з використанням медіа-запитів.
9	Контент сайту адаптується для вертикальної та горизонтальної орієнтації пристроїв.
10	Всі форми та поля введення повинні бути зручними для заповнення на сенсорних пристроях.
11	Макет має підтримувати зміну розташування меню залежно від розміру екрана, наприклад, перехід від горизонтального до "гамбургерного" меню.
12	Посилання і кнопки є достатньо великими для зручного натискання на сенсорних екранах.
13	Сайт підтримує адаптивне відображення відео та інших мультимедійних елементів.
14	Динамічні елементи, такі як слайдери, повинні змінювати розмір відповідно до ширини екрана.
15	Сайт підтримує можливість гнучкого налаштування відступів та полів, щоб уникнути горизонтального прокручування на мобільних пристроях.
16	Макет автоматично приховує або адаптує великі банери чи інші рекламні елементи на менших екранах.
17	Контент коректно відображатися без перекривання або накладання елементів незалежно від розміру екрана.
18	Форми для введення даних повинні автоматично налаштовувати тип клавіатури (наприклад, цифри для телефонного номера) для зручності введення на мобільних пристроях.

Нефункціональні вимоги для адаптивних макетів визначають, якими якісними характеристиками повинен володіти веб-сайт для забезпечення зручного

використання та ефективного функціонування. Система повинна демонструвати високу продуктивність, швидко завантажувати контент незалежно від типу пристрою або швидкості інтернет-з'єднання. Веб-сайт має бути надійним, забезпечуючи стабільну роботу без помилок або збоїв під час взаємодії з користувачем. Адаптивний макет є масштабованим, щоб мати можливість обслуговувати велику кількість користувачів без втрати якості роботи. Окрім цього, дизайн враховує зручність використання (юзабіліті), забезпечуючи інтуїтивну навігацію та легкий доступ до основних функцій на всіх пристроях. Також важливо враховувати сумісність з різними браузерами та операційними системами для коректного відображення контенту незалежно від обраної платформи. Розглянемо детальніше у табл. 2.6.

Таблиця 2.6.

Нефункціональні вимоги системи

№	Нефункціональна вимога
1	Висока продуктивність, швидке завантаження контенту незалежно від типу пристрою або швидкості інтернет-з'єднання.
2	Надійність, стабільна робота без помилок або збоїв під час взаємодії з користувачем.
3	Масштабованість для обслуговування великої кількості користувачів без втрати якості роботи.
4	Зручність використання (юзабіліті) з інтуїтивною навігацією та легким доступом до основних функцій на всіх пристроях.
5	Сумісність з різними браузерами та операційними системами для коректного відображення контенту.
6	Адаптивний дизайн повинен забезпечувати консистентність візуального стилю на всіх типах пристроїв.
7	Доступність для користувачів з обмеженими можливостями, відповідно до стандартів WCAG.
8	Безпека даних, включаючи шифрування чутливої інформації та захист від зловмисних атак.
9	Легкість в обслуговуванні, з можливістю швидкого оновлення та модифікації макетів при зміні вимог.
10	Відповідність стандартам веб-розробки для забезпечення високої якості коду і сумісності з новими технологіями.

Вимоги до безпеки визначають необхідність захисту даних користувачів і запобігання зловмисним атакам на веб-сайт. Всі чутливі дані, такі як особиста інформація та платіжні реквізити, повинні передаватися в зашифрованому вигляді, щоб забезпечити їх конфіденційність. Доступ до адміністративної панелі та чутливих функцій має бути обмежений і захищений складними паролями, а також,

при можливості, двофакторною автентифікацією. Необхідно проводити регулярні перевірки на вразливості, щоб виявляти та усувати потенційні загрози. Веб-сайт повинен забезпечувати захист від загроз, таких як SQL-ін'єкції, міжсайтові скрипти (XSS) та інші зловмисні дії. Логи доступу та активності користувачів повинні бути зафіксовані та аналізовані для виявлення підозрілої активності.

Вимоги до інтерфейсу включають необхідність створення інтуїтивно зрозумілого та зручного дизайну, що забезпечує комфортне використання на всіх пристроях. Інтерфейс повинен мати чітку структуру, з легким доступом до основних функцій і можливістю швидкої навігації. Важливо, щоб кольорові рішення, шрифти та елементи дизайну були узгоджені, що підвищує естетичну привабливість і впізнаваність бренду. Взаємодія з інтерфейсом повинна бути плавною, без затримок або зайвих анімацій, які можуть відволікати користувачів. Всі елементи управління, такі як кнопки та меню, повинні бути достатньо великими для зручного використання на сенсорних пристроях. Інтерфейс повинен враховувати різні категорії користувачів, включаючи людей з обмеженими можливостями, що вимагає дотримання стандартів доступності.

2.4. Проектування функціональних можливостей макету

Моделювання предметної області для розробки програмного модуля, що забезпечує інтелектуальну власність, є критично важливим етапом у створенні програмного забезпечення. На цьому етапі виявляються ключові сутності та встановлюються зв’язки між ними, що формує основу для подальшої розробки системи. У цьому вступі ми розглянемо основні сутності та їх взаємодії, що дозволить краще усвідомити основні об'єкти і процеси, що відбуваються в системі.

UML (Unified Modeling Language) — це універсальна мова візуального моделювання, створена для специфікації, візуалізації, проектування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. Ця мова є потужним і зручним інструментом моделювання, який ефективно застосовується для створення концептуальних, логічних і графічних моделей складних систем різного призначення.

Застосування UML базується на розумінні загальних принципів моделювання складних систем і специфіки об'єктно-орієнтованого проектування. Один із

основних принципів — це абстрагування, що передбачає включення в модель лише тих аспектів системи, які безпосередньо впливають на її функціональність чи цілі.

Ще один важливий принцип — це багатомоделювання, який говорить про те, що одна єдина модель не може адекватно відобразити всі аспекти складної системи. Це означає, що повна модель може включати кілька взаємопов'язаних представлень, кожне з яких відображає певний аспект її поведінки чи структури.

Також важливим є принцип ієрархічності, який вказує на те, що модель системи може бути досліджена на різних рівнях абстракції або деталізації в межах фіксованих представлень.

Застосування UML також сприяє покращенню комунікації в команді та з клієнтом, адже діаграми, які використовуються в UML, дозволяють графічно відображати структуру та поведінку системи, що спрощує її розуміння та аналіз.

Розглянемо діаграму прецедентів для програмного модуля. Актори в системі є зовнішніми сутностями, які взаємодіють із системою, виконуючи певні дії або пред’являючи вимоги. Прецеденти, у свою чергу, визначають функціональну поведінку системи, не вдаючись до деталей реалізації. Основна мета прецедентів — описати, що система повинна робити відповідно до вимог користувачів, без конкретизації реалізації.

Діаграма прецедентів складається з акторів, прецедентів (варіантів використання), які обмежені межами системи, а також асоціацій між акторами та прецедентами, а також зв’язків між самими прецедентами. Діаграма також може містити відносини узагальнення між акторами. Ця діаграма є засобом візуалізації моделі варіантів використання системи, де кожен елемент представляється відповідним символом. Розглянемо діаграму прецедентів на рис. 2.15.

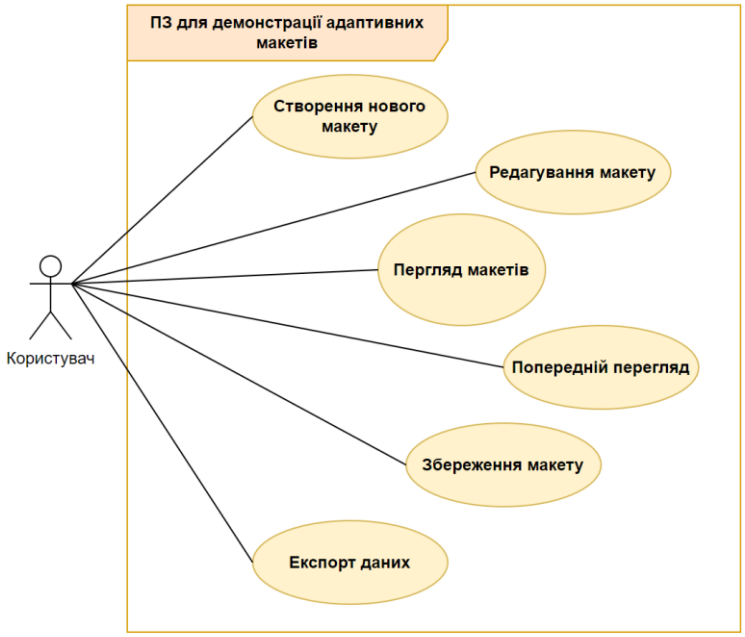

Рис.2.15. – Діаграма прецедентів системи створення адаптивних макетів

Діаграма прецедентів для програмного забезпечення, призначеного для демонстрації адаптивних макетів, ілюструє взаємодію користувача з системою. Основні функціональні можливості включають створення нового макету, редагування існуючого, перегляд макетів, попередній перегляд, збереження макета, а також експорт даних. Користувач може виконувати ці дії через інтерфейс системи, що дозволяє ефективно управляти адаптивними макетами.

Діаграма активності є візуальним інструментом, що демонструє послідовність дій і взаємозв'язки між ними в рамках певного процесу. Вона надає чітке уявлення про робочі етапи, зокрема виконання завдань, прийняття рішень та умови, які можуть вплинути на перебіг процесу.

Основні елементи діаграми активності включають дії, які представлені у вигляді прямокутників, та розгалуження, що показують можливі варіанти розвитку подій, оформлені у вигляді ромбів. Потоки, які з'єднують ці елементи, вказують на напрямок руху та послідовність виконання завдань.

Такі діаграми широко використовуються в бізнес-аналізі, програмуванні, управлінні проектами та навчанні, оскільки допомагають візуалізувати складні процеси, виявляти вузькі місця, а також покращувати комунікацію між учасниками проекту. Наприклад, для демонстраційного сайту адаптивних макетів, діаграма активності може відображати етапи від створення нового макету до його збереження та експорту, що дозволяє краще зрозуміти дії користувача і взаємодію

з системою. Розглянемо детальніше на рис. 2.16.

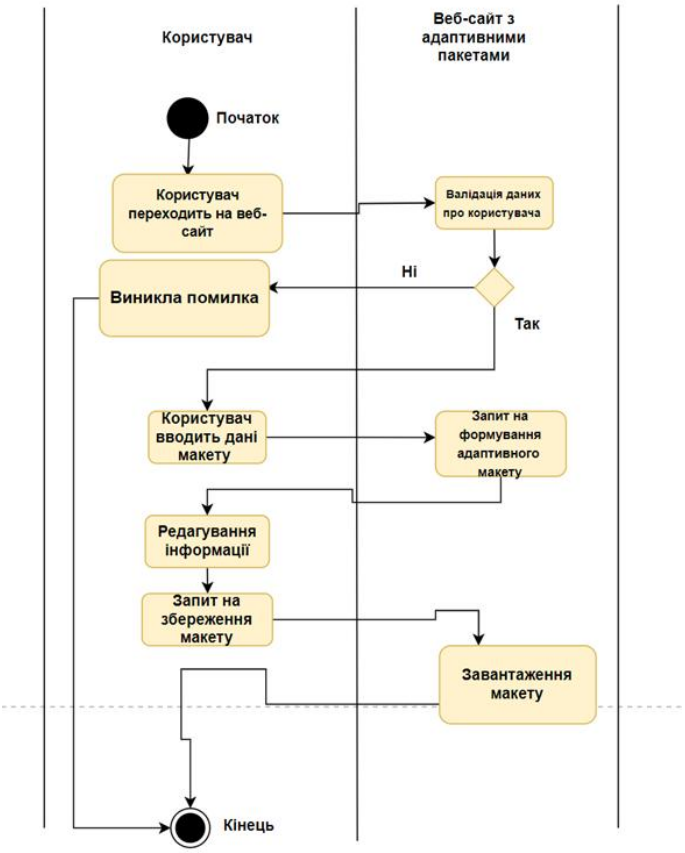

Рис.2.16. – Діаграма активності програмної системи створення адаптивних макетів

Діаграма активності є важливим інструментом для аналізу та оптимізації процесів, оскільки вона надає чітке уявлення про послідовність дій, взаємозв'язки між ними та можливі варіанти розвитку подій. Використання цієї діаграми в рамках проекту, наприклад, для демонстраційного сайту адаптивних макетів, дозволяє зрозуміти етапи роботи користувача, виявити потенційні проблеми та покращити загальну ефективність процесу. Завдяки візуалізації складних робочих етапів, діаграми активності сприяють покращенню комунікації між учасниками проекту та забезпечують більш ефективне управління ресурсами і часом. Таким чином, вони є незамінним інструментом у сфері управління проектами та бізнес-аналізу.

2.5. Висновок до другого розділу

У даному розділі було проведено детальний аналіз процесів проектування алгоритмічного та програмного забезпечення для розробки макетів. Основна увага

приділена теоретичним і практичним аспектам створення гнучких та адаптивних макетів, що є ключовими для забезпечення ефективної взаємодії з різними типами пристроїв та екранів.

У першу чергу, розглянуто теоретичні основи гнучких та адаптивних макетів. Це дозволило зрозуміти їхню важливість у контексті сучасних вимог до веб-дизайну. Особлива увага була приділена принципам, які забезпечують можливість коректного відображення інтерфейсів на різних платформах і пристроях.

Далі проаналізовано алгоритмічні підходи, які відіграють важливу роль у розробці макетів. Алгоритмічні методи дозволяють автоматизувати процеси створення адаптивних інтерфейсів, що суттєво полегшує роботу розробників і зменшує ризик помилок. Важливим етапом було також визначення ключових вимог до адаптивних веб-макетів, що забезпечують їхню функціональність і відповідність очікуванням користувачів.

На завершення, розглянуто процес проектування функціональних можливостей макетів, що має на меті не лише створення візуально привабливого інтерфейсу, а й забезпечення зручності використання. Проектування таких макетів спрямоване на оптимізацію взаємодії користувача з веб-додатком та підвищення ефективності роботи з ним.

Загалом, даний розділ сформував комплексне розуміння процесів розробки адаптивних макетів, що є основою для їх подальшого вдосконалення в контексті сучасних вимог до веб-інтерфейсів.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ АДАПТОВАНОГО МАКЕТУ

3.1. Розробка гнучкого та адаптивного макету

У рамках цього проекту, ми зосередили увагу на розробці гнучкого та адаптивного макету для спортивного веб-сайту кросфіт-клубу університету. Основна мета полягала в тому, щоб створити сучасний сайт, який ефективно комунікує з цільовою аудиторією — студентами, викладачами та іншими відвідувачами університету, що зацікавлені у фітнесі та спортивних активностях, зокрема у кросфіті. Це передбачало не лише стильний та функціональний дизайн, але й повну адаптацію до різних екранів і пристроїв, на яких користувачі переглядатимуть сайт.

Кросфіт, як один із сучасних та популярних видів спорту, що поєднує елементи високоінтенсивних інтервальних тренувань, важкої атлетики, гімнастики і кардіо, став ідеальним прикладом для створення динамічного і мотивуючого веб-сайту. Сайт мав відображати енергію і силу, які асоціюються з кросфітом, і водночас бути простим і легким для навігації, щоб користувачі легко знаходили інформацію про тренування, розклад занять, фото-галереї з подій клубу, а також змогли записатися на секції або отримати консультацію.

Тема кросфіту ідеально вписувалася в концепцію університетського клубу, оскільки спортивні секції в закладах вищої освіти відіграють важливу роль у підтриманні здорового способу життя студентів. Сайт кросфіт-клубу повинен був не лише розповідати про переваги цього виду тренувань, але й мотивувати молодь до активної участі, адже, як відомо, спорт підвищує фізичну витривалість, дисциплінує і сприяє поліпшенню академічних результатів.

Наш проект полягав у розробці багатофункціонального веб-сайту, здатного адаптуватися під різні пристрої, та забезпечити швидкий і зручний доступ до інформації для всіх категорій відвідувачів. Основні завдання включали:

1. створення сучасного та привабливого дизайну, який відображав би спортивну тематику і динамічність кросфіту.
2. Розробка адаптивної верстки, що автоматично підлаштовується під різні розміри екранів, дозволяючи користувачам однаково зручно взаємодіяти з

сайтом як на мобільних пристроях, так і на стаціонарних комп'ютерах.

3. Інтеграція інтерактивних елементів, таких як галереї, відео, форми для зворотного зв’язку і підписки на новини, що збільшує залученість користувачів і полегшує комунікацію з адміністрацією клубу.

4. Мотиваційний контент, який підкреслює важливість активного способу життя та переваги участі у спортивних заходах університету, зокрема заняттях кросфітом.

Проект розпочався з детального планування структури сторінки. Ми вирішили побудувати сайт на основі декількох ключових секцій, що легко сприймаються і швидко завантажуються на різних пристроях. Кожна секція мала свій унікальний контент, що відображає різні аспекти діяльності клубу, починаючи з розкладу тренувань та мотивуючих відео і закінчуючи інформацією про тренерів та фото звітами з подій.

Для забезпечення гнучкості ми використовували такі сучасні підходи до розробки, як flexbox та grid layout. Ці технології дозволяють легко розміщувати елементи сторінки і забезпечують плавний перехід від одного розміру екрану до іншого, що є важливим для зручності користувача. Важливим елементом також стало використання медіа-запитів для адаптації дизайну під різні пристрої. Ми детально пропрацювали відображення кожної секції на мобільних телефонах, планшетах та великих екранах, щоб користувач міг легко взаємодіяти з контентом незалежно від технічних можливостей свого пристрою.

Також значну увагу приділили роботі з формами — для підписки на новини клубу, реєстрації на тренування або отримання додаткової інформації. Кожна форма мала бути простою і зрозумілою, з чіткими інструкціями та мінімалістичним дизайном.

CSS Grid використовується для побудови сіток (груп елементів, які розташовані в декілька рядків або колонок), що дозволяє автоматично підлаштовувати елементи на сторінці залежно від розміру екрану. У коді CSS Grid використовується для кількох секцій, що можемо розглянути на рис. 3.1.

```
518  .training-types-list {
519    display: grid;
520    grid-template-columns: repeat(6, 1fr);
521    gap: 60px;
522  }
```

Рис.3.1. – Побудова сіток.training-types-list

Ця сітка складається з шести колонок, кожна з яких рівномірно займає доступний простір, що дозволяє відобразити контент у шість стовпців на широких екранах. Такий підхід є дуже ефективним для великих моніторів, де розмір екрану дозволяє одночасно відобразити велику кількість інформації без втрати читабельності. Можемо переглянути на рис.3.2.

Однак для забезпечення оптимального користувацького досвіду на пристроях з меншими екранами (наприклад, планшетах і смартфонах), сітка автоматично підлаштовується за допомогою медіа-запитів. При зменшенні ширини екрану, кількість колонок зменшується, щоб контент не виглядав занадто дрібним або тісно розташованим.

```
@media (max-width: 1280px) {
  .training-types-list {
    grid-template-columns: repeat(3, 1fr); /* Три колонки на середніх екр
  }
}
@media (max-width: 767px) {
  .training-types-list {
    grid-template-columns: repeat(2, 1fr); /* Дві колонки на малих екрана
  }
}
```

Рис.3.2. – Відображення роботи колонок

Варто звернути увагу що на поданому рисунку також використовуються медіа-запити які більш детально ми розглянемо далі. Також варто розглянути секцію .location на рис.3.3.

```
.location {
  display: grid;
  grid-template-columns: repeat(2, 1fr); /* Дві колонки, кожна
}
```

Рис.3.3. – Створення сітки з двох колонок

У секції з локацією використовується сітка з двох колонок. На менших екранах ця сітка змінюється на вертикальну структуру завдяки медіа-запиту на рис. 3.4.

```
@media (max-width: 1024px) {
  .location {
    flex-direction: column-reverse; /* Вертикальне відображен
  }
}
```

Рис.3.4. – Зміна структури завдяки медіа-запиту

Далі перейдемо до розгляду використання flexbox у програмному коді . Контейнер, у якому використовується flexbox, може динамічно змінювати розташування своїх дочірніх елементів. Коли простір змінюється (наприклад, при зміні розміру вікна браузера), елементи автоматично підлаштовуються під нові умови, залишаючись у межах контейнера. Розглянемо детальніше на рис. 3.5.

```
.header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  column-gap: 20px; /* Відстань між елементами */
}
```

Рис.3.5. – Використання flexbox

У поданому коді flexbox забезпечує рівномірне розташування елементів шапки (логотип, меню, кнопки) на горизонтальній лінії. Властивість justify-content: space-between розподіляє елементи таким чином, що вони розташовані на максимальній відстані один від одного, займаючи весь доступний простір. Окрім горизонтального вирівнювання, flexbox дозволяє легко вирівнювати елементи по вертикалі. Це особливо корисно для центрованих або вирівняних за серединою блоків, що продемонстровано на рис. 3.6.

```
.header-actions {
  display: flex;
  align-items: center; /* Центрує елементи по вертикалі */
  column-gap: 40px; /* Відстань між елементами */
}
```

Рис.3.6. – Вирівнювання елементів по вертикалі

У роботі активно використовується і flexbox і css grid і медіа-запити.

У нашій роботі медіа-запити активно використовуються для адаптації макета під різні розміри екранів. Вони дозволяють змінювати структуру та зовнішній вигляд елементів залежно від ширини екрана, щоб забезпечити зручність

користування як на великих екранах, так і на мобільних пристроях.

Важлива частина нашого макета, зокрема сітка для відображення елементів у секції "training-types", адаптується під різні розміри екранів. На великих екранах сітка складається з шести колонок, що дозволяє показувати багато елементів одночасно. Однак, коли ширина екрана зменшується, кількість колонок зменшується до трьох на середніх екранах і до двох на мобільних пристроях. Це дозволяє контенту залишатися зручним для перегляду без необхідності прокручування вбік.

Для таких секцій, як "location" та "motivation-card", медіа-запити використовуються для зміни розташування елементів. Наприклад, на великих екранах елементи можуть бути розташовані в два ряди (горизонтально), але на маленьких пристроях вони змінюють свою орієнтацію на вертикальну. Це гарантує, що контент залишиться структурованим і легко сприймається користувачем на будь-якому пристрої.

Медіа-запити також забезпечують автоматичне перенесення елементів, таких як меню навігації або кнопки, на нові рядки, коли екран стає занадто вузьким для їх горизонтального відображення. Це робить наш дизайн більш гнучким і адаптивним до різних розмірів екранів.

Ще однією важливою функцією медіа-запитів є зміна розміру шрифтів та відступів між елементами. На великих екранах використовуються більші шрифти та ширші відступи, що робить контент легким для читання. На менших екранах шрифти зменшуються, а відступи стають меншими, щоб елементи компактніше розташовувалися на сторінці і не займали надмірно багато місця.

Медіа-запити також використовуються для адаптації окремих інтерфейсних елементів, таких як кнопки та форми. Наприклад, на маленьких екранах кнопки можуть збільшуватися в розмірах для зручнішого натискання на сенсорних екранах. Такі зміни роблять інтерфейс більш зручним для користувача на мобільних пристроях. Частку використання технологій можемо розглянути на діаграмі 3.7.

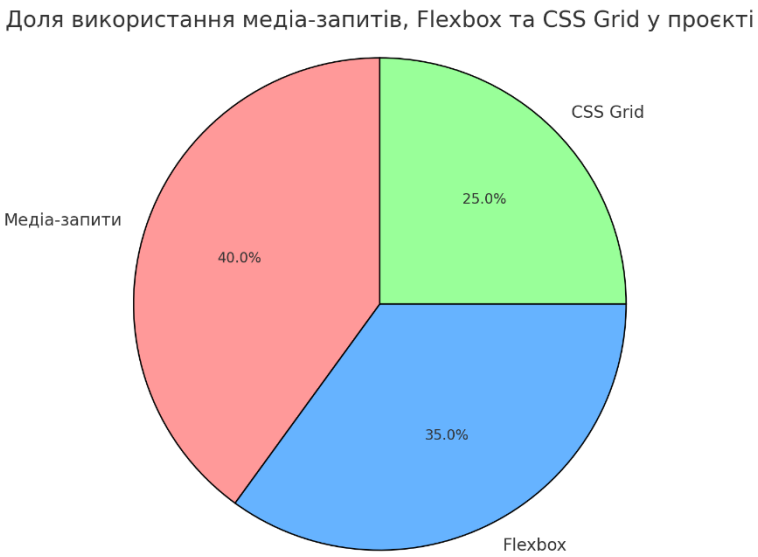

Рис.3.7. – Доля використання медіа-запитів у проєкті

Відповідно до аналізу, приблизно 40% всіх стилів у проєкті припадає на медіа-запити, які забезпечують адаптивність макета під різні розміри екранів. 35% становить Flexbox, що використовується для гнучкого вирівнювання елементів та управління простором у контейнерах. 25% відведено для CSS Grid, який дозволяє створювати складніші сітки для компонування контенту.

Це розшарування демонструє, що основний акцент у проєкті робиться на адаптивність через медіа-запити, тоді як Flexbox і CSS Grid використовуються для побудови гнучких і структурованих макетів.

3.2. Тестування та оптимізація макету

Тестування та оптимізація макету є критично важливими етапами в розробці будь-якого веб-проєкту. Після завершення дизайну та реалізації функціональних частин сайту постає питання забезпечення його стабільної та коректної роботи на всіх можливих пристроях і в різних браузерах. У наш час користувачі можуть відвідувати веб-сайти не лише з настільних комп'ютерів, а й зі смартфонів, планшетів та інших мобільних пристроїв, тому важливо, щоб сайт залишався зручним для всіх користувачів.

У нашому проєкті тестування зосереджено на адаптивності макету, швидкості завантаження сторінок та загальному користувацькому досвіді. Крім цього, оптимізація спрямована на покращення продуктивності, зменшення часу завантаження та забезпечення стабільної роботи веб-сторінки на всіх етапах її

взаємодії з користувачем.

Кросбраузерне тестування є важливим етапом у процесі розробки веб-сайтів, оскільки різні браузери можуть по-різному обробляти CSS, JavaScript та інші технології веб-розробки. Кожен браузер використовує власний рендеринг механізм для відображення сторінки, що може призвести до відмінностей у відображенні або функціонуванні окремих елементів сайту. Тому кросбраузерне тестування допомагає виявити та усунути можливі проблеми до того, як сайт потрапить до кінцевого користувача.

Основна мета цього процесу — забезпечити однаковий користувацький досвід для всіх відвідувачів сайту незалежно від браузера або платформи. Тестування дає змогу перевірити, як сайт виглядає та працює у найпопулярніших браузерах і операційних системах. Оскільки різні браузери мають власні алгоритми для обробки коду, важливо переконатися, що сайт виглядає і функціонує однаково в усіх середовищах.

Для кросбраузерного тестування можуть використовуватись як фізичні пристрої, так і інструменти для емуляції роботи браузерів та платформ. Найбільш поширеними інструментами для цього є BrowserStack та LambdaTest, які дозволяють перевіряти сайт у реальних умовах на різних версіях браузерів та операційних систем.

1. BrowserStack: цей інструмент забезпечує доступ до різних версій браузерів, операційних систем, а також мобільних пристроїв. Він дозволяє тестувати сайти як на реальних пристроях, так і в емуляторах, що дає можливість виявити навіть найдрібніші відмінності у відображенні контенту.

2. LambdaTest: це ще один популярний інструмент для тестування, який надає широкий спектр браузерів, платформ і пристроїв. LambdaTest також дозволяє тестувати мобільні та настільні версії сайтів, забезпечуючи точне відображення реального середовища користувачів.

Для забезпечення коректної роботи сайту необхідно протестувати його на таких браузерах:

- Google Chrome (найпопулярніший браузер у світі, його використовують більшість користувачів)
- Mozilla Firefox (відомий своєю підтримкою новітніх веб-стандартів)

- Microsoft Edge (браузер від Microsoft, який використовує той самий механізм рендерингу, що й Chrome)
- Safari (основний браузер для користувачів macOS та iOS)
- Opera (браузер з власними унікальними функціями та механізмами)

У таблиці 3.1. представлено результати тестування на різних браузерах і платформах. Для кожного браузера перевірялась коректність відображення макету та функціональність основних елементів, таких як кнопки, форми та медіа-контент.

Таблиця 3.1.

Результати тестування у різних браузерах

Браузер	Операційна система	Версія	Результат тестування
Google Chrome	Windows 10	96.0	Пройдено успішно
Google Chrome	macOS	96.0	Пройдено успішно
Продовження таблиці 3.1.			
Google Chrome	Android	95.0	Пройдено успішно
Mozilla Firefox	Windows 10	94.0	Пройдено успішно
Mozilla Firefox	macOS	94.0	Пройдено успішно
Microsoft Edge	Windows 10	95.0	Пройдено успішно
Safari	macOS	15.0	Пройдено успішно
Safari	iOS	15.0	Пройдено успішно
Opera	Windows 10	80.0	Пройдено успішно

Для тестування швидкості завантаження сайту використовуються інструменти, такі як Google PageSpeed Insights, GTmetrix або Lighthouse. Вони надають докладну інформацію про ключові фактори, що впливають на швидкість завантаження, та пропонують рекомендації щодо їх оптимізації. Один з основних підходів до оптимізації полягає в мінімізації ресурсів веб-сайту. Це стосується зокрема CSS, JavaScript та HTML, де зайві пробіли, коментарі та непотрібний код можуть бути видалені. Крім того, об’єднання кількох файлів в один дозволяє зменшити обсяг даних, що повинні бути завантажені браузером, і таким чином прискорити процес завантаження сторінки. Для мінімізації ресурсів використовуються інструменти на кшталт CSSNano або UglifyJS, які автоматично оптимізують файли перед публікацією на сервері.

Оптимізація зображень також є важливим аспектом, оскільки вони займають значну частину ресурсу будь-якого веб-сайту. Використання сучасних форматів, таких як WebP, дозволяє зберігати якісне відображення зображень при мінімальних розмірах файлів. Інструменти для стиснення, такі як TinyPNG або ImageOptim, допомагають зменшити розмір зображень без втрати якості. Також важливою є техніка "ледачого завантаження" (lazy loading), коли зображення завантажуються лише тоді, коли вони стають видимими на екрані користувача. Це значно зменшує загальний час завантаження сторінки, особливо на мобільних пристроях або при повільному інтернет-з'єднанні.

Використання кешування є ще одним ефективним методом оптимізації веб-сайту. Завдяки кешуванню статичні ресурси, такі як зображення, шрифти та стилі, зберігаються у браузері користувача, що дозволяє уникнути їх повторного завантаження під час повторних відвідувань сторінки. Це значно прискорює завантаження для повторних користувачів і знижує навантаження на сервер. Оптимізація шрифтів також впливає на швидкість завантаження. Сучасні формати шрифтів, зокрема WOFF2, забезпечують високу якість тексту при мінімальних розмірах файлів. Важливо використовувати лише ті шрифти та варіанти накреслення, які дійсно потрібні для дизайну, щоб зменшити їх кількість і, відповідно, розмір завантажуваних файлів.

Ще один спосіб оптимізації — це використання мережі доставки контенту (CDN). CDN дозволяє розміщувати копії статичних ресурсів веб-сайту на серверах, розташованих у різних частинах світу. Це скорочує затримки під час завантаження контенту для користувачів, незалежно від їхнього місцезнаходження. Використання CDN сприяє прискоренню завантаження сторінки та забезпеченню стабільної роботи сайту навіть у періоди високого навантаження. Графік внеску кожного методу можемо розглянути на рис. 3.8.

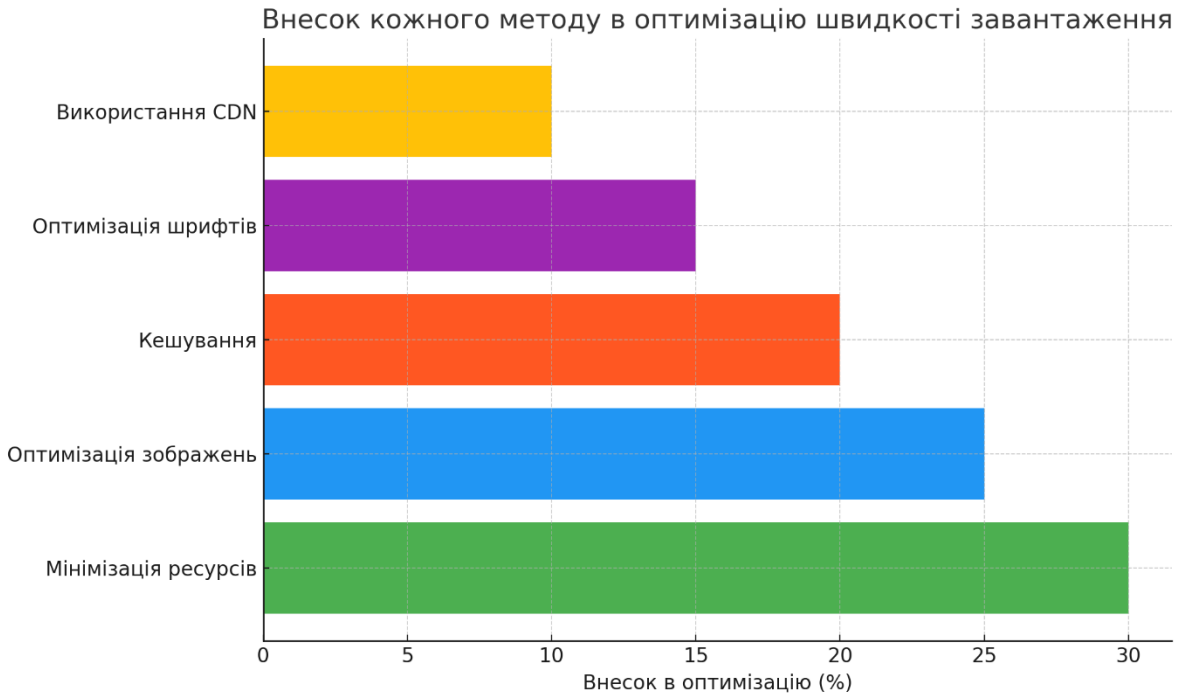

Рис.3.8. – Графік внеску кожного методу в оптимізацію швидкості завантаження

На графіку зображено внесок кожного з методів оптимізації в загальну швидкість завантаження веб-сторінок. Найбільший вплив має мінімізація ресурсів, яка забезпечує близько 30% оптимізації, тоді як оптимізація зображень та кешування займають важливе місце, складаючи відповідно 25% і 20%. Оптимізація шрифтів та використання мережі доставки контенту (CDN) також роблять свій внесок, хоча й менш значний, але разом ці методи забезпечують ефективну та швидку роботу веб-сторінок.

3.3. Огляд реалізованих можливостей

Ми зайшли на головну сторінку нашого веб-ресурсу, де представлено демонстраційний адаптивний макет, присвячений темі кросфіту. Інтерфейс сайту побудований з використанням сучасних технологій веб-розробки, що забезпечують гнучкість та швидкість завантаження. Детальніше на рис. 3.9.

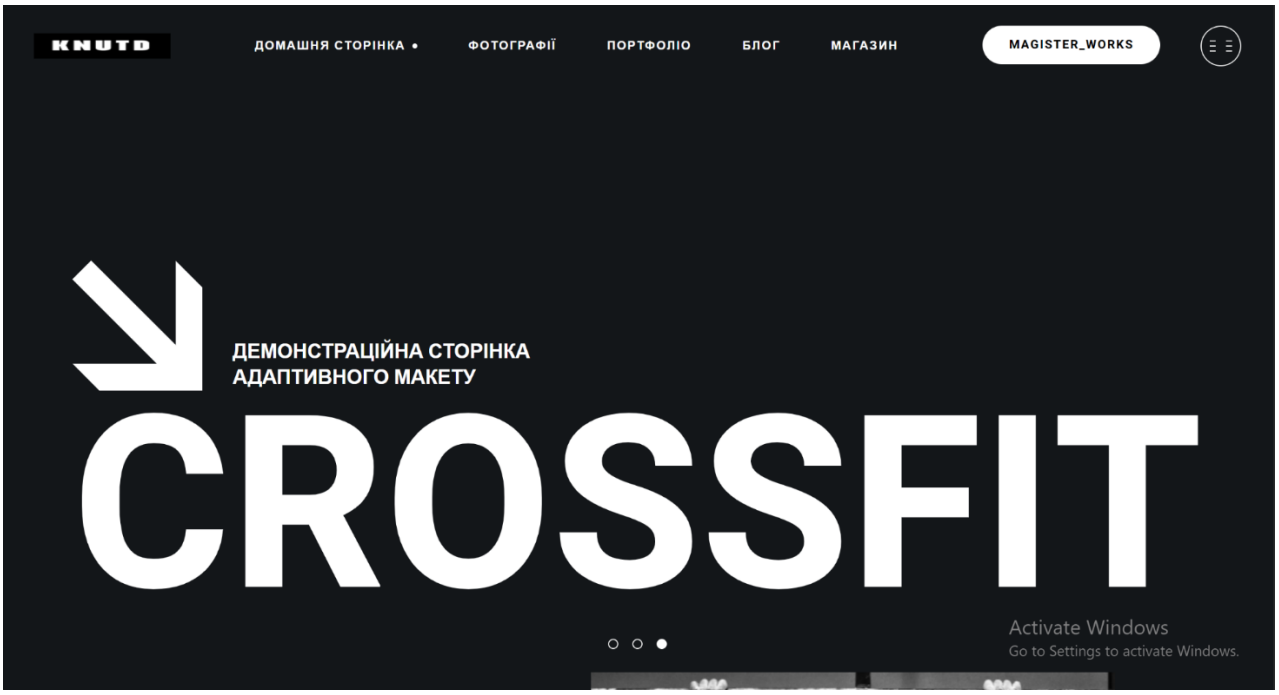

Рис.3.9. – Головна сторінка демонстраційного макету

На цій сторінці користувачі можуть ознайомитися з ключовими розділами ресурсу, такими як домашня сторінка, фотографії, портфоліо, блог та магазин. Логічно організоване меню дозволяє швидко переміщуватися між різними розділами сайту, а велика, привітальна графіка "Crossfit" створює сильне перше враження та підкреслює тематику веб-ресурсу. Розглянемо на рис. 3.10 зміну макету при зміні розмірів екрану.

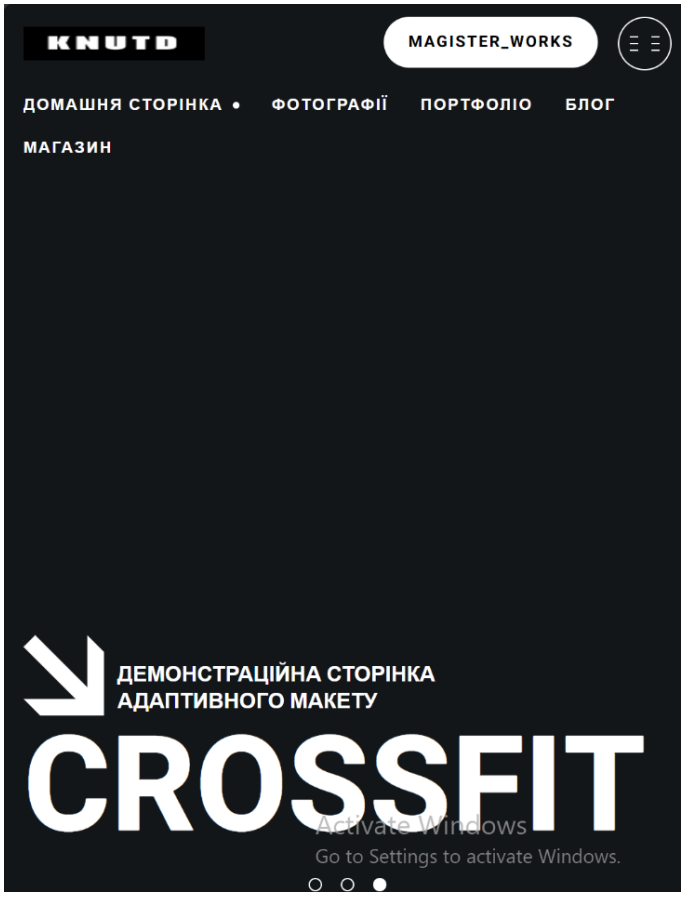

Рис.3.10. – Наший макет при зміні розмірів вікна

При зміні розміру екрану інтерфейс нашого сайту адаптується за допомогою технологій, таких як CSS Grid і Flexbox, що забезпечують плавне та гнучке перерозподілення елементів на сторінці. Коли користувач переглядає сайт на пристроях з різними розмірами екранів, наприклад на мобільних телефонах чи планшетах, структура сторінки автоматично підлаштовується. Завдяки медіа-запитам CSS, навігаційне меню, зображення, текстові блоки та інші елементи зберігають свою функціональність і зручність у використанні, забезпечуючи комфортний користувацький досвід на будь-якому пристрої.

Футер на нашому сайті є важливою частиною адаптивного макету, що підлаштовується під різні розміри екранів. Він містить необхідну інформацію для користувачів і водночас залишається простим та функціональним. При зміні розміру екрана, компоненти футера, такі як логотип, контактна інформація, години роботи та форма підписки, динамічно змінюють своє розташування завдяки використанню CSS Grid і Flexbox. Це дозволяє футеру залишатися читабельним та зручним як на широких екранах комп'ютерів, так і на мобільних пристроях. Розглянемо на рис. 3.11.

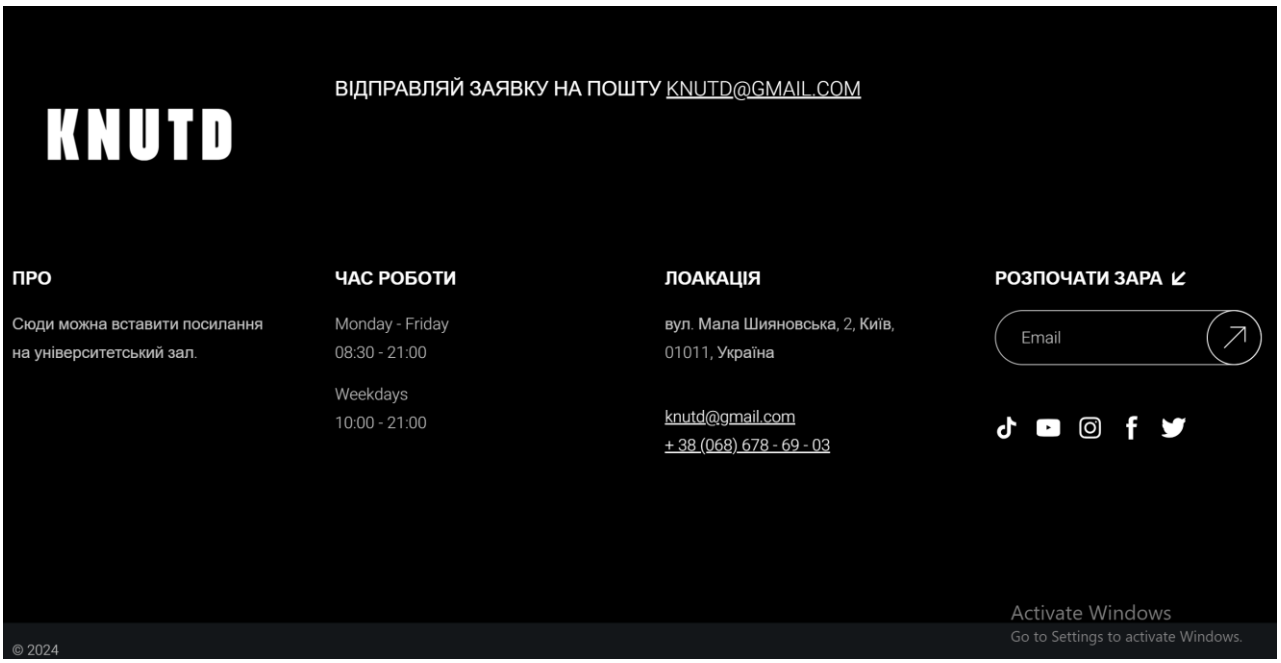

Рис.3.11. – footer нашого макету

Контент, розподілений у кілька колонок на великих екранах, у мобільній версії адаптується в стовпці, що дозволяє легко взаємодіяти з інформацією без втрати елементів або їх функціональності. Завдяки таким адаптивним технологіям,

футер залишається зручним для користувачів незалежно від пристрою, на якому він переглядається. Що можемо переглянути на рис. 3.12.

Рис.3.12. – Зміна розмірів макету відповідно до зміни розмірів вікна

Усі елементи контенту на нашому сайті були адаптовані за допомогою CSS та, що забезпечує його коректне відображення на різних пристроях і розмірах екранів. Завдяки використанню таких технологій, як CSS Grid, Flexbox і медіа-запити, макет підлаштовується під ширину вікна браузера, зберігаючи функціональність та зручність користування. Це дозволяє користувачам отримати однаково якісний досвід на будь-якому пристрої, від настільного комп'ютера до мобільного телефону.

3.4. Висновки до третього розділу

У третьому розділі було детально розглянуто та реалізовано процес створення адаптивного веб-макету, що підлаштовується під різні розміри екранів і пристроїв. Починаючи з розробки гнучкого дизайну, основним завданням було забезпечити, щоб користувачі, незалежно від того, з якого пристрою вони заходять на сайт, отримували однаково зручний і функціональний інтерфейс.

Було використано технології CSS Grid та Flexbox для побудови динамічного

макету, який здатний адаптуватися до різних розширень екрана. Завдяки використанню цих інструментів, вдалося реалізувати систему, в якій елементи сторінки можуть змінювати своє розташування та розміри відповідно до розмірів вікна браузера. Важливим аспектом стала організація контенту, яка завдяки цим технологіям залишалася цілісною та зрозумілою незалежно від пристрою.

Окрім базового налаштування адаптивності, у розділі також було проведено тестування та оптимізацію макету. Це включало перевірку відображення сторінки в різних браузерах та на різних платформах, щоб гарантувати коректну роботу всіх елементів і функцій. Особлива увага приділялася швидкості завантаження сторінок, яка була оптимізована завдяки мінімізації ресурсів, оптимізації зображень і використанню кешування. Використання медіа-запитів дозволило задавати різні стилі для різних розмірів екранів, забезпечуючи плавний перехід між ними.

У процесі оптимізації було досягнуто значних результатів у зниженні часу завантаження сторінок і підвищенні продуктивності сайту. Оптимізація зображень, використання сучасних форматів та техніка "ледачого завантаження" зробили роботу сайту більш ефективною. Кешування та використання CDN допомогли забезпечити швидке завантаження контенту незалежно від місця знаходження користувача, що покращило загальний користувацький досвід.

Загалом, результати, отримані в цьому розділі, продемонстрували важливість використання сучасних інструментів для створення гнучких і адаптивних веб-сторінок. Тестування підтвердило, що адаптивний макет відповідає вимогам сучасних стандартів веб-розробки, забезпечуючи зручність використання та високий рівень продуктивності. Впроваджені рішення дали змогу створити універсальний дизайн, що зберігає свою функціональність і привабливість на різних пристроях, що є ключовим фактором успіху в сучасному веб-просторі.

ВИСНОВКИ

Виконуючи даний проєкт, було досягнуто ряд поставлених завдань і цілей. Було визначено, що впровадження сучасних веб-технологій для створення адаптивного дизайну є перспективним напрямком розвитку, який значно підвищує зручність використання сайту для різних категорій користувачів. Адаптивність та гнучкість макетів дозволяють забезпечити коректне відображення веб-контенту на різних пристроях, що є важливою перевагою в сучасному світі, де мобільний трафік займає значну частку.

Проаналізовано і пояснено ключові технології, які були використані в проєкті, зокрема CSS Grid, Flexbox, та медіа-запити. Ці інструменти дозволили побудувати динамічний макет, який адаптується до розмірів екрану, зберігаючи при цьому функціональність і зручність. Це підтверджує важливість використання сучасних веб-технологій для підвищення доступності й зручності користувацького інтерфейсу.

У ході проєкту було створено адаптивний веб-ресурс, який демонструє сучасний підхід до побудови веб-дизайну. Дана платформа включає в себе елементи, які динамічно змінюють свою структуру при зміні розмірів екрана. Завдяки використанню таких технологій, як CSS Grid і Flexbox, було досягнуто плавного переходу між різними розмірами екранів без втрати якості відображення. Це забезпечує відмінний користувацький досвід як для користувачів настільних комп'ютерів, так і для мобільних пристроїв.

Користувачі можуть взаємодіяти з контентом незалежно від того, який пристрій вони використовують, що робить ресурс зручним для використання в будь-якому місці та в будь-який час, якщо є доступ до Інтернету. Це є важливою перевагою адаптивного дизайну, оскільки він забезпечує широкі можливості для користувачів і дозволяє підтримувати зручність і функціональність на всіх платформах.

Веб-ресурс, розроблений у рамках даного проєкту, має сучасний дизайн, інтуїтивно зрозумілий інтерфейс і високу продуктивність завдяки впровадженню принципів адаптивної верстки та оптимізації швидкості завантаження. Проєкт продемонстрував ефективність використання сучасних інструментів для створення гнучких, адаптивних і зручних веб-сайтів, які відповідають вимогам користувачів

незалежно від типу пристрою чи розміру екрану.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. HTML (HyperText Markup Language) – мова розмітки для створення веб-сторінок, що визначає структуру і зміст веб-документів.
2. CSS (Cascading Style Sheets) – каскадні таблиці стилів, що використовуються для опису зовнішнього вигляду веб-сторінок, таких як кольори, шрифти, відступи тощо.
3. Flexbox (Flexible Box Layout) – технологія CSS для створення гнучких макетів веб-сторінок, що дозволяє змінювати розташування елементів у контейнері відповідно до розмірів екрану.
4. CSS Grid – потужний інструмент CSS для побудови двовимірних макетів, що дозволяє розміщувати елементи у вигляді сітки з використанням рядків і стовпців.
5. Медіа-запити (Media Queries) – технологія CSS, що дозволяє застосовувати різні стилі в залежності від характеристик пристрою, наприклад, ширини екрану, дозволу чи орієнтації.
6. Оптимізація зображень (Image Optimization) – процес зменшення розміру файлів зображень без втрати їх якості з метою покращення швидкості завантаження сторінок.
7. Кешування (Caching) – технологія зберігання копій веб-сторінок або окремих ресурсів для прискорення їх завантаження при наступних зверненнях до них.
8. CDN (Content Delivery Network) – мережа доставки контенту, яка розподіляє статичні ресурси веб-сайту по різних серверах у світі для прискорення їх завантаження.
9. Lazy loading (Ледаче завантаження) – техніка, яка дозволяє завантажувати зображення та інші ресурси лише тоді, коли вони з'являються у видимій частині екрану користувача.
10. JavaScript (JS) – мова програмування, що використовується для додавання інтерактивних елементів та функціональності на веб-сторінках.

СПИСКИ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Марк Д. Мітчел. HTML і CSS: розробка і дизайн веб-сайтів. Видавництво: O'Reilly Media, 2012. 490 с.

2. Джон Дакетт. HTML і CSS: Візуальне навчання. Видавництво: Wiley, 2011. 512 с.

3. Бен Фрейн. Адаптивний веб-дизайн з використанням HTML5 і CSS3. Видавництво: Packt Publishing, 2015. 300 с.

4. Етан Маркотт. Адаптивний веб-дизайн. Видавництво: A Book Apart, 2011. 150 с.

5. Ерік А. Мейєр. CSS: Каскадні таблиці стилів. Видавництво: O'Reilly Media, 2017. 480 с.

6. Пол Адамс. HTML5 і CSS3: Будуємо адаптивні сайти. Видавництво: Manning Publications, 2018. 350 с.

7. Рейчел Ендрю, Естан Дей. Елементи CSS Grid Layout. Видавництво: A Book Apart, 2017. 128 с.

8. Янакопулос Д. Практика використання CSS Grid та Flexbox у адаптивній верстці. Видавництво: Packt Publishing, 2020. 200 с.

9. Лариса Вільямс. Адаптивний дизайн інтерфейсів з використанням медіа-запитів. Видавництво: SitePoint, 2016. 256 с.

10. Джен Сіммонс. Веб-дизайн з CSS Grid: Новий підхід до розмітки сторінок. Видавництво: New Riders, 2019. 320 с.

11. Крістофер Шмітт. Використання Flexbox для адаптивних веб-сторінок. Видавництво: Peachpit Press, 2014. 250 с.

12. Бос Берт. CSS – Офіційний довідник. Видавництво: Addison-Wesley Professional, 2019. 960 с.

13. Стівен Гарднер. Впровадження адаптивного дизайну: Шаблони та найкращі практики. Видавництво: Smashing Magazine, 2017. 176 с.

14. Кріс Койєр. CSS Flexbox: Глибоке занурення. Видавництво: CSS-Tricks, 2018. 140 с.

15. Вольфганг Хорн. Оптимізація швидкості завантаження сторінок за допомогою CSS та медіа-запитів. Видавництво: SitePoint, 2020. 220 с.

16. Патрік Лінч, Сара Гортон. Візуальний веб-дизайн: Принципи та практики. Видавництво: Peachpit Press, 2015. 352 с.

17. Кріс Касперскі. Практичні аспекти адаптивної верстки з використанням CSS Grid та Flexbox. Видавництво: Packt Publishing, 2021. 240 с.

18. Mozilla Developer Network. Посібник з використання медіа-запитів для адаптивної верстки. URL: https://developer.mozilla.org/uk/docs/Web/CSS/Media_Queries.

19. Can I Use. Інструмент перевірки сумісності CSS Flexbox та Grid Layout з різними браузерами. URL: https://caniuse.com/.

20. W3Schools. Адаптивний дизайн веб-сторінок. URL: https://www.w3schools.com/css/css_rwd_intro.asp.

ДОДАТКИ

Програмний код інтеграції макету на телефон

```
// 1. Мобільне меню
const burgerButton = document.querySelector('.header-burger-button');
const menu = document.querySelector('.header-menu');

burgerButton.addEventListener('click', function() {
  menu.classList.toggle('is-open');
  burgerButton.classList.toggle('is-active');
});

// 2. Слайдер для банеру
let currentSlide = 0;
const slides = document.querySelectorAll('.banner-pagination-item');
const totalSlides = slides.length;

function changeSlide(slideIndex) {
  slides.forEach((slide, index) => {
      slide.querySelector('.banner-pagination-button').classList.toggle('is-current', index === slideIndex);
  });
  document.querySelector('.banner-title').style.opacity = 0;
  setTimeout(() => {
     document.querySelector('.banner-title').textContent = ['Crossfit', 'Strongman', 'Endurance'][slideIndex]; // змінюємо текст банера
    document.querySelector('.banner-title').style.opacity = 1;
  }, 300);
  currentSlide = slideIndex;
}

slides.forEach((slide, index) => {
```

```
  slide.addEventListener('click', () => changeSlide(index));
});

setInterval(() => {
  currentSlide = (currentSlide + 1) % totalSlides;
  changeSlide(currentSlide);
}, 5000); // кожні 5 секунд

// 3. Відтворення/зупинка відео
const video = document.querySelector('.join-us-video');
const playButton = document.querySelector('.join-us-video-play-button');

playButton.addEventListener('click', function () {
  if (video.paused) {
    video.play();
    playButton.style.display = 'none';
  } else {
    video.pause();
    playButton.style.display = 'block';
  }
});

video.addEventListener('pause', function() {
  playButton.style.display = 'block';
});

// 4. Валідація форми
const form = document.querySelector('.join-us-form');
const nameInput = document.querySelector('#username');
const emailInput = document.querySelector('#email');

form.addEventListener('submit', function(event) {
```

```
  event.preventDefault();
  let hasError = false;

  if (!nameInput.value.trim()) {
   alert('Введіть своє ім’я');
   nameInput.focus();
   hasError = true;
  }

  if (!emailInput.value.includes('@')) {
   alert('Введіть коректний емейл');
   emailInput.focus();
   hasError = true;
  }

  if (!hasError) {
   alert('Дякуємо за реєстрацію!');
   form.reset();
  }
 });
```

Калькулятор BMI

```
// 5. Калькулятор BMI
 const bmiForm = document.querySelector('.calculate-form');
 const heightInput = document.querySelector('#height');
 const weightInput = document.querySelector('#weight');
 const bmiResult = document.querySelector('.calculate-table');

 bmiForm.addEventListener('submit', function(event) {
  event.preventDefault();

  const height = parseFloat(heightInput.value) / 100;
```

```
  const weight = parseFloat(weightInput.value);

  if (isNaN(height) || isNaN(weight)) {
    alert('Введіть коректні дані');
    return;
  }

  const bmi = (weight / (height * height)).toFixed(2);
  let status = '';

  if (bmi < 18.5) {
    status = 'Худощавий';
  } else if (bmi >= 18.5 && bmi <= 24.9) {
    status = 'Здоровий';
  } else if (bmi >= 25 && bmi <= 29.9) {
    status = 'Надлишкова вага';
  } else {
    status = 'Ожиріння';
  }

  bmiResult.innerHTML = `
    <tr>
      <td>${bmi}</td>
      <td>${status}</td>
    </tr>
  `;
});
```