КВАЛІФІКАЦІЙНА РОБОТА

Група МІПЗс-22

Піхоцький Н.Т.

2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

на правах рукопису

Піхоцький Назарій Тарасович

УДК 004.02

Дослідження та розробка нових підходів для забезпечення модульності, перевикористання коду та полегшення супроводу у веб-дизайні

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття кваліфікації магістра

Нормоконтроль

_____Стисло О.В.
(підпис, дата, розшифрування підпису)

Студент

_____ Піхоцький Н.Т.
(підпис, дата, розшифрування підпису)

Допускається до захисту
Завідувач кафедри

_____к.т.н., доц. Ващишак С.П.
(підпис, дата, розшифрування підпису)

Керівник роботи

_____ к.ф.м.н., доц. Бойчук А.М.
(підпис, дата, розшифрування підпису)

Івано-Франківськ – 2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

Освітній ступінь: «магістр»

Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

« 19 » лютого 2024 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Піхоцький Назарій Тарасович

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Дослідження та розробка нових підходів для забезпечення модульності, перевикористання коду та полегшення супроводу у веб-дизайні

керівник роботи:
Бойчук Андрій Михайлович, кандидат фізико-математичних наук, доцент

затверджена наказом вищого навчального закладу від « 26» червня 2023 року
№ 31/1с

2. Термін подання студентом роботи 16.02.2024

3.Вихідні дані роботи: Figma, WebFlow, Figma Jam

4.Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити)

1. Визначення проблематики модульності та пере використання коду, обґрунтування актуальності дослідження в сучасному веб-дизайні.
2. Аналіз існуючих методів забезпечення модульності та перевикористання коду у веб-дизайні.
3. Розробка нових підходів з дотриманням модульності та супроводу на сайт

5. Дата видачі завдання 29.06.2023

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Розділ	Консультант (прізвище, ініціали та посада)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Визначення проблематики модульності та перевикористання коду, обґрунтування актуальності дослідження в сучасному веб-дизайні	29.08.2023	Виконано
2.	Аналіз існуючих методів забезпечення модульності та перевикористання коду у веб-дизайні	20.09.2023	Виконано
3.	Розробка нових підходів з дотриманням модульності та супроводу на сайті	15.11.2023	Виконано
4.	Формування висновків	30.11.2023	Виконано
6.	Оформлення пояснювальної записки	22.12.2023	Виконано
7.	Оформлення графічного матеріалу та підготовка до захисту роботи	11.01.2024	Виконано

Студент _____ Піхоцький Н.Т.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Бойчук А.М.
(підпис) (прізвище та ініціали)

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Сторінка	Опис графічного матеріалу	Сторінка	Опис графічного матеріалу
31	Компонентна архітектура	79	Структура колекції
61	Переваги та недоліки Wix і WebFlow	79	Налаштування полів поста блогу

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці та дослідженні нових підходів для забезпечення модульності, перевикористання коду та полегшення супроводу у веб-дизайні.

У першому розділі було проведено аналіз модульності, перевикористання та розподіленої розробки, проведено обґрунтування актуальності розробки таких методів та оптимізація ресурсів, впровадження нового функціоналу, підвищення якості продукту.

У другому розділі був проведений аналіз методів наприклад методи забезпечення модульності та перевикористання коду, файлова організація, структура каталогів, організація коду з компонентами, порівняння підходів до організації архітектури.

У третьому розділі розглянуто використання інтерактивного веб-редактора WebFlow як інноваційного інструменту для спрощення розробки та візуальної взаємодії з елементами веб-додатків. Розроблено новий спосіб створення компонентів. Створено CMS Collections, Ecommerce Collections.

КЛЮЧОВІ СЛОВА: МОДУЛЬНІСТЬ, ПЕРЕ ВИКОРИСТАННЯ КОДУ, СУПРОВІД, ПІДХОДИ, ОПТИМІЗАЦІЯ, КОМПОНЕНТИ, МЕТОДИ, HTML, CSS, JAVASCRIPT, WEBFLOW, COLLECTION.

SUMMARY

The qualification work is devoted to the development and research of new approaches to ensure modularity, code reuse, and easier maintenance in web design.

The first chapter analyzed modularity, reuse, and distributed development, justified the relevance of developing such methods and optimized resources, introduced new functionality, and improved product quality.

The second section analyzes methods, such as methods of ensuring modularity and code reuse, file organization, directory structure, organization of code with components, comparison of approaches to architecture organization.

The third section discusses the use of the interactive web editor WebFlow as an innovative tool for simplifying development and visual interaction with web application elements. A new way of creating components has been developed. CMS Collections and Ecommerce Collections are created.

KEYWORDS: MODULARITY, CODE REUSE, MAINTENANCE, APPROACHES, OPTIMIZATION, COMPONENTS, METHODS, HTML, CSS, JAVASCRIPT, WEBFLOW, COLLECTION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

VCS – Система контролю версіями

UI – Користувацький інтерфейс

API - Прикладний програмний інтерфейс

MVP – Мінімально життєвий продукт

CMS – Система керування вміст

ВСТУП

Актуальність теми. Веб-дизайн є одним із найважливіших аспектів створення успішного веб-сайту. Він відповідає за те, як веб-сайт виглядає та функціонує, і має вирішальне значення для його візуальної привабливості, зручності використання та ефективності.

Одними з ключових принципів веб-дизайну є модульність та перевикористання коду. Модульна структура веб-сайту дозволяє легко додавати нові функції або змінювати існуючі, а перевикористання коду допомагає економити час і ресурси.

Супроводження веб-проектів також є важливим завданням. Воно включає в себе підтримку веб-сайту в актуальному стані, забезпечення його безпеки та надійності, а також усунення помилок та проблем.

Розробка нових підходів для забезпечення модульності, перевикористання коду та полегшення супроводу у веб-дизайні має важливу практичну цінність. Вона може призвести до створення більш ефективних, гнучких та універсальних веб-сайтів, які легше підтримувати.

Мета і завдання дослідження. Метою дослідження є розробка нових підходів для забезпечення модульності, перевикористання коду та полегшення супроводу у веб-дизайні.

Для досягнення цієї мети необхідно вирішити такі завдання:

- проаналізувати існуючі підходи до забезпечення модульності, перевикористання коду та полегшення супроводу у веб-дизайні;
- визначити недоліки існуючих підходів;
- розробити нові підходи, які усувають недоліки існуючих.

Об’єкт дослідження. Веб-дизайн у контексті створення та супроводу веб-сайтів та оптимізація створення веб сайтів для економії часу та ресурсів.

Предмет дослідження. Розробка підходів для забезпечення модульності, перевикористання коду та полегшення супроводу у веб-дизайні, а також компонентних частин та бібліотек для оптимізації часу його розробки.

Методи дослідження. Для досягнення поставлених завдань будуть використані такі методи дослідження:

- аналіз існуючих способів та принципів розробки веб дизайну;
- розробка нового підходу з використанням шаблонів та бібліотек;
- порівняння ефективності імплементації розроблених підходів з наявними методиками у веб дизайні.

Наукова новизна одержаних результатів. У результаті дослідження отримані такі наукові результати:

- розроблено новий підхід до забезпечення модульності у веб-дизайні, який базується на використанні абстракцій та інтерфейсів;
- розроблено новий підхід до перевикористання коду у веб-дизайні, який базується на використанні шаблонів та бібліотек;
- розроблено новий підхід до полегшення супроводу у веб-дизайні, який базується на використанні інструментів статичного аналізу та моніторингу продуктивності.

Практичне значення одержаних результатів. Одержані результати можуть бути використані для розробки більш ефективних, гнучких та універсальних веб-сайтів, які легше підтримувати.

Конкретні приклади практичного застосування отриманих результатів включають:

- створення веб-сайтів, які легко розширюються та адаптуються до нових вимог;
- скорочення часу та витрат на розробку та підтримку веб-сайтів;
- підвищення безпеки та надійності веб-сайтів.

Апробація результатів дослідження. Матеріали даної роботи мають певну практичну значущість та були впроваджені у практичну діяльність у

закладі “Університет Короля Данила”, що підтверджується листом від “15” лютого 2024 року.

Структура магістерської роботи. Магістерська робота складається зі вступу, трьох розділів, що розділені на 2 підрозділи в першому розділі, 4 підрозділи в другому розділі, та 6 підрозділів в третьому розділі, висновків, списку використаних джерел на 40 позицій. Загальний обсяг магістерської роботи становить 97 сторінок.

РОЗДІЛ 1. ВИЗНАЧЕННЯ ПРОБЛЕМАТИКИ МОДУЛЬНОСТІ ТА ПЕРЕВИКОРИСТАННЯ КОДУ, ОБҐРУНТУВАННЯ АКТУАЛЬНОСТІ ДОСЛІДЖЕННЯ В СУЧАСНОМУ ВЕБ-ДИЗАЙНІ

1.1 Забезпечення відновлюваності та супроводження

Веб-дизайн є динамічною та швидкозмінюваною галуззю, і проблеми модульності та перевикористання коду є актуальними для розвитку стійких, ефективних та легкозмінюваних веб-проектів. Відсутність модульності може призвести до труднощів у виправленні помилок, оновленні та розширенні проекту, що ускладнює супровід веб-додатків протягом тривалого періоду.

В сучасному інформаційному ландшафті, де швидкість розвитку та технологічні інновації зазнають стрімкого зростання, важливо підкреслити роль забезпечення відновлюваності та супроводження в контексті веб-додатків. Відсутність модульності в кодовій базі може призвести до виникнення серйозних труднощів у процесі виправлення помилок, вдосконалення та розширення проектів, що ставить під загрозу ефективність та стійкість веб-додатків протягом тривалого періоду.

Відсутність модульності ускладнює відновлення та супровід системи на кожному етапі її життєвого циклу. Першим із викликів є управління помилками. Без чіткого визначення та локалізації проблеми у великому коді розробникам стає складно вчасно та ефективно виправляти помилки, що може призвести до збоїв у роботі системи та негативно вплинути на користувачів.

Другим аспектом є оновлення. Відсутність чітко визначених модулів ускладнює процес впровадження нових функцій, виправлення помилок та важливих оновлень. Розробники змушені опрацьовувати великі блоки коду, що призводить до ризику неспростовних змін та непередбачуваних наслідків.

Однією з ключових стратегій для забезпечення відновлюваності та супроводження є впровадження модульності в архітектуру веб-додатків.

Модульний код дозволяє розробникам відокремлювати функціональність, що полегшує виявлення та усунення помилок. Кожен модуль може визначати свої функціональні обов'язки та має відокремлені точки зв'язку з іншими частинами.

Додатково, строге дотримання стандартів програмування та використання патернів проектування сприяють створенню стабільного та простого для супроводження коду. Чітка документація, автоматизоване тестування та відкритий обмін знаннями серед команди також є важливими елементами для ефективного супроводження.

Забезпечення відновлюваності та супроводження веб-додатків у сфері інформаційних технологій - це завдання, яке вимагає вдосконалення стратегій та використання передових практик в розробці. Модульність, правильна організація коду та систематичний підхід до супроводження є ключовими факторами для забезпечення довгострокової ефективності та стійкості веб-додатків в індустрії інформаційних технологій.

Комплексність проектів: зі зростанням складності веб-додатків з'являється виклик управління великим обсягом коду та його розміщенням в організований і легко змінюваний спосіб.

У сучасному веб-дизайні та розробці веб-додатків виникає надзвичайно складна проблема: як ефективно управляти розмаїттям та обсягом коду, щоб забезпечити легкість супроводу та швидкість розвитку. Із зростанням складності веб-додатків стає очевидним, що необхідно вдосконалювати стратегії управління кодовою базою для забезпечення організованості та легкої зміни коду розробки.

Ключовою проблемою є управління великим обсягом коду. Накопичення кодових ресурсів призводить до втрати структури та може ускладнити якісне супроводження. Відповідно до цього, важливо виробити стратегії, які сприяють модульності, перевикористанню коду та швидкій зміні.

Ефективне розділення проекту на модулі є ключовим елементом для забезпечення комплексності проектів. Модульність дозволяє відокремлювати функціонально різні частини проекту, що полегшує розробку, тестування та

зміни. Вона дозволяє розробникам концентруватися на окремих частинах системи, сприяючи при цьому високій якості та швидкості розробки.

Ефективна організація коду включає в себе використання шаблонів проектування, стандартів та чіткого документування. Це сприяє створенню стабільних та легко змінюваних систем. Забезпечення легкості змін важливо для адаптації до нових вимог та виправлення помилок без впливу на код.

Для управління комплексністю проектів у веб-дизайні потрібно системний та стратегічний підхід до модульності та організації коду. Ефективні стратегії модульності та виважене використання стандартів є ключовими складовими для забезпечення легкості супроводу та швидкості розвитку в індустрії інформаційних технологій.

Перевикористання коду: відсутність чіткої структури та перевикористання коду може призвести до надмірного дублювання та втрати часу на створення однотипних елементів.

У світі інформаційних технологій, де час – це ключовий ресурс, важливо визначити стратегії, що дозволяють ефективно використовувати код та уникнути надмірного дублювання. Відсутність чіткої структури та перевикористання коду може призвести до втрати часу та ресурсів на створення однотипних елементів, що ускладнює розвиток та супровід веб-додатків.

Однією з ключових проблем є надмірне дублювання коду, яке може виникнути при відсутності чіткої структури. Дублювання коду призводить до великої кількості однотипних елементів у кодовій базі, що робить систему важкою для розуміння та утримання. Розробники можуть витрачати значний час на пошук та виправлення проблем у кількох місцях, замість того, щоб ефективно працювати над усім проектом.

Другим аспектом є втрата часу на створення однотипних елементів. У випадку відсутності механізмів перевикористання коду розробники можуть постійно виготовляти аналогічні компоненти, не використовуючи наявних ресурсів та можливостей для оптимізації роботи.

Стратегією для подолання проблеми перевикористання коду є впровадження механізмів повторного використання та оптимізації кодової бази. Створення бібліотек, модульних компонентів та шаблонів дозволяє розробникам використовувати наявний код для створення нових функцій та елементів, уникнути дублювання та значно зменшити час розробки.

Використання патернів проектування, які дозволяють створювати гнучкі та масштабовані рішення, готові до подальшого використання. Регулярне проведення код-рев'ю та використання інструментів для виявлення дублювання також допомагає уникнути непотрібного дублювання та забезпечити консистентність кодової бази.

Перевикористання коду в індустрії інформаційних технологій вимагає впровадження ефективних стратегій та практик в розробці. Визначення чіткої структури та використання механізмів перевикористання дозволяє підтримувати високу продуктивність та ефективність у роботі над проектами, що допомагає уникнути зайвих витрат часу та ресурсів.

Розподілена розробка: з участю розподіленої розробки важливо мати чітко визначені модулі, які можна розробляти та тестувати незалежно.

У сучасній індустрії інформаційних технологій розподілена розробка стала невід'ємною частиною багатьох проектів. З участю розподіленої розробки важливо мати чітко визначені модулі, які можна розробляти та тестувати незалежно. Це стає ключовим чинником для успішного впровадження та ефективної співпраці розробних команд.

При розподіленій розробці є незручність управління та спілкування між розробницькими групами. Без чітко визначених модулів може виникнути конфлікт при одночасній роботі над однією частиною проекту різними командами. Це може призвести до затримок у виконанні завдань, невідповідності стандартам та погіршення загального контролю над проектом.

Також ускладнення тестування та відлагодження коду. Без чітко визначених модулів розробники можуть стикатися з труднощами в ідентифікації та ізоляції проблем, що робить процес тестування більш часо та

ресурсозатратним. Нездійснення незалежного тестування може призвести до появи помилок та недоліків, які важко виправити в подальшому.

Стратегією в управлінні розподіленою розробкою є визначення чітко визначених модулів та їхніх функціональних обов'язків. Кожен модуль повинен бути самодостатнім та має визначені інтерфейси для взаємодії з іншими модулями. Це дозволяє розробникам працювати над конкретними частинами проекту, не втрачаючи час на вирішення конфліктів та непорозумінь.

Важливим елементом є використання засобів для систематизації та автоматизації процесу розробки, таких як системи контролю версій та інструменти для автоматичного тестування. Це сприяє підтримці чіткої структури та взаємодії між розробницькими групами.

Розподілена розробка потребує від команд високого рівня організації та визначеності у структурі проекту. Чітко визначені модулі, що розробляються та тестуються незалежно, є важливим елементом для досягнення успіху в умовах розподіленої розробки. Це дозволяє забезпечити ефективність, контроль та швидкість у роботі над проектом.

1.2 Обґрунтування актуальності дослідження та розробки нових підходів для поліпшення структури веб-проектів

Веб-проекти є важливою частиною сучасного світу. Вони використовуються для різних цілей, таких як надання інформації, проведення бізнесу, спілкування та розваги. Для того, щоб веб-проекти були ефективними та успішними, вони повинні мати чітку та добре продуману структуру.

Існує ряд проблем, які можуть виникнути внаслідок поганої структури веб-проекту. Наприклад, такий проект може бути складним для навігації, важким для розуміння та використання, а також може бути не оптимальним для пошукової оптимізаційної системи.

Дослідження та розробка нових підходів для поліпшення структури веб-проектів є актуальним напрямком, оскільки вони можуть допомогти

вирішити ці проблеми та зробити веб-проекти більш ефективними, успішними, та зручними у використанні.

Деякі з конкретних переваг, які можуть бути отримані внаслідок поліпшення структури веб-проектів:

– **Покращена навігація:** користувачі зможуть швидше та легше знаходити необхідну інформацію та виконувати необхідні завдання.

– **Покращене розуміння:** користувачі зможуть краще зрозуміти структуру веб-проекту та його призначення.

– **Покращена ефективність:** кеб-проекти будуть працювати швидше та споживатимуть менше ресурсів.

– **Покращена пошукова оптимізація:** веб-проекти будуть краще відображатися в результатах пошуку, що призведе до збільшення кількості відвідувачів на сайті.

Дослідження та розробка нових підходів для поліпшення структури веб-проектів можуть здійснюватися в різних напрямках. Наприклад, можна розробляти нові методи аналізу структури веб-проектів, нові алгоритми для генерації структур веб-проектів, а також нові інструменти та технології для підтримки розробки веб-проектів з хорошою структурою.

З розвитком сучасних технологій та фреймворків, які спрямовані на покращення модульності та перевикористання коду, дослідження цих аспектів стає більш актуальним. Актуальність дослідження та розробки нових підходів для поліпшення структури веб-проектів, спрямованих на покращення модульності та перевикористання коду, може бути обґрунтована кількома ключовими представленими аргументами.

Зростання складності веб-додатків: із зростанням потреб користувачів у функціональних можливостях веб-додатків, вони стають все більш складними. Це вимагає ефективних стратегій управління кодом та його структурою для забезпечення читабельності, підтримуваності та розширюваності продукту.

Щоб ефективно управляти такими складними веб-додатками, потрібно застосовувати стратегії управління кодом та його структурою, які забезпечують:

– читабельність коду, структура проекту повинна бути такою, щоб код був легко зрозумілий іншими розробниками. Це допомагає уникнути заплутаності та помилок при роботі над проектом;

– підтримуваність коду, код повинен бути легко підтримуваним, тобто зміни та виправлення помилок повинні виконуватися з мінімальними зусиллями. Це досягається шляхом чіткої структури та документації коду;

– розширюваність проекту, структура проекту повинна бути гнучкою і дозволяти легко додавати новий функціонал без необхідності переписувати великі частини коду. Це забезпечує швидкий розвиток проекту та його відповідність змінним потребам користувачів.

Масштабованість проекту, код повинен бути організований таким чином, щоб проект можна було легко масштабувати при необхідності. Це важливо для забезпечення ефективної роботи проекту навіть при збільшенні обсягу даних або потоку користувачів.

Такі стратегії управління кодом та його структурою дозволяють забезпечити ефективне управління складними веб-додатками та забезпечити їхню успішну роботу у змінному середовищі.

Потреба в швидкому розвитку та змінах: сучасні веб-проекти часто вимагають швидкого впровадження нового функціоналу та змін. Це вимагає гнучкості та легкості у розширенні існуючого коду, що може досягатися за допомогою модульності та перевикористання коду.

Потреба в швидкому розвитку та змінах у сучасних веб-проектах дійсно відіграє ключову роль у вимогах до управління кодом та його структурою. Деякі аспекти, що демонструють актуальність модульності та перевикористання коду в цьому контексті.

Швидке впровадження нового функціоналу, стратегії модульності дозволяють розробникам створювати нові функції у вигляді незалежних модулів, які можна додавати до проекту без значних змін у вже існуючому коді. Це дозволяє швидко впроваджувати новий функціонал та реагувати на зміни вимог користувачів чи ринку.

Легка модифікація існуючого коду, модульний підхід до розробки дозволяє змінювати чи розширювати функціональність веб-додатку, не затрагуючи інші його частини. Це забезпечує гнучкість у розвитку проекту та зменшує ризик введення помилок.

Ефективне використання ресурсів, перевикористання коду дозволяє використовувати існуючий, перевірений та випробуваний код у різних частинах проекту. Це економить час розробки, зменшує кількість потрібних ресурсів та сприяє швидкому впровадженню змін.

Зменшення ризику помилок, розробка модульної архітектури дозволяє зменшити ризик введення помилок у процесі змін та розширень. Кожен модуль може бути тестований окремо, що полегшує виявлення та виправлення помилок в структурі системи.

Збереження якості коду, модульний підхід допомагає зберігати високу якість коду, оскільки він стимулює використання кращих практик розробки, таких як розділення відповідальностей та створення чітких інтерфейсів.

Модульність та перевикористання коду відіграють важливу роль у забезпеченні швидкого та ефективного розвитку сучасних веб-проектів, що постійно піддаються змінам та вимагають високої гнучкості.

Підвищення продуктивності розробників: застосування ефективних підходів до модульності та перевикористання коду допомагає розробникам швидше та ефективніше створювати новий функціонал та вирішувати проблеми в продукті.

Підвищення продуктивності розробників є ключовим аспектом в успішному веб-розробці, і використання ефективних підходів до модульності та перевикористання коду грає важливу роль у досягненні цієї мети. Деякі методи демонструють, як модульність та перевикористання коду сприяють підвищенню продуктивності розробників:

Швидше створення нового функціоналу, за допомогою модульного підходу, розробники можуть легко створювати новий функціонал,

використовуючи існуючі модулі та компоненти. Це дозволяє уникнути повторної розробки та швидко впроваджувати нові функції.

Ефективне використання часу, перевикористання коду дозволяє розробникам уникати написання однотипного коду заново. Вони можуть використовувати готові рішення та компоненти, що значно збільшує швидкість розробки та зменшує час, потрібний для вирішення завдань.

Зменшення кількості помилок, модульна архітектура сприяє розділенню функціональності на невеликі, незалежні компоненти, що полегшує тестування та виявлення помилок. Це дозволяє зменшити час, який витрачається на пошук та виправлення помилок.

Спрощення робочого процесу, модульність сприяє створенню чіткого розподілу робочих завдань між розробниками. Кожен може працювати над своїм невеликим модулем, не заважаючи іншим членам команди, що сприяє ефективнішому розподілу робочого часу та збільшує загальну продуктивність команди під час розробки.

Підвищення якості коду, використання готових модулів та компонентів, які перевірені та випробувані, дозволяє створювати високоякісний код. Це сприяє збільшенню швидкості розробки та зменшенню кількості помилок.

Застосування модульності та перевикористання коду допомагає розробникам прискорити процес розробки, підвищити якість коду та забезпечити ефективне використання робочого часу.

Оптимізація ресурсів. Перевикористання коду дозволяє оптимізувати використання ресурсів, оскільки забезпечує використання існуючого коду для різних частин проекту, замість повторного його написання.

Оптимізація ресурсів є одним із ключових аспектів у сучасній веб-розробці, і перевикористання коду грає важливу роль у досягненні цієї мети. Ось деякі способи, які демонструють, як перевикористання коду сприяє оптимізації ресурсів:

– економія часу, замість того, щоб писати новий код для кожної нової функції або компонента, розробники можуть перевикористовувати існуючий

код. Це дозволяє ефективно використовувати час розробки та зменшує необхідний обсяг роботи;

– ефективне використання людських ресурсів, розробники можуть спрямовувати свої зусилля на вирішення більш складних проблем або створення нового функціоналу, замість того, щоб витрачати час на написання стандартного коду, який вже був реалізований;

– зменшення обсягу коду, перевикористання коду дозволяє уникнути надмірної дублювання коду в проекті. Це сприяє зменшенню обсягу коду, що зроблює проект більш зрозумілим, легким у підтримці та модифікації;

– ефективне використання комп'ютерних ресурсів, оскільки перевикористання коду допомагає уникнути надмірного споживання пам'яті та обчислювальних ресурсів, це сприяє покращенню продуктивності веб-додатка та забезпеченню більш ефективного використання серверних ресурсів;

– зниження витрат на розробку та підтримку, оскільки перевикористання коду допомагає зменшити час, необхідний для розробки нового функціоналу та виправлення помилок, це також дозволяє знизити витрати на розробку та підтримку веб-додатків;

– перевикористання коду є важливим інструментом для оптимізації ресурсів у веб-розробці, який дозволяє ефективно використовувати доступні людські та комп'ютерні ресурси для досягнення більш ефективного та ефективного розробки веб-додатків.

Покращення якості продукту: ефективна модульність та перевикористання коду допомагають уникнути багатьох типових помилок, що може підвищити якість та надійність веб-додатків.

Так, покращення якості продукту є одним із ключових вигод від використання ефективної модульності та перевикористання коду у веб-розробці. Ось кілька способів, які демонструють, як ці підходи сприяють покращенню якості та надійності веб-додатків:

– зменшення кількості помилок, розділення функціоналу на невеликі, незалежні модулі дозволяє зменшити ризик введення помилок у коді. Кожен

модуль може бути тестований окремо, що допомагає виявляти та виправляти помилки на ранніх етапах розробки;

– підвищення стабільності, використання перевірених та випробуваних модулів забезпечує більшу стабільність веб-додатку. Ці модулі вже пройшли перевірку та тестування, що зменшує ризик виникнення непередбачених проблем у роботі додатку;

– забезпечення чистого коду, модульний підхід стимулює розробників писати більш чистий та організований код. Це сприяє підвищенню читабельності, зрозумілості та підтримуваності кодової бази;

– швидка реакція на зміни, будучи розділеними на невеликі модулі, веб-додатки стають більш гнучкими та легко змінюються. Це дозволяє швидко реагувати на зміни в вимогах користувачів або ринку та швидко впроваджувати необхідні зміни.

Висновки до розділу 1

У першому розділі було проведено аналіз модульності, перевикористання та розподіленої розробки, проведено обґрунтування актуальності розробки таких методів та оптимізація ресурсів, впровадження нового функціоналу, підвищення якості продукту. Проведено аналіз конкуретних переваг таких як покращена навігація, розуміння, ефективність та пошукова оптимізація. Модульність та перевикористання коду є невід'ємними принципами веб-дизайну, які сприяють створенню стійких, ефективних, легкозмінюваних та якісних веб-проектів. Дослідження та розробка нових підходів у цій сфері є актуальним напрямком, який може значно спростити розробку веб сайтів.

РОЗДІЛ 2. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ЗАБЕЗПЕЧЕННЯ МОДУЛЬНОСТІ ТА ПЕРЕВИКОРИСТАННЯ КОДУ У ВЕБ-ДИЗАЙНІ

2.1 Аналіз існуючих методів

Модульність та перевикористання коду є важливими принципами веб-дизайну, які дозволяють покращити структуру, надійність та масштабованість веб-проектів. Модулярність дозволяє розбивати веб-проект на окремі модулі, які можна розробляти та підтримувати незалежно один від одного. Перевикористання коду дозволяє повторно використовувати вже написаний код у різних частинах веб-проекту, що економить час і зусилля розробників.

Існує ряд різних методів забезпечення модульності та перевикористання коду у веб-дизайні. У цій статті ми розглянемо основні з них, а також порівняємо різні підходи до організації коду та архітектури веб-проектів.

Методи забезпечення модульності та перевикористання коду:

– **Файлова організація:** найпростіший спосіб забезпечення модульності та перевикористання коду у веб-дизайні - це використовувати файлову організацію. При цьому кожний модуль розміщується у окремому файлі. Цей метод є простим і зрозумілим, але він має ряд обмежень. Наприклад, він не дозволяє ефективно управляти залежностями між модулями;

– **Структура каталогів:** більш ефективним способом забезпечення модульності та перевикористання коду є використання структури каталогів. При цьому модулі групуються за певним принципом, наприклад, за функціональністю або областю застосування. Цей метод дозволяє краще управляти залежностями між модулями, а також полегшує навігацію по коду;

– **Організація коду за компонентами:** ще одним ефективним методом забезпечення модульності та перевикористання коду є організація коду за компонентами. Компоненти - це самостійні модулі, які мають власний

інтерфейс і реалізацію. Компоненти можна повторно використовувати у різних частинах веб-проекту;

– **Організація коду за шаблонами:** шаблони - це ще один спосіб забезпечення модульності та перевикористання коду. Шаблони - зразки коду, які можна використовувати для створення нових модулів. Шаблони дозволяють стандартизувати код і полегшити його розробку та підтримку.

Порівняння підходів до організації коду та архітектури, кожен з методів забезпечення модульності та перевикористання коду має свої переваги та недоліки. Файлова організація є найпростішою, але вона має обмеження щодо управління залежностями між модулями. Структура каталогів є більш ефективною, але вона вимагає більш детального планування. Організація коду за компонентами є ще більш ефективною, але вона вимагає більшого досвіду від розробників. Організація коду за шаблонами є найпростішою, але вона може призвести до повторення коду.

При виборі підходу до організації коду та архітектури веб-проекту необхідно враховувати такі фактори:

– **розмір та складність веб-проекту:** для великих і складних веб-проектів необхідні більш ефективні методи організації коду, такі як організація коду за компонентами або шаблонами;

– **складність бізнес-логіки:** для веб-проектів з складною бізнес-логікою необхідно використовувати методи організації коду, які дозволяють легко управляти залежностями між модулями;

– **досвід розробників:** для веб-проектів, які розробляються командою з досвідченими розробниками, можна використовувати більш складні методи організації коду в розробці.

Модулярність та перевикористання коду є важливими принципами веб-дизайну, які дозволяють покращити структуру, надійність та масштабованість веб-проектів. Існує ряд різних методів забезпечення модульності та перевикористання коду, кожен з яких має свої переваги та недоліки. При виборі підходу до організації коду та архітектури веб-проекту

необхідно враховувати такі фактори, розмір та складність веб-проекту, складність бізнес-логіки та досвід розробників.

2.2 Вивчення сучасних тенденцій та викликів у веб-розробці

Веб-розробка галузь, яка постійно розвивається. З'являються нові технології, змінюються вимоги до веб-проектів, а також зростає конкуренція між розробниками. Щоб бути успішним веб-розробником, важливо знати про сучасні тенденції у цій галузі.

З основних тенденцій у веб-розробці, які спостерігаються в останні роки:

– **Перехід на мобільні пристрої:** все більше людей використовують мобільні пристрої для доступу до Інтернету. Це означає, що веб-проекти повинні бути адаптивними або розробленими спеціально для мобільних пристроїв чи планшетів.

– **Зростання популярності штучного інтелекту та машинного навчання:** штучний інтелект і машинне навчання все частіше використовуються у веб-розробці. Ці технології дозволяють створювати більш інтуїтивно зрозумілі та персоналізовані веб-додатки.

– **Розвиток додатків реального часу:** веб-додатки реального часу дозволяють користувачам взаємодіяти з веб-сайтами та веб-додатками в режимі реального часу. Ці додатки стають все більш популярними, оскільки вони забезпечують більш плавний і інтерактивний досвід користувача.

– **Зростання популярності хмарних технологій:** хмарні технології дозволяють веб-розробникам створювати інфраструктуру для веб-проектів без необхідності інвестувати в власне обладнання та програмне забезпечення. Це робить веб-розробку більш доступною та економічно ефективною.

– **Виклики у веб-розробці:** разом із новими тенденціями у веб-розробці виникають і нові виклики. Основні виклики, з якими стикаються веб-розробники на сьогодні.

– **Зростаюча складність веб-проектів**: веб-проекти стають все більш складними, оскільки вони включають в себе все більше функцій і можливостей. Це може ускладнити розробку, тестування та підтримку таких проектів.

– **Потреба в спеціалізованих знаннях:** веб-розробка стає все більш спеціалізованою. Розробники повинні мати глибокі знання в різних областях, таких як програмування, дизайн, безпека та маркетинг.

– **Зростаюча конкуренція:** веб-розробка - це конкурентна галузь. Для того, щоб бути успішним веб-розробником, важливо постійно розвиватися та вдосконалювати свої навички.

2.3 Методи забезпечення модульності у веб-дизайні

2.3.1 Модульна розробка

Цей підхід до проектування програмного забезпечення, який передбачає розбиття програми на невеликі модулі або компоненти. Кожний модуль відповідає за конкретну функціональність. Модульна розробка має ряд переваг, зокрема:

1. покращує читабельність коду та модулі є більш компактними та зрозумілими, ніж великі, неструктуровані програми. Це полегшує розробникам розуміти, як працює код, і знаходити помилки;
2. полегшує роботу над окремими частинами проекту, модулі можна розробляти та тестувати незалежно один від одного. Це дозволяє розробникам працювати над різними модулями одночасно, що може скоротити час розробки;
3. покращує масштабованість та гнучкість, можна легко додавати модулі, змінювати або видаляти, не впливаючи на інші частини програми. Це дозволяє адаптувати програму до нових вимог.

Проект розділяється на невеликі модулі або компоненти. Першим кроком у модульній розробці є розбиття проекту на невеликі модулі. Модулі повинні

бути досить великими, щоб виконувати конкретну задачу, але не такими великими, щоб їх було важко зрозуміти та підтримувати.

Існує кілька різних способів розбити проект на модулі. Одним із поширених підходів є використання структури "головний модуль - підмодулі". Головний модуль відповідає за загальну логіку програми, а підмодулі відповідають за конкретні функції.

Іншим підходом є використання структури "компоненти". Компоненти є незалежними модулями, які можуть бути повторно використані.

Кожний модуль повинен відповідати за конкретну функціональність. Це допоможе забезпечити читабельність коду та полегшити роботу над окремими частинами проекту чи продукту.

Наприклад, проект веб-сайту можна розбити на такі модулі: модуль навігації відповідає за відображення навігаційної панелі на веб-сайті, модуль рендерингу відповідає за відображення веб-сторінок на екрані, модуль бази даних відповідає за доступ до бази даних.

Покращує читабельність коду та полегшує роботу над окремими частинами проекту. Модульна розробка покращує читабельність коду та полегшує роботу над окремими частинами проекту. Це відбувається тому, що модулі є більш компактними та зрозумілими, ніж великі, неструктуровані програмні продукти чи сайти.

Наприклад, якщо модульний код веб-сайту розділений на такі модулі, як модуль навігації, модуль рендерингу та модуль бази даних, то розробники можуть легко зрозуміти, як працює кожний модуль. Це полегшує їхню роботу над окремими частинами проекту.

Застосування модульної розробки у веб-дизайні. Модульна розробка широко використовується у веб-дизайні. Вона є ефективним способом створення веб-сайтів, які є читабельними, маштабованими та гнучкими.

Кілька прикладів того, як модульна розробка може бути використана у веб-дизайні:

– **для створення веб-сайтів з багатомовною підтримкою:** модулі можна використовувати для розбиття веб-сайту на окремі частини для кожної мови. Це полегшує додавання підтримки для нових мов;

– **для створення веб-сайтів з адаптивним дизайном:** модулі можна використовувати для створення веб-сайту, який автоматично адаптується до різних розмірів екранів. Це полегшує використання веб-сайту на різних комп'ютерах, планшетах, смартфонах;

– **для створення веб-сайтів з підтримкою різних пристроїв:** модулі можна використовувати для створення веб-сайту, який може бути доступний через різні пристрої, такі як комп'ютери, планшети та смартфони. Це полегшує використання веб-сайту на різних пристроях.

Модульна розробка є ефективним способом створення веб-сайтів, які є читабельними, масштабованими та гнучкими. Вона має ряд переваг, зокрема:

1. покращує читабельність коду;
2. полегшує роботу над окремими частинами проекту;
3. покращує масштабованість та гнучкість.

2.3.2 Використання фреймворків та бібліотек для забезпечення модульності у веб-дизайні

Фреймворки та бібліотеки є потужними інструментами, які можуть допомогти веб-розробникам поліпшити модульність своїх проектів.

Фреймворки являють собою набори інструментів та бібліотек, які надають стандартну структуру та шаблони для розробки веб-сайтів. Вони можуть допомогти веб-розробникам поліпшити читабельність, масштабованість та гнучкість своїх проектів.

Бібліотеки - мають у собі набори функцій та класів, які можна використовувати для виконання конкретних завдань. Вони можуть допомогти веб-розробникам економити час і зусилля на розробку коду.

Використання фреймворків для забезпечення модульності. Деякі фреймворки, такі як React, Angular, або Vue.js, сприяють компонентній архітектурі. Компоненти, незалежні модулі коду, які можуть бути повторно використані в різних проектах (рис. 2.1).

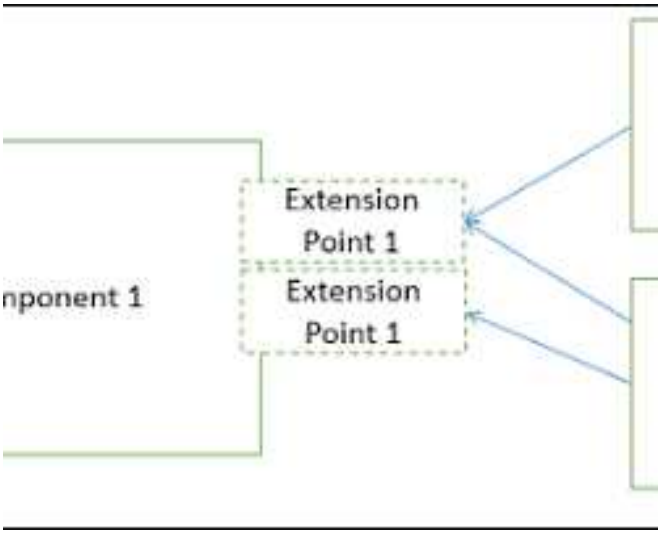


Рисунок 2.1 – Компонентна архітектура

Компоненти можуть бути використані для розбиття веб-сайту на окремі частини, кожна з яких відповідає за конкретну функціональність. Це може поліпшити читабельність коду та полегшити роботу над окремими частинами.

Переваги використання фреймворків для забезпечення модульності:

– **покращена читабельність коду:** компоненти є більш компактними та зрозумілими, ніж великий, неструктурований код. Це полегшує розробникам розуміти, як працює код, і знаходити помилки;

– **полегшена робота над окремими частинами проекту:** компоненти можна розробляти та тестувати незалежно один від одного. Це дозволяє розробникам працювати над різними компонентами одночасно, що може скоротити час розробки проекту;

– **покращена масштабованість та гнучкість:** компоненти можна легко додавати, змінювати або видаляти, не впливаючи на інші частини проекту. Це дозволяє адаптувати проект до нових вимог.

Використання бібліотек для забезпечення модульності. Бібліотеки можуть бути використані для забезпечення модульності веб-проектів різними способами. Наприклад, бібліотеки можуть бути використані для:

– **розподілу коду на окремі модулі:** можуть містити код, який відповідає за конкретні функції. Це може поліпшити читабельність коду та полегшити роботу над окремими частинами проекту;

– **перевикористання коду:** бібліотеки можуть мати код, який вже був розроблений і протестований. Це може економити час і зусилля розробників;

– **забезпечення стандартизації коду:** бібліотеки можуть містити код, написаний у стандартному стилі. Це може полегшити розуміння та підтримку.

Переваги використання бібліотек для забезпечення модульності:

1. **покращена читабельність коду:** бібліотеки код яких відповідає за окремі функції, тим самим покращити читабельність коду та спростити роботу над унікальними частинами проекту;
2. **економія часу та зусиль:** бібліотеки можуть в собі мати код, який вже був створений і протестований. Це економить час розробників;
3. **забезпечення стандартизації коду:** у бібліотеках є код, який написаний у стандартному стилі, що спрощує розуміння та підтримку коду.

Фреймворки та бібліотеки є потужними інструментами, які можуть допомогти веб-розробникам поліпшити модульність своїх проектів. Вони можуть допомогти веб-розробникам поліпшити читабельність, масштабованість та гнучкість своїх проектів.

2.3.3 Компонентна архітектура

Компонентна архітектура - підхід до проектування програмного забезпечення, який передбачає організацію проекту навколо невеликих незалежних компонентів. Компоненти відповідають за конкретну функціональність або виджет.

Організація проекту навколо невеликих незалежних компонентів. Компонентна архітектура передбачає розбиття проекту на невеликі компоненти, які є незалежними та можуть бути повторно використані в різних проектах. Це має ряд переваг, зокрема, покращує читабельність коду тим самим компоненти є більш компактними та зрозумілими, ніж великий, неструктурований код. Полегшує розробникам розуміти, як працює код, і знаходити помилки. Спрощує роботу над окремими частинами проекту, та компоненти можна розробляти та тестувати незалежно один від одного. Це дозволяє розробникам працювати над різними компонентами одночасно, що може скоротити час розробки проекту. Покращує масштабованість та гнучкість, компоненти можна легко додавати, змінювати або видаляти, не впливаючи на інші частини проекту. Це дозволяє адаптувати проект до нових вимог.

Зручно для використання та тестування окремих частин проекту. Компонентна архітектура робить проект більш зручним для використання та тестування окремих частин. Це відбувається тому, що компоненти є самодостатніми і не залежать від інших компонентів.

Наприклад, якщо компонентна архітектура використовується для розробки веб-сайту, то кожна веб-сторінка може бути представлена як компонент. Це полегшує розробникам тестувати окремі веб-сторінки, не тестуючи весь веб-сайт.

Застосування компонентної архітектури у веб-дизайні. Компонентна архітектура широко використовується у веб-дизайні. Вона є ефективним способом створення веб-сайтів, які є читабельними, масштабованими та гнучкими.

Приклади того, як компонентна архітектура може бути використана у веб-дизайні:

- для створення веб-сайтів з адаптивним дизайном: компоненти можна використовувати для створення веб-сайту, який автоматично адаптується до різних розмірів екранів. Це полегшує використання веб-сайту на комп'ютерах, телефонах, планшетах, пристроях;

- для створення веб-сайтів з підтримкою різних пристроїв: компоненти можна використовувати для створення веб-сайту, який може бути доступний через різні пристрої, такі як комп'ютери, планшети та смартфони. Це полегшує використання веб-сайту на різних пристроях;

- для створення веб-сайтів з підтримкою різних мов: компоненти можна використовувати для створення веб-сайту, який може бути доступний на різних мовах. Це полегшує адаптацію веб-сайту до нових мов.

Компонентна архітектура є ефективним способом створення веб-сайтів, які є читабельними, масштабованими та гнучкими.

Вона має ряд переваг, зокрема:

1. покращує читабельність коду;
2. полегшує роботу над окремими частинами проекту;
3. покращує масштабованість та гнучкість;
4. зручно для використання та тестування окремих частин проекту.

2.3.4 Патерни проектування

Патерни проектування є шаблонами, які описують типові проблеми та їх вирішення у програмуванні. Вони можуть бути використані для покращення якості коду, його читабельності та масштабованості.

Використання різних патернів проектування. У веб-дизайні існує безліч різних патернів проектування, які можуть бути використані для вирішення різних завдань. Деякі з найпоширеніших патернів проектування, які використовуються у веб-дизайні, включають:

- **Singleton:** цей патерн забезпечує, щоб у програмі існувала лише одна копія певного класу. Він може бути використаний для забезпечення глобального доступу до ресурсу, такого як база даних або файл;
- **Observer:** цей патерн дозволяє об'єктам повідомляти один одного про зміни. Він може бути використаний для реалізації таких функцій, як сповіщення користувачів про зміни на веб-сайті;

– **Dependency Injection:** цей патерн дозволяє об'єктам отримувати доступ до інших об'єктів, які вони потребують, через ін’єкцію. Він може бути використаний для зменшення залежностей між об'єктами та полегшення тестування продукту чи системи.

Патерни можуть забезпечити стандартизацію та зменшення залежностей між різними частинами коду. Патерни проектування можуть забезпечити стандартизацію та зменшення залежностей між різними частинами коду. Це може полегшити читання, розуміння та підтримку коду.

Стандартизація. Патерни проектування можуть допомогти стандартизувати код, що полегшує його читання та розуміння. Наприклад, якщо всі веб-сайти використовують патерн Singleton для доступу до бази даних, то розробникам буде легше зрозуміти, як працює код.

Зменшення залежностей. Патерни проектування можуть допомогти зменшити залежності між різними частинами коду. Це може полегшити тестування та обслуговування коду. Наприклад, якщо об'єкти отримують доступ до інших об'єктів через ін'єкцію, то ці об'єкти не залежать один від одного. Це означає, що їх можна тестувати незалежно один від одного.

Застосування патернів проектування у веб-дизайні. Патерни проектування можуть бути використані для вирішення широкого спектру завдань у веб-дизайні.

Singleton може бути використаний для забезпечення глобального доступу до ресурсів, таких як база даних або файл. Наприклад, Singleton може бути використаний для забезпечення глобального доступу до констант або глобальних змінних. Observer може бути використаний для реалізації таких функцій, як сповіщення користувачів про зміни на веб-сайті. Наприклад, Observer може бути використаний для сповіщення користувачів про нові повідомлення або події. Dependency Injection може бути використаний для зменшення залежностей між об'єктами. Наприклад, Dependency Injection може бути використаний для забезпечення того, щоб об'єкти отримували доступ до інших об'єктів через ін'єкцію.

Патерни проектування є потужним інструментом, який може бути використаний для покращення якості коду, його читабельності та масштабованості. Вони можуть бути використані для вирішення широкого спектру завдань у веб-дизайні.

2.3.5 Модульні тести

Модульні тести тип тестування програмного забезпечення, який фокусується на тестуванні окремих модулів коду. Модуль - це найменша частина коду, яка може бути протестована окремо.

Написання модульних тестів для кожного компонента або модуля. Модульні тести повинні бути написані для кожного компонента або модуля коду. Це допоможе забезпечити, щоб кожен компонент працював належним чином і не містив помилок.

Допомагає виявляти та виправляти помилки на ранніх етапах розробки. Модульні тести можуть допомогти виявити та виправити помилки на ранніх етапах розробки. Це може заощадити час та гроші, оскільки помилки, виявлені на ранніх етапах, легше виправити, ніж помилки, виявлені пізніше.

Принципи модульного тестування. Модульні тести повинні дотримуватися таких принципів:

- незалежність: модульні тести повинні бути незалежними один від одного. Це означає, що один модульний тест не повинен залежати від результатів іншого модульного тесту;

- повна перевірка: модульні тести повинні повністю перевірити функціональність модуля. Це означає, що модульний тест повинен перевірити всі можливі сценарії використання модуля;

- автоматизація: модульні тести повинні бути автоматизованими. Це означає, що модульні тести повинні виконуватися автоматично, без необхідності втручання людини.

Переваги модульного тестування. Модульні тестування має ряд переваг, зокрема, покращує якість коду. Модульні тести допомагають виявити та виправити помилки на ранніх етапах розробки. Це призводить до підвищення якості коду. Збільшує надійність коду, модульні тести допомагають забезпечити, що код працює належним чином. Може призвести до підвищення надійності коду. Сприяє повторному використанню коду, модульні тести допомагають зробити код більш повторюваним.

Застосування модульного тестування у веб-дизайні. Модульні тестування широко використовується у веб-дизайні. Вона є ефективним способом забезпечення якості та надійності веб-сайтів.

Для тестування компонентів веб-сторінок, модульні тести можуть бути використані для тестування окремих компонентів веб-сторінок, таких як кнопки, текстові поля та спливаючі вікна.

Для тестування функціональності веб-сайту, модульні тести можуть бути використані для тестування функціональності веб-сайту, таких як реєстрація користувачів, вхід у систему та кошик покупок.

Для тестування безпеки веб-сайту, модульні тести можуть бути використані для тестування безпеки веб-сайту, таких як перевірка наявності уразливостей у коді.

Модульні тестування є потужним інструментом, який може бути використаний для забезпечення якості та надійності веб-сайтів. Вони можуть бути використані для тестування окремих компонентів веб-сторінок, функціональності веб-сайту та безпеки веб-сайту.

2.3.6 Використання препроцесорів та збірників

Препроцесори та збірники потужні інструменти, які можуть допомогти веб-розробникам покращити якість та ефективність їхнього коду.

Використання препроцесорів CSS (наприклад, Sass або LESS) для підтримки модульності у стилі

Препроцесори CSS - інструменти, які дозволяють розширити можливості CSS. Вони можуть використовуватися для додавання нових функцій, таких як модульність, імпорт та експорт.

Модульність CSS спосіб організації CSS-коду в окремі модулі. Це може зробити код більш читабельним, зрозумілим та повторюваним.

Препроцесори CSS, такі як Sass та LESS, підтримують модульність CSS за допомогою таких функцій, як:

1. **створення модулів:** препроцесори CSS дозволяють створювати модулі CSS, які можна імпортувати в інші модулі;
2. **імпорт та експорт:** препроцесори CSS дозволяють імпортувати та експортувати CSS-код з інших модулів;
3. **перейменування:** препроцесори CSS дозволяють перейменовувати CSS-правила та властивості.

Переваги використання препроцесорів CSS для підтримки модульності. Використання препроцесорів CSS для підтримки модульності має ряд переваг, зокрема, покращена читабельність коду, модульний CSS-код більш читабельний та зрозумілий, ніж немодульне CSS-код. Повторюваність коду, Модульний CSS-код можна повторювати в різних проектах, зменшення розміру файлів. Модульний CSS-код можна компонувати в один файл, що зменшує розмір файлів. Використання збірників, таких як Webpack або Parcel, для управління модульністю в JavaScript

Збірники - це інструменти, які збирають в один файл усі необхідні модулі JavaScript. Вони можуть використовуватися для управління модульністю JavaScript та забезпечення того, щоб код працював належним чином.

Модульність JavaScript спосіб організації JavaScript-коду в окремі модулі. Це може зробити код більш читабельним, зрозумілим та повторюваним.

Збірники, такі як Webpack або Parcel, підтримують модульність JavaScript за допомогою таких функцій:

1. **включення модулів:** збірники дозволяють включати модулі JavaScript з інших файлів.

2. **залежності:** збірники дозволяють визначати залежності між модулями JavaScript.

3. **агрегація модулів:** збірники дозволяють згрупувати модулі JavaScript в один файл.

Переваги використання збірників для управління модульністю. Використання збірників для управління модульністю має ряд переваг. Таких як, покращена читабельність коду, модульний JavaScript-код більш читабельний та зрозумілий, ніж немодульне JavaScript-код. Повторюваність коду, модульний JavaScript-код можна повторювати в різних проектах. Зменшення розміру файлів, модульний JavaScript-код можна компонувати в один файл, що зменшує розмір файлів системи.

Застосування препроцесорів та збірників у веб-дизайні. Препроцесори та збірники широко використовуються у веб-дизайні. Вони є ефективним способом покращення якості та ефективності веб-сайтів.

Як препроцесори та збірники можуть бути використані у веб-дизайні, наприклад для створення модульних стилів. Препроцесори CSS можуть бути використані для створення модульних стилів, які можуть бути повторно використані в різних частинах веб-сайту. Також для управління модульністю JavaScript. Збірники можуть бути використані для управління модульністю JavaScript, що може полегшити тестування та обслуговування коду.

Препроцесори та збірники потужні інструменти, які можуть допомогти веб-розробникам покращити якість та ефективність їхнього коду. Вони можуть бути використані для створення модульних стилів та управління модульністю JavaScript.

2.3.7 Чистий код

Чистий код - код, який є читабельним, зрозумілим і підтримуваним. Та має ряд переваг:

- покращена читабельність: чистий код легко читати та розуміти. Це може полегшити роботу веб-розробникам, які працюють над проектом;

- повторюваність коду: чистий код можна повторно використовувати в різних проектах. Це може заощадити час та зусилля веб-розробників;

- зручність тестування: чистий код легше тестувати. Це може допомогти забезпечити якість коду.

Дотримання принципів чистого коду, таких як SOLID, що сприяє зрозумілості та модульності коду:

– SOLID: абревіатура для п'яти принципів проектування програмного забезпечення, які сприяють створенню чистого коду;

– Single Responsibility Principle (SRP): кожна клас або метод повинен мати лише одну відповідальність;

– Open-Closed Principle (OCP): клас або модуль повинен бути відкритим для розширення, але закритим для модифікації;

– Liskov Substitution Principle (LSP): підкласи повинні бути взаємозамінними зі своїми суперкласами;

– Interface Segregation Principle (ISP): не слід змушувати клієнтів залежати від інтерфейсів, які вони не використовують;

– Dependency Inversion Principle (DIP): високі рівні повинні залежати від низьких рівнів, а не навпаки.

Дотримання цих принципів може допомогти веб-розробникам створити код, який є зрозумілим, модульним і підтримуваним.

Зменшення залежностей між компонентами та використання інтерфейсів. Залежність являє собою зв'язок між двома компонентами, який дозволяє одному компоненту використовувати функції іншого компонента. Залежності можуть ускладнити код, зробити його менш читабельним і утруднити тестування окремих частин або усієї системи.

Зменшення залежностей між компонентами може допомогти покращити чистоту коду. Одним із способів зменшити залежності є використання інтерфейсів. Інтерфейс контракт, який визначає, які функції повинен надавати

компонент. Компоненти можуть взаємодіяти один з одним, лише якщо вони реалізують один і той самий інтерфейс.

Використання інтерфейсів може допомогти зменшити залежності між компонентами, оскільки компоненти будуть взаємодіяти лише через інтерфейс, а не через конкретний клас.

Приклади застосування принципів чистого коду у веб-дизайні:

- створення читабельних та зрозумілих веб-сторінок: веб-розробники можуть використовувати принципи чистого коду для створення веб-сторінок, які легко читати та розуміти користувачам. Наприклад, веб-розробники можуть використовувати принцип SRP для того, щоб розділити код веб-сторінки на окремі частини, кожна з яких відповідає за конкретну функцію;

- створення модульних веб-сайтів: веб-розробники можуть використовувати принципи чистого коду для створення веб-сайтів, які є модульними. Це може полегшити обслуговування та розширення веб-сайту. Наприклад, веб-розробники можуть використовувати принцип SOLID для того, щоб створити компоненти веб-сайту, які мають лише одну відповідальність і які можуть бути повторно використані в різних частинах веб-сайту;

- створення тестованого веб-коду: веб-розробники можуть використовувати принципи чистого коду для створення веб-коду, який легко тестувати. Це може допомогти забезпечити якість веб-коду. Наприклад, веб-розробники можуть використовувати принцип ISP для того, щоб розбити веб-код на невеликі, незалежні частини, які можна легко протестувати.

Чистий код важливий аспект веб-дизайну. Він може допомогти веб-розробникам створювати веб-сайти, які є читабельними, зрозумілими та підтримуваними різними пристроями.

2.3.8 Системи контролю версій

Системи контролю версій (СКВ) являє інструменти, які дозволяють відстежувати зміни в коді. Вони є важливим інструментом для веб-розробників,

оскільки вони можуть допомогти зберегти історію змін у коді, відновити попередні версії коду та співпрацювати з іншими розробниками.

Використання систем контролю версій, таких як Git, для ефективного управління змінами СКВ можна використовувати для ефективного управління змінами в коді за допомогою таких функцій:

– **версії:** зберігають історію змін у коді. Це дозволяє веб-розробникам відстежувати зміни в коді та відновити попередні версії коду;

– **філії:** дозволяють веб-розробникам створювати гілки коду. Це дозволяє веб-розробникам експериментувати з новими функціями або виправленнями без впливу на основну версію коду;

– **об'єднання:** дозволяють веб-розробникам об'єднувати зміни з різних гілок коду. Це дозволяє веб-розробникам співпрацювати з іншими розробниками та об'єднувати їхні зміни в один код.

Використання систем контролю версій, таких як Git, для розгалуження коду. Розгалуження коду, процес створення нової гілки коду з основної версії коду. Це дозволяє веб-розробникам експериментувати з новими функціями або виправленнями без впливу на основну версію коду.

Розгалуження коду можна використовувати для таких цілей:

– **експериментування:** розгалуження коду можна використовувати для експериментування з новими функціями або виправленнями без впливу на основну версію коду. Якщо експеримент не вдається, веб-розробник може просто видалити гілку коду;

– **розробка нових функцій:** розгалуження коду можна використовувати для розробки нових функцій. Після того, як функція буде розроблена та протестована, її можна об'єднати з основною версією коду;

– **виправлення помилок:** код можна використовувати для виправлення помилок. Після того, як помилка буде виправлена, її можна об'єднати з основною версією коду.

Приклади використання СКВ у веб-дизайні. Для збереження історії змін, системи контролю версій можна використовувати для збереження історії

змін у коді веб-сайту. Це може бути корисно для відстеження змін у коді, відновлення попередніх версій коду та виявлення помилок у коді.

Щоб співпрацювати з іншими розробниками, системи контролю версій можна використовувати для співпраці з іншими розробниками над веб-сайтом. Це може бути корисно для об'єднання змін від різних розробників та забезпечення того, що всі розробники працюють над однією і тією ж версією.

Щоб розробити нові функцій, СКВ можна використовувати для розробки нових функцій для веб-сайту. Це може бути корисно для експериментування з новими функціями без впливу на основну версію коду.

СКВ є важливим інструментом для веб-розробників. Вони можуть допомогти зберегти історію змін у коді, відновити попередні версії коду та співпрацювати з іншими розробниками.

Ці підходи можуть використовуватися окремо або в комбінації, залежно від конкретних потреб та вимог проекту.

2.4 Методи забезпечення перевикористання коду у веб-дизайні

2.4.1 Шаблони

Шаблони мають набори CSS-правил, які можна використовувати для створення веб-сторінок. Шаблони можуть бути використані для забезпечення перевикористання коду, оскільки вони дозволяють веб-розробникам повторно використовувати один і той самий код для створення різних веб-сторінок.

Шаблони працюють, визначаючи структуру та стиль веб-сторінки. Вони можуть містити такі елементи:

– **блоки контенту:** області веб-сторінки, які можуть бути заповнені різним контентом цього блоку;

– **стилі:** визначаються як CSS-правила, які визначають, як елементи на веб-сторінці будуть виглядати.

Переваги використання шаблонів:

1. **заощадження часу та зусиль:** шаблони можуть допомогти веб-розробникам заощадити час та зусилля, оскільки вони не повинні писати код для створення кожної веб-сторінки з нуля;

2. **покращена якість коду:** шаблони можуть допомогти покращити якість коду, оскільки код перевірявся та тестувався в попередньому проекті;

3. **зручність обслуговування:** шаблони можуть полегшити обслуговування коду, оскільки зміни потрібно вносити лише в один екземпляр.

Приклади використання шаблонів. Шаблони можна використовувати для створення веб-сайтів із багато сторінок, оскільки вони дозволяють веб-розробникам використовувати один і той самий код для створення різних сторінок веб-сайту.

Шаблони можна використовувати для створення веб-сайтів із динамічним контентом, оскільки вони дозволяють веб-розробникам заповнювати блоки контенту різним контентом, наприклад, новинами, статтями або продуктами.

Шаблони можна використовувати для створення веб-сайтів із адаптивним дизайном, оскільки вони дозволяють веб-розробникам використовувати один і той самий код для створення веб-сторінок, які добре виглядають на пристроях.

Загальні рекомендації щодо використання шаблонів. Використовуйте шаблони для визначення структури та стилю веб-сторінки. Шаблони слід використовувати для визначення структури та стилю веб-сторінки, а не для визначення конкретного контенту, який буде відображатися на веб-сторінці. Створюйте шаблони, які можна легко налаштувати. Шаблони слід створювати таким чином, щоб їх можна було легко налаштувати для різних проектів. Використайте шаблони для створення симпліфікації. Шаблони можна використовувати для створення симпліфікації, яка може бути використана для створення різних веб-сторінок.

Шаблони потужний інструмент, який може допомогти веб-розробникам заощадити час та зусилля, а також покращити якість коду.

2.4.2 Бібліотеки UI-компонентів

Бібліотеки UI-компонентів мають набори готових компонентів, які можна використовувати для створення веб-сторінок. Бібліотеки UI-компонентів можуть бути використані для забезпечення перевикористання коду, оскільки вони дозволяють веб-розробникам повторно використовувати один і той самий код для створення різних веб-сторінок.

Бібліотеки UI-компонентів працюють, надаючи веб-розробникам готові компоненти, які можна використовувати для створення веб-сторінок. Компоненти містять такі елементи:

1. **HTML-код:** HTML-код визначає структуру компонента;
2. **CSS-код:** CSS-код визначає стиль компонента;
3. **JavaScript-код:** JavaScript-код визначає поведінку компонента.

Переваги використання бібліотек UI-компонентів. Бібліотеки UI-компонентів мають ряд переваг. Заощадження часу та зусиль, бібліотеки UI-компонентів можуть допомогти веб-розробникам заощадити час та зусилля, оскільки вони не повинні писати код для створення кожного компонента з нуля. Покращена якість коду, бібліотеки UI-компонентів можуть допомогти покращити якість коду, оскільки код перевірявся та тестувався в попередньому проекті. Зручність обслуговування, бібліотеки UI-компонентів можуть полегшити обслуговування коду, оскільки зміни потрібно вносити лише в один екземпляр коду.

Приклади використання бібліотек UI-компонентів:

– бібліотеки UI-компонентів можна використовувати для створення веб-сайтів із багато сторінок, оскільки вони дозволяють веб-розробникам використовувати один і той самий код для створення різних сторінок веб-сайту;

– UI-компоненти можна використовувати для створення веб-сайтів із динамічним контентом, оскільки вони дозволяють веб-розробникам налаштовувати компоненти для відображення різних типів контенту;

– для створення веб-сайтів із адаптивним дизайном: Бібліотеки UI-компонентів можна використовувати для створення веб-сайтів із адаптивним дизайном, оскільки вони дозволяють веб-розробникам використовувати один і той самий код для створення веб-сторінок, які добре виглядають на пристроях.

Загальні рекомендації щодо використання бібліотек UI-компонентів. Для забезпечення ефективного використання бібліотек UI-компонентів веб-розробникам слід дотримуватися таких рекомендацій:

– **використовуйте бібліотеки UI-компонентів, які відповідають вашим потребам:** перед використанням бібліотеки UI-компонентів важливо переконатися, що вона відповідає вашим потребам;

– **вивчіть бібліотеку UI-компонентів:** перед використанням бібліотеки UI-компонентів важливо вивчити її документацію, щоб зрозуміти, як її використовувати;

– **налаштуйте бібліотеки UI-компонентів:** багато бібліотек UI-компонентів дозволяють налаштувати компоненти для задоволення ваших конкретних потреб на сайті.

Також важливо при використанні бібліотек UI-компонентів слід також враховувати:

1. **стиль:** бібліотеки UI-компонентів часто мають унікальний стиль. Перед використанням бібліотеки UI-компонентів важливо переконатися, що її стиль відповідає стилю вашого веб-сайту.;
2. **підтримка:** важливо переконатися, що бібліотека UI-компонентів підтримується її розробниками.

2.4.3 Модульність CSS

Модульність CSS, метод організації CSS-коду таким чином, щоб його можна було легко повторно використовувати. Модульний CSS може допомогти веб-розробникам заощадити час та зусилля, а також покращити якість коду.

Основні принципи модульного CSS включають:

– **розбиття CSS-коду на модулі:** CSS-код слід розбивати на окремі модулі, які відповідають за конкретну функцію;

– **використання селекторів:** селектори можна використовувати для визначення, які елементи CSS-коду будуть застосовні;

– **використання властивостей CSS:** властивості CSS можна використовувати для визначення стилю елементів.

Модульний CSS має ряд переваг, зокрема:

1. **заощадження часу та зусиль:** модульний CSS може допомогти веб-розробникам заощадити час та зусилля, оскільки вони не повинні писати код для створення кожного модуля з нуля;

2. **покращена читабельність та зрозумілість коду:** модульний CSS може допомогти покращити читабельність та зрозумілість коду, оскільки він організований у невеликі, легко зрозумілі модулі;

3. **покращена масштабованість:** модульний CSS може допомогти покращити масштабованість коду, оскільки його можна легко розширити.

4. **покращена підтримка:** модульний CSS може допомогти покращити підтримку коду, тому його можна легко тестувати та підтримувати.

Існує кілька різних типів модулів CSS, які можна використовувати для організації CSS-коду. Модулі стилів містять CSS-правила, які визначають стиль елементів. Модулі компонентів містять CSS-код, який визначає структуру та стиль компонента. Модулі властивостей містять CSS-правила, які визначають стиль елементів за допомогою однієї або кількох властивостей CSS.

Приклади використання модульного CSS:

– **для створення веб-сайтів із багато сторінок:** модульний CSS можна використовувати для створення веб-сайтів із багато сторінок, оскільки він дозволяє веб-розробникам використовувати один і той самий CSS-код для створення різних сторінок веб-сайту;

– **для створення веб-сайтів із динамічним контентом:** модульний CSS можна використовувати для створення веб-сайтів із динамічним контентом,

оскільки він дозволяє веб-розробникам налаштувати CSS-код для відображення різних типів контенту;

– **для створення веб-сайтів із адаптивним дизайном:** модульний CSS можна використовувати для створення веб-сайтів із адаптивним дизайном, оскільки він дозволяє веб-розробникам використовувати один і той самий CSS-код для створення веб-сторінок, які добре виглядають на різних пристроях.

Загальні рекомендації щодо використання модульного CSS. Для забезпечення ефективного використання модульного CSS веб-розробникам слід дотримуватися таких рекомендацій:

– **розбивайте CSS-код на невеликі, легко зрозумілі модулі:** модулі не повинні бути занадто великими або складними, інакше вони стануть важко читати та підтримувати;

– **використовуйте зрозумілі назви для модулів:** назви модулів повинні чітко відображати їх призначення;

– **дотримуйтесь стандартів CSS:** дотримання стандартів CSS допоможе вам зробити ваш CSS-код більш читабельним та зрозумілим для інших розробників.

Модульний CSS потужний інструмент, який може допомогти веб-розробникам заощадити час та зусилля, а також покращити якість коду. При правильному використанні модульний CSS може зробити ваш CSS-код більш читабельним, зрозумілим, масштабованим та підтримуваним.

2.4.4 Блокова структура HTML/CSS

Блокова структура HTML/CSS, метод організації веб-сторінок таким чином, щоб їх можна було легко повторно використовувати. Блокова структура заснована на використанні блочних елементів HTML, які займають всю ширину контейнера, в якому вони знаходяться.

Основні принципи блочної структури:

1. використання блочних елементів HTML: Блочні елементи HTML, такі як div, ul, ol, і li, займають всю ширину контейнера;

2. використання CSS для стилізації блочних елементів: CSS можна використовувати для стилізації блочних елементів, таких як розмір, розташування та колір.

Переваги блочної структури. Покращена читабельність та зрозумілість коду, блокова структура робить код більш читабельним та зрозумілим, оскільки елементи розташовані в логічній послідовності. Покращена масштабованість, блокову структуру можна легко масштабувати, додаючи або видаляючи блоки. Покращена підтримка, блокову структуру легко підтримувати, оскільки зміни можна вносити в один блок, не впливаючи на інші блоки.

Приклади використання блочної структури, у веб-дизайні:

- **для створення веб-сайтів із багато сторінок:** блокову структуру можна використовувати для створення веб-сайтів із багато сторінок, оскільки вона дозволяє веб-розробникам використовувати один і той самий CSS-код для створення різних сторінок веб-сайту;
- **для створення веб-сайтів із динамічним контентом:** блокову структуру можна використовувати для створення веб-сайтів із динамічним контентом, оскільки вона дозволяє веб-розробникам налаштувати CSS-код для відображення різних типів контенту;
- **для створення веб-сайтів із адаптивним дизайном:** блокову структуру можна використовувати для створення веб-сайтів із адаптивним дизайном, оскільки вона дозволяє веб-розробникам використовувати один і той самий CSS-код для створення веб-сторінок, які добре виглядають на пристроях.

Загальні рекомендації щодо використання блочної структури. Для забезпечення ефективного використання блочної структури веб-розробникам слід дотримуватися таких рекомендацій:

- **використовуйте блочні елементи HTML для створення логічної структури веб-сторінки:** блокові елементи HTML можна використовувати для створення логічної структури веб-сторінки, розділяючи її на різні секції;

– **використовуйте CSS для стилізації блочних елементів:** CSS можна використовувати для стилізації блочних елементів, таких як розмір, розташування та колір;

– **слідуйте стандартам HTML/CSS:** дотримання стандартів HTML/CSS допоможе вам зробити ваш код більш читабельним та зрозумілим для інших розробників.

Є деякі рекомендації які можуть допомогти веб-розробникам використовувати її більш ефективно. Використовуйте контейнери для групування пов'язаних елементів. Контейнери можуть використовуватися для групування пов'язаних елементів, що може полегшити їх стилізацію та управління. Використовуйте ID-атрибути для ідентифікації елементів. ID-атрибути можна використовувати для ідентифікації елементів, що може бути корисно для створення динамічних веб-сторінок. Використовуйте клас-атрибути для групування елементів. Класи-атрибути можна використовувати для групування елементів, що може бути корисно для стилізації груп елементів.

Використання блочної структури HTML/CSS може допомогти веб-розробникам створити веб-сторінки, які є більш читабельними, зрозумілими, масштабованими та підтримуваними.

Блокова структура HTML/CSS - це потужний інструмент, який може допомогти веб-розробникам заощадити час та зусилля, а також покращити якість коду. При правильному використанні блочна структура може зробити ваш код більш читабельним, зрозумілим, масштабованим та підтримуваним.

2.4.5 Модульне тестування

Модульне тестування метод тестування програмного забезпечення, який фокусується на тестуванні окремих модулів коду. Модульне тестування може допомогти веб-розробникам забезпечити перевикористання коду, оскільки воно дозволяє тестувати код у відокремленому середовищі.

Модульне тестування працює шляхом створення тестів для кожного модуля коду. Тести повинні перевіряти, чи працює модуль коду належним чином. Модульні тести зазвичай пишуть самі веб-розробники.

Переваги модульного тестування:

– **покращена якість коду:** модульне тестування може допомогти покращити якість коду, оскільки воно допомагає виявляти помилки на ранніх етапах розробки;

– **полегшена підтримка коду:** модульне тестування може полегшити підтримку коду, оскільки воно дозволяє тестувати код без необхідності запуску повного додатка;

– **покращена продуктивність:** Модульне тестування може покращити продуктивність, оскільки воно дозволяє веб-розробникам швидко і легко тестувати код.

Приклади використання модульного тестування у веб-дизайні. Для тестування функцій, модульне тестування можна використовувати для тестування функцій, таких як функції додавання або видалення елементів з веб-сторінки. Для тестування компонентів, модульне тестування можна використовувати для тестування компонентів, таких як кнопки або меню. Для тестування API, модульне тестування можна використовувати для тестування API, таких як API для доступу до бази даних.

Для забезпечення ефективного використання модульного тестування веб-розробникам. Почніть тестувати код якомога раніше, щоб виявити помилки на ранніх етапах розробки. Пишіть тести для кожного модуля коду, щоб переконатися, що він працює належним чином. Використовуйте автоматизоване модульне тестування, щоб полегшити тестування коду.

Для використання модульного тестування використовуйте такі методи:

– **використовуйте фреймворки модульного тестування**: фреймворки модульного тестування можуть допомогти веб-розробникам полегшити написання та запуск модульних тестів;

– **інтегруйте модульне тестування в процес розробки:** інтегруйте модульне тестування в процес розробки, щоб забезпечити те, що код тестується на регулярній основі;

– **тримайте модульні тести в актуальному стані:** тримайте модульні тести в актуальному стані, щоб вони продовжували перевіряти код належним чином.

Використання модульного тестування може допомогти веб-розробникам створити веб-додатки, які є більш надійними та якісними.

Модульне тестування потужний інструмент, який може допомогти веб-розробникам забезпечити перевикористання коду та покращити якість коду. При правильному використанні модульне тестування може допомогти веб-розробникам створити веб-додатки, які є більш надійними та якісними.

2.4.6 Рефакторинг і оптимізація

Рефакторинг і оптимізація, два процеси, які можуть допомогти веб-розробникам забезпечити перевикористання коду. Рефакторинг - це процес зміни коду без зміни його функціональності. Оптимізація процес поліпшення ефективності коду системи.

Рефакторинг може допомогти веб-розробникам забезпечити перевикористання коду, оскільки він дозволяє їм переробляти код таким чином, щоб його можна було легше зрозуміти і використовувати.

Рефакторинг може включати такі завдання:

– **розбиття коду на більш дрібні модулі:** тим самим зробить код легшим та допоможе зрозуміти і підтримувати;

– **найменування змінних і функцій:** покращить читабельність коду;

– **використання абстракцій:** зробить код більш універсальним і придатним для повторного використання.

Оптимізація може допомогти веб-розробникам забезпечити перевикористання коду, оскільки вона може полегшити використання коду в

різних контекстах. Оптимізація може включати, використання ефективних алгоритмів що може покращити продуктивність коду. Також використання ефективних структур даних, може покращити продуктивність коду. Використання ефективних методів кодування може покращити продуктивність коду.

Приклади використання рефакторингу і оптимізації:

– **рефакторингу компонентів:** можна використовувати для рефакторингу компонентів, таких як кнопки або меню, щоб їх можна було легше використовувати в різних контекстах;

– **оптимізації коду CSS:** можна використовувати для оптимізації коду CSS, щоб він завантажувався швидше;

– **оптимізації коду JavaScript:** можна використовувати для оптимізації коду JavaScript, щоб він виконувався швидше.

Загальні рекомендації щодо використання рефакторингу і оптимізації:

1. почніть рефакторинг якомога раніше, щоб уникнути необхідності вносити зміни в код пізніше;
2. використовуйте автоматизований рефакторинг, щоб полегшити рефакторинг коду;
3. почніть оптимізацію якомога раніше, щоб уникнути необхідності вносити зміни в код пізніше;
4. використовуйте профілювання, щоб визначити, де код можна оптимізувати.

Існують такі шляхи використовувати рефакторинг і оптимізацію більш ефективно. Використовуйте фреймворки рефакторингу, фреймворки рефакторингу можуть допомогти веб-розробникам полегшити рефакторинг коду. Використовуйте фреймворки оптимізації, фреймворки оптимізації можуть допомогти веб-розробникам полегшити оптимізацію коду. Інтегруйте рефакторинг і оптимізацію в процес розробки, щоб забезпечити те, що код рефакторингується і оптимізується на регулярній основі.

Використання рефакторингу і оптимізації може допомогти веб-розробникам створити веб-додатки, які є більш ефективними та придатними для повторного використання.

Рефакторинг і оптимізація має два важливі процеси, які можуть допомогти веб-розробникам забезпечити перевикористання коду. При правильному використанні рефакторинг і оптимізація можуть допомогти веб-розробникам створити веб-додатки, які є більш ефективними та придатними для повторного використання.

2.4.7 Блоки та керування залежностями

Блоки та керування залежностями, два методи, які можуть допомогти веб-розробникам забезпечити перевикористання коду. Блоки мають невеликі, автономні частини коду, які можна повторно використовувати в різних контекстах. Керування залежностями - це процес управління взаємодією між різними модулями коду.

Блоки можуть допомогти веб-розробникам забезпечити перевикористання коду, оскільки вони дозволяють створювати автономні частини коду, які можна легко інтегрувати в інші модулі.

Блоки можна створювати для різних цілей, наприклад:

- **компоненти:** блоки можна використовувати для створення компонентів, таких як кнопки або меню;
- **функції:** блоки можна використовувати для створення функцій, таких як функції додавання або видалення елементів з веб-сторінки;
- **API:** блоки можна використовувати для створення API, таких як API для доступу до бази даних.

Керування залежностями може допомогти веб-розробникам забезпечити перевикористання коду, оскільки воно дозволяє визначати, які модулі коду залежать від інших модулів.

Керування залежностями може включати такі завдання:

– **використання пакетних менеджерів**: пакетні менеджери можуть допомогти веб-розробникам встановлювати та керувати модулями коду;

– **використання систем керування залежностями:** системи керування залежностями можуть допомогти веб-розробникам визначати та управляти залежностями між модулями коду.

Приклади використання блоків і керування залежностями у веб-дизайні:

1. **для створення компонентів:** блоки можна використовувати для створення компонентів, таких як кнопки або меню. Керування залежностями можна використовувати для визначення, компонент від інших компонентів;

2. **для створення веб-додатків:** блоки можна використовувати для створення веб-додатків, таких як інтернет-магазини або блоки. Керування залежностями можна використовувати для визначення, які модулі коду необхідні для створення веб-додатка.

Для забезпечення ефективного використання блогів і керування залежностями веб-розробникам слід дотримуватися таких правил. Використовуйте стандартизовані імена для модулів, стандартизовані імена можуть допомогти веб-розробникам легко визначати, які модулі залежать від інших модулів. Почніть з невеликих модулів, щоб їх було легше зрозуміти і підтримувати. Використовуйте автоматизовані інструменти, щоб допомогти вам керувати залежностями. Щоб використовувати блоки і керування залежностями більш ефективно, можна використати фреймворки блогів можуть допомогти веб-розробникам полегшити створення і використання блогів. Використовуйте системи керування залежностями, системи керування залежностями можуть допомогти веб-розробникам полегшити визначення і управління залежностями між модулями коду. Інтегруйте блоки і керування залежностями в процес розробки, щоб забезпечити те, що блоки і керування залежностями використовуються на регулярній основі.

Використання блогів і керування залежностями може допомогти веб-розробникам створити веб-додатки, які є більш ефективними та придатними для повторного використання

При правильному використанні блоки і керування залежностями можуть допомогти веб-розробникам створити веб-додатки, які є більш ефективними та придатними для повторного використання.

2.4.8 Використання дизайн-систем

Дизайн-системи мають набори компонентів, стилів та інших ресурсів, які можна використовувати для створення веб-додатків. Дизайн-системи можуть допомогти веб-розробникам забезпечити перевикористання коду, оскільки вони пропонують широкий спектр готових до використання компонентів, які можна легко інтегрувати в веб-додатки.

Дизайн-системи засновані на таких принципах:

1. **компоненти:** дизайн-системи складаються з компонентів, які представляють окремі елементи веб-сторінок, такі як кнопки, меню та форми;
2. **стилі:** дизайн-системи включають стилі, які визначать зовнішній вигляд компонентів;
3. **ресурси:** дизайн-системи можуть також включати інші ресурси, такі як шаблони, код і документи.

Переваги використання дизайн-систем. Покращена продуктивність дизайн-системи може допомогти веб-розробникам підвищити продуктивність, оскільки вони пропонують широкий спектр готових до використання компонентів. Дизайн-системи можуть допомогти веб-розробникам підвищити якість коду, оскільки вони використовують стандартизовані компоненти та стилі. Масштабованість дизайн-системи може допомогти веб-розробникам масштабувати веб-додатки, оскільки вони пропонують набори компонентів, які можна легко налаштувати.

Приклади використання дизайн-систем:

– **для створення веб-сайтів:** дизайн-системи можна використовувати для створення веб-сайтів, таких як корпоративні сайти або інтернет-магазини;

– **для створення веб-додатків:** дизайн-системи можна використовувати для створення веб-додатків, таких як інтернет-банкінг або соціальні мережі;

– **для створення адаптивних веб-додатків:** дизайн-системи можна використовувати для створення адаптивних веб-додатків, які добре виглядають на різних пристроях.

Вибирайте дизайн-систему, яка відповідає вашим потребам тому при виборі дизайн-системи враховуйте свої конкретні потреби та вимоги. Навчіться використовувати дизайн-систему та перед використанням дизайн-системи обов'язково вивчіть її документацію. Інтегруйте дизайн-систему в свій процес розробки, щоб забезпечити те, що дизайн-система використовується на регулярній основі.

Для використання дизайн-системи більш ефективно є такі методи:

– **використовуйте інструменти для створення дизайн-систем:** існує ряд інструментів, які можуть допомогти веб-розробникам створити дизайн-системи;

– **створюйте власні дизайн-системи:** якщо ви не знайшли дизайн-системи, яка відповідає вашим потребам, ви можете створити власну.

Використання дизайн-систем може допомогти веб-розробникам створити веб-додатки, які відповідають потребам їх користувачів.

Дизайн-системи є потужним інструментом, який може допомогти веб-розробникам забезпечити перевикористання коду та створити веб-додатки, які є більш ефективними та придатними для повторного використання.

Висновки до розділу 2

У другому розділі був проведений великий аналіз методів наприклад методи забезпечення модульності та перевикористання коду, файлова

організація, структура каталогів, організація коду з компонентами, порівняння підходів до організації архітектури. Проведено аналіз впливу штучного інтелекту та машинного навчання на розробку додатків в реальному часі. Розглянуто компонентну архітектуру та патерни проектування та тестування окремих модулів, проаналізовано модульність в коді.

РОЗДІЛ 3 РОЗРОБКА НОВИХ ПІДХОДІВ З ДОТРИМАННЯМ МОДУЛЬНОСТІ ТА СУПРОВОДУ НА САЙТІ

3.1 Використання інтерактивного веб-редактора WebFlow як інноваційного інструменту для спрощення розробки та візуальної взаємодії з елементами веб-додатків

Webflow платформа, яка поєднує в собі візуальний редактор, CMS та хостинг, що робить її потужним інструментом для веб-розробки.

Webflow дозволяє створювати візуальний інтерфейс та використовує drag-and-drop інтерфейс, що робить його доступним для людей без глибоких знань програмування.

Webflow спрощує розробку та пропонує бібліотеку готових компонентів та шаблонів, що економить час та зусилля розробників. Дозволяє створювати візуальну взаємодію дизайнерам, та дозволяє розробникам візуалізувати та тестувати інтерфейс користувача на ходу.

Інноваційні можливості сервісу пропонують ряд інноваційних функцій, таких як:

- анімації та переходи;
- Responsive design;
- SEO-оптимізація;
- CMS система.

У безкоштовній версії, Webflow пропонує план, який дозволяє вам створити один веб-сайт.

Webflow можна використовувати:

1. дизайнерами дозволяє створювати красиві та функціональні веб-сайти без написання коду;
2. розробниками може використовуватися для швидкого створення прототипів та MVP;

3. підприємцями сервіс може використовуватися для створення веб-сайтів для своїх бізнесів різного масштабу;

4. використання WebFlow як інструменту для веб-розробки є порівняно новим явищем, тому його дослідження все ще тривають.

Порівняння WebFlow з іншими веб-редакторами. У порівнянні з такими редакторами як Wix який пропонує простий у використанні візуальний редактор для створення веб-сайтів, Squarespace, що пропонує стильні шаблони та інструменти для створення веб-сайтів, та WordPress з відкритим кодом, який має широкий спектр можливостей для створення веб-сайтів, WebFlow має декілька вагомих переваг.

Функціональність WebFlow пропонує широкий спектр функцій, анімації та переходи, responsive design, SEO-оптимізація, CMS системи.

На відміну від Wix що пропонує менший набір функцій, ніж WebFlow, але він простіший у використанні. Squarespace який має стильні шаблони та інструменти для створення веб-сайтів, але він менш гнучкий, ніж WebFlow. Та WordPress який надає широкий спектр можливостей, але він може бути складним для початківців.

Що ж казати за ціну то WebFlow пропонує безкоштовний план, а також платні плани, які починаються від $12 на місяць. Wix безкоштовний план, а також платні плани, які починаються від $4.50 на місяць. Squarespace надає безкоштовний пробний період, та платні плани, які починаються від $12 на місяць. WordPress пропонує безкоштовний хостинг, але вам потрібно буде придбати доменне ім'я та хостинг.

Простота використання WebFlow, має криву навчання, але він простіший у використанні, ніж WordPress. Wix - найпростіший у використанні веб-редактор, але він пропонує менший набір функцій. Squarespace - простий у використанні веб-редактор, який пропонує стильні шаблони. WordPress може бути складним для початківців, але він пропонує широкий спектр можливостей.

Підтримка WebFlow пропонує онлайн-документацію, чат-підтримку та форум. Wix пропонує онлайн-документацію, чат-підтримку та телефонну

підтримку. Squarespace пропонує онлайн-документацію, чат-підтримку та email-підтримку. WordPress має велику спільноту користувачів, які можуть допомогти вам з будь-якими проблемами.

Підсумовуючи переваги та недоліки, продемонструємо WebFlow та Wix на діаграмах на рисунку 3.1.

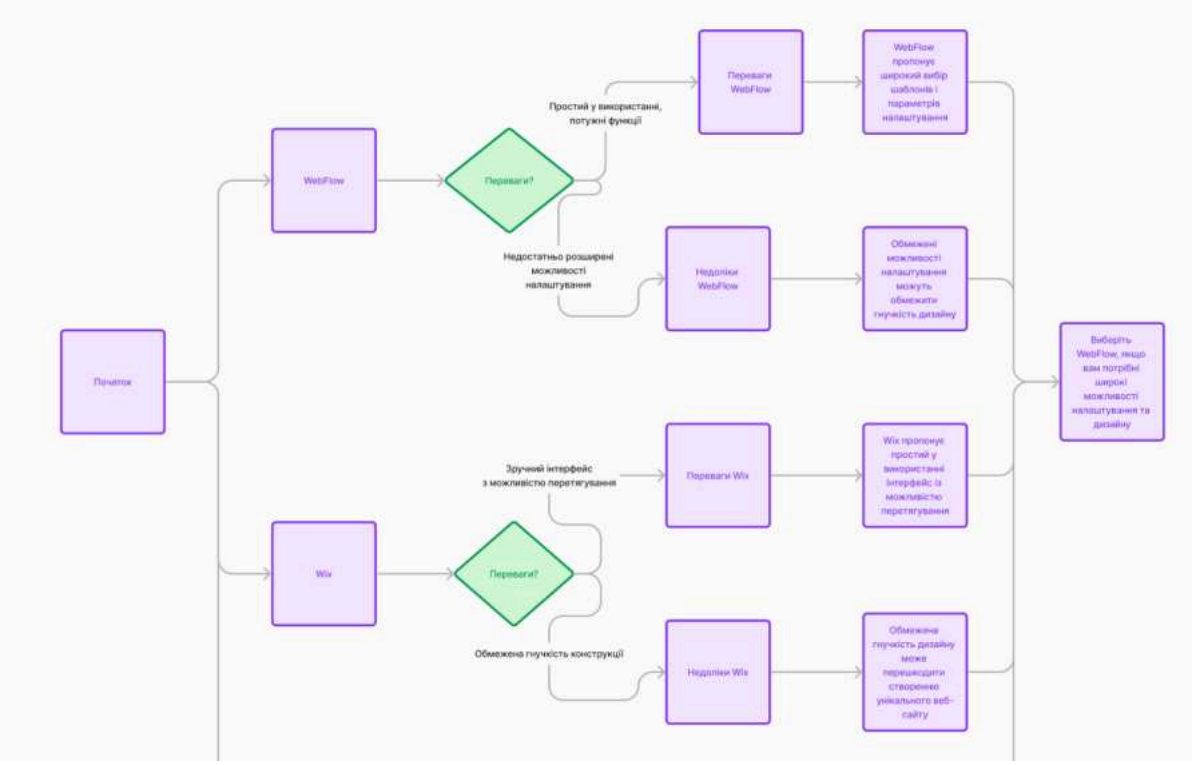


Рисунок 3.1 – Переваги та недоліки Wix і WebFlow

WebFlow переваги простий у використанні, потужні функції, пропонує широкий вибір шаблонів і параметрів налаштування. WebFlow недоліки недостатньо розширені можливості налаштування, обмежені можливості налаштування можуть обмежити гнучкість дизайну.

Wix переваги зручний інтерфейс з можливістю перетягування, пропонує простий у використанні інтерфейс із можливістю перетягування. Wix недоліки обмежена гнучкість конструкції, гнучкість дизайну може перешкодити створенню унікального веб-сайту.

Squarespace переваги інтуїтивно зрозумілий, пропонує професійні шаблони та інтуїтивно зрозумілий інтерфейс. Squarespace недоліки обмежена

інтеграція сторонніх розробників, обмежена стороння інтеграція може обмежити функціональність веб-сайту.

WordPress переваги велика спільнота, широкі можливості плагінів, має вагому спільноту та великий вибір плагінів. WordPress недоліки крута крива навчання, вимагає обслуговування, у WordPress високий поріг входу в навчання та доволі не легке обслуговування.

Порівняння Squarespace та WordPres їх недоліки та переваги на діаграмах на рисунку 3.2.

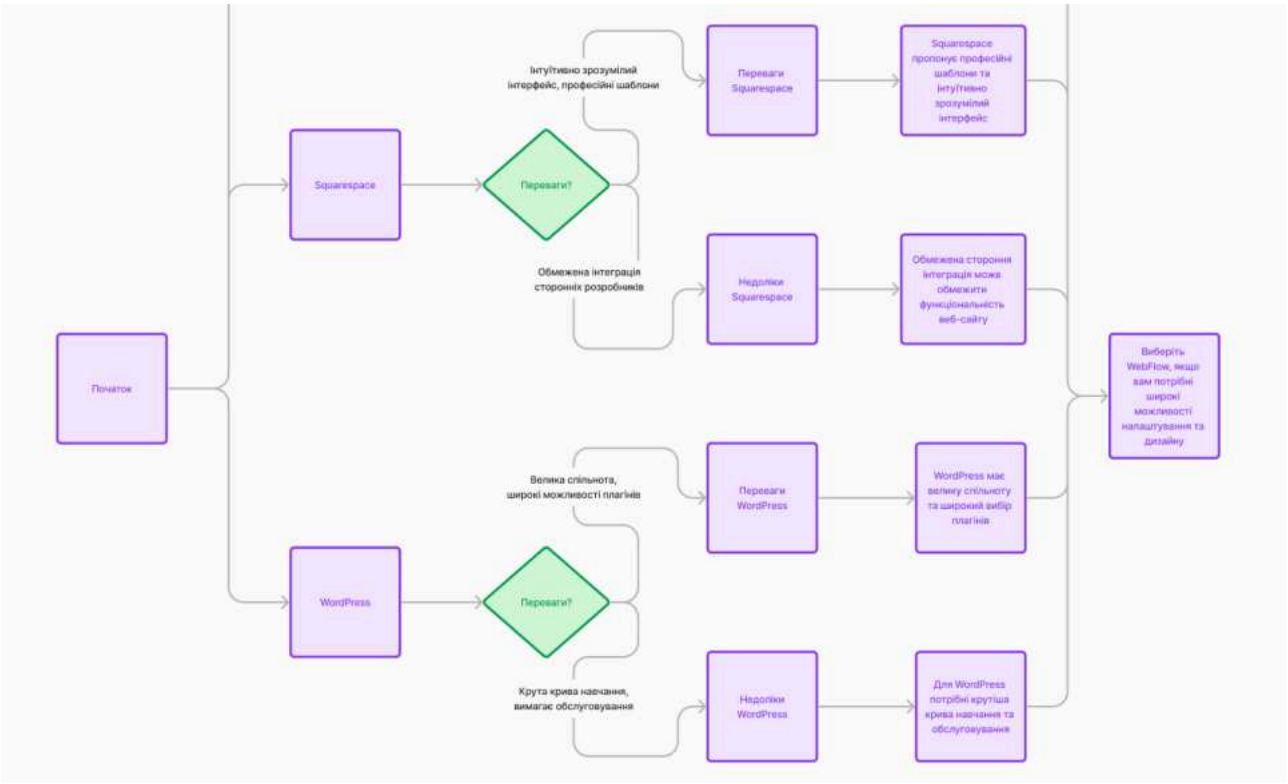


Рисунок 3.2 – Переваги та недоліки Squarespace і WordPress

Як висновок WebFlow потужний веб-редактор, який пропонує широкий спектр функцій. Він простіший у використанні, ніж WordPress, але він складніший, ніж Wix and Squarespace. WebFlow є хорошим вибором для користувачів, які шукають потужний веб-редактор з гнучкими можливостями. Але який веб-редактор вибрати, залежить від ваших потреб і бюджету.

Розробка нових методів та інструментів для WebFlow. WebFlow потужна платформа для веб-розробки, яка постійно розвивається. Існує багато можливостей для розробки нових методів та інструментів для WebFlow.

Створення шаблонів та бібліотек компонентів для повторного використання чудовий спосіб:

1. Підвищити продуктивність, тому що шаблони та компоненти дозволяють розробникам швидко створювати веб-сайти та веб-додатки без необхідності писати код з нуля.

2. Підвищити якість, шаблони та компоненти гарантують, що веб-сайти та веб-додатки будуть мати послідовний дизайн та високу якість коду.

3. Знизити витрати, шаблони та компоненти можуть допомогти розробникам економити час та гроші.

Існує два основних типи шаблонів та компонентів, статичні ці шаблони та компоненти не змінюються після їх створення. Динамічні шаблони та компоненти можуть змінюватися залежно від даних або контексту.

Їх можна створити шаблони та компоненти вручну, використовуючи HTML, CSS та JavaScript. За допомогою інструментів, існує багато інструментів, які дозволяють створювати шаблони та компоненти візуально.

Після того, як ви створили шаблони та компоненти, ви можете поділитися своїми шаблонами та компонентами з іншими розробниками на веб-сайтах, таких як ThemeForest або GitHub або ж використовувати свої шаблони та компоненти у своїх проектах, щоб економити час та гроші.

3.2 Розробка нового способу створення компонентів, який буде більш простим і ефективним

Компоненти дозволяють підтримувати послідовний, ефективний та масштабований робочий процес дизайну, створюючи настроювані блоки з елементів та дочірних елементів. Використовуйте ці блоки на вашому веб-сайті

та змінюйте їх в одному місці, щоб уникнути індивідуального редагування кожного повторниго макета.

Компоненти можна використовувати для:

- створення однакового вмісту на екземпляр, що дозволить створити однакові копії повторюваних макетів, які мають точно такий же вміст, як, наприклад, бари, колонтитули та форми реєстрації. Редагуйте їх в одному місці і побачите, що ці зміни впливають на кожен екземпляр цього компонента;

- створення унікального вмісту на екземпляр. Можна замінити повторювані макети з різним текстом, зображенням, відео чи багатим текстом, щоб підтримувати послідовний дизайн, надаючи кожному екземпляру унікальний вміст. Щоб створити новий компонент потрібно перш за все створити елемент який ми перетворюватимемо в компонент.

До прикладу створюємо компонент меню сайту:

1. В панелі інструментів вибираємо Add Element в лівому верхньому куті (рис. 3.3).

2. Обираємо Link Block та створюємо структуру меню (рис. 3.3).

3. Створюємо структуру меню, аналогічним методом добавляючи елементи (рис. 3.4.).

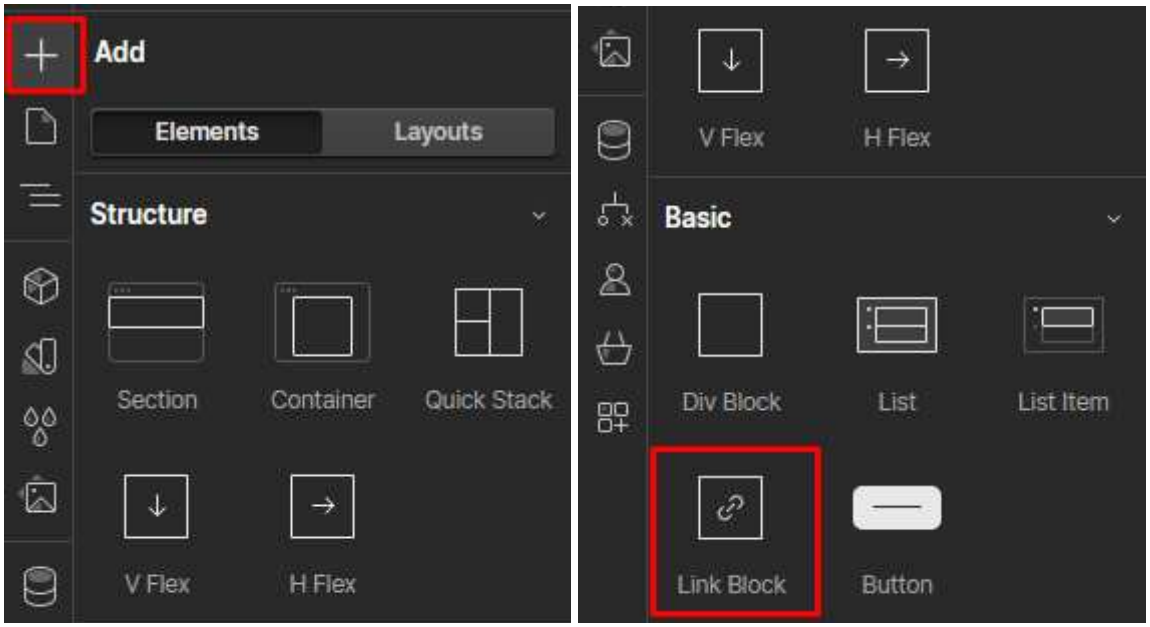


Рисунок 3.3 – Створення елементу меню

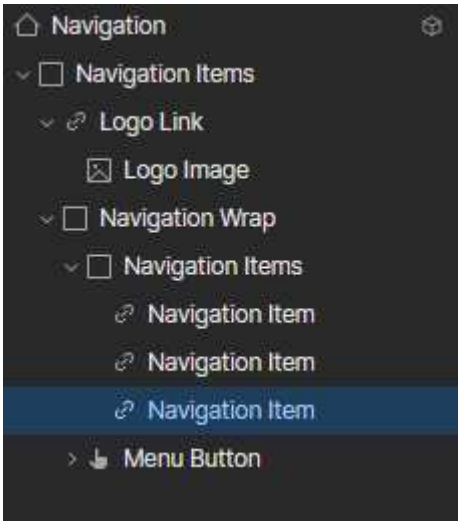


Рисунок 3.4 – Структура меню

Задаємо ім’я класу для елементів в панелі справа у вкладці Style на рисунку 3.5 та назву посилання наприклад Home, About та куди вона посилатиметься у вкладці Settings на рисунку 3.6.

4. Задаємо куди посилатиметься кнопка меню.

5. Перетворюємо батьківський блок меню в компонент:

- вибираємо батьківський елемент (блок) (рис. 3.7);
- відкриваємо панель Style або Element settings (рис. 3.8);
- нажимаємо Create component зверху панелі (рис. 3.8);
- задаємо елементу ім’я і нажимаємо Create (рис. 3.9).

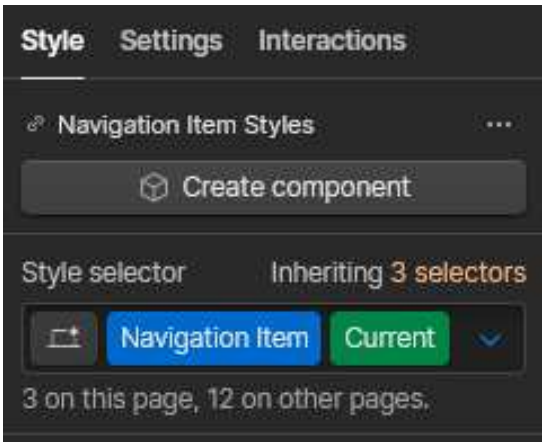


Рисунок 3.5 – Надання класу елементу

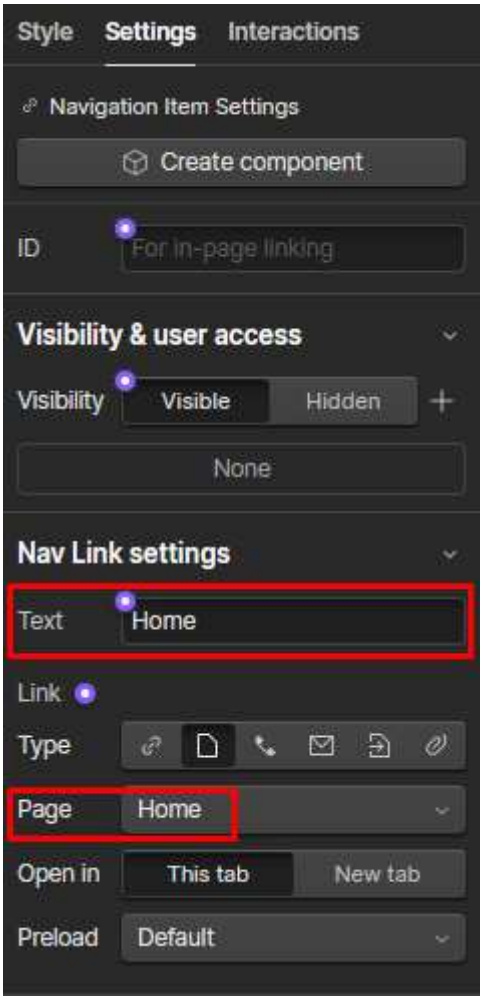


Рисунок 3.6 – Надання ім’я елемента меню

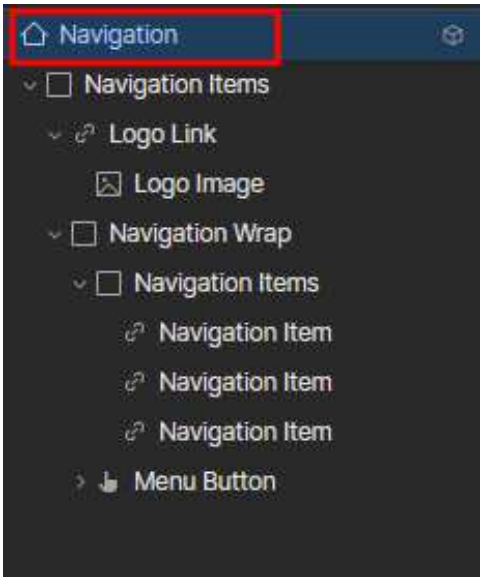


Рисунок 3.7 – Батьківський блок меню

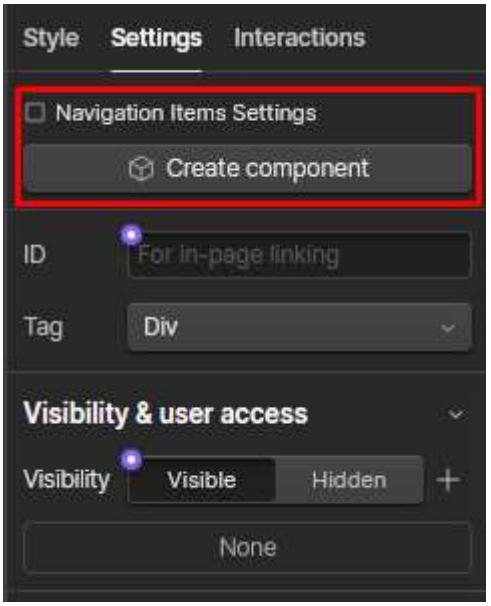


Рисунок 3.8 – Створення компонента

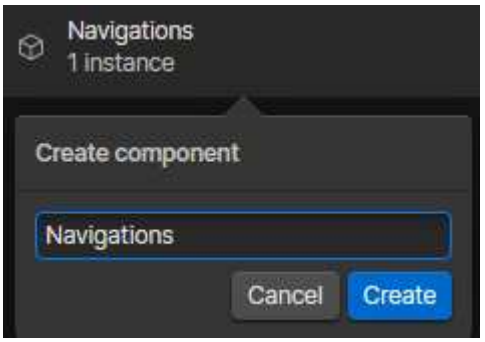


Рисунок 3.9 – Задаємо назву компонента

Після створення компонента ви зможете редагувати основний компонент. Усі зміни основного компонента будуть оновлені у всіх елементах цього компонента сайту чи системи.

Після створення компонента ви можете повторно використовувати його в будь-якому місці вашого веб-сайту.

Виконавши наступні дії:

- відкрийте компонентну панель;
- натисніть та перетягніть потрібний компонент на будь-яку сторінку вашого сайту;
- перетягніть компонентний екземпляр безпосередньо на полотно або в панель елементів.

Вибравши компонентний екземпляр, ви побачите його виділеним та окресленим зеленим кольором. Відкрийте панель «Компоненти», щоб побачити, скільки разів кожен із ваших компонентів використовувався на вашому сайті (рис. 3.10).

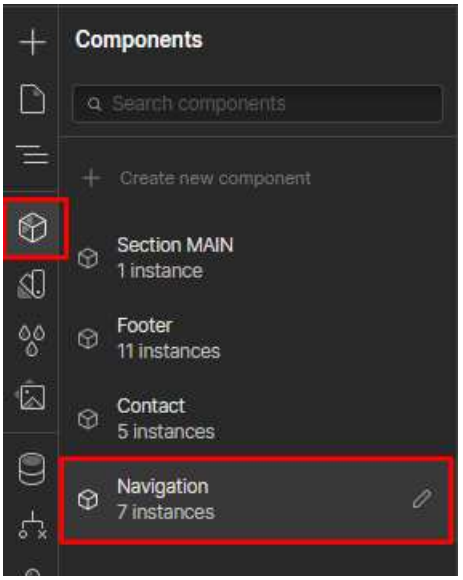


Рисунок 3.10 – Пере використання компонента

Ви можете двічі клацнути компонент, щоб ввести основний компонент і переглянути ієрархію елементів компонента на панелі Навігатора. Ви зможете бачити ієрархію елементів компонента лише з основного компонента (тобто ви не зможете переглянути ієрархію елементів компонента в межах екземпляра компонента системи).

Налаштування компонента. Щоб внести однакові зміни у всіх версіях компонента, вам потрібно буде відредагувати основний компонент. Ви можете редагувати основний компонент у будь-якій версії компонента:

- подвійне клацання екземпляра компонента;
- вибір екземпляра та натискання значка “ олівець ” на етикетці;
- вибір екземпляра та натискання на значок “ гайковий ключ ” у верхньому правому куті правої панелі.

Вибравши варіант компонента, ви можете знайти значення за замовчуванням для властивостей вашого компонента на правій панелі.

Зауважте, що це відображає значення за замовчуванням для властивостей, якщо у вас вже встановлені властивості.

Властивості компонентів дозволяють визначити конкретні значення елементів (наприклад, текст, багатий текст, зображення, відео та видимість) в межах основного компонента, який можна змінити на екземплярі компонентів. Або ви можете підключити властивості компонентів до поля CMS, щоб отримати дані зі своїх колекцій CMS.

Редагування основного компонента вплине на кожен екземпляр компонента (якщо цей екземпляр компонентів не має властивостей компонентів, застосованих до конкретних елементів всередині компонента). Під час редагування основного компонента ви можете змінити порядок структури та елемента, щоб одночасно впливати на всі екземпляри компонентів. Наприклад, у вас може бути дизайн картки, встановлений як компонент. Компонент картки містить елемент заголовка, елемент абзацу та елемент кнопки — з елементом кнопки, розміщеним у самому дні картки. Якщо ви відредагуєте основний компонент картки та перемістіть кнопку знизу картки вгору, ця зміна вплине на всі екземпляри компонента картки на вашому сайті.

Редагування основного компонента впливають на всі екземпляри цього компонента, якщо спеціально не змінено в екземплярі компонентів. Але при потребі ви можете відредагувати компонент, відвязавши його, після чого ви отримаєте повну структуру без прив'язки до класу компонента, що дозволяє редагувати його без змін основного компонента та всіх його дочірніх елементів та модулів які прив'язані.

3.3 Перевикористання компонентів та модулів для створення веб продукту, інтеграція дизайн з Figma в WebFlow

У попередньому підрозділі було розглянуто процес створення компонента. А зараз розглянемо створення компонентної системи з компонентів та модулів, для полегшення інтеграції їх у веб продукт.

Для цього використаємо сервіс Figma, в якому будемо створювати структуру та компоненти для майбутнього сайту. За основу візьмемо сайт для фотографа та його послуг.

Перший екран сайту є важлою частиною сайту, так як воно формує перше враження користувача про сайт. Зазвичай на ньому розміщають картинку яка передає сенс сайту, текст який закликає користувача до дій на цьому сайті, та меню навігації. Також перший екран повинен не нагружати користувача візуальними елементами та повинно давати чітке бачення про що цей сайт. Тому перш за все розбиваємо перший екран на змістовні блоки, щоб дотримуватись так званого правила коробки що на рисунку 3.11. Правило коробки має на меті розбити екран на декілька блоків, щоб пізніше можна було коректно відобразити та адаптовувати сторінку.

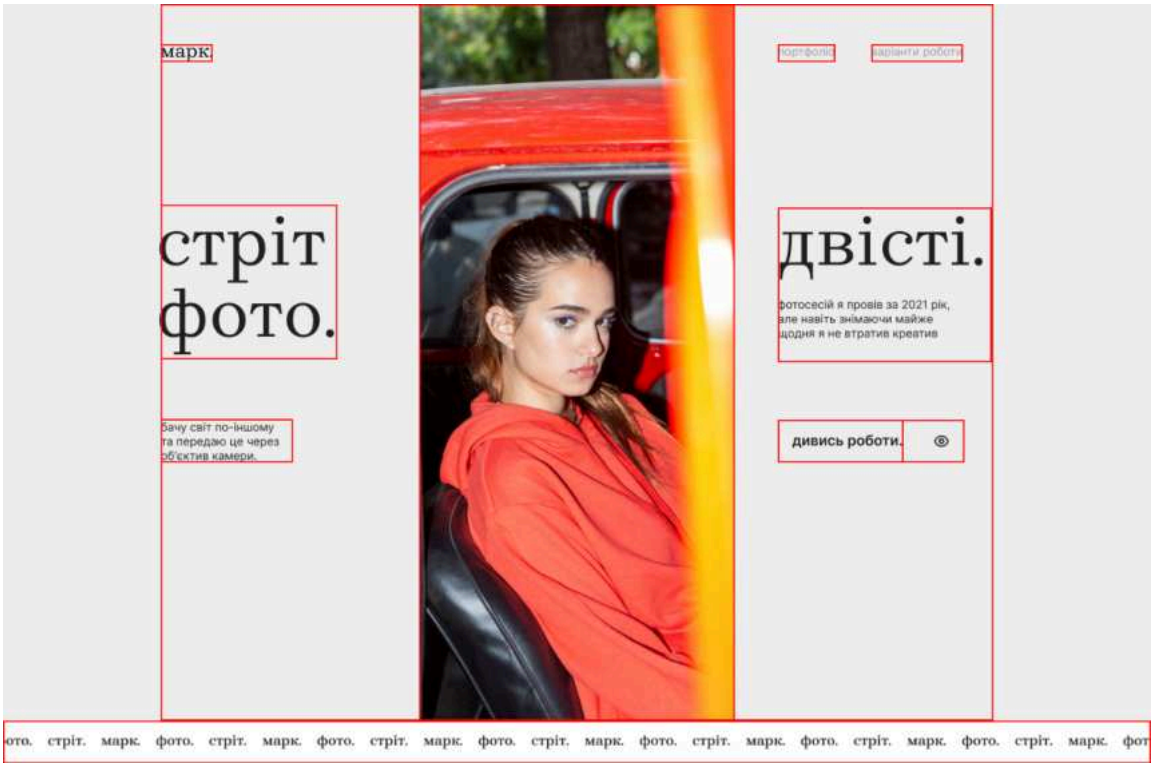


Рисунок 3.11 – Перший екран сайту

Аналогічно розбиваємо і інші екрани сайту. Наступним кроком буде задати кожній групі Auto Layout (рис. 3.12) та (рис. 3.13).

Рисунок 3.12 – Другий екран

Рисунок 3.13 – Auto Layout

В параметрах Auto Layout вибираємо налаштування що відповідають за вирівнювання Align item, чи буде переноситись контент на новий рядок якщо ширини блоку не дозоляє його розмістити Wrap, та відступ від елементів gap between item.

Після чого коли весь наш сайт розбитий та адаптований, використовуємо плагін Figma to WebFlow. Цей плагін розроблений самою компанією WebFlow, що використовує API Figma та API WebFlow. Плагін в бібліотеці плагінів Figma (рис. 3.14).

Рисунок 3.14 – Figma to WebFlow

Плагін в системі виглядає так (рис. 3.15).

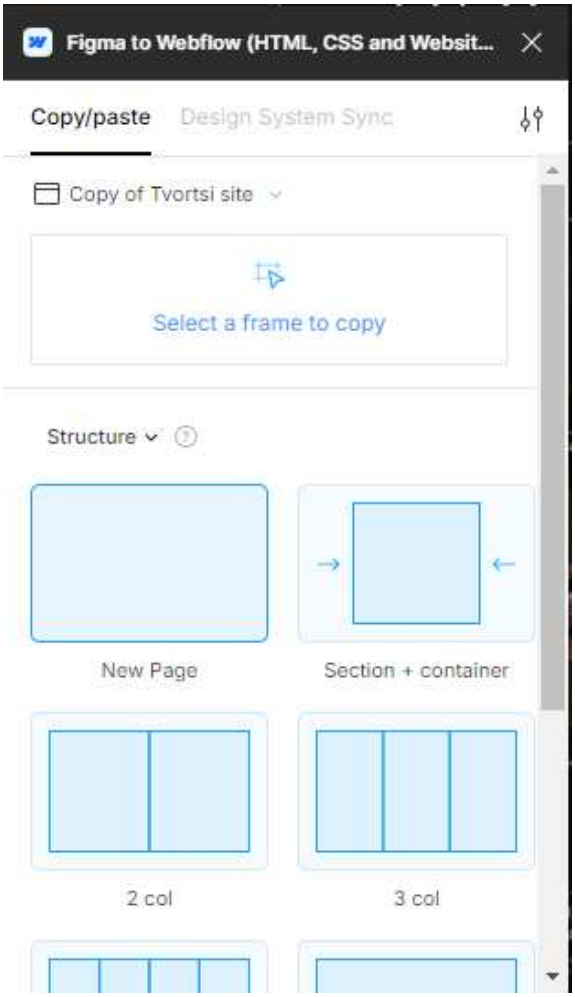


Рисунок 3.15 – Плагін в системі

Важливо зауважити що всі блоки повинні бути в Auto Layout (рис. 3.16.)

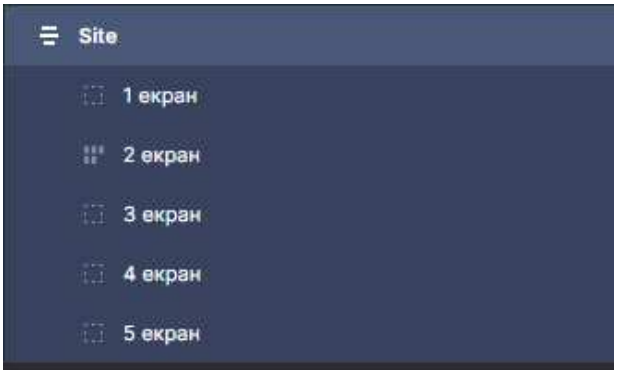


Рисунок 3.16 – Структура сторінки

Вибираємо наш Frame та нажимаємо copy to webflow (рис 3.17.)

Рисунок 3.17 – Імпорт дизайну в Webflow

Після імпорту ми отримуємо чудову структуру сайту яку можемо перетворити за допомогою інструменту створення компонентів в модулі/компоненти та в подальшому перетворити в бібліотеку Layout та розмістити його на webflow libraries community (рис. 3.18).

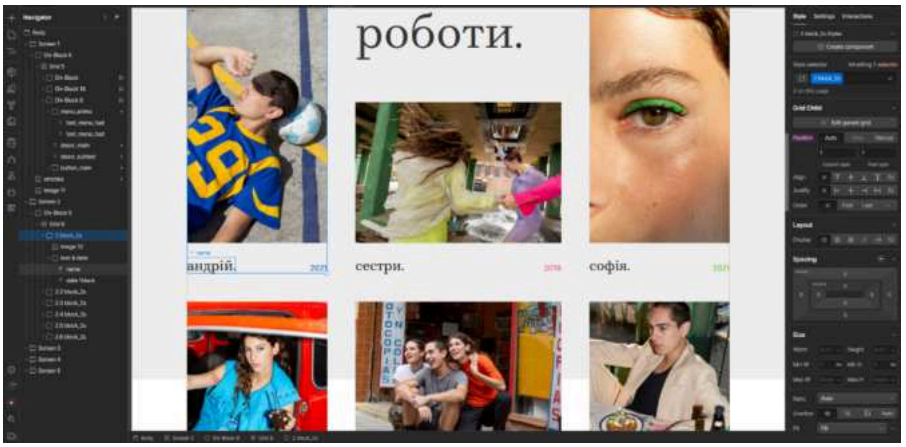


Рисунок 3.18 – Структура сайту після імпорту

3.4 Створення CMS Collections за допомогою модулів та їх налаштування

Колекція схожа на базу даних, це місце де можна зберігати весь вміст, який можна динамічно використовувати в рамках проекту. Навіть якщо немає вмісту для початку, створення колекції дає можливість використовувати фіктивний вміст. Таким чином можна приступити до проектування та динамічної розробки.

У будь-якому проекті можна отримати бажаний доступ до CMS. Як відомо, вміст який ми розміщуємо в колекції - ці елементи - вміст вводиться в поля (рис. 3.19).

Рисунок 3.19 – Структура колекції

Коли в планах є створити колекцію, можна скористатися одним із попередніх налаштувань, які дають хорошу відправну точку, або створити колекцію з нуля. Щоб створити колекцію CMS, перейдіть на панель CMS та натисніть значок Create new Collection у верхньому правому куті панелі. Тоді ви зможете розробити свою колекцію на панелі CMS (рис. 3.20).

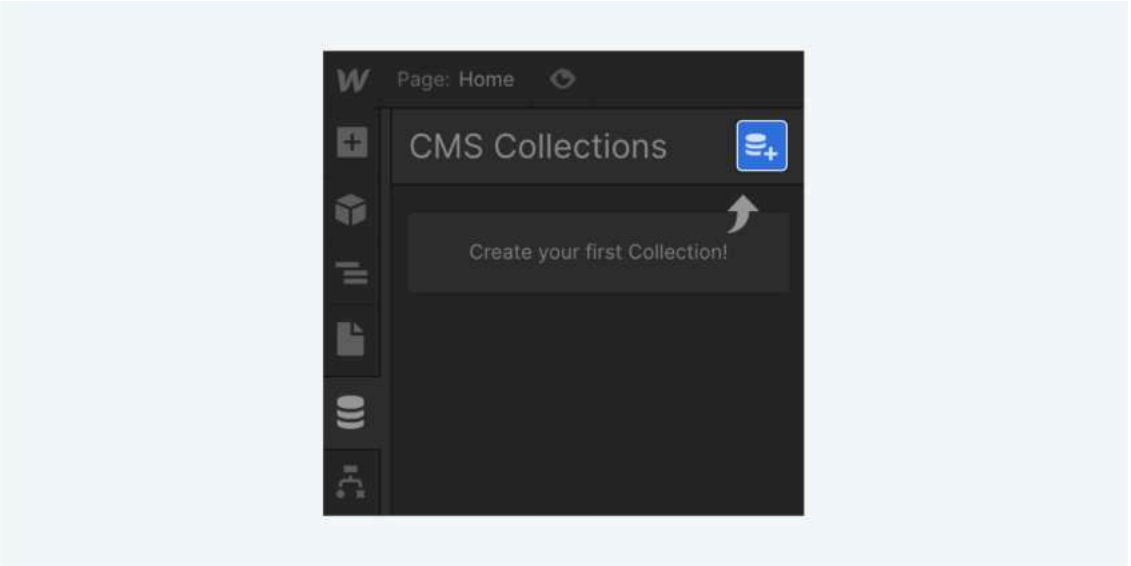


Рисунок 3.20 – Створення CMS Collection

Після чого дійсно можна створити будь що (рис. 3.21).

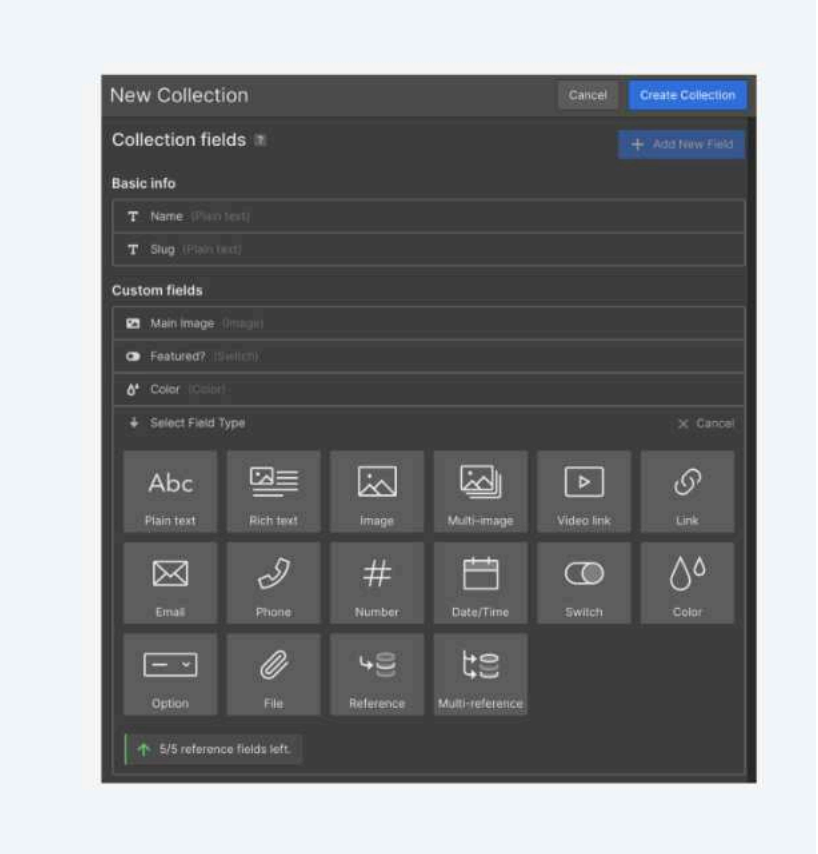


Рисунок 3.21 – Доступні елементи для створення

Поля можна повністю налаштовувати, і мати справді детальний контроль над деталями. Коли створюємо з нуля, ми можемо налаштувати поля в колекції

відповідно до вмісту, з яким працюємо. Для прикладу скористаємось попередньою настройкою Post Blog (рис 3.22).

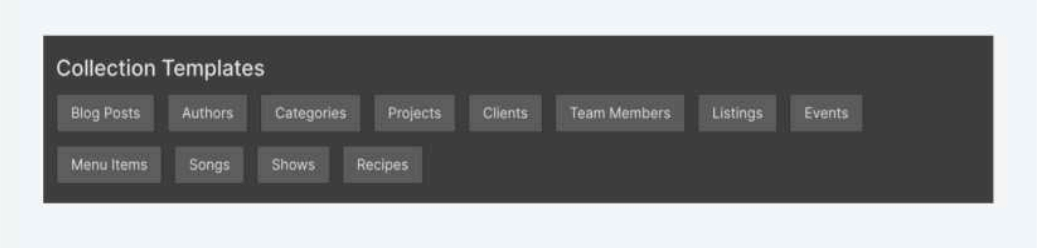


Рисунок 3.22 – Заготовки колекції

На що слід звернути увагу оскільки розглядаємо це, те що поля не є елементами HTML а це просто поля, де зберігаються дані. Після того як буде створено колекції, ми зможемо прив'язати цей вміст до фактичних елементів у проектах та можемо використовувати його практично скрізь на рисунку 3.23 та рисунку 3.24.

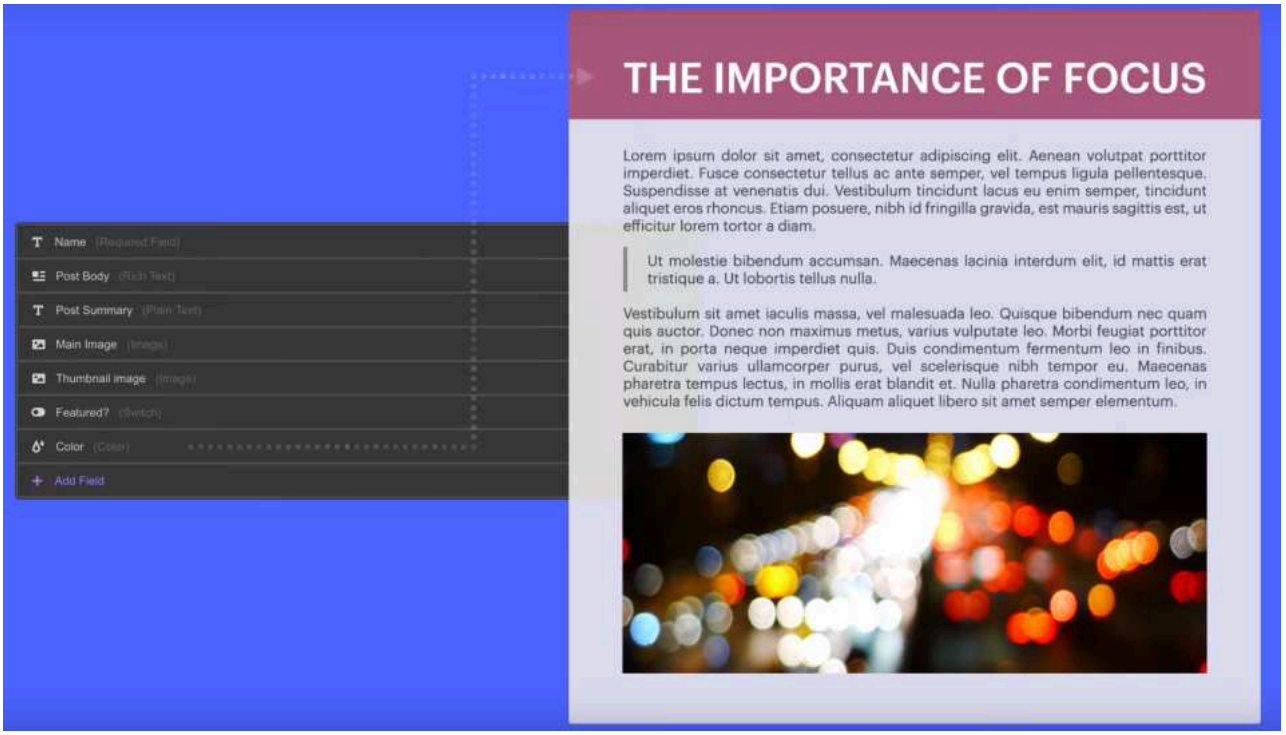


Рисунок 3.23 – Поля та контент

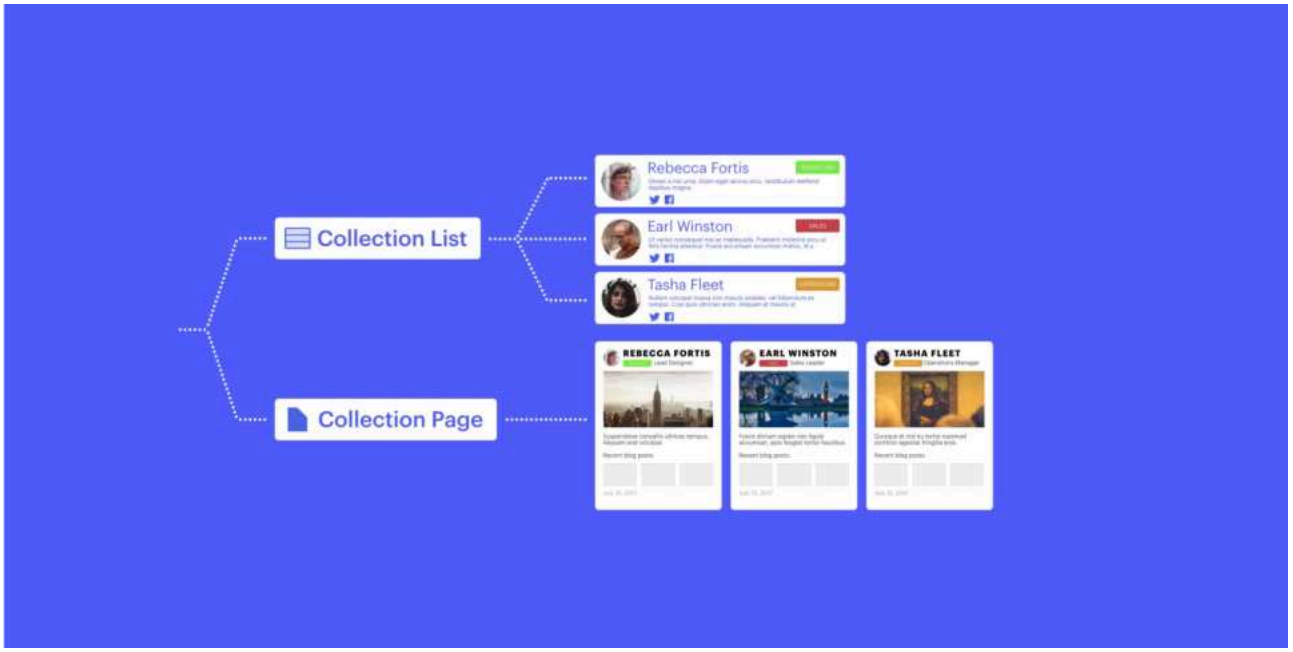


Рисунок 3.24 – Структура колекції та елементів

Пізніше зможемо додавати або змінювати вміст із редактора. Отже, коли налаштовуєм колекцію побачимо попередній перегляд для кожного відповідно поля. Як виглядатиме макет під час використання функціоналу редактора. Редактор - це той варіант, який показуємо якщо хочемо, щоб інші співавтори посту/блогу або користувачі які входять та додають власний контент наповнення. Коли закінчили створення створення колекції нажимаємо Create Collection. Тепер можна створити за шаблоном декілька елементів зразу з рандомним вмістом, що заповнить вашу колекцію (рис. 3.25).

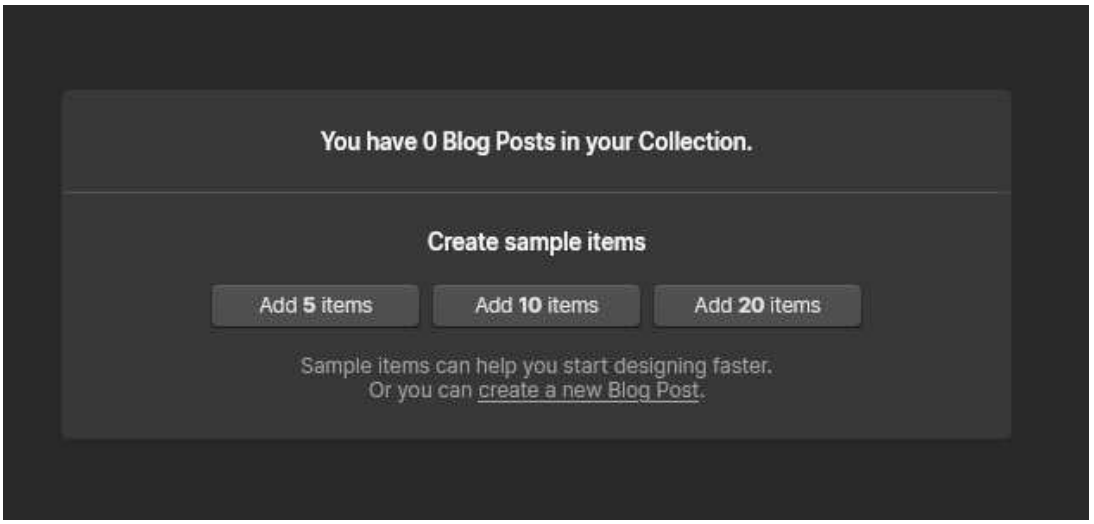


Рисунок 3.25 – Створення елементів колекції

Звичайно після можна редагувати та імпортувати ваш вміст в колекцію але тимчасовий допоможе вам швидко створити і побачити результат на рисунку 3.26.

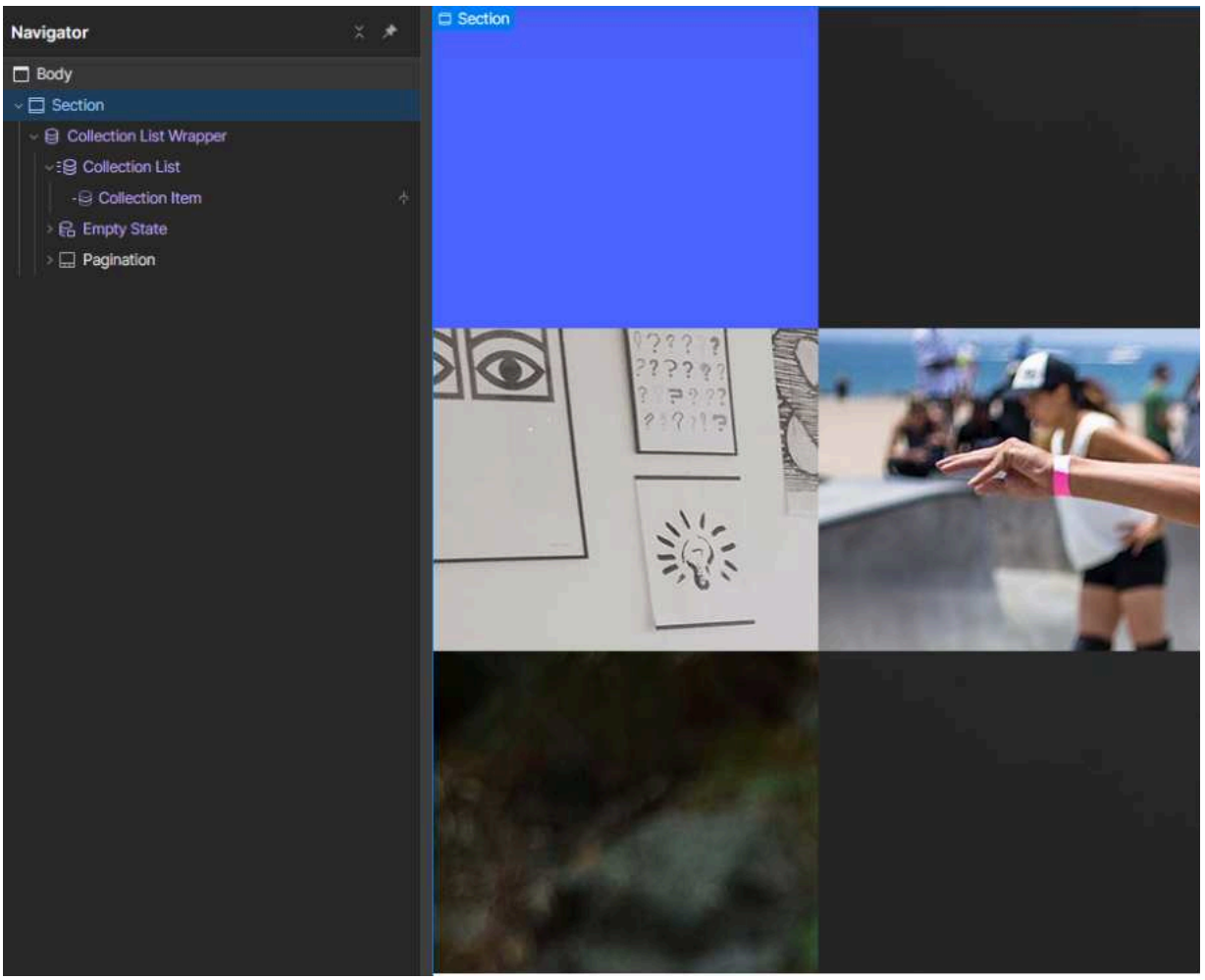


Рисунок 3.26 – Імпорт колекції на сайт

Налаштування CMS Collection

Раніше було створено 10 елементів колекції, щоб налаштувати їх потрібно натиснути на іконку Collection та перейти на колекцію з назвою Blog Posts, нажати кнопку налаштування де відкриється налаштування структури поста нашого блоку (рис. 3.27).

Підкоректуємо структуру поста видаливши зайві поля та залишивши такі поля як Name, Slug, Post Body, Post Summary, Main image (рис. 3.28).

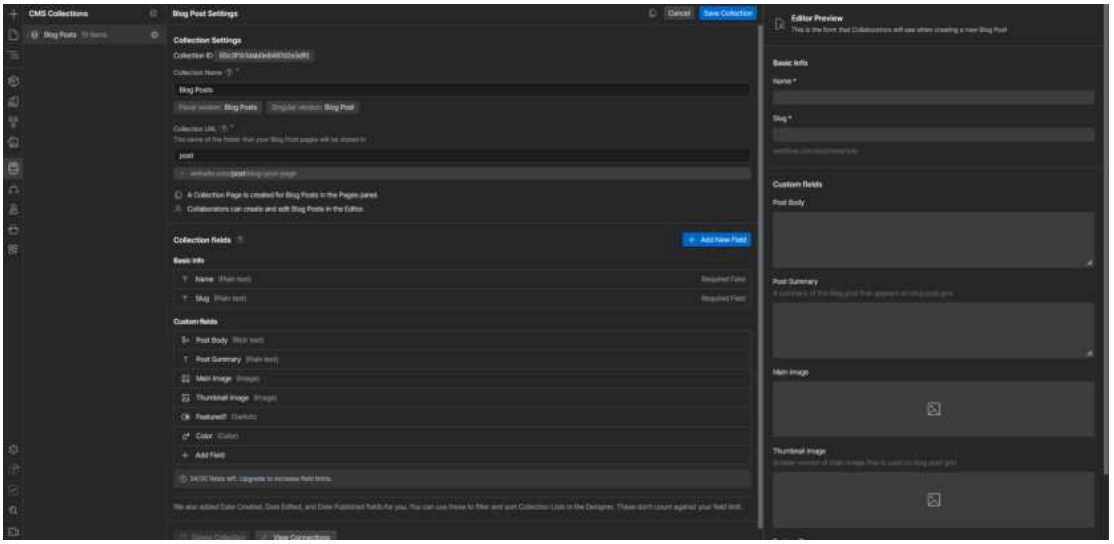


Рисунок 3.27 – Структура колекції

Рисунок 3.28 – Налаштування полів поста блогу

При наведенні в нас відображається превью структури зправа на панелі (рис. 3.29).

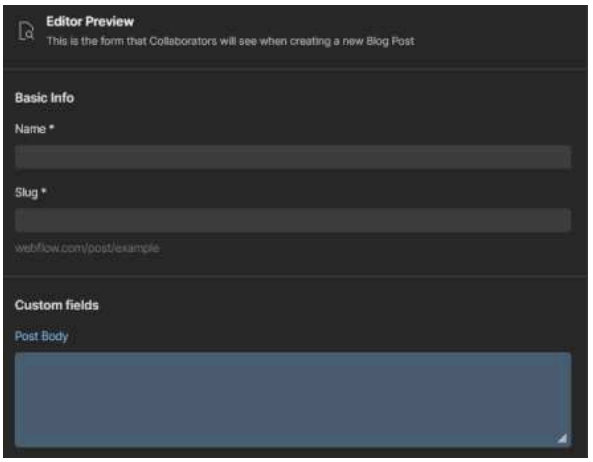


Рисунок 3.29 – Превью структури поста

Зберігаємо колекцію і переходимо в налаштування на самому веб-сайті. В Body сторінки додаємо CMS Collection нажавши на плюсик та в полі CMS, перетягуємо на сторінку або просто нажимаємо правою кнопкою миші, що на рисунку 3.30.

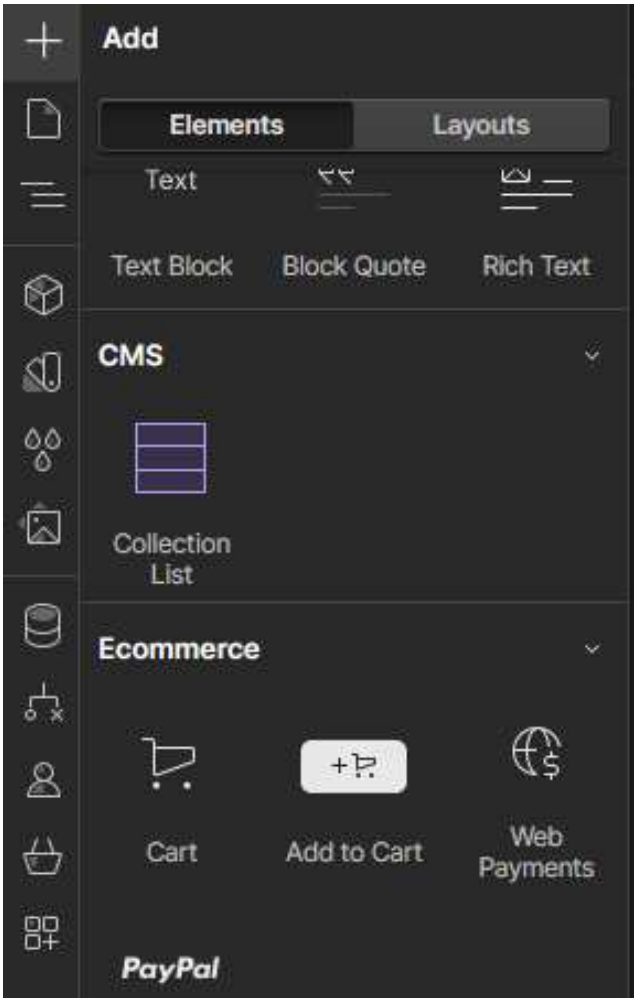


Рисунок 3.30 – Імпорт колекції

Після того як додали колекцію, вибираємо звідки буде тягнутись контент поста (рис. 3.31).

Тут бачимо, що можемо вибирати не тільки з колекції а і з Ecommerce вкладки, додаючи карточки товарів або категорії товарів. Так як і те і інше являє собою колекції або базу даних з певними параметрами. Тому у Source вибираємо Blog Post.

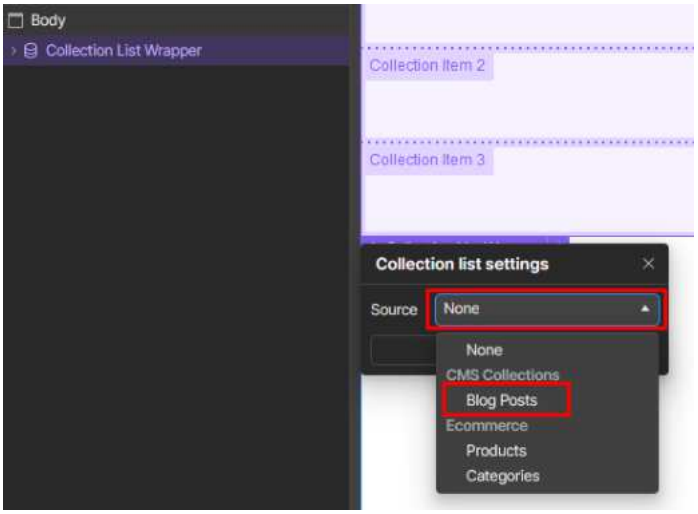


Рисунок 3.31 – Імпорт контенту в пост

Вибираємо йому структуру в три колонки (рис. 3.32).

Рисунок 3.32 – Розташування постів

Тепер щоб вони відобразились потрібно їм задати розміри в лівій панелі Style, надавши їм висоту та ширину (рис. 3.33).

Рисунок 3.33 – Надання розмірів посту

Додаємо генерацію випадкових зображень із колекції WebFlow, в вкладці налаштування (рис. 3.34).

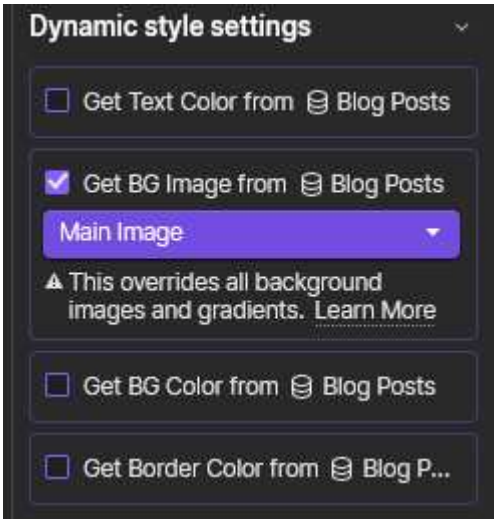


Рисунок 3.34 – Імпорт картинок в пост

Після того як налаштували Колекцію і добавили її на сайт, на цьому робота не закінчується, тому що після додавання колекції на сайт вона відображається некоректно, без вирівнювання картинки посту, та без тексту. Щоб виправити це потрібно додати всі поля колекції які були створені раніше та за допомогою базових навичок CSS стилів налаштувати їх вигляд в лівій панелі. Тим самим задаючи всім наявним та наступним елементам один шаблон стилів, щоб в подальшому можна було перед використовувати шаблон та проводити легкий супровід сайту. Добавимо в нашу колекцію div блоки та загорнемо їх батьківськими блоками, додамо поля з описом, задомо їм розмір шрифт та відступи.

Спочатку блоку колекції з назвою Collection List Wrapper задамо Flex box параметр, Direction: Vertical, Align: Center, Don't Wrap тому що у нас все буде в одному потоці по вертикалі.

Елементу Collection List 2 задаєм ширину 80% і висоту 100%, position: static. В результаті у нас блоки йдуть потоком вирівнювання по центру, тому всі нові пости будуть вже вирівнюватися по центру (рис. 3.35).

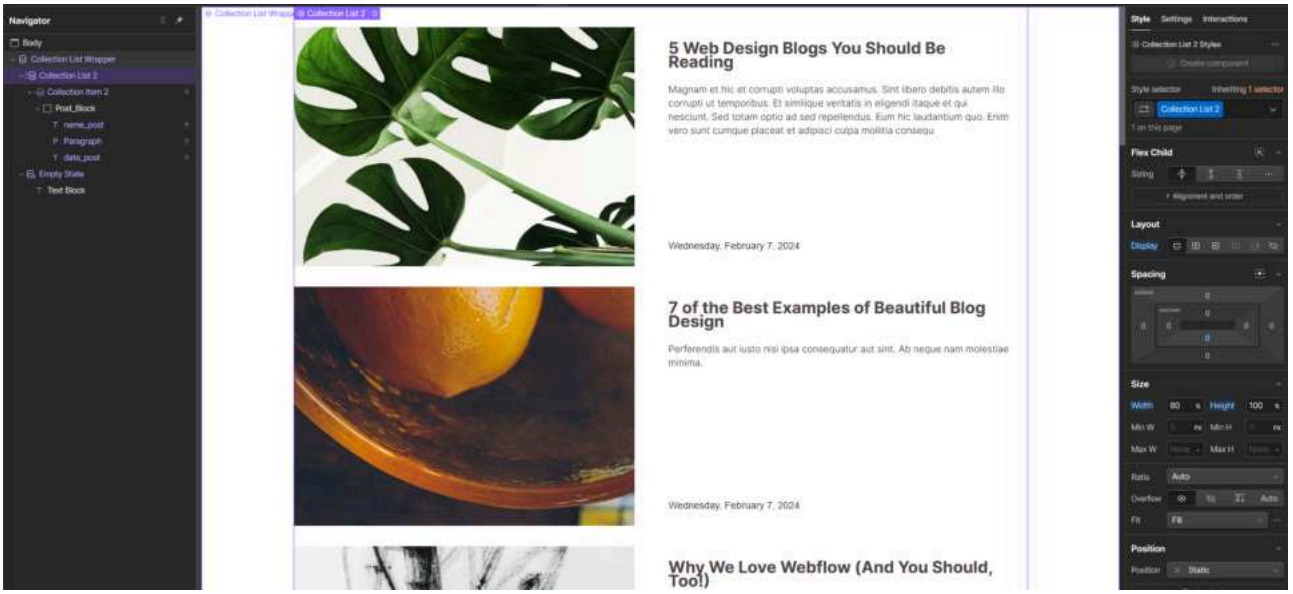


Рисунок 3.35 – Вирівнювання контенту сайту

Елемент Collection Item 2 являється являється основним постом який має в собі такі параметри як name(назва поста) paragraph(текст поста) date_post (дата публікації поста) але ви на нього можемо повісити клас картинки та додати інші поля в середину цього елемента. Створимо div з назвою Post_Block дотримуючись конвенції іменування, для зрозумілості що собою являє елемент та за що відповідає. В елемент Post_Block додаємо текстове поле з ім'ям name_post, текстове поле з датою публікації date_post, та параграф з текстом поста Paragraph.

Отож елементу Collection Item 2 задамо такі параметри margin top and bottom 30px, ширину 500px, висоту 350 px, position: relative (рис. 3.36).

Тепер перейдмо до елемента Post_Block, йому ви задамо ширину 100% та висоту 100%, і він отримає розміри батьківського блоку Collection Item 2. Далі margin 110% і position: relative, тим самим ми відділили все що в середині Post_Block і розмістили справа від картинки на відстані ширини батьківського блоку +10%. Елементу name_post задаємо наступні параметри margin top & bottom 20 px, position: static, шрифт Inter 700 - Bold, розмір 24, колір #4b4444. Елемент Paragraph залишаємо незмінним тільки міняємо значення Weight 300 - Light, розмір шрифту 14. date_post винесемо в самий низ position: absolute bottom в результаті отримаємо таку структуру поста (рис. 3.37).

Рисунок 3.36 – Collection Item 2

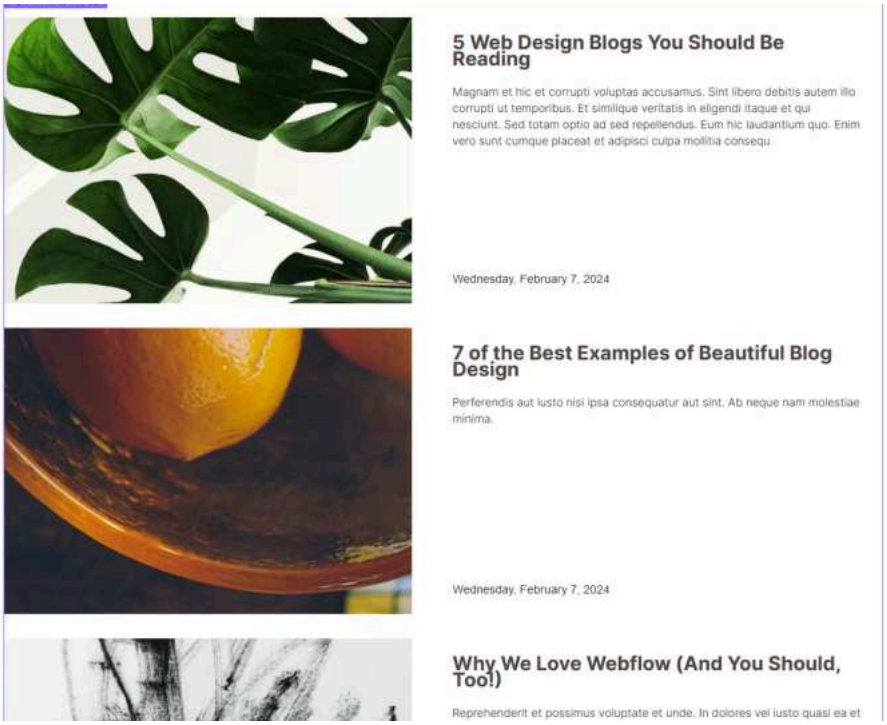


Рисунок 3.37 – Структура постів

Результатом розробки CMS системи є створення шаблону поста для сайту блогу в якому є 10 елементів, які ми можемо в будь який момент використати, створити новий та настроювати за потреби (рис. 3.38).

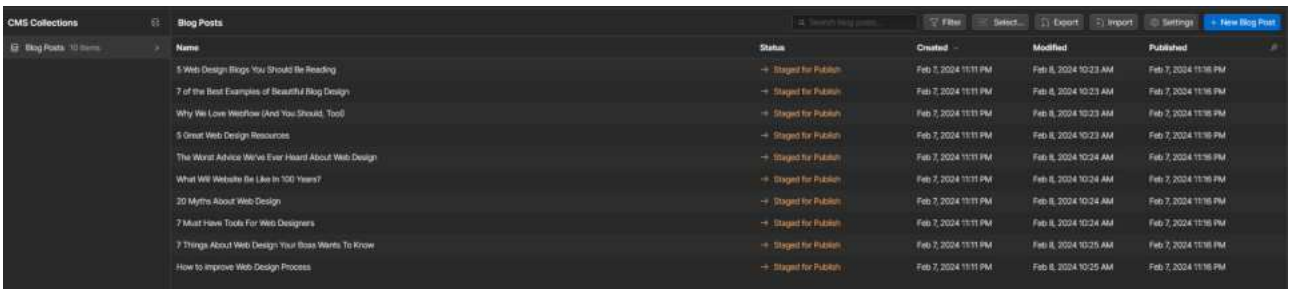


Рисунок 3.38 – Пости в колекції

3.5 Створення Ecommerce Collections та налаштування картки продукту

Створення картки продукту має таку ж структуру як і створення колекцій але з своїми додатковими властивостями.

Перш за все зайдемо в e-commerce налаштування, зліва кнопка корзини (рис. 3.39).

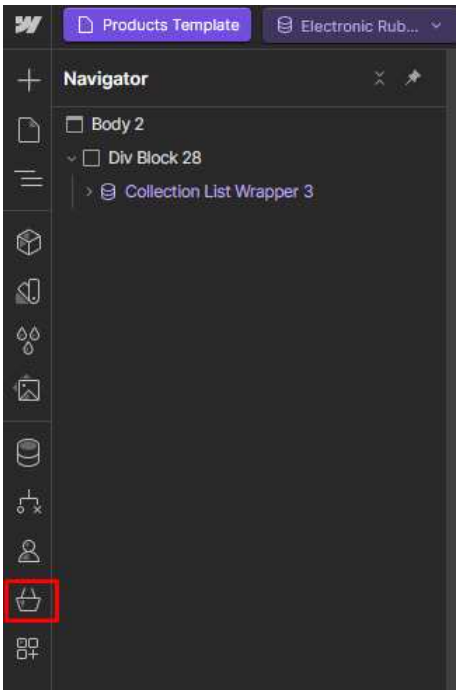


Рисунок 3.39 – E-commerce налаштування

Переходимо в вкладку Products та створюємо 5 товарів, в результаті нам генерує 5 товарів які ми можемо налаштовувати (рис. 3.40).

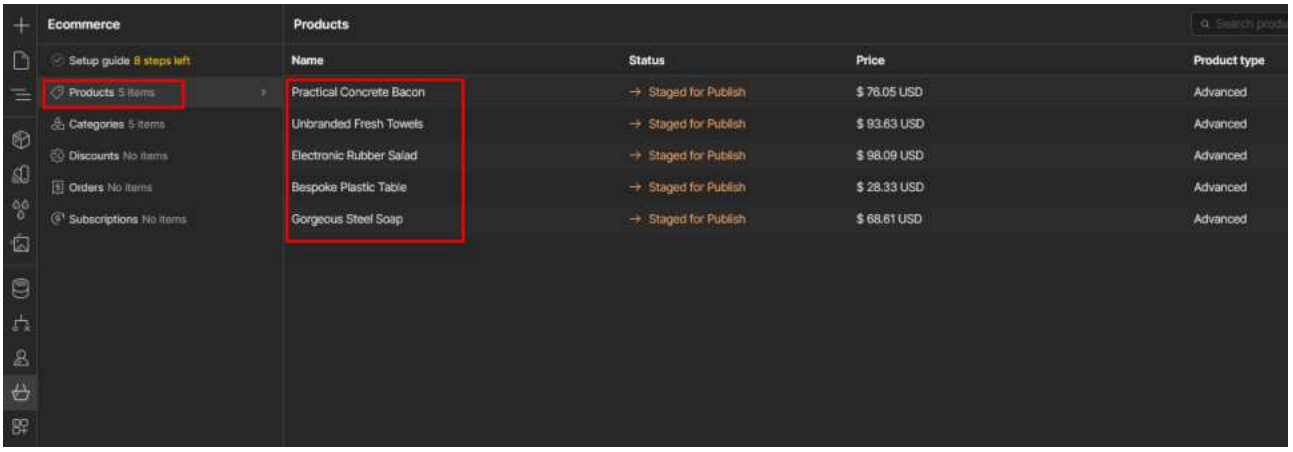


Рисунок 3.40 – Створені продукти

Звичайно можна їх створити вручну але для старту та розуміння як працюють продукти їх було згенеровано. Наступний крок відкрити панель налаштування всіх продуктів, щоб побачити їх поля та з чим працюватимемо.

Щоб перейти в налаштування нажимаємо іконку налаштування на владці Products (рис. 3.41).

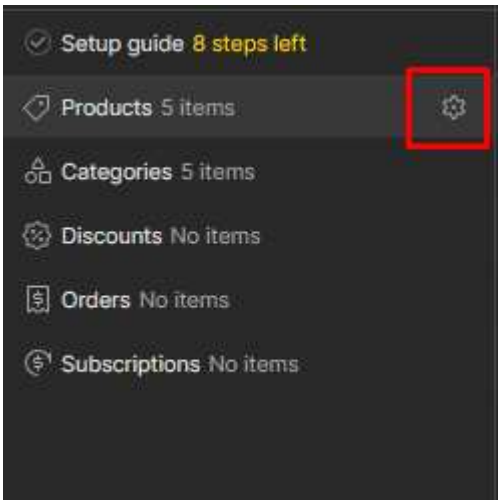


Рисунок 3.41 – Налаштування продукту

Коли перейшли в налаштування, можна помітити що тут така ж сама структура налаштування де ми бачимо поля та їх попередній вигляд, та які

параметри може приймати поле. Виберемо тільки такі поля як product type, name, main img, price. Тепер можна їх добавляти на сайт., протягнувши кнопку collection в панелі інструментів або нажавши на неї.

Тепер вони імпортувались на сайт, наступна дія це відкрити налаштування Collection Item та вибрати саме наш продукт (рис. 3.42).

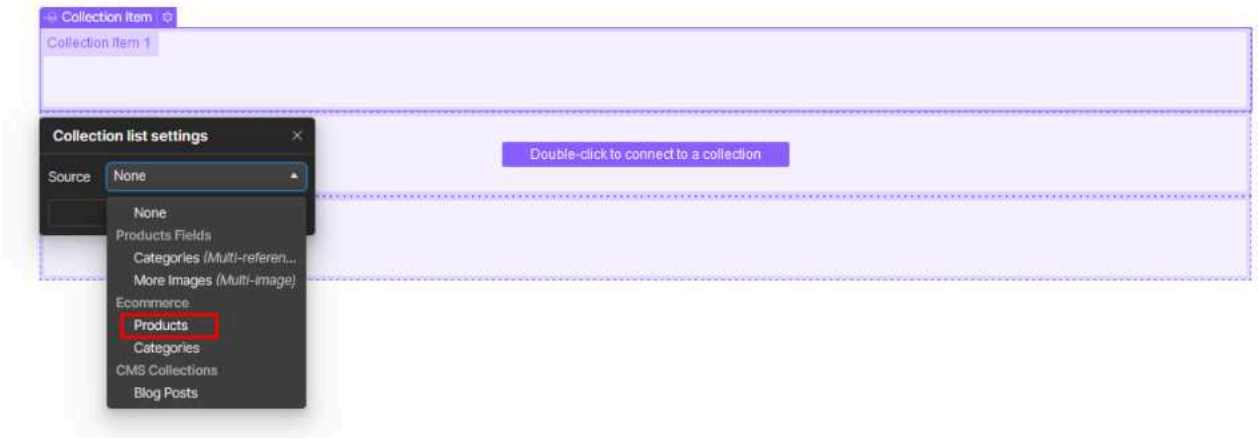


Рисунок 3.42 – Підключення колекції продуктів

Після чогоу нас створиться порожня сітка розташування продуктів але ми їх ще не побачимо, так як не задала розміри, задаючи розміри елементу Collection item ми задаємо всім зразу ті ж налаштування. Тож для початку огорнемо елементи main img та name в div блок (рис. 3.43).

Рисунок 3.43 – Структурування карточки

Додамо div блок для price та product type та отримаємо структуру карточки товару (рис. 3.44).

Рисунок 3.44 – Структура карточки товару

Переіменовуємо елменти та їх класи відповідно конвенції іменування та переходимо до підключення фото та надання стилів для фото товару (рис. 3.45).

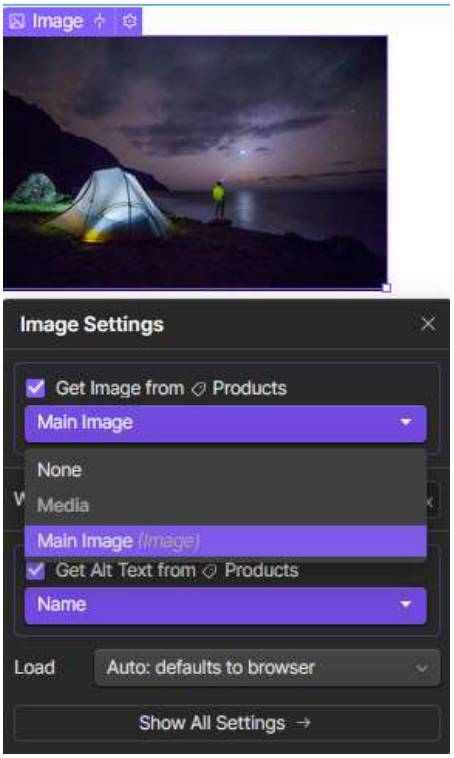


Рисунок 3.45 – Підключення фото з колекції

Вибираємо image переходимо в налаштування та обираємо Main Image та Alt text обираємо Name, щоб коли у нас не прогрузилась картинка, то було ім'я товару замість фото.

Отримуємо результат на (рис. 3.46).

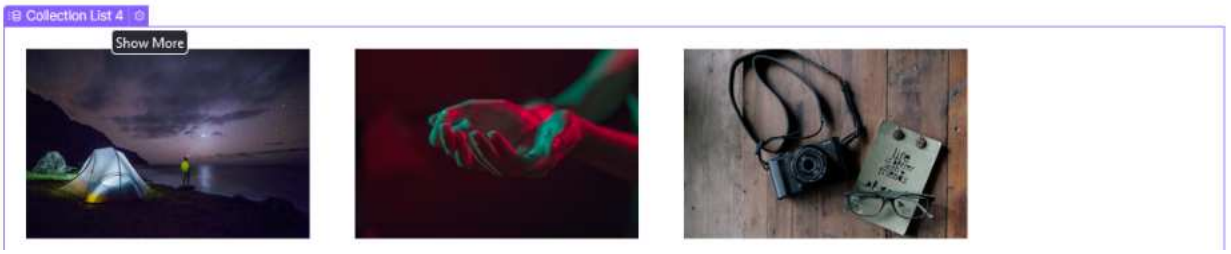


Рисунок 3.46 – Фото товару

Перейдемо до налаштування Collection item задаємо 250 px ширину, що в наслідку дасть всім елементам дочірнім таку ж ширину, тобто product_card буде мати таку ж ширину і зображення в ньому також. Для простору між товарами добавимо margin 20 px з всіх сторін (рис. 3.47).

Рисунок 3.47 – Налаштування розмірів та відступів

Елементу name_product задаємо шрифт Inter, 600 - Semi Bold, розмір 16px. Вирівнюємо його по лівому краю і отримуємо товар з ім'ям та зображенням (рис. 3.48).

Рисунок 3.48 – Зображення та ім'я

Переходимо до налаштування ціни та назви типу продукту, вибираємо елемент price та переходимо в налаштування змінимо йому на Inter, 700 - Bold, розмір 16px, color #6db86f та вирівнюємо по лівому краю (рис. 3.49).

Рисунок 3.49 – Налаштування стилів ціни

Елементу product_type задамо наступні стилі Inter, 600 - Semi Bold, 14px, color #a780ac на цьому стилів закінчено (рис. 3.50).

Рисунок 3.50 – Налаштування стилів типу продукту

В результаті у нас виходить не поганий шаблон для карточки товару який можна ще налаштовувати додавати поля та елементи (рис. 3.51).

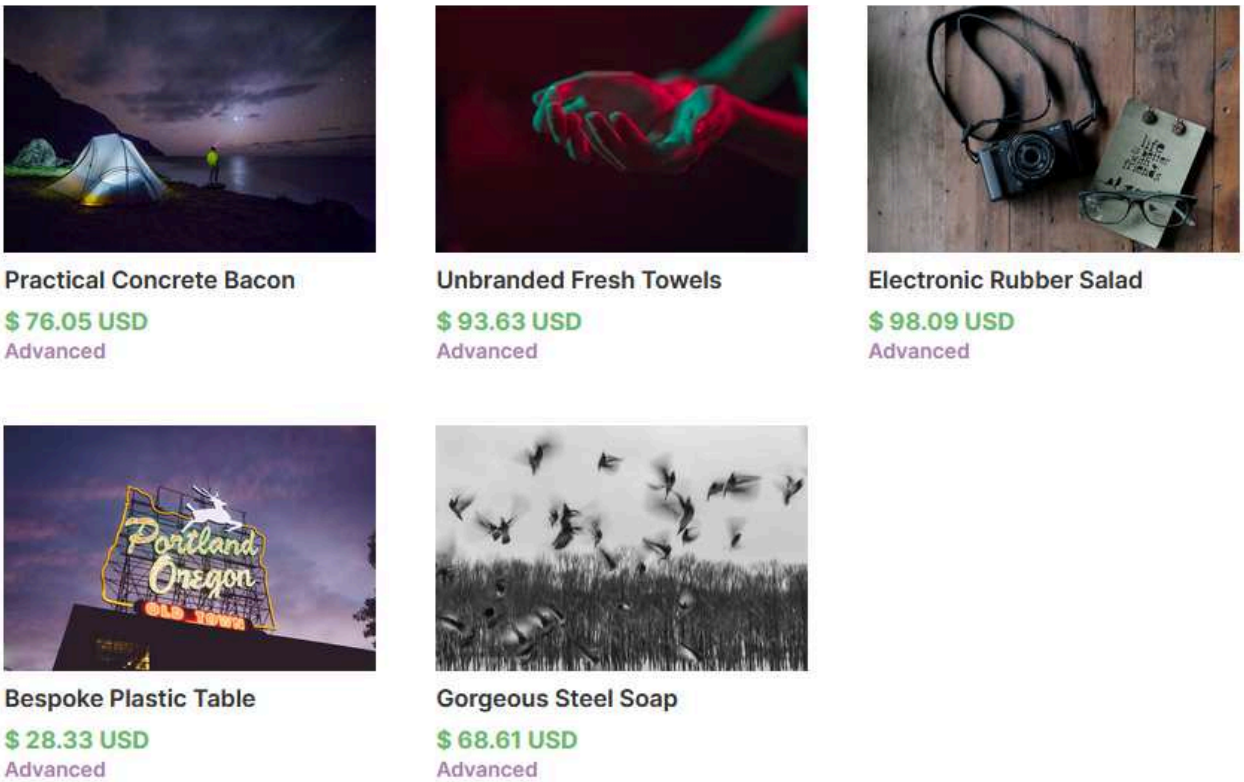


Рисунок 3.51 – Карточки товарів

3.6 Порівняння робочого процесу розробки: звичайний розробник і розробник Webflow з розробленими методами

Процес розробки звичайного розробника, зрозумійте типові кроки та процеси, пов’язані з розробкою веб-сайту звичайним розробником.

Крок 1: збір вимог, співпраця із зацікавленими сторонами, щоб зібрати вимоги до проекту, включаючи функціональність, дизайн і цільову аудиторію.

Крок 2: планування та дизайн, створення плану проекту, окресливши часові рамки, основні етапи та результати. Розробка макету веб-сайту, каркасу та інтерфейсу користувача.

Крок 3: Front-end розробка, написання коду HTML, CSS і JavaScript, щоб створити інтерфейс веб-сайту. Забезпечення оперативності, доступності та кросбраузерну сумісність.

Крок 4: Back-end розробка, розробка функціональних можливостей веб-сайту, такі як сценарії на стороні сервера, інтеграція бази даних та інтеграція API.

Крок 5: тестування та налагодження, ретельна перевірка функціональності, продуктивності і безпеки веб-сайту. Виявлення та виправлення будь-яких помилок чи проблеми.

Крок 6: розгортання, підготовка веб-сайт до розгортання, включаючи налаштування хостингу, налаштування домену та остаточне тестування. Розгортання веб-сайту на живому сервері.

Крок 7: технічне обслуговування та оновлення, відстеження продуктивності веб-сайту, безпеку та відгуки користувачів. Регулярне оновлення, виправлення помилок та вдосконалення за потреби.

Робочий процес розробки Webflow, виконайте такі кроки, щоб створити веб-сайт за допомогою Webflow.

Крок 1: дизайн і макет, створення дизайну і макету веб-сайту за допомогою візуального редактора Webflow. Використання інтерфейсу

перетягування, щоб додавати елементи, налаштовувати стилі та впорядковувати вміст нашого сайту.

Крок 2: адаптивний дизайн, перевірити, що веб-сайт адаптивний і оптимізований для різних пристроїв. Використання адаптивних інструментів дизайну Webflow, щоб налаштувати макет, точки зупинки та взаємодію для перегляду на мобільних пристроях, планшетах і комп'ютерах.

Крок 3: взаємодія та анімація, додавання взаємодії та анімації для покращення взаємодії з користувачем. Використання вбудованих засобів взаємодії Webflow або створення власних анімацій за допомогою CSS і JavaScript.

Крок 4: управління вмістом, керування вмістом веб-сайту та оновлення його за допомогою системи керування вмістом (CMS) Webflow. Створення динамічного вмісту, додавання колекцій та підключення до зовнішніх джерел даних чи баз даних.

Крок 5: співпраця та публікація, співпраця з членами команди та клієнтами, ділячись проектом у Webflow. Перегляд та надсилання відгуку за допомогою функції коментування. Коли все буде готово, опублікування веб-сайту у спеціальному домені або на хостингу Webflow.

Аналіз продуктивності. Порівняння робочих процесів розробки звичайного розробника та розробника Webflow, щоб визначити, який із них є більш продуктивним.

Переваги звичайного робочого процесу розробника:

- повний контроль над кодом і налаштуваннями;
- можливість роботи з будь-яким стеком технологій;
- гнучкість для впровадження складних функцій.

Переваги Webflow Developer Workflow:

- не вимагає кодування, візуальний інтерфейс;
- швидке прототипування та ітерація;
- чуйний дизайн із коробки.

Недоліки звичайного робочого процесу розробника:

- трудомістке кодування та налагодження;
- залежність від технічних навичок;
- крута крива навчання для початківців.

Недоліки Webflow Developer Workflow:

- обмеження в налаштуваннях і гнучкості;
- залежність від платформи Webflow.

Висновок до розділу 3

У третьому розділі розглянуто використання інтерактивного веб-редактора WebFlow як інноваційного інструменту для спрощення розробки та візуальної взаємодії з елементами веб-додатків. Та проведено порівняння з іншими інструментами такими як Wix, Squarespace, WordPress. Також проведено порівняння робочого процесу розробки: звичайний розробник і розробник Webflow з розробленими методами. Розроблено новий спосіб створення компонентів. Розглянуто інструмент Figma та роботу з плагінами та правилом коробки для розробки сайтів. Продемонстровано перевикористання компонентів та модулів для створення веб продукту, інтеграція дизайн з Figma в WebFlow. Створено CMS Collections за допомогою модулів та створено Ecommerce Collections і карти продукту.

ВИСНОВКИ

В кваліфікаційній роботі було проведено аналіз модульності, перевикористання коду, обґрунтування актуальності розробки методів. Проведено аналіз конкуретних методів та взагальному методів розробки. Розглянуто впливу ШІ та машинного навчання на розробку. Розглянуто компонентну архітектуру та патерни проектування та тестування окремих модулів цієї системи.

Було розроблено нові підходи до створення компонентів та модулів, за допомогою інструментів Figma та Webflow. Також проведено порівняння робочого процесу розробки: звичайний розробник і розробник Webflow з розробленими методами. Використовували плагіни та правила коробки як новий метод для розробки сайтів в сполученні з Webflow.

Продемонстровано перевикористання компонентів та модулів для створення веб продукту, інтеграція дизайн з Figma в WebFlow. Створено CMS Collections за допомогою модулів та створено Ecommerce Collections.

Детально розглянуто та описано створення компонентів, колекцій, модулів, та їх стилізація і налаштування. Описано всі етапи розробки від створення архітектури сайту до розробки окремих елементів. Здійснено огляд функціональних можливостей розроблених підходів та створених компонентів, колекцій для сайту.

У висновку кваліфікаційної роботи було проведено великий робочий процес що сприятиме покращенню ефективності процесу розробки, забезпечує легкість супроводу веб-додатків та зниження трудомісткості внесення змін. Крім того, розроблені підходи допомагають створити високоякісні продукти з кращим користувацьким досвідом, що сприяє підвищенню конкурентоспроможності на ринку.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Martin, Robert C. Clean Code: A Handbook of Agile Software Craftsmanship: книга. Prentice Hall, 2008. 464 с.

2. Hunt, Andrew., David Thomas. The Pragmatic Programmer: Your Journey to Mastery: книга. Addison-Wesley Professional, 2019. 352 с.

3. Freeman, Eric., Elisabeth Robson., Bert Bates. Head First Design Patterns: A Brain-Friendly Guide: книга. O'Reilly Media, 2004. 692 с.

4. McConnell, Steve. Code Complete: A Practical Handbook of Software Construction: книга. Microsoft Press, 2016. 916 с.

5. Documenting web technologies, including CSS, HTML, and JavaScript : веб-сайт. URL: https://developer.mozilla.org/en-US/ (дата звернення: 12.11.2023)

6. Martin, Robert C. Clean Architecture: A Craftsman's Guide to Software Structure and Design: книга. Prentice Hall, 2017. 432 с.

7. Crockford, Douglas. JavaScript: The Good Parts: книга. O'Reilly Media, 2008. 153 с.

8. Allsopp, Jon. Modular Web Design: Creating Reusable Components for User Experience Design and Documentation: книга. New Riders, 2013. 340 с.

9. Wroblewski, Luke. Mobile First: книга. A Book Apart, 2011. 130 с.

10. Ousterhout, John K. A Philosophy of Software Design: книга. Yaknyam Press, 2018. 190 с.

11. Documentation WebFlow: веб-сайт. URL: https://university.webflow.com/#home-docs (дата звернення: 27.11.2023)

12. Journal of Web Engineering: веб-сайт. URL: https://journals.riverpublishers.com/index.php/JWE (дата звернення: 10.12.2023)

13. CSS-Tricks: веб-сайт. URL: https://css-tricks.com/guides/ (дата звернення: 16.12.2023)

14. Opportunities for AI in Accessibility: веб-сайт. URL: https://alistapart.com/article/opportunities-for-ai-in-accessibility/ (дата звернення: 18.12.2023)

15. Learn Responsive Web Design: веб-сайт. URL: https://www.freecodecamp.org/news/responsive-web-design-certification-redesigned/ (дата звернення: 26.12.2023)

16. Stack Overflow: веб-сайт. URL: https://stackoverflow.com/ (дата звернення: 26.12.2023)

17. GitHub: веб-сайт. URL: https://github.com/ (дата звернення: 26.12.2023)

18. CodePen: веб-сайт. URL: https://codepen.io/ (дата звернення: 26.12.2023)

19. Google Developers: веб-сайт. URL: https://developers.google.com/focus/web-development (дата звернення: 26.12.2023)

20. Web Design Weekly: веб-сайт. URL: https://www.umso.com (дата звернення: 27.12.2023)

21. Responsive Design Weekly: веб-сайт. URL: https://css-tricks.com/responsive-web-design-weekly/ (дата звернення: 27.12.2023)

22. Bootstrap: веб-сайт. URL: https://getbootstrap.com/ (дата звернення: 28.12.2023)

23. Web Designer Depot: веб-сайт. URL: https://www.webdesignerdepot.com/ (дата звернення: 03.01.2024)

24. Material Design: веб-сайт. URL: https://m3.material.io/ (дата звернення: 08.01.2024)

25. SitePoint: веб-сайт. URL: https://www.sitepoint.com/ (дата звернення: 10.01.2024)

26. InfoQ: веб-сайт. URL: https://www.infoq.com/ (дата звернення: 12.01.2024)

27. W3Schools: веб-сайт. URL: https://www.w3schools.com/ (дата звернення: 13.01.2024)

28. Tuts+: веб-сайт. URL: https://tutsplus.com/ (дата звернення: 16.01.2024)

29. Designmodo: веб-сайт. URL: https://designmodo.com/ (дата звернення: 19.01.2024)

30. Speckyboy Design Magazine: веб-сайт. URL: https://speckyboy.com/ (дата звернення: 20.01.2024)

31. Патерни у проєктуванні інтерфейсів: веб-сайт. URL: https://telegraf.design/paterny-u-proyektuvanni-interfejsiv/ (дата звернення 22.01.2024)

32. Рефакторинг: веб-сайт. URL: https://foxminded.ua/refaktorynh/ (дата звернення 22.01.2024)

33. UI-фреймворки та бібліотеки JavaScript. веб-сайт. URL: https://devzone.org.ua/post/ui-freimvorki-ta-biblioteki-javascript (дата звернення 23.01.2024)

34. Дизайн-системи. веб-сайт. URL: https://cases.media/article/oglyad-dizain-sistem (дата звернення 24.01.2024)

35. Предпроцесори CSS. веб-сайт. URL: https://mate.academy/blog/front-end-and-js/css-preprocessors/ (дата звернення 26.01.2024)

36. Створюй компоненти в Figma з розумом. веб-сайт. URL: https://ux.pub/aldr_cwa/stvoriui-komponienti-v-figma-z-rozumom-1f9k (дата звернення 28.01.2024)

37. Norman, Don. The Design Of Everyday Things: книга. Basic Books, 2019. 368 с.

38. Saffer, Dan. Designing for Interaction: книга. New Riders, 2017. 239 с.

39. Gothelf, Jeff. Lean UX: Designing Great Products: книга. O'Reilly, 2016. 184 с.

40. Frost, Brad. Atomic Design: книга. Brad Frost, 2016. 189 с.

метадані

Заголовок

Дослідження та розробка нових підходів для забезпечення модульності, перевикористання коду та полегшення супроводу у веб-дизайні

Автор

Піхоцький.Н.Т. Науковий керівник / Експерт

підрозділ

King Danylo University

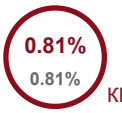
Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

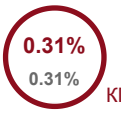
Заміна букв		0
Інтервали		0
Мікропробіли		2
Білі знаки		0
Парафрази (SmartMarks)	a	7

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КП 2

25
Довжина фрази для коефіцієнта подібності 2

15278
Кількість слів

116195
Кількість символів

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	http://repository.ukd.edu.ua:8080/bitstream/handle/123456789/379/%D0%91%D0%B0%D0%BB%D0%B0%D0%BD%D1%8E%D0%BA_%D0%86_%D0%86_%D0%9A%D0%92%D0%90%D0%9B%D0%86%D0%A4%D0%86%D0%9A%D0%90%D0%A6%D0%86%D0%98%CC%86%D0%9D%D0%90_%D0%A0%D0%9E%D0%91%D0%9E%D0%A2%D0%90.pdf?sequence=1	48	0.31 %
2	http://repository.ukd.edu.ua:8080/bitstream/handle/123456789/379/%D0%91%D0%B0%D0%BB%D0%B0%D0%BD%D1%8E%D0%BA_%D0%86_%D0%86_%D0%9A%D0%92%D0%90%D0%9B%D0%86%D0%A4%D0%86%D0%9A%D0%90%D0%A6%D0%86%D0%98%CC%86%D0%9D%D0%90_%D0%A0%D0%9E%D0%91%D0%9E%D0%A2%D0%90.pdf?sequence=1	15	0.10 %