МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерних наук

ЛІПЕЦЬКИЙ Роман Олегович

Інтелектуалізована веб-система генерування сайтів на основі заданого контенту / Intellectualized Web System for Sites Generating based on Given Content

спеціальність: 121 - Інженерія програмного забезпечення
освітньо-професійна програма - Інженерія програмного забезпечення

Кваліфікаційна робота

Виконав студент групи ІПЗм-21
Р. О. Ліпецький

_____

Науковий керівник:
к.т.н., доцент, В. І. Манжула

_____

Кваліфікаційну роботу
допущено до захисту:

"___" _____ 20___ р.

Завідувач кафедри
_____ **А. В. Пукас**

ТЕРНОПІЛЬ - 2022

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DOM - об’єкта модель документа

SEO – пошукова оптимізація сайту

Об'єкт – колекція властивостей та методів

Парсер – програма яка забезпечує синтаксичний аналіз контенту

Вступ

Актуальність теми. У зв’язку з повномасштабною війною, перед більшістю підприємств постає питання – як залучити нових клієнтів в такій складній ситуації за допомогою інтернету. Тому все більше компаній звертаються за послугами розробки сайтів.

Розробка сайту – це складний процес, який задіює велику кількість спеціалістів, через що вартість створення сайту є досить високою. В сьогоднішніх реаліях не всі компанії можуть дозволити собі розробку сайту. Тому виникає питання – як автоматизувати процес створення сайту, щоб знизити вартість розробки, та зробити її більш доступною.

Для вирішення даної проблеми було розроблено систему, яка генерує веб-сайт на основі контенту користувача.

Алгоритми розробленої системи, дозволяють без будь яких технічних знань створити сайт в декілька простих дій.

Мета дослідження. Для виконання поставленої мети потрібно виконати наступні завдання:

- виконати аналіз існуючих систем розробки сайтів;
- обґрунтувати вибір алгоритмів;
- розробити систему автоматичного генерування веб-сайту.

Предмет дослідження. Предметом дослідження є алгоритм аналізу даних.

Об'єктом дослідження. Об’єктом дослідження є системи розробки веб-сайтів.

Методи дослідження: теоретичний аналіз і систематизація науково-теоретичних та методичних джерел; алгоритмізації процесів з використанням сучасних пакетів прикладних програм.

Наукова новизна отриманих результатів. Наукова новизна отриманих результатів полягає в алгоритмі аналізу вхідних даних та перетворенню тексту в повноцінний веб-сайт.

Практичне значення одержаних результатів полягає в тому, що програмним забезпеченням реалізовано алгоритм аналізу вхідних даних, ідентифікація типу переданого контенту, перетворення його під підтримувальний тип даних, та генерації шаблону сайту на основі отриманих результатів алгоритму.

Особистий внесок здобувача. Основні результати магістерської роботи отримані автором самостійно на основі власних ідей та розробок.

Публікації:

1. Ліпецький Р.О.

РОЗДІЛ 1
ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ СИСТЕМИ АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ ВЕБ-САЙТУ

1.1 Коротка характеристика об’єкту управління

Сьогодні як ніколи є актуальним питання організації продажу продукції чи послуг в мережі інтернет. Тому все більше компаній звертаються до веб-студій для розробки персоналізованого веб-сайту. Проте вартість створення якісного веб-сайту є досить великою, оскільки потребує великих затрат людських ресурсів, тому не всі компанії мають фінансову можливість замовити такий програмний продукт, зокрема і через повномасштабну війну в нашій країні. І тут постає питання, як співставити бажання компанії отримати веб-сайт з її фінансовими можливостями. Тому в даній дипломній роботі запропоновано систему, яка дозволяє створити сайт на основі контенту, який надає користувач, без посередництва ІТ-спеціаліста.

Останні декілька років спостерігається тенденція зростання кількості пошукових запитів на предмет розробки сайту, дана тема є досить актуальною, тому розробка такої системи дозволить зайняти певне місце, в сфері надані послуг розробки.

Варто відмітити що створення сайту є складним та тривалим процесом, який задіює велику кількість спеціалістів з різних сфер, тому в результаті вартість розробки сайту є досить високою[20].

Першим кроком розробки сайту є створення технічного завдання на основі інформації яку надає клієнт. Даний етап можна також назвати як стратегія створення сайту, оскільки детально описуються усі наступні етапи розробки. В технічному завданні також описуються усі вимоги, та побажання клієнта щодо розробки сайту [7].

Далі створюється прототип сайту або ще як його називають «мокап». Мокап – це схематичне відображення сторінки, яке дозволяє визначитися зі структурою та функціоналом майбутнього сайту. Зазвичай прототип сайту створюється в інструментах розробки інтерфейсу, такий як Figma. На Рис.1.1 показано вигляд прототипу мобільної версії сайту [10].

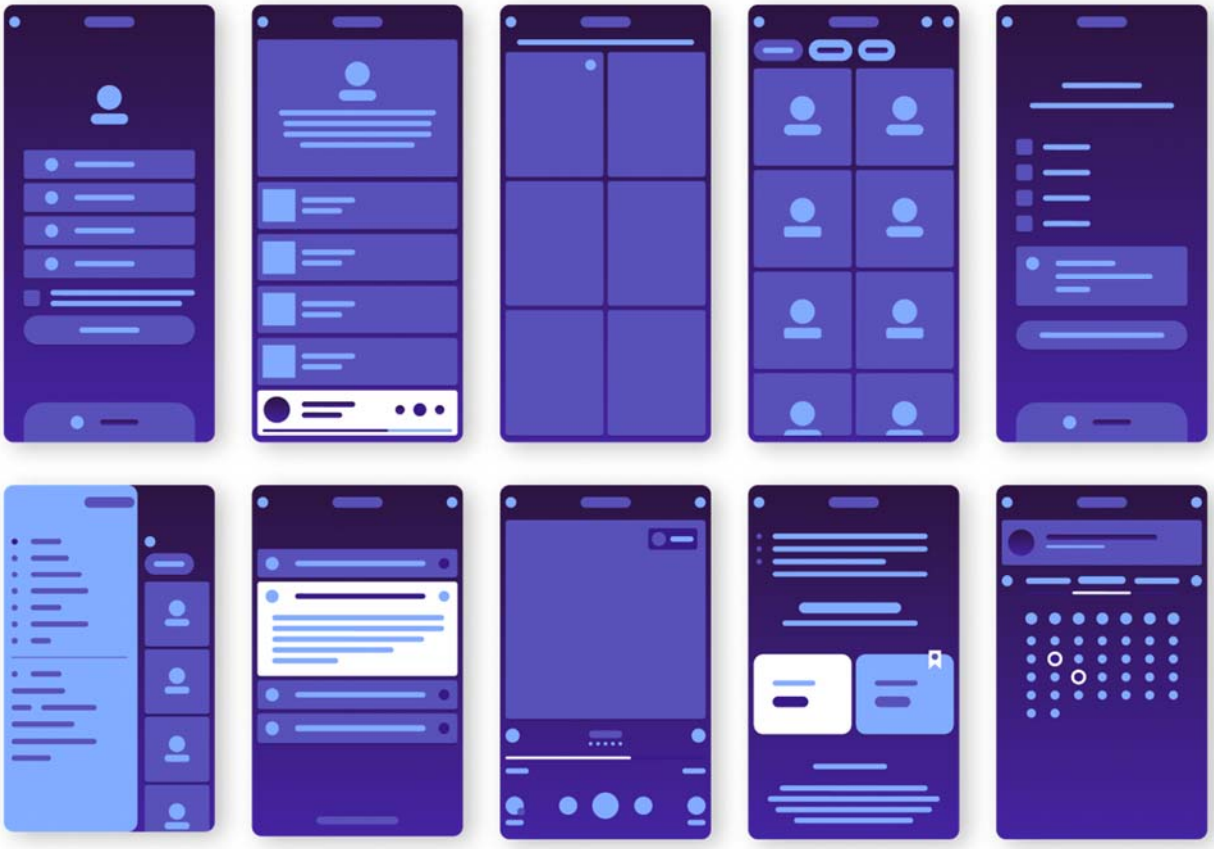

Рис.1.1. Мокап мобільної версії сайту [1]

Після розробки прототипу сайту виконується верстка дизайну, тобто розробка зовнішнього вигляду сайту. Даний етап є досить важливим оскільки впливає на перше враження користувача від сайту. При розробці дизайну необхідно розуміти основи UX, а саме принципи поведінки користувача на сайті, та його попередній досвід. Це дозволяє створити дизайн, який буде цікавий потенційному клієнту, та буде мати найвищий показник конверсії [18].

Наступним кроком є верстка коду сайту. На даному етапі створюється структура сайту за допомогою стандартизованої мови розмітки документів HTML та описуються стилі елементів сторінки за допомогою мови CSS [11], виконується адаптація сайту під мобільні пристрої, та оптимізується код [20].

Після виконання усіх вище згаданих етапів необхідно наповнити сайт контентом. Для цього, на основі наданої користувачем інформації, маркетологи пишуть тексти та підбирають необхідні медіа матеріали, такі як:

- зображення;
- відео;
- графіки.

На цьому процес розробки сайту завершений. Далі необхідно купити домене ім’я та розмістити файли сайту на сервері. Після цих дій, сайт буде опублікований в мережі інтернет [22].

Загальний порядок розробки сайту відображено на Рис.1.2.

Рис.1.2. Етапи розробки сайту.

Виходячи з наведеної інформації, можна дійти висновку, що створення сайту є досить складним та довгим процесом, який потребує спеціальних технічних знань, тому необхідно автоматизувати ці процес, щоб будь-який користувач, мав можливість створити повноцінний сайт в декілька простих дій, затративши при цьому невеликий проміжок часу.

Розроблена система дозволяє спростити вищезгаданий процес створення сайту до декількох дій, автоматизувавши процеси написання коду та створення дизайну.

На Рис.1.3 зображено необхідні дії користувача для отримання сайту в розроблені системі.

Рис.1.3. Алгоритм взаємодії користувача з системою

Система представляє собою невеликий веб-застосунок, який містить органи управління генерацією сайту [12]. Вигляд програмного забезпечення виконаний в мінімалістичному стилі та поділений на дві частини, на такі як:

- інтерфейс, де розміщується редактор тексту та поля вводу користувацької інформації;
- вікно результату, в якому буде відображатися згенерований сайт.

Оскільки дизайн відіграє дуже важливу роль при першому вражені користувача від сайту, важливим є визначення найбільш популярного стилю веб-сайту. На основі зібраної інформації зроблено висновок, що на даний момент, мінімалістичний стиль є найбільш популярним, це обумовлено тим що, сайт з таким дизайном, краще фокусує користувача на інформації, що в свою чергу покращує показники конверсії, тому в розроблену систему було закладено принципи мінімалістичного дизайну [1; 2]. На Рис.1.4 зображено приклад мінімалістичного дизайну.

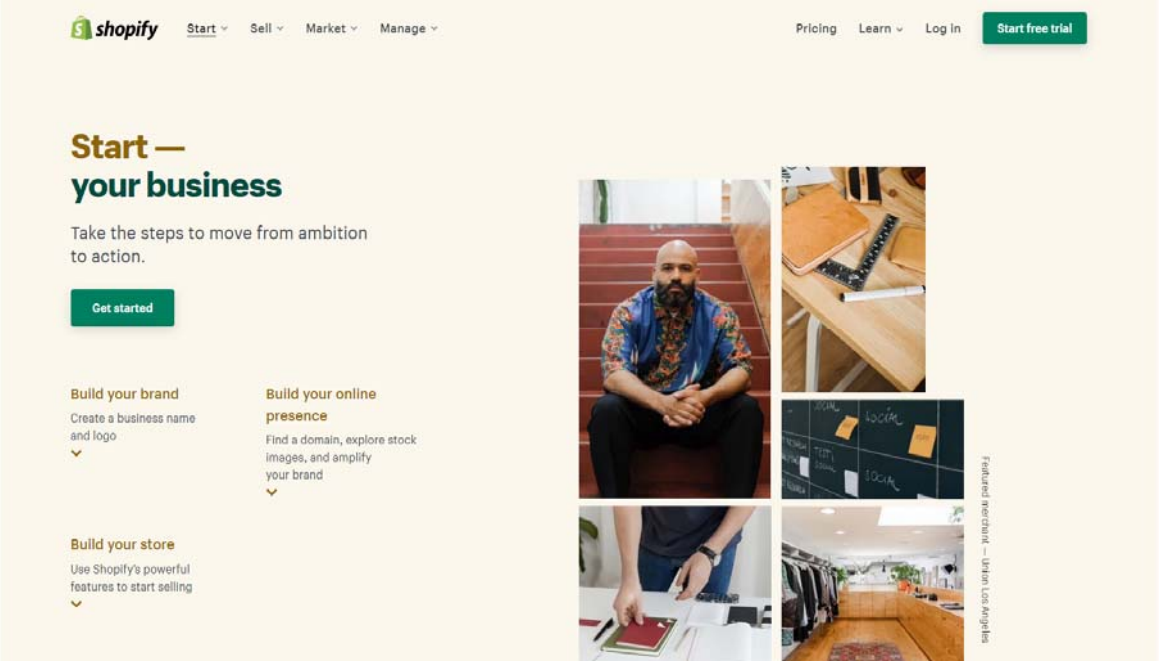

Рис.1.4. Головна сторінка Shopify [2]

Початок роботи системи завжди розпочинається з заповнення користувацьких полів, таких як: контактні дані, назва сайту тощо. Далі користувачу необхідно перенести контент в текстовий редактор, розставляючи при цьому пусті рядки, що буде сигналізувати системі про закінчення секції. Коли всі налаштування готові та контент перенесено, можна запускати генератор.

Перш ніж система згенерує структуру сторінки, варто перетворити контент, який вставив користувач на зрозумілу програмі сутність. Для вирішення цієї задачі, було використано DOM-parser [24], який дозволяє парсити HTML код та перетворювати його в масив даних. Під час виконання роботи парсера, відбувається аналіз розмірів елементів та їх запис в об'єкт даних, що дозволить в подальшому на основі отриманих даних згенерувати структуру сторінки. Згенерований парсером об'єкт необхідно перетворити в структуровані об'єкти даних, де кожний вкладений об'єкт відповідає за одну секцію сторінки [4]. Для реалізації цього розроблено алгоритм, на основі об'єктно орієнтовного програмування, який дозволяє генерувати нові екземпляри класу в циклі, та поміщати їх в об'єкт в залежності від проходження перевірок на наявність певних спеціальних значень атрибутів, які розмежовують секції та колонки сторінки [9].

За рендеринг сторінки відповідає JS бібліотека React, яка дозволяє створювати користувацькі інтерфейси, та зменшує час перетворення об'єкту даних на повноцінну HTML сторінку[13].

В результаті користувач отримує самодостатній односторінковий сайт, який відповідає усім вимогам валідності HTML коду та містить SEO мета атрибути, а також характеризується високими показниками швидкості.

Алгоритми даної системи передбачають їх подальшу модифікацію, під певні задачі, що має позитивно вплинути на подальший розвиток розробленого програмного забезпечення.

Як висновок можна стверджувати, що розроблений програмний продукт, має великий потенціал для розвитку. Характеризується зручним інтерфейсом, та високою швидкістю генерації сайту. Також в межах даної система можна реалізувати певний функціонал конструкторів сайтів, а саме перетягування блоків та можливість генерувати різні стилі дизайну на основі штучного інтелекту, що дозволить зручніше використовувати дану систему, і зробить її наступною віткою еволюції конструкторів сайтів.

1.2 Аналіз існуючих систем розробки сайтів

Розроблену систему можна віднести до категорії конструкторів сайту, але головною відмінністю та перевагою від існуючих програмних рішень є те, що користувач не бере безпосередньої участі в розробці сайту.

Конструктор сайтів – це спеціалізоване програмне забезпечення яке завдяки інтерфейсу перетягування, дозволяє в короткі терміни створити веб сайт [21]. Йому притаманні наступні характеристики:

- інтеграція зі сторонніми сервісами;
- підтримка SEO інструментів;
- велика кількість додатків, які розширюють функціональні можливості системи;
- відсутність навички написання коду.

Принцип роботи конструктора полягає в перетворенні взаємодії користувача з інтерфейсом в код. Зазвичай конструктор надає готовий набір шаблонів та функціоналу, який користувач може використати при розробці сайту [21].

Варто зазначити, що існують генератори статичних сайтів, але вони мають фундаментальні відмінності від запропонованої в даній роботі системи, оскільки використовуються в більшості програмістами для створення технічної документації, та не застосовуються в веб-розробці, тому що не мають потрібного програмного забезпечення для реалізації усіх вимог та потреб веб-сайту. Тим не менш щоб розгорнути таку систему на своїй віртуальній машині, потрібно володіти досить великим багажем технічних знань, тому звичайному користувачу не під силу це реалізувати, бо дана система не вирішує дану проблему [15]. По цій причині в межах даної роботи увага буде акцентуватися основний на конструкторах сайтів, оскільки розроблена система має на меті

перевершити показники ефективності таких систем та автоматизувати процес створення сайтів.

Основною перевагою конструктора над стандартною верстkою сайту є те, що сайт можна створити в досить короткі терміни та без написання коду. Сайт буде адаптований під різні пристрої, та матиме базовий набір функціоналу, такий як:

- контактна форма;
- модальне вікно;
- відео плеєр, тощо;

Але звичайний пересічний користувач не зможе створити сайт на конструкторі, оскільки не володіє базовими знаннями веб-розробки.

Також основним недоліком конструктора сайту є те, що DOM сторінки є дуже великим, через надміру кількість атрибутів та класів HTML елементів. Принцип роботи конструктора побудований так, що він підключає на сторінку велику кількість ресурсів, таких як CSS стилі та JS скріпти, які можуть не використовуватися, що в свою чергу дуже сильно впливає на показник швидкодії сайту [21].

За рекомендацією Google веб-сайт повинен завантажуватися менші ніж за 1 секунду, якщо сторінка завантажуватиметься довше, тоді зростає вірогідність що користувач покине сайт, не дочекавшись завантаження [14].

Основними вимогами Google до веб сайтів є:

- уникнення блокування рендерингу сторінки JS або CSS ресурсами;
- завантаження лише тих ресурсів, які використовуються на даній сторінці;
- вбудовування критичного CSS в код сторінки, оскільки це дозволить позбутися блокування візуалізації;

- об’єднання великої кількісті ресурсів в один файл. Виконавши дану вимогу можна зменшити час очікування завантаження сторінки, та кількість запитів до сервера.

Варто виділити основні переваги та недоліки найпопулярніших представників конструкторів сайтів, а саме: Weebly, GoDaddy, IONOS By 1&1, Wix, Squarespace, HostGator.

Weebly тривалий час займає перші місця як один з найкращих інструментів для створення сайту, завдяки низькій вартості, простоті використання та гнучкості. Зазвичай його використовують для створення комерційних сайтів, таких як інтернет-магазин або сайт-каталог. Дана система надає можливість змінити шаблон сайту, зберігши при цьому вміст користувача [3].

На Рис.1.5. зображено головну сторінку Weebly.

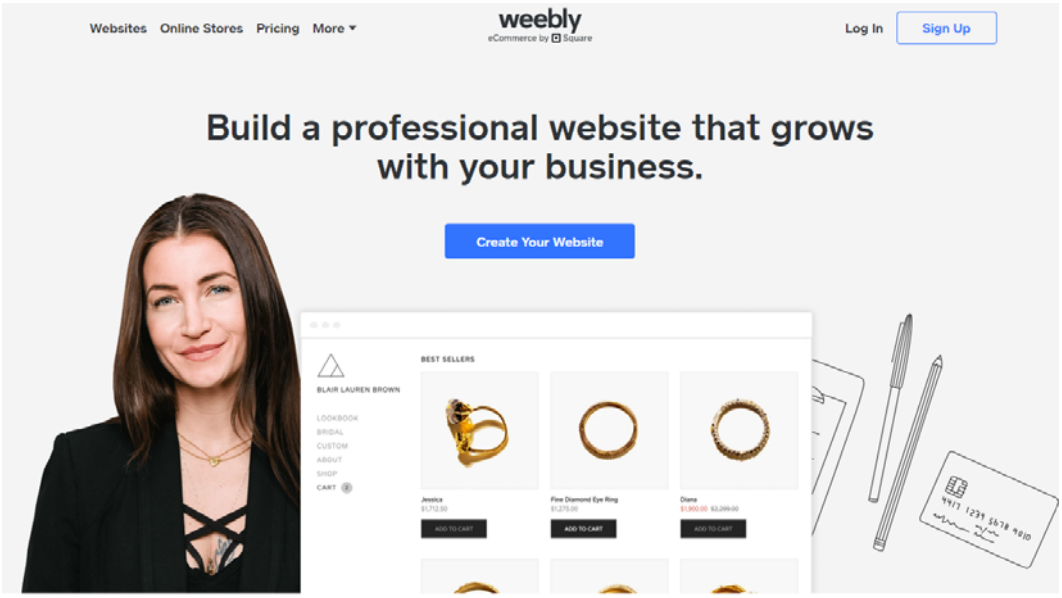

Рис.1.5. Головна сторінка сайту Weebly [3]

Переваги:

- легкий у використанні;
- велика кількість безкоштовних шаблонів;

- невелика вартість підписки;
- більше 300 додатків;
- необмежене сховище даних.

Недоліки:

- обмежений функціонал налаштувань сайту;
- неповна чат та телефонна підтримка.

Конструктор від компанії GoDaddy дозволяє на базі штучного інтелекту створювати шаблон вашого сайту на основні вподобань користувача. Оскільки GoDaddy також надає послуги реєстратора доменів та веб-хостингу, користувач досить легко може розмістити сайт створений на конструкторі в мережі інтернет [3].

На Рис.1.6. зображено головну сторінку GoDaddy.

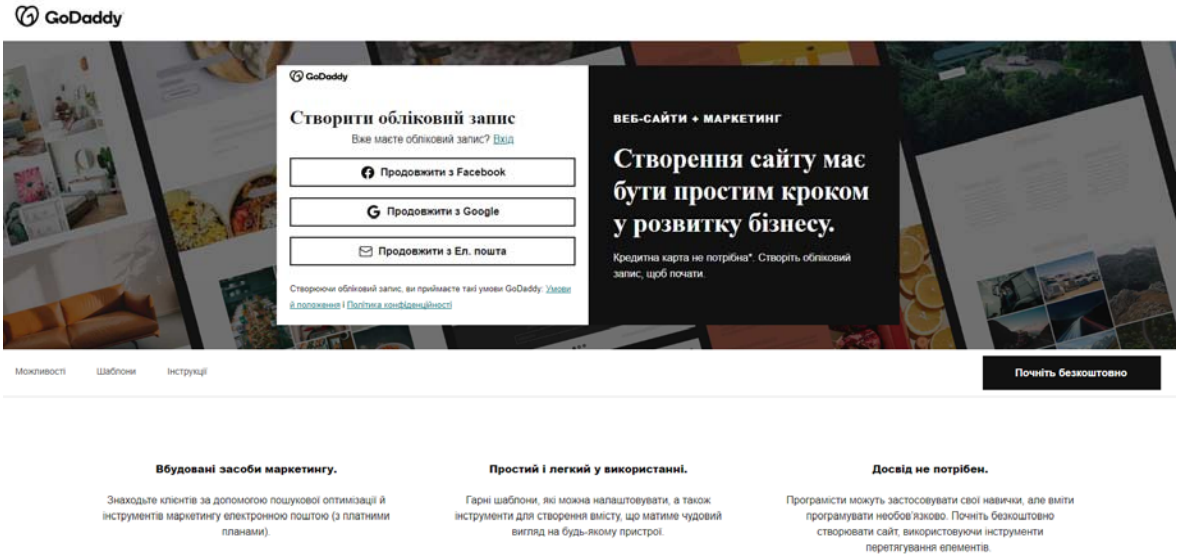

Рис.1.6. Головна сторінка GoDaddy [3]

Переваги:

- для генерації дизайну використовується штучний інтелект;
- підтримка клієнтів 24/7;

- доступність тарифних планів конструктора.

Недоліки:

- додаткові застосунки для сайту, є досить дорогими;
- обмежене налаштування сайту;
- обмежена кількість додатків;
- висока вартість технічної підтримки.

Головною особливістю конструктора IONOS By 1&1, яка відрізняє його від інших конструктів є те, що для різних груп користувачів можна персоналізувати вміст сторінки. Це дозволяє розширити аудиторію сайту, та конкретизувати інформацію для кожного з відвідувачів [3].

На Рис.1.7. зображено головну сторінку IONOS By 1&1.

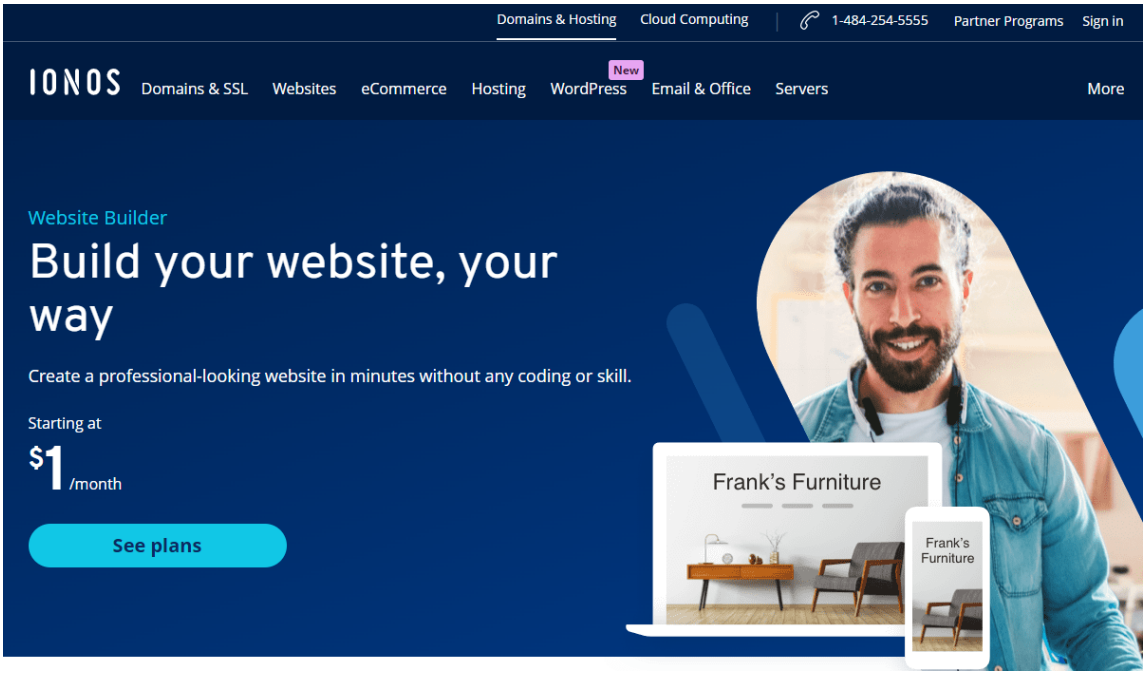

Рис.1.7. Головна сторінка IONOS By 1&1 [3]

Переваги:

- підтримує мультимовність;
- хороші відгуки користувачів;

- цілодобова підтримка;
- персоналізація вмісту сторінки;
- невелика вартість щомісячної підписки.

Недоліки:

- реклама конструктора відображається на сайті;
- застарілі шаблони.

Wix, мабуть один з найпопулярніших конструкторів сайтів в країнах СНГ. Використовується для створення односторінкових сайтів. Має безкоштовний тариф, який дозволяє розміщувати сайт на під-домені Wix, але через це на сайті буде відображатися реклама сайту wix.com [3].

На Рис.1.8. зображено головну сторінку WIX.

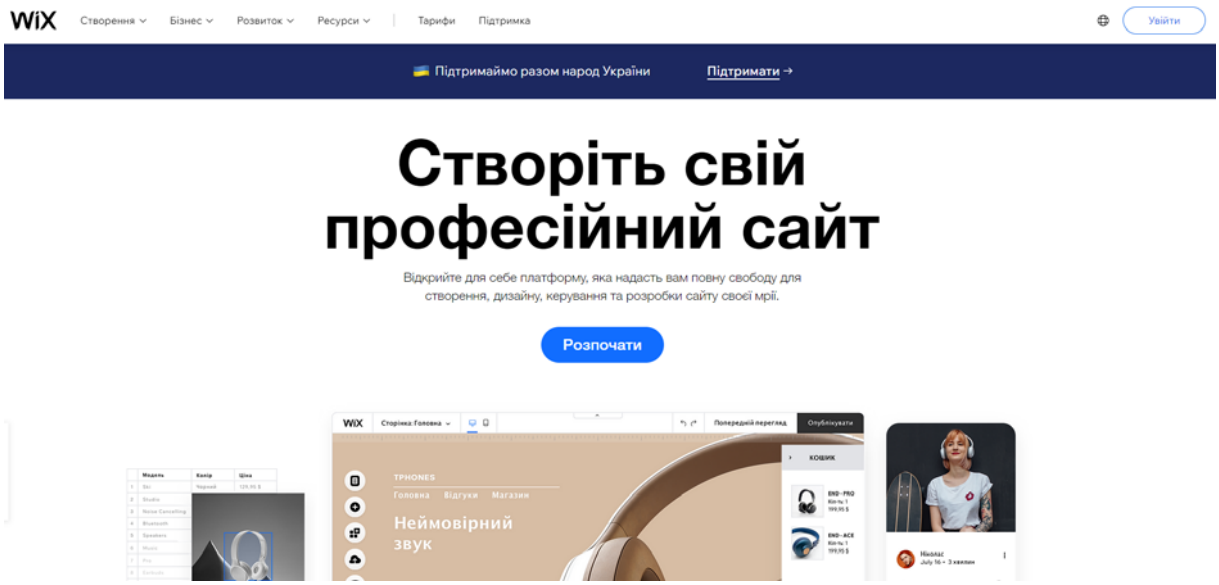

Рис.1.8. Головна сторінка WIX [3]

До основних переваг можна віднести:

- безкоштовний тарифний план;
- велика кількість готових шаблонів;
- простий у використані.

Недоліки:

- обмежений термін зберігання сайту;
- відсутні інструменти SEO, що унеможливлює просування сайту;
- на сайті відображається реклама сайту wix.com.

Squarespace не одноразово був номінований на нагороду за найкращий дизайн, завдяки свої шаблонам, які розроблені професійними веб-дизайнерами. Також завдяки вбудовані технології AMP, створені сайти мають високі показники швидкості, що позитивно впиває на просування сайту в пошуковій видачі [3].

На Рис.1.9. зображено головну сторінку Squarespace.

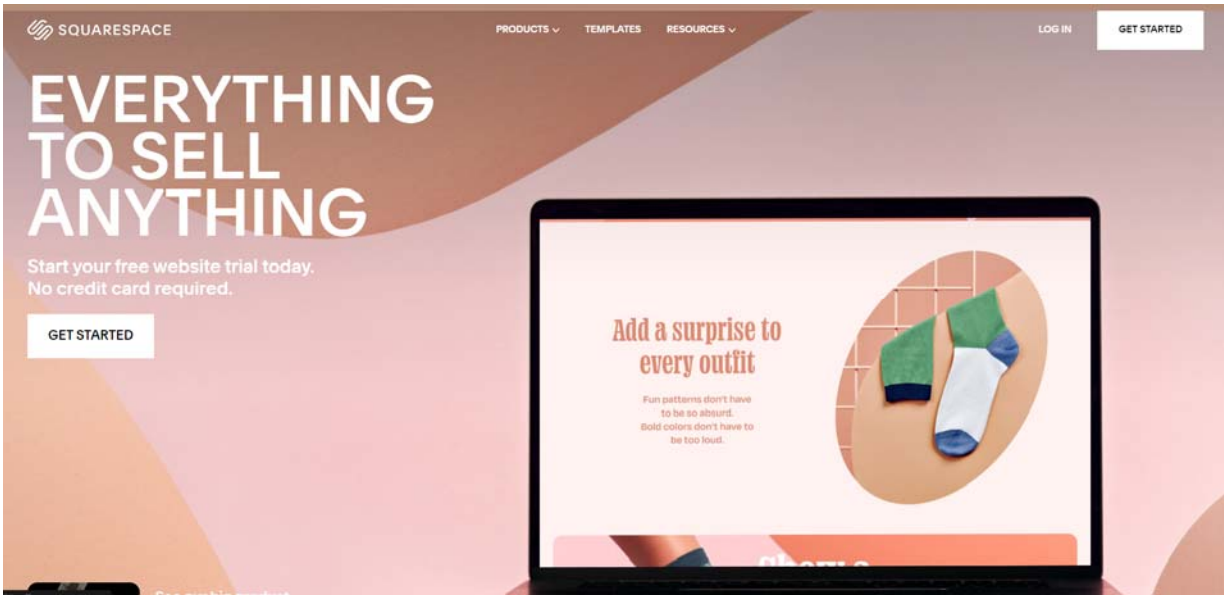

Рис.1.9. Головна сторінка Squarespace [3]

До основних переваг можна віднести:

- сучасний дизайн;
- простий у використанні;
- доступна технологія AMP;
- велика кількість додатків.

Недоліки:

- обмежена підтримка;
- обмежені налаштування шаблонів;
- висока вартість використання в порівнянні з іншими конструкторами.

HostGator конструктор має зрозумілий інтерфейс та доступні ціни. Найдешевший тариф підтримує електрону комерцію, проте є певні обмеження, які виражаються в кількості товарів, що можна опублікувати. Даний конструктор підходить для невеликих компаній, оскільки надає обмежений функціонал розробки сайту [3].

На Рис.1.10. зображено головну сторінку HostGator.

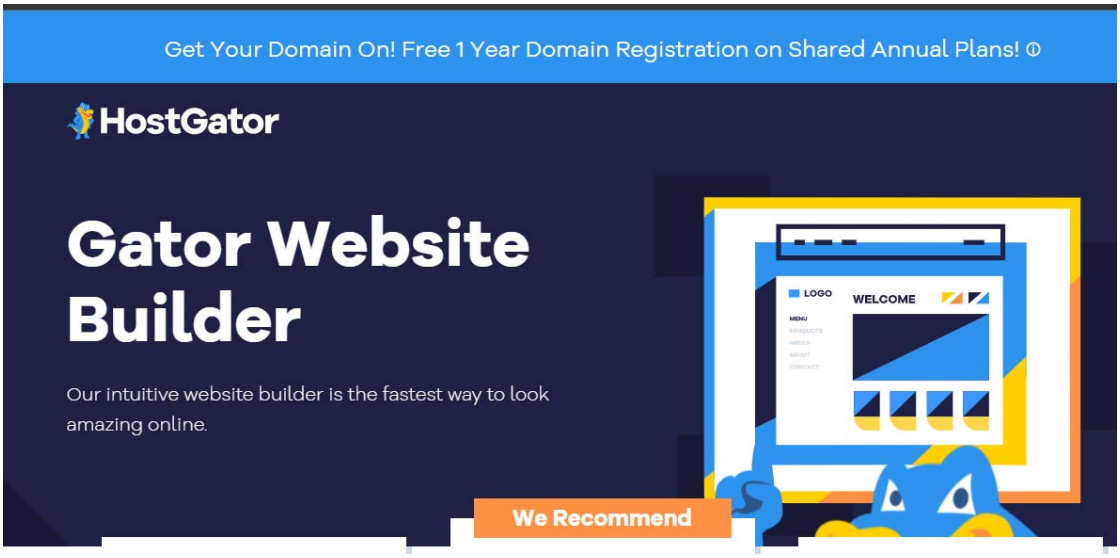

Рис.1.10. Головна сторінка HostGator [3]

Переваги:

- цілодобова підтримка;
- доступні ціни;
- простий у використанні.

Недоліки:

- велика комісія;

- відсутній функціонал бронювання;
- застарілий дизайн;
- обмежені налаштування шаблону.

Основним недоліком усіх описаних конструкторів, в порівняні з розробленою системою, є те що, без знань роботи з конструктором, користувач не зможе самостійно створити сайт.

Аналізуючи ринок веб-застосунків, які дозволяють створювати сайти, можна дійти висновку, що розроблена система є унікальною в своєму роді, та немає аналогів, оскільки створює сайт в автоматичному режимі, та не потребує спеціальних технічних знань.

Звичайно як і будь яке програмне рішення, запропонована в даній роботі система теж має свої переваги та недоліки.

На Рис.1.11. показано основні переваги та недоліки системи.

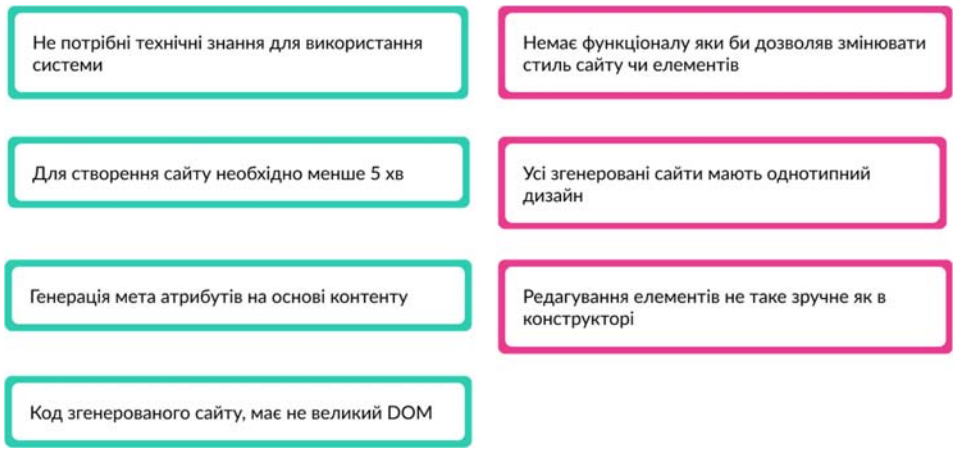

Рис.1.11. – Переваги та недоліки розробленої системи

Для встановлення системи достатньо розархівувати код програмного забезпечення в директиву, після чого система готова до використання.

Алгоритм програмного забезпечення написаний на JS з використанням бібліотеки React, що дозволяє запускати програму в будь якому браузері

Код який генерує система, має мінімум тегів та атрибутів, що робить сторінку легкою, та з невеликим DOM.

Для використання системи користувачу не потрібні спеціальні технічні знання в сфері розробки сайтів.

Система генерує мета атрибути на основі контенту, це дозволить в подальшому робити SEO просування сайту в пошуковій видачі [23].

За допомогою даного програмного забезпечення можна генерувати багато-сторінкові сайти, це розширює спектр використання системи.

Сайти створюються в мінімалістичному дизайні з дотриманням усіх правил валідності та семантики коду.

Функціонал даної системи можна досить легко розширювати завдяки використанню принципів об’єктно орієнтованого програмування в її основі.

Проте, дана система не дозволяє перетягувати блоки як це роблять конструктори сайтів. Це можна реалізувати за допомогою користувацьких інтерфейсів React [13].

Блоки виводяться в порядку того, як вони були задані в редактор, тобто якщо потрібно змінити розташування контенту, користувачу потрібно змінити його положення в текстовому редакторі, це не так зручно як в конструкторах.

Дана система не має функціоналу який дозволяє більш детальніше модифікувати кожний блок сторінки окремо. Технічних обмежень для реалізації такого функціоналу немає.

Система генерує однотипні сайті. Стиль згенерованих сайтів буде однаковий, оскільки закладено один стиль. Даний функціонал можна розширити та надати можливість користувачу вибирати стиль сайту, або впровадити штучний інтелект який буде генерувати стиль сайту на основі вподобань користувача, як це реалізовано в конструкторі Godaddy [3].

Виходячи з описаних переваг та недоліків, можна зробити висновок що дана система має досить великий потенціал для подальшого розвитку.

Впровадження штучного інтелекту, розширення функціоналу дозволить покращити вихідні результати. Технічних обмежень для її вдосконалення немає. Також дана система є хорошим інструментом для розробки сайтів, як для звичайного користувача, так і допоміжним інструментом для програміста, якому потрібно швидко згенерувати HTML сторінку, яку в подальшому можна модифікувати відповідно до поставленої задачі.

1.3 Постановка задачі дослідження

З вищенаведеної інформації, стає зрозумілим, що необхідно:

- автоматизувати процес створення сайту, задля збереження економічної вигоди та часу клієнта. Система повинна видавати результат в декілька простих дій, що дозволить користувачеві, без технічних знань, створити сайт;
- врахувати можливість подальшого розширення функціоналу при розробці програмного забезпечення;
- розробити алгоритми рендерингу згенерованого сайту, з сформованого об’єкту даних.

Головною задачею буде реалізація методу конвертування текстових елементів в валідний HTML код. Також необхідно розробити текстовий редактор, який дозволятиме користувачу задавати відповідні параметри текстовими елементам, на основі яких буде генерувати об’єкт даних. Даний об’єкт даних повинен бути структурований, та містити інформацію про кожний елемент.

Для системи необхідно розробити алгоритм аналізу розмірів контенту, на основі якого буде реалізовуватися структура сторінки. Даний алгоритм, повинен розрізняти типи даних та розширювати спектр підтримуваних HTML тегів.

Інтерфейс системи повинен бути інтуїтивно зрозумілий та зручний у використанні для звичайного користувача.

Результатом даної системи повинен бути файл, який буде містити усі необхідні для коректної роботи сайту складові.

Висновки до розділу 1

1. Проведено аналіз розробленої системи;
2. Виконано порівняльний аналіз сучасних системи розробки сайтів;
3. Визначено основі процеси які потрібно автоматизувати, для пришвидшення розробки сайту;
4. Описано основні алгоритми збору та аналізу поданих користувачем даних, та їх подальшої обробки;
5. Визначено завдання дослідження для спрощення процесу розробки нескладних HTML сайтів.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Обґрунтування вибору методу аналізу даних

Одним з основних алгоритмі даної системи, являється метод аналізу даних, який застосовується під час роботи парсера DOM-дерева сторінки [19]. Парсер зчитує крок за кроком подану йому структуру даних, аналізує її та в результаті створює абстрактне синтаксичне дерево, яке необхідно перетворити на зрозумілу системі сутність [8]. Для цього розроблено методи аналізу контенту які виконуються в ході роботи парсера [24].

На Рис.2.1 зображено принцип роботи парсера JavaScript:

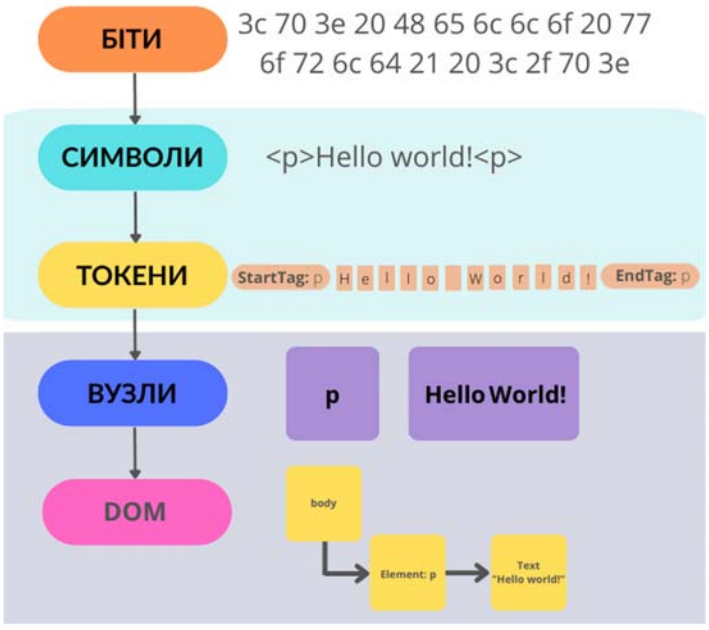

Рис.2.1. Алгоритм роботи парсера JavaScript [24]

Під час роботи парсера, першим викликається метод визначення типу поточного елемента. Його функціоналі можливості базуються на роботі оператора строгого порівняння в поєднанні з оператором “логічне або”, де

властивість .tagName поточного елемента порівнюється з підтримуваними системою тегами, та записує значення в відповідну властивість об’єкта.

Розроблена система підтримує такі типи HTML тегів:

- теги заголовків;
- параграф;
- зображення.

Для того щоб розбити сторінку на секції необхідно визначити чи попередній елемент був пустим, для цього в поточного елемента викликається метод .innerText.length, який повертає кількість символів. Якщо кількість елементів рівна нулю, тоді в параметр closeDiv об’єкта записується closeCol, якщо попередній елемент теж пустий, тоді записується значення closeRow.

На рисунку 2.2 зображеного схематичний вигляд об’єкта згенерованого парсером.

```
Object {
  0: {
      tag: 'h1',
      content: 'Text',
      closeDiv: null
  }
  1: {
      tag: 'p',
      content: 'Paragraph content',
      closeDiv: null
  }
  2: {
      tag: null,
      content: null,
      closeDiv: 'closeCol'
  }
  3: {
      tag: null,
      content: null,
      closeDiv: 'closeRow'
  }
  ....
}
```

Рис.2.2. Згенерований парсером об’єкт даних

Отриманий об’єкт даних, необхідно розбити на декілька об’єктів, кожен з яких буде відповідати за окрему секцію сторінки. Тому за допомогою циклів проітеруємо наш об’єкт. Під час виконання ітерації, проводиться аналіз розмірів поточного елемента за нижче описаними алгоритмами.

Для перевірки розміру текстового елементу, використовується наступний алгоритм:

1. перевіряємо тип поточного елемента;
2. за допомогою методу .length, отримуємо кількість символі текстового елемента;
3. виконуємо операцію обрахунку, де першим операндом слугує кількість символів елемента, а другим певне визначене число для кожного з підтримуваних системою тегів;
4. за допомогою функції parseInt() перетворюємо значення з плаваючою крапкою, на цілочисельне число з типом Integer.

Отримане значення буде сигналізувати системі про те скільки колонок Boostrap даний елемент займе на сторінці.

Оскільки JavaScript не може визначити розміри зображення по його посиланню, потрібно для початку добавити даний елемент в потік, тому було розроблено наступний алгоритм визначення розміру [9]:

1. перевіряємо тип поточного елемента;
2. завдяки new Image() створюємо новий екземпляр HTMLImageElement, та додаємо його на сторінку;
3. за допомогою методу addEventListener("load") очікуємо поки завантажиться зображення;
4. після завантаження зображення для контексту даного методу викликаємо .naturalWidth;
5. далі отримане значення передається в функцію визначення розмірів елемента.

Розміри елементів записуються в об’єкт даних у відповідну властивість. Далі визначається найбільший розмір елементу в колонці, завдяки методу, який перевіряє розмір поточного та попереднього елемента, якщо поточне значення більше ніж попереднє, тоді при закінченні ітерації колонки, записуємо дане значення в властивість size об’єкта колонки.

При проходженні циклу, метод перевіряє наявність closeCol або closeRow, якщо дана перевірка є істиною, то поточний об’єкт закривається, і створюється новий екземпляр об’єкту, таким чином в результаті генерується об’єкт який містить об’єкти рядів які в собі містять об’єкти колонок. Вигляд такого об’єкту зображений на Рис.2.3.

```
Object {
  Row 1 object : {
    Col 1 object: {
      Element object: {
        tag: 'h1',
        content: 'Text',
      },
      Element object: {
        tag: 'p',
        content: 'Text',
      },
      size: 6
    },
    Col 2 object: {
      Element object: {
        tag: 'h2',
        content: 'Text',
      },
      Element object: {
        tag: 'p',
        content: 'Text',
      },
      size: 6
    }
  },
  ......
}
```

Рис.2.3. Структурований об’єкт даних

Даний об’єкт згенерує HTML код в вигляді двох колонок розміром col-6. В лівій колонці буде заголовок h1 та невеликий параграф, в правій колонці буде заголовок рівня h2 та параграф.

2.2. Розробка архітектури програмної системи

Архітектура розробленої системи, складається зовнішніх та локальних файлів, які необхідні для роботи програмного забезпечення. Дотримуючись стандартів розробки програмного забезпечення, JavaScript код розділявся на файли за логікою роботи [17]. Оскільки від порядку підключення залежить коректність роботи системи, тому ресурси які відповідають за візуальну частину програмного забезпечення підключаються в головну частину сторінки <head>. Файли які відповідають за функціональні можливості додатку асинхронно підключаються в нижній частині сторінки <footer>. Це потрібно для того щоб, мінімізувати затримку відображення [14]. На Рис.2.4. зображено схему виконання локальних та зовнішніх ресурсів програмного забезпечення.

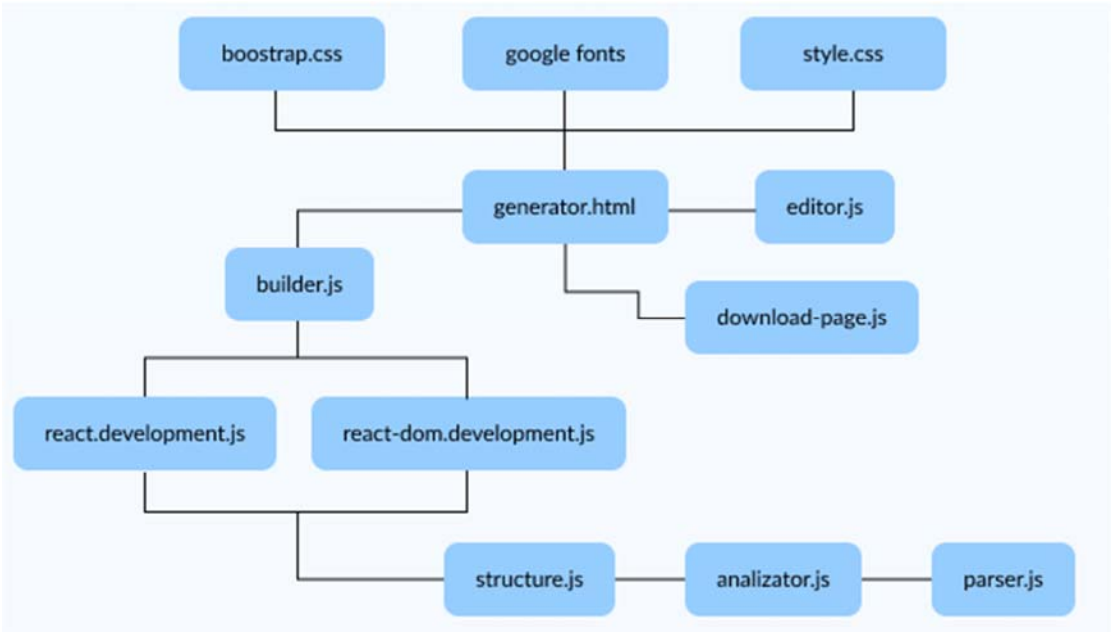

Рис.2.4. Схема ресурсів системи

При розробці програмного забезпечення використовувалися принципи функціонального програмування які включать в себе:

- використання чистих функцій;
- уникнення мутації даних;
- не використання shared state;
- уникнення side-effects.

Завдяки дотриманню даних парадигм програмування, реалізований код системи, відповідає останнім стандартам ECMASript 11.

ECMASript 11 – це стандартизація JS коду, яка була розроблена для того щоб уникнути розбіжності між різними версіями даної мови програмування [16].

Огляд функціоналу архітектури розпочнемо з файлу editor.js, оскільки він спрацьовує першим, після будь якої дії користувача з редактором. Даний файл відповідає за функціонал редактора, який дозволяє форматувати виділені елемент контенту.

Алгоритм роботи редактора полягає в виділені, та застосуванню формування по змінні властивостей активного елемента.

Для розробки вищезгаданого функціоналу в області редактора, для HTML був включений режим редагування, що надає можливість форматувати текстові елемент. Даний режим вмикається за допомогою властивості document.designMode, що дає можливість використовувати функцію execCommand(), яка як аргументи приймає параметри редагування елементів в середині обраної області [9].

Коли користувач заповнив відповідні поля, і натиснув ‘Згенерувати’, першим виконується файл parser.js, який за допомогою метода .parseFromString() зчитує текстові елементи з редактора, після чого пропускає зібраний масив даних через цикл, в якому виконується перевірка типу елемента, та його розмірів [24]. Під час кожної ітерації, відбувається мапування даних в один об’єкт. Даний об’єкт містить інформацію про елементи, межі колонок, та рядів.

Алгоритм роботи parser.js показано на рисунку 2.5

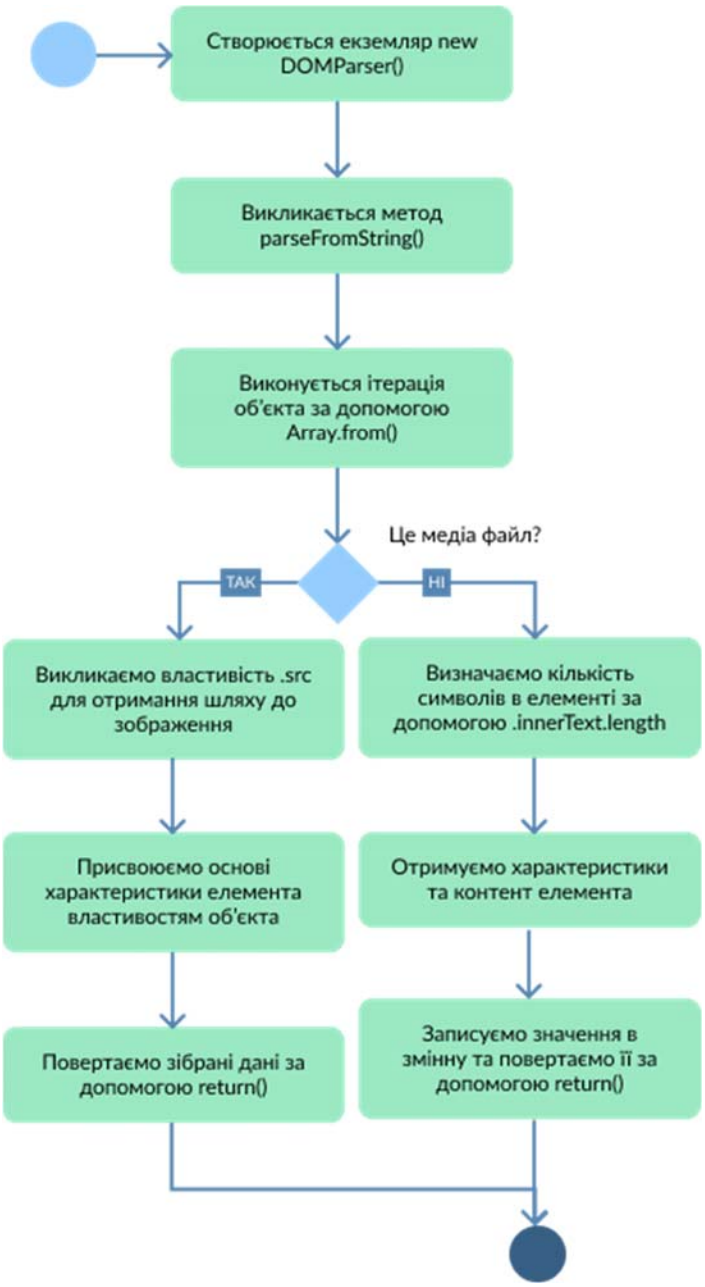

Рис.2.5. Алгоритм роботи функції parser()

Сформований парсером об’єкт передається в файл structure.js, який за допомогою файлу analizator.js визначає розміри елемента та формує структуру сторінки. Для створеннях нових екземплярів об’єкта необхідно створити Class з відповідними властивостями та методом.

На рисунку 2.7 зображено алгоритм класу CreateElementClass.

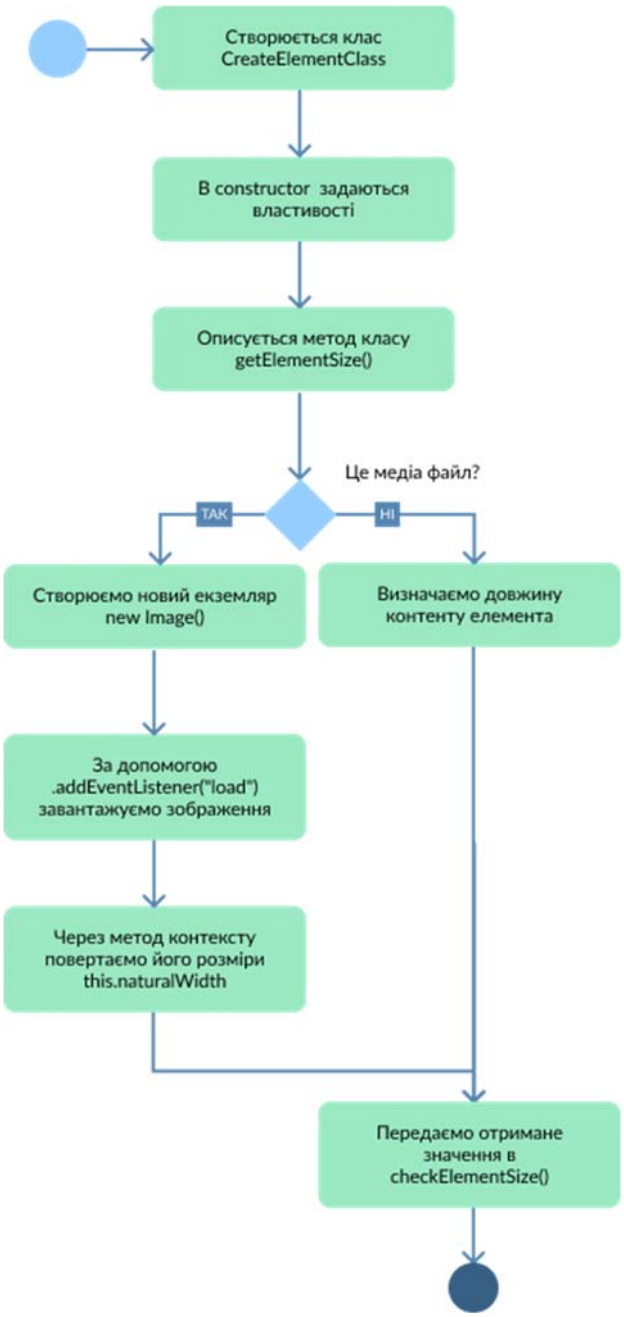

Рис.2.7. Алгоритм роботи класу CreateElementClass

Даний клас містить метод getElementSize(), який на основі переданих аргументів, а саме контенту та тегу, повертає кількість колонок які займає елемент на сторінці, це необхідно для формування сітки сторінки. Це дозволяє

виконувати виклик методу в нового екземпляра об’єкта та отримувати його розміри.

Вищезгаданий клас необхідний для функції createPageStructure(), алгоритм якого описаний на рисунку 2.8

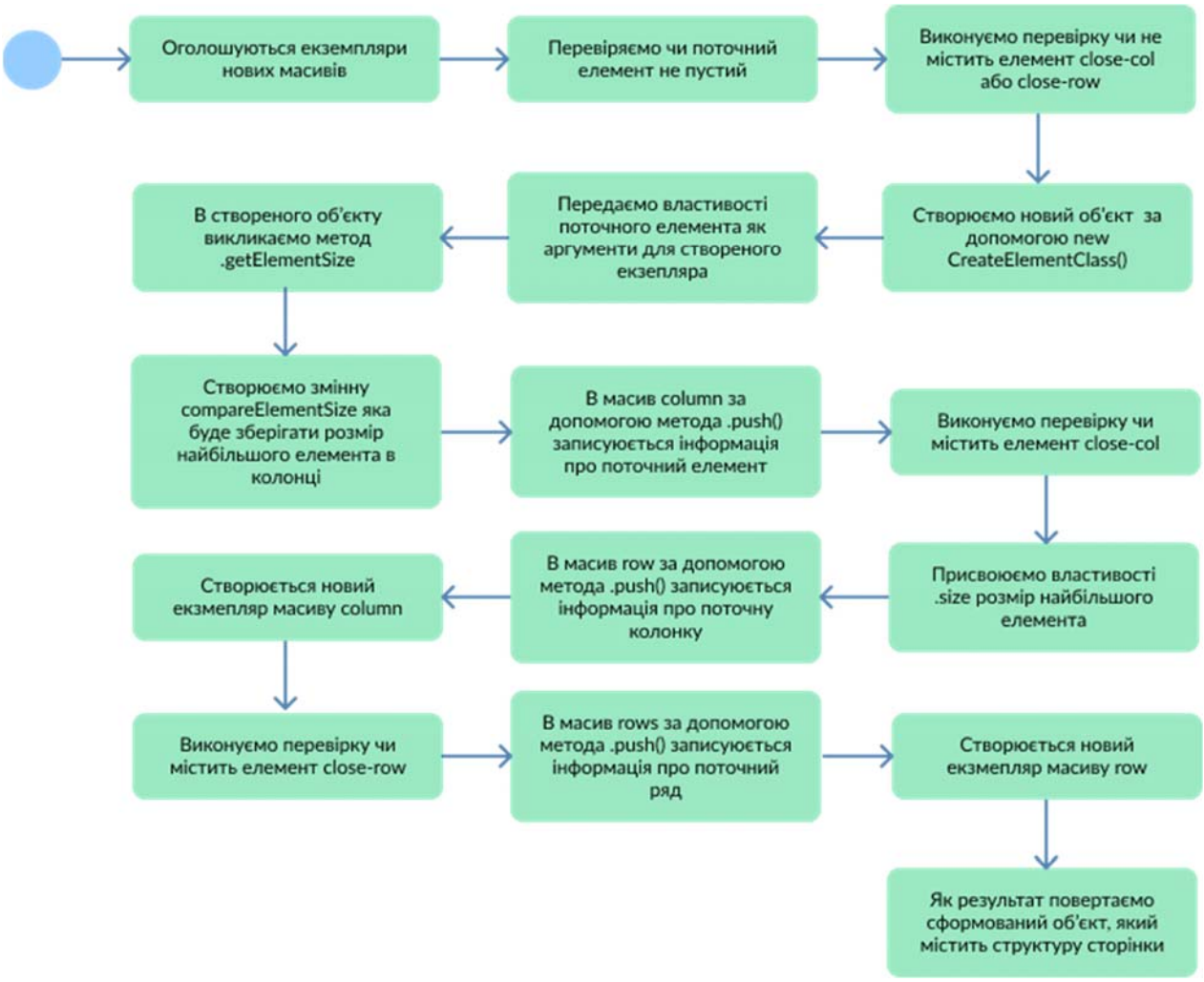

Рис.2.8. Алгоритм роботи фунції createPageStructure()

Описана функція, на основі переданих аргументів, виконує фільтрацію отриманого об’єкта в результаті чого генерується структурований об’єкт даних. Реалізований об’єкт схематично відображає згенеровану структуру сайту, та містить усю передану інформацію про елементи а саме:

- кількість колонок яку займає елемент;

- тег елемента;
- контент елемента.

Завершальним етапом роботи генератора є рендеринг отриманого структурованого об’єкта даних, за допомогою бібліотеки React. За це відповідає файл builder.js, який містить в собі ряд функції, які дозволяють створювати компоненти та виконувати рендеринг отриманого контенту [13]. До таких функції можна віднести:

- getContentElements() – завдяки селектору та методу .innerHTML отримує вміст редактора;
- React.createElement() – створює компонент, як аргументи приймає назву тегу та вміст який він повинен містити;
- getElement() – повертає HTML код елемента та визначає тег переданого контенту;
- getElements() – формує масив елементів колонки та створює компонент;
- getColumns() – реалізує компонент колонок та запис елементів в масив рядів;
- getRows() – описує добавлення масиву ряду в масив рядів та формує компонен ряду;
- getHead() – як результат повертає компоненти <head>, мета тегів та ресурсів сайту;
- getBody() – повертає згенеровану структуру видимої користувачеві частини сайту, також містить виклик getHead() і getfooter();
- getPage() – збирає усі компоненти в одby основний компонент, який буде містити усі необхідні для роботи сайту властивості;
- root.render() – виконує рендеринг сторінки в відведене поле.

Виходячи з описаних функції можна дійти висновку що, завдяки такій сегментації коду зберігається можливість вносити певні зміни в кожний з етапів генерації сторінки, що в свою чергу розширює функціональні можливості застосування системи та перспективи її подальшого розвитку.

На Рис.2.9 зображено метод рендерингу згенерованої сторінки.

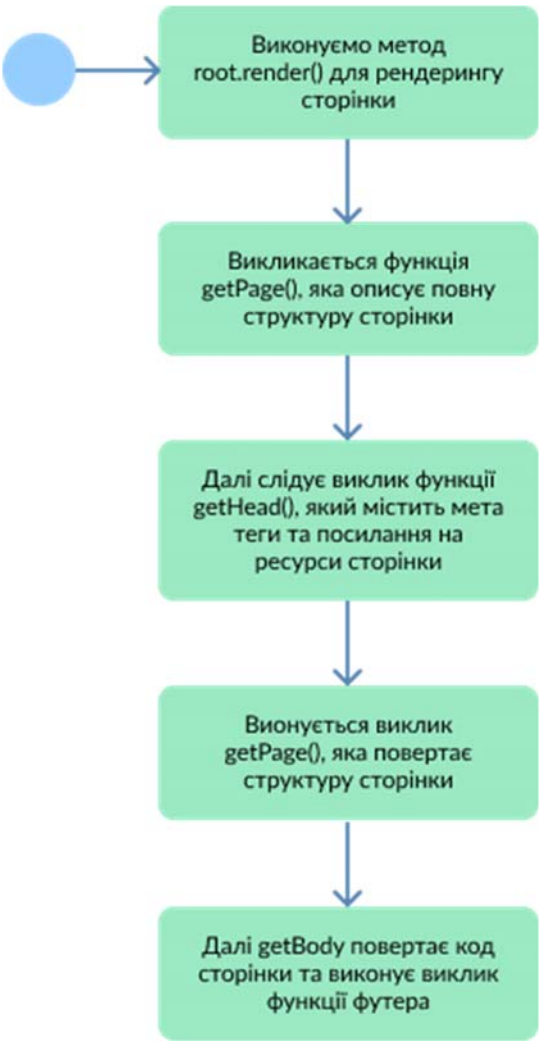

Рис.2.9. Алгоритм роботи createPageStructure

Для кожного елемента згенерованого об’єкта даних, створюються компоненти на основі їх властивостей, це дозволяє більш гнучно контролювати усі процеси рендренгу сторінки.

Перед виконанням рендерингу усі сформовані компоненти збираються в один основний компонент, який за допомогою root.render() буде вбудовано в відповідне вікно генератора, де користувач зможе переглянути отриманий результат.

За збереження отриманого результату, відповідає файл download.js. Даний файл має функцію download(), яка як аргумент отримує назву файлу та контент, який необхідно помістити в файл. Алгоритм описаного файлу показано на рисунку 2.10.

Рис.2.10. Алгоритм роботи download.js

Висновки до розділу 2

1. Спроектовано системі зв’язки між компонентами програмного забезпечення.
2. Розроблено алгоритм аналізу переданого користувачем вмісту на основі його типу та властивостей.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ І ДОСЛІДЖЕННЯ ЗАСТОСУНКА

3.1 Реалізація системної частини системи

Для розробки, описаної в даній магістерській роботі системи, використовувалися актуальні на даний момент технології JavaScript, React, Microsoft Visual Studio та Bootstrap.

JavaScript – це скриптова мова програмування, яка створювалася для виконання не складних сценаріїв. Проте розвиток даної технології показав її великий потенціал, що дозволило використовувати дану мову програмування як для складних високо навантажених проектів, так і для простих веб додатків [9].

React – це JavaScript бібліотека яка дозволяє створювати складні користувацькі інтерфейси на основі інкапсульованих компонентів. Дана бібліотека є досить гнучкою у використанні, що дозволяє застосовувати її як основу для різного роду веб-додатків. В розробленій системі, React використовується для рендерингу згенерованої сторінки [16].

Зовнішній вигляд редактора та згенерованого сайту, побудований на основі CSS бібліотеки Bootstrap. Основною ідеєю Bootstrap є те що вона містить готовий набір класів, які мають певні визначені властивості та стилі [6].

Вищезгадана бібліотека дозволяє пришвидшити розробку сайтів, оскільки надає готовий набір класів. Дану бібліотеку можна підключити як зі стороннього сервера, так і локально.

Для перетворення текстових елементів в HTML компоненти, було розроблено текстовий редактор. Алгоритм роботи редактора полягає в відстеженні виділених елементів, та застосуванню до них функції, які огортають елемент в обраний користувачем тег.

Реалізація даної системи розпочинається з оголошення змінних:

```
let optionsButtons = document.querySelectorAll(".option-button");
let advancedOptionButton = document.querySelectorAll(".adv-option-button");
let writingArea = document.getElementById("text-input");
```

Для модифікації тексту, було розроблено функцію modifyText() яка застосовує визначені методи для виділеного тексту. Код modifyText() описаний нижче:

```
const modifyText = (command, defaultUi, value) => {
  document.execCommand(command, defaultUi, value);
};
```

Для пересуванню по шкалі змін контенту в редакторі використовується відстеження кліку користувача за допомогою addEventListener() в тілі якого знаходиться вищеописана функція modifyText(). Код описаного методу:

```
optionsButtons.forEach((button) => {
   button.addEventListener("click", () => {
      modifyText(button.id, false, null);
   });
});
```

Щоб виконати модифікацію виділеного елемента, було використано метод перебору масиву доступних html компонентів, та відстеження кліку за допомогою addEventListener():

```
advancedOptionButton.forEach((button) => {
   button.addEventListener("change", () => {
      modifyText(button.id, false, button.value);
   });
});
```

За допомогою представленого коду вдалося реалізувати текстовий редактор, який перетворює текстовий контент на html елементи.

Далі перейдемо до огляду системи генерації структури сайту.

Розгляд алгоритмів системи розпочнемо з функції парсера, який зчитує передану інформацію, та обробляє її в залежності від проходження перевірок. Для кращого розуміння принципу роботи парсера розглянемо алгоритм його роботи.

Спочатку оголошується новий екземпляр DOMParser(), що дозволить працювати з потоком сторінки [5]. Після чого за допомогою методу .parseFromString() виконуються парсинг html елементів з переданого в аргумент контенту. Далі в створений екземпляр DOM за допомогою селектора отримуємо body та присвоюємо його відповідній змінній. Для змінних присвоєно тип const оскільки вони не будуть змінювати своє значення:

```
const parser = new DOMParser();
const DOM = parser.parseFromString(content, "text/html");
const body = DOM.querySelector("body");
```

Виконується декларація змінних, та присвоюється тип let, тому що вони будуть змінювати своє значення в ході виконання функції:

```
let lastOpenedTag = null;
let previousecloseDiv = false;
```

Отриманий масив даних необхідно перетворити з html елементів в об’єкти, тому для цього використано цикл Array.from(), де в ітеруємий об’єкт передаються дочірні елементи body. Завдяки flatMap повертаємо новий масив з елементів які будуть ітеруватися:

```
const json = Array.from(body.children).flatMap((element) => {
```

Далі виконується оголошення змінних, які відповідають за ім’я елемента, його розміри та контент:

```
let tag = element.tagName.toLowerCase();
let src = element.src ?? element.firstChild.src;
let elementSize = element.innerText.length;
```

```
let closeDiv = checkDivIsClose(elementSize, previousecloseDiv);
```

Для властивості tag поточного елемента буде присвоєно результат виконання методу .tagName.toLowerCase(), який поверне ім’я тега в нижньому реєстрі.

Щоб отримати посилання на зображення необхідно звернутися до атрибута поточного елемента та присвоїти змінній src отримане значення.

Завдяки функції checkDivIsClose() визначаються чи закрита колонка або ряд, на основі розмірів попереднього елемента. Тобто якщо поточний елемент має розмір 0 тоді властивості closeDiv присвоюється значення close-col, якщо ж попередній елемент теж має розмір 0 тоді йому присвоюється значення close-row.

Код функції checkDivIsClose():

```
let checkDivIsClose = (size, previouseIsClose) => {
   if (size == 0) {
      return previouseIsClose ? 'close-row' : 'close-col'
   }
}
```

В залежності від проходження перевірок, будуть виконуватися різні частини коду, що дозволяє розділити властивості для різних типів елементів:

```
if (tag === "img" || element.firstChild.tagName == "IMG") {
  lastOpenedTag = null

  return {
    tag: 'img',
    content: src,
    closeDiv: null,
  };
}

const textElement = {
  tag: tag,
  content: element.innerText,
  closeDiv: closeDiv ?? null,
}

previousecloseDiv = elementSize == 0 ? true : false;
```

```
if (tag === 'p' || tag === 'div') {
  if (lastOpenedTag === null) {
    lastOpenedTag = textElement
  }
}

if (tag === 'h1') {
  lastOpenedTag = textElement
}

return textElement;
});

return json;
```

Після проходження усіх перевірок та виконання необхідних функції, return повертається об’єкт даних сайту.

Згенерований парсером об’єкт необхідно структурувати та відфільтрувати, для цього розроблена функція createPageStructure(), яка в своїй роботі створює нові екземпляри класу CreateElementClass [9].

CreateElementClass містить в собі конструктор який декларує локальні змінні, на основі переданих аргументів:

```
class CreateElementClass {

  constructor(tag, content, closeDiv) {
    this.tag = tag ?? null;
    this.content = content ?? null;
    this.closeDiv = closeDiv ?? null;
  }
```

В межах описаного класу реалізовано метод визначення розмірів елемента який базується на перевірці типу елемента, та розмірів його вмісту:

```
getElementSize(content, tag) {
  let elementLength = null;

  if (tag == "img") {

    const img = new Image();
```

```
    img.addEventListener("load", function () {
      return this.naturalWidth;
    });
    img.src = content;

    elementLength = img.width;

  } else {
    elementLength = content.length;
  }

  return checkElementSize(elementLength, tag)
}
}
```

Для визначення розмірів текстового елемента виконується виклик методу .length, який повертає кількість символів.

Щоб визначити розмір зображення потрібно спочатку створити новий екземпляр об’єкта Image(), далі добавити його в потік сторінки та викликати метод .naturalWidth, який поверне ширину зображення.

Також в даному класі використовується функція checkElementSize(), яка визначає кількість колонок яку займає елемент на сторінці:

```
let checkElementSize = (size, tag) => {

  if (tag == "p" || tag == "div") {
    return parseInt(size / 50);
  } else if (tag == "h1") {
    return parseInt(size / 5);
  } else if (tag == "img") {
    return parseInt(size / 100);
  }

  return 4
};
```

Функція createPageStructure() виконує структуризацію отриманого з парсера об’єкта. Тому для кращого розуміння алгоритму роботи функції, розглянемо її реалізацію.

Для початку необхідно створити нові екземпляри масивів, та задекларувати змінні, які будуть використовуватися в ході роботи функції:

```
let rows = new Array();
let row = new Array();
let column = new Array();
let columnElement = {};
let currentItem = 0;
let previouseElementSize = null;
let compareElementSize = null;
```

За допомогою цикла for of перебираємо об’єкт який міститься в змінній content. Цикл for of при кожній ітерації присвоює змінній element, поточну властивість об’єкта. Особливістю даного циклу є те що він працює тільки з об’єктами [9].

```
for (let element of content) {
```

Оскільки парсер зберігає усі синтаксичні одиниці, необхідно відсіяти елементи у яких відсутній контент:

```
if (element.content == null) {
   continue;
}
```

Далі потрібно виконати перевірку чи властивість closeDiv поточного елемента немає значень close-col або close-row. Якщо дана умова є істиною, тоді потрібно створити новий екземпляр класу CreateElementClass, визначити розмір поточного елемента завдяки методу .getElementSize() і присвоїтити його змінній elementSizeCurrent.

За допомогою тернарного оператора в змінній compareElementSize визначимо чи розмір поточного елемента більший за попередній, та присвоїмо відповідний результат змінній. Після чого за допомогою метода .push добавимо новий об’єкт елемента в масив column:

```
        if (element.closeDiv !== "close-col" && element.closeDiv !== "close-row") {
          columnElement = new CreateElementClass(element.tag, element.content,
element.closeDiv);
          elementSizeCurrent = columnElement.getElementSize(element.content, element.tag)
          compareElementSize = elementSizeCurrent > previouseElementSize ?
elementSizeCurrent : previouseElementSize;
          column.push(columnElement);
        }
```

Далі виконується перевірка яка дозволяє записати зібрані значення для колонки в масив row, та очищає compareElementSize, оскільки значення вже присвоєно властивості колонки, і для наступної ітерації воно повинно бути пустим, щоб не виконувалося порівняння елементів одної колонки з іншою, після чого очищається поточний масив column:

```
if (element.closeDiv == "close-col" || element.tag == "img") {
  column.size = compareElementSize;
  compareElementSize = null;

  row.push(column);

  column = new Array();
}
```

Наступна перевірка визначає чи поточний елемент закриває колонку, або чи є він останнім елементом в об’єкті content. Якщо отримується true, тоді для останньої колонки присвоюється розмір найбільшого елемента, та за допомогою метода push в масив row записуються дані змінної column, після чого масив row записуються в масив rows.

Коли всі операції запису виконано успішно, відбувається очищення змінних, за допомогою конструктора new:

```
if (element.closeDiv == "close-row" || currentItem == content.length - 1) {
  if (row.length !== 0) {
    column.size = compareElementSize
    compareElementSize = null;
```

```
      row.push(column);
      rows.push(row);

      row = new Array();
      column = new Array();
    }
  }
```

Коли сформовано структурований об’єкт даних, необхідно перетворити властивості об’єкту на DOM елементи, тому було розроблено наступні компоненти сторінки:

- Компонент елемента:

```
function getElement(elementData, index) {
  let element = [];

  if (elementData['tag'] == 'img') {
    element.push( React.createElement("img", {
      key: index,
      className: "img-fluid",
      src: elementData['content']
    }));
  }
  else {
    element.push(React.createElement(elementData['tag'], {
      key: {
        index
      }
    }, elementData['content']));
  }
  return element;
}
```

- Компонент елементів колонки:

```
function getElements(content) {
  let elements = [];
  let currentElementIndex = 0;

  for (element of content) {
    elements.push(getElement(element, currentElementIndex));

    currentElementIndex++;
  }
```

```
  return elements;
}
```

- Компонент усіх колонок в ряді:

```
function getColumns(content) {
  let columns = [];
  let currentColumn = 0;

  for (column of content) {
    let columnClass = ['col-' + column['size'], 'mb-4'];

    columns.push( React.createElement("div", {
      key: currentColumn,
      className: columnClass.join(" ")
    }, getElements(column)
        ));
    currentColumn++;
  }
  return columns;
}
```

- Компонент усіх рядів:

```
function getRows(content) {
  let rows = [];
  let isHeroBlock = null;
  let currentRow = 0;

  for (row of content) {
    isHeroBlock = currentRow == 0 ? 'hero-block' : '';

    let rowsClass = ['row', 'align-items-center', 'justify-content-between', 'mb-4', 'pb-5',
isHeroBlock, row['class'], getRowClass(row)];

    rows.push( React.createElement("div", {
      key: currentRow,
      className: rowsClass.join(" ")
    }, getColumns(row)));
    currentRow++;
  }
  return rows;
}
```

- Компонент Head частини:

```
function getHead() {
  let siteTitle = document.querySelector('h1').innerText;
  let siteDescrition = document.querySelector('p').innerText;

  return React.createElement("head", null,
    React.createElement("title", null, siteTitle),
    React.createElement("meta", {
      name: "descrition",
      content: siteDescrition
    }),
    React.createElement("meta", {
      name: "viewport",
      content: "width=device-width, initial-scale=1.0"
    }), React.createElement("title", null, "Rich Text Editor"), React.createElement("link", {
      rel: "stylesheet",
      href: "https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.1.1/css/all.min.css"
    }), React.createElement("link", {
      href: "https://fonts.googleapis.com/css2?family=Poppins&display=swap",
      rel: "stylesheet"
    }), React.createElement("link", {
      rel: "stylesheet",
      href: "style.css"
    }), React.createElement("link", {
      href: "https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css",
      rel: "stylesheet",
      integrity: "sha384-
Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRi",
      crossOrigin: "anonymous"
    }));
}
```

- Компонент Header:

```
function getHeader() {
  let siteName = document.querySelector('.site-name').value;

  return React.createElement("div", {
    className: "header container-lg d-flex justify-content-between align-items-center py-4
mb-5"
  }, React.createElement("div", {
    className: "logo"
  }, siteName), React.createElement("a", {
    className: "btn btn-primary"
  }, "Request"));
}
```

- Компонент Body:

```
function getBody(content) {
    return React.createElement("body", null, getHeader(), React.createElement("div", {
    className: "container-lg"
  }, getRows(content)), getfooter());
}
```

- Компонент Footer:

```
function getfooter() {
  return React.createElement("footer", {
    className: "text-center p-4"
  }, "2022. Site generated by Roman");
}
```

- Компонент усієї сторінки:

```
function getPage(props) {
    return React.createElement("html", null, getHead(), getBody(props.content));
}
```

Усі компоненти описуються за допомогою функції, яка як аргумент приймає об’єкт даних з інформацією про поточний елемент. Самі компоненти створюються за допомогою методу createElement().

Завдяки такі реалізації компонентів, система є досить гнучкою в плані подальшої підтримки та розвитку.

Після реалізації функції компонентів, необхідно виконати рендеринг сторінки, це стає можливим завдяки наступній реалізації, цього методу:

```
let getContentElements = () => document.querySelector("#text-input").innerHTML;
const root = ReactDOM.createRoot(document.getElementById('result'));

document.querySelector('#generate').addEventListener("click", () => {
  let content = createPageStructure(parser(getContentElements()));

  root.render( React.createElement(getPage, {
    content: content
  }));
```

});

Стрілкова функція getContentElements() використовується для отримання вмісту редактора, значення якої передається в функцію парсера, де виконується аналіз переданого контенту, та генерація об’єкту сторінки.

Далі для ReactDOM виконується функція createRoot, яка включається паралельний режим, та в аргументи приймає елемент для якого він має бути ввімкнений [13].

За допомогою функції addEventListener() виконується відстеження кліку по кнопці запуску, тому в тіло вище згаданої функції описується функціонал рендерингу сторінки.

Отриманий з getContentElements() об’єкт передається як аргумент в функцію createPageStructure(), в якій виконується фільтрація та структуризація даних сторінки.

Результат виконання останньої функції записується в змінну content, яка в подальшому передається як аргумент в функцію створення компоненту React.createElement(), яка в свою чергу присвоюється як аргумент функції рендерингу сторінки .render()[13].

Згенерована сторінка добавляється в відповідне поле в редакторі, де користувач може її проглянути. Якщо результат задовільнив користувача він натискає кнопку «Скачати сайт».

За скачування файлів сайту відповідає функція download(), яка як аргументи приймає назву файлу, та контент. Створює елемент <a> для якого завдяки методу .setAttribute() додаються атрибут href, який містить посилання на ресурс, та download, який дає браузеру зрозуміти що це посилання для скачування. Після чого створений елемент за допомогою метода .appendChild додається на сторінку, та після скачування файлу видаляється[19].

Завдяки такій реалізації скачування згенерованого сайту відбувається стандартними методами браузера.

Код функції download():

```
function download(filename, content) {
    let element = document.createElement('a');
    element.setAttribute('href', 'data:text/plain;charset=utf-8,' +
encodeURIComponent(content));
    element.setAttribute('download', filename);

    element.style.display = 'none';
    document.body.appendChild(element);

    element.click();

    document.body.removeChild(element);
}
```

Для виклику даної функції використано метод .addEventListener(), який дозволяє відстежувати клік по кнопці. В блоці вище описаної функції виконується декларація змінних назви файлу, та контенту, який отримується за допомогою метода .innerHTML з вікна результату генератора. Далі добавляється виклик функції download() з переданими атрибутами :

```
document.getElementById("dwn-btn").addEventListener("click", function(){

    let content = document.getElementById("result").innerHTML;
    let filename = "index.html";

    download(filename, content);
}, false);
```

Після виконання попередньої функції користувач отримує файл index.html, який містить усі необхідні ресурси для роботи сайту.

3.2. Реалізація прикладної частини застосунка

Для запуску розробленої системи потрібно відкрити в вікні браузера файл generator.html, після цього система готова до роботи.

Основна ідея розробленої системи полягає в простоті використання та інтуїтивно зрозумілому інтерфейсі. Тому дана система створена в мінімалістичному дизайні. На Рис.3.1 зображено органи управління генератором.

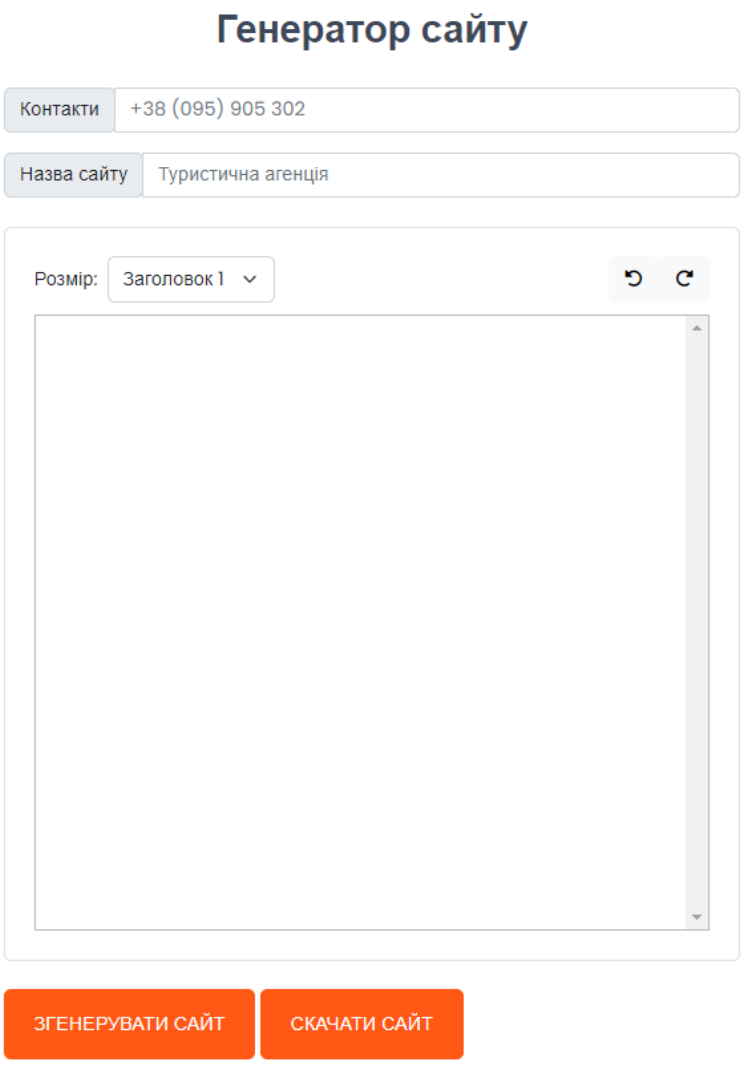

Рис.3.1. Органи управління генератора

Розроблена система має наступні органи управління:

- поле контактної інформації;
- назва сайту;
- розмір елемента:
  - заголовок (1-6);
  - параграф;
- кнопки навігації по змінах контенту
- текстове поле для контенту сайту
- кнопка «Згенерувати сайт»
- кнопка «Скачати сайт»

Для того щоб згенерувати сайт потрібно наповнити редактор інформацією встановлюючи при цьому розміри для заголовків. Щоб добавити зображення потрібно скопіювати його на будь-якому сайті та вставити в редактор. Коли всі поля заповнені, потрібно натиснути кнопку «Згенерувати сайт». Результат генерації показаний на Рис.3.2.

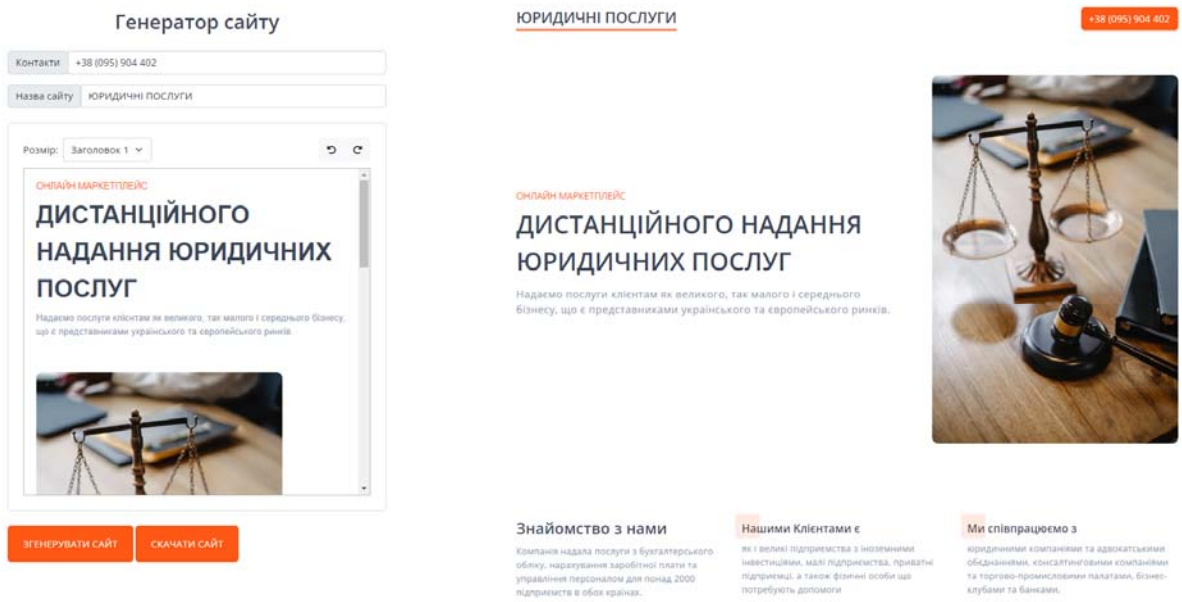

Рис.3.2. Результат генерації сайту

Щоб скачати сайт потрібно натиснути кнопку «Скачати сайт», після чого користувачу буде запропоновано зберегти файл собі на комп’ютер. Вміст скачаного файлу можна побачити на Рис.3.3.

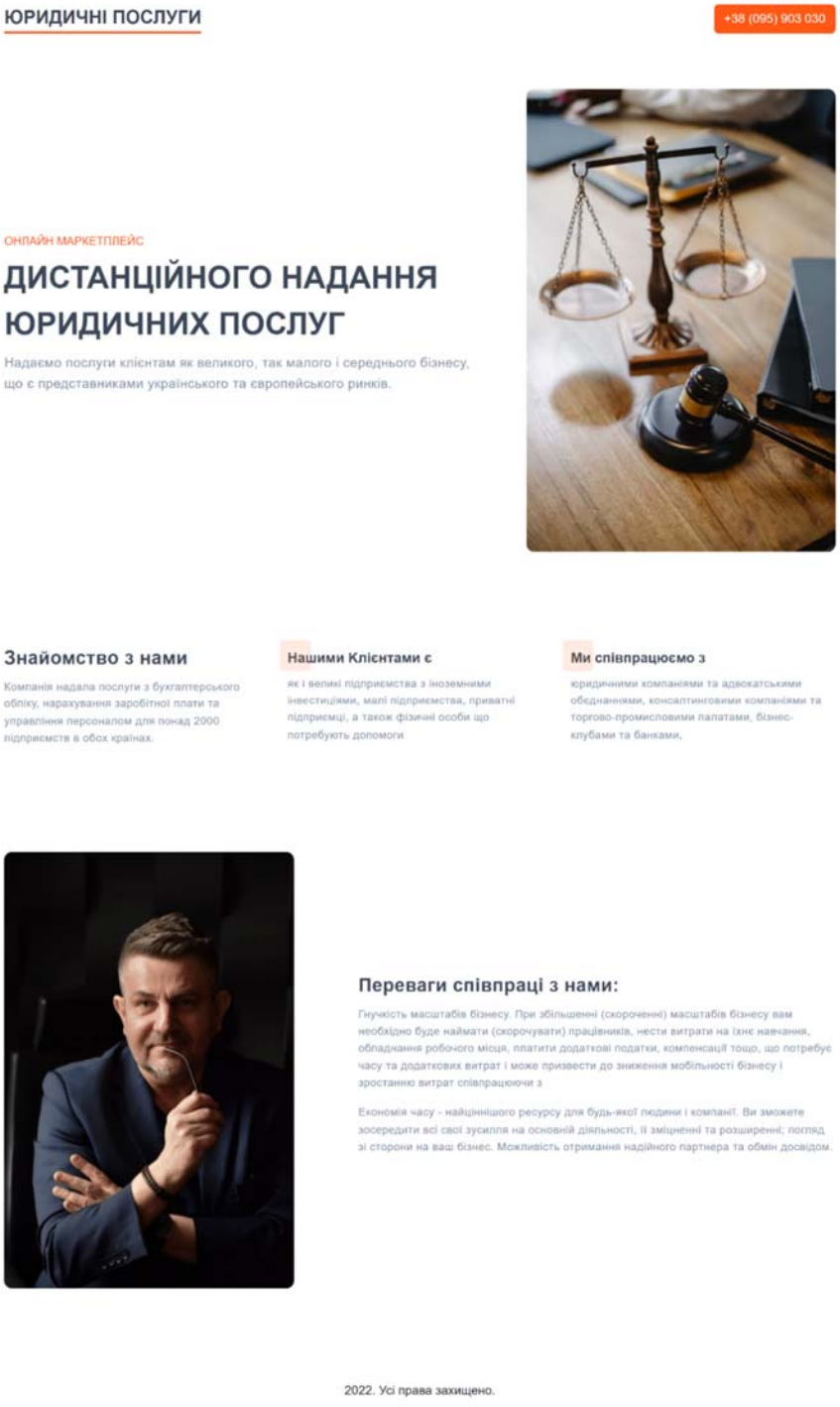

Рис.3.3. Згенерований системою сайт

На даному етапі система підтримує тільки зовнішні посилання на зображення та ресурси, що дозволило зменшити розміри згенерованого сайту до одного файлу.

Завдяки алгоритмам системи, згенерований код сторінки містить мінімум HMTL тегів та атрибутів, що позитивно впливає на показники швидкості.

Завдяки тому що система в своїх алгоритмах використовує CSS бібліотеку bootstrap, стилі сайту завантажуються з віддаленого сервера. На Рис.3.4 показаний код згенерованої сторінки.

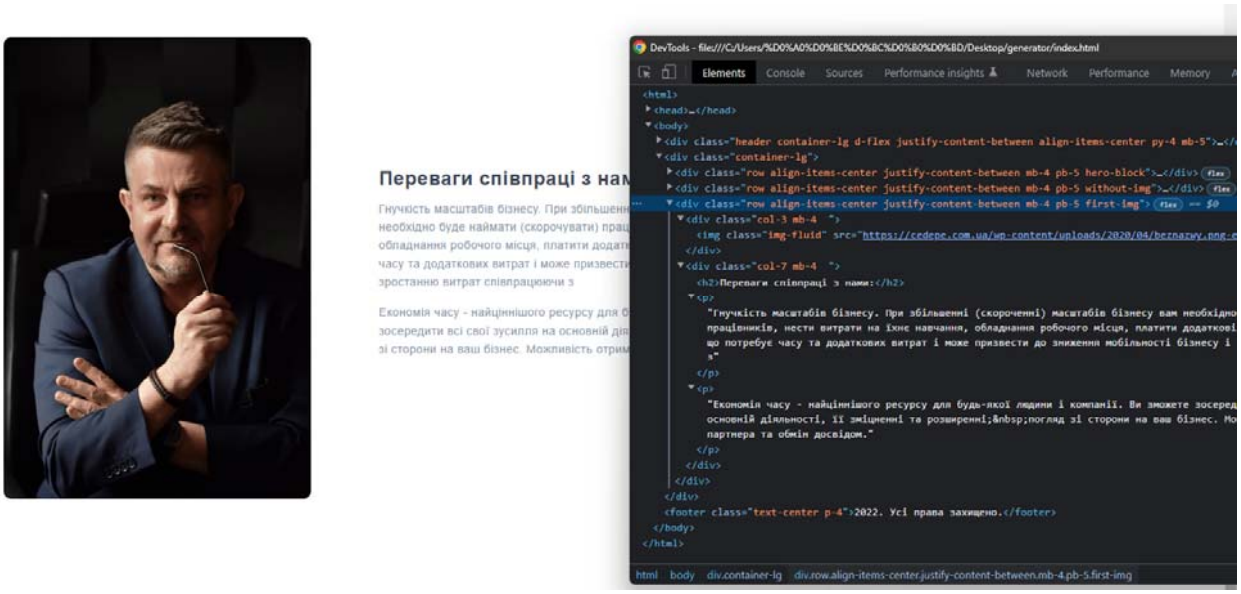

Рис.3.4. Код згенерованої сторінки

Висновки до розділу 3

1. На основі розробленого алгоритму, реалізовано функціональний код системи. Детально описано кожний з процесів, які відбуваються під час роботи системи.
2. Продемонстровано процес створення сайту в розробленій системі, та результат її роботи.

ВИСНОВКИ

В магістерській роботі розроблено систему, яка дозволяє автоматизувати процес розробки сайту. Реалізовані алгоритми системи дозволяють створювати сайт в декілька простих дій, без спеціальних технічних знань в сфері веб-розробки. Також було створено алгоритм аналізу вхідних даних, який дозволяє визначати тип та розміри елементу, на основі чого генерується структура сайту.

Розроблена система може бути як повноцінним рішенням для розробки сайтів, так і інструментом конвертації тексту в HTML код.

Проведено аналіз наявних на даний момент рішень, які дозволяють створювати сайти. Було визначено основі переваги та недоліки конструкторів сайтів в порівнянні з розробленою системою. До основних недоліків таких систем можна віднести:

- вартість розробки веб сайту;
- необхідні технічні знання;
- складний інтерфейс.

До переваг таких систем над розробленою можна віднести:

- готовий набір шаблонів;
- детальніша модифікація кожного з елементів сторінки;
- можливість змінити стиль сайту.

Виходячи з наведеної інформації можна стверджувати, що розроблена система, має великий потенціал для подальшого розвитку, оскільки завдяки розробленим алгоритмам, немає обмежень в плані модифікації функціональних можливостей системи.

Проведено дослідження процесу розробки сайтів, на основі аналізу роботи веб-студій. Алгоритми розробленої системи базуються на основі аналізу рекомендацій Google, щодо розробки веб сайтів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 12 brilliant examples of web design to inspire you in 2022. Hotjar: Website Heatmaps & Behavior Analytics Tools. URL: https://www.hotjar.com/web-design/examples/ (дата звернення: 13.11.2022).

2. 30 best web designers – plerdy. Plerdy – tools to improve conversions, usability analysis and SEO. URL: https://www.plerdy.com/blog/top-30-web-dizaynirov/ (дата звернення: 13.11.2022).

3. Best website builder (top 10 november 2022). Forbes Advisor. URL: https://www.forbes.com/advisor/business/software/best-website-builders/ (дата звернення: 13.11.2022).

4. Data structures: objects and arrays :: eloquent javascript. Eloquent JavaScript. URL: https://eloquentjavascript.net/04_data.html (дата звернення: 13.11.2022).

5. DOM Enlightenment - Exploring the relationship between JavaScript and the modern HTML DOM. DOM Enlightenment - Exploring the relationship between JavaScript and the modern HTML DOM. URL: http://domenlightenment.com/ (дата звернення: 13.11.2022).