МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: «Web-орієнтована система підтримки конструювання сайтів для дизайн-агентств»

Здобувача групи ІТ.м-23 Василець Данило Андрійович
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Викори
ідей, результатів і текстів інших авторів мають посилання на відповідне джер

_____ Данило ВАСИЛЕЦЬ
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник д-р. техн. наук, професор Євгеній ЛАВРОВ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІТ

Світлана ВАЩЕНКО
«_____» _____ 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Василець Данило Андрійович
(прізвище, ім'я, по батькові)

1 Тема кваліфікаційної роботи «Web-орієнтована система підтримки конструювання сайтів для дизайн-агентств»

затверджена наказом по університету від «08» листопада 2023 р. № 1249-VI

2 Термін здачі студентом кваліфікаційної роботи «18» грудня 2023 р.

3 Вхідні дані до кваліфікаційної роботи інформація про працівника дизайн-агентства, інформація про агентство, перелік шаблонів конструювання сайтів.

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, постановка задачі, проектування web-орієнтованої системи підтримки конструювання сайтів для дизайн-агентств, реалізація web-орієнтованої системи підтримки конструювання сайтів для дизайн-агентств

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації) актуальність роботи, мета та задачі проекту, аналіз аналогів, порівняльна таблиця аналогів, функціональні вимоги, засоби реалізації, моделювання проекту (контекстна діаграма, діаграма декомпозиції), діаграма варіантів використання, модель бази даних, демонстрація проекту, висновки.

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Дослідження предметної області	04.09.23-19.09.23	
2	Формування мети і задач	15.09.23-19.09.23	
3	Аналіз аналогів та проблем використання	20.09.23-27.09.23	
4	Виявлення вимог до проекту	28.09.23-02.10.23	
5	Планування робіт та розробка макету	03.10.23-06.10.23	
6	Вибір засобів реалізації	07.10.23-09.10.23	
7	Проектування інформаційної системи	10.10.23-19.10.23	
8	Розробка інформаційної системи	20.10.23-09.11.23	
9	Тестування та завершення роботи	10.11.23-13.11.23	
10	Оформлення пояснювальної записки	14.11.23-01.12.23	

Магістрант _____ Данило ВАСИЛЕЦЬ

Керівник роботи _____ д-р. техн. наук, професор Євгеній ЛАВРОВ

АНОТАЦІЯ

Тема кваліфікаційної роботи магістра «Web-орієнтована система підтримки конструювання сайтів для дизайн-агентств ».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 42 найменувань, додатків. Загальний обсяг роботи − 75 сторінок, у тому числі 51 сторінок основного тексту, 4 сторінки списку використаних джерел, 19 сторінок додатків.

Актуальність роботи полягає в тому, що із зростанням конкуренції у цифровому просторі, потреба у високоякісних, ефективних системах підтримки конструювання сайтів стає необхідністю. Ці системи стають критично важливими для успішної роботи дизайн- агентств, оскільки вони не лише спрощують процес створення сайтів, а й визначають їхню конкурентоспроможність в сучасному цифровому середовищі.

Мета роботи: створення інформаційної системи для спрощення і прискорення комплексного процесу конструювання веб-ресурсів за допомогою сучасних інструментів web-розробки на основі JavaSctipt та NodeJS, імплементованих у вигляді казуального інтерфейсу.

Ключові слова: TypeScript, NodeJS, NestJS, NuxtJS, SSR, сервер, клієнт, web-дизайн, конструювання сайтів.

ЗМІСТ

ВСТУП

Актуальність. Сьогодення - епоха, коли віртуальний простір стає не просто майданчиком для представлення бізнесу, але й ключовим інструментом взаємодії з аудиторією. Для дизайн-агентств створення веб-сайтів - не просто рекламний елемент, але й можливість продемонструвати їхню експертність та креатив. Із зростанням конкуренції у цифровому просторі, потреба у високоякісних, ефективних системах підтримки конструювання сайтів стає необхідністю. Ці системи стають критично важливими для успішної роботи дизайн- агентств, оскільки вони не лише спрощують процес створення сайтів, а й визначають їхню конкурентоспроможність в сучасному цифровому середовищі.

Об’єкт дослідження. Процес розробки web-орієнтованої системи підтримки конструювання сайтів для дизайн-агентств.

Предмет дослідження. Підтримка конструювання сайтів для дизайн-агентств.

Гіпотеза. Розробка та впровадження веб-орієнтованої системи підтримки конструювання сайтів для дизайн-агентств дозволить збільшити продуктивність процесу розробки, покращить якість створених сайтів, спростить співпрацю між дизайнерами та клієнтами, а також знизить час, необхідний для створення та запуску веб-проектів. Ця система дозволить оптимізувати робочі процеси, підвищить конкурентоспроможність дизайн-агентств на ринку та забезпечить більш ефективне використання ресурсів під час розробки веб-сайтів.

Структура. Робота складається зі вступу, постановки задачі дослідження, вибору методики та інструментів для рішення поставленої проблеми, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

Ця робота присвячена розробці веб-орієнтованої системи підтримки конструювання веб-сайтів для дизайн- агентств. Вона спрямована на створення ефективного інструментарію, який допоможе впроваджувати ідеї та концепції у віртуальному просторі швидко, легко та ефективно. Предметом цієї роботи є розробка програмної системи, що сприятиме оптимізації процесів створення веб-ресурсів для

дизайн- агентств, забезпечуючи їм можливість швидко реалізовувати свої концепції в онлайн-просторі та пристосовувати їх до потреб клієнтів.

Метою даної роботи є розробка та впровадження веб-орієнтованої системи, яка спростить та поліпшить процес конструювання веб-сайтів для дизайн- агентств, забезпечуючи їм інструменти для створення унікальних, функціональних та естетичних веб-ресурсів, що відповідають сучасним вимогам та очікуванням користувачів.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Огляд сучасного стану веб-дизайну

1.1.1. Тенденції у веб-дизайні

Сучасний веб-дизайн, як відображення технологічного прогресу та різноманітних потреб користувачів, виявляє тенденцію до мінімалізму. Ця філософія не обмежується лише стилем, а створює спрощені та зрозумілі веб-інтерфейси, де користувачі отримують чітку навігацію та доступ до важливої інформації. Мінімалістичний підхід відображається у простоті та лаконічності веб-сайтів, що відповідає сучасним очікуванням [1].

Окрім цього, анімація та мікроінтеракції набувають великого значення в сучасному веб-дизайні. Ці елементи не лише естетично прикрашають інтерфейс, але й значно покращують користувацький досвід. Вони роблять взаємодію цікавішою та залучають увагу, сприяючи покращенню взаємодії з веб-сайтом.

Треба враховувати, що важливість інтерфейсів мобільних пристроїв зростає, тому адаптація до різних розмірів екранів та оптимізація швидкості завантаження стають надзвичайно вагомими. Це зумовлює необхідність адаптації веб-сайтів під різні пристрої та забезпечення їхньої швидкості та ефективності для зручного використання користувачами [2].

Спостерігається значний розвиток інтерактивності та візуальності у сучасному веб-дизайні. Поглиблення функціональності та збільшення інтерактивних можливостей відкривають широкі можливості для залучення користувачів. Елементи якісного веб-дизайну включають анімацію, відеофони, 3D-графіку та інші ефективні засоби для залучення та утримання уваги користувачів.

Забезпечення безпеки веб-дизайну є критичним аспектом у сучасному цифровому середовищі. Під час розробки веб-сайтів, важливо враховувати й уникати загроз, які можуть виникнути через недоліки безпеки. Основні аспекти, на які варто звернути увагу, включають захист від зловмисних атак, безпеку персональних даних, а також загальну безпеку веб-сайту від вразливостей. При розробці веб-дизайну слід приділяти увагу заходам безпеки від потенційних атак. Однією з основних загроз є кібератаки, такі як SQL ін’єкції, кросс-сайтові сценарії та DDoS-атаки, що можуть

призвести до порушення роботи веб-сайту чи навіть крадіжки даних. Запобігання цим атакам включає в себе застосування відповідних технічних заходів, таких як регулярні оновлення програмного забезпечення та використання захисних антивірусних програм [3-5]. Окрім цього, безпека особистих даних користувачів є надзвичайно важливою. Враховуючи дотримання регулятивних норм (зокрема GDPR), веб-дизайнери повинні вживати заходів для збереження конфіденційності та безпеки особистих даних, які збираються та зберігаються на веб-сайті. Це включає в себе шифрування даних, використання безпечних протоколів передачі даних та інші методи захисту [3-5]. Також важливо пам'ятати про загальну безпеку веб-сайту від вразливостей, які можуть бути використані для несанкціонованого доступу чи зміни інформації. Це включає перевірку на вразливість веб-додатків, застосування правильних налаштувань безпеки сервера та мережі, а також використання захисних механізмів, таких як firewall. Загалом, забезпечення безпеки веб-дизайну є постійним процесом, який вимагає уважності та постійного вдосконалення заходів безпеки від розробників для збереження даних та функціональності веб-сайту перед потенційними загрозами [3-5].

Останнім часом зростає популярність темної теми веб-дизайну. Такий дизайн набуває великої популярності завдяки своїй естетиці, яка приваблює багатьох користувачів та надає веб-сайтам сучасний вигляд.

Також, персоналізація, адаптивний дизайн та використання VR-технологій вже не є просто приємним додатком. Вони стають вимогою для веб-сайтів, що прагнуть успішно конкурувати на ринку. Ці аспекти забезпечують індивідуальність та відповідність потребам користувачів, що є ключовими для створення успішних веб-платформ [6].

1.1.2. Методи розробки веб-сайтів для дизайн-агентств

Зважаючи на унікальність та специфіку дизайн-агентств, методи розробки їх веб-сайтів вимагають комплексного підходу та особливої уваги до деталей. Адаптивний дизайн є одним із ключових аспектів, оскільки він забезпечує коректне відображення сайту на різних пристроях - від настільних комп'ютерів до смартфонів

та планшетів. Це забезпечує оптимальний користувацький досвід незалежно від технічних параметрів пристрою.

Одним з етапів розробки є створення контенту, який не лише відображає професійні можливості агентства, а й стимулює зацікавлення відвідувачів. Важливо враховувати, що сайт має бути не лише естетичним, а й інформативним, передаючи ключову інформацію про послуги та роботи замовника [7].

SEO-оптимізація також відіграє важливу роль у розробці веб-сайтів. Вона спрямована на підвищення видимості сайту у пошукових системах, що є критичним для залучення нових клієнтів та розвитку бізнесу. Для цього важливо вибрати та використовувати правильні ключові слова, оптимізувати метатеги та створювати унікальний контент [8].

Оптимізація швидкості завантаження сторінок також має велике значення. Швидкий та ефективний сайт забезпечує не лише задоволення від користування, але й позитивний вплив на рейтинг сайту в пошукових системах, що покращує його позиції в пошукових запитах [8].

Не менш важливим є аналіз конкурентів та врахування їхніх стратегій в розробці веб-сайту. Вивчення сильних та слабких сторін конкурентів дозволяє побудувати більш ефективну стратегію для власного сайту.

1.1.3. Збалансованість між креативністю та функціональністю

Досягнення збалансованості між креативністю та функціональністю стає ключовим завданням. Креативність, як важливий елемент дизайну, вимагає оригінальності та інновацій. Для дизайн-агентств, важливо відображати унікальний стиль та естетичну привабливість на веб-сайті, але разом з цим, забезпечувати його ефективність та корисність для відвідувачів [9].

Функціональність веб-сайту для дизайн-агентства полягає у створенні не лише естетичного інтерфейсу, але й у забезпеченні зручності та легкості в користуванні. Правильна навігація, швидкий доступ до інформації та логічно побудовані розділи стають ключовими елементами функціональності.

Збалансований підхід включає в себе створення дизайну, який відображає візуальну привабливість та в той же час не ушкоджує функціональність веб-сайту. Це означає, що кожен елемент дизайну, будучи креативним, також повинен бути функціонально корисним для відвідувачів [10].

Досягнення оптимального балансу між креативністю та функціональністю веб-дизайну вимагає постійного вдосконалення та розглядання відгуків користувачів. Аналіз потреб та реакцій аудиторії допомагає впроваджувати веб-елементи, які є як креативними, так і практично корисними для відвідувачів.

Збереження цілісності дизайну та функціональності веб-сайту є важливим для забезпечення відмінного користувацького досвіду. Баланс між цими аспектами стає ключовим фактором у забезпеченні успіху веб-проектів для дизайн-агентств [11].

1.2. Аналіз потреб дизайн-агентств у веб-просторі

1.2.1. Вимоги до дизайну

Для дизайн-агентств надзвичайно важливо мати унікальність у веб-сайтах своїх клієнтів. Вони прагнуть до створення дизайну, який відповідає їхнім власним стандартам та бренду. Індивідуальність — це ключовий аспект, який дозволяє виділятися серед конкурентів та створювати унікальний імідж.

Дизайн-агентства працюють з різними клієнтами, які мають різні потреби та вимоги. Тому їм важливо мати можливість швидко та ефективно адаптувати веб-сайти під унікальні потреби кожного клієнта.

Вимога до дизайну полягає в тому, щоб він відображав і підтримував бренд клієнта. Візуальний ідентифікатор та унікальний стиль дозволяють створювати та зміцнювати імідж бренду, що є важливим для будь-якої компанії [12].

Оскільки дизайн-агентства працюють у сфері креативності та естетики, вимога до дизайну — це його візуальна привабливість. Не лише функціональність, але й гармонія кольорів, композиція та асиметрія є важливими аспектами [13].

Розвиток технологій у сфері веб-дизайну вимагає постійного оновлення та впровадження нових ідей. Дизайн-агентства шукають інноваційні підходи та оригінальні концепції для своїх проектів.

Зручний інтерфейс для управління контентом та можливість швидко та легко оновлювати інформацію на сайті — це важливі вимоги для забезпечення актуальності та релевантності інформації на веб-сайті.

1.2.2. Функціональність веб-сайту

Інтерактивність та анімація. Здатність веб-сайту створювати враження взаємодії та живості — важливий аспект. Інтерактивність, така як анімовані елементи, плавні переходи та відповідь на дії користувача, може покращити враження від сайту.

Мультимедіа контент. Можливість відтворення відео, аудіо файлів, анімацій та використання різноманітних мультимедійних форматів.

Функції зворотного зв'язку. Наявність форм для зв'язку, коментарів, можливість надсилання запитів чи відгуків — важлива функція для взаємодії з клієнтами та відвідувачами сайту.

Мобільна сумісність та респонсивний дизайн. Забезпечення коректної роботи та зручного відображення сайту на будь-яких пристроях — від смартфонів до планшетів та комп'ютерів.

Функціональність CMS. Важливим елементом є можливість швидко та зручно оновлювати контент на сайті за допомогою систем управління контентом, що спрощує процес редагування та додавання інформації.

Системи навігації та пошуку. Чітка та інтуїтивно зрозуміла система навігації, можливість швидкого пошуку не лише по контенту, а й по категоріях та характеристиках є важливими для зручного використання сайту користувачами [14].

1.2.3. SEO-принципи та маркетингові стратегії

Розглянемо ключові аспекти SEO-принципів та маркетингових стратегій для веб-сайту [15] **:**

Оптимізація контенту для пошукових систем. Це включає в себе використання ключових слів та фраз, створення унікального та цікавого контенту, який відповідає запитам користувачів і підвищує його позиції у пошукових системах.

Створення мобільно-дружнього інтерфейсу. Оскільки використання мобільних пристроїв для доступу до інформації постійно зростає, важливо мати мобільно-оптимізований сайт, що вплине на позиції у пошукових системах.

Стратегії внутрішнього та зовнішнього будування посилань. Розвиток ефективних стратегій отримання якісних зовнішніх посилань та організація внутрішніх посилань на сайті для поліпшення SEO.

Аналіз та вдосконалення метрик. Важливим етапом є постійний аналіз ключових показників, таких як відвідуваність, час на сайті, перехід від сторінки тощо, для вдосконалення стратегій та контенту.

Локальний SEO та оптимізація для місцевого пошуку. Для дизайн-агентств, що працюють в певному регіоні, важливо оптимізувати контент для місцевого пошуку, використовуючи ключові слова та фрази, що популярні в цьому регіоні.

Співпраця з іншими ресурсами та блогерами. Важливим елементом є партнерство з іншими впливовими сайтами та блогерами для підвищення авторитету та розповсюдження контенту.

1.3. Технологічний огляд

1.3.1. Мови програмування та їхні фреймворки: HTML/CSS, JavaScript (React, Angular, Vue.js).

HTML/CSS. HTML (HyperText Markup Language) та CSS (Cascading Style Sheets) відіграють ключову роль у створенні структури та візуального оформлення веб-сторінок. HTML визначає структуру документа, тоді як CSS відповідає за його оформлення та вигляд [16].

JavaScript. JavaScript є однією з найпопулярніших мов програмування для веб-розробки. Разом із фреймворками та бібліотеками, такими як React, Angular та Vue.js, JavaScript стає потужним інструментом для розробки високопродуктивних та інтерактивних веб-додатків.

React. Фреймворк React надає можливість розробки інтерфейсів користувача та є популярним вибором для створення односторінкових додатків. Він використовує віртуальний DOM для швидкості та простоти розробки.

Angular. Angular - це структурний фреймворк, який дозволяє створювати великі та складні односторінкові додатки. Він пропонує широкий набір інструментів для розробки та тестування додатків.

Vue.js. Vue.js володіє простотою в освоєнні та використанні, що робить його привабливим для новачків у веб-розробці. Він пропонує гнучкість та високу продуктивність, що робить його популярним вибором для невеликих та середніх проєктів.

Інші інструменти. Крім зазначених, існують різноманітні інструменти та бібліотеки, такі як jQuery, Bootstrap, SASS/LESS, які допомагають розширити можливості розробки та полегшити процеси створення веб-сайтів та додатків [16].

1.3.2. CMS: WordPress, Drupal, Joomla.

CMS (Системи управління контентом) відіграють ключову роль у розробці веб-сайтів та управлінні їх вмістом. Ось деякі з популярних CMS:

WordPress є однією з найпопулярніших платформ для створення веб-сайтів у світі. Ця система управління контентом відома своєю простотою використання та гнучкістю. Вона почала як платформа для створення блогів, але з часом перетворилася на потужний інструмент для будь-яких веб-проектів. Основна перевага полягає в тому, що WordPress надає можливість швидко створювати сайти будь-якої складності без глибоких знань програмування. Базується на системі плагінів та тем, що дозволяє розширювати його функціонал і зовнішній вигляд. У користувача є доступ до тисячі різних безкоштовних та платних тем і плагінів, що дає можливість налаштувати сайт під будь-які потреби. Пропонує широкі можливості для оптимізації SEO: вбудовані інструменти для редагування метатегів, постів, категорій, зручний URL-форматування, та інше. Це дозволяє створювати вміст, що оптимізований під пошукові системи, що в свою чергу покращує рейтинг сайту.

Однією з головних переваг є також активна спільнота користувачів та розробників. Тому якщо у вас виникли питання або проблеми, ви завжди можете знайти відповіді на них в онлайн-форумах або отримати підтримку від спеціалістів. WordPress залишається перспективною платформою для створення будь-яких веб-сайтів – від особистих блогів до корпоративних порталів чи комерційних платформ. Його простота використання та гнучкість роблять його привабливим для широкого кола користувачів [17].

Drupal - це ще одна популярна CMS, яка знайшла широке застосування веб-розробки. Її функціональність та гнучкість дозволяють створювати різноманітні веб-сайти - від особистих блогів до корпоративних порталів та урядових ресурсів. Однією з ключових особливостей є великий спектр можливостей для редагування, створення та управління контентом. Основною перевагою є його гнучкість та масштабованість. Він може використовуватися для створення веб-сайтів різної складності, починаючи від простих односторінкових сайтів до складних мережевих порталів. Ця CMS має велику кількість модулів, що дозволяє розширювати її функціональність з урахуванням конкретних потреб проекту. Drupal також відомий своєю високою безпекою та стійкістю до вразливостей. Велика спільнота розробників та користувачів постійно вдосконалює систему, виправляє помилки та забезпечує її безпеку. Проте, порівняно з іншими CMS, може вимагати більше технічних знань для початку роботи. Він може здаватися складним для новачків, хоча й надає значні можливості для досвідчених користувачів. Це може бути більшим плюсом для веб-розробників, які шукають рішення для складних або специфічних проектів. Загалом, Drupal є потужною CMS, яка відповідає вимогам різних видів веб-проектів, проте вона вимагає певного рівня технічної експертизи для повноцінного використання своїх можливостей [18].

Joomla відома своєю простотою в установці та використанні, що робить її популярним вибором для початківців та досвідчених користувачів. Має широкий спектр функцій, які дозволяють створювати веб-сайти різних типів: від блогів і сайтів-візитівок до корпоративних порталів чи інтернет-магазинів. Однією з ключових переваг Joomla є її велика кількість додатків та розширень, які доступні для

розробників. Це дозволяє розширювати функціональність сайту, використовуючи різні компоненти та модулі для досягнення конкретних цілей. Система відома своєю відносною простотою в управлінні контентом. Вона має інтуїтивний інтерфейс, що робить процес редагування та додавання контенту зручним для користувачів з різним рівнем досвіду. Однак Joomla має свої обмеження та недоліки. У порівнянні з іншими CMS, вона може виявитися менш гнучкою, особливо для розширення складних функцій або модифікації інтерфейсу. Також, хоча вона проста для початку, може мати певні обмеження у масштабуванні для великих проектів. Усупереч цим обмеженням, Joomla залишається популярним рішенням для тих, хто шукає простий спосіб створення веб-сайтів з базовими функціями та підтримкою спільноти для допомоги та розвитку [19].

1.3.3. Інструменти для дизайну: Adobe Creative Suite, Sketch, Figma.

Adobe Creative Suite. Adobe Creative Suite є інтегрованою колекцією програм для роботи з графікою, дизайну та мультимедіа. Складається з різних програм, таких як Photoshop, Illustrator, InDesign, Premiere Pro, та багатьох інших, кожна з яких спеціалізується на певних аспектах творчості та редагування.

Photoshop - це редактор графіки, що надає можливості для роботи з фотографіями, створення графічних ефектів та маніпулювання зображеннями. Illustrator - це програма для векторного малювання, яка дозволяє створювати векторні графічні об'єкти. InDesign - це програма для верстки, спеціалізована на створенні публікаційного контенту, такого як книги, брошури та журнали.

Premiere Pro - це відеоредактор, який дозволяє монтажувати, обробляти та оптимізувати відео. Ця колекція програм розроблена для професійних користувачів у галузі дизайну, мультимедіа та відеоредакції, що дозволяє їм створювати вражаючий та професійний контент за допомогою різноманітних інструментів та функцій.

Sketch. Sketch - це векторний графічний редактор, спеціалізований на дизайні інтерфейсів користувача для веб-сайтів та мобільних додатків. Він відомий своєю простотою використання та великим набором плагінів, що полегшують роботу з дизайном [20].

Figma. Figma - це веб-платформа для дизайну, яка дозволяє командам спільно працювати над проектами в режимі реального часу. Вона поєднує в собі можливості векторного редактора та інструментів для прототипування, спрощуючи спільну роботу дизайнерів та розробників [21].

Ці інструменти стали необхідними у світі веб-дизайну, надаючи широкі можливості для створення, редагування та прототипування інтерфейсів користувача та графічних компонентів для веб-сайтів. Вони дозволяють дизайнерам працювати ефективно та створювати професійні та залучені веб-проекти [22].

1.4. Аналіз конкуренції та аналогічних продуктів

Було проведено аналіз веб-сайтів - аналогів. Було розглянуто їхню графіку, використані технології, функціональність, структуру та загальний дизайн.

1.4.1. Конструктори сайтів

Wix. Платформа для створення сайтів, яка надає можливості навіть користувачам без технічного досвіду веб-розробки. Вона відома своєю простотою та інтуїтивним інтерфейсом, що дозволяє швидко створювати веб-сайти, що виглядають професійно, за допомогою перетягування та розміщення елементів (Drag-and-Drop). Однією з головних переваг Wix є його багатий вибір готових шаблонів, які користувач може використовувати як основу для створення власного дизайну. Це робить процес створення сайту простим і швидким, адже користувач може вибрати той шаблон, який відповідає його потребам або бізнесу. Wix пропонує широкий спектр функцій, таких як блоги, магазини, резервування онлайн, адаптивний дизайн і SEO-інструменти. Це дозволяє створювати різноманітні типи веб-сайтів і пристосовувати їх до різних потреб користувачів [23].

Недоліком Wix може бути обмеженість у масштабуванні та налаштуванні складних функцій порівняно з іншими платформами. Також, у деяких випадках використання безкоштовного плану може обмежувати можливості користувача та

вимагати оновлення до платних планів для додаткового функціоналу. Однак, Wix залишається популярним серед користувачів, які шукають швидкий і простий спосіб створення веб-сайтів без глибоких знань веб-розробки. Його інтуїтивний інтерфейс та гнучкість в роботі з дизайном зробили його важливим гравцем на ринку рішень для веб-сайтів [24].

Squarespace. Інтегрована платформа для створення веб-сайтів, що надає засоби для швидкого і простого розгортання сайтів будь-якого типу. Його особливість полягає в естетичному дизайні та високій функціональності. Платформа пропонує різноманітні стилізовані шаблони, які дозволяють користувачам створювати стильні та ті, що виглядають професійно, веб-сайти. Squarespace пропонує широкий спектр інструментів для веб-дизайну, включаючи інтуїтивний редактор, який дозволяє вносити зміни у макети, кольори, шрифти та інші елементи сайту. Платформа також має вбудовані функції для створення інтернет-магазинів, блогів, галерей та інших типів веб-сайтів. Однією з переваг Squarespace є його простота використання та інтуїтивний інтерфейс, що дозволяє користувачам без технічних навичок створювати власні сайти. Крім того, платформа надає хостинг, що зменшує необхідність в розміщенні веб-сайту на іншому хостинг-провайдері [25].

Недоліком Squarespace може бути обмежена гнучкість у налаштуванні окремих елементів веб-сайту, порівняно з іншими більш гнучкими платформами. Також, вона може бути менш пасуючою для складних або великих проектів, де потрібне велике розширення функціональності.

Усі ці особливості роблять Squarespace привабливим для тих, хто шукає швидке та естетичне рішення для створення веб-сайтів, особливо для митців, фотографів, та малих бізнесів, які вважають дизайн важливою частиною свого онлайн-присутності [26].

1.4.2. Інші аналогічні платформи

Shopify. Спеціалізується на створенні та розгортанні електронних комерційних платформ для продажу товарів та послуг в Інтернеті. Ця платформа надає гнучкість

та можливості для підприємців, бізнесів будь-якого масштабу та галузі, щоб легко створювати, налаштовувати та управляти своїми онлайн-магазинами [27].

Основною перевагою Shopify є його інтуїтивний інтерфейс та легкість використання. Платформа пропонує велику кількість готових тем та шаблонів для дизайну, що дозволяє швидко створювати естетично привабливі сторінки. Крім того, вона має ряд вбудованих інструментів для обробки платежів, управління запасами, аналітики та звітності, які полегшують керування електронним бізнесом.

Ще однією з важливих переваг є можливість інтеграції з різноманітними додатками та засобами, які розширюють функціональні можливості сайту. Також, Shopify пропонує добре розроблену систему підтримки користувачів, що сприяє вирішенню питань та підтримці під час роботи з платформою [27].

1.5. Вивчення користувацького досвіду (UX) та дизайну інтерфейсу (UI)

В розділі вивчення UX/UI, ключовою точкою є аналіз елементів та тестування для забезпечення оптимального користувацького досвіду та ефективного дизайну інтерфейсу.

Елементи UX/UI

Елементи UX/UI включають у себе різні аспекти, спрямовані на створення ефективної та зручної взаємодії між користувачем і веб-сайтом. Один з ключових елементів - це навігація. Вона орієнтує користувача на сайті, роблячи його легкодоступним та зрозумілим. Ефективна навігація дозволяє швидко знайти потрібну інформацію та забезпечує легкість переміщення між різними розділами веб-сайту. Ретельне планування структури меню, пошукових систем та внутрішніх посилань допомагає підтримувати ефективну навігацію.

Ще одним елементом UX/UI є взаємодія з користувачем. Вона описує способи, якими користувачі взаємодіють з елементами сайту та взаємодіють з контентом. Зручність використання, якість відповіді на взаємодію користувача з веб-сайтом, швидкість завантаження сторінок - це всі важливі аспекти взаємодії. Коректна робота

кнопок, форм, відображення повідомлень про помилки та логічна послідовність операцій забезпечують зручну взаємодію з сайтом.

Важливим елементом є також візуальний дизайн. Це охоплює аспекти, які стосуються естетичності та привабливості веб-сайту. Візуальний дизайн включає в себе колірну палітру, типографіку, графічні елементи та структуру контенту. Ці аспекти сприяють створенню привабливого та естетичного дизайну, який привертає увагу користувачів.

Поміркованість використання анімацій та переходів, спритність у використанні простору на сторінці, адаптивність веб-сайту для різних пристроїв - це також важливі аспекти UI/UX, що сприяють поліпшенню взаємодії користувача з сайтом. Оптимізація веб-сайту для різних пристроїв (мобільні телефони, планшети, комп'ютери) гарантує доступність і зручність використання в будь-який час і в будь-якому місці [28-31].

Розробка ефективного UX/UI вимагає ретельного вивчення поведінки користувачів. Аналіз психології та привичок користувачів допомагає створити інтуїтивно зрозумілі та легкі у використанні інтерфейси. Це означає визначення та відображення потреб цільової аудиторії через спеціальні інструменти, такі як персони або сценарії використання. Результат - сайт, який ефективно вирішує завдання користувача та задовольняє його очікування.

Системи аналітики, тестування та збору даних грають ключову роль у вдосконаленні UX/UI. Вони дозволяють збирати і аналізувати дані про поведінку користувачів на сайті, що надає можливість виявляти проблемні місця та вдосконалювати їх. Тестування UX/UI може включати тести A/B, тести користувацької взаємодії, опитування та спостереження, спрямовані на отримання зворотного зв'язку [28-31].

Застосування принципів доступності є важливим для забезпечення того, щоб веб-сайт був доступним для всіх користувачів, незалежно від їхніх можливостей чи технічних обмежень. Розуміння і впровадження принципів доступності дозволяє створити веб-сайт, який може використовувати широке коло людей.

Необхідність постійного удосконалення та вдосконалення UX/UI призводить до того, що розробники та дизайнери стежать за новими технологіями та практиками. Сучасні тренди включають в себе використання штучного інтелекту для персоналізації веб-сайтів, використання VR/AR-технологій для взаємодії з контентом, а також використання геолокації для розширення функціональності.

Одним із ключових аспектів успішного UX/UI є реактивність та адаптивність. Вміння сайту адаптуватися до різних пристроїв, включаючи мобільні, планшети та настільні комп'ютери, важливо для забезпечення зручного користування та позитивного враження від сайту.

Створення UX/UI - це постійний процес, що вимагає не лише створення естетичного дизайну, а й постійного вдосконалення функціональності та сприятливого досвіду для кожного користувача [32-33].

Тестування UX/UI

Тестування UX/UI включає в себе різні методики, спрямовані на оцінку користувацького досвіду та інтерфейсу веб-сайту.

Однією з ключових методик тестування є проведення спостережень за користувачами під час їх взаємодії з сайтом. Це може включати спостереження їхніх дій, реакцій на взаємодію з інтерфейсом, виявлення проблемних моментів та фіксацію реакцій на окремі функції або елементи сайту.

Крім цього, проведення тестів A/B є ефективним способом порівняння ефективності різних варіантів дизайну чи функціоналу. За допомогою таких тестів можна порівнювати два або більше варіантів певних елементів, функцій чи навіть дизайну та визначати, який з них викликає більш позитивну реакцію у користувачів. Також існують опитування та фокус-групи, які дозволяють отримати зворотній зв'язок від користувачів. Опитування можуть включати запитання про сприйняття інтерфейсу, його зручність, а також про конкретні проблеми чи пропозиції щодо вдосконалення. Фокус-групи дозволяють глибше розібратися в вимогах користувачів, їхніх потребах та перевагах.

Ще однією важливою частиною тестування є аналіз метрик. Це включає в себе вимірювання різних параметрів, таких як час, який користувач проводить на сайті, кількість відвідувань певних сторінок, частота використання певних функцій тощо. Ці метрики допомагають виявити слабкі місця та вдосконалити UX/UI.

Нарешті, важливим етапом тестування є врахування зворотного зв'язку від користувачів. Це може бути навіть важливіше за тестування функціоналу самого сайту. Відгуки користувачів можуть виявити ті аспекти, які не відобразилися у попередніх тестах, але є критичними для поліпшення веб-сайту [34-45].

Метрики успішності UX/UI

Метрики успішності UX/UI включають в себе широкий спектр параметрів, що оцінюють користувацький досвід та ефективність веб-сайту.

Один із ключових показників - це час, який користувачі проводять на веб-сайті. Він може свідчити про залученість та цікавість користувачів контентом, дизайном та інтерфейсом. Однак варто пам'ятати, що довгий час перебування на сайті не завжди означає позитивний досвід – це може також вказувати на те, що користувачі не можуть знайти потрібну інформацію.

Кількість відвідувань окремих сторінок або розділів сайту - це ще один показник. Він може вказати на те, які частини сайту є найбільш цікавими або корисними для відвідувачів.

Конверсія – це той показник, який оцінює, наскільки успішно користувачі виконують певні дії на сайті, наприклад, підписка на новини, замовлення товару, заповнення форми тощо.

Метрика відмови (bounce rate) вказує на те, скільки користувачів залишають сайт без переходу на інші його сторінки. Високий bounce rate може бути показником проблем зі зручністю, відповідністю контенту або іншими аспектами сайту.

Задоволення користувачів відображається через опитування, відгуки та рейтинги. Відповіді користувачів щодо зручності взаємодії з сайтом, візуального оформлення та загального досвіду можуть надати важливі вказівки щодо подальших покращень.

Нарешті, додатковими метриками можуть бути скорочення часу на виконання певних завдань на сайті, показник використання ключових функцій, взаємодія з мобільною версією сайту тощо. Вони допомагають отримати більш детальний огляд процесу взаємодії користувачів з сайтом та можливість виявити потенційні проблеми чи місця для покращень [36].

Підсумковий аналіз результатів

Об'єднуючи аспекти навігації, ефективної взаємодії та аналізу результатів тестів, отримуємо можливість створити дизайн інтерфейсу, який буде відповідати потребам та очікуванням користувачів, покращуючи їхній загальний досвід використання продукту.

1.6. Огляд інноваційних технологій та майбутніх тенденцій

1.6.1. VR/AR технології веб-дизайну: можливості, застосування в дизайні

VR/AR (віртуальна та доповнена реальність) стають каталізаторами інновацій у веб-дизайні. Одна з їхніх ключових можливостей - це створення зовсім нового способу сприйняття веб-сайтів. VR дозволяє користувачеві зануритися у повноцінне віртуальне середовище, де кожен аспект сторінки стає тривимірним, і користувач може взаємодіяти з елементами сайту у "реальному" просторі.

Застосування VR/AR у веб-дизайні не обмежується простими демонстраціями. Навпаки, це може бути використано для створення інтерактивних презентацій продуктів або послуг, де клієнти можуть "випробувати" товари чи перевірити певні функції. Додаток AR для веб-дизайну може використовуватися для віртуального огляду простору під час планування будівництва або для підвищення інтерактивності інтернет-магазинів.

Технології VR/AR дозволяють дизайнерам створювати унікальні враження для користувачів. Наприклад, вони можуть побачити продукт у деталях, обертаючи його на всі боки або випробувати функціонал сайту через віртуальний інтерфейс. Це

революційно змінює спосіб, яким користувачі сприймають інформацію та взаємодіють з веб-сайтами [37].

1.6.2. Інтернет речей (IoT) в дизайні: взаємодія з різними пристроями, адаптованість до IoT

Інтернет речей (IoT) відкриває безліч нових можливостей для веб-дизайну, надаючи змогу взаємодії з різними пристроями та апаратним забезпеченням. В цьому контексті, веб-дизайн відкриває широкі перспективи, оскільки пристрої, які підтримують IoT, включаються в мережу Інтернету, тож веб-розробники мають можливість створювати інтерфейси, що взаємодіють з цими пристроями.

Однією з ключових переваг IoT в дизайні є здатність створювати веб-сайти, які адаптовані під різні пристрої, що входять до цієї мережі. Це означає, що веб-дизайн повинен бути гнучким і здатним адаптуватися під різні екранні розміри та специфікації пристроїв. Застосування IoT у веб-дизайні вимагає від розробників не лише розуміння принципів мобільного дизайну, але й урахування можливостей і особливостей підключених до мережі пристроїв.

Для веб-дизайнерів, важливо розуміти, що збільшення кількості пристроїв, підключених до мережі Інтернет речей, призводить до зростання обсягів даних, що обробляються та відображаються на веб-сторінках. Відповідно, використання IoT вимагає ефективного управління обсягами інформації, забезпечуючи її точність та вчасність.

Одним із викликів, пов'язаних із застосуванням IoT у веб-дизайні, є забезпечення безпеки даних, оскільки підключені пристрої можуть бути уразливими перед зловмисниками. Розробники повинні враховувати цей аспект і використовувати механізми шифрування та захисту для забезпечення конфіденційності даних, що передаються між веб-сайтом та підключеними пристроями.

Особливо важливим для веб-дизайну в контексті IoT є розуміння споживчих звичок і поведінки користувачів, які використовують підключені пристрої. Це дозволяє створювати інтерфейси, які не лише ефективно взаємодіють з пристроями, але й забезпечують користувачам зручність та функціональність.

В цілому, застосування IoT у веб-дизайні відкриває широкі перспективи для створення більш інтерактивних та адаптивних веб-сайтів, які відповідають потребам сучасних споживачів і забезпечують надійну та безпечну взаємодію з підключеними пристроями [38-40].

1.6.3. Майбутні перспективи: розвиток AI в дизайні, blockchain у веб-розробці

Майбутність у веб-розробці виглядає яскраво завдяки розширенню використання штучного інтелекту (AI) у дизайні та блокчейн технологій у веб-розробці. Штучний інтелект поступово займає своє місце в дизайні, пропонуючи надзвичайно цікаві можливості для створення унікальних інтерфейсів. Алгоритми машинного навчання можуть аналізувати користувацькі дані та впливати на процес створення дизайну, враховуючи індивідуальні вподобання та потреби користувачів [41].

Крім того, використання блокчейн технологій у веб-розробці поступово набуває популярності. Блокчейн може забезпечити більш високий рівень безпеки, непідробність даних та захист особистої інформації. У веб-розробці, це може означати створення більш безпечних та надійних механізмів для зберігання конфіденційних даних користувачів, забезпечуючи їхню приватність та безпеку.

Розвиток AI в дизайні може привести до створення автоматизованих інструментів, які спростять процес розробки веб-сайтів та інтерфейсів. Вони зможуть швидко створювати макети та пропозиції, враховуючи попередні вимоги клієнта. Крім того, використання AI у веб-дизайні може підвищити персоналізацію веб-сайтів, пропонуючи користувачам більш індивідуалізований досвід взаємодії з сайтом.

Щодо блокчейну, його застосування у веб-розробці може допомогти у створенні розподілених мереж, що дозволить зберігати дані більш безпечно та ефективно. Ця технологія також може бути використана для покращення систем монетизації веб-сайтів через криптовалютні рішення, що відкриває нові можливості для оплати та взаємодії з контентом в Інтернеті.

Загалом, інтеграція штучного інтелекту в дизайні та розробці веб-сайтів, а також використання блокчейну, показують величезний потенціал для покращення та розвитку технологій у цих сферах. Вони можуть стати ключовими елементами майбутнього веб-дизайну, надаючи більші можливості, безпеку та інновації [42].

2. ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1. Мета та задачі дослідження

Об’єкт дослідження: Процес розробки web-орієнтованої системи підтримки конструювання сайтів для дизайн-агентств.

Предмет дослідження: Підтримка конструювання сайтів для дизайн-агентств.

Гіпотеза: Розробка та впровадження веб-орієнтованої системи підтримки конструювання сайтів для дизайн-агентств дозволить збільшити продуктивність процесу розробки, покращить якість створених сайтів, спростить співпрацю між дизайнерами та клієнтами, а також знизить час, необхідний для створення та запуску веб-проектів. Ця система дозволить оптимізувати робочі процеси, підвищить конкурентоспроможність дизайн-агентств на ринку та забезпечить більш ефективне використання ресурсів під час розробки веб-сайтів.

Практична цінність: Дослідження спрямоване на вдосконалення процесу конструювання сайтів для дизайн- агентств, забезпечення унікальних можливостей. Система буде відзначатися використанням передових технологій, таких як швидкий та зручний інтерфейс, адаптований під потреби користувача, інтуїтивно зрозумілі інструменти для створення візуально привабливих та функціональних сайтів.

Мета: створення інформаційної системи для спрощення і прискорення комплексного процесу конструювання веб-ресурсів за допомогою сучасних інструментів, імплементованих у вигляді казуального інтерфейсу.

Задачі:

- Проаналізувати обрану предметну область, існуючі аналоги та потрібні для створення нової системи технології.
- Сформувати мету, завдання дослідження та функціональні вимоги, Спланувати план розробки.
- Розробити архітектуру та логіку інформаційної системи конструювання сайтів.
- Реалізувати серверну та клієнтську частину інформаційної системи, базу даних.

Функціональні вимоги:

- Серверна частина система має дотримуватись стандартів REST архітектури.
- Система повинна надавати спосіб реєстрації/авторизації користувачів.
- В системі має бути зручна організація особистих проєктів користувача у виглядів списку.
- Система має надавати простий та інтуїтивний інтерфейс конструювання проєктів-сайтів користувача.
- Вихідний код сайту, що конструюється, має бути в зручному доступі користувача у вигляді директорії файлів.

3. МОДЕЛЮВАННЯ

3.1. Структурно-функціональне моделювання процесу

Було проведено структурно-функціональне моделювання процесу підтримки конструювання сайтів для дизайн-агентств. Результати були візуалізовані за допомогою додатку Bpwin. Вихідні схеми наведено на рисунках у вигляді контекстної діаграми в нотації IDEF0 (рисунок 3.1) та перший рівень декомпозиції попередньої діаграми (рисунок 3.2).

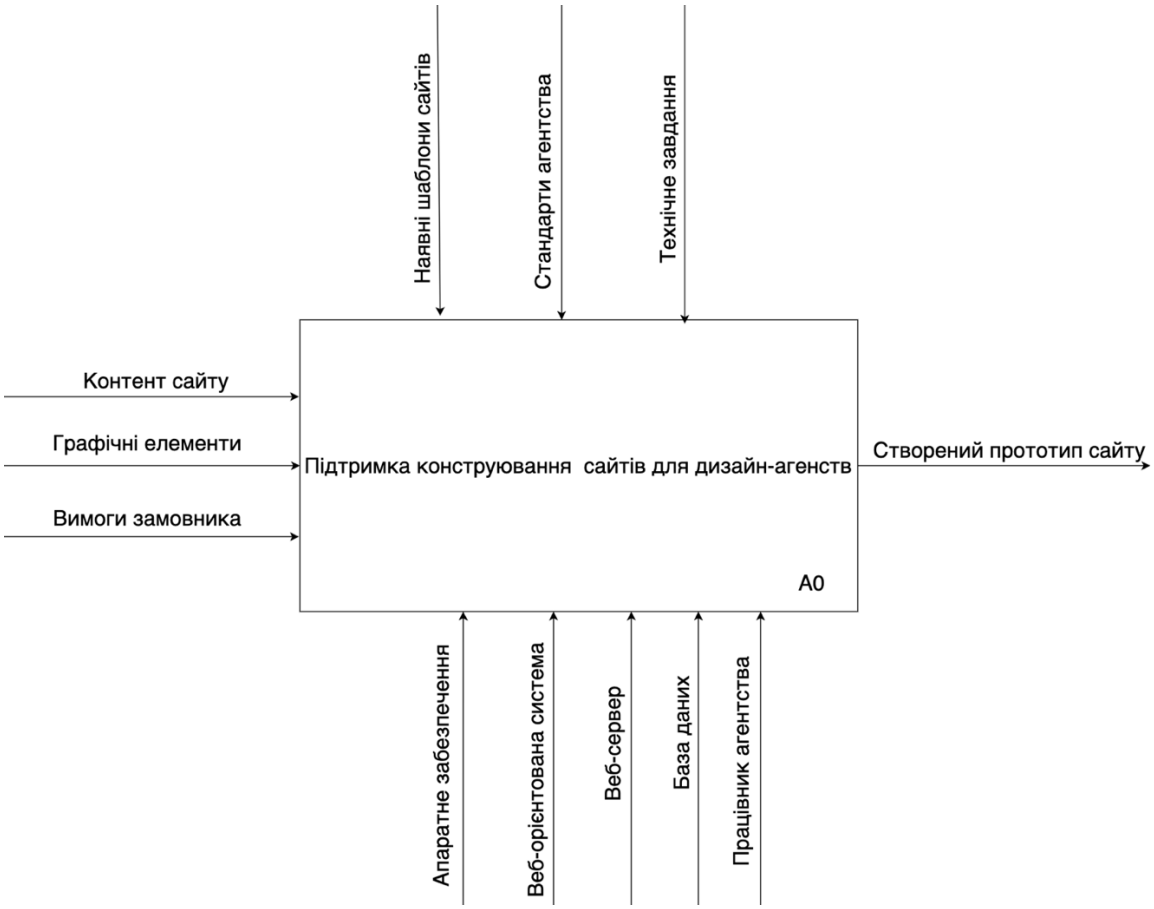

Рисунок 3.1 – Контекстна діаграма IDEF0

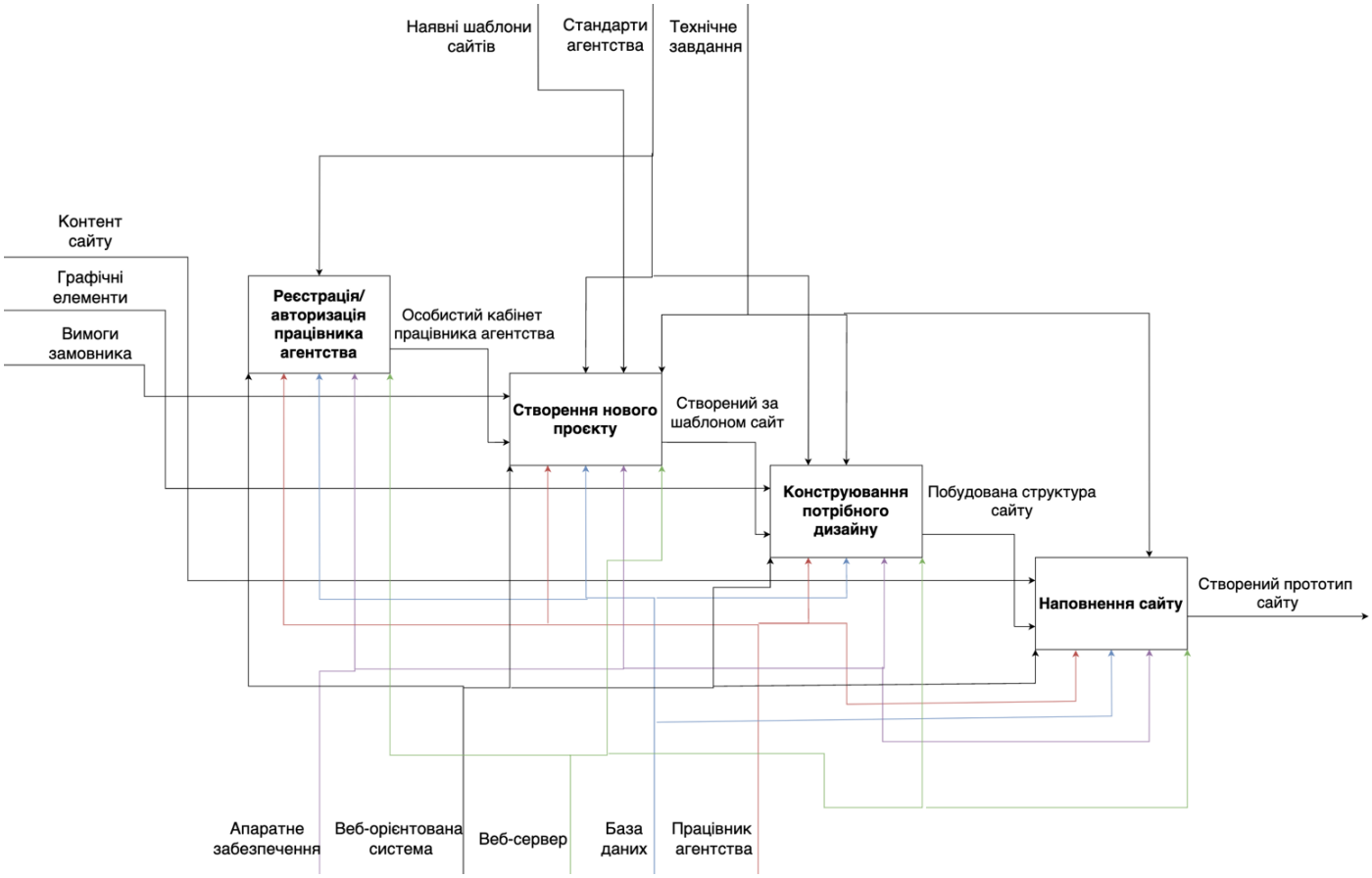

Рисунок 3.2 – Декомпозиція першого рівня

3.2. Проектування моделі бази даних

За результатами проектування моделі бази даних було виділено основні потрібні сутності для функціонування системи.

Сутність user. Містить основну інформацію про користувача (його особисті дані, роль у системі, а також зовнішні ключі для зв’язку із сайтами, що були ним створені, якщо користувач не є адміністратором).

Сутність site. Містить інформацію про користувача, котрим було створено сайт, назву сайту та вкладені відомості про шаблон сайту.

Сутність refreshToken. Містить потрібну інформацію для ключу поновлення сесії користувача.

Сутність template. Містить інформацію про назву, базове зображення, та шлях до структури директорії шаблону для створення сайту.

Далі на рисунку зображено модель бази даних, реалізованої в СУБД PostgreSQL (рисунок 3.3).

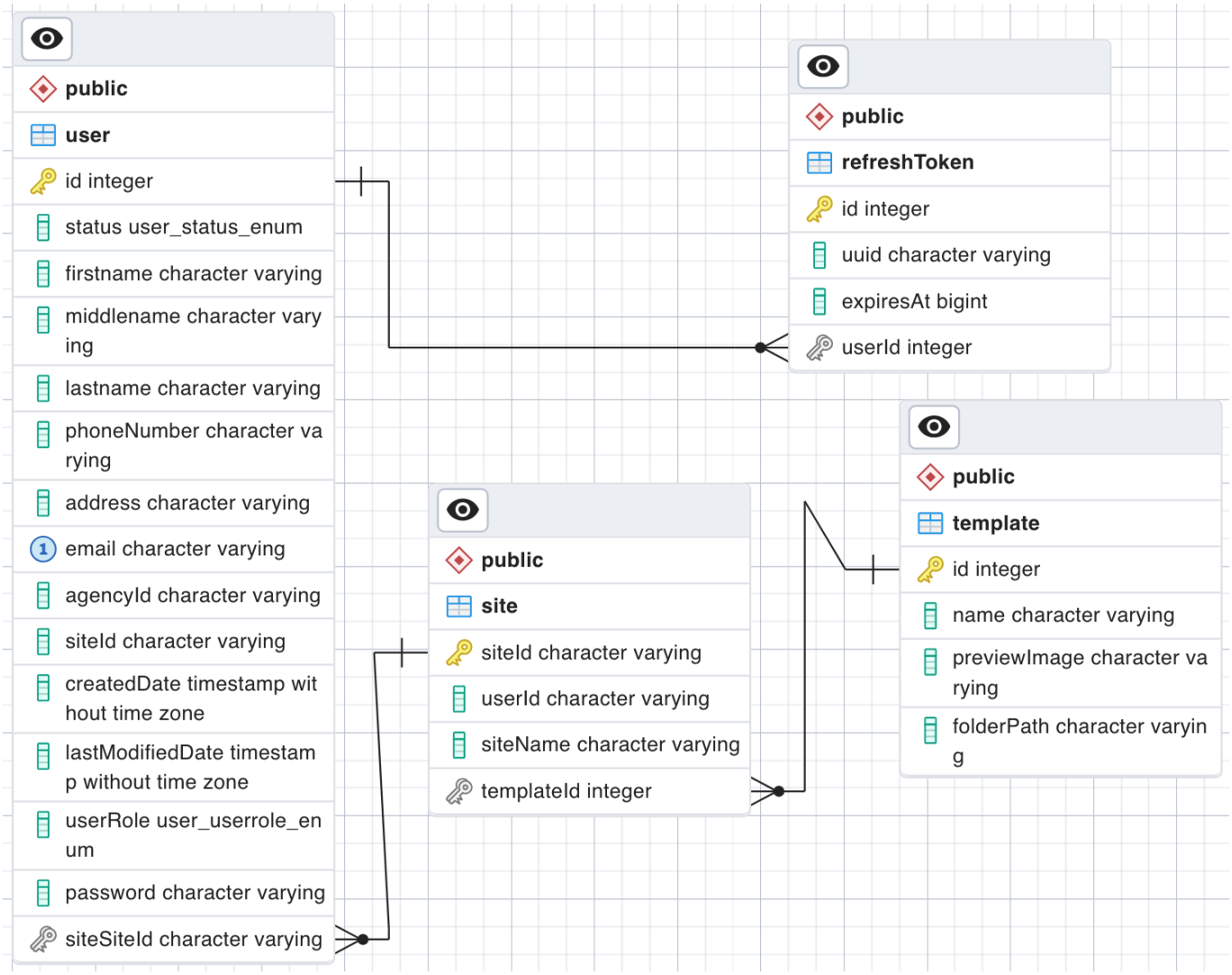

Рисунок 3.3 – Модель бази даних

3.3. Моделювання варіантів використання

В цьому підрозділі зображено діаграма варіантів використання системи (рисунок 3.4). Серед основних акторів було обрано працівника агентства, адміністратора сайту та базу даних.

Працівник агентства взаємодіє з базою даних для реєстрації свого облікового запису у системі та авторизації, а також для виконання основних функцій сайту, серед яких створення та редагування сайту.

Адміністратор сайту за допомогою взаємодії з базою даних може створювати, редагувати та видаляти облікові записи користувачів, а також додавати, редагувати та видаляти шаблони сайтів, на основі яких працівники агентства мають можливість створювати нові проекти.

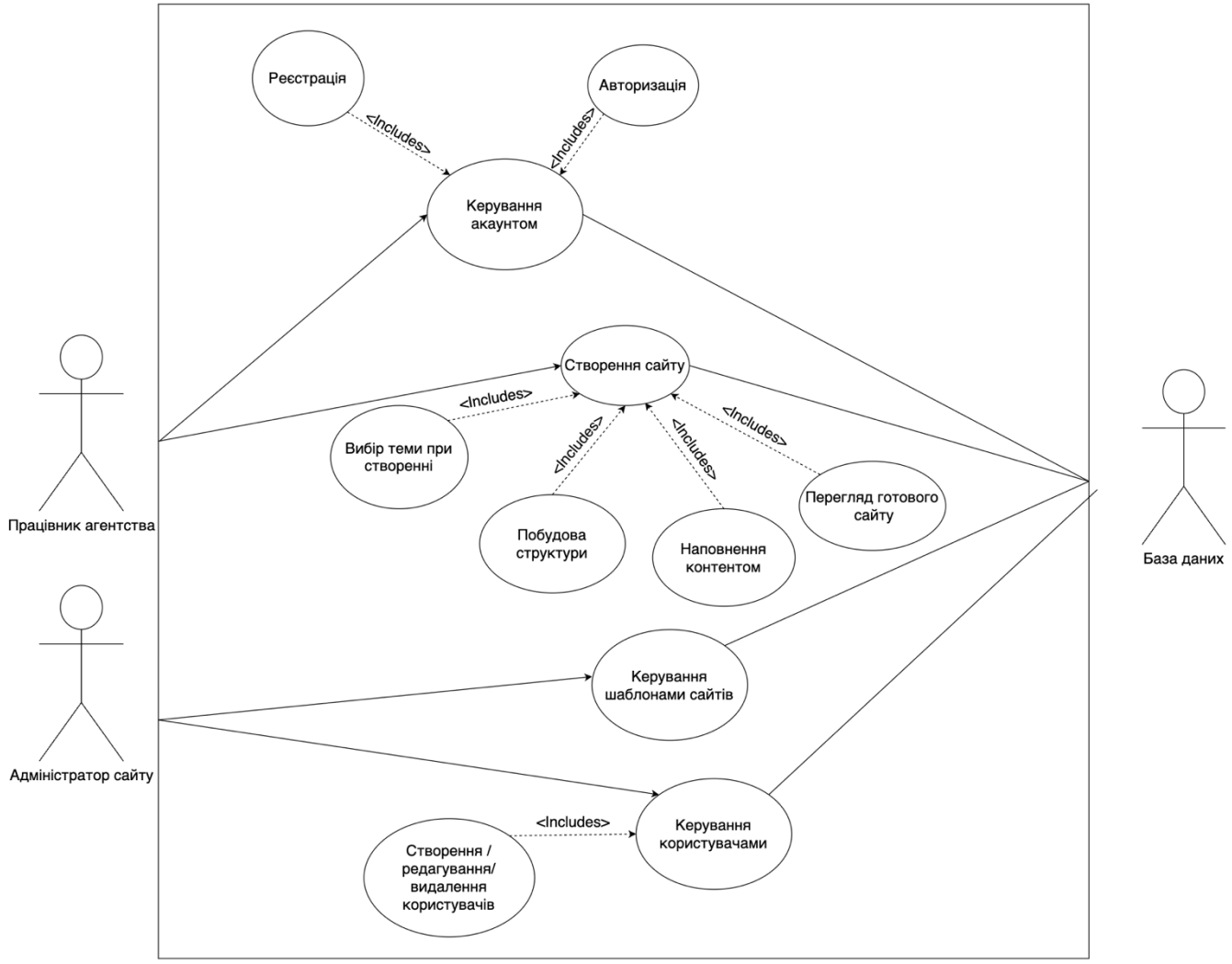

Рисунок 3.4 – Модель варіантів використання системи

4. ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1. Розробка програмного продукту

Згідно з задачами до проекту та функціональними вимогами для розробки web-орієнтованої системи було обрано наступні засоби:

Мова. Основною мовою для створення системи було обрано TypeScript, яка є розширенням до JavaScript з додаванням типізації. Типізації дає розробникам більше можливостей для опису потрібних модулів, класів та інтерфейсів, що робить систему більш структурованою та зрозумілою.

Серверна частина. Для реалізації серверної частини було обрано фреймворк NestJS, серед переваг якого є модульність та масштабованість, добре структурований Dependency Injection та підтримка REST API.

Клієнтська частина. Клієнтська частина реалізована за допомогою NuxtJS. Оскільки цей фреймворк підтримує рендеринг на стороні сервера (SSR), досвід користувача буде кращим, бо відображення контенту під час конструювання сайтів буде більш швидким. Також цей фреймворк надає змогу побудови в тому числі і статичних сайтів, які є кінцевим продуктом web-орієнтованої системи, що розроблюється.

Бази даних. Для побудови основних взаємодій в додатку було обрано реляційну систему керування базами даних PostgreSQL, оскільки вона надає добре структуроване і надійне сховище даних. Для зберігання об’єктів, що містять дані про контент сайтів користувачів було використано документну базу даних MongoDB.

Для запуску серверу було використано функцію bootstrap() фреймоворку NestJS (фр. коду 4.1.)

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule, {
    logger: LoggerConfigFactory(process.env.LOG_LEVEL),
  });
  app.useGlobalPipes(new ValidationPipe());
  const config = app.get(ConfigService);
  const port = config.get<string>(ConfigConstants.port) || 5000;
```

```
  const micrositeAssetsPath = config.get<string>(
    ConfigConstants.micrositesAssetsPath,
  );
  const staticAssetsRoute = config.get<string>(
    ConfigConstants.staticAssetsRoute,
  );

  app.use(`/${staticAssetsRoute}`, express.static(micrositeAssetsPath));

  await app.listen(port, () => console.log(`Server started on PORT: ${port}`));

  const url = await app.getUrl();

  console.log('running on url:', url);
}
bootstrap();
```

Фрагмент коду 4.1 – Запуск серверу

Далі, для підключення до баз даних, було створено два модулі, MongoModule (фр. коду 4.2.) та PostgresModule (фр. коду 4.3.)

```
MongooseModule.forRootAsync({
    imports: [ConfigModule],
    useFactory: async (configService: ConfigService) => ({
      uri: configService.get<string>(ConfigConstants.mongodbURI),
      dbName: configService.get<string>(ConfigConstants.mongoDbName),
      useNewUrlParser: true,
      useUnifiedTopology: true,
    }),
    inject: [ConfigService],
  }),
```

Фрагмент коду 4.2 – MondoModule

```
TypeOrmModule.forRootAsync({
    inject: [ConfigService],
    useFactory: (configService: ConfigService) =>
      ({
        type: 'postgres' as const,
        host: configService.get<string>(ConfigConstants.typeOrmHost),
        port: Number(configService.get<string>(ConfigConstants.typeOrmPort)),
        username: configService.get<string>(ConfigConstants.typeOrmUsername),
        password: configService.get<string>(ConfigConstants.typeOrmPassword),
        database: configService.get<string>(ConfigConstants.typeOrmDatabase),
        entities: [
          ...
        ],
        synchronize: Boolean(
          configService.get<string>(ConfigConstants.typeOrmSynchronize),
        ),
```

```
        logging:
          configService.get<string>(ConfigConstants.typeOrmLogging) === 'true'
            ? true
            : false,
        } as TypeOrmModuleOptions),
    }),
```

Фрагмент коду 4.3 – PostgresModule

Також, згідно із стандартами RESTful архітектури, було побудовано модулі кінцевих точок доступу до серверу. Для прикладу візьмемо модуль керування створенням сайтів та його основні операції створення (фр. коду 4.4.), отримання (фр. коду 4.5.), редагування (фр. коду 4.6.) та видалення (фр. коду 4.7.) сайтів.

```
@Post('site')
  async addSite(
    @Body() addSiteParams: AddSiteDto,
    @Request() req: RequestDto,
  ): Promise<Microsite> {
    const addedSite: Microsite = await this.micrositeService.addSite(
      addSiteParams,
      req.user.id,
    );

    return addedSite;
  }
```

Фрагмент коду 4.4 – Створення сайту

```
@Get('sites/:id')
  async getMicrosite(
    @Param('id') id: string,
    @Request() req: RequestDto,
  ): Promise<Microsite> {
    await this.micrositeService.isAuthorizedUser(id, req.user.id);

    const microsite: Microsite = await this.micrositeService.getMicrosite(id);

    return microsite;
  }
```

Фрагмент коду 4.5 – Отримання даних сайту

```
@Put('section')
  async updateSection(
    @Body() section: UpdateSectionDto,
    @Request() req: RequestDto,
  ): Promise<Section> {
    await this.micrositeService.isAuthorizedUser(section.siteId, req.user.id);

    const updatedSection: Section = await this.micrositeService.updateSection(
      section,
    );

    return updatedSection;
  }
```

Фрагмент коду 4.6 – Редагування вмісту сайту

```
Delete('site')
  async deleteSite(@Query('id') id: string) {
    await this.micrositeService.deleteSite(id);
  }
```

Фрагмент коду 4.7 – Видалення сайту

Для створення та керування даними про користувачів було побудовано модуль адміністратора.

```
@Post('user')
  async addUser(@Body() user: AddUserDto): Promise<User | null> {
    const addedUser = await this.adminService.addUser(user, false);

    return addedUser;
  }
```

Фрагмент коду 4.8 – Створення користувача

```
@Get('users')
  async getAllUsers(): Promise<User[]> {
    const users = await this.adminService.getAllUsers();

    return users;
  }
```

Фрагмент коду 4.9 – Отримання списку усіх користувачів

```
@Get('users/:id')
  async getUserById(@Param('id') id: string): Promise<User> {
    const user = await this.adminService.getUserById(id);

    return user;
  }
```

Фрагмент коду 4.10 – Отримання конкретного користувача

```
@Put('user')
  async updateUser(@Body() user: UpdateUserDto): Promise<User> {
    const updatedUser = await this.adminService.updateUser(user);

    return updatedUser;
  }
```

Фрагмент коду 4.11 – Редагування даних користувача

```
@Delete('user')
  async deleteUser(
    @Query('id') id: string,
    @Request() req: RequestDto,
  ): Promise<void> {
    if (req.user.id.toString() !== id.toString()) {
      await this.adminService.deleteUser(id);
    }

    return;
  }
```

Фрагмент коду 4.12 – Видалення користувача

Для сутності користувача було створено інтерфейс User (фр. коду 4.8.)

```
export default interface User {
  id: string;
  status: UserStatus;
  firstname: string;
  middlename: string;
  lastname: string;
  phoneNumber: string;
  address: string;
  email: string;
  agencyId: string;
  siteId?: string;
  createdDate: string;
  lastModifiedDate: string;
  userRole: UserRole;
  password: string;
  refreshTokens?: RefreshToken[];
  }
```

Фрагмент коду 4.13 – Інтерфейс User

На клієнтській частині для взаємодії із сервером було використано плагін Axios.

```
import { NuxtAxiosInstance } from '@nuxtjs/axios'

let $axios: NuxtAxiosInstance

export function initializeAxios(axiosInstance: NuxtAxiosInstance) {
  $axios = axiosInstance
}

export { $axios }
```

Фрагмент коду 4.14 – Підключення плагіну Axios

Точкою входу в клієнтський додаток у фреймворці Nuxt є директорія Pages (сторінки), яка власне і задає структуру шляхів web-системи. В додатку вона має наступний вигляд:

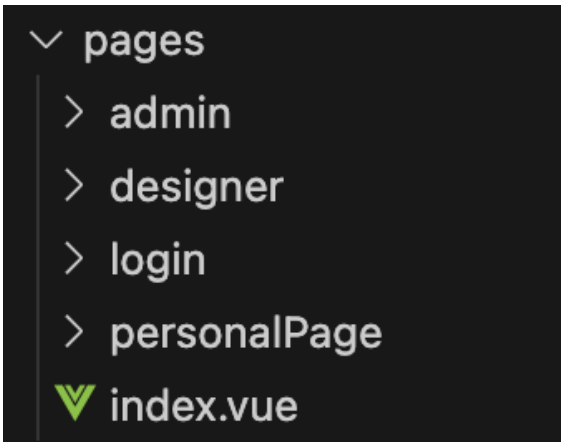

Рисунок 4.1 – Структура web-система

Директорія Admin. Містить сторінки для актора ролі адміністратора, серед яких керування користувачами, шаблонами сайтів.

Директорія Designer. Містить всі сторінки для створення і конструювання сайтів, серед яких є основна сторінка редагування сайту та суміжні підсторінки.

Директорія Login. Містить сторінки реєстрації та авторизації користувачів.

Директорія PersonalPage. Складена з однієї основної сторінки, яка демонструє список сайтів користувача (працівника агентства)

Index.vue. Вітальна сторінка сайту, яка демонструє користувачу доступні йому дії.

Початком дій користувача є авторизація, форма якої наведена далі (рисунок 4.2.)

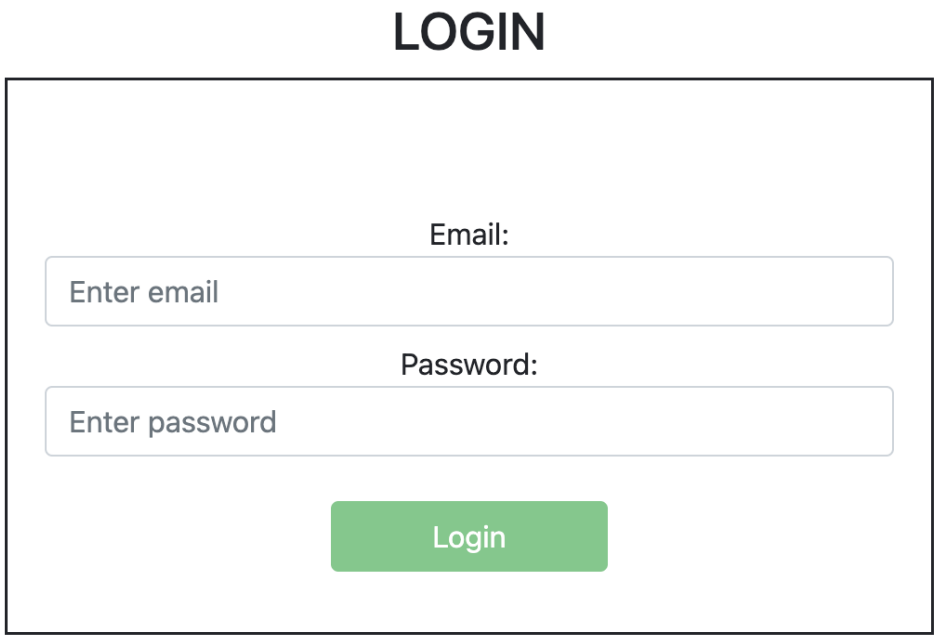

Рисунок 4.2 – Форма авторизації користувача

Далі користувач потрапляє на сторінку із сайтами (рисунок 4.3.), які були створені ним. Для додавання нового сайту потрібно натиснути кнопку BUILD A NEW SITE. На рисунку 4.4. наведено форму створення нового сайту.

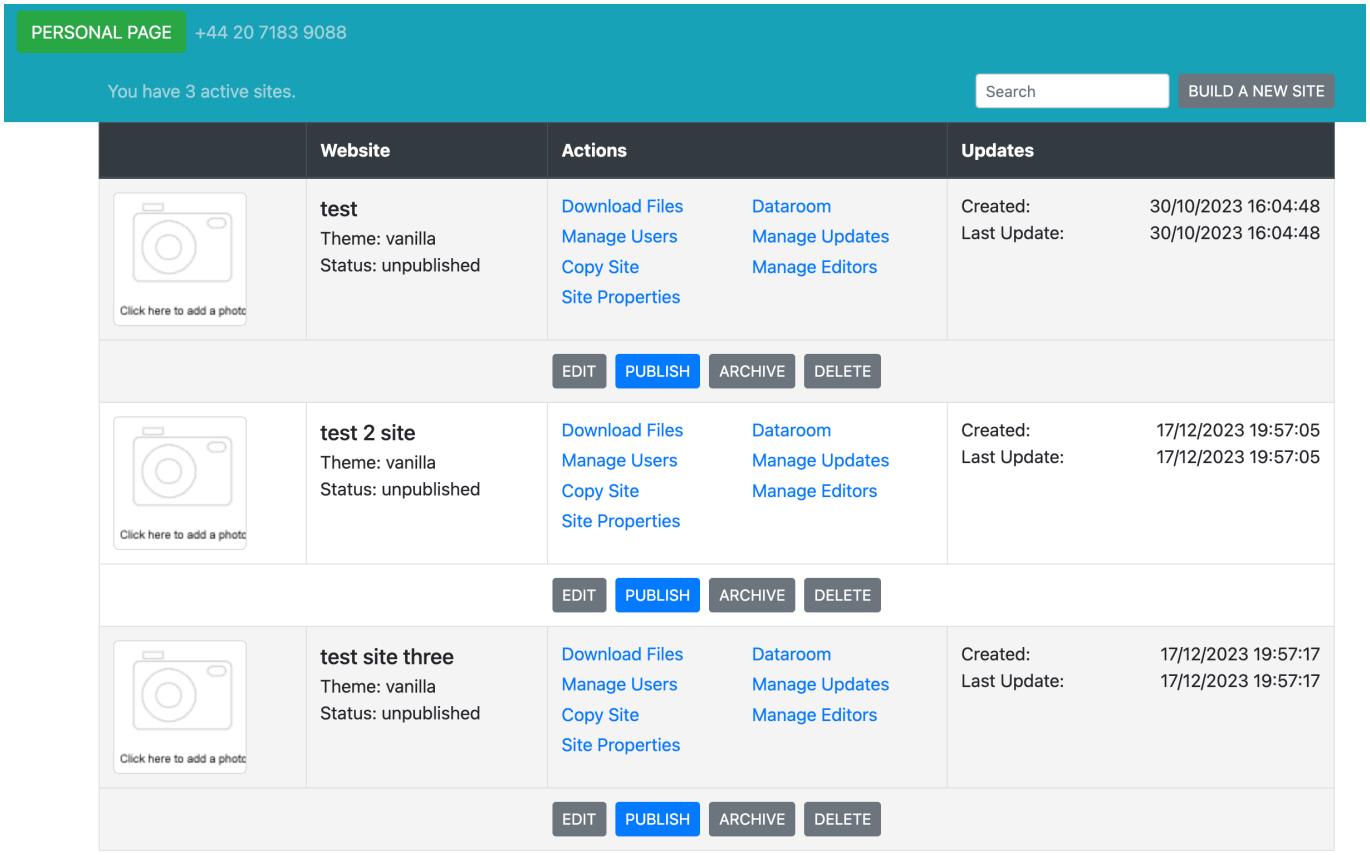

Рисунок 4.3 – Сторінка із сайтами користувача

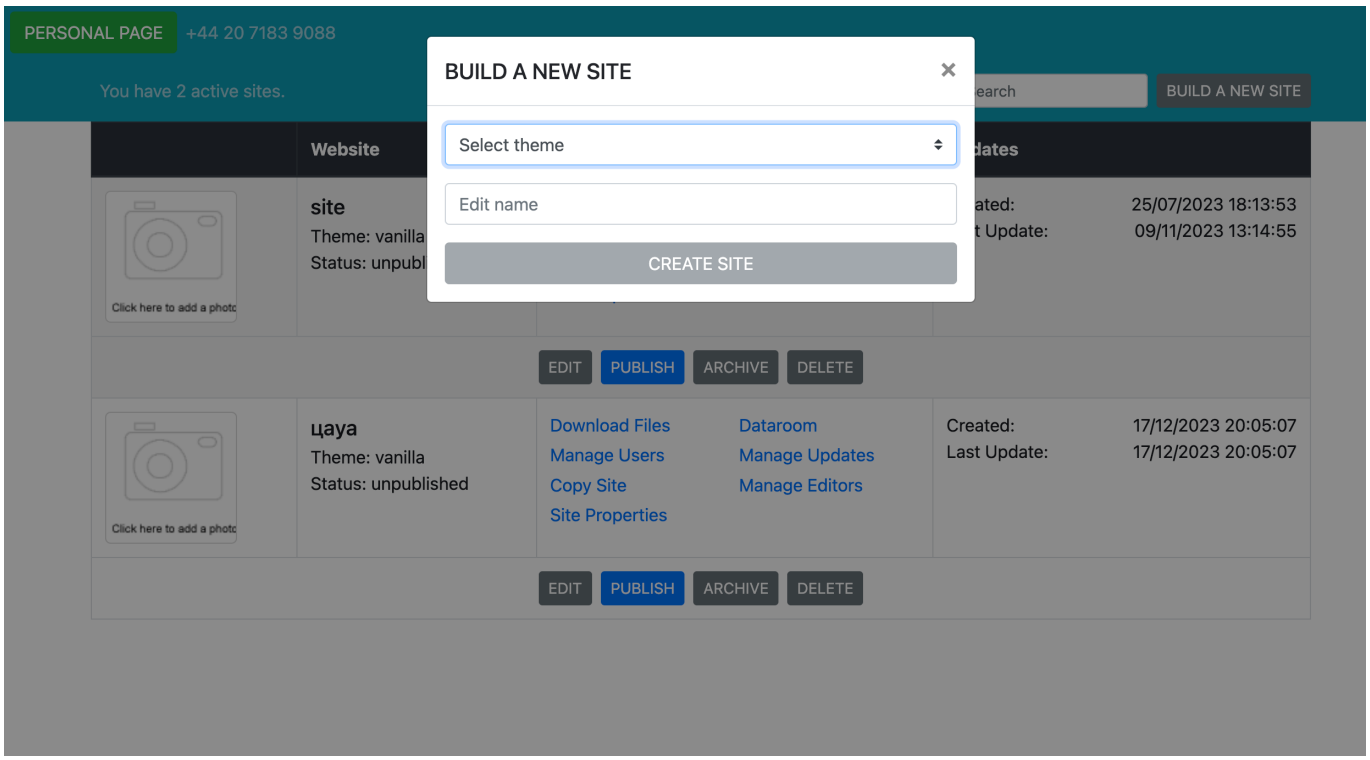

Рисунок 4.4 – Додавання нового сайту

Для редагування створеного сайту за допомогою натискання на кнопку Edit користувач потрапляє на сторінку конструювання (рисунок 4.5.)

Рисунок 4.5 – Сторінка конструювання сайту

Сторінка конструювання поділена на 3 зони: список елементів для додавання на сайт (ліворуч), макет сайту(праворуч знизу) та кнопки керування сайтом (праворуч зверху). При натисканні на елемент з’являється меню редагування елементу (рисунок 4.6., рисунок 4.7.)

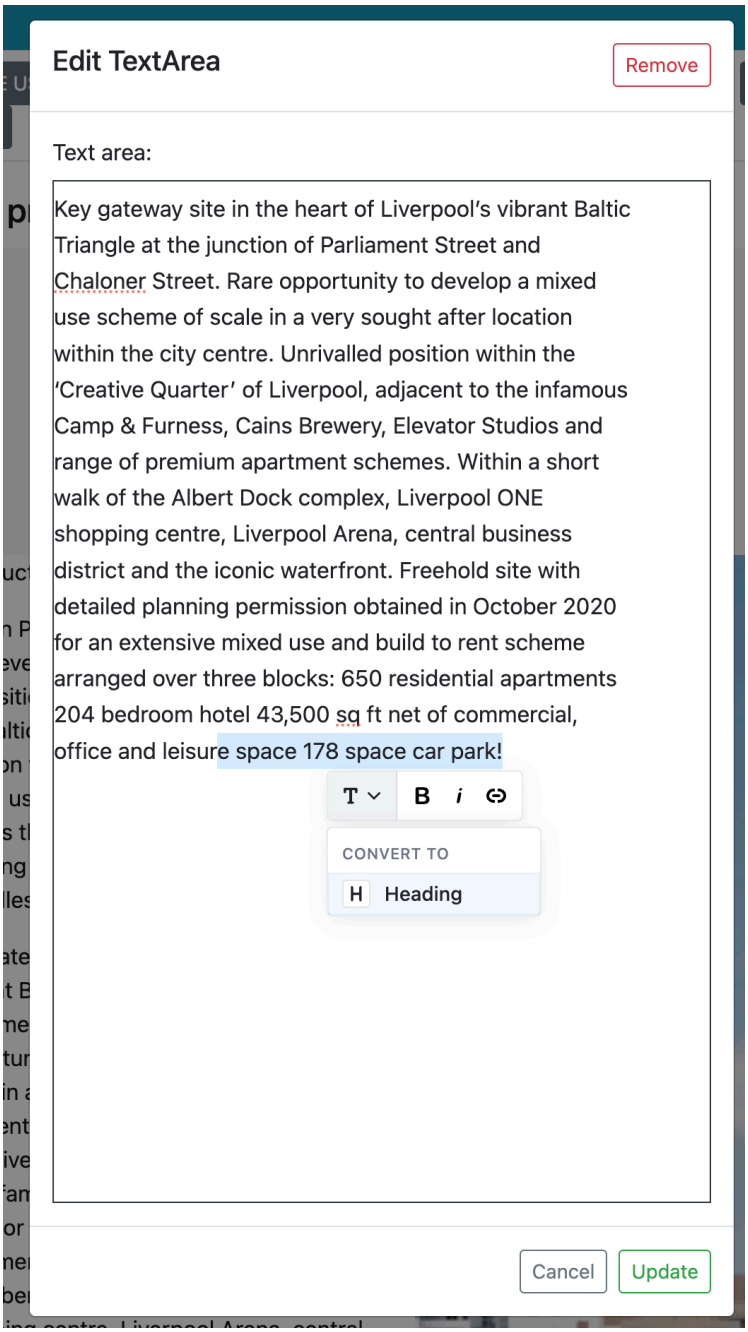

Рисунок 4.6 – Редагування текстового елементу

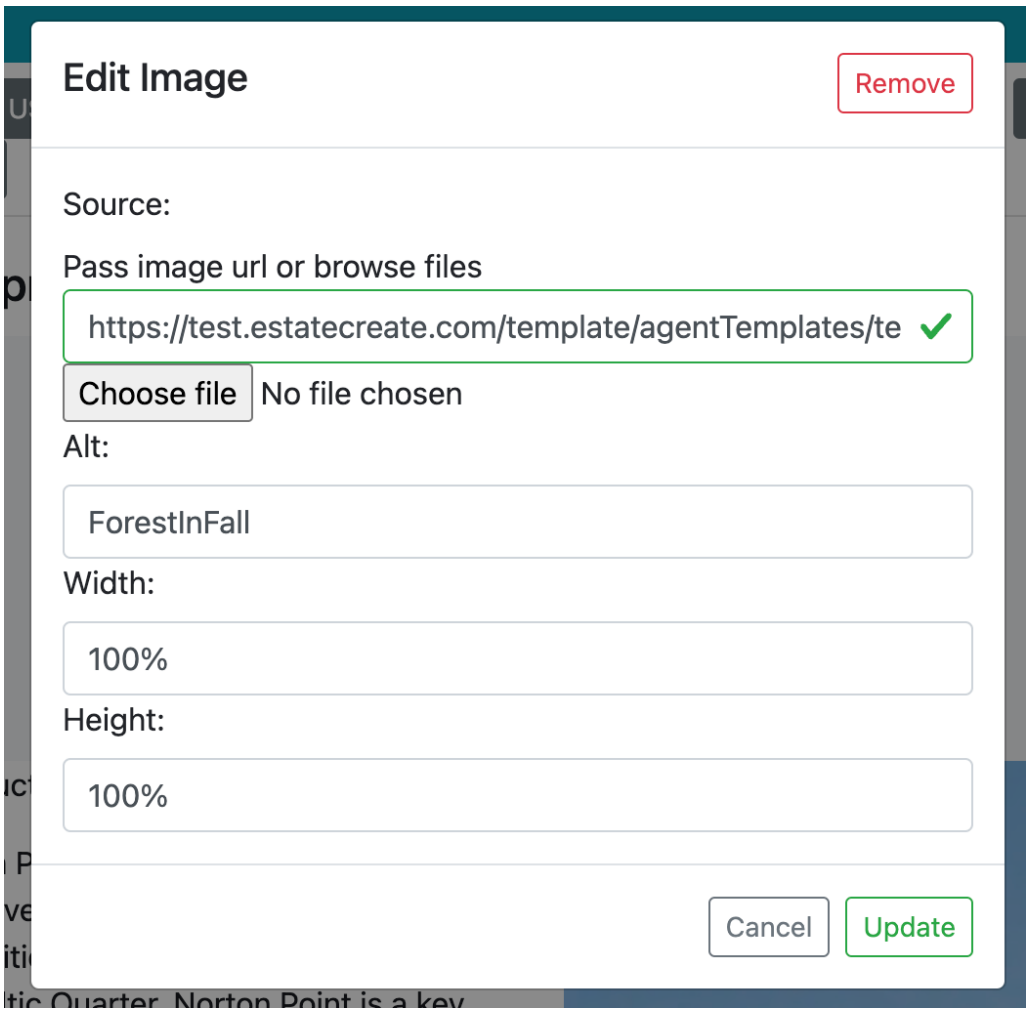

Рисунок 4.7 – Редагування елементу з зображенням

Додати новий елемент на сайт можливо двома способами:

- натисканням на елемент із списку. В такому випадку елемент додається в зону контенту в кінець списку.
- перетасуванням елементу в потрібну зону (Drag And Drop). Таким чином можна додати елемент в потрібне місце потрібної зони макету сайту.

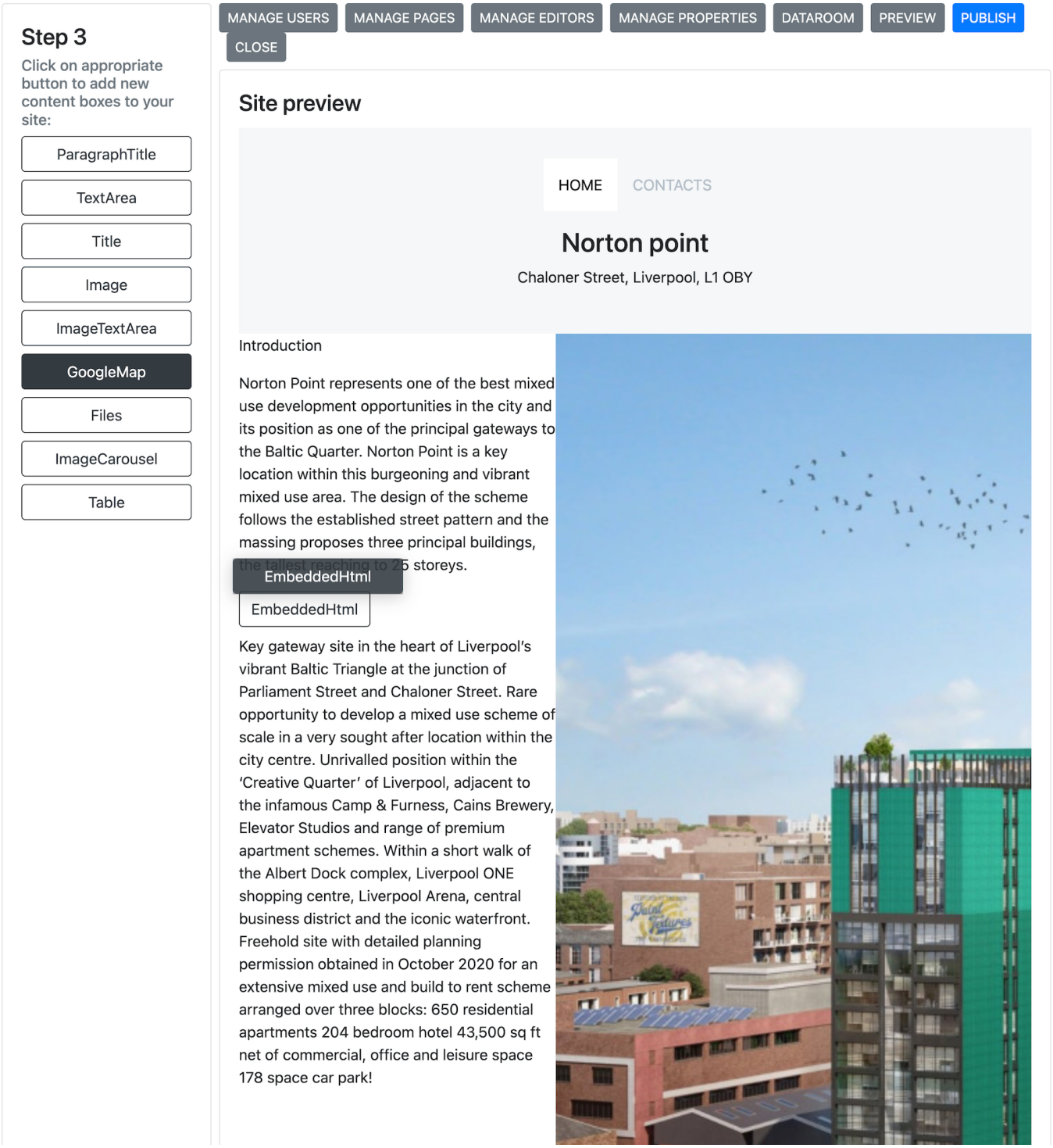

Рисунок 4.8 – Додавання елементу за допомогою перетаскування

Система надає можливість попереднього перегляду створеного сайту за допомогою натискання кнопки PREVIEW.

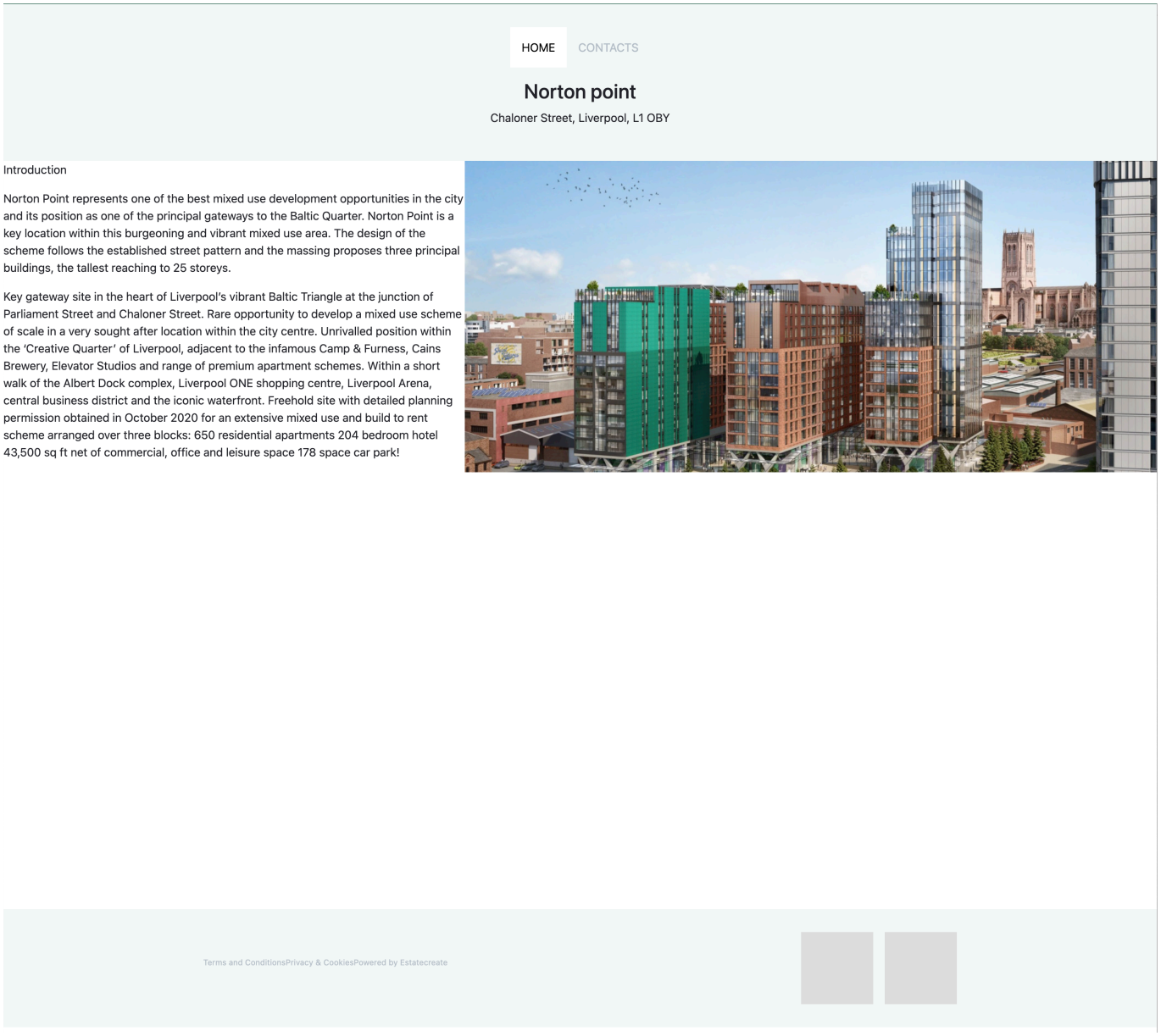

Рисунок 4.9 – Попередній перегляд сайту у режимі PREVIEW

Одною з головних можливостей web-орієнтованої системи є публікація сайту за допомогою кнопки PUBLISH. В такому випадку сконструйований сайт компілюється на сервері та перетворюється в статичний, завантажується на платформу AWS (Amazon Web Services).

```
public async publishSite(site: Microsite): Promise<PublishedSiteInfo> {
  const micrositePath = join(this.micrositesPath, site.id);
  const isSiteInitialized = await exists(join(micrositePath, 'node_modules'));
  if (!isSiteInitialized) {
    await this.micrositeBuilder.installSiteDependencies(
      micrositePath,
      site.id,
```

```
    );
  }
  await this.setConfigToStatic(micrositePath, site.name, site.id);
  await this.generateStaticSite(micrositePath, site.id);
  await this.uploadStaticSite(site.id);
  let newPublishedSiteInfo: PublishedSiteInfo;

  if (site.publishedSiteInfo && site.publishedSiteInfo.distributionId) {
    newPublishedSiteInfo = await this.enablePublishedSite(
      site.publishedSiteInfo,
    );
  } else {
    newPublishedSiteInfo = await this.createCloudfrontDistribution(site.id);
  }
  return newPublishedSiteInfo;
}
```

Фрагмент коду 4.15 – Публікація сайту

Для зберігання даних статичного сайту використовується AWS S3 (об’єктне сховище).

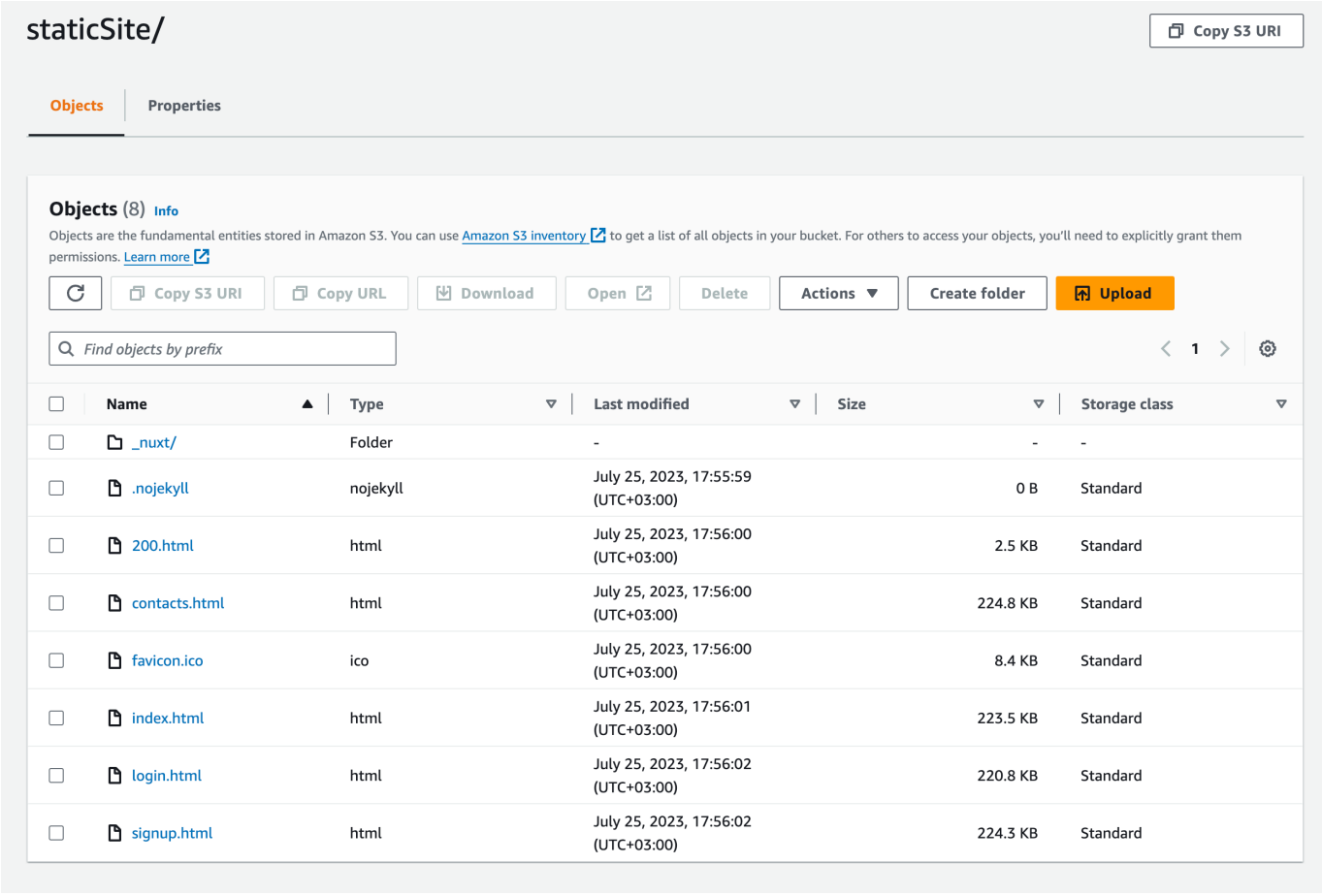

Рисунок 4.10 – Статичний сайт на AWS S3

Для доступу до даного сайту використовується AWS CloudFront. Статичний сайт формується як дистрибуція, отримує власну URL адресу.

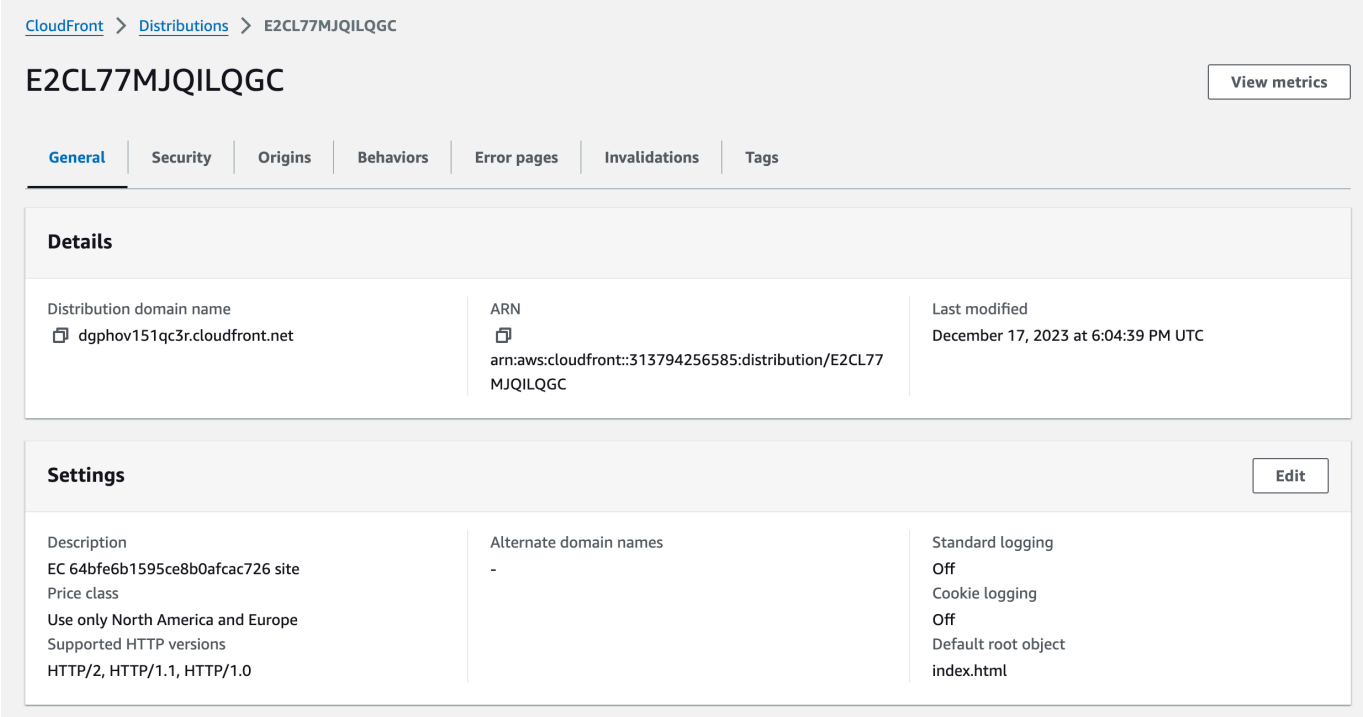

Рисунок 4.11 – Дистрибуція статичного сайту на AWS Cloudfront

4.2. Тестування програмного продукту

Для тестування стабільної роботи системи було проведено перевірку за наступними критеріями:

- Цілісність підключення між клієнтом та сервером
- Цілісність передачі даних між клієнтом та сервером
- Стабільність поведінки системи при втраті зв’язку між клієнтом та сервером
- Стабільність поведінки системи при навантаженні на клієнт або сервер

Рисунок 4.12 – Характеристики апаратної системи для тестування

Цілісність підключення між клієнтом та сервером.

Було програмно створено структуру вхідних точок серверної REST архітектури. За допомоги цієї структури з потрібними валідними даними було проведено послідовне навантаження серверу за допомогою додатку Postman. Далі ці вхідні точки було використано за допомогою клієнтської частини. За результатами тестування цілісності підключення було встановлено, що середній час відгуку на запит не відрізняється в залежності від способу виконання запиту, тому підключення між клієнтом та сервером є цілісним.

Цілісність передачі даних між клієнтом та сервером.

За допомогою браузеру Google Chrome та Safari було протестовано зв’язок між клієнтом та сервером. Згідно із даними із браузерного розширення InspectConnection було сформовано результат, що незалежно від браузеру виконання, встановлення з’єднання клієнтом та сервером, відправка запиту до

серверу, отримання відповіді на клієнті та багатократне повторення попередніх дій видає цілісний стабільний результат із стабільною середньою швидкістю.

Стабільність поведінки системи при втраті зв'язку між клієнтом та сервером.

Було протестовано сценарій, в якому було здійснено відправку запиту з клієнтської частини до відключеного серверу. Було отримано результат, що система вірно інформує користувача про некоректність роботи серверу та не виконує неконтрольовані транзакції дій.

Стабільність поведінки системи при навантаженні на клієнт або сервер.

Було протестовано систему на стабільність поведінки системи при навантаженні. Задля цього було використано додаток Postman. Додаток симулював виконання одночасних запитів від 100, 200, 500, 1000 та 3000 користувачів. На заданій апаратній системі для тестування сервер забезпечував стабільну роботу системи на рівні 100, 200 та 1000 запитів. Наступне підвищення кількості користувачів підвищувало витрату ресурсів, значно подовжуючи час відгуку. Результат свідчить про достатній рівень стабільності поведінки системи під навантаженням.

ВИСНОВКИ

У ході дослідження, що було присвячене web-орієнтованій системі підтримки конструювання сайтів для дизайн-агентств, було проведено детальний аналіз методів та технологій, спрямованих на спрощення і прискорення комплексного процесу конструювання веб-ресурсів за допомогою сучасних інструментів, імплементованих у вигляді казуального інтерфейсу.

Перед створенням програмного забезпечення було проведено глибоке дослідження предметної області web-розробки, аналіз існуючих аналогів та технологій, що потрібні для створення бажаної системи. На базі отриманої інформації було сформовано мету, завдання для дослідженні та функціональні вимоги до системи.

Як результат, за допомогою мови TypeScript, фреймворків NestJS та NuxtJS та систем керування базами даних MongoDB та PostgreSQL було розроблено та реалізовано серверну та клієнтську частини web-орієнтованої системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ulrik Soderstrom, Lovisa Carlsson, and Thomas Mejtoft. 2019. Comparing Millennials View on Minimalism and Maximalism in Web Design. In Proceedings of the 31st European Conference on Cognitive Ergonomics (ECCE '19). Association for Computing Machinery, New York, NY, USA, 92–95. DOI:10.1145/3335082.3335104
2. Zimmermann, G., Strobbe, C., Ziegler, D. (2019). Inclusive Responsiveness – Why Responsive Web Design Is Not Enough and What We Can Do About This. In: Di Bucchianico, G. (eds) Advances in Design for Inclusion. AHFE 2018. Advances in Intelligent Systems and Computing, vol 776. Springer, Cham. DOI: 10.1007/978-3-319-94622-1_20
3. Kainz, O., Nečeda, S., Michalko, M., Murin, M., & Nováková, I. (2023). Educational Web Solution for Cyber Security. *2023 21st International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 270-276.
4. Lovakumari, M., & Priyadarshini, C. (2022). Novel Network Testbed of Adaptive Cyber Defense System Design for Web Data security. *International Journal of Advanced Research in Science, Communication and Technology*.
5. M. Lydia, G. Edwin Prem Kumar & A. Immanuel Selvakumar (2023) Securing the cyber-physical system: a review, Cyber-Physical Systems, 9:3, 193-223, DOI: 10.1080/23335777.2022.2104378
6. Lavrov, E., Pasko, N., Krivodub, A. Automated analysis of ergonomic measures in discrete control systems (2015) Eastern-European Journal of Enterprise Technologies, 4 (3), pp. 16-22. DOI: 10.15587/1729-4061.2015.48050
7. Mary Deaton. 2003. The elements of user experience: user-centered design for the Web. interactions 10, 5 (September + October 2003), 49–51. DOI: 10.1145/889692.889709
8. M. K. Mittal, N. Kirar and J. Meena, "Implementation of Search Engine Optimization : Through White Hat Techniques," 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), Greater Noida, India, 2018, pp. 674-678, DOI: 10.1109/ICACCCN.2018.8748337.

9. Fielding, R.T., & Taylor, R.N. (2000). Principled design of the modern Web architecture. *Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium*, 407-416.
10. Panda, S.K., Swain, S.K., Mall, R. (2015). Measuring Web Site Usability Quality Complexity Metrics for Navigability. In: Jain, L., Patnaik, S., Ichalkaranje, N. (eds) Intelligent Computing, Communication and Devices. Advances in Intelligent Systems and Computing, vol 308. Springer, New Delhi. DOI:10.1007/978-81-322-2012-1_41
11. Duka M, Sikora M, Strzelecki A. From Web Catalogs to Google: A Retrospective Study of Web Search Engines Sustainable Development. *Sustainability*. 2023; 15(8):6768. DOI: 10.3390/su15086768
12. Park, C.W., Macinnis, D.J., & Eisingerich, A.B. (2016). Brand architecture design and brand naming decisions. DOI: 10.4324/9781315796789.ch8
13. Doosti, B., Crandall, D.J., & Su, N.M. (2017). A Deep Study into the History of Web Design. *Proceedings of the 2017 ACM on Web Science Conference*.
14. Rukshan Alexander, David Murray, and Nik Thompson. 2017. Cross-Cultural Web Design Guidelines. In Proceedings of the 14th International Web for All Conference (W4A '17). Association for Computing Machinery, New York, NY, USA, Article 10, 1–4. DOI: 10.1145/3058555.3058574
15. Krüger, Z. (2023). The Art of SXO: Placing UX Design Methods into SEO Best Practices. *The Art of SXO*. DOI:10.1007/978-1-4842-9212-9
16. Laure Philips, Coen De Roover, Tom Van Cutsem, and Wolfgang De Meuter. 2014. Towards Tierless Web Development without Tierless Languages. In Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software (Onward! 2014). Association for Computing Machinery, New York, NY, USA, 69–81. DOI:10.1145/2661136.2661146
17. Lin, J., Sayagh, M., & Hassan, A.E. (2022). The Co-evolution of the WordPress Platform and Its Plugins. *ACM Transactions on Software Engineering and Methodology, 32*, 1 - 24.
18. Ana B. Sánchez, Sergio Segura, and Antonio Ruiz-Cortés. 2014. The Drupal framework: a case study to evaluate variability testing techniques. In Proceedings of the

8th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS '14). Association for Computing Machinery, New York, NY, USA, Article 11, 1–8. DOI: 10.1145/2556624.2556638

19. Wakode, B., & Chaudhari, D.N. (2013). STUDY OF CONTENT MANAGEMENT SYSTEMS JOOMLA AND DRUPAL. *International Journal of Research in Engineering and Technology, 02*, 569-573.
20. Spirintsev, V. (2022). Choosing the optimal environment for visual development of a graphical user interface. *System technologies*. DOI: 10.34185/1562-9945-1-138-2022-07
21. Wang, J., Xu, Z., Wang, X., & Lu, J. (2022). A Comparative Research on Usability and User Experience of User Interface Design Software. *International Journal of Advanced Computer Science and Applications*.
22. A. Kumar, A. Kumar, H. Hashmi and S. A. Khan, "WordPress: A Multi-Functional Content Management System," 2021 10th International Conference on System Modeling & Advancement in Research Trends (SMART), MORADABAD, India, 2021, pp. 158-161, DOI: 10.1109/SMART52563.2021.9675311.
23. Sharma, G., Kute, A., Jadhav, N., Kolhe, N., & Tiwari, A. (2022). Review of Web Content Management Systems and their Increasing Demand in Market. *International Journal for Research in Applied Science and Engineering Technology*.
24. Pillai, A., Shinohara, K., & Tigwell, G.W. (2022). Website Builders Still Contribute To Inaccessible Web Design. Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility.
25. Martin, S. (2017). Introduction to Squarespace. In: The Definitive Guide to Squarespace. Apress, Berkeley, CA. DOI: 10.1007/978-1-4842-2937-8_1
26. Thakur, B.K. (2022). Analysis & Comparative study of LAMP CMS. *INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT*. DOI: 10.55041/ijsrem16601
27. Gary Dushnitsky, Bryan K. Stroube, Low-code entrepreneurship: Shopify and the alternative path to growth, Journal of Business Venturing Insights, Volume 16, 2021, e00251, ISSN 2352-6734, DOI: 10.1016/j.jbvi.2021.e00251.

28. C. Machado and J. C. Campos, "Towards the integration of user interface prototyping and model-based development," 2021 International Conference on Graphics and Interaction (ICGI), Porto, Portugal, 2021, pp. 1-8, DOI: 10.1109/ICGI54032.2021.9655284.

29. V. Kompaniets, A. Lyz and A. Kazanskaya, "An Empirical Study of Goal Setting in UX/UI-design," 2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT), Tashkent, Uzbekistan, 2020, pp. 1-5, DOI: 10.1109/AICT50176.2020.9368570.

30. G. Goel, P. Tanwar and S. Sharma, "UI-UX Design Using User Centred Design (UCD) Method," 2022 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2022, pp. 1-8, DOI: 10.1109/ICCCI54379.2022.9740997.

31. Lavrov, E., Pasko, N., Siryk, O., Mukoseev, V., Dubovyk, S. Automation of reliability assessment of functional elements of flexible automated production based on functional network methodology CEUR Workshop Proceedings, 2020, 2740, pp. 357–364

32. Todorov, T., & Dochkova-Todorova, J. (2023). Accessible UX/UI Design. *2023 International Conference Automatics and Informatics (ICAI)*, 362-366.

33. Rajesh.P, Selvadurai.M, Saranya Selvamani.V, Chandrasekar.P(2022). Fundamentals of UX/UI (An Approach to Design Principles) (1st ed., pp. 1-162). Jupiter Publications consortium,ISBN:978-93-91303-38-9, DOI: 10.47715/JPC.B.87.2022.9789391303389

34. N. A. Hidayah, Zulfiandri, A. Rafiiuddin, Y. Durachman and E. Rustamaji, "User Experience Design Analysis Using Lean UX Method," 2021 9th International Conference on Cyber and IT Service Management (CITSM), Bengkulu, Indonesia, 2021, pp. 1-6, DOI: 10.1109/CITSM52892.2021.9588905.

35. Єгорова , І. ., & Ільченко , К. . (2022). ОСОБЛИВОСТІ ТЕСТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА. *Вісник Національного технічного університету «ХПІ». Серія: Нові рішення у сучасних технологіях*, (4(14), 18–23. DOI:10.20998/2413-4295.2022.04.03

36. Cooper A. About Face: The Essentials of Interaction Design, Wiley, 2014, no. 4, 720 p., DOI: 10.1057/palgrave.ivs.950066.

37. Ostrovskaya, K.Y., Guda, A.N., & Romanyuk, K. (2022). Research and development of the method design UX / UI design internet – platforms. System technologies.

38. Tsunajima, T., Nishiuchi, N. (2020). Visual Environment Design of VR Space for Sequential Reading in Web Browsing. In: Stephanidis, C., Antona, M. (eds) HCI International 2020 - Posters. HCII 2020. Communications in Computer and Information Science, vol 1225. Springer, Cham. DOI:10.1007/978-3-030-50729-9_18

39. L. d. l. Torre, J. Chacon, D. Chaos, R. Heradio and R. Chandramouli, "Using IoT-Type Metadata and Smart Web Design to Create User Interfaces Automatically," in IEEE Transactions on Industrial Informatics, vol. 19, no. 3, pp. 3109-3118, March 2023, DOI: 10.1109/TII.2022.3186638.

40. Song, L., & García-Valls, M. (2022). Improving Security of Web Servers in Critical IoT Systems through Self-Monitoring of Vulnerabilities. *Sensors (Basel, Switzerland), 22*.

41. Eric York. 2023. Evaluating ChatGPT: Generative AI in UX Design and Web Development Pedagogy. In Proceedings of the 41st ACM International Conference on Design of Communication (SIGDOC '23). Association for Computing Machinery, New York, NY, USA, 197–201. DOI: 10.1145/3615335.3623035

42. Sridevi, S., Karpagam, G.R., Kumar, B.V., & Umamaheswari, J. (2021). Investigation on Blockchain Technology for Web Service Composition: A Case Study. *Int. J. Web Serv. Res., 18*, 1-23.

Додаток А

Планування робіт

У галузі дизайну веб-сайтів та розвитку онлайн-простору помітний стрімкий ріст попиту на системи підтримки конструювання сайтів. Лише нещодавно створення веб-сайту було особливим заходом, а сьогодні це щоденний процес для багатьох компаній. Зростаюча кількість замовників у цій галузі свідчить про те, що дизайнери веб-сайтів і клієнти цінують можливість швидко та ефективно створювати візуально привабливі та функціональні сайти завдяки вдосконаленим системам конструювання. Крім того, такі системи не лише сприяють створенню веб-ресурсів, але й надають можливості для спілкування, співпраці, роботи над проектами та навіть особистого розвитку фахівців в цій галузі. Тому постійне удосконалення систем підтримки конструювання сайтів позитивно впливатиме на продуктивність дизайн-агентств та комфорт користувачів, які використовують ці системи для розробки веб-проектів.

Деталізація мети проекту методом SMART.

Таблиця А.1 – Деталізація мети проекту методом SMART

Specific(конкретна)	Створення Web-орієнтована система підтримки конструювання сайтів для дизайн-агентств із необхідним функціоналом, що описаний у вимогах технічного завдання.
Measurable (вимірювана	Створений web-додаток, який складається із 6 сторінок: вітальна сторінка, авторизація, реєстрація, перелік сайтів, що проектуються, конструктор сторінок проекту, конструктор дизайну сторінок.
Achievable (досяжна, узгоджена)	Для успішної реалізації проекту є підтверджене технічне завдання, володіння спеціалізованими знаннями і вміннями, а також доступ до використання потрібних

	технологій: HTML, CSS, JavaScript, TypeScript, NodeJS, Nuxt, Nest, SQL
Relevant (реалістична)	Створений сервіс спростить процес конструювання і дизайну веб-додатків як для замовників, так і для виконавців, спростить комунікацію між ними.
Time-framed (обмежена в часі)	Строк досягнення цілі проекту узгоджено між замовником та виконавцем як завершення до кінця другого курсу магістратури (30 грудня 2023 року).

Планування змісту робіт. WBS (Work Breakdown Structure), або структура розбиття робіт, є графічним відображенням елементів проекту, які організовані в ієрархічну структуру для створення єдиного цілого – продукту проекту. Ця структура розподілу завдань спрямована на розбиття робіт на частини, які є ключовими складовими проекту і сприяють організації командної роботи. Елементи цієї розбитої структури можуть представляти продукти, дані або послуги. Крім того, WBS надає основу для детальної оцінки термінів і контролю за графіком робіт. На найвищому рівні розташовано основний продукт проекту. Основні дії та заходи, необхідні для досягнення мети проекту, відображені на різних рівнях цієї розбитої структури. Розбиття робіт триває до того моменту, поки вони не стають елементарними, тобто досягнення кожного завдання має чітко визначений результат і відповідального виконавця, що дозволяє розрахувати витрати робочого часу та час виконання.

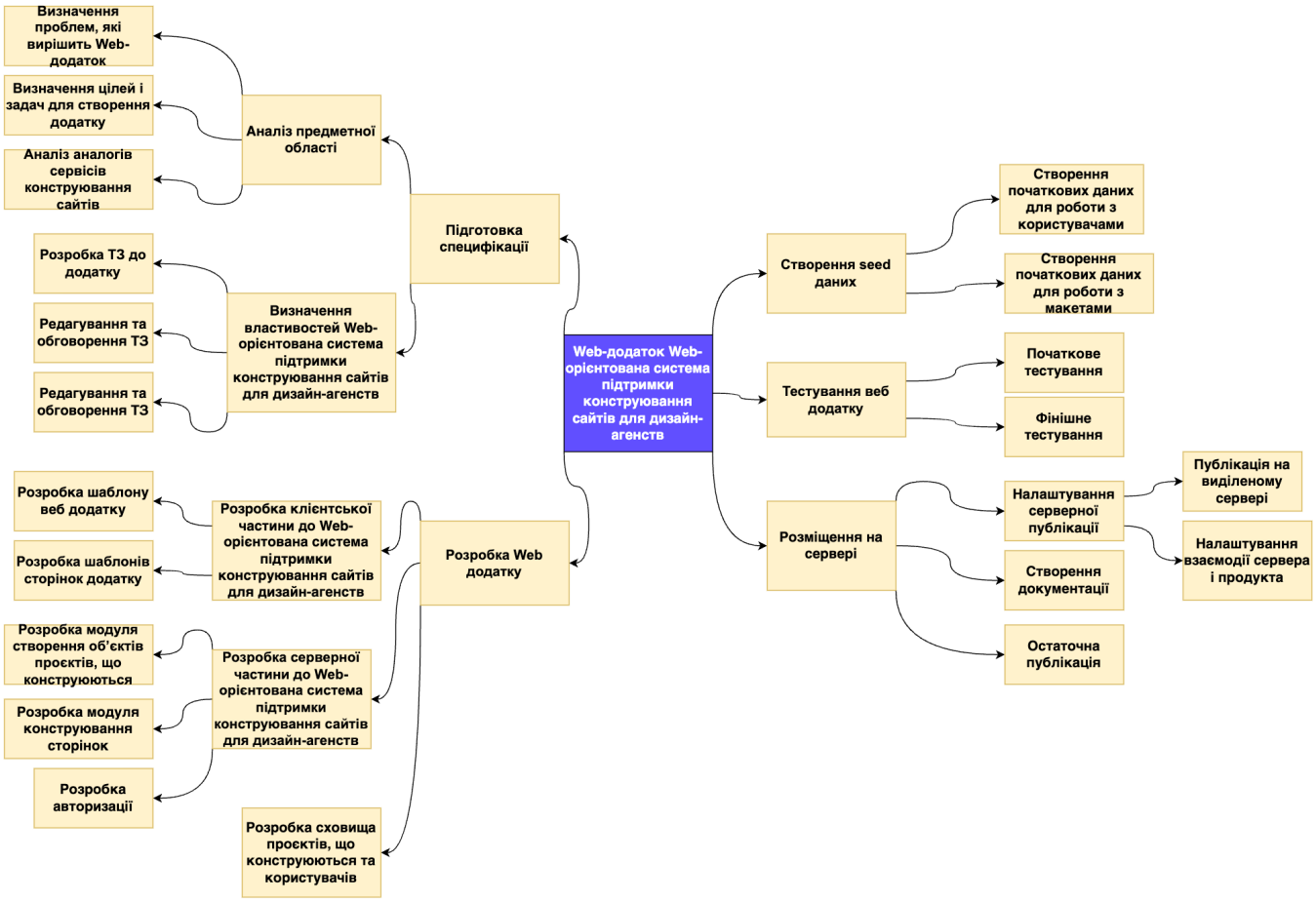

Рисунок А.1 - WBS-структура робіт

Планування структури виконавців. Наступним етапом після розбиття процесів є розробка організаційної структури виконавців, або OBS. Це графічне відображення учасників або відповідальних осіб, які беруть участь у реалізації проекту. Ролі цих осіб включають в себе організацію та виконання елементарної роботи, яка визначена у WBS. Кожна елементарна робота може розглядатися як окремий проект.

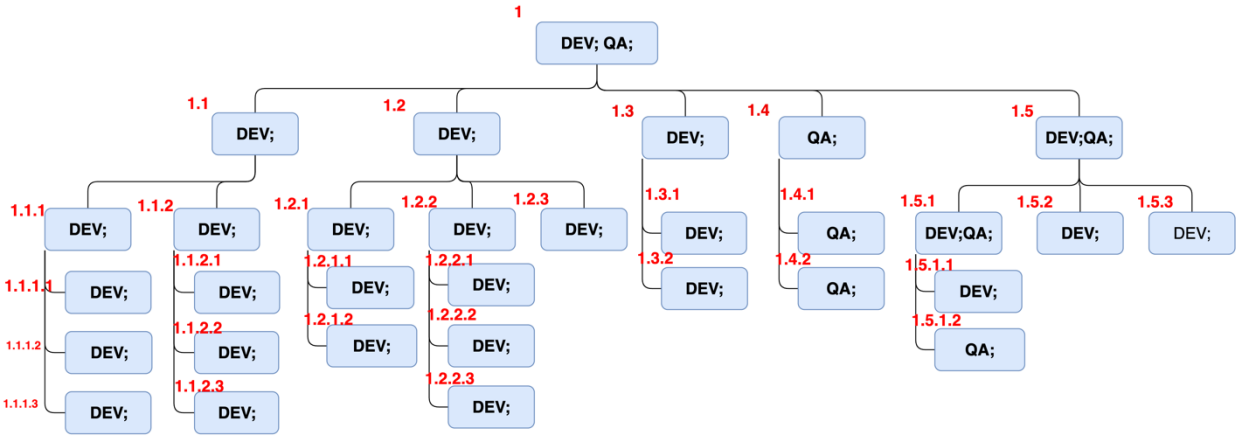

Рисунок А.2 - OBS-структура робіт

Таблиця А.2 – Виконавці проекту

Роль	**ПІБ**	**Проєктна роль**
Розробник (DEV)	Василець Д.А.	Виконання розробки серверної та клієнтської частин.
Тестувальник (QA)	Василець Д.А.	Тестування функціоналу та дизайну продукту.
Керівник проєкту	Лавров Є.А.	Формулювання завдань на розробку проєкту.

Побудова календарного графіку.

Назва задачі	Тривалість	Початок	Кінець
Web-додаток Web-орієнтована система підтримки конструювання сайтів для дизайн-агенств	**90 днів**	**04.09.23**	**01.12.23**
Підготовка специфікації	**12 днів**	**04.09.23**	**16.09.23**
Аналіз предметної області	**6 днів**	**04.09.23**	**10.09.23**
Визначення проблем, які вирішить Web-додаток	1 день	04.09.23	05.09.23
Визначення цілей і задач для створення додатку	2 дні	05.09.23	07.09.23
Аналіз аналогів сервісів конструювання сайтів	3 дні	07.09.23	10.09.23
Визначення властивостей Web-орієнтована система підтримки конструювання сайтів для дизайн-агенств	**5 днів**	**11.09.23**	**16.09.23**
Розробка ТЗ до додатку	3 дні	11.09.23	13.09.23
Редагування та обговорення ТЗ	1 день	14.09.23	14.09.23
Затвердження ТЗ	1 день	15.09.23	16.09.23
Розробка Web додатку	**45 днів**	**17.09.23**	**30.10.23**
Розробка клієнтської частини до Web-орієнтована система підтримки конструювання сайтів для дизайн-агенств	**16 днів**	**17.09.23**	**03.10.23**
Розробка шаблону веб додатку	2 дні	17.09.23	19.09.23
Розробка шаблонів сторінок додатку	14 днів	20.09.23	03.10.23
Розробка серверної частини до Web-орієнтована система підтримки конструювання сайтів для дизайн-агенств	**21 днів**	**04.10.23**	**25.10.23**
Розробка модуля створення об'єктів проєктів, що конструюються	7 днів	04.10.23	10.10.23
Розробка модуля конструювання сторінок	7 днів	11.10.23	17.10.23
Розробка авторизації	7 днів	18.10.23	25.10.23
Розробка сховища проєктів, що конструюються та користувачів	**5 днів**	**26.10.23**	**30.10.23**
Створення seed даних	**14 днів**	**31.10.23**	**14.11.23**
Створення початкових даних для роботи з користувачами	**7 днів**	**31.10.23**	**06.11.23**
Створення початкових даних для роботи з макетами	**7 днів**	**07.11.23**	**14.11.23**
Тестування веб додатку	**10 днів**	**15.11.23**	**25.11.23**
Початкове тестування	**7 днів**	**15.11.23**	**20.11.23**
Фінішне тестування	**5 днів**	**21.11.23**	**25.11.23**
Розміщення на сервері	**5 дні**	**26.11.23**	**01.12.23**
Налаштування серверної публікації	**2 дні**	**26.11.23**	**27.11.23**
Публікація на виділеному сервері	1 день	26.11.23	26.11.23
Налаштування взаємодії сервера і продукта	1 день	27.11.23	27.11.23
Створення документації	**1 день**	**28.11.23**	**28.11.23**
Остаточна публікація	**2 дні**	**29.11.23**	**01.12.23**

Рисунок А.3 – Календарний графік проєкту

Управління ризиками проекту. Під час проведення якісної оцінки ризиків потрібно визначити ті з них, які потребують негайного усунення. Реакція на ризики буде відповідати їхній ступеню важливості. Після цього виконується кількісне

оцінювання ризиків. Кількісна та якісна оцінка можуть здійснюватися паралельно чи окремо, що залежить від потреб проекту в забезпеченні безпеки.

Таблиця А.3 – Рейтинг ризиків з урахуванням ймовірності виникнення та впливу.

Оцінка	**Ймовірність виникнення**	**Вплив ризику**	**Тип ризику**
1	Низька	Низький	Прийнятний
2	Середня	Середній	Виправданий
3	Висока	Високий	Недопустимий

Таблиця А.4 – Матриця ймовірності та впливу

Ймовірність виникнення ризику	**Вплив ризику**				
	0,05	**0,1**	**0,2**	**0,4**	**0,8**
0,9	0,045	0,09	0,18	0,36	0,72
0,7	0,035	0,07	0,14	0,28	0,56
0,5	0,025	0,05	0,10 **R3, R4**	0,20	0,40
0,3	0,015	0,03 **R1**	0,06	0,12	0.24
0,1	0,005	0,01	0,02	0,04	0,08 **R2**

Таблиця А.5 – Таблиця оцінювання ризику за рівнем

№	Назва	Межі
1	Прийнятні	$0,005 \leq R \leq 0,05$
2	Виправдані	$0,05 < R \leq 0,14$
3	Недопустимі	$0,14 < R \leq 0,72$

Таблиця А.6 – Ризики та стратегії реагування на них

ID	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	Тип стратегії реагування	План А	План Б
RS_1	Новий	Розбіжності між керівником проекту (або командою) та замовником.	Низька	Середній	0.02	Зменшення	1. Покращити взаємини між замовником та менеджером проєкту. 2. Дотримуватися правил ділового етикету під час спілкування. 3. Забезпечити комфортні	При виявленні непорозумінь важливо уточнити корінь проблеми, обговорити її та відновити (створити) здорову атмосферу у відносинах.

							умови для співпраці.	
RS_2	Новий	З'явлення альтернативного продукту.	Низька	Високий	0.08	Прийняття	1. Здійснити попереднє дослідження стану розробки альтернативних продуктів. 2. Обрати унікальну стратегію для розробки власного проєкту.	
RS_3	Новий	Малопрецизне завдання для розробки.	Середня	Середній	0.1	Зменшення	1. Чітко обговорити з замовником всі вимоги і створити технічне	У випадку виявлення невідповідностей деяких характеристик програмно

							завдання, що буде затверджене обома сторонами. 2. Сформувати глосарій для уникнення розбіжностей у розумінні термінів. 3. Забезпечити періодичний контроль замовником за етапами роботи.	го продукту заявленим вимогам важливо уважно й чітко визначити, що було виконано неправильно та внести відповідні корективи.
RS_4	Відкритий	Обмежена кваліфіка	Середня	Середній	0.1	Зменшення	1. Постійно підвищу	Замінити співробітн

		ція учасників проектного колективу.					вати кваліфікацію персоналу через участь у спеціалізованих курсах з чітко визначеною періодичністю. 2. Регулярно оновлювати знання з використанням онлайн-ресурсів, що відповідають сучасним трендам.	ика на більш досвідченого або, використовуючи резерв часу, перегляну ти розподіл завдань у проєкті і змінити виконавців, щоб врахувати зміни в дедлайнах та додати додаткові обов'язки вже існуючому члену проєктної команди.

Додаток Б

Лістинг програмного коду

```
import {
  forwardRef,
  Logger,
  MiddlewareConsumer,
  Module,
  NestModule,
} from '@nestjs/common';
import { MicrositeController } from './microsites.controller';
import { MicrositeBuilder } from './microsites-builder.service';
import { MicrositeService } from './microsites.service';
import { MicrositeControlsTemplatesService } from './microsites-control-templates.service';
import { MicrositeSiteThemeService } from './microsites-theme.service';
import MicrositeDbModule from './database/microsite.db.module';
import { MicrositePreviewService } from './microsites-preview.service';
import { APP_FILTER } from '@nestjs/core';
import { AllExceptionsFilter } from 'src/error/exception.filter';
import { MicrositePreviewProxyMiddleware } from 'src/middlewares/microsite-preview-
proxy.middleware';
import { PreviewConstants } from 'src/utils/constants/preview.constants';
import AdminDbModule from 'src/admin/database/admin.db.module';
import { MicrositePublishService } from './microsites-publish.service';
import { FilesModule } from 'src/files/files.module';
import SQSModule from 'src/sqs/sqs.module';

@Module({
  imports: [
    MicrositeDbModule,
    AdminDbModule,
    forwardRef(() => SQSModule),
    forwardRef(() => FilesModule),
  ],
  controllers: [MicrositeController],
  providers: [
    MicrositeService,
    MicrositeBuilder,
    MicrositeSiteThemeService,
    MicrositeControlsTemplatesService,
    MicrositePreviewService,
    {
      provide: APP_FILTER,
      useClass: AllExceptionsFilter,
    },
    Logger,
    MicrositePublishService,
  ],
  exports: [MicrositeService, MicrositePublishService],
})
export class MicrositeModule implements NestModule {
  configure(consumer: MiddlewareConsumer) {
```

```
    consumer
      .apply(MicrositePreviewProxyMiddleware)
      .forRoutes(PreviewConstants.micrositePreviewResource);
  }
}



import { Logger, Module } from '@nestjs/common';
import SQSModule from 'src/sqs/sqs.module';
import { AdminController } from './admin.controller';
import { AdminService } from './admin.service';
import AdminDbModule from './database/admin.db.module';

@Module({
  imports: [AdminDbModule, SQSModule],
  controllers: [AdminController],
  providers: [AdminService, Logger],
  exports: [AdminService],
})
export class AdminModule {}



import { ConfigService } from '@nestjs/config';
import { Logger, Module } from '@nestjs/common';
import { JwtModule } from '@nestjs/jwt';
import { PassportModule } from '@nestjs/passport';
import { LocalStrategy } from './strategies/local.strategy';
import { JwtStrategy } from './strategies/jwt.strategy';
import { AuthService } from './auth.service';
import AdminDbModule from '../admin/database/admin.db.module';
import { ConfigConstants } from 'src/utils/constants/config.constants';
import { AuthController } from './auth.controller';
import { JwtRefreshTokenStrategy } from './strategies/refresh.strategy';
import { ScheduleModule } from '@nestjs/schedule';
import { AdminService } from 'src/admin/admin.service';
import SQSModule from 'src/sqs/sqs.module';

@Module({
  imports: [
    AdminDbModule,
    PassportModule,
    JwtModule.registerAsync({
      useFactory: (configService: ConfigService) => {
        return {
          secret: configService.get<string>(
            ConfigConstants.jwtAccessTokenSecretKey,
          ),
          signOptions: {
            expiresIn: configService.get<string | number>(
              ConfigConstants.jwtAccessTokenExpirationTime,
```

```
            ),
          },
        };
      },
      inject: [ConfigService],
    }),
    ScheduleModule.forRoot(),
    SQSModule,
  ],
  providers: [
    AuthService,
    LocalStrategy,
    JwtStrategy,
    Logger,
    ConfigService,
    JwtRefreshTokenStrategy,
    AdminService,
  ],
  controllers: [AuthController],
  exports: [AuthService],
})
export class AuthModule {}


import { Module } from '@nestjs/common';
import { AuthModule } from 'src/auth/auth.module';
import { FilesModule } from 'src/files/files.module';
import { CronsAuthService } from './crons.auth.service';
import { CronsFilesService } from './crons.files.service';
import { CronsService } from './crons.service';

@Module({
  imports: [FilesModule, AuthModule],
  providers: [CronsService, CronsFilesService, CronsAuthService],
})
export class CronsModule {}


import { format, transports } from 'winston';
import DailyRotateFile = require('winston-daily-rotate-file');
import {
  utilities as nestWinstonModuleUtilities,
  WinstonModule,
} from 'nest-winston';

const { printf } = format;

const humanReadableFormatter = printf(
  ({ level, message, context, timestamp }) => {
    return `${timestamp} ${level.toLocaleUpperCase()} [${context}] ${message}`;
  },
);
```

```
const logFormatter = format.combine(
  format.errors({ stack: true }),
  format.splat(),
  format.timestamp({ format: 'YYYY-MM-DDTHH:mm:ss.sssZ' }),
  humanReadableFormatter,
  format.uncolorize(),
);

const transportConsole = new transports.Console({
  format: format.combine(
    format.timestamp({ format: 'YYYY-MM-DD, HH:mm:ss' }),
    nestWinstonModuleUtilities.format.nestLike(),
  ),
});

const transportError: DailyRotateFile = new DailyRotateFile({
  filename: 'error-log-%DATE%.log',
  dirname: 'logs/error',
  json: false,
  datePattern: 'YYYY-MM-DD',
  zippedArchive: true,
  handleExceptions: true,
  maxSize: '20m',
  maxFiles: '30d',
  level: 'error',
});

export const LoggerConfigFactory = (logLevel: string) => {
  switch (logLevel) {
    case 'info': {
      return WinstonModule.createLogger({
        level: logLevel,
        format: logFormatter,
        transports: [
          new DailyRotateFile({
            filename: 'info-log-%DATE%.log',
            dirname: 'logs/info',
            json: false,
            datePattern: 'YYYY-MM-DD',
            zippedArchive: true,
            maxSize: '20m',
            maxFiles: '30d',
            level: 'info',
          }),
          transportError,
          transportConsole,
        ],
      });
    }

    case 'debug': {
```

```
      return WinstonModule.createLogger({
        level: logLevel,
        format: logFormatter,
        transports: [
          new DailyRotateFile({
            filename: 'debug-log-%DATE%.log',
            dirname: 'logs/debug',
            json: false,
            datePattern: 'YYYY-MM-DD',
            zippedArchive: true,
            maxSize: '20m',
            maxFiles: '30d',
            level: 'debug',
          }),
          transportError,
          transportConsole,
        ],
      });
    }

    case 'verbose': {
      return WinstonModule.createLogger({
        level: logLevel,
        format: logFormatter,
        transports: [
          new DailyRotateFile({
            filename: 'verbose-log-%DATE%.log',
            dirname: 'logs/verbose',
            json: false,
            datePattern: 'YYYY-MM-DD',
            zippedArchive: true,
            maxSize: '20m',
            maxFiles: '30d',
            level: 'verbose',
          }),
          transportError,
          transportConsole,
        ],
      });
    }

    default: {
      return WinstonModule.createLogger({
        format: logFormatter,
        transports: [transportConsole],
      });
    }
  }
};


import { Module } from '@nestjs/common';
```

```
import { ConfigModule, ConfigService } from '@nestjs/config';
import { MongooseModule } from '@nestjs/mongoose';
import { ConfigConstants } from 'src/utils/constants/config.constants';
import {
  MongoMicrosite,
  MongoMicrositeSchema,
} from './schemas/microsite.schema';

@Module({
  imports: [
    MongooseModule.forRootAsync({
      imports: [ConfigModule],
      useFactory: async (configService: ConfigService) => ({
        uri: configService.get<string>(ConfigConstants.mongodbURI),
        dbName: configService.get<string>(ConfigConstants.mongoDbName),
        useNewUrlParser: true,
        useUnifiedTopology: true,
      }),
      inject: [ConfigService],
    }),
    MongooseModule.forFeature([
      { name: MongoMicrosite.name, schema: MongoMicrositeSchema },
    ]),
  ],

  exports: [
    MongooseModule.forFeature([
      { name: MongoMicrosite.name, schema: MongoMicrositeSchema },
    ]),
  ],
})
export class MongoModule {}



import { Module } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { TypeOrmModule, TypeOrmModuleOptions } from '@nestjs/typeorm';
import { ConfigConstants } from 'src/utils/constants/config.constants';
import CorsOriginEntity from './entities/cors-origin.entity';
import RefreshTokenEntity from './entities/refresh-token.entity';
import RestrictedFileControlEntity from './entities/restricted-file-control.entity';
import RestrictedPageEntity from './entities/restricted-page.entity';
import SiteEntity from './entities/site.entity';
import TemplateEntity from './entities/template.entity';
import UserEntity from './entities/user.entity';
import DataroomNodeEntity from './entities/dataroom-node.entity';
import DataroomActionEntity from './entities/dataroom-action.entity';

@Module({
  imports: [
    TypeOrmModule.forRootAsync({
```

```
      inject: [ConfigService],
      useFactory: (configService: ConfigService) =>
        ({
          type: 'postgres' as const,
          host: configService.get<string>(ConfigConstants.typeOrmHost),
          port: Number(configService.get<string>(ConfigConstants.typeOrmPort)),
          username: configService.get<string>(ConfigConstants.typeOrmUsername),
          password: configService.get<string>(ConfigConstants.typeOrmPassword),
          database: configService.get<string>(ConfigConstants.typeOrmDatabase),
          entities: [
            UserEntity,
            SiteEntity,
          ],
          synchronize: Boolean(
            configService.get<string>(ConfigConstants.typeOrmSynchronize),
          ),
          logging:
            configService.get<string>(ConfigConstants.typeOrmLogging) === 'true'
              ? true
              : false,
        } as TypeOrmModuleOptions),
    }),
    TypeOrmModule.forFeature([
      UserEntity,
      SiteEntity,
    ]),
  ],
  exports: [
    TypeOrmModule.forFeature([
      UserEntity,
      SiteEntity,
    ]),
  ],
})
export default class TypeORMModule {}



import Microsite from '../interfaces/microsite.interface';
import { Page } from '../interfaces/page.interface';
import { DeletePageDto } from '../dto/delete-page.dto';
import { UpdatePageDto } from '../dto/update-page.dto';
import { CopyPageDto } from '../dto/copy-page.dto';
import { UserMicrositeDto } from '../dto/user-microsites.dto';
import { UpdateControlDto } from '../dto/update-control.dto';
import { Control } from '../interfaces/control.interface';
import { AddControlDto } from '../dto/add-control.dto';
import { DeleteControlDto } from '../dto/delete-control.dto';
import { UpdateSectionDto } from '../dto/update-section.dto';
import { Section } from '../interfaces/section.interface';
import PublishedSiteInfo from '../interfaces/published-site-info.interface';
import { UpdateRegistrationOptionsDto } from '../dto/update-reg-options.dto';
```

```
import { SiteStatus } from '../enums/site-status.enum';
import { UpdateDataroomInfoDto } from '../dto/update-dataroom-info.dto';

export default abstract class MicrositeDatabase {
  abstract getSite(siteId: string): Promise<Microsite>;

  abstract getUserMicrosites(
    siteIdArray: string[],
    userId: string,
  ): Promise<UserMicrositeDto[]>;

  abstract addSite(site: Microsite): Promise<Microsite>;

  abstract addPage(page: Page, siteId: string): Promise<Page>;

  abstract updatePage(page: UpdatePageDto): Promise<Page>;

  abstract copyPage(page: CopyPageDto): Promise<Page>;

  abstract deletePage(page: DeletePageDto): Promise<Page>;

  abstract deleteSite(id: string): Promise<void>;

  abstract updateControl(control: UpdateControlDto): Promise<Control>;

  abstract addControl(control: AddControlDto): Promise<Control>;

  abstract deleteControl(control: DeleteControlDto): Promise<void>;

  abstract updateSection(section: UpdateSectionDto): Promise<Section>;

  abstract updatePublishedSiteInfo(
    siteId: string,
    publishedSiteInfo: PublishedSiteInfo,
  ): Promise<void>;

  abstract updateSiteStatus(siteId: string, status: SiteStatus): Promise<void>;

}



import { StreamableFile } from '@nestjs/common';
import { createReadStream, createWriteStream, existsSync } from 'fs';
import {
  copyFile,
  mkdir,
  rm,
  readdir,
  readFile,
  stat,
  writeFile,
```

```
} from 'fs/promises';
import { join, parse } from 'path';

export async function writeToFile(path: string, data: string) {
  return writeFile(path, data);
}

export async function openReadStream(pathToFile: string, fileName: string) {
  return createReadStream(join(pathToFile, fileName));
}

export async function openWriteStream(pathToFile: string, fileName: string) {
  if (!(await exists(pathToFile))) {
    await createDirectory(pathToFile);
  }

  return createWriteStream(join(pathToFile, fileName));
}

export async function readTextFromFile(path: string) {
  return readFile(path, 'utf-8');
}

export async function createStreamableFile(
  pathToFile: string,
  fileName: string,
): Promise<StreamableFile> {
  const file = createReadStream(join(pathToFile, fileName));

  return new StreamableFile(file);
}

export async function writeBufferToFile(path: string, data: Buffer) {
  return writeFile(path, new Uint8Array(data));
}

export async function isDirectory(path: string) {
  const stats = await stat(path);

  return stats.isDirectory();
}

export async function readDirectory(path: string) {
  return readdir(path);
}

export async function createDirectory(path: string) {
  return mkdir(path, { recursive: true });
}

export async function remove(path: string) {
  return rm(path, {
```

```
    force: true,
    recursive: true,
    retryDelay: 200,
    maxRetries: 5,
  });
}

export async function copyToFile(src: string, dest: string) {
  return copyFile(src, dest);
}

export async function exists(src: string) {
  const parsedPath = parse(src);

  const isDirExists = existsSync(parsedPath.dir);

  if (isDirExists) {
    return existsSync(src);
  }

  return false;
}

export async function copyRecursive(src: string, dest: string) {
  if (!(await this.exists(src))) {
    return;
  }

  if (await isDirectory(src)) {
    if (!(await this.exists(dest))) {
      await mkdir(dest);
    }

    const children = await readdir(src);

    for (const childItemName of children) {
      await this.copyRecursive(
        join(src, childItemName),
        join(dest, childItemName),
      );
    }
  } else {
    await copyFile(src, dest);
  }
}

export async function readBinaryFromFile(path: string) {
  return readFile(path);
}
```